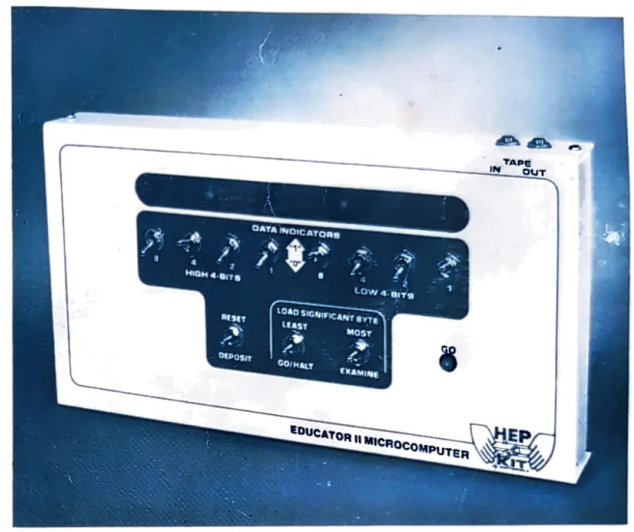
$4.00

Educator II Microcomputer Kit

HEP KIT by MOTOROLA



KEY FEATURES

- Designed specifically for the Educator II.
- Regulated 5.0 ± 5% volts d.c. output @ 1.0 amps.
- 60 Hz real time clock available (approximately 5.1V peak-to-peak).
- Complete kit — all parts, cabinet and construction manual.
- Easy, one evening construction.

Assembly and Operation Manual

Model HK-1000

EDUCATOR II MICROCOMPUTER SYSTEM

Construction and Instruction Manual

First Edition

TABLE OF CONTENTS

INTRODUCTION

PART 1. CONSTRUCTION PHASE

PART 2. INSTRUCTION PHASE

LIST OF ILLUSTRATIONS

PART 1. CONSTRUCTION PHASE

PART 2: INSTRUCTION PHASE

LIST OF TABLES

PART 1. CONSTRUCTION PHASE

PART 2. INSTRUCTION PHASE

Introduction

Congratulations! You are now the owner of a powerful microcomputer system that is educational, practical, and fun. Educator II combines a simple control panel with all of the power of a modern LSI (Large Scale Integration) microprocessor and associated components based on M6800 technology. Easy-to-follow step-by-step assembly instructions are provided in Part 1 of this book to aid you in assembling Educator II as easily and accurately as possible. After you have completed assembly and familiarized yourself with each of the front panel switches, you'll be able to use the microprocessor programming instructions you'll learn in Part 2 to develop your own programs. Of course, writing programs is only the beginning of the microprocessor experience; you'll also want to be able to use your system as quickly as possible to perform all of the tasks and operational routines you desire. For this reason, simple operating instructions are provided in Part 3 to show you how to load your program, review and debug your program, and transfer your completed program to a cassette tape for future use. The more you use Educator II, the more uses you'll find for its processing power. To get you started in the right direction, several practical and easy-to-follow programs are provided in Part 4.

As you gain experience with Educator II, you'll want to further explore computer architecture and software tools such as "subroutines", "interrupts", and "stack manipulations". For the advanced microcomputer user, your dealer has additional manuals that further describe the capabilities of the M6800-type microprocessor used in Educator II. These include both "Software" or programming manuals and "Hardware" or applications manuals.

If you require more memory than the 128 bytes (8 bits = 1 byte) supplied with Educator II, an optional 128 byte memory element may be purchased from your dealer and added directly to the Educator II printed circuit board. If even more memory is desired, additional memory modules (up to 65,536 bytes) can be connected to the Educator II through the use of the expander port located on the left side of the printed circuit board. All of the Educator II's Input/ Output connections are also provided for your convenience at the 44-pin connector located on the right side of the board. You may find it more convenient to add connectors to the basic board if expansion of the basic Educator II system is anticipated. These connectors may also be obtained from your dealer. Ample room has been provided within the case for these connectors.

Welcome to the world of microcomputers.

Part 1
Construction Phase

SECTION 1
KIT CONTENTS

1.1 IDENTIFYING AND SEPARATING KIT PARTS

Before beginning construction of your Educator II, it's necessary to identify and separate all of its piece parts to make assembly an easier job. If you've constructed kits before or have some familiarity with electronic components, you will recognize most of the parts in this kit. For those who are constructing their first kit, this section lists the various components used, shows their schematic symbol, and provides an illustration of what they look like so that you'll have no trouble identifying them during assembly. After you've identified and inventoried all of the parts, you'll be ready to start building. If any of the parts are missing from your kit, refer to the instructions provided at the back of this manual.

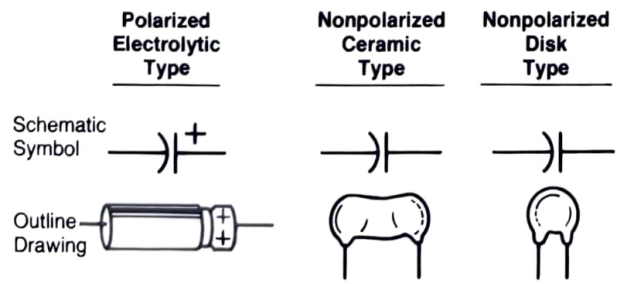
You've probably noticed that some of the parts are embedded in black foam pads. These foam pads are conductive and used to prevent a build-up of static electricity on these parts, which could render them inoperative. DO NOT REMOVE these parts from the black foam pads until instructed to do so during assembly. When you do remove them, avoid touching the leads.

HELPFUL HINT

When identifying parts and separating them into like groups, you may find it helpful to pre-bend the leads on resistors, capacitors, and diodes to a 90° angle. After doing this, these parts can then be separated into groups and inserted into a piece of paper or cardboard which will hold them until assembly. You can then identify each group of parts by marking their values on the paper or cardboard.

1.1.1 CAPACITORS

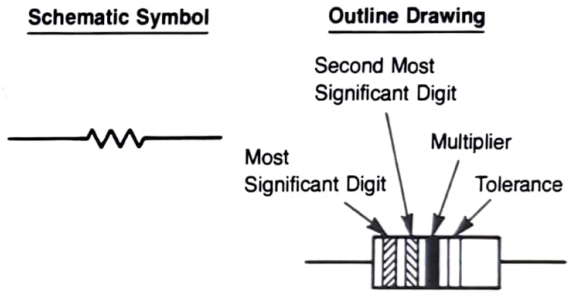
Thirteen capacitors of three different types are included in this kit: one polarized electrolytic, two nonpolarized ceramic, and ten nonpolarized disk. The schematic symbol and outline drawing for each of these types is shown below.



When connecting either a ceramic or disk capacitor, either lead may be inserted and soldered into either hole in the printed circuit board, since polarity (the orientation of the positive and negative leads) is not important. An electrolytic capacitor, however, must have the lead marked with the + sign inserted and soldered into the hole in the circuit board that is marked with a corresponding + sign. The only electrolytic capacitor used in this kit is C3, a 10 microfarad (μf) capacitor.

1.1.2 RESISTORS

Resistors are the most common part used in Educator II, with a total of 26 included in the kit. Resistor values are coded on the resistor package by colored circular bands. This is known as the resistor color code. The schematic symbol and outline drawing for the resistor is shown below.



The color code on the resistor is divided into two parts: value and tolerance. The value of the resistor is indicated by three colored bands, with the band closest to the end representing the most significant digit of the value. The second band from the end indicates the second most significant digit and the third band indicates the value of the multiplier used to obtain the actual resistance value.

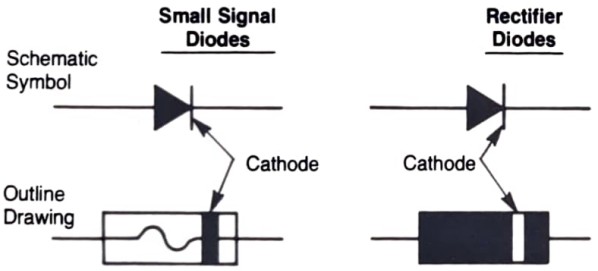
The tolerance of a resistor is the amount, in percent, that the resistor can vary from the value indicated by its color code. Tolerance is indicated by either a gold or silver band (representing ±5% and ±10% respectively). If no tolerance is marked on the resistor, then it is assumed to have a tolerance of ±20% of the indicated value. Thus, if a resistor were marked with a red band, a black band, a brown band, and a silver band, it would have a value of 200 ohms with a tolerance of ±10% (or ±20 ohms). The following table shows the value indicated by the color of the band. Use this table when separating resistors into groups of like values and when installing the resistors during assembly. (As an aid to you during assembly, all resistor values and the color code for each will be provided during assembly steps in which resistors are installed. NO tolerances will be given.)

TABLE 1-1. Resistor Color Code

Color	Value	Multiplier	Tolerance
Black	0	1	
Brown	1	10	
Red	2	100	
Orange	3	1,000	
Yellow	4	10,000	
Green	5	100,000	
Blue	6	1,000,000	
Violet	7	10,000,000	
Gray	8	100,000,000	
White	9	1,000,000,000	
Gold			±5%
Silver			±10%
No Color			±20%

1.1.3 DIODES

This kit contains seven diodes: six small signal type and one rectifier type. Although the same schematic symbol is used for both, their physical appearance is different. The rectifier diode package is black with a silver band at one end. The small signal diode is enclosed in a transparent glass case with a black band at one end. The schematic symbol and outline drawing for each type is shown below.

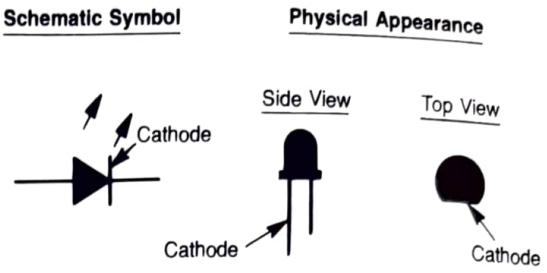


Aside from their outward appearances, both types of diodes operate in the same manner: they permit current to flow in only one direction. This property of a diode is known as rectification. Because of this property, diodes (like the electrolytic capacitors previously discussed) have a polarity that must be observed when installing them onto the printed circuit board. The polarity of a diode is indicated by a single color band (normally black or silver). This band indicates the cathode of the diode (refer to the diode illustrations). For your convenience, the schematic symbol for the diode is printed on the kit's printed circuit board in every location a diode is to be installed. Observe the polarity indicated by this symbol when installing each of the diodes.

Since diodes are semiconductor devices, they can be permanently damaged by excess heat during installation. Therefore, avoid prolonged use of the soldering iron when installing these parts.

1.1.4 LIGHT-EMITTING DIODES (LEDs)

Light-emitting diodes (LEDs) of the type used in this kit look like small red bubbles with two parallel leads extending from the bottom. Nine LEDs are included in this kit. The following illustration shows the schematic symbol and physical appearance for this type of LED device.

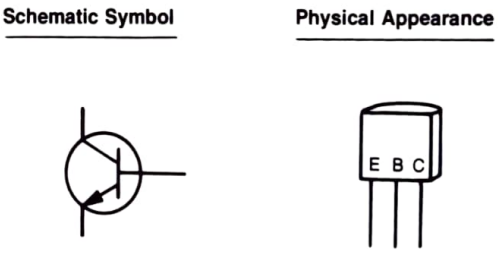


As in the case of the small signal or rectifier diodes, the LED must be properly oriented when installed onto the printed circuit board. The polarity of the LEDs used in this kit is indicated by the flat side on the case and by the different length of the two leads (refer to the physical appearance illustrations). The flat side (or longer lead) indicates the cathode of the device.

Since LEDs are semiconductor devices, they can be permanently damaged by excess heat during installation. Therefore, avoid prolonged use of the soldering iron when installing these parts.

1.1.5 TRANSISTORS

Only one transistor (an NPN-type) is used in your kit, making it easy to find. Its schematic symbol and physical appearance is:



The transistor must be oriented in the proper manner during installation. As an aid to proper orientation, the device is clearly marked on the flat side of the case in the following manner (read from left to right): E (emitter), B (base), and C (collector).

Since transistors are semiconductor devices, they can be permanently damaged by excess heat during installation. Therefore, avoid prolonged use of the soldering iron when installing this part.

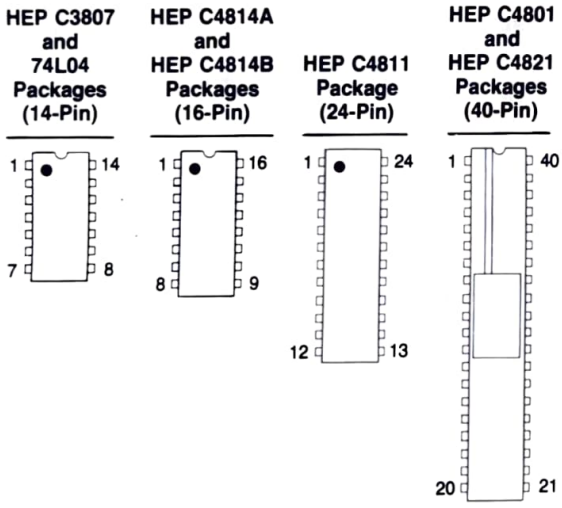
1.1.6 INTEGRATED CIRCUITS (ICs)

Integrated circuits (ICs) are rectangular in shape with gold or silver colored pins protruding from two edges of the package and bent in approximately a 90° angle. Eight ICs are included in this kit in four different size packages: three 14-pin packages, two 16-pin packages, one 24-pin package, and two 40-pin packages. You should have no trouble identifying the different ICs, since their device numbers are printed on top of the package.

NOTE

You will notice that three of the packages (the 24-pin and both 40-pin packages) are embedded in black conductive foam pads. These foam pads electrically connect all of the pins on these devices together to prevent static electricity from developing on them. This could permanently damage these devices. Therefore, *DO NOT* remove these ICs from the conductive foam pads until you are ready to install them during assembly.

Pin numbers are not normally printed on ICs. Instead, a mark (a small depression or raised bump) is made on the case to indicate pin 1. On some devices (such as the 40-pin packages), a semicircular indentation is molded into one end of the case. When these devices are held with the indentation pointing up, pin 1 is located at the top left-hand corner of the package (the 40-pin package also has a 1 printed in this corner). The following outline drawings are provided to assist you in determining the correct pin number for each IC included with this kit. Each of these package outlines is identified by the part number both printed on the package and shown on the schematic diagram.



1.1.7 SWITCHES

Eleven miniature toggle switches are supplied with this kit, each contained in a small, sealed plastic envelope along with its mounting hardware. Three of these switches are spring-loaded, center-off types, while the remaining eight are of the single-pole, double-throw variety. You can recognize the difference immediately: the toggle part of the spring-loaded, center-off switches protrudes directly out of the body of the switch, while the toggle part of the single-pole, double-throw switch protrudes at an angle from the body of the switch. The hardware supplied with each switch consists of two $^{5}/_{16}$'' retaining nuts, one lockwasher, and one flat locating washer. DO NOT REMOVE components from the plastic envelopes. Wait until instructed to do so during assembly.

1.2 PARTS INVENTORY

The following table (Table 1-2) provides a complete list of all the parts included in your Educator II kit. If any of these parts are missing, refer to the back of this manual for further instructions. A replacement parts list is also provided in Appendix C. This list provides the Part Number for each part used in this kit.

TABLE 1-2. Parts Inventory

PART	QUANTITY
Capacitors	
220pf (picofarad)	2
10μf (microfarad)	1
0.1μf (microfarad)	10
Resistors	
12,000 ohms (12K ohms) (Brown, Red, Orange)	2
470 ohms (Yellow, Violet, Black)	2
2,200 ohms (2.2K ohms) (Red, Red, Red)	7
3,000 ohms (3K ohms) (Orange, Black, Red)	2
270 ohms (Red, Violet, Black)	9
10,000 ohms (10K ohms) (Brown, Black, Orange)	3
1,000 ohms (1K ohms) (Brown, Black, Red)	1
Diodes	
Small Signal Type (glass case)	6
Rectifier Type (black case)	1
Light-Emitting Diodes (LEDs)	9
Transistors	1
Integrated Circuits (ICs)	
74L04 (Hex Inverters)	2
HEP C3807 (Dual Monostable Multivibrator)	1
HEP C4814A (ROM, Most Significant Digit)	1
HEP C4814B (ROM, Least Significant Digit)	1
HEP C4811 (RAM, 128 byte)	1
HEP C4821 (PIA — Peripheral Interface Adapter)	1
HEP C4801 (MPU — Microprocessing Unit)	1
Switches	
Spring-loaded center off toggle	8
Single-pole, double-throw toggle	3
Sockets	
16-pin	2
24-pin	1
40-pin	2
Miniature Phone Jacks	2
Case (Top and Bottom)	1
Sheet Metal Screws	8
Rubber Feet	4
Printed Circuit Board	1
Resin Core Solder	4 feet
Fuse (1 Amp Slo-Blo)	1
Wire	1 foot
Plastic Ferrule With Collar	1
Wire Clamp	1
Instruction Manual	1

1.3 RECOMMENDED SUPPLEMENTARY PARTS

Table 1-3 lists additional parts which you may find useful in operating your Educator II. These parts are available from your dealer. If the specified item cannot be found, an equivalent can be used.

TABLE 1-3. Recommended Supplementary Parts

ITEM	SPECIFICATIONS	MFR	PART NO.
Power Supply	1 Amp (minimum) @ 5Vdc	HEP (MOT)	HK1001
Connectors	44-pin, double readout edge of board connector with solder lugs on .156'' centers	CINCH VIKING SAE	50-44S-30 2VK22D/1-2 SAC-22D/1-2
Random Access Memory (with socket)	128 byte	HEP (MOT)	C4811

SECTION 2 ASSEMBLY

2.1 RECOMMENDED TOOLS AND TEST EQUIPMENT

Now that you've checked all of the parts and separated them into related groups, you're just about ready to start building. All that's left is to make sure you have the right tools for the job. The list of suggested tools and test equipment provided in Table 1-4 contains the items normally required to build the kit, but you may substitute these items with similar tools or test equipment. Those items marked with an asterisk (*) are optional during assembly, but you'll need the volt-ohmmeter for testing.

TABLE 1-4. Recommended Tools and Test Equipment

1. $^{5}/_{16}$'' box end wrench
2. Straight slot screwdriver
3. Needlenose pliers
4. Diagonal wire cutters
5. Soldering iron (47 Watts maximum)
* 6. Dampened sponge (to wipe tip of iron)
* 7. Circuit cleaner (trichloroethane)
* 8. Stiff-bristled brush
* 9. Volt-Ohmmeter (VOM)

* Optional items for use during construction.

2.2 SOLDERING TECHNIQUES

In order for your kit to operate properly and for you to obtain professional-looking results, your kit must be properly soldered. If this is your first kit, or if you haven't done any soldering for some time, take a few minutes to read this paragraph and then practice your soldering technique BEFORE construction begins. You'll be glad you did.

When soldering metals together, solder flux must be used before soldering to clean the metal surfaces and free them from oxides. Although several different types of flux are used, depending upon the type of soldering involved, only one type of flux is acceptable for use with electronic circuits: resin. The solder you received in your kit has a resin core. Therefore, as you heat the solder with your iron, the resin core melts first to clean the metal surfaces and then the solder itself melts to electrically and mechanically join the two metals together. If you should run out of the solder supplied with the kit during the construction phase, you can get more almost anyplace, but make sure it's resin core. DO NOT USE ACID CORE SOLDER. After construction is completed, you'll notice that a brownish colored, very hard residue remains at some of the points you soldered. This is the resin flux. Even though the resin is nonconductive, it should be removed. This can be done easily by using a good quality circuit cleaner (such as trichloroethane) and a stiff bristled brush. A little cleaner on the brush and a few swipes over the surface of the printed circuit board will remove the residue. DON'T try to chip the resin off with a sharp object as you may cut one of the printed circuit tracks on the board.

The main objective of good soldering is to use just enough heat to obtain a good electrical and mechanical connection. If your solder joint isn't mechanically good, your kit could fail, due to thermal stresses caused by the heating up and cooling down of circuit components during and after use. These thermal stresses are enough to break a poorly made solder joint, thus opening them electrically. Of all possible failures, ones of this type can be the hardest to troubleshoot and find. So be careful. Apply enough heat to the metal surfaces you are joining to make the solder spread freely and show the contour of the connection under the solder. Enough heat must be used so that the solder can actually penetrate the metal surfaces, making an unbroken path over which electricity can travel. You are not using enough heat if the solder barely melts and forms a rounded ball of rough, flaky solder.

A soldering iron of 25 to 45 Watts is recommended. Any iron in this range with a clean chisel-shaped tip will supply the correct amount of heat to make a good solder connection. Keep the iron tip brightly coated with solder. When necessary, wipe the hot tip clean with a dampened sponge. If you are using an old tip, clean it before you start soldering.

Here is a step-by-step procedure on how to solder.

1. Coat the tip of a hot iron with solder. Then, firmly press the flat side of the tip against the parts to be soldered together. Keep the iron there while you apply the solder between the metal to be soldered and the iron tip. Use only enough solder to flow over all surfaces of the connection and all wires in the connection. Remove the iron immediately after achieving these results.

2. Do not move parts until the solder hardens. If you accidently move a part as the solder is hardening, apply your iron and reheat.

3. Check your soldering. You have a good connection if your solder has flowed over all surfaces to be connected, following the shape of the surfaces. It should appear smooth and bright. You have not used enough heat if your connection is rough and flaky looking or if the solder has formed a round ball instead of spreading evenly. Also check to see that no solder connections or "bridges" have formed between adjacent pins or printed wiring tracks after *every* soldering operation.

2.3 INSTALLATION OF PARTS

Now to put it all together — here's how to properly assemble your Educator II kit. The steps in this procedure have been carefully designed to help you build the kit in the best sequence for ease of assembly. Note that each group of parts (switches, sockets, resistors, etc.) is positioned and soldered in place in separate portions of the assembly procedure. This will eliminate questions like, "Did I leave anything out?". Some of the components are positioned very close to others, making it more difficult, as construction progresses, to read the printing on the circuit board and find the location of each part. For this reason, photographs taken during actual construction have been included and the parts installed during that step are clearly labeled on each. Mark off each part of the illustrations as each part is installed. Follow the steps in sequence to obtain the least complicated and most enjoyable construction method.

A checkout procedure is provided following some of the assembly steps to verify that the step just performed has been done correctly. These checks are divided into three different categories: continuity checks, resistance checks, and power-on checks. Performing the recommended checks makes it easier for you to isolate a malfunction if one does occur. If a check fails, refer to the installation procedure for the component being checked. (Each step of the checkout procedure lists the component being checked.) After all checkout procedures are performed and the correct indications have been obtained, the Educator II should operate properly the first time.

Certain good construction practices should be observed during assembly to obtain professional-looking results. These are:

1. When soldering, use as little heat as possible, since many of the components in this kit are semiconductor devices which can be damaged by excess heat.
2. Apply solder only to the reverse side of the board where components are *not* mounted.
3. After each point is soldered, verify that no solder "bridges" or pieces of solder have shorted together adjacent points on the printed circuit board.
4. Observe polarity of all polarized components.
5. After soldering each component onto the printed circuit board, carefully trim the excess lead protruding through the board with a pair of diagonal wire cutters. Trim as close as possible to the printed circuit board without cutting the soldered joint.

So now you're ready to begin. Follow the instructions carefully, check off each one as it's performed, and have fun.

NOTE

Throughout the following assembly procedures, all references to the front side of the board refer to the side on which both the components and the printing are located. All references to the top, bottom, left-hand, and right-hand side of the board refer to the corresponding areas of the board as viewed from the front in a manner that permits you to read the labels printed on the board. Each photograph in the assembly procedure also indicates these board positions.

2.3.1 INSTALLATION OF SOCKETS, SWITCHES, AND FUSE

The first step of the installation procedure is to install the sockets, switches, and fuse. Refer to Figure 1-1 during this procedure.

Assembly

1. Locate one 40-pin Integrated Circuit (IC) socket and install it in the U3 position shown on the printed circuit board. Note that a circular indentation is molded into one end of this socket to indicate pin 1. This indentation should be oriented to point to the right-hand side of the board. Carefully solder each of the 40 pins on the reverse side of the board.

HELPFUL HINT

You may find it helpful to slightly bend several of the pins protruding through the back side of the board prior to soldering. This will help hold the socket in position.

2. Locate the other 40-pin socket and repeat step 1. This time, however, install the socket in the U4 position.

3. Locate one 24-pin socket and insert it in the U5 position on the board. (Actually, this socket could also be inserted in the position marked U6. However, this position should be reserved for the optional memory element. If you have purchased this optional memory, install the socket for it at this time.) Solder each of the 24 pins protruding through the back side of the board.
4. Locate one 16-pin IC socket and insert it in the U7 position on the board. Orientation is not critical for this socket. Solder each pin on the back side of the board.
5. Locate the other 16-pin socket and repeat step 4. This time, however, install the socket in the U7 position.
6. Locate and remove the eight single-pole, double-throw toggle switches from their plastic envelopes. Remove the mounting hardware. Find eight of the $^5/_{16}$'' nuts and install one on each of the switches. Tighten only finger-tight. Put the remaining mounting hardware into one of the empty envelopes for future use. Keep the eight switches in a separate group.
7. Locate and remove the three spring-loaded, center-off toggle switches from their plastic envelopes. Remove the mounting hardware. Find three of the $^5/_{16}$'' nuts and install one on each of the switches. Tighten only finger-tight. Put the remaining mounting hardware into an envelope for future use. Keep these three switches in a group separate from the switches of the previous step.
8. While holding the board in one hand, install the eight single-pole, double-throw switches into the holes marked S1 through S8 on the printed circuit board. Note that each of these switches has three solder lugs on the bottom. These lugs will fit into the holes provided on the board. Orientation of these switches is not important. DO NOT SOLDER THE SWITCHES AT THIS STEP.
9. Install the three spring-loaded, center-off toggle switches into the holes marked S9 through S11 on the printed circuit board. Note that these switches also have three solder lugs on the bottom. These lugs will fit into the holes provided on the board. DO NOT SOLDER THESE SWITCHES AT THIS STEP.
10. Holding the front cover of the case in the other hand, carefully position the cover so that all eleven switches protrude from the holes in the cover. At this point you should be able to hold the circuit board, and the case cover will sit evenly on the shoulder of the eleven switches.
11. Carefully turn the cover and board upside down. You should now be able to hold the cover, and the board will be held in position by the eleven switches.

HELPFUL HINT

During the next step of the assembly procedure, small blocks or other items can be used to support the cover in an inverted position, making it easier for you to solder.

12. Solder each terminal of all eleven switches to the printed circuit board. After all of the solder joints have cooled sufficiently to harden, the case cover can be removed. Note that the foregoing procedure enables the cover to slip very easily over all of the switches.
13. Locate the fuse and insert the leads into the holes provided on the board. (The fuse is located at the top, right-hand side of the board.) Solder the fuse leads on the back of the board and trim the excess leads.

Continuity Check

Perform the following continuity checks to verify that all of switches and the fuse have been installed correctly. The sockets cannot be checked for continuity. Therefore, visually inspect each of the solder connections on the sockets to ensure that a good solder joint has been achieved and that no pins have been unknowingly shorted together.

Switch	Position	From	To
S1	Down	P2-22/Z	P2-J
S2	Down	P2-22/Z	P2-H
S3	Down	P2-22/Z	P2-K
S4	Down	P2-22/Z	P2-L
S5	Down	P2-22/Z	P2-P
S6	Down	P2-22/Z	P2-R
S7	Down	P2-22/Z	P2-S
S8	Down	P2-22/Z	P2-T
S9	Down	P2-22/Z	P2-V
S9	Up	Note 1	
S10	Down	P2-22/Z	P2-U
S10	Up	Note 1	
S11	Down	P2-22/Z	P2-Y
S11	Up	Note 1	

Notes:
1. Cannot be checked at this step.
2. P2 refers to connector pads located on the right-hand edge of the board. The connector pads on the front side of the board are numbered from top to bottom from P2-1 to P2-22. The connector pads on the back side of the board are numbered from top to bottom from P2-A to P2-Z. Note that P2-22 and P2-Z are connected together. The number for each connector pad is also shown on Figure 1-1.

Fuse F1

Connect one lead of the VOM to pin P1-A (located on the front side of the board in the top left-hand corner) and the other lead to pin P2-1 (located on the front side of the board in the top right-hand corner). You should obtain a reading of 0 ohms on the VOM.

2.3.2 INSTALLATION OF RESISTORS

The next step is to install the resistors. In this procedure, all of the resistors of one value are installed in separate steps to avoid installing the wrong resistor value. Figure 1-2 shows the location of all resistors installed in this procedure.

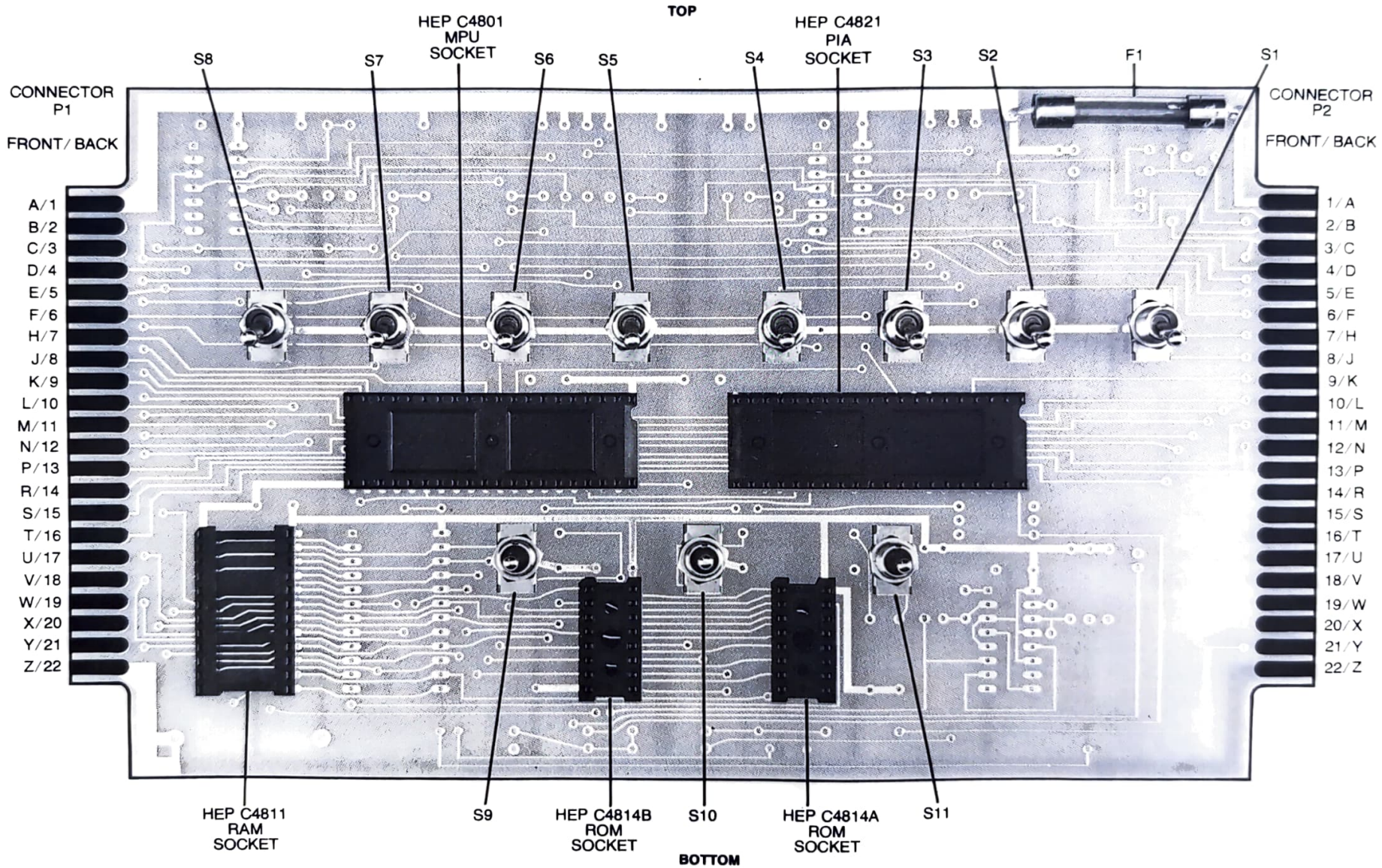


FIGURE 1-1. Installation of Sockets, Switches, and Fuse

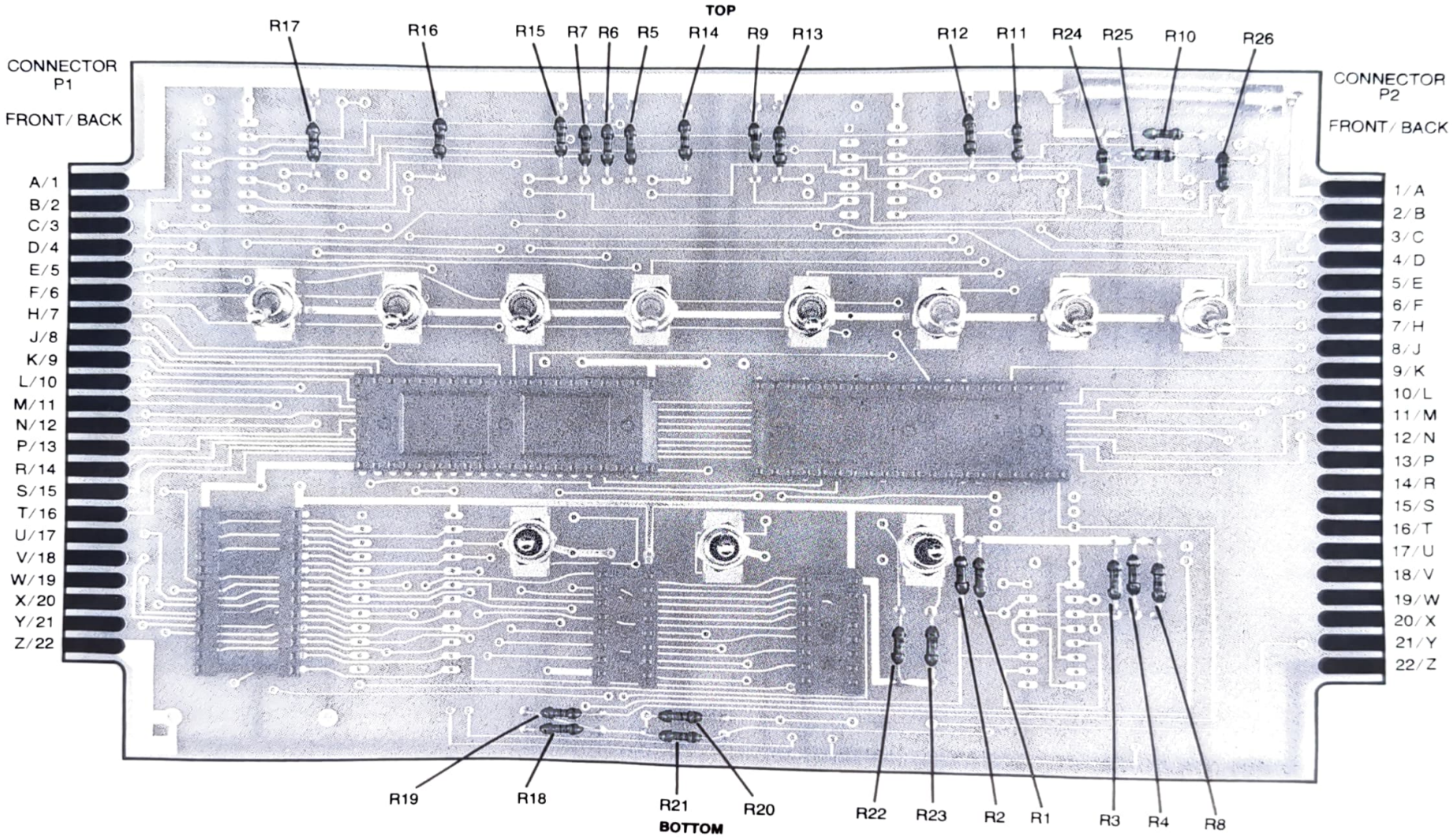


FIGURE 1-2. Installation of Resistors

Assembly

1. Locate and install the two 12,000 ohm (12K ohm) resistors (Brown, Red, Orange) into the holes marked R1 and R3. (These resistors are located in the bottom right-hand corner of the board). Carefully solder the leads of these resistors on the back of the printed circuit board. Trim off the excess lead lengths.
2. Locate and install the two 470 ohm resistors (Yellow, Violet, Brown) into the holes marked R2 and R4. (These resistors are located in the bottom right-hand corner of the board, next to the resistors installed in the preceding step.) Carefully solder the leads of these resistors on the back of the printed circuit board. Trim off the excess lead lengths.
3. Locate and install the seven 2,200 ohm (2.2K ohm) resistors (Red, Red, Red) into the holes marked R5 and R18 through R23. (Resistor R5 is located in the middle of the top edge of the board, while resistors R18 through R23 are located in the middle of the bottom edge of board.) Carefully solder the leads of these resistors on the back of the printed circuit board. Trim off the excess lead lengths.
4. Locate and install the two 3,000 ohm (3K ohm) resistors (Orange, Black, Red) into the holes marked R6 and R7. (These resistors are located in the middle of the top edge of the board, next to resistor R5.) Carefully solder the leads of these resistors on the back of the printed circuit board. Trim off the excess lead lengths.
5. Locate and install the nine 270 ohm resistors (Red, Violet, Brown) into the holes marked R8 and R10 through R17. (Resistor R8 is located in the bottom right-hand corner of the board, next to resistor R4, while resistors R10 through R17 are located across the top of the board.) Carefully solder the leads of these resistors on the back of the printed circuit board. Trim off the excess lead lengths.
6. Locate and install the three 10,000 ohm (10k ohm) resistors (Brown, Black, Orange) into the holes marked R9, R24, and R25. (Resistor R9 is located in the middle of the top edge of the board, while resistors R24 and R25 are located in the top right-hand corner of the board.) Carefully solder the leads of these resistors on the back of the printed circuit board. Trim off the excess lead lengths.
7. Locate and install one 1,000 ohm (1K ohm) resistor (Brown, Black, Red) into the holes marked R26. (This resistor is located on the top right-hand corner of the printed circuit board.) Carefully solder the leads of this resistor on the back of the printed circuit board. Trim off the excess lead lengths.

Resistance Check

Perform the following resistance checks to verify that all of the resistors have been installed correctly. The resistance measurements provided in this check are approximations only, since all ohmmeter readings vary slightly from one VOM to another. The readings you obtain, however, should be very close, even if not exact.

Resistor	Check
R1,R2,R3	Not checked at this time.
R4	Connect VOM between connector P1-A/1 and P1-D. VOM should indicate approximately 470 ohms.
R5	Connect VOM between connector P1-A/1 and P1-5. VOM should indicate approximately 2K ohms.
R6	Connect VOM between connector P1-A/1 and P1-N. VOM should indicate approximately 3K ohms.
R7	Connect VOM between connector P1-A/1 and P1-L. VOM should indicate approximately 3K ohms.
R8	Not checked at this time.
R9	Connect VOM between connector P2-4 and P2-1. VOM should indicate approximately 10K ohms.
R10-R17	Not checked at this time.
R18	Connect VOM between connector P2-1/A and P2-22/Z. Move switch S9 to the up position. VOM should indicate approximately 2.2K ohms.
R19	Connect VOM between connector P2-1/A and P2-22/Z. Move switch S9 to the down position. VOM should indicate approximately 2.2K ohms.
R20	Connect VOM between connector P2-1/A and P2-22/Z. Move switch S10 to the up position. VOM should indicate approximately 2.2K ohms.
R21	Connect VOM between connector P2-1/A and P2-22/Z. Move switch S10 to the down position. VOM should indicate approximately 2.2K ohms.
R22	Connect VOM between connector P2-1/A and P2-22/Z. Move switch S11 to the up position. VOM should indicate approximately 2.2K ohms.
R23	Connect VOM between connector P2-1/A and P2-22/Z. Move switch S11 to the down position. VOM should indicate approximately 2.2K ohms.
R24	Connect VOM between connector P2-1/A and P2-2. VOM should indicate approximately 10K ohms.
R25,R26	Connect VOM between connector P2-3 and P2-22/Z. VOM should indicate approximately 21K ohms.

2.3.3 INSTALLATION OF CAPACITORS

The next step is to install the capacitors. In this procedure, all of the capacitors of one value are installed in separate steps to avoid installing the wrong value. Figure 1-3 shows the location of all capacitors installed in this procedure. Since capacitors cannot be checked with a VOM, no resistance or continuity checks are performed after installing these parts. These parts can only be checked during operation.

Assembly

1. Locate and install the two 220pf (picofarad) mica capacitors (dark brown) into the holes marked C1 and C2. (These capacitors are located in the bottom right-hand corner of the board.) Carefully solder the leads of these capacitors on the back of the printed circuit board. Trim off the excess lead lengths.

2. Locate and install the one 10μf (microfarad) electrolytic capacitor into the holes marked C3. (This capacitor is a polarized type indicated by the + sign marked on one end. Observe polarity when installing. It is located in the bottom left-hand corner of the board.) Carefully solder the leads of this capacitor on the back of the printed circuit board. Trim off the excess lead lengths.

3. Locate and install the ten 0.1μf (microfarad) disk capacitors (yellow) into the holes marked C4 through C13. (These capacitors are spread around the board, so refer to Figure 1-3 for parts location.) Carefully solder the leads of these capacitors on the back of the printed circuit board. Trim off the excess lead lengths.

2.3.4 INSTALLATION OF DIODES, LEDS, AND TRANSISTOR

The next step is to install the diodes, LEDs, and transistor. In this step of the procedure, each group of components is installed separately to avoid installing the wrong part. Figure 1-4 shows the location of these parts. These parts will be checked during the following installation step.

Assembly

1. Locate and install the six small signal diodes into the holes marked CR10 through CR15. (Diodes CR10 and CR11 are located in the top left-hand corner of the board, diode CR12 is located in the bottom right-hand corner of the board, and diodes CR13 through CR15 are located just below the socket of U4. All of these diodes must be oriented properly when installed. Refer to paragraph 1.1.3 for orientation.) Carefully solder the leads of these diodes on the back of the printed circuit board. AVOID EXCESS HEAT. Trim off excess lead lengths.

2. Locate and install the one rectifier diode into the holes marked CR16. (This diode is located in the top right-hand corner of the board. This diode must be oriented properly when installed. Refer to paragraph 1.1.3 for orientation.) Carefully solder the leads of this diode on the back of the printed circuit board. AVOID EXCESS HEAT. Trim off excess lead lengths.

3. Locate and install eight of the LEDs into the holes marked CR1 through CR8. (These LEDs are located across the top of the board. They must be oriented properly when installed. The flat side of the case should point toward the bottom of the board. Refer to paragraph 1.1.4 for further orientation information.) Carefully solder the leads of these LEDs on the back of the printed circuit board. AVOID EXCESS HEAT. Trim off excess lead lengths.

4. Locate the final LED and insert it into the holes marked CR9. DO NOT SOLDER PART AT THIS STEP. (This LED is located in the bottom right-hand corner of the board and must be properly oriented when installed. The flat side of the case should point toward the bottom of the board. Refer to paragraph 1.1.4 for further orientation information.)

5. Locate the plastic, two-part ferrule and, from the front side of the front cover, insert the longer part through the hole marked GO. Next, take the shorter portion (the ferrule collar) and, from the rear of the front cover, press firmly over the longer portion protruding through from the front side. (A spot of glue will keep the two halves together, but don't get any on the inside of the ferrule. Allow the glue to dry before proceeding to the next step.)

6. Install the board into the front cover of the case. From the back of the board, push on the leads of LED CR9 until it is inserted into the ferrule in the front cover. Carefully solder the leads of the LED on the back of the printed circuit board. AVOID EXCESS HEAT. Trim off excess lead lengths and remove board from front cover.

NOTE

Instead of mounting the LED to the board, it may also be inserted into the ferrule on the front cover and connected to the board with wires. However, if this is done, make sure you observe the orientation of the LED.

7. Locate and install the transistor into the holes provided in the top right-hand corner of the board. The flat side of the transistor should point toward the bottom of the board. (Refer to paragraph 1.1.5 for further orientation information. Carefully solder the leads of the transistor on the back of the printed circuit board. AVOID EXCESS HEAT. Trim off excess lead lengths.

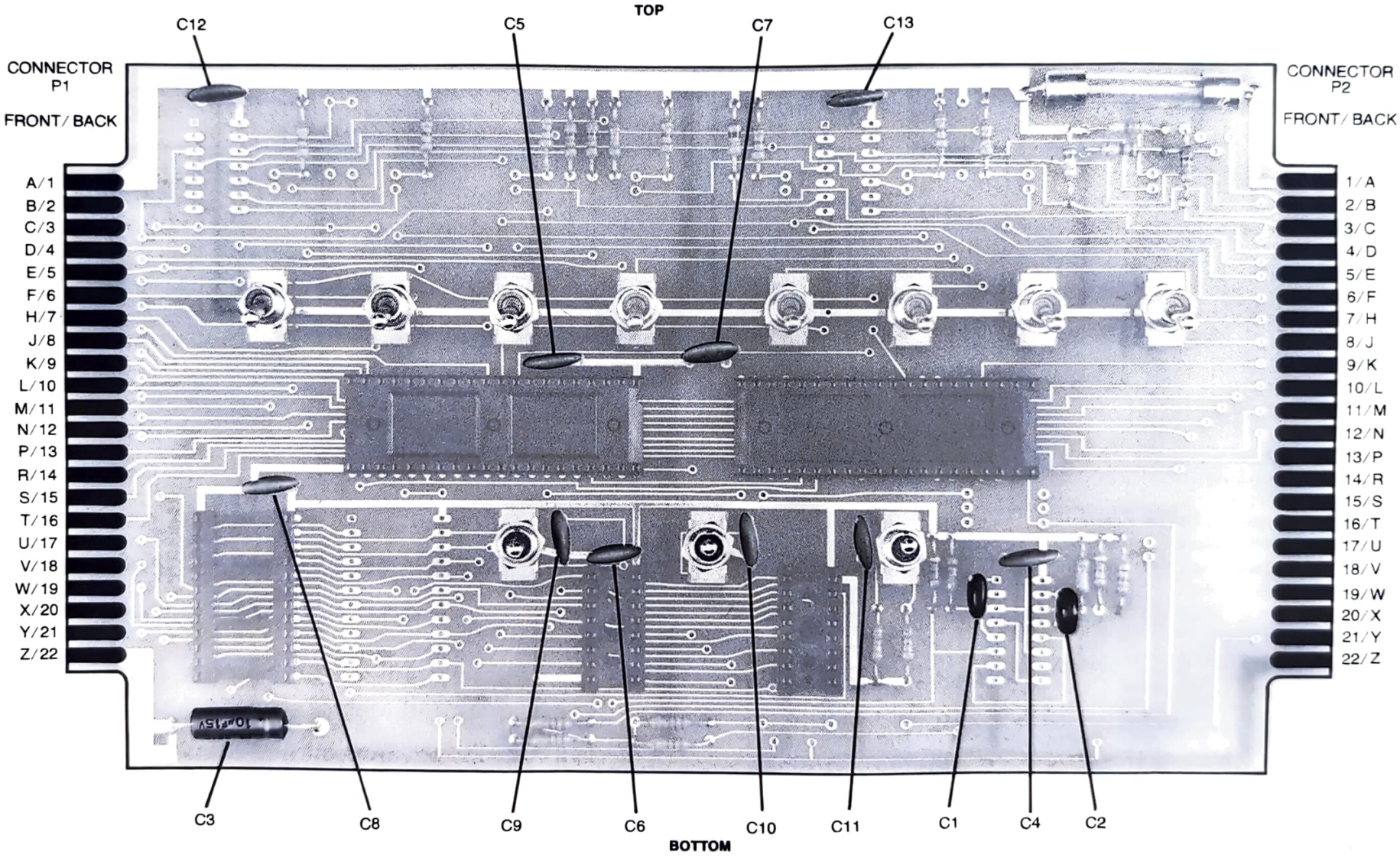


FIGURE 1-3. Installation of Capacitors

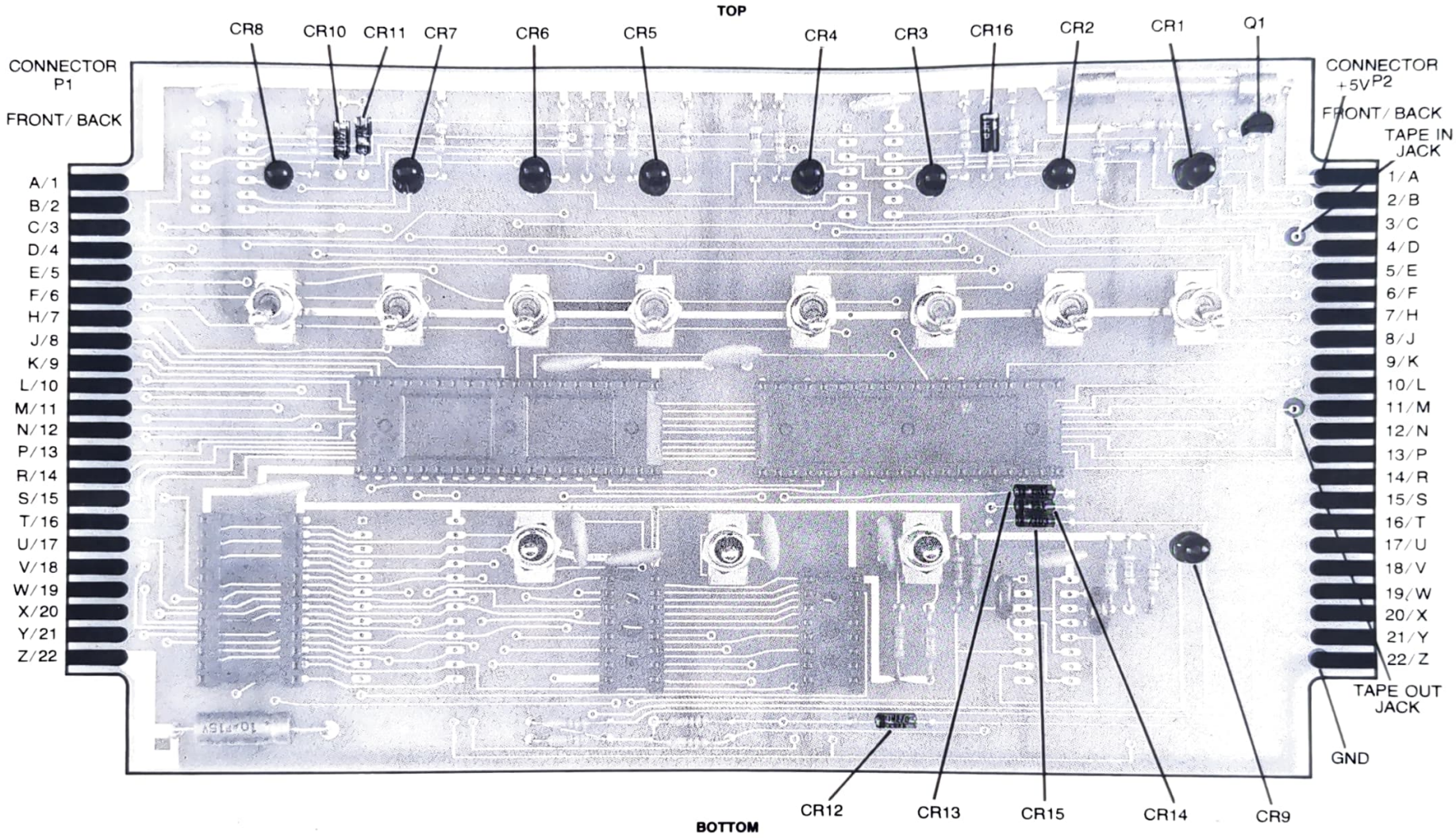


FIGURE 1-4. Installation of Diodes, LEDs, and Transistor

2.3.5 INSTALLATION OF INTEGRATED CIRCUITS (ICs)

The next step in the assembly process is to install the various Integrated Circuits (ICs). In this portion, the devices that must be soldered onto the printed circuit board are installed first, followed by a power-on check. After this power-on check is completed and the results specified by the check have been obtained, the PIA (HEP C4821) and the MPU (HEP C4801) are then installed into their respective sockets and another power-on check is performed. Following this power-on check, the two ROMs (HEP C4814A and HEP C4814B) are installed and another power-on check is conducted. After successfully completing this check, the last IC (RAM — HEP C4811) is installed and checked. By following the above procedure precisely, you can minimize the possibility of an error being made. Each of the power-on checks is used to verify that the preceding assembly step was performed correctly. If any of the power-on checks fails to produce the indicated results, carefully check the IC device installed in the previous step for bent pins, proper orientation, etc. Also verify that all of the pins on the socket being used have been properly soldered on the back side of the printed circuit board. This should resolve any difficulty you may have. DON'T FORGET TO OBSERVE THE PROPER POLARITY OF THE ICs WHEN INSTALLING THEM INTO THE SOCKETS. FAIURE TO DO THIS MAY RENDER THEM INOPERATIVE. Figure 1-5 shows the location and proper orientation of all ICs installed during this procedure.

Assembly Step #1

1. Locate one 74L04 IC (14-pin package) and install it in the U1 position in the top left-hand corner of the printed circuit board. Note that in one corner of the IC package is a small, circular depression. This indicates pin 1. This end of the package should be oriented to point to the top of the board. Carefully solder each of the 14 pins on the back side of the board. AVOID EXCESS HEAT.
2. Locate the other 74L04 IC and repeat step 1. This time, however, install the IC in the U2 position located in the middle of the top edge of the board.
3. Locate and install the HEP C3807 in the U9 position in the bottom right-hand corner of the board. Note that at one end of this device is a small circular notch molded into the end of the case. This notch should be oriented to point to the top of the board. Carefully solder each of the 16 pins on the back side of the board. AVOID EXCESS HEAT.

Power-On Check #1

1. Connect the positive lead of a +5Vdc (1 Amp minimum) power supply to connector P2-1/A and the negative lead to connector P2-22/Z.
2. Apply power.
3. Observe that all nine of the LEDs illuminate. If all of them don't, consult the schematic diagram at the back of this manual to determine which circuit is associated with the LED that doesn't illuminate. If none of them illuminate, check your fuse.
4. Remove power.

Assembly Step #2

1. Locate the one HEP C4821 (PIA) supplied in the kit. Remove it from the black foam pad, being careful not to touch the pins, and then install it in socket U4 located to the right of center on the board. Note that at one end of this device is a small circular notch molded into the end of the case. This notch should be oriented to point to the right-hand side of the board. When inserting the device into the socket, be very careful to first align each of the 40 pins with its respective contact in the socket. (If you can't get them to align exactly, bend the pins slightly until all of them do align.) After you have all the pins aligned, slowly but firmly press it into the socket. (If you place one thumb at each end of the device and press slowly in a rocking motion, the device will go in easier.) DON'T FORCE THIS PART INTO THE SOCKET.
2. Locate the one HEP C4801 (MPU) supplied in the kit. Remove it from the black foam pad, being careful not to touch the pins, and then install it in socket U3 located to the left of center on the board. Use the installation procedure described in the previous step to install this part.

Power-On Check #2

1. Connect the positive lead of a +5 Vdc (1 Amp minimum) power supply to connector P2-1/A and the negative lead to connector P2-22/Z.
2. Apply power.
3. Observe that the eight LEDs in the row across the board (CR1 through CR8) are illuminated and that the ninth LED (CR9) is *not* illuminated.
4. Remove power.

Assembly Step #3

1. Locate the one HEP C4814A (Most significant digit ROM) supplied in the kit. Install it in socket U8 located just right of center near the bottom edge of the board. Note that at one end of this device is a small circular notch molded into the end of the case. This notch should be oriented to point to the top of the board. When inserting the device into the socket, be very careful to first align each of the 16 pins with its respective contact in the socket. Then, slowly but firmly, press it into the socket. DO NOT FORCE THIS PART.
2. Locate the one HEP C4814B (Least significant digit ROM) supplied in the kit. Install it in socket U7 located just left of center near the bottom edge of board. Use the installation procedure described in the previous step to install this part.

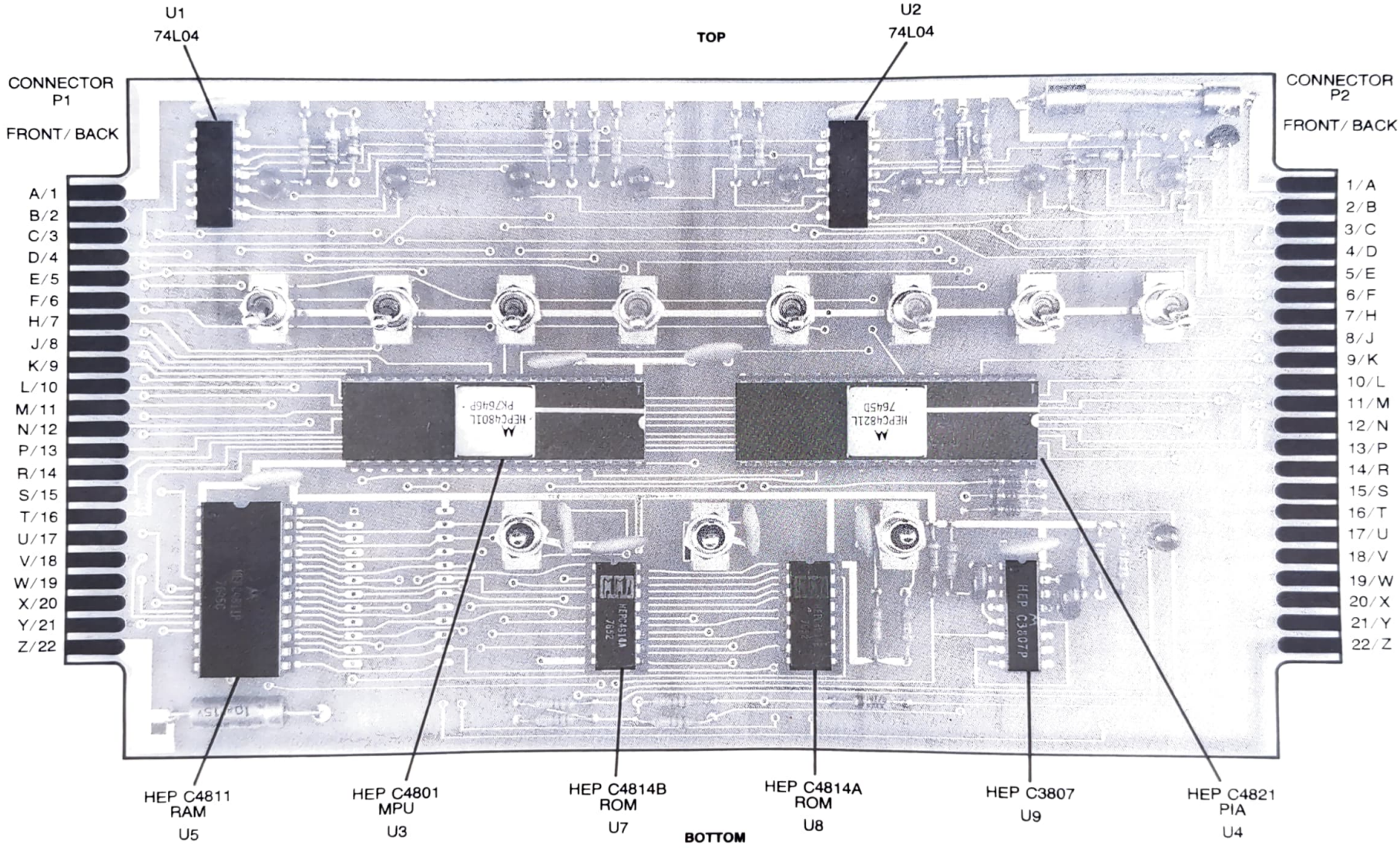


FIGURE 1-5. Installation of Integrated Circuits (ICs)

Power-On Check #3

1. Connect the positive lead of a +5Vdc (1 Amp minimum) power supply to connector P2-1/A and the negative lead to connector P2-22/Z.
2. Apply power.
3. Observe that the eight LEDs in the row across the board (CR1 through CR8) are *not* illuminated and that the ninth LED (CR9) is illuminated.
4. Position the lower left-hand spring-loaded center off toggle switch to the UP position and hold in that position. Observe that the eight LEDs in the row across the board (CR1 through CR8) are now illuminated and that the ninth LED (CR9) is *not* illuminated.
5. Remove power.

NOTE

If the indications listed in the above procedure are not obtained, you probably have the two ROMs in the wrong sockets. If CR1 through CR8 are illuminated when power is initially applied and CR9 is *not* illuminated, and this indication does not change when the toggle switch is positioned UP, then exchange the two ROMs in their sockets. The preceding power-on check should then be performed again.

Assembly Step #4

1. Locate the one HEP C4811 (128 byte RAM) supplied in the kit. Remove it from the black foam pad, being very careful not to touch the pins, and install it in socket U5 located in the bottom left-hand corner of the board. Note that at one end of the device is a small circular notch molded into the end of the case. This notch should be oriented to point to the top of the board. When inserting the device into the socket, be very careful to first align each of the 24 pins with the respective contact in the socket. Then, slowly but firmly, press it into the socket. DO NOT FORCE THIS PART.

NOTE

If you have also purchased the optional memory element, install it into socket U6 at this time.

Power-On Check #4

1. Position the eight switches in the row across the board (S1 through S8) to point DOWN.
2. Connect the positive lead of a +5Vdc (1 Amp minimum) power supply to connector P2-1/A and the negative lead to connector P2-22/Z.
3. Apply power.
4. Observe that the only LED illuminated is the one in the bottom right-hand corner of the board (CR9).
5. Momentarily position the middle switch in the bottom row of switches to point UP and observe that LED CR9 blinks off and then once again illuminates. None of the other LEDs should be illuminated at this time.
6. Momentarily position the left-hand switch in the bottom row of switches (S9) to point DOWN. Observe that the right-hand LED in the row of eight LEDs across the board illuminates (CR1).
7. Momentarily position the left-hand switch in the bottom row of switches (S9) to point DOWN. Observe that this time the right-hand LED in the row of eight LEDs across the board (CR1) goes out and the LED directly to its left (CR2) illuminates.
8. Momentarily position the left-hand switch in the bottom row of switches (S9) to point DOWN. Observe that both LED CR1 and LED CR2 are illuminated.
9. DO NOT REMOVE POWER. INSTEAD, PROCEED TO THE NEXT STEP.

2.4 TESTING OF FINISHED KIT

Now that you have completed construction of the Educator II's printed circuit board, you'll want to completely test it to make sure that it operates correctly. If you've performed all of the preceding checks and the correct indication have been obtained,then the tests provided in this paragraph will be very quick. However, if any of the previous checks has failed and the problem was not found and corrected, you will have to troubleshoot your kit to locate the problem BEFORE performing this final check.

The test performed in this paragraph is known as the Blink Routine. A routine is a predetermined set of instructions. This routine is part of the program stored in the two ROMs (U7 and U8). It will accept your inputs, begin operating at your command, and will stop at your command. To verify that the routine has performed correctly, the eight LEDs in a row across the board will flash (blink) on and off in a set pattern. The following procedure explains how to initiate and execute (operate) this test.

1. Momentarily position the left-hand switch in the bottom row of switches (S9) to point UP, and observe that all eight of the LEDs in the row across the board (CR1 through CR8) illuminate while the remaining LED (CR9) extinguishes. Observe that when the switch is released and goes back to the upright position, the eight LEDs extinguish and LED CR9 illuminates. This RESETS the Educator II.
2. Position the eight switches in the row across the board as follows (this sets the starting address):

S8	S7	S6	S5	S4	S3	S2	S1
UP	UP	UP	UP	UP	DN	DN	UP

3. Momentarily position the middle switch (S10) in the bottom row of switches to point UP and observe that CR9 is still the only LED that is illuminated. This initiates the program at the starting address set in the previous step.
4. Position the eight switches in the row across the board as follows (this sets the first byte of the program data into the RAM memory):

S8	S7	S6	S5	S4	S3	S2	S1
UP	DN	DN	DN	DN	DN	DN	DN

5. Momentarily position the left-hand switch in the bottom row of switches (S9) to point DOWN and observe that the LEDs indicated by an X are illuminated.

CR8	CR7	CR6	CR5	CR4	CR3	CR2	CR1
X	X	X	X	X		X	

6. Position the eight switches in the row across the board as follows (this sets the second byte of the program data):

S8	S7	S6	S5	S4	S3	S2	S1
DN	UP	UP	UP	UP	DN	DN	DN

7. Momentarily position the left-hand switch in the bottom row of switches (S9) to point DOWN and observe that the LEDs indicated by an X are illuminated.

CR8	CR7	CR6	CR5	CR4	CR3	CR2	CR1
X	X	X	X	X		X	X

8. Momentarily position the middle switch (S10) in the bottom row of switches to point DOWN and observe that the LEDs indicated by an X are illuminated in the following sequence. Note that this pattern is repeated.

	CR8	CR7	CR6	CR5	CR4	CR3	CR2	CR1
1.	X	X	X	X	X		X	X
2.						X		
1.	X	X	X	X	X		X	X
2.						X		

9. Momentarily position the middle switch (S10) in the bottom row of switches to point DOWN and observe that the LEDs indicated by an X are illuminated. (Note that the display does not blink.)

CR8	CR7	CR6	CR5	CR4	CR3	CR2	CR1
	X	X	X	X	X	X	

2.5 FINAL ASSEMBLY

At last, the board assembly is finished and the operational testing has been completed. You are now ready to use the Educator II. All that remains is to solder the leads for the recorder phone jacks, connect the power supply using either the optional connectors or by soldering wires directly to the board, install the rubber feet on the bottom of the case, and install the Educator II board in the case. These final assembly steps are provided in this paragraph.

1. Remove the two miniature phone jacks and their mounting hardware from the plastic envelope. Install each of the phone jacks into one of the holes in the side of the front cover of the case.
2. Connect the wire to the tape in (E2) and tape out (E1) terminals on the printed circuit board. These points are indicated on the printed circuit board and are shown in Figure 1-5.

NOTE

If you have purchased optional circuit board connectors, the phone jack connections may be made on them. The TAPE IN jack should be connected to connector P2 pin 3 and the TAPE OUT 3 jack should be connected to connector P2 pin M.

3. Remove the four rubber feet and eight sheet metal screws from the plastic envelope. Insert one sheet metal screw into each of the rubber feet and fasten the feet to the bottom of the case.
4. Solder the leads for the power supply into the holes provided on the printed circuit board and shown in Figure 1-5.

NOTE

If you have purchased optional connectors, the power supply connections should be made to them. The positive lead of the power supply should be connected to connector P2 pins 1 and A, while the negative lead should be connected to connector P2 pins 22 and Z. Pins 1 and A are located directly across from each other, one on each side of the board. Pins 22 and Z are located at the other end of the connector, directly across from each other.

5. Remove the lockwashers previously removed from the plastic envelopes containing the switches, and install one lockwasher on each of the switches.
6. Install the printed circuit board into the front cover of the case. (The flat locating washers may either be discarded or may be installed over the switches after the board has been installed into the cover and the switches are protruding from the front panel.)
7. Using the nuts supplied with the switches and a $^{5}/_{16}''$ box end wrench, install and tighten each nut onto the eleven switches.
8. Solder the wire connected to connector P2-3 to the TAPE IN phone jack and the wire connected to connector P2-M to the TAPE OUT jack.
9. Install the bottom of the case to the front cover and fasten the two halves of the case together with the four remaining sheet metal screws. (Be sure to feed the two power supply leads through the hole in the side panel of the bottom cover.)
10. Install the wire clamp onto the power supply leads and insert the wire clamp into the hole in the side of the case.

2.6 FAMILIARIZATION WITH FRONT PANEL SWITCHES AND INDICATORS

Now that your Educator II is finished, take a few moments to familiarize yourself with all of the front panel switches and indicators (refer to Figure 1-6). As new terms are introduced, a brief explanation will be included within parenthesis. After you have become familiar with the front panel, you'll want to proceed to the operation phase.

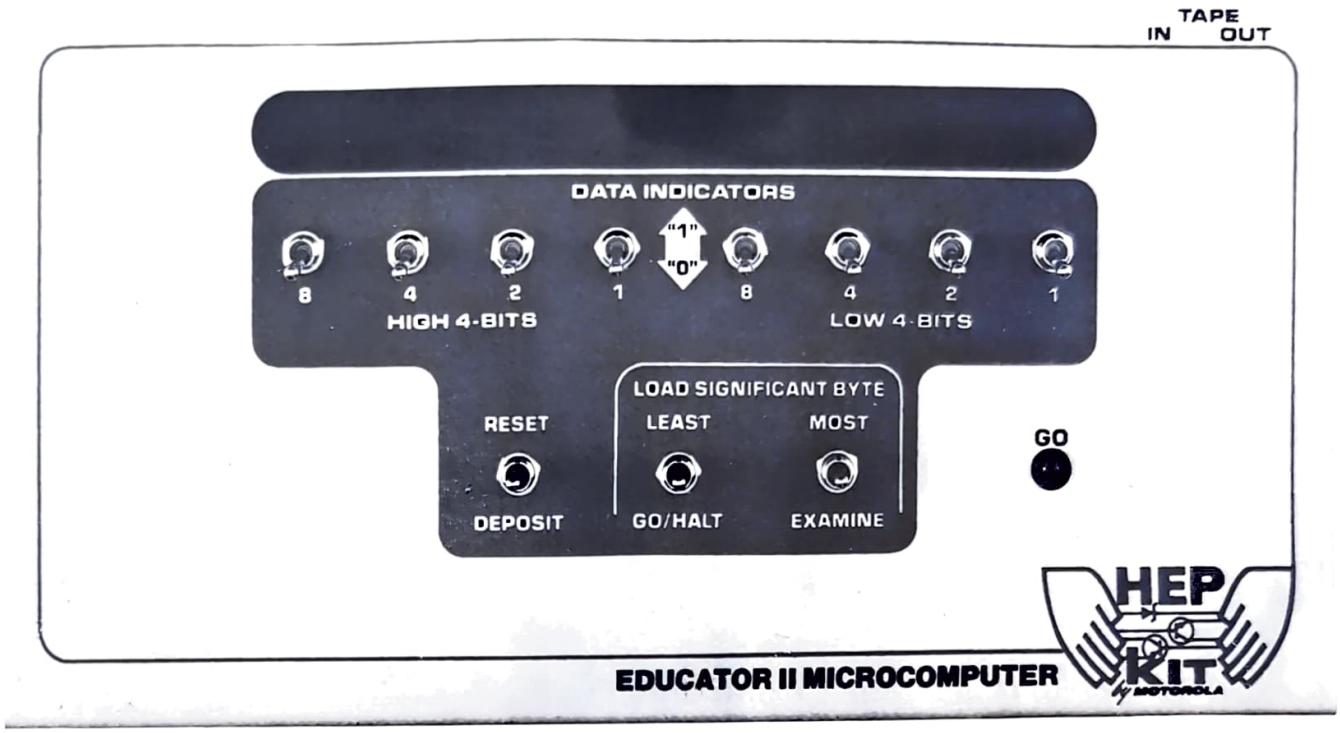


FIGURE 1-6. Educator II Front Panel View

DON'T! Take time to learn about the hardware that makes up the Educator II and the software that you can use to produce your programs. You'll find that operating the kit will be more enjoyable and accurate.

1. The eight switches located across the front panel and labeled 8 4 2 1 8 4 2 1 (read left to right) are used to enter your programs into the Educator II. Note that these switches are separated into two groups: the HIGH 4-BITS and the LOW 4-BITS. All of the programs entered into the Educator II use an 8-bit word (bit = high or low voltage levels, word = a group of bits). This word is normally divided into the high order 4 bits and the low order 4 bits. Whether the voltage level of each bit is a high level or a low level depends upon the position of these eight switches. When a switch is positioned to point UP, a high level voltage (+5Vdc) is produced. This is normally referred to as a logic 1 level. When a switch is positioned to point DOWN, a low level (ground) is produced. This is normally referred to as a logic 0 level. Note that the front panel indicates the position of the switch to produce these logic 1 and logic 0 levels. Thus, a group of eight logic 1s and logic 0s, in any combination, produces a word of instruction to the Microprocessing Unit (MPU).
2. The eight DATA INDICATORS located in a row and covered by the red plexiglass are used to read the data stored in the Educator II. When an indicator (LEDs CR1 through CR8) is illuminated, it indicates a logic 1 level. When extinguished, a logic 0 is indicated. Thus, by use of the eight switches and eight indicators, you can write your program into the Educator and read the results.
3. The RESET/DEPOSIT switch performs two functions: it provides a master system reset when positioned to RESET and it provides the means of depositing your program (which you entered on the front panel switches) into the system when positioned to DEPOSIT.
4. The LEAST/GO/HALT switch performs two system functions: it provides the means of loading in the least significant byte of the starting address (the location of the first byte of data in memory) when positioned to LEAST and it provides the means of stopping and starting execution (operation) of your program when positioned to GO/HALT.
5. The MOST/EXAMINE switch performs two system functions: it loads the most significant byte of the starting address, if required, when positioned to MOST and it permits you to examine the contents of memory in a step-by-step manner when positioned to EXAMINE. Since only 128 bytes of RAM are supplied in the kit with another 128 bytes optional, the MOST position of this switch is not normally required. However, if more memory is added, the MOST position would be required.
6. The GO indicator (CR9) illuminates when the program execution is halted and is *not* illuminated during execution.

Part 2
Instruction Phase

SECTION 1
HARDWARE

1.1 INTRODUCTION

The term "Hardware" refers to the individual devices and parts used to build a computer system. In the Educator II, this includes: the Microprocessing Unit (MPU), the Peripheral Interface Adapter (PIA), the Random Access Memory (RAM), and the Read Only Memory (ROM). The technology used to produce this Educator II hardware is the same as that used to produce the M6800 family of Large Scale Integration (LSI) parts. This technology includes nMOS (n-channel Metal Oxide Semiconductor) circuitry and advanced packaging techniques. A schematic diagram is included in Appendix A for the Educator II.

The MPU forms the nucleus of Educator II. It communicates with the rest of the system via a 16-bit address bus used to access up to 65,536 (2^{16}) separate memory locations. Data read from or written into these memory locations is then transferred to or from the MPU via an 8-bit, bidirectional data bus. A Read/ Write (R/W) output of the MPU controls the direction of all data transfers within the system.

The PIA provides a bidirectional Input/Output (I/O) interface between the MPU and all external equipment (such as your cassette recorder/player).

The RAM provides temporary storage for all of your programs entered into the system via the front panel switches, while the ROM is used to store a permanent set of instructions used to control the operation of the microcomputer system itself.

In the following paragraphs, a detailed description of each of the major elements used in the Educator II is provided to expand your knowledge of its internal operation. This knowledge will help you understand the architecture (or design) of your system and the instructions used to operate it.

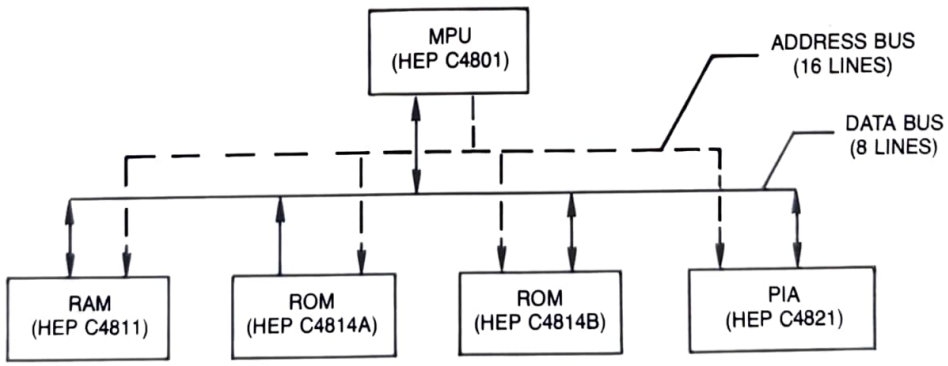
1.2 DESCRIPTION OF MICROPROCESSING UNIT (MPU)

The microprocessing unit (MPU) used in the Educator II is the HEP version of the well-known Motorola MC6800 MPU. This MPU is enclosed in a 40-pin package. A simplified diagram is provided in Figure 2-1 which shows, in block diagram form, the internal structure of the MPU and identifies all of its input and output signals. Each of these internal blocks and all of the input/output signals are described in the following paragraphs. A brief theory of operation for the MPU is then provided.

1.2.1 MPU INTERNAL STRUCTURE

The internal structure of the MPU consists of:

1. Two accumulators (A and B). Each accumulator is 8 bits (one byte) long. They are used to hold operands and data from an internal arithmetic logic unit.
2. One index register (X) of 16 bits (2 bytes) used primarily to store a memory address in the Indexed mode of memory addressing. The memory address stored in the index register may be either incremented or decremented.



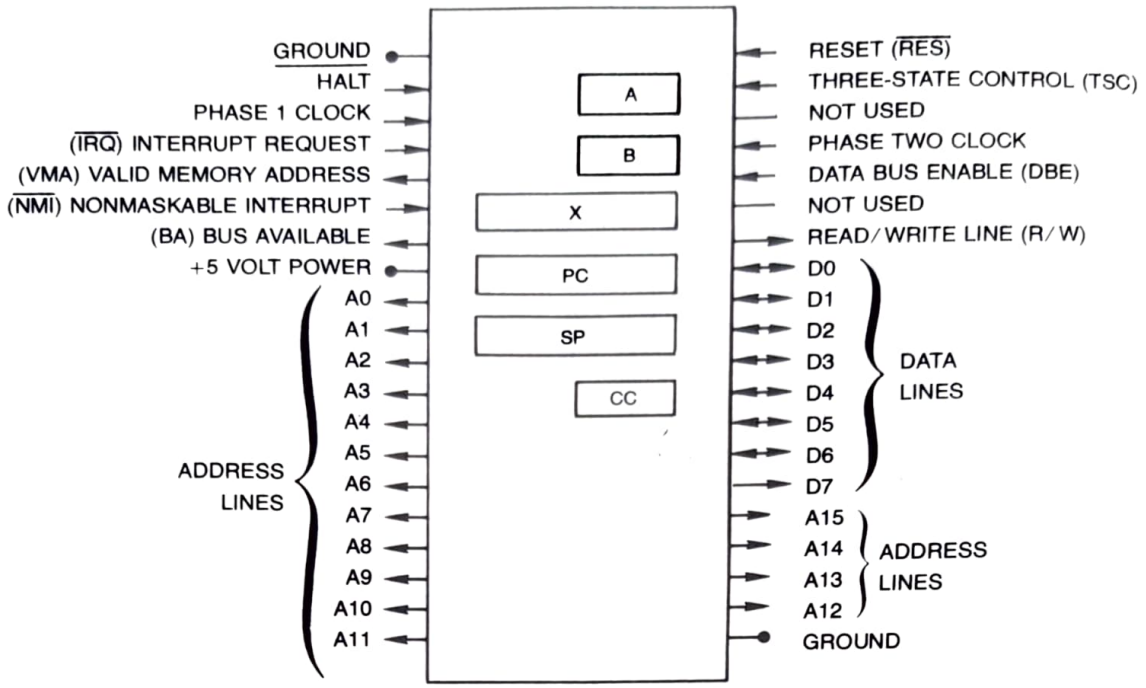


FIGURE 2-1. MPU Simplified Block Diagram

3. One Program Counter (PC) register of 16 bits (2 bytes) that contains the address of the next byte of data to be retrieved from memory. When the current address contained in the program counter is placed on the address bus by the MPU, the program counter will be automatically incremented (advanced one step).
4. One Stack Pointer (SP) register of 16 bits (2 bytes), used to store the first address of the seven memory locations into which the current contents of the five MPU registers (listed in Table 2-1) may be temporarily stored if the MPU has to leave the program routine being executed to perform some other function (such as an interrupt instruction). When the MPU is instructed to interrupt the routine currently being performed, the contents of the lowest byte of the Program Counter are loaded into the memory location specified by the address stored in the Stack Pointer. The address in the Stack Pointer is then decremented by one step and the highest byte of the Program Counter is then loaded into the memory location specified by the new address in the Stack Pointer. This action continues (decrement the address in the Stack Pointer, then load one byte of data from a register in the MPU into one byte of memory in the stack) until the contents of the last register in the MPU (the Condition Code Register) are loaded into the stack. The MPU will then execute the operation initiating the interrupt. After completing this operation, the MPU will then begin to restore the contents of its internal registers, starting with the last byte of data that had been loaded into the stack before the interrupt occurred (the contents of the Condition Code Register). The address stored in the Stack Pointer is then incremented by one step and the next byte of data stored in the stack is unloaded back into the next MPU register (the contents of Accumulator B). This action continues until the contents of the first register to be loaded into the stack (the contents of the lowest byte of the Program Counter) have been unloaded back into the register. Table 2-1 shows the sequence in which this register data is loaded into stack and the sequence used to unload data from the stack back into the registers. Thus, by using various interrupt and branch instructions, the MPU can be cleared to perform another routine. After that routine has been completed, the program can then be reentered at the interrupted step and completed. This technique is referred to as subroutine nesting and is used when writing programs in which one subroutine has multiple applications.

TABLE 2-1. Operating Sequence of the Stack Pointer

INTO STACK	STACK POINTER ADDRESS	MPU REGISTER DATA SAVED IN STACK	FROM STACK
1	Starting Address	Lower byte of Program Counter	7
2	Address — 1	Higher byte of Program Counter	6
3	Address — 2	Lower byte of Index Register	5
4	Address — 3	Higher byte of Index Register	4
5	Address — 4	Contents of Accumulator A	3
6	Address — 5	Contents of Accumulator B	2
7	Address — 6	Contents of Condition Code Register	1

5. One Condition Code Register (CCR) of 8 bits. Each individual bit gets set or cleared as the result of executing an instruction. The primary use of this register is in the execution of the conditional branch instruction. The information currently contained in the CCR depends upon the results of the last instruction executed that affected the CCR. A diagram showing the data contained in the CCR is provided in Figure 2-2. Each of these bits is explained in Table 2-2. Note that bits 6 and 7 of the Condition Code Register are not used.

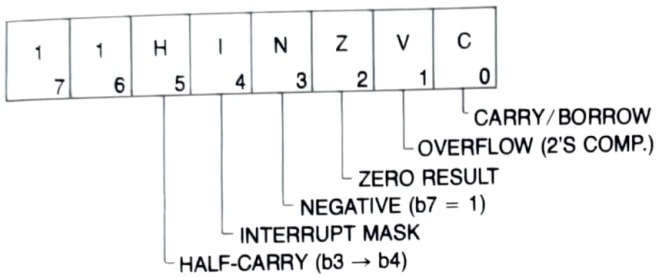


FIGURE 2-2. Condition Code Register Diagram

TABLE 2-2. Condition Code Register

BIT #	BIT NAME	DESCRIPTION
0	C bit Carry/Borrow	During addition of two binary numbers, this bit gets set (C = logic 1) to indicate that a carry has occurred (the binary result exceeds an 8-bit magnitude). If this bit is a logic 0, the carry bit is considered reset and no carry has occurred. Carry/Borrow Example: A = 1000 0000 B = 1000 0000 A + B = [1] 0000 0000
1	V bit (Overflow)	This bit is set (V = logic 1) whenever two's complement overflow results from an arithmetic operation and is reset (V = logic 0) if two's complement overflow does not occur. (Two's complement arithmetic is used by the MPU to perform binary subtraction. Thus, if a two's complement overflow occurs, the result will be a negative number.)
2	Z bit (Zero result)	This bit is set (Z = logic 1) if the result of an arithmetic operation is zero and is reset (Z = logic 0) when the result is anything but zero.
3	N bit (Negative)	This bit is set (N = logic 1) if bit 7 of an arithmetic operation is set (equal to 1). This indicates that the result of an arithmetic operation is negative. This bit is reset (N = logic 0) if bit 7 is equal to 0.
4	I bit (Interrupt Mask)	If this bit is set (I = logic 1), the MPU cannot respond to an interrupt requested by any external device.
5	H bit (Half-Carry)	This bit is set (H = logic 1) if, during the execution of any instruction, a carry from bit 3 to bit 4 occurs. This bit is reset (H = logic 0) if no carry occurs during execution of any instruction. Half-Carry Example: A = 0000 1000 B = 0000 1000 A + B = 0001 0000

1.2.2 MPU INPUT/OUTPUT SIGNALS

The MPU has 35 separate input/output signals. These are:

1. *Read/Write (R/W).* An output signal used to control data flow in the system. When the R/W signal is a logic 1 (the normal standby level of this signal), data is read into the MPU. When the R/W signal is a logic 0, data is written from the MPU into the addressed device.
2. *Valid Memory Address (VMA).* This output signal, when a logic 1, indicates to all devices connected to the address bus that there is a valid address on the address bus.
3. *Data Bus Enable (DBE).* This input signal, when a logic 1, permits data to be output during a write cycle. During an MPU read cycle, the data bus drivers will be internally disabled in the MPU.
4. *Interrupt Request ($\overline{IRQ}$).* This input from the PIA requests that an interrupt sequence be generated within the machine. The MPU will wait until it completes the current instruction being executed before it recognizes this request. At that time, if the interrupt mask bit in the Condition Code Register is not set (interrupt masked), the MPU will begin an interrupt sequence. At the start of the interrupt sequence, the Index Register, Program Counter, Accumulators, and Condition Code Register are stored away in the stack (refer to step 4 of paragraph 1.2.1). Next, the MPU will respond to the interrupt request by setting the interrupt mask bit high so that no further interrupts may occur. At the end of the cycle, a 16-bit address will be loaded onto the bus. This address accesses memory locations n-6 and n-7 in the ROM (where n is the address of the highest ROM address). The two bytes of data stored at address locations n-6 and n-7 comprise a vectoring address used by the MPU to branch to an interrupt routine contained in the ROM.
5. *Phase One (ϕ1) & Phase Two (ϕ2) Clocks.* These two pins are used for a two-phase, non-overlapping clock. In the Educator II, this clock runs at a rate of approximately 625 KHz (kilohertz).
6. *Restart ($\overline{RES}$).* This input is used to start the MPU after the power is initially applied or when the RESET/DEPOSIT switch on the front panel is positioned to RESET. When a positive edge is detected on the input, the MPU will begin the restart sequence by starting execution of a routine contained in the ROM to initialize the processor. All the higher order address lines will be forced high. For the restart, the last two highest memory locations in the ROM will be accessed (n and n-1). These two memory locations contain an address which is then loaded into the program counter to tell the MPU where program execution is to begin.

7. *Nonmaskable Interrupt (NMI).* This input requests that a nonmaskable-interrupt sequence (an interrupt that cannot be masked by the I bit in the Condition Code Register) be generated within the MPU. As with the Interrupt Request signal, the MPU will complete the current instruction being executed before it recognizes the NMI signal.

   The Index Register, Program Counter, Accumulators, and Condition Code Register are then stored away in the stack (refer to step 4 of paragraph 1.2.1). At the end of the cycle, a 16-bit address will be loaded onto the bus. This address accesses memory locations n-2 and n-3 in the ROM (where n is the address of the highest ROM address). The two bytes of data stored at address locations n-2 and n-3 comprise a vectoring address used by the MPU to branch to a nonmaskable interrupt routine contained in the ROM.

8. *Go/Halt (G/H).* When this input is at high level, the MPU will fetch the instruction addressed by the program counter and start execution. When low, all activity in the MPU will be halted. In the Halt mode, the MPU will stop at the end of an instruction. The Bus Available signal will be at a logic 1 level, the Valid Memory Address signal will be at a logic 0 level, and all other outputs will be effectively disconnected during a Halt.

   The Halt line must go low with the leading edge of the phase one signal (ϕ1) to insure single instruction operation. If the halt line does not go low with the leading edge of phase one, one or two instruction operations may result, depending on when the halt line goes low in relation to the phasing of the clock.

9. *Bus Available (BA).* The Bus Available signal will normally be in the logic 0 state. When activated, it will go to the logic 1 state, indicating that the MPU has stopped and that the address bus is available. This will occur if the GO/HALT line is in the Halt (low) mode or the MPU is in a "Wait" state as the result of some instruction.

10. *Three-State Control (TSC).* This input causes all of the address bus outputs and the Read-Write line to go into the off or high impedance state (effectively disconnecting them from the circuit). The Valid Memory Address and Bus Available signals will also be forced low. The data bus is not affected by TSC and has its own enable (Data Bus Enable). The ϕ1 clock must be held in the high state for this function to operate properly. The address bus will then be available for other devices to directly address memory. Since the MPU is a dynamic device, it must be refreshed periodically or destruction of data stored in the MPU registers will occur.

11. *Address Bus (A0-A15).* Sixteen pins are used for the address bus. All of the outputs have three-state bus drivers. When the output is turned off, it is essentially an open circuit.

12. *Data Bus (D0-D7).* Eight pins are used for the data bus. This bus is bidirectional, transferring data to and from the memory and peripheral devices. It also has three-state output buffers to effectively remove them from the bus.

1.2.3 MPU THEORY OF OPERATION

Up to this point, only the physical aspects of the MPU have been described. It is now time to explain how the MPU operates. In order to do this in the most understandable sequence, the theory of operation begins with a discussion of the two-phase MPU clock, since all operations performed by the MPU are initiated and executed with these two clock phases. The restart operation is described next, since this is the first operation performed upon application of power. This is followed by a description of the various MPU operations and, finally, the MPU interrupt operation is described.

1.2.3.1 Clock Operation. The MPU requires two symmetrical non-overlapping clocks. This is accomplished by generating two separate clock phases: ϕ1 and ϕ2. In the Educator II, the clock frequency is approximately 625 KHz. Figure 2-3 shows the relationship of each of these clock phases.

During phase 1 (ϕ1 is a logic high), an address will be placed on the address bus by the MPU, and the VMA

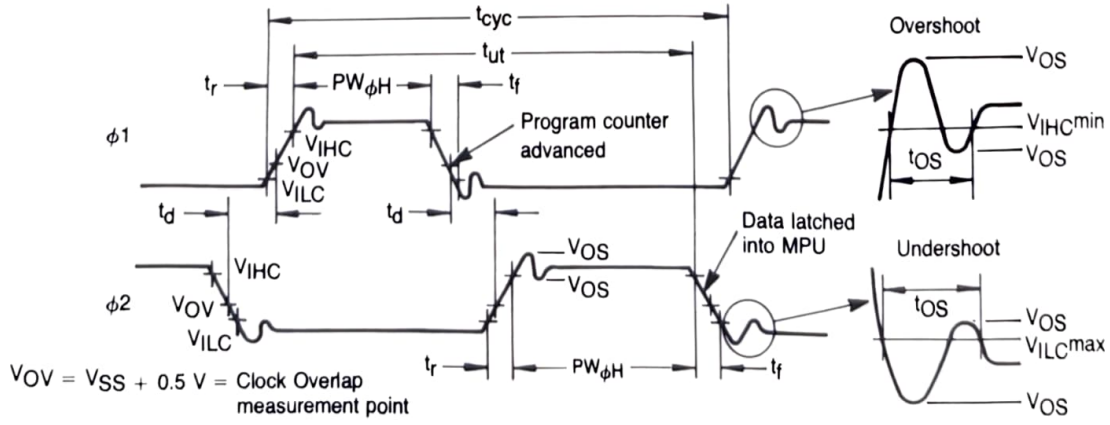


FIGURE 2-3. MPU Clock Waveforms

signal will go to a logic 1 level, indicating that a valid address has been placed on the bus. This address is obtained from either the Program Counter, Stack Pointer, or Index register, depending upon the instruction decoded by the MPU. During the high to low transition of the $\phi 1$ clock pulse (the trailing edge), the Program Counter will be automatically incremented one step (if required by the program). When the $\phi 2$ clock pulse changes to a logic high level, the bidirectional data bus will be enabled to transfer data between the MPU and an external device (such as the ROM, RAM, or PIA). When the trailing edge of the $\phi 2$ clock pulse occurs (high to low transition), the data on the data bus is latched (stored) into the MPU. This data is then decoded by the MPU to determine the operation to be performed. This operation continues until a Halt instruction is detected or power is removed from the system. If a Halt is detected, the $\phi 1$ clock will be held at a logic high level and the $\phi 2$ clock will be held at a logic low level until the Halt is removed.

1.2.3.2 Restart Operation. A restart (or reset) sequence is initiated every time power is applied or the RESET/DEPOSIT switch on the front panel of the Educator II is momentarily positioned to RESET. When a restart is initiated, the Reset signal goes to a logic low level for a minimum of eight $\phi 1$ and $\phi 2$ clock cycles, and the Halt signal is held at a logic high level (inhibiting the Halt function). During this time, the I bit (Interrupt mask bit) in the Condition Code Register will be set to a logic high level to prevent the MPU from responding to an interrupt instruction. The data loaded at the second highest and the highest locations in the ROM memory will then be loaded into the higher and lower byte, respectively, of the Program Counter. The data now contained in the Program Counter is the starting address of the initialization program stored in the ROM. When the first $\phi 1$ clock pulse occurs (changes to a logic 1 level), this starting address is loaded from the Program Counter onto the address bus, and the VMA signal changes to a logic high level. Thus, when this starting address is loaded onto the address bus, the byte of data stored at that starting address will be loaded onto the data bus by the $\phi 2$ clock and latched into the MPU. The MPU then decodes this byte of data to determine the next operation to be performed. (Remember, the address stored in the Program Counter is incremented by one step every time an address is loaded onto the address bus.) The next address (starting address + 1) in the Program Counter is then loaded onto the address bus to retrieve the second byte of the initialization program. This operation continues sequentially through the initialization routine stored in the ROM until the complete program has been executed. The microcomputer system starting conditions have now been set by the initialization routine, permitting your program to be entered and executed.

TABLE 2-3. Executable Instructions

ABA	ADD ACCUMULATORS
ADC	ADD WITH CARRY
ADD	ADD WITHOUT CARRY
AND	LOGICAL AND
ASL	ARITHMETIC SHIFT LEFT
ASR	ARITHMETIC SHIFT RIGHT
BCC	BRANCH IF CARRY CLEAR
BCS	BRANCH IF CARRY SET
BEQ	BRANCH IF EQUAL TO ZERO
BGE	BRANCH IF GREATER OR EQUAL TO ZERO
BGT	BRANCH IF GREATER THAN ZERO
BHI	BRANCH IF HIGHER
BIT	BIT TEST
BLE	BRANCH IF LESS OR EQUAL TO ZERO
BLS	BRANCH IF LOWER OR SAME
BLT	BRANCH IF LESS THAN ZERO
BMI	BRANCH IF MINUS
BNE	BRANCH IF NOT EQUAL TO ZERO
BPL	BRANCH IF PLUS
BRA	BRANCH ALWAYS
BSR	BRANCH TO SUBROUTINE
BVC	BRANCH IF OVERFLOW CLEAR
BVS	BRANCH IF OVERFLOW SET
CBA	COMPARE ACCUMULATORS
CLC	CLEAR CARRY
CLI	CLEAR INTERRUPT MASK
CLR	CLEAR
CLV	CLEAR OVERFLOW
CMP	COMPARE
COM	COMPLEMENT
CPX	COMPARE INDEX REGISTER
DAA	DECIMAL ADJUST ACCUMULATOR A
DEC	DECREMENT
DES	DECREMENT STACK POINTER
DEX	DECREMENT INDEX REGISTER
EOR	EXCLUSIVE OR
INC	INCREMENT
INS	INCREMENT STACK POINTER
INX	INCREMENT INDEX REGISTER
JMP	JUMP
JSR	JUMP TO SUBROUTINE
LDA	LOAD ACCUMULATOR
LDS	LOAD STACK POINTER
LDX	LOAD INDEX REGISTER
LSR	LOGICAL SHIFT RIGHT
NEG	NEGATE
NOP	NO OPERATION
ORA	INCLUSIVE OR
PSH	PUSH DATA
PUL	PULL DATA
ROL	ROTATE LEFT
ROR	ROTATE RIGHT
RTI	RETURN FROM INTERRUPT
RTS	RETURN FROM SUBROUTINE
SBA	SUBTRACT ACCUMULATORS
SBC	SUBTRACT WITH CARRY
SEC	SET CARRY
SEI	SET INTERRUPT MASK
SEV	SET OVERFLOW
STA	STORE ACCUMULATOR
STS	STORE STACK POINTER
STX	STORE INDEX REGISTER
SUB	SUBTRACT
SWI	SOFTWARE INTERRUPT
TAB	TRANSFER ACCUMULATORS
TAP	TRANSFER ACCUMULATORS TO CONDITION CODE REG
TBA	TRANSFER ACCUMULATORS
TPA	TRANSFER CONDITION CODE REG TO ACCUMULATOR A
TST	TEST
TSX	TRANSFER STACK POINTER TO INDEX REGISTER
TXS	TRANSFER INDEX REGISTER TO STACK POINTER
WAI	WAIT FOR INTERRUPT

1.2.3.3 MPU Operation. The byte of data loaded into the MPU from any address location may be either an operation code (op code) or data. Seventy-two op codes (or executable instructions) can be decoded by the MPU. A single byte of data can represent any decimal number from 0 to 255 (256 steps). When writing MPU programs in machine language (binary representations of the 72 executable instructions), you may use any of these 72 instructions to obtain the results you desire. The only additional requirement is to determine the addressing mode to be used. The 72 executable instructions are listed, in alphabetic order, in Table 2-3.

Many of the 72 executable instructions can be used in one of six different MPU addressing modes. These addressing modes are: Inherent/Accumulator, Immediate, Direct, Extended, Indexed, and Relative. By using a combination of the 72 executable instructions with the six different operating modes, the MPU can perform 192 individual operations. (Note that not all 72 instructions can be used in all six addressing modes.) In addition to determining (along with the executable instructions) the operation to be performed by the MPU, the addressing modes are also used to determine the number of total bytes needed to perform each instruction. The following steps describe each of the six addressing modes.

1. *Inherent/Accumulator Addressing Mode.* Instructions used in this addressing mode require only one byte of data (the executable instruction only) because they are used to control only the internal registers in the MPU (thus, there is no need for an address) or because the addressing information is inherently contained in the instruction. Fifty-one executable instructions may be used in the Inherent/Accumulator addressing mode. Table 2-4 lists each in mnemonic form. Note that only those instructions indicated by an asterisk (*) are inherently addressed, while the remaining instructions are used to directly manipulate data contained in the five MPU registers.

TABLE 2-4. Inherent/Accumulator Addressing Mode Instructions.

*NOP	CLI	INS	*WAI	DECA	ASLB
TAP	SEI	PULA	*SWI	INCA	ROLB
TPA	SBA	PULB	NEGA	TSTA	DECB
INX	CBA	DES	COMA	CLRA	INCB
DEX	TAB	TXS	LSRA	NEGB	TSTB
CLV	TBA	PSHA	RORA	COMB	CLRB
SEV	DAA	PSHB	ASRA	LSRB	
CLC	ABA	*RTS	ASLA	RORB	
SEC	TSX	*RTI	ROLA	ASRB	

*Inherent instructions.

2. *Immediate Addressing Mode.* Instructions used in this addressing mode require two or three bytes of data, depending upon the specific instruction being used. This addressing mode is used to immediately perform an instruction. Twenty-three executable instructions may be used in the Immediate addressing mode. Table 2-5 lists each in mnemonic form. Note that only those instructions indicated by an asterisk (*) require three bytes; all the rest need only two. Three bytes are necessary when loading a 16-bit address into either the index register or the stack pointer (instruction, higher byte, lower byte).

TABLE 2-5. Immediate Addressing Mode Instructions.

SUBA	EORA	SUBB	ADCB
CMPA	ADCA	SBCB	ORAB
SBCA	ORAA	ANDB	ADDB
ANDA	ADDA	BITB	*LDX
BITA	*CPX	LDAB	CMPB
LDAA	*LDS	EORB	

*Instructions requiring 3 bytes. All others need only 2 bytes.

3. *Direct Addressing Mode.* Instructions used in this addressing mode require only two bytes of data. This addressing mode is used to directly address a memory location within a range of 256 address locations from the current address location. Twenty-seven executable instructions may be used in the Direct addressing mode. Table 2-6 lists each in mnemonic form.

TABLE 2-6. Direct Addressing Mode Instructions.

SUBA	EORA	SUBB	EORB
CMPA	ADCA	CMPB	ADCB
SBCA	ORAA	SBCB	ORAB
ANDA	ADDA	ANDB	ADDB
BITA	CPX	BITB	LDX
LDAA	LDS	LDAB	STX
STAA	STS	STAB	

4. *Extended Addressing Mode.* Instructions used in this addressing mode require three bytes of data. This addressing mode is used exactly like the Direct addressing mode, except that the range of directly addressable memory locations is extended from 255 to the full range of 65,536. Forty executable instructions may be used in the Extended addressing mode. Table 2-7 lists each in mnemonic form.

TABLE 2-7. Extended Addressing Mode Instructions.

SUBA	ADCA	SBCB	ADDB	ASL
CMPA	ORAA	ANDB	LDX	ROL
SBCA	ADDA	BITB	STX	DEC
ANDA	CPX	LDAB	NEG	INC
BITA	LDS	STAB	COM	TST
LDAA	STS	EORB	LSR	JMP
STAA	SUBB	ADCB	ROR	CLR
EORA	CMPB	ORAB	ASR	JSR

5. *Indexed Addressing Mode.* Instructions used in this addressing mode require two bytes of data. The second byte contains an offset number which is added to the contents of the index register to form a new address. The new address is the location in memory which contains the data for the operation or is the destination for

the data. The new address is held in a temporary address register so as not to alter the original contents of the index register. Forty executable instructions may be used in the Indexed addressing mode. Table 2-8 lists each in mnemonic form.

TABLE 2-8. Indexed Addressing Mode Instructions.

NEG	INC	BITA	JSR	LDAB
COM	TST	LDAA	LDS	STAB
LSR	JMP	STAA	STS	EORB
ROR	CLR	EORA	SUBB	ADCB
ASR	SUBA	ADCA	CMPB	ORAB
ASL	CMPA	ORAA	SBCB	ADDB
ROL	SBCA	ADDA	ANDB	LDX
DEC	ANDA	CPX	BITB	STX

6. *Relative Addressing Mode.* Instructions used in this addressing mode must only be used to transfer the program to some place other than the next sequential address (branch instructions). Transfers are limited to 126 memory locations back or 129 memory locations forward from the current address location. This instruction requires two bytes of data. Sixteen executable instructions may be used in the Relative addressing mode. Table 2-9 lists each in mnemonic form.

TABLE 2-9. Relative Addressing Mode Instructions.

BRA	BEQ	BLT
BHI	BVC	BGT
BLS	BVS	BLE
BCC	BPL	BSR
BCS	BMI	
BNE	BGE	

1.2.3.4 Interrupt Operation. In addition to all of the MPU operations previously described, one other operation can be executed that must be explained: interrupts.

Two types of interrupts can be executed by the MPU: a hardware interrupt (either maskable or nonmaskable) and a software interrupt. A hardware interrupt is an interrupt initiated by some hardware action that occurs external to the microcomputer system. These types of interrupts may be of either the maskable ($\overline{IRQ}$) or nonmaskable ($\overline{NMI}$) variety. A maskable hardware interrupt ($\overline{IRQ}$) will be ignored if the Interrupt mask bit (I bit) in the MPU Condition Code Register (bit 4) has been previously set to a logic 1 level. If this bit is not set, the maskable hardware interrupt will be honored. A nonmaskable hardware interrupt ($\overline{NMI}$) will always be honored by the MPU, regardless of whether the I bit in the Condition Code Register is set or reset. Interrupts of this type cannot be masked (ignored).

A software interrupt is an interrupt that is initiated by an instruction in the program ($\overline{SWI}$). This type of interrupt cannot be masked.

When any of these three types of interrupts ($\overline{IRQ}$, $\overline{NMI}$, or $\overline{SWI}$) is detected by the MPU, the MPU will first complete the current instruction being executed and then the contents of the five MPU registers will be sequentially pushed (decremented) into the stack. (This operation is more fully explained in step 4 of paragraph 1.2.1.) The interrupt routine will then be initiated.

The interrupt routine is normally a program written by you, the machine's programmer. The first thing that any interrupt routine must do is to determine the type of interrupt. If the type of interrupt is determined to be a nonmaskable interrupt ($\overline{NMI}$), the mask bit (I bit) is set (to prevent another interrupt from being honored by the MPU). The addresses of the third and second highest memory locations in the ROM are then loaded into the higher and lower bytes, respectively, of the Program Counter. If the type of interrupt is determined to be a software interrupt ($\overline{SWI}$), the mask bit (I bit) is set. This time, however, the fifth and fourth highest memory locations in the ROM are then loaded into the higher and lower bytes, respectively, of the Program Counter. If the type of interrupt is determined to be an interrupt request ($\overline{IRQ}$), the mask bit is set and the seventh and sixth highest memory locations in the ROM are loaded into the higher and lower bytes, respectively, of the Program Counter. You will notice that each of the three interrupts retrieves a 16-bit address (a vector address) from a location in the ROM, and that these vector address locations are next to each other (contiguous) in the ROM memory. The memory map illustrated in Figure 2-4 shows the locations in the ROM where these three vector addresses are stored. (An address map is a diagram used to show how the various address locations relate to each other.)

After the selected interrupt vector has been loaded into the Program Counter, the address is then loaded onto the address bus to continue the interrupt routine. After the interrupt routine has been completed, the MPU returns to execute the program that was being executed at the time the interrupt occurred. For this reason, when writing an interrupt routine, the programmer must remember to use an RTI instruction (Return from Interrupt) as the last instruction of the interrupt routine. If this is not done, the MPU cannot return to the program being executed when the interrupt occurred.

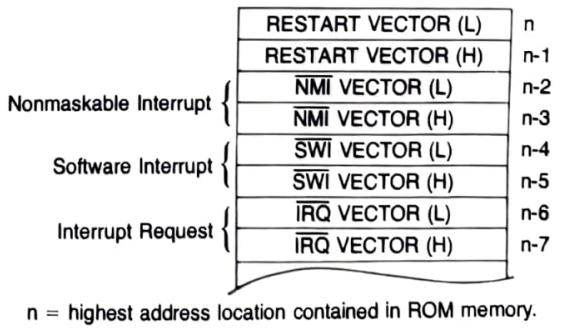


FIGURE 2-4. Interrupt Vectors Memory Map

When the MPU returns to the main program from an interrupt routine, the data must first be pulled (incremented) from the stack to restore the contents of the five MPU registers to their condition prior to the occurrence of the interrupt. (A more detailed description of this operation is provided in step 4 of paragraph 1.2.1.) The MPU can now continue execution of the original program.

1.3 DESCRIPTION OF PERIPHERAL INTERFACE ADAPTER (PIA)

The Peripheral Interface Adapter (PIA) used in the Educator II is the HEP version of the well-known Motorola MC6820 PIA. The PIA is enclosed in a 40-pin package and is used to interface external (peripheral) equipment (such as your cassette recorder/player). The PIA communicates with the MPU via an 8-bit bidirectional data bus and with the peripheral equipment via two 8-bit, bidirectional data buses. Each input/output line within each of the peripheral data buses may be programmed to act as an input line or an output line. A simplified block diagram of the PIA is provided in Figure 2-5.

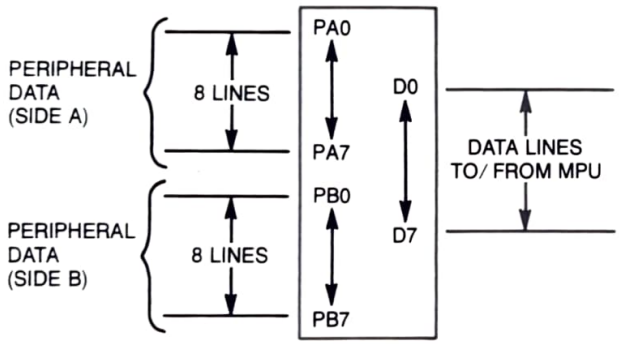


FIGURE 2-5. PIA Block Diagram

1.3.1 PIA INTERNAL STRUCTURE

The internal structure (shown in Figure 2-6) of the PIA consists of:

1. Two Data Direction Registers (DDRA and DDRB), used to determine the direction of data flow on each individual peripheral data line (PA0 through PA7 and PB0 through PB7). This is accomplished by the MPU loading a logic 1 or logic 0 into the eight bit positions of the DDR. A logic 1 or logic 0 causes the corresponding peripheral data line to function as an output or an input, respectively.
2. Two Output Registers (ORA and ORB), used to store data on the MPU data bus during an MPU write operation (only those registers that are programmed to be outputs will store data from the MPU data bus during a write operation). During an MPU read operation, the data present on peripheral lines programmed as inputs is transferred directly to the MPU data bus.
3. Two Control Registers (CRA and CRB) allow the MPU to establish and control the operating modes of the peripheral control lines, CA1, CA2, CB1, and CB2. These four lines, in conjunction with the two Interrupt Status Control circuits (A and B), control the information passed back and forth between the MPU and the peripheral equipment.
4. Two Peripheral Interface circuits (A and B). These two circuits are used to provide both input and output interfacing for each of the peripheral data lines. Due to differing circuitry between the A and B sides, the results of reading positions programmed as outputs differ slightly. During an MPU read of the A side, data present on the peripheral lines will be placed onto the MPU data bus, regardless of whether the lines are programmed as outputs or inputs. On the B side, output buffering prevents this from occurring.
5. One set of Data Bus Buffers (DBB) used to interface the internal PIA circuitry with the MPU data bus.
6. One Bus Input Register (BIR) used to provide temporary storage of each byte of data loaded into the PIA from the MPU data bus.
7. One set of Chip Select and Read/Write Control circuitry used to select the side of the PIA that will be used (A or B side) and to control whether a read or write operation will be performed.

1.3.2 PIA INPUT/OUTPUT SIGNALS

The PIA has 36 separate input/output signals. These are:

1. *Peripheral Data Lines PA0 through PA7.* Each of these eight data lines, which interface with the outside world, can be programmed to act as either an input or an output. This is accomplished by setting a logic 1 into the corresponding bit in the Data Direction Register (DDR) if the line is to be an output, or a logic 0 into the DDR if the line is to be an input. When the data on the peripheral data lines are read into the MPU by a load instruction, those lines which have been designated as input lines (0 in the DDR) will be gated directly to the data bus and into the selected register in the MPU.

   On the other hand, when an output data instruction (such as a Store A into the PIA-STAA PIA) is executed by the MPU, data will be transferred via the data bus to the peripheral data register. A logic 1 output will cause a logic high level on the corresponding data line, and a logic 0 output will cause a logic low level on the corresponding data line.
2. *Peripheral Data Lines PB0 through PB7.* The eight data lines which interface with the outside world on the B side may be programmed to act as either an input or an output in the same manner as described in the preceding step. The output buffers driving these lines have three-state capability, allowing them to be effectively disconnected when the peripheral data line is

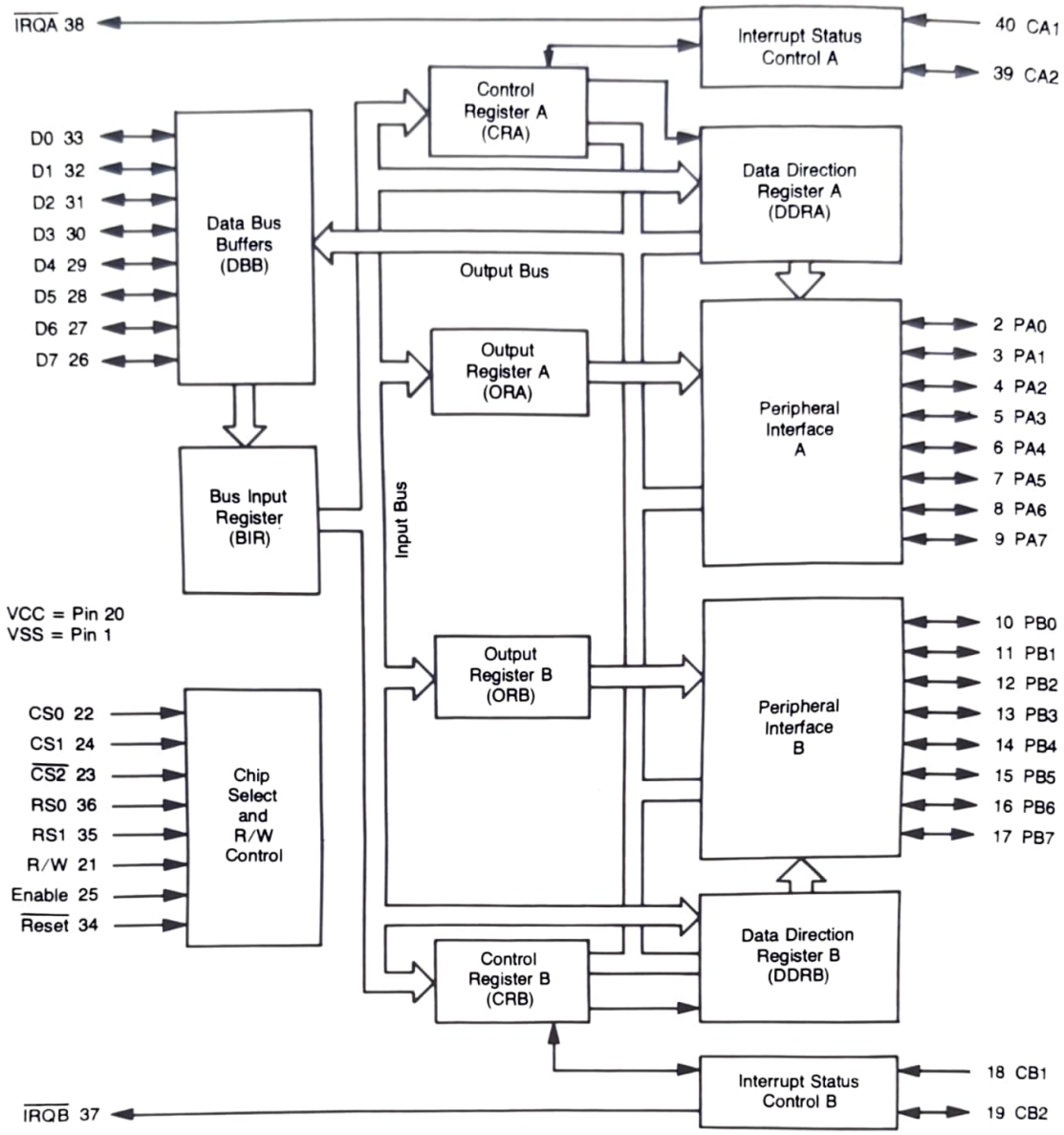


FIGURE 2-6. PIA Internal Structure Block Diagram

used as an input. (Three-state capability means that, beside the normal logic 0 and logic 1 states for digital circuits, this device also has the capability of a third, high-impedance state. When operated in the high-impedance state, the device is effectively removed from the circuit.)

3. *Data lines D0 through D7.* These eight, bidirectional data lines permit the transfer of the data to/from the PIA and the MPU. The MPU receives data and sends data from and to the outside world via these eight lines. The data output drivers are three-state devices that remain in the high-impedance (off) state except when the MPU performs a PIA read operation.
4. *Chip Select Lines (CS0, CS1, $\overline{CS2}$).* These lines are tied to the address lines of the MPU. It is through these lines that a particular PIA is selected (addressed). For selection of a PIA, the CS0 and CS1 lines must be high and the $\overline{CS2}$ must be low. After the chip selects have been addressed, they must be held in that state for the duration of the E (enable) pulse, which is the only timing signal supplied by the MPU to the PIA. This enable pulse (E) is normally the $\phi 2$ clock.
5. *Enable Line (E).* The enable pulse (E) is the only timing signal that is supplied to the PIA by the MPU. Timing on all other signals is referenced to the leading or trailing edges of the E pulse.

6. *Reset Line (RS).* This line is used to reset all registers in the PIA to a logic 0. This is normally used during a reset or power on operation. The normal state of this line is a logic 1 level. The transition of high to low to high resets all registers in the PIA.
7. *Read/Write Line (R/W).* This signal is generated by the MPU to control the direction of the data transfers on the Data Bus. A logic low level on the PIA Read/Write line enables the input buffers, and data is transferred from the MPU to the PIA (MPU write) on the falling edge of the E (ϕ2) signal (if the device has been selected). A high on the Read/Write line sets up the PIA for a transfer of data to the data bus (MPU read). The PIA output buffers are enabled when the proper address and the enable pulse are present, thus transferring data to the MPU.
8. *Interrupt Request Lines ($\overline{IRQA}$ and $\overline{IRQB}$).* These lines are used to interrupt the MPU (either directly or indirectly) through the use of interrupt priority circuitry. All interrupt request lines may be tied together. Interrupts are serviced by a software routine that sequentially reads and tests the two control registers in each PIA looking for interrupt flag bits (Bits 6 and 7) that are set. When the MPU reads the Peripheral Data Register, the Interrupt Flags (Bit 6 and Bit 7) are cleared and the Interrupt Request is cleared.
9. *Interrupt Input Lines (CA1 and CB1).* These input only lines to the PIA set the interrupt flag (Bit 7) of the control registers in the PIA.
10. *Peripheral Control Lines (CA2 and CB2).* These lines can be programmed to act either as an interrupt input or as a peripheral output. The function of these lines is controlled by Control Registers A and B (Bits 3, 4, and 5).

1.3.3 CONTROL REGISTER A (CRA).

During the following description of Control Register A, refer to the Control Register A diagram provided in Figure 2-7.

7	6	5	4	3	2	1	0
IRQA1	IRQA2	CA2 CONTROL			DDRA	CA1 CONTROL	

FIGURE 2-7. Control Register A (CRA)

1.3.3.1 CA1 Control (Bits 0 and 1). Peripheral control line CA1 is an input only line which may be used to initiate an interrupt by setting the interrupt flag IRQA1 (Bit 7) of Control Register A (CRA). Bits 0 and 1 of CRA are used to determine how the interrupt is to be handled.

After the MPU reads Peripheral Data Register A, the IRQA1 and IRQA2 (Bits 6 and 7) will be cleared.

As shown in the chart at the bottom of the page, Bit 1 is the Edge Programming Bit. A logic 0 in bit 1 programs the interrupt flag bit 7 (IRQA1) to respond to a negative transition (edge) on CA1.

Bit 0 of the control register is the CA1 interrupt Mask Programming Bit. If Bit 0 is a logic 0, the setting of the interrupt flag bit 7 (IRQA1) will not allow the interrupt pin $\overline{IRQA}$ to go low. If bit 0 contains a logic 1, the $\overline{IRQA}$ pin will go low when the flag bit 7 changes to a logic 1 level.

1.3.3.2 Data Direction Access Control (DDRA)-(Bit 2). This bit, in conjunction with the register select lines (RS0 and RS1), is used to select either the Peripheral Data Register or the Data Direction Register. To address the A side control register, RS1 is set to a logic 0 and RS0 is set to a logic 1.

RS1	RS0	CRA (Bit 2)	Register Selected
0	0	1	Peripheral Data Register A
0	0	0	Data Direction Register A
0	1	X	Control Register A

Transition of Interrupt Input Line CA1	Status of Bit 1 in CRA (Edge)	Status of Bit 0 in CRA (Mask)	IRQA1 (Interrupt Flag) Bit 7 of CRA	Status of $\overline{IRQA}$ Line (MPU Interrupt Request)
	0	0	1	MASKED (Remains High)
	0	1	1	GOES LOW (Processor Interrupted)
	1	0	1	MASKED (Remains High)
	1	1	1	GOES LOW (Processor Interrupted)

(All other combinations of CA1 transition and status of bit 0 and bit 1 will be ignored.)

Transition of Input CA2	Status of Bit 5 in CRA (I/O Control)	Status of Bit 4 in CRA (Edge)	Status of Bit 3 in CRA (Mask)	IRQA2 (Interrupt Flag) Bit 6 of CRA	Status of IRQA Line (MPU Interrupt Request)
	0	0	0	1	MASKED (Remains High)
	0	0	1	1	GOES LOW (Processor Interrupted)
	0	1	0	1	MASKED (Remains High)
	0	1	1	1	GOES LOW (Processor Interrupted)

(All other combinations of CA2 transition and status of bit 3 and bit 4 will be ignored.)

Bit 5 of Control Register A is the Input/Output Programming Bit for CA2. If Bit 5 contains a logic 0, CA2 is programmed as an interrupt input line. When this occurs, the programming of bits 4 and 3 have the same usage as bits 1 and 0. Bit 4 is the Edge Programming Bit for CA2. Bit 3 is the Interrupt Mask Bit for CA2.

1.3.3.3 CA2 Control (Bits 3, 4, and 5) as an Interrupt Input. When Bit 5 is a logic 1, CA2 is programmed as an output, and bits 4 and 3 are used to program one of the following three modes of operation: Handshake Mode, Pulse Mode, or Bit 3 Following Mode. When used in the Handshake Mode or Pulse Mode, refer to figures 2-8 and 2-9 for illustrations showing the operation of the CA2 output. However, if the Bit 3 Following Mode is used, the output level of CA2 will follow the logic level of bit 3 (Bits 4 and 5 must be logic 1 level.).

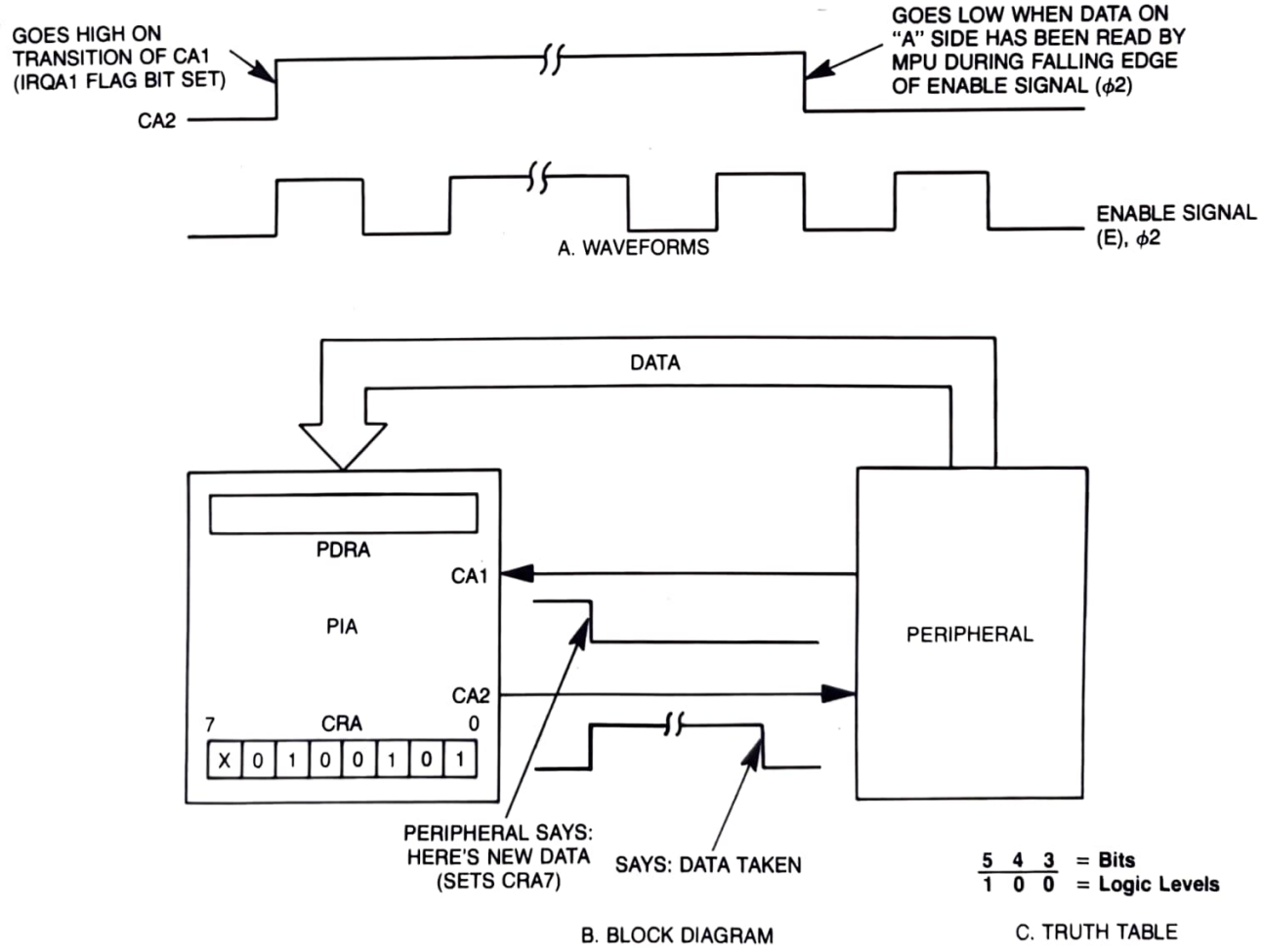


FIGURE 2-8. Handshake Mode Diagram (A Side)

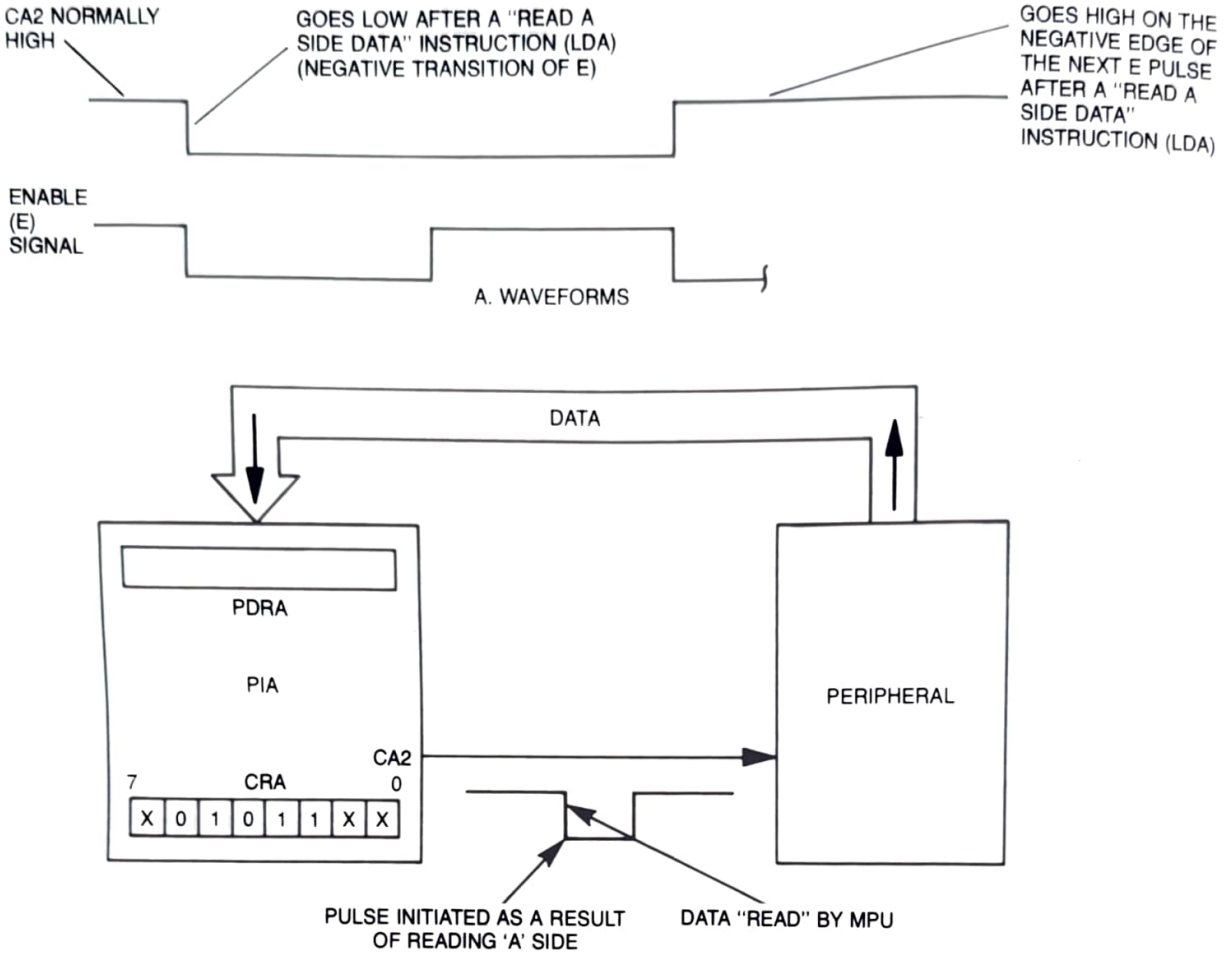


FIGURE 2-9. Pulse Mode Diagram (A Side)

1.3.4 CONTROL REGISTER B (CRB)

During the following description of Control Register B, refer to the Control Register B diagram provided in Figure 2-10.

7	6	5	4	3	2	1	0
IRQB1	IRQB2	CB2 CONTROL			DDRB	CB1 CONTROL	

FIGURE 2-10. Control Register B (CRB)

1.3.4.1 CB1 Control (Bits 0 and 1) Peripheral control line CB1 is an input only line which may be used to initiate an interrupt by setting the interrupt flag IRQB1 (Bit 7) of Control Register B (CRB). Bits 0 and 1 of CRB are used to determine how the interrupt is to be handled.

After the MPU reads Peripheral Data Register B, the IRQB1 (Bit 7) and IRQB2 (Bit 6) will be cleared.

Transition of Interrupt Input Line CB1	Status of Bit 1 in CRB (Edge)	Status of Bit 0 in CRB (Mask)	IRQB1 (Interrupt Flag) Bit 7 of CRB	Status of $\overline{\text{IRQB}}$ Line (MPU Interrupt Request)
	0	0	1	MASKED (Remains High)
	0	1	1	GOES LOW (Processor Interrupted)
	1	0	1	MASKED (Remains High)
	1	1	1	GOES LOW (Processor Interrupted)

(All other combinations of CB1 transition and status of bit 0 and bit 1 will be ignored.)

Bits 1 and 0 of Control Register B have the same programming use and logic as Bits 1 and 0 of the Control Register A; that is, bit 1 is for Edge Programming Bit for CB1 and bit 0 is the Interrupt Flag Mask Bit for CB1.

1.3.4.2 Data Directon Access Control (DDRB)-(Bit 2). This bit, in conjunction with the register select lines (RS0 and RS1), is used to select either the Peripheral Data Register or the Data Direction Register. To address the B side control register, RS1 is set to a logic 1 and RS0 is set to a logic 1.

RS1	RS0	CRB (Bit 2)	Register Selected
1	0	1	Peripheral Data Register B
1	0	0	Data Direction Register B
1	1	X	Control Register B

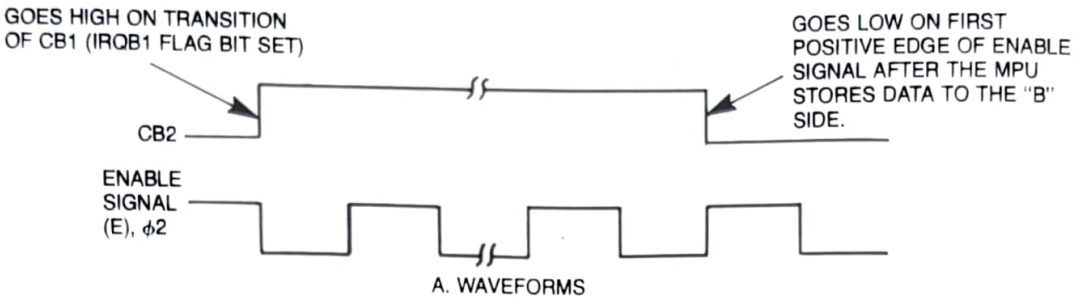
Transition of Input CB2	Status of Bit 5 in CRB (I/O Control)	Status of Bit 4 in CRB (Edge)	Status of Bit 3 in CRB (Mask)	IRQB2 (Interrupt Flag) Bit 6 of CRB	Status of IRQB Line (MPU Interrupt Request)
	0	0	0	1	MASKED (Remains High)
	0	0	1	1	GOES LOW (Processor Interrupted)
	0	1	0	1	MASKED (Remains High)
	0	1	1	1	GOES LOW (Processor Interrupted)

(All other combinations of CB1 transition and status of bit 3 and bit 4 will be ignored.)

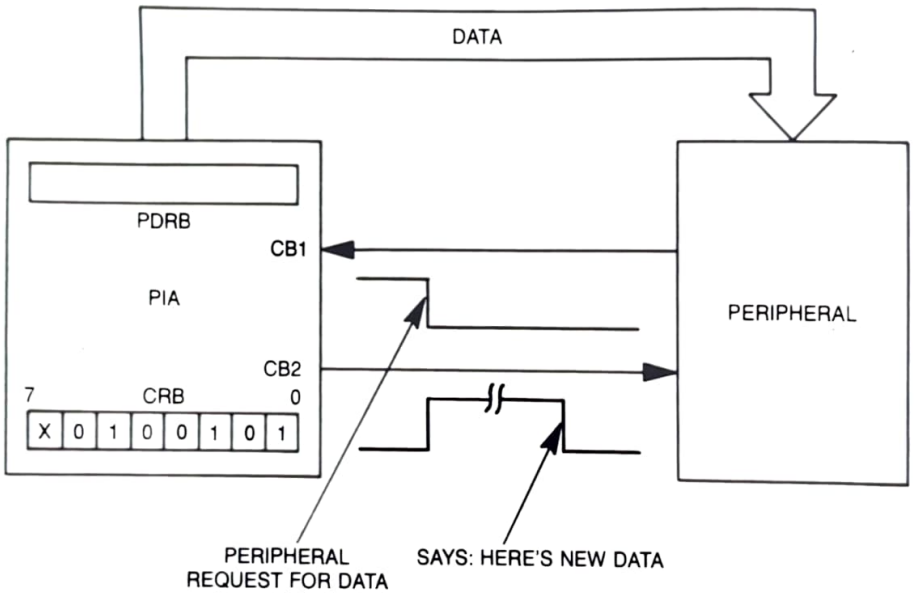
1.3.4.3 CB2 Control (Bits 3, 4, and 5 of CRB) as an Interrupt Input. Bits 3, 4, and 5 of the control register determine the function of this line.

The programming of bits 3, 4, and 5 in Control Register B has the same use as bits 3, 4, and 5 in Control Register A. Control Register B programs CB1 and CB2, while Control Register A programs CA1 and CA2.

1.3.4.4 CB2 Control (Bits 3, 4, and 5 of CRB) as an Interrupt Output. When used in the Handshake Mode or Pulse Mode, refer to Figures 2-11 and 2-12 for illustrations showing the operation of CB2 used as an output. However, if the Bit 3 Following Mode is used, the output level of CB2 will follow the logic level of bit 3 (bits 4 and 5 must be logic 1 levels).



A. WAVEFORMS



B. BLOCK DIAGRAM

5	4	3	= Bits
1	0	0	= Logic Levels

C. TRUTH TABLE

FIGURE 2-11. Handshake Mode Diagram (B Side)

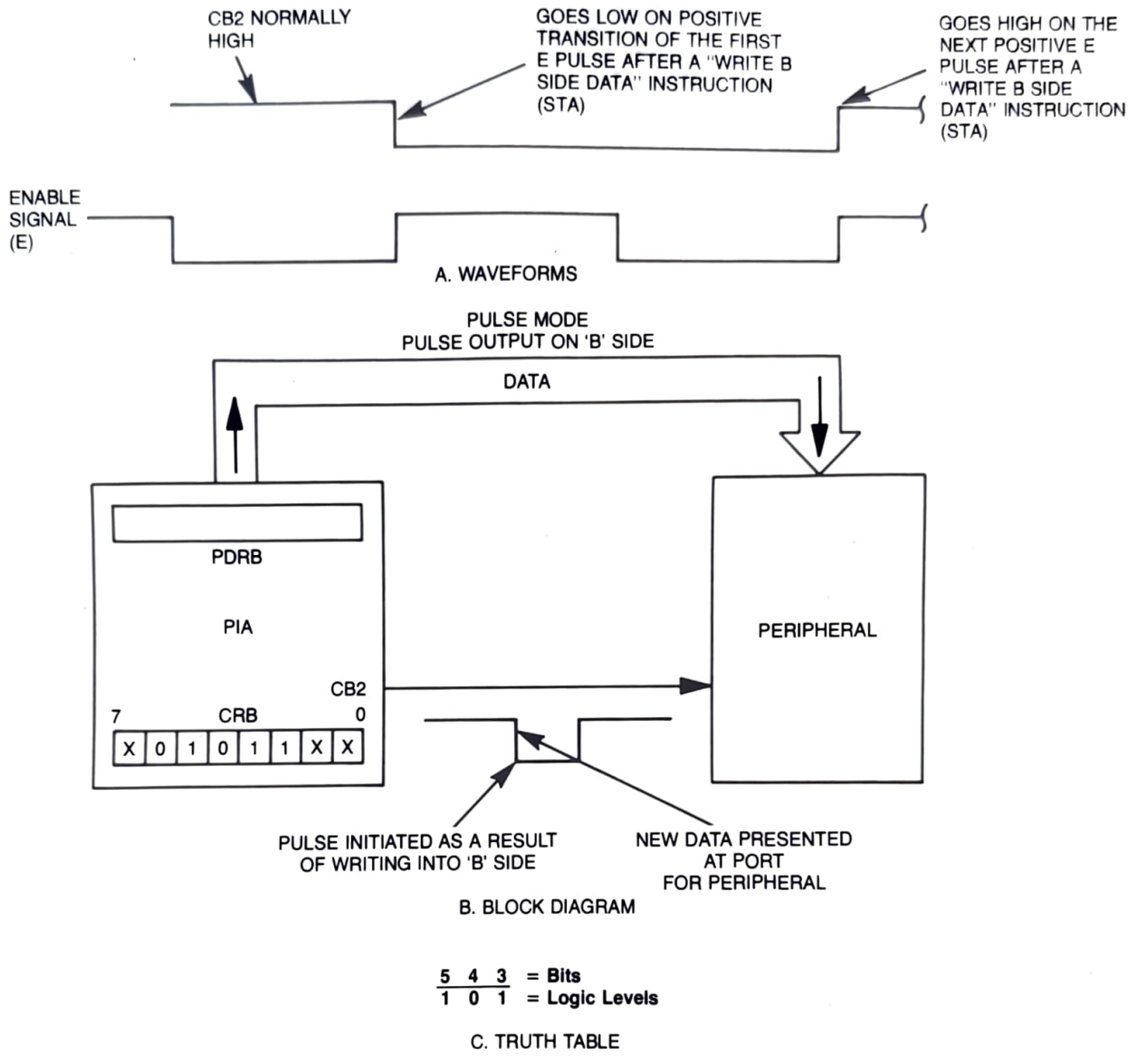


FIGURE 2-12. Pulse Mode Diagram (B Side)

1.3.5 ADDRESSING THE PIA

Before addressing the PIA, the Data Direction (DDR) must first be loaded with the bit pattern that defines how each line is to function (as an input or an output). A logic 1 in the Data Direction Register defines the corresponding line as an output, while a logic 0 defines the corresponding line as an input. Since the DDR and the Peripheral Data Lines have the same address, Control Register bit 2 determines which register is being addressed. If bit 2 in the Control Register is a logic 0, then the DDR is addressed. If bit 2 in the Control Register is a logic 1, the Peripheral Data Register is addressed. Therefore, it is essential that the DDR be loaded first before setting bit 2 of the Control Register.

1.4 DESCRIPTION OF MEMORY

Two types of memory elements are used in the Educator II: a static Random Access Memory (RAM) and a Read Only Memory (ROM). The following paragraphs describe each of these devices.

1.4.1 RANDOM ACCESS MEMORY (RAM)

The HEP C4811 is a static 128-byte Random Access Memory (RAM) enclosed in a 24-pin package. It has 8 data bus lines, 7 address bus lines, 6 chip select lines, 2 power inputs (+5Vdc and ground), and a Read/Write line. A functional block diagram of this device is provided in Figure 2-13.

In order to access this RAM, the CS0 and CS3 inputs must be held at a logic 1 level while the $\overline{CS1}$, $\overline{CS2}$, $\overline{CS4}$, and CS5 inputs must be held at a logic 0 level. To read data from the RAM, the R/W input must be held high. When low, data may be written into the RAM.

1.4.2 READ ONLY MEMORY (ROM)

The HEP C4814A and B are 512 x 4-bit Read Only Memories used to store the Educator II Resident Firmware program. This program is used to control the operation of the microcomputer system itself, including the Read/Write operation for the cassette recorder/player. The HEP C4814A contains the higher 4 bits of the program, while the HEP C4814B contains the lower 4 bits. A program listing for the Educator II Resident Firmware is provided in Appendix B.

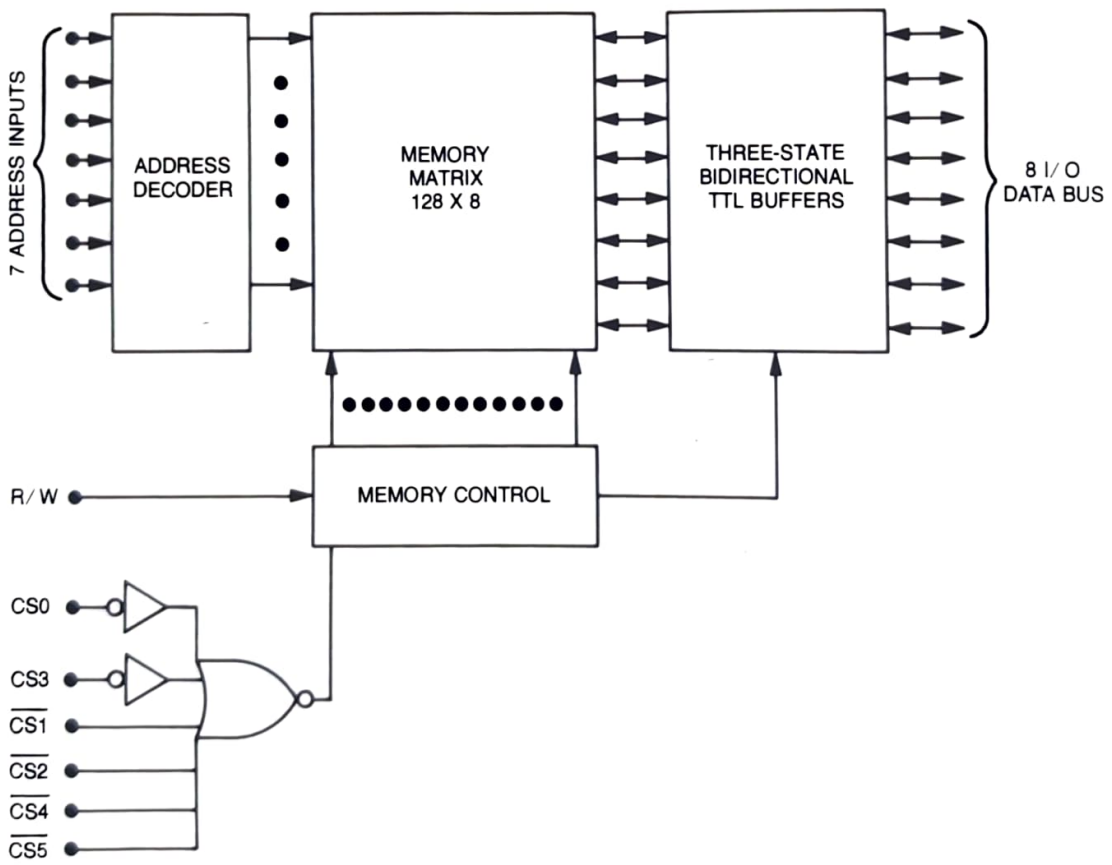


FIGURE 2-13. HEP C4811 RAM Functional Block Diagram

SECTION 2
SOFTWARE

2.1 INTRODUCTION

The term "Software" refers to the instructions used to program a computer system to perform some function or operation. Now that you know the basics of how the "Hardware" operates, you'll want to know how to program the Educator II to perform your own functions. The following paragraphs provide the necessary programming information. But, first, let's look at the numbering system upon which all of the microcomputer system's instructions are based.

2.2 NUMBER SYSTEMS

As you'll recall from previous paragraphs, the Educator II's Microprocessing Unit (MPU) is an eight-bit system. It has 8 data lines, 16 address lines, and functions with both 8 and 16-bit registers. Therefore, it is convenient to use the hexadecimal number system (base 16) when discussing the programming of your microcomputer system. However, before concentrating on the hexadecimal number system, let's look at several other number systems used in both everyday life and in digital computers.

2.2.1 DECIMAL (BASE 10)

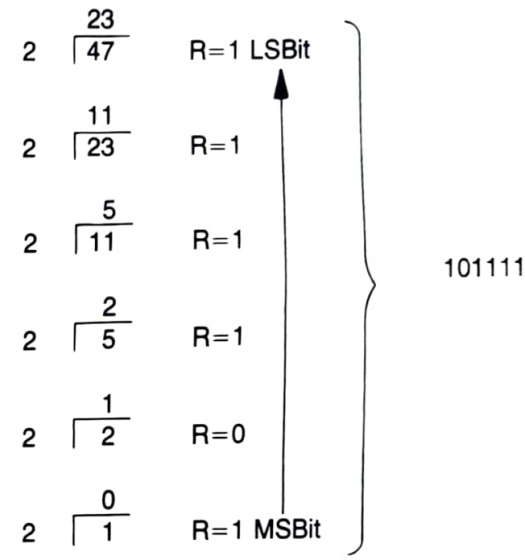
The most familiar number system (and the one you use in everyday life) is the decimal or base 10 number system (such as 0, 1, 2, 3, 4, 5, 6, 7, 8, and 9). What does a typical decimal (base 10) number represent? Take, for example, 2743. The number 2743 (expressed as 2743_{10} in the base 10 number system) really represents:

$$3x10^0 + 4x10^1 + 7x10^2 + 2x10^3$$

As you can see, the least significant digit (LSD), is a 3, and the most significant digit is a 2.

2.2.2 BINARY (BASE 2)

In digital computers like your Educator II, numbers are presented in binary or base 2 (such as 0 and 1). In order for you to "converse" with your microcomputer system, you must convert your normal decimal numbers into binary and vice versa. One method of converting decimal (base 10) numbers to binary (base 2) numbers is known as: repeated division by 2. For example, using the decimal number 47 (expressed as 47_{10} in the base number system):



Converting 101111_2 back to a decimal (base 10) number, we have:

$$\begin{aligned} 101111_2 &= 1x2^0 + 1x2^1 + 1x2^2 + 1x2^3 + 0x2^4 + 1x2^5 \\ &= 1x1 + 1x2 + 1x4 + 1x8 + 0x16 + 1x32 \\ &= 1 + 2 + 4 + 8 + 0 + 32 \\ &= 47_{10} \end{aligned}$$

In general, when converting from a number in any base number system to a number in base 10, proceed as follows:

$$(A_0 B^0 + A_1 B^1 + A_2 B^2 + A_3 B^3 + A_4 B^4 - - - + A_n B_n)$$

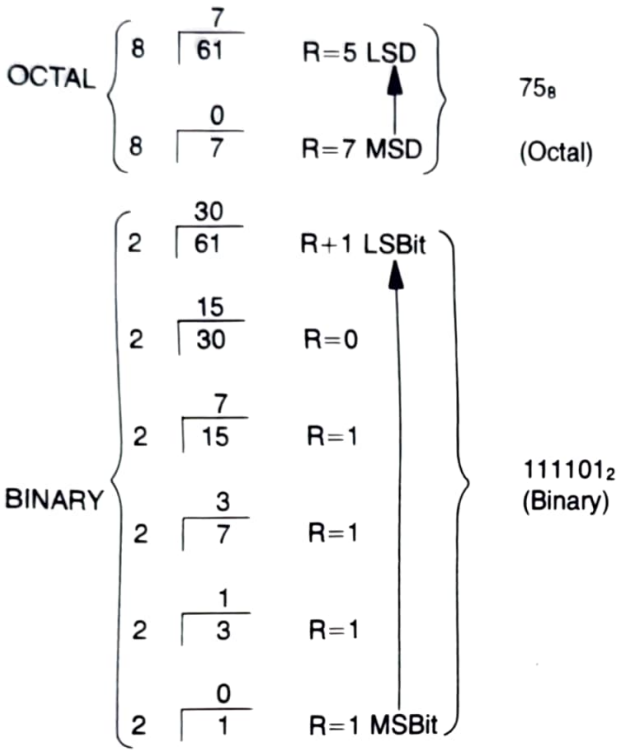
Where: A is the particular digit in the original number corresponding to its position to the left of the decimal point, and
B is the base of the number system.

In the example just completed (101111):

$A_0 = 1$, $A_1 = 1$, $A_2 = 1$, $A_3 = 1$, $A_4 = 0$, and $A_5 = 1$ and $B = 2$ (base 2).

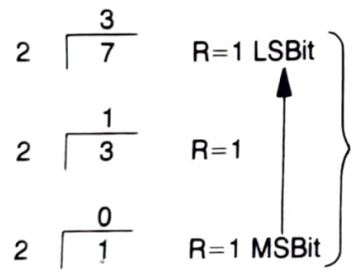
2.2.3 OCTAL (BASE 8)

Another Number System used with digital computers is octal (or base 8), since octal is a more convenient way of representing binary 2. To illustrate this convenience, the conversion of the number 61 in base 10 to a number in both base 8 and base 2 (using the method of repeated division) is shown below:

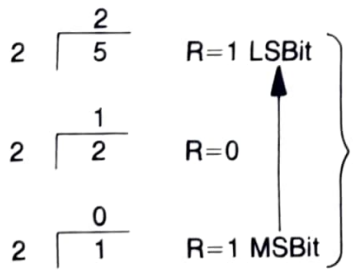


As a proof that $75_8 = 111101_2$, convert each digit of 75 base 8 to base 2 by continuous division.

Convert 7_8 to base 2



Convert 5_8 to base 2



This demonstrates that an octal number (Base 8) can be used to easily represent a string of binary bits (Base 2). Therefore, $75_8 = 111101_2$, which is the same pattern of 1's and 0's as derived by converting from base 10 to base 2.

2.2.4 HEXADECIMAL (BASE 16)

As previously mentioned, the Educator II's MPU utilizes both 8 and 16-bit registers. But, when trying to use the octal number system, there is a slight problem, since each digit in octal represents 3 binary bits (8 and 16 bits cannot be grouped evenly in groups of 3). This is resolved with the hexadecimal number system (commonly referred to as hex). Hexadecimal is a base 16 number system and can be handled in exactly the same manner as base 8 or base 2. In hexadecimal, four bits (in binary) represent one hexadecimal number. Thus, an eight-bit register can be represented by a 2-digit hex number. To illustrate this, assume that the binary number 01100111 exists in an eight-bit register. If this bit pattern is divided into two 4-bit groups of 0110 and 0111, then the hex representation would be 67_{16}. The following example is offered as a proof:

$$01100111_2 = \underline{1x2^0} + \underline{1x2^1} + \underline{1x2^2} + \underline{0x2^3} + \underline{0x2^4} + \underline{1x2^5} + \underline{1x2^6} + \underline{0x2^7}$$

MSB LSB $= 1 + 2 + 4 + 0 + 0 + 32 + 64 + 0$

$= 103_{10}$

and

LS Digit (LS half-byte)

$67_{16} = 7x16^0 + 6x16^1$

MS Digit (MS half-byte)

$= 7x1 + 6x16$

$= 7 + 96$

$= 103_{10}$

Therefore,

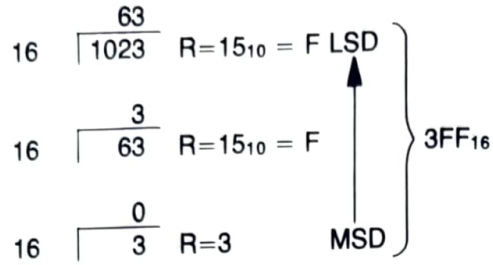
$$67_{16} = 01100111_2 = 103_{10}$$

From this simple example, one might wonder how hexadecimal digits (base 16) are represented for numbers above 9. The following table shows the solution to this dilemma.

TABLE 2-10. Hexadecimal Numbers

Base 10 (Decimal)	Base 16 (Hexadecimal)
0	0
1	1
2	2
3	3
4	4
5	5
6	6
7	7
8	8
9	9
10	A
11	B
12	C
13	D
14	E
15	F

To convert any decimal (base 10) number to hexadecimal (base 16), the repeated division method is once again used. As an example, convert 1023_{10} to hexadecimal.



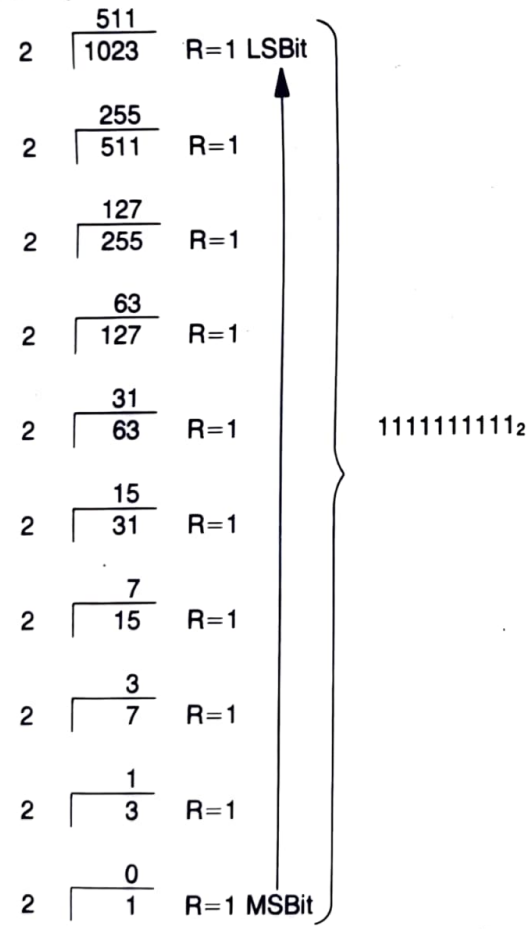
Therefore: $1023_{10} = 3FF_{16}$

As a check, convert $3FF_{16}$ back to a base 10 number.

$$\begin{aligned} 3FF_{16} &= 15x16^0 + 15x16^1 + 3x16^2 \\ &= 15x1 + 15x16 + 3x256 \\ &= 15 + 240 + 768 \\ &= 1023_{10} \end{aligned}$$

To further elaborate on the relationship between hexadecimal and binary, convert 1023_{10} to binary and then back to hexadecimal. First,

1023_{10} to binary



Now, by arranging this number into three groups of four bits each and then converting each group into its hexadecimal counterpart, the result is 1023_{10}. Represented in hexadecimal, this is $1111111111_2 = 0011\ 1111\ 1111_2 = 3FF_{16}$. Where $0011_2 = 3_{16}$ and $1111_2 = F_{16}$.

In summary, remember that each hexadecimal (Base 16) digit is a representation of 4 binary bits. It is easy to convert from hex to binary and binary to hex. For convenience, a limited conversion chart is presented in Table 2-11.

TABLE 2-11. Number Systems Conversion Chart

Decimal	Octal	Hexadecimal	Binary
0	0	0	0000 0000
1	1	1	0000 0001
2	2	2	0000 0010
3	3	3	0000 0011
4	4	4	0000 0100
5	5	5	0000 0101
6	6	6	0000 0110
7	7	7	0000 0111
8	10	8	0000 1000
9	11	9	0000 1001
10	12	A	0000 1010
11	13	B	0000 1011
12	14	C	0000 1100
13	15	D	0000 1101
14	16	E	0000 1110
15	17	F	0000 1111
16	20	10	0001 0000
17	21	11	0001 0001
18	22	12	0001 0010
19	23	13	0001 0011
20	24	14	0001 0100
21	25	15	0001 0101
22	26	16	0001 0110
23	27	17	0001 0111
24	30	18	0001 1000
25	31	19	0001 1001
26	32	1A	0001 1010
27	33	1B	0001 1011
28	34	1C	0001 1100
29	35	1D	0001 1101
30	36	1E	0001 1110
31	37	1F	0001 1111
32	40	20	0010 0000
33	41	21	0010 0001
34	42	22	0010 0010
35	43	23	0010 0011
36	44	24	0010 0100
37	45	25	0010 0101
38	46	26	0010 0110
39	47	27	0010 0111
40	50	28	0010 1000

2.2.5 TWO'S COMPLEMENT NUMBERS

The Educator II does not perform subtraction directly, so the method of 2's complement addition is used to accomplish the subtraction. The 2's complement of any binary number is its *additive inverse*. That is, a binary number plus its 2's complement always equals zero.

$$\begin{array}{r} 11011011 \\ +00100101 \\ \hline 00000000 \end{array}$$ 2's Complements of 11011011

How is the 2's complement of a binary number computed? There are several methods. One way of calculating the 2's complement is to take the number to be converted, invert all the digits, then add one.

For example: Find the 2's complement of

11011011.		(91_{10})
First invert	01011011	
Equals	10100100	
Add 1	+ 1	
	10100101	
Therefore	10100101	is the 2's complement of 0101101.

The following examples of subtraction use the 2's complement addition method.

1) Given 61 — 12 = ? (Base 10)

	Binary Notation	Hex Notation
61 =	00111101	−3D
−12 =	−00001100	−0C

To do the subtraction, first convert 00001100 (12_{10}) to a 2's complement number.

	Binary Notation		Hex Notation
	00001100		0C
Invert	11110011		F3
Add	+00000001		+01
	11110100	2's complement	

So the subtraction becomes 2's complement addition

	00111101		3D
	+11110100		+4F
Ans	00110001	Ans	31

As a check

61_{10}
-12_{10}
$=49_{10} = 00110001_2 = 31_{16}$

2) Given 61 — 2 = ? (Base 10)

	Binary Notation	Hex Notation
61 =	00111101	3D
− 2 =	00000000	−02

Doing the 2's complements addition.

	Binary Notation		Hex Notation
	00111101		3D
	+11111110		FE
Ans	00111011	Ans	3B

2.3 MICROCOMPUTER PROGRAMMING

Writing a microcomputer program is similar to giving very specific instructions to an intellectually underdeveloped being. However, the microcomputer system does have the advantage of a very good memory that enables it to repeat a given task time after time. Rather than communicating verbally with our system, some other means of loading our tasks into its memory must be found. In the Educator II, the communication media is a series of front panel switches and LED indicators. The switches allow you to instruct the microcomputer system, while the LED indicators are the means used for the microcomputer system to answer you.

Microprocessor Units (MPU's), like other logic devices, understand only very simple commands written in a very simple format. For the MPU used in the Educator II, this format consists of evaluating the positions of 8 front panel switches and acting in a special way for each combination. Five different commands are shown below along with their equivalent switch positions, hexadecimal (hex) operation code, mnemonic, and addressing mode.

It's now apparent why 8 switches in a row are provided on the front panel of the Educator II. These switches supply commands (or instructions) to the processor as well as the numbers on which these instructions operate.

Although every instruction could be entered and executed immediately, you would quickly become exhausted working your way through a problem. Instead, a program or set of instructions is usually entered into the memory from the front panel switches and executed all together. It is when operating in this mode that the microcomputer speed is readily apparent. For this reason, three more switches (in addition to the 8 used to enter the instructions) are included on the front panel. These center off, momentary contact switches allow the user to store many instructions into the Educator II's memory (limited only by memory size), examine them for accuracy, and execute a particular program. The MPU in the Educator II is capable of handling memories with up to 65,536 addressable locations. However, as supplied in this kit, the Educator II has internal program memory for only 128 instructions, of which 14 are usable by you (the remaining 14 are used by the program stored in the ROM). If you've added the optional memory element, you have an additional 128 address locations for instructions.

MPU COMMAND (INSTRUCTION)	SWITCH POSITIONS	HEX OP CODE	MNEMONIC	ADDRESSING MODE
Add two numbers	0001 1011	1B	ABA	Inherent
Subtract two numbers	0001 0000	10	SBA	Inherent
Increment a number	0100 1100	4C	INC A	Inherent
Decrement a number	0100 1010	4A	DEC A	Inherent
Perform a logic OR on two other switch patterns	1000 1010	8A	ORA A	Immediate

In the Educator II, the address locations available for your program start at decimal 128 and go to decimal 242. (In hex notation, this is 80 to F1.) The optional RAM extends this range from 0 to 242 (decimal). (This is 0 to F1 in hex notation.)

At this point, you may wonder how numbers get into the processor to be executed. *All* information (including instructions and data) enters and leaves the microcomputer via memory. Therefore, all data must be either contained in the program (which is stored in memory) or exist in the bit pattern established by the front panel switches. This correctly implies that the eight front panel switches may be addressed like a memory location. These switches are located at 4000_{16}, while the LED indicators are located at 4002_{16}. When programs are written, there are usually many numbers interspersed with the instructions to provide the data for each instruction. Writing programs usually begins with stating your objectives in a logical manner and then converting these statements into a suitable machine language.

As previously mentioned in the Hardware section (Part 2, Section 1), all programming operations consist of using 72 executable instructions in one of six different addressing modes. These addressing modes and executable instructions are described in the following paragraphs. For additional programming information, refer to Motorola's M6800 PROGRAMMING REFERENCE MANUAL.

2.3.1 ADDRESSING MODES

Each of the 72 executable instructions can be used in at least one of the following six addressing modes:

1. Inherent/Accumulator Addressing
2. Immediate Addressing
3. Direct Addressing
4. Extended Addressing
5. Indexed Addressing
6. Relative Addressing

2.3.1.1 Inherent/Accumulator Addressing. Twenty-five of the instructions (mnemonic operators such as LDA= Load Accumulator) are used to specify one or more registers in the MPU which contain operating instructions or in which results are saved (Inherent Addressing). Thirteen of the instructions can be used to address either Accumulator A or Accumulator B in the MPU (Accumulator Addressing). In either case, only one byte of data is required for instructions used in this mode.

2.3.1.2 Immediate Addressing. Instructions used in the Immediate Addressing mode contain the actual value to be used in the execution of this instruction. This addressing mode requires either one or two bytes in addition to the instruction byte itself. Two bytes are used when the value is less than or equal to 255 (decimal). An instruction of this type is shown below.

2.3.1.3 Direct and Extended Addressing. For Direct Addressing, the instruction consists of two bytes of machine code. The second byte will contain the address in binary form. For Extended Addressing, the instruction consists of three bytes of machine code. The second byte will contain the highest 8 bits of the address, while the third byte will contain the lowest 8 bits. For both Direct and Extended Addressing, the address contained in the second byte (and third byte) is the absolute numerical address.

2.3.1.4 Indexed Addressing. In this addressing mode, the numerical address is variable and dependent upon the contents of the index register in the MPU. The value of the data stored in the second byte of the instruction is added to the contents of the index register to determine the numerical address.

2.3.1.5 Relative Addressing. The Relative Addressing mode requires two bytes of data and can only be used by conditional branch instructions such as BEQ (Branch if Equal to Zero), the unconditional branch instruction BRA (Branch Always), or the Branch to Subroutine BSR. None of these instructions can use any other addressing mode.

MNEMONIC INSTRUCTION	BYTE NUMBER	HEX VALUE	BINARY VALUE	OPERATION
LDAA 255	1 2	86 FF	1000 0110 1111 1111	Load Accumulator A with the value 255

Three bytes are used when the value is between 255 and 65535 (decimal). An instruction of this type is shown below.

MNEMONIC INSTRUCTION	BYTE NUMBER	HEX VALUE	BINARY VALUE	OPERATION
LDS 2445	1 2 3	8E 09 8D	1000 1110 0000 1001 1000 1101	Load Stack Pointer register with address stored at memory location

For the Relative Addressing mode to be valid, there is a rule that limits the distance in the machine language program from the branch instruction to the destination of the branch. This rule is that the address of the destination must be within the range specified by the following formula:

$$(PC + 2) - 128 < D < (PC + 2) + 127$$

where: PC = address of the current location
CD = address of the destination of the branch instruction.

Thus: The address of the destination must be less than 126 locations backward (−) from the present location, or less than 129 locations forward (+).

2.3.2 EXECUTABLE INSTRUCTIONS

The MPU will respond to 72 different executable instructions (previously listed in alphabetic order in Table 2-3). Each of these instructions is explained in the following steps (in these steps, ACCA or ACCB means Accumulator A or Accumulator B, and CCR means Condition Code Register).

1. **ABA** (Add Accumulator B to Accumulator A). Adds the contents of ACCB to ACCA and places the results in ACCA. This instruction can be used only in the Inherent Addressing mode and consists of only one byte of machine code. The CCR is affected as follows:
   - H: Set if there was a carry from bit 3; cleared otherwise.
   - I: Not affected.
   - N: Set if most significant bit of the result is set; cleared otherwise.
   - Z: Set if all bits of the result are cleared; cleared otherwise.
   - V: Set if there was two's complement overflow as a result of the operation; cleared otherwise.
   - C: Set if there was a carry from the most significant bit of the result; cleared otherwise.
2. **ADC** (Add With Carry). Adds the contents of the C bit in the CCR to the sum of the contents of ACCA (or ACCB) and the contents of the specified memory location, and places the result in ACCA (or ACCB). This instruction can be used in any of the following addressing modes: Immediate, Direct, or Indexed (two byte instructions) or Extended (three byte instruction). The CCR is affected as follows:
   - H: Set if there was a carry from bit 3; cleared otherwise.
   - I: Not affected.
   - N: Set if most significant bit of the result is set; cleared otherwise.
   - Z: Set if all bits of the result are cleared; cleared otherwise.
   - V: Set if there was two's complement overflow as a result of the operation; cleared otherwise.
   - C: Set if there was a carry from the most significant bit of the result; cleared otherwise.
3. **ADD** (Add Without Carry). Adds the contents of ACCA (or ACCB) and the contents of the specified memory location, and places the result in ACCA (or ACCB). This instruction can be used in the same addressing modes as previously shown for ADC (step 2). The CCR is affected as follows:
   - H: Set if there was a carry from bit 3; cleared otherwise.
   - I: Not affected.
   - N: Set if most significant bit of the result is set; cleared otherwise.
   - Z: Set if all bits of the result are cleared; cleared otherwise.
   - V: Set if there was two's complement overflow as a result of the operation; cleared otherwise.
   - C: Set if there was a carry from the most significant bit of the result; cleared otherwise.
4. **AND** (Logical AND). Performs logical "AND" between the contents of ACCA (or ACCB) and the contents of the specified memory location, and places the result in ACCA (or ACCB). (Each bit of the selected accumulator *after* execution will be the logical "AND" of the corresponding bits of the data stored in the memory location and of the selected accumulator *before* execution.) This instruction can be used in the same addressing modes as previously shown for ADC (step 2). The CCR is affected as follows:
   - H: Not affected.
   - I: Not affected.
   - N: Set if most significant bit of the result is set; cleared otherwise.
   - Z: Set if all bits of the result are cleared; cleared otherwise.
   - V: Cleared.
   - C: Not affected.
5. **ASL** (Arithmetic Shift Left). Shifts all bits of ACCA (or ACCB) or the contents of one memory location one place to the left. The bit 0 position will be loaded with a zero. The C bit is loaded from the most significant bit (bit 7) of the selected accumulator or contents of the memory location. It can be used in either the Inherent addressing mode (one byte instruction), Indexed mode (two byte instruction), or Extended mode (three byte instruction). The CCR is affected as follows:
   - H: Not affected.
   - I: Not affected.
   - N: Set if most significant bit of the result is set; cleared otherwise.
   - Z: Set if all bits of the result are cleared; cleared otherwise.
   - V: Set if, after the completion of the shift operation, EITHER (N is set and C is cleared) OR (N is cleared and C is set); cleared otherwise.
   - C: Set if, before the operation, the most significant bit of the ACCX or M was set; cleared otherwise.

6. **ASR** (Arithmetic Shift Right). Shifts all bits of ACCA (or ACCB) or the contents of one memory location one place to the right. Bit 7 is held constant. Bit 0 is loaded into the C bit. Both the addressing modes and the CCR are the same as previously shown for ASL (step 5).

7. **BCC** (Branch if Carry Clear). Tests the logic state of the C bit and causes a branch if the C bit is cleared (logic 0). Refer to the BRA instruction in step 20 for further details on the execution of the branch.) This instruction can only be used in the Relative addressing mode (two byte instruction). The logic states of the six CCR bits are not affected.

8. **BCS** (Branch if Carry Set). Tests the logic state of the C bit and causes a branch if the C bit is set (logic 1). (Refer to the BRA instruction in step 20 for further details on the execution of the branch.) This instruction can only be used in the Relative addressing mode (two byte instruction). The logic states of the six CCR bits are not affected.

9. **BEQ** (Branch if Equal). Tests the logic state of the 2 bit and causes a branch if the 2 bit is set (logic 1). (Refer to BRA instruction for further details on the execution of this branch.) This instruction can only be used in the Relative addressing mode (two byte instruction). The logic states of the six CCR bits are not affected.

10. **BGE** (Branch if Greater Than or Equal to Zero). Causes a branch if both the N bit and V bit of the CCR are set (logic 1) *or* cleared (logic 0). If the BGE instruction is executed immediately following the execution of a CBA, CMP, SBA, or SUB instruction, the branch will occur if and only if the two's complement number (inverse of the binary number + 1) represented by the minuend (such as stored in ACCA) was greater than or equal to the two's complement number represented by the subtrahend (such as stored in a memory location). (Refer to the BRA instruction in step 20 for further details on the exection of this branch.) This instruction can only be used in the Relative addressing mode (two byte instruction). The logic states of the six CCR bits are not affected.

11. **BGT** (Branch if Greater than Zero). Causes a branch if both the Z bit of the CCR s cleared (logic 0) *and* the N bit and V bit are set (logic 1) *or* cleared (logic 0). The remaining information for this instruction is the same as previously discussed for the BGE instruction (step 10).

12. **BHI** (Branch if Higher). Causes a branch if both the C bit in the CCR *and* the Z bit are cleared (logic 0). The remaining information for this instruction is the same as previously discussed for the BGE instruction (step 10).

13. **BIT** (Bit Test). Perform the logical "AND" comparison of the contents of ACCA (or ACCB) and the contents of one memory location and modifies the N bit, Z bit, and V bit of the CCR accordingly. Neither the contents of the selected accumulator or memory location are affected. (Each bit of the result would be the logical "AND" of the corresponding bits of the memory location and the selected accumulator.) This instruction can be used in the Immediate, Direct, or Indexed addressing mode (two byte instructions) or in the Extended mode (three byte instruction). The CCR is affected as follows:

    H: Not affected.
    I: Not affected.
    N: Set if most significant bit of the result of the "AND" would be set; cleared otherwise.
    Z: Set if all bits of the result of the "AND" would be cleared; cleared otherwise.
    V: Cleared.
    C: Not affected.

14. **BLE** (Branch if Less Than or Equal to Zero). Causes a branch if the Z bit in the CCR is set (logic 1) *or* the N bit is set and the V bit is cleared (logic 0) *or* the N bit is cleared and the V bit is set. The remaining information for this instruction is the same as previously discussed for the BGE instruction (step 10).

15. **BLS** (Branch if Lower or Same). Causes a branch if the C bit in the CCR *or* the Z bit is set (logic 1). The remaining information for this instruction is the same as previously discussed for the BGE instruction (step 10).

16. **BLT** (Branch if Less Than Zero). Causes a branch if the N bit of the CCR is set (logic 1) and the V bit is cleared (logic 0) *or* the N bit is cleared and the V bit is set. The remaining information for this instruction is the same as previously discussed for the BGE instruction (step 10).

17. **BMI** (Branch if Minus). Tests the state of the N bit in the CCR and causes a branch if N is set (logic 1). (Refer to the BRA instruction in step 20 for further details on the execution of this branch.) This instrution can only be used in the Relative addressing mode (two byte instruction). The logic states of the six CCR bits are not affected.

18. **BNE** (Branch if Not Equal). Tests the state of the Z bit in the CCR and causes a branch if the Z bit is cleared (logic 0). (Refer to the BRA instruction in step 20 for further details on the execution of this branch.) This instruction can only be used in the Relative addressing mode (two byte instruction). The logic states of the six CCR bits are not affected.

19. **BPL** (Branch if Plus). Tests the state of the N bit is cleared (logic 0). (Refer to the BRA instruction in step 20 for further details on the execution of this branch.) This instruction can only be used in the Relative addressing mode (two byte instruction). The logic states of the six CCR bits are not affected.

20. **BRA** (Branch Always). Unconditional branch to the address given by:

    PC = PC + 2 + R

    where: PC = address in Program Counter
    R = Relative address stored as a two's complement number (inverse of a binary number + 1) in the second byte of machine code following the branch instruction.

    This instruction can only be used in the Relative addressing mode (two byte instruction). The logic states of the six CCR bits are not affected.

    NOTE

    The program specifies the destination of any branch instruction by its absolute address. You must calculate the Relative address value by subtracting (or adding) the destination address from (or to) the current value of the Program Counter (PC).

21. **BSR** (Branch to Subroutine). The Program Counter (PC) is incremented by 2, and the last significant byte of the contents of the PC is pushed into the stack. The stack pointer is then decremented by 1, and the most significant byte of the contents of the PC is then pushed into the stack. The stack pointer is again decremented by 1, and a branch occurs to the location specified by the program. (Refer to the BRA instruction in step 20 for further details on the execution of this branch.) This instruction can only be used in the Relative addressing mode (two byte instruction). The logic states of the six CCR bits are not affected.

22. **BVC** (Branch if Overflow Clear). Tests the state of the V bit in the CCR and causes a branch if the V bit is cleared (logic 0). (Refer to the BRA instruction in step 20 for further details on the execution of this branch.) This instruction can only be used in the Relative addressing mode (two byte instruction). The logic states of the six CCR bits are not affected.

23. **BVS** (Branch if Overflow Set). Tests the state of the V bit in the CCR, and causes a branch if the V bit is set (logic 1). (Refer to the BRA instruction in step 20 for further details on the execution of this branch.) This instruction can only be used in the Relative addressing mode (two byte instruction). The logic states of the six CCR bits are not affected.

24. **CBA** (Compare Accumulators). Compares the contents of ACCA and the contents of ACCB and sets the condition codes, which may be used for arithmetic and logic conditional branches. This instruction can only be used in the Inherent addressing mode (one byte instruction). The CCR is affected as follows:

    H: Not affected.
    I: Not affected.
    N: Set if the most significant bit of the result of the subtraction would be set; cleared otherwise.
    Z: Set if all bits of the result of the subtraction would be cleared; cleared otherwise.
    V: Set if the subtraction would cause two's complement overflow; cleared otherwise.
    C: Set if the subtraction would require a borrow into the most significant bit of the result; clear otherwise.

25. **CLC** (Clear Carry). Clears the C bit in the CCR. This instruction can only be used in the Inherent addressing mode (one byte instruction). Only the C bit of the CCR is affected by this instruction.

26. **CLI** (Clear Interrupt Mask). Clears the I bit in the CCR, enabling the MPU to service an interrupt from a peripheral device if signalled by a logic 1 on the "Interrupt Request" control input. This instruction can only be used in the Inherent addressing mode (one byte instruction). Only the I bit of the CCR is affected by this instruction.

27. **CLR** (Clear). The contents of ACCA (or ACCB) or a memory location are replaced with zeros. This instruction may be used in the Inherent addressing mode (one byte instruction), Indexed addressing mode (two byte instruction), or Extended addressing mode (three byte instruction). The N bit in the CCR is cleared (logic 0) and the Z bit is sent (logic 1).

28. **CLV** (Clear Two's Complement Overflow Bit). Clears the two's complement overflow bit (V bit) in the CCR. This instruction can only be used in the Inherent addressing mode (one byte instruction). Only the V bit in the CCR is affected by this instruction.

29. **CMP** (Compare). Compares the contents of ACCA (or ACCB) and the contents of the specified memory location to determine the logic states of the N, Z, V, and C bits in the CCR. These bits may be subsequently used for controlling conditional branching. The contents of the selected accumulator and memory location are not affected. This instruction may be used in the Immediate, Direct, or Indexed addressing mode (two byte instructions) or the Extended addressing mode (three byte instruction). The CCR is affected as follows:

    H: Not affected.
    I: Not affected.
    N: Set if the most significant bit of the result of the subtraction would be set; cleared otherwise.
    Z: Set if all bits of the result of the subtraction would be cleared; cleared otherwise.
    V: Set if the subtraction would cause two's complement overflow; cleared otherwise.
    C: Carry is set if the absolute value of the contents of memory is larger than the absolute value of the accumulator; reset otherwise.

30. **COM** (Complement). Replaces the contents of ACCA (or ACCB) or a single memory location with its one's

complement. (Each bit of the contents of the selected accumulator or memory location is replaced with the complement of that bit.) This instruction may be used in the Inherent addressing mode (one byte instruction), the Indirect addressing mode (one byte instruction), the Indirect addressing mode (two byte instruction), and the Extended addressing mode (three byte instruction). The CCR is affected as follows:

H: Not affected.
I: Not affected.
N: Set if most significant bit of the result is set; cleared otherwise.
Z: Set if all bits of the result are cleared; cleared otherwise.
V: Cleared.
C: Set.

31. **CPX** (Compare Index Register). The more significant byte of the contents of the index register is compared with the contents of the byte of memory at the address specified by the program. The less significant byte of the contents of the index register is compared with the contents of the next byte of memory, at one plus the address specified by the program. The Z bit in the CCR is set or reset accordingly to the results of these comparisons, and may be used subsequently for conditional branching. The N and V bits, although determined by this operation, are not intended for conditional branching. The C bit is not affected by this operation. This instruction may be used in either the Direct or Indirect addressing modes (two byte instructions) or in the Immediate or Extended addressing modes (three byte instructions). The CCR is affected as follows:

H: Not affected.
I: Not affected.
N: Set if the most significant bit of the result of the subtraction from the more significant byte of the index register would be set; cleared otherwise.
Z: Set if all bits of the result of both subtractions would be cleared; cleared otherwise.
V: Set if the subtraction from the more significant byte of the index register would cause two's complement overflow; cleared otherwise.
C: Not affected.

32. **DAA** (Decimal Adjust Accumulator A). If the contents of ACCA and the states of the carry/borrow bit (C) and the half carry bit (H) in the CCR are all the result of applying any ABA, ADD, or ADC operation to binary-coded-decimal results (with or without an initial carry), the DAA operation will adjust the contents of ACCA and the C bit to represent the correct binary-coded-decimal sum and the correct state of the carry. This instruction can only be used in the Inherent addressing mode (two byte instruction). The CCR is affected as follows:

H: Not affected.
I: Not affected.
N: Set if most significant bit of the result is set; cleared otherwise.
Z: Set if all bits of the result are cleared; cleared otherwise.
V: Not affected.
C: Set or reset according to the same rule as if the DAA and an immediately preceding ABA, ADD, or ADC were replaced by a hypothetical binary-coded-decimal addition.

33. **DEC** (Decrement). Subtract one from the contents of ACCA (or ACCB) or from the contents of a specified memory location. The N, Z, and V bits of the CCR are set or reset according to the results of this operation. The C bit is not affected. This instruction may be used in the Inherent addressing mode (one byte instruction), Indexed addressing mode (two byte instruction), or Extended addressing mode (three byte instruction). The CCR is affected as follows:

H: Not affected.
I: Not affected.
N: Set if most significant bit of the result is set; cleared otherwise.
Z: Set if all bits of the result are cleared; cleared otherwise.
V: Set if there was two's complement overflow as a result of the operation; cleared otherwise. Two's complement overflow occurs if and only if (ACCX) or (M) was 80 before the operation.
C: Not affected.

34. **DES** (Decrement Stack Pointer). Subtract one from Stack Pointer. This instruction may only be used in the Inherent addressing mode (one byte instruction). The condition of the CCR is not affected.

35. **DEX** (Decrement Index Register). Subtract one from the Index Register. This instruction can only be used in the Inherent instruction mode (one byte instruction). Only the Z bit in the CCR is affected by this instruction, and it is set (logic 1) if all bits of the result are cleared (logic 0). Otherwise, the Z bit is cleared.

36. **EOR** (Exclusive OR). Performs the logical "Exclusive OR" function between the contents of ACCA (or ACCB) and the contents of a specified memory location. The results are then placed in the selected accumulator (A or B). (Each bit of the accumulator, after execution of this instruction, will be the logical "Exclusive OR" of the corresponding bit in the memory location and the selected accumulator before execution.) This instruction may be used in either the Immediate, Direct, or Indexed addressing mode (two byte instruction) or the Extended addressing mode (three byte instruction). The CCR is affected as follows:

H: Not affected.
I: Not affected.
N: Set if most significant bit of the result is set; cleared otherwise.
Z: Set if all bits of the result are cleared; cleared otherwise.
V: Cleared.
C: Not affected.

37. **INC** (Increment). Add one to the contents of ACCA (or ACCB) or to the contents of the specified memory location. This instruction may be used in either the Inherent addressing mode (one byte instruction), the Indexed addressing mode (two byte instruction), or the Extended addressing mode (three byte instruction). The CCR is affected as follows:

H: Not affected.
I: Not affected.
N: Set if most significant bit of the result is set; cleared otherwise.
Z: Set if all bits of the result are cleared; cleared otherwise.
V: Set if there was two's complement overflow as a result of the operation; cleared otherwise. Two's complement overflow will occur if and only if (ACCX) or (M) was 7F before the operation.
C: Not affected.

38. **INS** (Increment Stack Pointer). Add one to the Stack Pointer. This instruction can only be used in the Inherent addressing mode (one byte instruction). The condition of the CCR is not affected.

39. **INX** (Increment Index Register). Add one to the Index Register. This instruction can only be used in the Inherent addressing mode (one byte instruction). Only the Z bit in the CCR is affected, and it is set (logic 1) if all 16 bits of the result are cleared (logic 0). Otherwise, the Z bit is cleared.

40. **JMP** (Jump). A jump occurs to the instruction sorted at the specified numerical address location. This numerical address is obtained according to the rules for Indexed addressing (two byte instruction) or Extended addressing (three byte instruction). The condition of the CCR is not affected.

41. **JSR** (Jump to Subroutine). The Program Counter is incremented two (Indexed addressing mode) or three (Extended addressing mode) bytes, depending on the addressing mode, and is then pushed onto the stack one byte at a time. The Stack Pointer now points to the next empty memory location in the stack. A jump then occurs to the instruction stored at the numerical address. The numerical address is obtained according to the rules for the Indexed (two byte instruction) or the Extended (three byte instruction) addressing mode. The condition of the CCR is not affected.

42. **LDA** (Load Accumulator). Loads the contents of memory into the accumulator. This instruction may be used in either the Immediate, Direct, or Indexed addressing mode (two byte instructions) or the Extended addressing mode (three byte instructions). The CCR is affected as follows:

H: Not affected.
I: Not affected.
N: Set if most significant bit of the result is set; cleared otherwise.
Z: Set if all bits of the result are cleared; cleared otherwise.
V: Cleared.
C: Not affected.

43. **LDS** (Load Stack Pointer). Loads the more significant byte of the Stack Pointer from the address location specified by the program and loads the less significant byte of the Stack Pointer from the next byte of memory, at one plus the address specified by the program. This instruction may be used in the Direct or Indexed addressing mode (two byte instructions) or in the Immediate or Extended addressing mode (three byte instructions). The CCR is affected as follows:

H: Not affected.
I: Not affected.
N: Set if most significant bit of the stack pointer is set by the operation; cleared otherwise.
Z: Set if all bits of the stack pointer are cleared by the operation; cleared otherwise.
V: Cleared.
C: Not affected.

44. **LDX** (Load Index Register). Loads the more significant byte of the Index Register with the contents of the memory at the specified address and then loads the less significant byte of the Index Register from the next byte of the memory, at one plus the address specified by the instruction. This instruction may be used in the Direct or Indexed addressing modes (two byte instructions) or the Immediate or Extended addressing modes (three byte instructions). The CCR is affected as follows:

H: Not affected.
I: Not affected.
N: Set if the most significant bit of the index register is set by the operation; cleared otherwise.
Z: Set if all bits of the index register are cleared by the operation; cleared otherwise.
V: Cleared.
C: Not affected.

45. **LSR** (Logical Shift Right). Shifts all bits of the specified accumulator (A or B) or the specified memory location one place to the right. Bit 7 is loaded with a zero, and the C bit (of the CCR) is loaded from the least significant bit of ACCA (or ACCB) or the memory location. This instruction may be used in the Inherent addressing mode (one byte instruction may be used in the Inherent addressing mode (one byte instruction), the Indexed addressing mode (two byte instruction), or the Extended addressing mode (three byte instruction).

The CCR is affected as follows:

H: Not affected.
I: Not affected.
N: Cleared.
Z: Set if all bits of the result are cleared; cleared otherwise.
V: Set if, after the completion of the shift operation, EITHER (N is set and C is cleared) OR (N is cleared and C is set); cleared otherwise.
C: Set if, before the operation, the least significant bit of the ACCX or M was set; cleared otherwise.

46. **NEG** (Negate). Replaces the contents of the specified accumulator (A or B) or the specified memory location with its two's complement. (Note than an 80_{16} is not changed.) This instruction may be used in the Inherent addressing mode (one byte instruction), the Indexed addressing mode (two byte instruction), or the Extended addressing mode (three byte instruction). The CCR is affected as follows:

    H: Not affected.
    I: Not affected.
    N: Set if most significant bit of the result is set; cleared otherwise.
    Z: Set if all bits of the result are cleared; cleared otherwise.
    V: Set if there would be two's complement overflow as a result of the implied subtraction from zero; this will occur if and only if the contents of ACCX or M is 80.
    C: Set if there would be a borrow in the implied subtraction from zero; the C bit will be set in all cases except when the contents of ACCX or M is 00.

47. **NOP** (No Operation). This is a single-word instruction (Inherent addressing mode) which causes only the Program Counter to be incremented by one. No other registers are affected, including the CCR.

48. **ORA** (Inclusive OR). Performs a logical "OR" operation between the contents of the specified accumulator (A or B) and the contents of the specified memory location. The results are placed in the specified accumulator. After the operation, each bit of the specified accumulator will be the logical "OR" of the corresponding bits of the specified memory location and the accumulator before the operation.) This instruction may be used in the Immediate, Direct, or Indexed addressing mode (two byte instructions) or in the Extended addressing mode (three byte instruction). The CCR is affected as follows:

    H: Not affected.
    I: Not affected.
    N: Set if most significant bit of the result is set; cleared otherwise.
    Z: Set if all bits of the result are cleared; cleared otherwise.
    V: Cleared.
    C: Not affected.

49. **PSH** (Push Data Onto Stack). The contents of the specified accumulator (A or B) are stored in the stack at the address contained in the Stack Pointer, and are then decremented by one. This instruction may only be used in the Inherent addressing mode (one byte instruction). The condition of the CCR is not affected.

50. **PUL** (Pull Data From Stack). The Stack Pointer is incremented and the specified accumulator (A or B) is then loaded from the stack at the address indicated by the Stack Pointer. This instruction may only be used in the Inherent addressing mode (one byte instruction). The condition of the CCR is not affected.

51. **ROL** (Rotate Left). Shifts all bits of the specified accumulator (A or B) or the contents of the specified memory location one place to the left. Bit 0 is loaded from the C bit of the CCR and bit 7 is then loaded into the C bit. This instruction may be used in the Inherent addressing mode (two byte instruction), the Indexed addressing mode (two byte instruction), or the Extended addressing mode (three byte instruction). The CCR is affected as follows:

    H: Not affected.
    I: Not affected.
    N: Set if most significant bit of the result is set; cleared otherwise.
    Z: Set if all bits of the result are cleared; cleared otherwise.
    V: Set if, after the completion of the operation, EITHER (N is set and C is cleared) OR (N is cleared and C is set); cleared otherwise.
    C: Set if, before the operation, the most significant bit of the ACCX or M was set; cleared otherwise.

52. **ROR** (Rotate Right). Shifts all bits of the specified accumulator (A or B) or the contents of the specified memory location one place to the right. Bit 7 is loaded from the C bit of the CCR and bit 0 is then loaded into the C bit. This instruction may be used in the Inherent addressing mode (one byte instruction), the Indexed addressing mode (two byte instruction), or the Extended addressing mode (three byte instruction). The CCR is affected as follows:

    H: Not affected.
    I: Not affected.
    N: Set if most significant bit of the result is set; cleared otherwise.
    Z: Set if all bits of the result are cleared; cleared otherwise.
    V: Set if, after the completion of the operation, EITHER (N is set and C is cleared) OR (N is cleared and C is set); cleared otherwise.
    C: Set if, before the operation, the least significant bit of the ACCX or M was set; cleared otherwise.

53. **RTI** (Return from Interrupt). The contents of the CCR, ACCA, ACCB, Index Register, and Program Counter will be restored from the stack. Note that the interrupt

mask bit (I bit) will be reset if, and only if, the corresponding bit stored in the stack is a logic 0. This instruction may only be used in the Inherent addressing mode (one byte instruction). The only bit in the CCR that is affected by this instruction is the I bit, as previously discussed.

54. **RTS** (Return from Subroutine). The Stack Pointer is incremented by 1. The contents of the memory location addressed by the Stack Pointer are loaded into the 8 highest bits of the Program Counter. The Stack Pointer is then once again incremented by one, and the contents of that memory location are loaded into the 8 lowest bits of the Program Counter. This instruction may only be used in the Inherent addressing mode (one byte instruction). The condition of the CCR is not affected.

55. **SBA** (Subtract Accumulators). Subtracts the contents of ACCB from the contents of ACCA, and places the result in ACCA. The contents of ACCB are not affected. This instruction can only be used in the Inherent addressing mode (once by instruction). The CCR is affected as follows:

    H: Not affected.
    I: Not affected.
    N: Set if most significant bit of the result is set; cleared otherwise.
    Z: Set if all bits of the result are cleared; cleared otherwise.
    V: Set if there was two's complement overflow as a result of the operation.
    C: Carry is set if the absolute value of accumulator B plus previous carry is larger than the absolute value of accumulator A; reset otherwise.

56. **SBC** (Subtract with Carry). Subtracts the contents of the indicated memory location and the C bit of the CCR from the contents of the indicated accumulator (A or B), and places the result in the indicated accumulator. This instruction may be used in the Immediate, Direct, or Indexed addressing modes (two byte instructions) or the Extended addressing mode (three byte instruction). The CCR is affected as follows:

    H: Not affected.
    I: Not affected.
    N: Set if most significant bit of the result is set; cleared otherwise.
    Z: Set if all bits of the result are cleared; cleared otherwise.
    V: Set if there was two's complement overflow as a result of the operation; cleared otherwise.
    C: Carry is set if the absolute value of the contents of memory plus previous carry is larger than the absolute value of the accumulator; reset otherwise.

57. **SEC** (Set Carry). Sets (logic 1) the carry bit (C bit) in the CCR. This instruction may only be used in the Inherent addressing mode (one byte instruction). Only the logic state of the C bit is affected by this instruction.

58. **SEI** (Set Interrupt Mask). Sets (logic 1) the interrupt mask bit (I bit) in the CCR. When this is set, the MPU is inhibited from servicing an interrupt from a peripheral device. This condition will remain until the interrupt mask has been cleared (logic 0). This instruction may only be used in the Inherent addressing mode (one byte instruction). Only the logic state of the I bit is affected by this instruction.

59. **SEV** (Set Two's Complement Overflow Bit). Sets (logic 1) the two's complement overflow bit (V bit) in the CCR. This instruction may only be used in the Inherent addressing mode (one byte instruction). Only the logic state of the V bit is affected by this instruction.

60. **STA** (Store Accumulator). Stores the contents of the specified accumulator (A or B) into the indicated memory location. The contents of the specified accumulator remain unchanged. This instruction may be used in the Direct or Indexed addressing modes (two byte instructions) or the Extended addressing mode (one byte instruction). The CCR is affected as follows:

    H: Not affected.
    I: Not affected.
    N: Set if most significant bit of the contents of ACCX is set; cleared otherwise.
    Z: Set if all bits of the contents of ACCX are cleared; cleared otherwise.
    V: Cleared.
    C: Not affected.

61. **STS** (Store Stack Pointer). Stores the more significant byte of the Stack Pointer in memory at the address specified by the program, and stores the less significant byte of the Stack Pointer at the next location in memory (one plus the address specified by the program). This instruction may be used in the Direct or Indexed addressing modes (two byte instructions) or in the Extended addressing mode (three byte instruction). The CCR is affected as follows:

    H: Not affected.
    I: Not affected.
    N: Set if most significant bit of the stack pointer is set; cleared otheriwise.
    Z: Set if all bits of the stack pointer are cleared; cleared otherwise.
    V: Cleared.
    C: Not affected.

62. **STX** (Store Index Register). Stores the more significant byte of the Index Register in memory at the address specified by the program, and then stores the less significant byte of the Index Register at the next location in memory (one plus the address specified by the program). This instruction may be used in the Direct or Indexed addressing modes (two byte instructions) or the Extended addressing mode (three byte instruction). The CCR is affected as follows:

H: Not affected.
I: Not affected.
N: Set if most significant bit of the index register is set; cleared otherwise.
Z: Set if all bits of the index register are cleared; cleared otherwise.
V: Cleared.
C: Not affected.

63. **SUB** (Subtract). Subtracts the contents of the specified memory location from the contents of the indicated accumulator (A or B), and places the result in the specified accumulator. This instruction may be used in the Immediate, Direct, or Indexed addressing modes (two byte instructions) or the Extended addressing mode (three byte instruction). The CCR is affected as follows:

H: Not affected.
I: Not affected.
N: Set if most significant bit of the result is set; cleared otherwise.
Z: Set if all bits of the result are cleared; cleared otherwise.
V: Set if there was two's complement overflow as a result of the operation; cleared otherwise.
C: Set if the absolute value of the contents of memory are larger than the absolute value of the accumulator; reset otherwise.

64. **SWI** (Software Interrupt). The Program Counter is incremented by one, and then the contents of the Program Counter (2 bytes), Index Register (2 bytes), ACCA (1 byte), ACCB (1 byte), and finally, the contents of the CCR (1 byte). The Stack Pointer is decremented by one after each byte is loaded into the stack. The interrupt mask bit (I bit) in the CCR is then set (logic 1). The Program Counter is then loaded with the address stored in the software interrupt pointer located at memory locations n-5 and n-4 (where n is the address corresponding to a high state on all lines of the address bus -FFFF). This instruction may only be used in the Inherent addressing mode (one byte instruction). Only the logic state of the I bit is affected by this instruction.

65. **TAB** (Transfer from Accumulator A to Accumulator B). Moves the contents of ACCA into ACCB. The former contents of ACCB are lost. The contents of ACCA are not affected. This instruction may only be used in the Inherent addressing mode (one byte instruction). The CCR is affected as follows:

H: Not affected.
I: Not affected.
N: Set if most significant bit of the contents of the accumulator is set; cleared otherwise.
Z: Set if all bits of the contents of the accumulator are cleared; cleared otherwise.
V: Cleared.
C: Not affected.

66. **TAP** (Transfer from Accumulator A to Processor Condition Codes Register). Transfers bit positions 0 through 5 of ACCA to the corresponding bit positions of the CCR. The original contents of ACCA remain unchanged. This instruction may only be used in the Inherent addressing mode (one byte instruction). The condition of the CCR is determined by the transfer of data from ACCA.

67. **TBA** (Transfer from Accumulator B to Accumulator A). Moves the contents of ACCB into ACCA. The former contents of ACCA are lost. The contents of ACCB are not affected. This instruction may only be used in the Inherent addressing mode (one byte instruction). The CCR is affected as follows:

H: Not affected.
I: Not affected.
N: Set if most significant accumulator bit is set; cleared otherwise.
Z: Set if all accumulator bits are cleared; cleared otherwise.
V: Cleared.
C: Not affected.

68. **TPA** (Transfer from Processor Condition Codes Register to Accumulator A). Transfers bit positions 0 through 5 of the CCR to the corresponding bit positions of the ACCA. The original contents of the CCR remain unchanged. This instruction may only be used in the Inherent addressing mode (one byte instruction).

69. **TST** (Test). Set bits N and Z of the CCR according to the contents of the specified accumulator (A or B) or memory location. This instruction may be used in the Inherent addressing mode (one byte instruction), the Indexed addressing mode (two byte instruction), or the Extended addressng mode (three byte instruction). The CCB is affected as follows:

H: Not affected.
I: Not affected.
N: Set if most significant bit of the contents of ACCX or M is set; cleared otherwise.
Z: Set if all bits of the contents of ACCX or M are cleared; cleared otherwise.
V: Cleared.
C: Cleared.

70. **TSX** (Transfer from Stack Pointer to Index Register). Loads the Index Register with one plus the contents of the Stack Pointer. The contents of the Stack Pointer remain unchanged. This instruction may only be used in the Inherent addressing mode (one byte instruction). The contents of the CCR are not affected by this instruction.

71. **TXS** (Transfer from Index Register to Stack Pointer). Loads the Stack Pointer with the contents of the Index register, minus one. The contents of the Index Register remain unchanged. This instruction may only be used

in the Inherent addressing mode (one byte instruction). The contents of the CCR are not affected by this instruction.

72. **WAI** (Wait for Interrupt). The Program Counter is incremented by one and then the Program Counter (2 bytes), Index Register (2 bytes), ACCA (1 byte), ACCB (1 byte), and finally the contents of the CCR (1 byte). The Stack Pointer is decremented by one after each byte is loaded into the stack. Execution of the program is then suspended until an interrupt from a peripheral device is signalled by the interrupt request control input changing to a logic low state. When an interrupt occurs, and providing that the interrupt mask bit (I bit) is cleared (logic 0), execution proceeds as follows:
    a. The interrupt mask bit (I bit) is set (logic 1).
    b. The Program Counter is loaded with address stored in the internal pointer at memory locations n-7 and n-6 (where ns is the address corresponding to a high state on all lines of the address bus).

    This instruction may only be used in the Inherent addressing mode (one byte instruction).

2.4 WRITING PROGRAMS

By now, you may be somewhat overwhelmed by all of the programming information presented in the preceding paragraphs. Don't despair. It's really not all that complex. All you have to do is become familiar with the various instructions and you can make your Educator II do many things for you. The best way to become familiar with the instructions is to write small programs to begin with and then make progressively larger programs as you gain more experience. To help you with entering your programs, all 72 of the executable instructions in each of the six addressing modes are listed in alphabetic order in the following table (Table 2-12), along with their respective binary equivalents. Therefore, when you enter your program, you can find the exact front panel switch settings for each of the instructions in your program. In addition, the following paragraphs will help you get started in writing your own programs.

TABLE 2-12. Executable Instructions to Binary Code Conversion Chart

MNEMONIC INSTRUCTION	ADDRESSING MODE	BINARY NUMBER
1) ABA	Inherent	0001 1011
2) ADCA	Immediate	1000 1001
	Direct	1001 1001
	Extended	1011 1001
	Indexed	1010 1001
ADCB	Immediate	1100 1001
	Direct	1101 1001
	Extended	1111 1001
	Indexed	1110 1001
3) ADDA	Immediate	1000 1011
	Direct	1001 1011
	Extended	1011 1011
	Indexed	1010 1011
ADDB	Immediate	1100 1011
	Direct	1101 1011
	Extended	1111 1011
	Indexed	1110 1011
4) ANDA	Immediate	1000 0100
	Direct	1001 0100
	Extended	1011 0100
	Indexed	1010 0100
ANDB	Immediate	1100 0100
	Direct	1101 0100
	Extended	1111 0100
	Indexed	1110 0100
5) ASL	Extended	0111 1000
	Indexed	0110 1000
ASLA	Inherent	0100 1000
ASLB	Inherent	0101 1000
6) ASR	Extended	0111 0111
	Indexed	0110 0111
ASRA	Inherent	0100 0111
ASRB	Inherent	0101 0111
7) BCC	Inherent	0010 0100
8) BCS	Relative	0010 0101
9) BEQ	Relative	0010 0111
10) BGE	Relative	0010 1100

MNEMONIC INSTRUCTION	ADDRESSING MODE	BINARY NUMBER
11) BGT	Relative	0010 1110
12) BHI	Relative	0010 0010
13) BITA	Immediate	1000 0101
	Direct	1001 0101
	Extended	1011 0101
	Indexed	1010 0101
BITB	Immediate	1100 0101
	Direct	1101 0101
	Extended	1111 0101
	Indexed	1110 0101
14) BLE	Relative	0010 1111
15) BLS	Relative	0010 0011
16) BLT	Relative	0010 1101
17) BMI	Relative	0010 1011
18) BNE	Relative	0010 0110
19) BPL	Relative	0010 1100
20) BRA	Relative	0010 0000
21) BSR	Relative	1000 1101
22) BVC	Relative	0010 1000
23) BVS	Relative	0010 1001
24) CBA	Inherent	0001 0001
25) CLC	Inherent	0000 1100
26) CLI	Inherent	0000 1110
27) CLR	Extended	0111 1111
	Indexed	0110 1111
CLRA	Inherent	0100 1111
CLRB	Inherent	0101 1111
28) CLV	Inherent	0000 1010
29) CMPA	Immediate	1000 0001
	Direct	1001 0001
	Extended	1011 0001
	Indexed	1010 0001
CMPB	Immediate	1100 0001
	Direct	1101 0001
	Extended	1111 0001
	Indexed	1110 0001
30) COM	Extended	0111 0011

MNEMONIC INSTRUCTION	ADDRESSING MODE	BINARY NUMBER
	Indexed	0110 0011
COMA	Inherent	0100 0011
COMB	Inherent	0101 0011
31) CPX	Immediate	1000 1100
	Direct	1001 1100
	Extended	1011 1100
	Indexed	1010 1100
32) DAA	Inherent	0001 1001
33) DEC	Extended	0111 1010
	Indexed	0110 1010
DECA	Inherent	0100 1010
DECB	Inherent	0100 1010
34) DES	Inherent	0011 0100
35) DEX	Inherent	0000 1001
36) EORA	Immediate	1000 1000
	Direct	1001 1000
	Extended	1011 1000
	Indexed	1010 1000
EORB	Immediate	1100 1000
	Direct	1101 1000
	Extended	1111 1000
	Indexed	1110 1000
37) INC	Extended	0111 1100
	Indexed	0110 1100
INCA	Inherent	0100 1100
INCB	Inherent	0101 1100
38) INS	Inherent	0011 0001
39) INX	Inherent	0000 1000
40) JMP	Extended	0111 1110
	Indexed	0110 1110
41) JSR	Extended	1010 1101
	Indexed	1001 1101
42) LDAA	Immediate	1000 0110
	Direct	1001 0110
	Extended	1011 0110
	Indexed	1010 0110
LDAB	Immediate	1100 0110
	Direct	1101 0110
	Extended	1111 0110
	Indexed	1110 0110
43) LDS	Immediate	1000 1110
	Direct	1001 1110
	Extended	1011 1110
	Indexed	1010 1110
44) LDX	Immediate	1100 1110
	Direct	1101 1110
	Extended	1111 1110
	Indexed	1110 1110
45) LSR	Extended	0111 0100
	Indexed	0110 0100
LSRA	Inherent	0100 0100
LSRB	Inherent	0101 0100
46) NEG	Extended	0111 0000
	Indexed	0110 0000
NEGA	Inherent	0100 0000
NEGB	Inherent	0101 0000
47) NOP	Inherent	0000 0001
48) ORAA	Immediate	1000 1010
	Direct	1001 1010
	Extended	1011 1010
	Indexed	1010 1010
ORAB	Immediate	1100 1010
	Direct	1101 1010
	Extended	1111 1010
	Indexed	1110 1010
49) PSHA	Inherent	0011 0110
	Inherent	0011 0111
50) PULA	Inherent	0011 0010
PULB	Inherent	0011 0011
51) ROL	Extended	0111 1001
	Indexed	0110 1001
ROLA	Inherent	0100 1001
ROLB	Inherent	0101 1001
52) ROR	Extended	0111 0110
	Indexed	0110 0110
RORA	Inherent	0100 0110
RORB	Inherent	0101 0110
53) RTI	Inherent	0011 1011
54) RTS	Inherent	0011 1001
55) SBA	Inherent	0001 0000
56) SBCA	Immediate	1000 0010
	Direct	1001 0010
	Extended	1011 0010
	Indexed	1010 0010
SBCB	Immediate	1100 0010
	Direct	1101 0010
	Extended	1111 0010
	Indexed	1110 0010
57) SEC	Inherent	0000 1101
58) SEI	Inherent	0000 1111
59) SEV	Inherent	0000 1011
60) STAA	Direct	1001 0111
	Extended	1011 0111
	Indexed	1010 0111
STAB	Direct	1101 0111
	Extended	1111 0111
	Indexed	1110 0111
61) STS	Direct	1001 1111
	Extended	1011 1111
	Indexed	1010 1111
62) STS	Direct	1101 1111
	Extended	1111 1111
	Indexed	1110 1111
63) SUBA	Immediate	1000 0000
	Direct	1001 0000
	Extended	1011 0000
	Indexed	1010 0000
SUBB	Immediate	1100 0000
	Direct	1101 0000
	Extended	1111 0000
	Indexed	1110 0000
64) SWI	Inherent	0011 1111
65) TAB	Inherent	0001 0110
66) TAP	Inherent	0000 0110
67) TBA	Inherent	0001 0111
68) TPA	Inherent	0000 0111
69) TST	Extended	0111 1101
	Indexed	0110 1101
TSTA	Inherent	0100 1101
TSTB	Inherent	0101 1101
70) TSX	Inherent	0011 0000
71) TXS	Inherent	0011 0101
72) WAI	Inherent	0011 1110

2.4.1 SIMPLE PROGRAM SEGMENTS

Instruction and data bytes within the program are differentiated simply by the sequence in which they occur in the program. When a Reset is initiated, the processor is synchronized with the program. If the program is faulty and the correct number of bytes following the instruction are not present, then the MPU will interpret data as an instruction. When the data and instructions get out of sequence, they will remain out of sequence until a Reset is once again initiated. This condition is referred to as a processor crash. Since all instruction and data sets may be either 1, 2, or 3 bytes in length, you'll have to be careful to match the desired instruction with the correct number of data bytes.

Before complicating the issue any further, let's use a few instructions to write and examine the operation of a simple program.

PROBLEM NUMBER 1 :

Add 2 and 3

ADDRESS (Hex)	MNEMONIC INSTRUCTION	PROGRAM (Hex)	DESCRIPTION
80	LDAA	86	Load Accumulator A
81	2	02	Data = 2
82	ADDA	8B	Add to Accumulator A
83	3	03	Data = 3

Note that address 80 is located at the bottom of RAM memory. (Remember, the standard size of RAM memory supplied with Educator II extends from 80 to F1. If you purchased the optinal 128 byte RAM memory, you can start your program at 00.) Note also that all of the program steps are written sequentially into memory. This is necessary because the MPU automatically steps to the next byte of memory unless otherwise instructed by the program. Before starting to load your program following a Reset of the microcomputer system, no meaningful numbers exist in the system. Therefore, the first thing that must be done is load the accumulator (in this case, accumulator A). The first instruction of the program (LDAA) tells the MPU to load the next byte of data stored in memory (address location 81) into accumulator A. Thus, the first number to be added in Problem 1 (2) is loaded into memory directly following the LDAA instruction. The next instruction (8B) may then be loaded into memory location 82 to tell the MPU to add the contents of the following byte of memory (3) to the contents of Accumulator A (2). Thus, after execution of the complete program, the number 5 is contained in accumulator A.

Sometimes the referenced data does not directly follow the instruction. Instead, information bytes follow that tell the MPU where, in memory, the data can be found. Rather than refer to these information bytes as either data or address bytes, they are often referred to as operands (information operated on by an instruction). The example shown in Problem 2 provides this type of programming. (Note that this example is the same as the example provided in Problem 1, except for the method used to write the program.)

PROBLEM NUMBER 2 :

Add 2 and 3

Where: 2 is located at address location 97
3 is located at address location 98

ADDRESS (Hex)	MNEMONIC INSTRUCTION	PROGRAM (Hex)	DESCRIPTION
80	LDAA	B6	Load Accumulator A.
81	00	00	Most Significant Byte of Address.
82	97	97	Least Significant Byte of Address.
83	ADDA	BB	Add to Accumulator A.
84	00	00	Most Significant Byte of Address.
85	98	98	Least Significant Byte of Address.
97	02	02	Data to be loaded into Accumulator A.
98	03	03	Data to be added to Accumulator A.

As you can guess, the same answer (5) results from executing either of the programs shown in the two previous examples. The only difference is that two different addressing modes have been used. In the first example, the Immediate addresing mode (data immediately follows instruction) was used, while in the second example, the Extended addressing mode (two byte address follows instruction) was used. The latter method of programming is very useful when the numbers to be added are changed to obtain different results. Thus, instead of having to rewrite the entire program to obtain results using different numbers, all you have to do is change the numbers stored in two memory locations (such as memory locations 97 and 98 in the previous example).

Only one more problem still remains: How do you detect the results of the operation? As previously mentioned, in the Educator II, the eight LED indicators are used to provide you with the results of program executions. But, in order to present the results on the LED indicators, a store operation must be executed. You may also recall that the address location of the LED indicators is 4002. Thus, in order to detect the results of executing any operation or program, you must store the results at address location 4002. The example shown below presents the program for Problem 1, but adds the store instruction in order to permit you to read the results.

PROBLEM NUMBER 3 :

Add 2 and 3. Display results on LEDs.

ADDRESS (Hex)	MNEMONIC INSTRUCTION	PROGRAM (Hex)	DESCRIPTION
80	LDAA	86	Same as Problem 1
81	2	02	
82	ADDA	8B	
83	3	03	
84	STAA	B7	Store Accumulator A
85	40	40	Most Significant Byte of LED Address.
86	02	02	Least Significant Byte of LED Address.

Note that the store instruction must be used in the Extended addressing mode, since the address of the LED indicators (4002) requires two bytes.

Of course, we've only covered the use of Accumulator A in the preceding examples. We could have also written the same program using Accumulator B. In addition to these two accumulators, a third (phantom) accumulator can also be used when memory referenced instructions (all instructions beginning with 7) are included. Although this phantom accumulator cannot be seen in our simplified block diagram of Figure 2-1, its existence is apparent to the programmer from looking at the instruction set. For example, if a Complement instruction (COM=73_{16}) were executed on a memory location, the following sequence would occur:

1. Decode instruction (op code = 73).
2. Read in two bytes of address following op code.
3. Load byte at specified address into phantom accumulator.
4. Perform complement function.
5. Store modified byte back into specified memory location.

As you can see, these memory referenced instructions can save a lot of programming, since they are equivalent to a Load Accumulator, Complement Accumulator, and Store Accumulator series of instructions.

2.4.2 COMPLETING PROGRAMS

Every program must have a beginning and an ending. What we've written so far are simply program segments and not completely self-contained programs. To begin a program, you must tell the MPU where to go to begin program execution. Since any of 65,536 memory locations may be addressed, a specific start vector must be defined. In the Educator II, the start vector is automatically taken care of by the program stored in the ROM (refer to paragraph 1.2.3.4). Program endings, however, must be prepared by you when writing your programs, or the processor will crash. Two different types of program terminations that are easy to use are jumps to the start of the program and local jump loops. To illustrate both of these types of program terminations, let's once again look at the addition sample shown in Problem 3.

PROBLEM NUMBER 4 :

Jump to Beginning Termination

ADDRESS (Hex)	MNEMONIC INSTRUCTION	PROGRAM (Hex)	DESCRIPTION
80	LDAA	86	
81	2	02	
82	ADDA	8B	
83	3	03	Same as Problem 3
84	STAA	B7	
85	40	40	
86	02	02	
87	JMP	7E	Jump to address 0080,
88	00	00	which is the beginning
89	80	80	of the program.

Note that the program repeats itself until halted by the programmer. Since the answer is always 5, the display doesn't change. On the other hand, if you'd like only one pass at the problem, then a local loop can be created to terminate your program. This type of termination is illustrated in the following example.

PROBLEM NUMBER 5 :

ADDRESS (Hex)	MNEMONIC INSTRUCTION	PROGRAM (Hex)	DESCRIPTION
80	LDAA	86	
81	2	02	
82	ADDA	8B	
83	3	03	Same as Problem 3.
84	STAA	B7	
85	40	40	
86	02	02	
87	JMP	7E	Jump to address 0087,
88	00	00	which is the beginning
89	87	87	of the jump instruction itself.

In this example, the program executes the Add routine once and simply loops (or circles) on itself indefinitely.

Unfortunately, most programs require that a series of instructions be performed in sequence. If the program is the same, but the values change, the programmer may find that the program can be greatly simplified by making use of the stack (stack manipulation). In the following example, a simple stack manipulation is illustrated to show you how to add a series of numbers and obtain the results.

PROBLEM NUMBER 6 :

Add 1, 2, 3, 4, and 5

ADDRESS (Hex)	MNEMONIC INSTRUCTION	PROGRAM (Hex)	DESCRIPTION
80	LDAA	86	
81	00	00	Load and Store Most
82	STAA	97	Significant Byte (00) of
83	EE	EE	Stack Pointer address stored at address location 00EE.
84	LDAA	86	
85	DF	DF	Load and Store Least
86	STAA	97	Significant Byte (DF)
87	EF	EF	of Stack Pointer address stored at address location 00EF.
88	LDS	8E	
89	00	00	Loads Stack Pointer with starting address of
8A	EE	EE	stack (00DF) stored at address location 00EE.
8B	LDAB	C6	Load ACCB with 5_{16}.
8C	5	05	
8D	PSHB	37	Push contents of ACCB (5) into stack.
8E	LDAB	C6	Load ACCB with 4_{16}.
8F	4	04	
90	PSHB	37	Push contents of ACCB (4) into stack.
91	LDAB	C6	Load ACCB with 3_{16}.
92	3	03	
93	PSHB	37	Push contents of ACCB (3) into stack.

PROBLEM NUMBER 6 :

Add 1, 2, 3, 4, and 5

ADDRESS (Hex)	MNEMONIC INSTRUCTION	PROGRAM (Hex)	DESCRIPTION
94	LDAB	C6	Load ACCB with 2_{16}.
95	2	02	
96	LDAA	86	Load ACCA with 1_{16}.
97	1	01	
98	ABA	1B	Add ACCB to ACCA. Store results in ACCA.
99	PULB	33	Pull data (3) from stack. Load into ACCB.
9A	ABA	1B	Add ACCB to ACCA. Store results in ACCA.
9B	PULB	33	Pull data (4) from stack. Load into ACCB.
9C	ABA	1B	Add ACCB to ACCA. Store results in ACCA.
9D	PULB	33	Pull data (5) from stack. Load into ACCB.
9E	ABA	1B	Add ACCB to ACCA. Store results in ACCA.
9F	STAA	B7	Store ACCA at address location 4002 (location of LED indicators).
A0	40	40	
A1	02	02	
A2	JMP	7E	Jump to address 00A2, which is the beginning of the jump instruction itself.
A3	00	00	
A4	A2	A2	
DD		03	STACK. Contents loaded into stack during program execution. (Remember, Stack Pointer is decremented when loaded, incremented when unloaded.)
DE		04	
DF		05	
EE		00	VECTOR ADDRESS. Points to (contains) the starting address of the program stack which is loaded into Stack Pointer.
EF		DF	

When the above program is executed, the pattern displayed on the eight LED indicators will be: 0000 1111. This display represents the decimal number 15.

When preparing a program in which you intend to use a stack manipulation, one of the first things you must do is assign the starting address of the stack. Remember, when the stack is loaded from the MPU, it is decremented before the next byte of data is loaded. Therefore, remember to leave enough room in memory to prevent the stack from overwriting your program. This is normally done by starting your main program at the bottom of the RAM memory (in this case, 80), while assigning the stack a starting address near the very top of the RAM memory (00DF in the above program). The above program example shows one method of establishing the starting address (vector address) of the stack (this is done by the operation performed between addresses 80 and 87). Between addresses 88 and 8A, this vector address (00DF) is loaded from memory (address locations 00EE and 00EF) into the Stack Pointer in the MPU. The stack is now ready to be used during program execution.

The data to be used during program execution is then loaded into the stack by instructions located between address locations 8B and 93. At addresses 94 and 95, accumulator B is loaed with the 2, while accumulator A is loaded with a 1 at addresses 96 and 97. At address 98, the contents of both accumulators are added together and the results are stored in accumulator A. The next byte of data is then pulled from the stack and loaded into accumulator B (address 99). This action continues until all data has been pulled from the stack and all accumulator additions have been completed (address 9E). After the last addition, a binary number (15) will remain in accumulator A. This number is then loaded into the LED indicators for display.

For more information concerning the Hardware used in the Educator II, consult Motorola's M6800 MICROPROCESSOR APPLICATIONS MANUAL. For further Software information, refer to Motorola's M6800 PROGRAMMING REFERENCE MANUAL.

Part 3
Operation Phase

SECTION 1
LOADING YOUR PROGRAM

1.1 INTRODUCTION

Now that you've built your kit, learned about its internal operation, and read the fundamentals needed to write programs, you'll want to load your programs into the Educator II for execution. The following paragraphs will tell you how this is done.

1.2 INITIALIZING YOUR PROGRAM

To initialize the Educator II, all that must be done is to apply power and momentarily position the RESET/DEPOSIT switch to RESET. (You will recall that just applying power will initialize the Educator II. However, it is always best to manually initiate a Reset sequence just to guarantee that the sequence is performed). After the Reset sequence has been completed, you may begin loading your program.

1.3 LOADING YOUR PROGRAM

In order to load your program, you must first set the starting address of where you want the program to begin. If you have *not* installed the optional memory device, your program can be started at any point from 80_{16} to $F1_{16}$. Remember, however, that the Program Counter in the MPU is automatically incremented by one after each address location is loaded. Thus, in order to have the greatest amount of memory storage for your program, it is recommended that you begin all programs at 80.

If you have installed the optional 128 byte memory device, you may begin your programs at 00_{16} and load them up to address $F1_{16}$. (Address $F2_{16}$ through FF_{16} are reserved for the Educator II itself. This area is referred to as the scratchpad memory area.)

To load your program, perform the following steps:

1. After the Reset sequence has been completed by momentarily positioning the RESET/DEPOSIT switch to RESET, enter the starting address, in binary, on the eight front panel switches. The starting address is normally 80=1000 0000.
2. After the starting address has been established by the eight front panel switches, momentarily position the LEAST/GO/HALT switch to the LEAST position to load the starting address.

NOTE

Since the Educator II is equipped with only 128 bytes (or 256 bytes) of RAM, only the Least Significant Byte of the starting address needs to be loaded. (The Most Significant Byte is assumed to be 00.) However, if you have constructed modules of additional memory to be used with your Educator II, you would have to load the Most Significant Byte of the starting address first by momentarily positioning the MOST/EXAMINE switch to MOST. You would then load the Least Significant Byte of the starting address by momentarily positioning the LEAST/GO/HALT switch to LEAST. In this way, any of the 65,536 memory locations can be addressed.

3. After the starting address has been loaded, the first byte of your program can be entered at the starting address. This is done by setting the eight front panel switches to correspond to the binary equivalent of your first instruction.
4. After the eight front panel switches have been set to enter the desired instruction, momentarily position the RESET/DEPOSIT switch to DEPOSIT. This loads the first byte of your program into the memory location selected by the starting address. The address is then automatically incremented by one to permit you to load the second byte of your program into memory.
5. Continue loading your program, byte by byte, into memory by repeating steps 3 and 4. After your program has been completely loaded, proceed to the next section (Section 2).

SECTION 2
REVIEWING YOUR PROGRAM

2.1 INTRODUCTION

After your program has been completely loaded into memory, it should be reviewed for accuracy, especially if the program was long and required a lot of time to enter. The Educator II provides a very simple means of examining your program.

2.2 LOADING THE STARTING ADDRESS

The starting address must once again be loaded in order to examine your program. This is done in the same manner as previously explained for loading your program.

1. Enter the program starting address on the eight front panel switches (normally 80).
2. Momentarily position the LEAST/GO/HALT switch to the LEAST position. This loads the starting address and displays the first byte of your program.

2.3 EXAMINING YOUR PROGRAM

After the program starting address has been loaded and the data stored at the starting address has been displayed on the LED indicators, you may compare this displayed data with the first byte of your program to guarantee that they are the same. You can then check each byte of the program stored in memory by positioning the MOST/EXAMINE switch to EXAMINE. Each time this switch is positioned to EXAMINE, the Program Counter is advanced by one to the next address location and the contents are displayed.

If an error is found in the program stored in memory, you can easily correct it by entering the proper data on the eight front panel switches and momentarily positioning the RESET/DEPOSIT switch to DEPOSIT. This will replace the data presently stored at the address location with the data entered from the eight front panel switches. After all changes have been made, you should once again examine the program to make sure it's correct. After your complete program has been loaded and verified, you can execute the complete program by momentarily positioning the LEAST/GO/HALT switch to GO/HALT.

You can stop program execution at any time by once again momentarily depressing the LEAST/GO/HALT switch to GO/HALT. This can be done because the GO/HALT position of this switch provides an alternate action function.

If your program does not perform correctly, refer to the next section (Section 3) for information on debugging it (finding the errors).

SECTION 3
DEBUGGING YOUR PROGRAM

3.1 INTRODUCTION

After entering and checking your program, you may still have some problems due to an incorrect program operation. Since any program operation error occurs only during program execution, trying to find it could be difficult. However, the instruction set used with your Educator II contains a Software Interrupt (SWI) instruction that can be very helpful in debugging (finding the errors in) your program.

3.2 USING SOFTWARE INTERRUPTS

The Software Interrupt instruction (SWI=$3F_{16}$) is very helpful for program development and termination. It effectively acts like a software halt, just as if the GO/HALT switch on the front panel had been momentarily toggled. When an SWI instruction is executed, the microcomputer system displays the address of the SWI instruction and returns to the Halt mode, signified by the illumination of the GO indicator.

Programs may also be terminated by using the SWI instruction as the last instruction in a program. However, if this is done, you cannot read the results of program execution on the LED indicators, since the address of the SWI instruction is displayed before entering the Halt routine.

After program execution has been halted by an SWI instruction, execution can be continued by momentarily positioning the LEAST/GO/HALT switch to GO/HALT. This permits several SWI instructions to be included into a new program at its initial writing to check program execution through various branches or jumps. Loss of an anticipated halt can help pinpoint trouble areas.

In addition to simply determining that the program has reached a certain point, you can also examine the MPU's internal registers when an SWI instruction occurs by successively depressing the EXAMINE switch. The first time the MOST/EXAMINE switch is momentarily positioned to EXAMINE following an SWI instruction, the contents of the Condition Code Register (stored at address $F4_{16}$) will be displayed on the LED indicators. Other registers can similarly be reviewed from successive memory locations. These locations are listed as follows.

MEMORY LOCATION (Hex)	CONTENTS
F4	CCR
F5	ACCB
F6	ACCA
F7	Index Register (high byte)
F8	Index Register (low byte)
F9	Program Counter (high byte)
FA	Program Counter (low byte)

When troubleshooting a new program, it is a common practice to replace an existing instruction with an SWI instruction in order to check the registers for clues to the failure. Before restarting the program, the SWI instruction should be replaced with the original instruction, and the vector address stored at F9 and FA (refer to above list) should be replaced by the start address of the program.

The ability to continue programs after an SWI instruction occurs allows you to halt an instruction sequence and input further instructions. For example, successive bytes may be entered under program control by pausing at the SWI and then continuing by another GO.

SECTION 4
CASSETTE OPERATION

4.1 INTRODUCTION

Data stored in the Educator II memory may be recorded onto an audio quality cassette through most recorders. Likewise, previously developed programs may be reloaded from a cassette tape. This capability greatly facilitates program writing and debugging, and expands the scope of uses for the Educator II.

The cassette is handy for program storage after you have loaded your initial program. Rather than running the risk of destroying some or all of the program during the first, possibly erratic execution, deposit it on cassette before trying the first execution. Likewise, record every significant modification before it's executed. This will save time in the long run.

Each cassette "RECORD" consists of four basic components: 1) Tone Leader, 2) Beginning Address, 3) Data Field, and 4) Tone Trailer. Functions 1 and 4 enable the Cassette Read program to recognize the beginning and ending of a record. The Beginning Address area (2 bytes) relieves the operator from having to indicate where the data is to be stored. Data fields of virtually any length (up to the RAM capability) can be entered by a single record read.

Cassette record (read) and playback (write) routines may also be used as subroutines to your other programs. If the operation is to be totally automatic, some means of implementing tape drive control from a PIA pin must be designed. You must also make certain that the cassette drive has been given sufficient time to come up to speed before commencing a cassette write operation.

If the read routine is to be totally subroutined, the record number to be read must be stored into memory location FF *in 2's complement* form before initiating the read routine. For example, the first record would be FF; the second FE, and so on. If you wish to enter the number directly through the switches or otherwise convert the binary number to 2's complement, the instruction NEG may be used. For example, instruction NEGA (40_{16}) will change the contents of ACCA to 2s complement.

Entry for the cassette write subroutine is 8300_{16} with the stack at FF before entry. Address for the start of dump is at F0 and F1; end of dump is F2 and F3.

4.2 RECORD PROCEDURE

Two key items must be known and well-defined before recording can be accomplished: what is to be recorded and where. The Educator II must be told the data area to be dumped (loaded onto tape in one continuous operation). The user must also select the tape or portion of tape where the record will be recorded. The following procedure provides a step-by-step method to record.

1. Connect the Educator II TAPE OUT phone jack to the recorder's Record, Input, or similarly marked input jack used for entering non-microphonic signals.
2. Place the tape in the position to be recorded. (Rewind to start a first record at the beginning of the tape.)
3. Load the address of the first byte of data to be recorded into memory locations F1 and F2 as follows:
   A. Momentarily position the RESET/DEPOSIT switch to RESET.
   B. Set the eight data switches to F1 (1111 0001).
   C. Momentarily position the LEAST/GO/HALT switch to LEAST.
   D. Set the eight data switches to the Most Significant Byte of the starting (first) address you want to record.
   E. Momentarily position the RESET/DEPOSIT switch to DEPOSIT.
   F. Set the eight data switches to the Least Significant Byte of the starting (first) address you want to record.
   G. Momentarily position the RESET/DEPOSIT switch to DEPOSIT.
4. Load the address of the last byte of data to be recorded into memory locations F3 and F4 as follows:
   H. Set the eight data switches to the Most Significant Byte of the ending (last) address you want to record.
   I. Momentarily position the RESET/DEPOSIT switch to DEPOSIT.
   J. Set the eight data switches to the Least Significant Byte of the ending (last) address you want to record.
   K. Momentarily position the RESET/DEPOSIT switch to DEPOSIT.
5. Set up the start vector address to cassette dump program as follows:
   A. Set the eight data switches to F9 (1111 1001).
   B. Momentarily position the LEAST/GO/HALT switch to LEAST.
   C. Set the eight data switches to 82 (1000 0010).
   D. Momentarily position the RESET/ DEPOSIT switch to DEPOSIT.
   E. Set the eight data switches to CF (1100 1111).
   F. Momentarily position the RESET/DEPOSIT switch to DEPOSIT.
6. Active tape recorder in its record mode.
7. Momentarily position the LEAST/GO/HALT switch to GO/HALT. The LED display will flash and change as data is transferred to the recorder. At the end of transmission, an 01 (0000 0001) will be displayed and the GO LED will illuminate.
8. Rewind the tape, disconnect the Educator II from the recorder, and verify that the data was recorded by listening to the normal audio playback.(You will probably have to lower the volume of the audio output signal.)

4.3 PLAYBACK PROCEDURE

Since the data recorded on the cassette contains a starting address, there are only two items that you must be concerned with to transfer programs back to the Educator II's memory from the cassette: cassette preparation and program vector address. The cassette playback program has excellent syllabic audio rejection even at the output clipping levels used for data playback. Verbal headers and comments may be added between tape records to clarify their intent, destination address, origination date or issue number. Bit 7 of the display will show audio modulation but bit 0 should remain extinguished until the tone leader is detected. Two entries for a cassette playback operation are available. One option (at $82F9_{16}$) allows reading of the first cassette file that is encountered. The other option (at $82F3_{16}$) reads a file number put into the data switches.

The following procedure provides a step-by-step method to playback data from the first file recorded on the cassette.

1. Connect the Educator II TAPE II phone jack to the recorder's External, External Speaker, Output, or similarly marked output jack.
2. Establish the starting vector address for the cassette read program as follows:
   A. Momentarily position the RESET/DEPOSIT switch to RESET if the Educator II is not already halted.
   B. Set the eight data switches to F9 (1111 1001).
   C. Momentarily position the LEAST/GO/HALT switch to LEAST.
   D. Set the eight data switches to 82 (1000 0010).
   E. Momentarily position the RESET/DEPOSIT switch to DEPOSIT.
   F. Set the eight data switches to F9 (1111 1001).
   G. Momentarily position the RESET/DEPOSIT switch to DEPOSIT.
   H. Momentarily position the LEAST/GO/HALT switch to GO/HALT.Note that the number 80 (1000 0000) will be displayed on the LED indicators. This indicates that the program is looking for a tone leader.
   I. Start tape recorder for playback. (When the display shows an 81, the tone leader has been detected.).
   J. After the program is completely loaded, a status indication will appear on the LED indicators and the GO indicator LED will illuminate. The status indication is coded as follows:

      02 (0000 0010) — NO MEMORY AT LOAD ADR. Program was detected but no RAM memory was at the address location indicated by the header or subsequent bytes.

      03 (0000 0011) — SYNC ERROR. Usually detected when the playback speed is radically different from the recorded speed. This could be caused by low tape recorder voltage during either record or playback.

      04 (0000 0100) — TONE TRAILER DETECTED. This indicates that no errors occurred during the playback operation. This is the normal indication.

The following procedure provides a step-by-step method to playback data from the file indicated by the eight data switches.

1. Connect the Educator II for playback operation as instructed in step 1 of the preceding procedure.
2. Momentarily position the RESET/DEPOSIT switch to RESET.
3. Set the eight data switches to F9 (1111 1001).
4. Momentarily position the LEAST/GO/HALT switch to LEAST.
5. Set the eight data switches to 82 (1000 0010).
6. Momentarily position the RESET/DEPOSIT switch to DEPOSIT.
7. Set the eight data switches to F3 (1111 0011).
8. Momentarily position the RESET/DEPOSIT switch to DEPOSIT.
9. Set the eight data switches to the desired file number of the record to be transferred (do not use zero).
10. Momentarily position the LEAST/GO/HALT switch to GO/HALT. Note that the number 80 (1000 0000) will be displayed on the LED indicators. This indicates that the program is looking for the tone leader of the desired record on cassette.
11. Start tape recorder for playback.

Part 4
Application Programs

SECTION 1
INSERTING DATA INTO EXISTING PROGRAMS

Title: EDIT INSERT

Purpose: To enter data into memory without overwriting existing code such as a DEPOSIT.

Application: When program revisions are required and "patches" are not desirable.

Description: Data on the front panel switches is inserted at the *panel address* when the program is executed. Successive execution of GO/HALT (G/H) will perform consecutive inserts in incrementally higher memory locations. Existing data in memory is moved up one location for each insert. The top of "Edit Buffer" is assumed to be BD_{16}. This may be moved by changing location D_{16}. The program begins at hex D4 and runs to EE. To use the program:

1. Load code
2. RESET
3. Data Switches = FA
4. LS Byte
5. Data Switches = D4
6. DEPOSIT
7. Data Switches = Location of insertion
8. LS Byte (Set panel address at insert point)
9. Data Switches = Data Switches = Data to be inserted
10. G/H
11. If more data is to be inserted, repeat steps 9 and 10.

EXAMPLE: Insert a "store into display" into a program that adds two numbers.

Present Program

HEX ADR	Data	
80	B6 4000	LDAA Switch
83	3F	SWI
84	F6 4000	LDAB Switch
87	1B	ABA
88	20 FE	SELF BRA

Desired Program

HEX ADR	Data	
80	B6 4000	
83	3F	
84	F6 4000	
87	1B	
88	B7 4002	DESIRED ADDITION
8B	20 FE	

Procedure for Insertion, assuming that Edit Insert is in memory:

1. RESET
2. Set Edit Insert start vector (D4) into FA.
3. Put insert address (88) into data switches and panel address by LS Byte.
4. Put B7 into switches and insert at 88 by depressing G/H.
5. Put 40 into swtiches and insert at 89 by G/H.
6. Put 02 into switches and insert at 8A by G/H.
7. Put 88 into switches, LS Byte, and review additions by EXAMINE.

EDIT INSERT

	Source	ADR	Object DATA
	LDX # TOP	D4	CE 00BD
UP	LDAA O,X	D7	A6 00
	STAA 1,X	D9	A7 01
	CPX PANADR	DB	9C FD
	BEQ PUSH	DD	27 03
	DEX	DF	09
	BRA UP	E0	20 F5
PUSH	LDAA DRA	E2	B6 4000
	STAA 0,X	E5	A7 00
	LDX PANADR	E7	DE FD
	INX	E9	08
	STX PANADR	EA	DF FD
	JMP EXEC	EC	7E 82A6

SECTION 2
DELETING DATA FROM EXISTING PROGRAMS

Title: EDIT DELETE

Purpose: To delete program bytes and preserve memory space as compared to NOP (01) DEPOSITS. Branches that cross the delete address must be recalculated.

Application: On execution, EDIT DELETE will move all bytes above the panel address down one memory position.

Description: The author's program begins at BE and ends at D3. However, the code may be entered anywhere since the only absolute address is a jump to the front panel executive. Top of Edit buffer is assumed to be 80_{16}. Program execution results in all bytes at panel address plus one (to 80_{16}) being moved down in memory by one location. Data at 80_{16} is duplicated in $7F_{16}$.

Procedure:

1. Load Edit Delete and establish vector to DELETE.
2. Put address of byte to be deleted into panel address by data switches and LS Byte.
3. Depress GO/HALT (G/H).
4. Display contains next byte that has been moved into panel address. Successive G/H will delete bytes at the designated panel address and drop the affected buffer area by one address.

EDIT DELETE

Assembly Language			Machine Code	
	LDX	PANADR	DE	FD
L00P4	LDAA	1, X	A6	01
	STAA	0, X	A7	00
	INX		08	
	CPX	#$80	8C	0080
	BNE	L00P4	26	F6
	LDX	PANADR	DE	FD
	LDAA	0, X	A6	00
	STAA	DRB	B7	4002
	JMP	82A6	7E	82A6

SECTION 3
SIMPLE LIGHT MEASUREMENT FOR EXPOSURE CONTROL

Title: RMSR

Purpose: To measure the relative value of resistance placed between CB2 and +5 VDC.

Application: Measuring relative light falling on a CdS cell from an enlarger. Light value gives indication of exposure time for a manual timer or may be passed on to another routine that controls the enlarger "ON" time.

Description: External components consist of a capacitor and the resistor or CdS cell under measurement. The resistor is connected between V_{DD} (+5 VDC) and PIA pin CB2 (pin 4 of P2). A capacitor (.0047 μF in the author's case) is connected between CB2 and ground. Measurement action is as follows:

1. Drain change from capacitor by setting CB2 to output low.
2. Zero 16 bit counter.
3. Set CB2 to high impedance input and start counter.
4. Increment counter while polling CB2 for capacitor voltage to exceed its threshold (2V).
5. If counter overflows, the program jumps to "BLINK" routine. Otherwise, the display will contain the counter MSB unless it is zero, in which case the LSB is displayed.

The author's system was initially calibrated using 1K, 10K, 100K, and 1MΩ resistors in place of the photocell. Calibration indicated that the displayed counter had a relation of 1 count per 2KΩ of resistance. This theoretically gives the system an upper range of about 130MΩ and a lower bound of 1KΩ. However, noise on the cell leads and input leakage on CB2 lowers the effective range to something less. When placed in a *very* dark room, the system will overflow and go into the BLINK routine, indicating that low light levels are at least measurable if not linearly related.

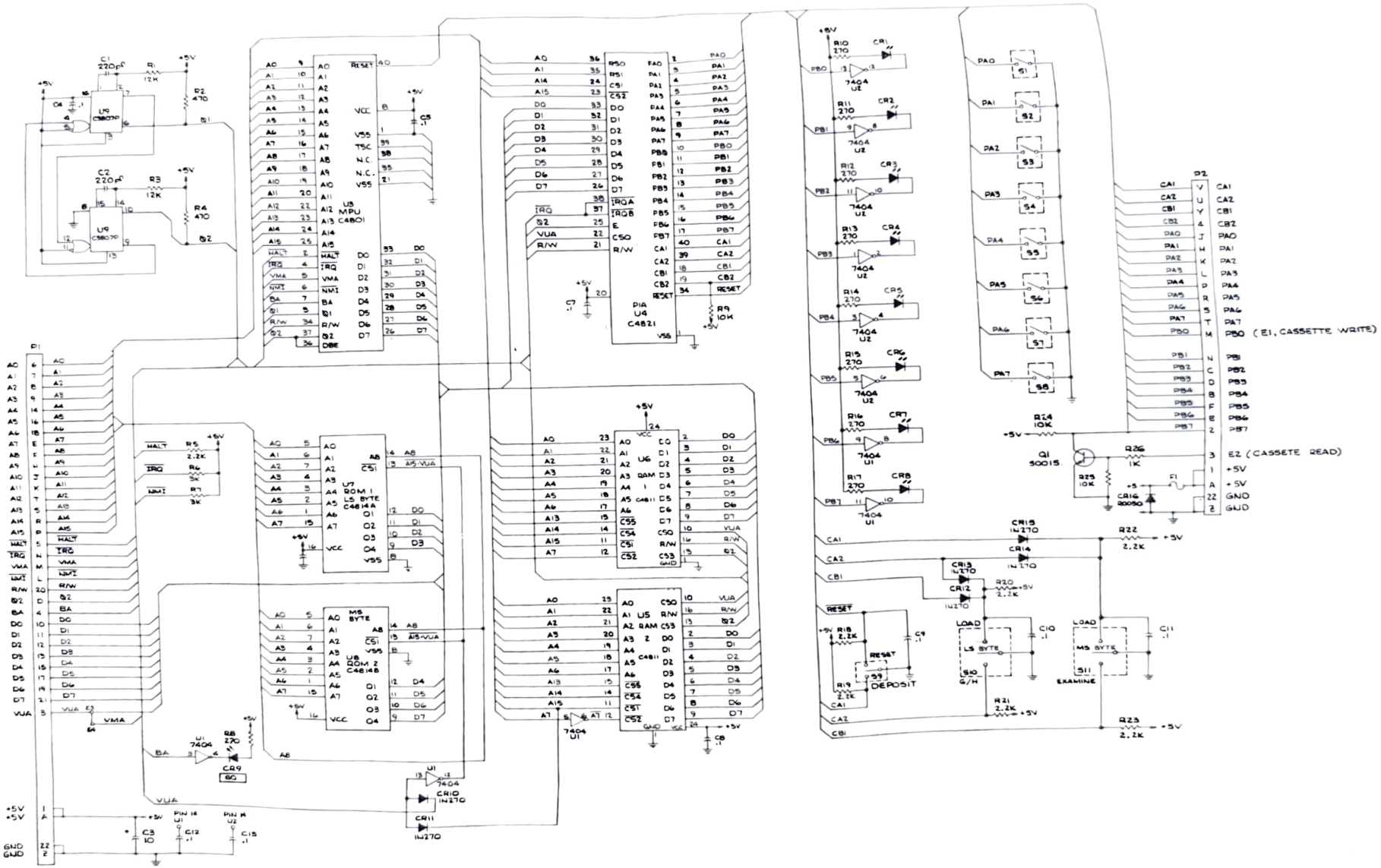
There was good correlation between the MPU system measurement and a good quality DVM under working conditions. Good enlargements were made on POLYCONTRAST R-C paper with PC-3 filter when the *diffused* light measurement resulted in an 06_{16} reading in the display. Since this was the MSB of the counter and the decimal equivalent is about 1500, the cell value is about 3MΩ. Exposure time for proper density of the pictures was 10 seconds.

This routine cannot be HALTed without going to BLINK occasionally when reactivated. If you wish to stop the conversion, use RESET. A succeeding G/H will then start the program at its beginning again.

PROGRAM R-MEASURE

	MNEMONIC		MACHINE CODE		ADR
START	LDAA	#$34	86	34	00
	STAA	$4003	37	4003	02
	LDAA	#$FF	86	FF	05
DEC	DEC A		4A		07
	BNE	DEC	26	FD	08
	LDAA	#$14	86	14	0A
	STAA	$4003	B7	4003	0C
	LDX	0000	CE	0000	0F
INCR	INX		08		12
	BEQ	OVRFLW	27	16	13
	LDAA	$4003	B6	4003	15
	ROLA		49		18
	BPL	INC	2A	F7	19
	STX	$F1	DF	F1	1B
	LDAA	$F1	96	F1	1D
	BNE	RETURN	26	02	1F
	LDAA	$F2	96	F2	21
RETURN	STAA	$4002	B7	4002	23
	TST	$4002	7D	4002	26
	BRA	START	20	D5	29
OVRFLW	JMP	BLINK	7E	8278	2B

Appendix A
Educator II
Schematic Diagram



1. RESISTANCE VALUES ARE IN OHMS, CAPACITANCE VALUES ARE IN MICROFARADS. RESISTORS ARE 1/4 WATT.

NOTES UNLESS OTHERWISE SPECIFIED:

Appendix B
Educator II
Resident Firmware
Program

```
00001                    NAM     DBUG
00002                *2/2/77     VER. 1.0
00003                *
00004                        OPT     O,S
00005                *
00006                ***********************************
00007                *    RAM SPACE ALLOCATION
00008                *
00009                *    FLAG - GO/HALT STATUS
00010                *
00011                *    PANADR - TWO BYTES FOR PANEL ADR
00012                *
00013                *    STACK1 - STORAGE FOR STACK POINTER
00014                *             ON ENTRY TO "HALT".  POINTS
00015                *             TO STACK USED FOR GO RTI
0001C                *
00017                *
00018                *    STACK2 - "BOTTOM" OF STACK AREA USED IN
00019                *             GO/HALT OPERATIONS.  TARGET
00020                *             PROGRAM START VECTOR WILL BE
00021                *             AT FA (PCL) AND F9 (PCH)
00022                *
00023                ***********************************
00024                *
00025 00F0                       ORG     $F0
00026                *
00027 00F0 0002      BEGADR RMB     2
00028 00F2 0001      LAST   RMB     1
00029 00F3 0001      STACK2 RMB     1
00030 00F4 0001      CCR    RMB     1
00031 00F5 0001      ACCB   RMB     1
00032 00F6 0001      ACCA   RMB     1
00033 00F7 0001      IXH    RMB     1
00034 00F8 0001      IXL    RMB     1
00035 00F9 0001      PRGHI  RMB     1
00036 00FA 0001      PRGLO  RMB     1
00037 00FB 0002      STACK1 RMB     2
00038 00FD 0002      PANADR RMB     2
00039 00FF 0001      FLAG   RMB     1
00040                *
00041                *
00042                *
00043          00F6  BYTE   EQU     ACCA
00044          4000  DRA    EQU     $4000
00045          4002  DRB    EQU     DRA+2
00046          4001  CRA    EQU     DRA+1
00047          4003  CRB    EQU     DRB+1
00048          2200  STACK  EQU     $2200
00049                *
00050                *
00051 8200                       ORG     $8200
00052                *
00053                *
00054                *    SUBROUTINE DEPOSIT     *
```

```
00055                    *
00056                    *
00057 8200 E6 00    DEP     LDA B  0,X       CLR INT FLAG IN PIA
00058 8202 85 02            BIT A  #02       DISPLAY DATA IF ZERO; ADDRESS
00059 8204 26 15            BNE    ADR
00060 8206 DE FD    DATA1   LDX    PANADR    LOAD IX WITH PANEL ADR
00061 8208 E7 00            STA B  0,X       PLACE PANEL DATA INTO MEMORY
00062 820A E1 00            CMP B  0,X
00063 820C 26 6A            BNE    BLINK
00064 820E F7 4002          STA B  DRB       PLACE DATA INTO DISPLAY
00065 8211 08               INX
00066 8212 DF FD            STX    PANADR    INCREMENT AND STORE PANEL ADD
00067 8214 86 0F            LDA A  #$0F      SET INT FOR POS EDGE
00068 8216 B7 4001  STORA   STA A  CRA
00069 8219 20 24            BRA    BACK1
00070 821B D6 FE    ADR     LDA B  PANADR+1
00071 821D E7 02            STA B  2,X       PUT LSBYTE OF ADR INTO DISP.
00072 821F 86 0D            LDA A  #$0D      SETS INT FOR NEG EDGE
00073 8221 20 F3            BRA    STORA
00074                    *
00075                    *
00076                    *****   SUBROUTINE EXAMINE     ;*****
00077                    *
00078                    *
00079                    *
00080 8223 DE FD    EXAM    LDX    PANADR    GET PANEL ADR FOR DATA FETCH
00081 8225 C5 02            BIT B  #$2
00082 8227 27 09            BEQ    ADR2      NEG EDGE=ADR IN DISPLAY
00083 8229 A6 00    DATA2   LDA A  0,X
00084 822B B7 4002          STA A  DRB       PUT OLD DATA INTO DISPLAY
00085 822E C6 35            LDA B  #$35
00086 8230 20 0A            BRA    STORB
00087 8232 08       ADR2    INX              INCREMENT PANEL ADR
00088 8233 DF FD            STX    PANADR    STORE NEW PANEL ADR
00089 8235 96 FE            LDA A  PANADR+1  GET LSBYTE OF PANEL ADR
00090 8237 B7 4002          STA A  DRB       LSBYTE OF PANEL ADR INTO DISP
00091 823A C6 37            LDA B  #$37
00092 823C F7 4003  STORB   STA B  CRB
00093 823F 20 68    BACK1   BRA    BACK
00094                    *
00095                    *
00096                    *********  IRQ ROUTINE **********
00097                    *
00098                    *      CHECK FOR CB2 INTERRUPT *****
00099                    *
00100                    *
00101       8241    IRQ     EQU    *
00102 8241 F6 4003          LDA B  CRB
00103 8244 C5 40            BIT B  #$40
00104 8246 27 03            BEQ    DELAY
00105 8248 7E 0080          JMP    $80
00106                    *
00107                    *
00108                    ********  DEBOUNCE ROUTINE *******
```

```
00109                    *
00110                    *
00111 824B CE 0960 DELAY  LDX     #2400     SET DEBOUNCE TIME
00112 824E 09      D2     DEX
00113 824F 26 FD          BNE     D2
00114 8251 CE 4000        LDX     #DRA      SET UP IX FOR PIA ACCESSES
00115                    *
00116                    *
00117                    *  IRQ SERVICE CONTINUED **
00118                    *
00119                    *
00120 8254 E6 03          LDA B   3,X
00121 8256 A6 01          LDA A   1,X
00122 8258 2B 06          BMI     D1        CA1 = DEPOSIT
00123 825A 85 40          BIT A   #$40      NOT CA1, TEST CA2
00124 825C 26 08          BNE     D3        IF CA2, TEST CB1
00125 825E 20 C3          BRA     EXAM
00126 8260 85 40    D1    BIT A   #$40
00127 8262 26 58          BNE     MSBYTE
00128 8264 20 9A          BRA     DEP       CA1 AND NOT CA2
00129 8266 C5 80    D3    BIT B   #$80
00130 8268 26 58          BNE     LSBYTE
00131 826A 20 18          BRA     GOHALT
00132                    *
00133                    *
00134                    *
00135                    *
00136                    *
00137                    *
00138                    *    ENTRY FOR RAM DUMP  $00 - $F0
00139                    *
00140                    *
00141                    *
00142 826C CE 0000        LDX     #0
00143 826F DF F0          STX     $F0
00144 8271 CE 00F0        LDX     #$F0
00145 8274 DF F2          STX     $F2
00146 8276 20 57          BRA     TAPOUT
00147                    *
00148                    *
00149                    *
00150                    ***** BLINKING DISPLAY ROUTINE ********
00151                    *
00152                    *
00153                    *
00154 8278 CE C1A8 BLINK  LDX     #25000
00155 827B 0E             CLI
00156 827C 73 4002        COM     DRB
00157 827F 09      LOOP   DEX
00158 8280 26 FD          BNE     LOOP
00159 8282 20 F4          BRA     BLINK
00160                    *
00161                    *
00162                    *
```

```
00163                    *       GO/HALT  SUBROUTINE    *
00164                    *
00165                    *
00166 8284 85 10    GOHALT BIT A  #$10      MASK FOR EDGE BIT
00167 8286 27 0B           BEQ    OK        IF ZERO, NEG EDGE
00168 8288 86 0D           LDA A  #$0D
00169 828A A7 01           STA A  1,X
00170 828C 6D 00           TST    0,X
00171 828E 96 FF           LDA A  FLAG
00172 8290 27 17           BEQ    BACK
00173 8292 3B              RTI
00174 8293 86 1D    OK     LDA A  #$1D      SET FOR POS EDGE
00175 8295 A7 01           STA A  1,X
00176 8297 96 FF           LDA A  FLAG
00177 8299 27 19           BEQ    GO        ZERO=GO; FF=HALT
00178 829B 30       HALT   TSX
00179 829C A6 06           LDA A  6,X       FETCH PCL FROM STACK
00180 829E 4A              DEC A
00181 829F B7 4002  HALT3  STA A  DRB       PUT PCL-1 INTO DISPLAY
00182 82A2 9F FD    HALT2  STS    PANADR    STORE STACK AT PANEL ADR
00183 82A4 9F FB           STS    STACK1    STORE USER'S STACK POINTER
00184 82A6 7F 00FF         CLR    FLAG      RESET FLAG FOR "GO"
00185 82A9 8E 2200  BACK   LDS    #STACK    RESTORE PANEL PHANTOM STACK
00186 82AC 7D 4002         TST    DRB       CLR INT
00187 82AF 7D 4000         TST    DRA
00188 82B2 0E              CLI              READY FOR GO
00189 82B3 3E              WAI
00190 82B4 9E FB    GO     LDS    STACK1    RESTORE USER'S STACK POINTER
00191 82B6 73 00FF         COM    FLAG      SET FLG FOR HALT
00192 82B9 6D 00           TST    0,X       CLR INT FLG
00193 82BB 3B              RTI              LOADS "OLD" STACK INTO PROCSR
00194                    *
00195                    *
00196                    ******  SUBROUTINE M S BYTE  ********
00197                    *
00198                    *
00199 82BC A6 00    MSBYTE LDA A  0,X
00200 82BE 97 FD           STA A  PANADR
00201 82C0 20 04           BRA    FETCH
00202                    *
00203                    *
00204                    *     SUBROUTINE LOAD LSBYTE    *
00205                    *
00206                    *
00207 82C2 A6 00    LSBYTE LDA A  0,X       LOAD SWITCH INFO INTO ACCA
00208 82C4 97 FF           STA A  PANADR+1   STORE LSBYTE INTO PANEL ADR
00209 82C6 DE FD    FETCH  LDX    PANADR    GET DATA AT PANEL ADR
00210 82C8 A6 00           LDA A  0,X
00211 82CA B7 4002         STA A  DRB       DISPLAY DATA IN PANEL ADR
00212 82CD 20 DA           BRA    BACK
00213                    *
00214                    *
00215                    *
00216 82CF 8E 00FF  TAPOUT LDS    #$FF
```

```
00217 82D2 8D 2C          BSR    INITIA
00218                *
00219                *
00220 82D4 4F       START  CLR A
00221 82D5 97 F9          STA A  PRGHI
00222 82D7 5F       START1 CLR B
00223 82D8 D7 FF          STA B  FLAG
00224 82DA D7 F4          STA B  CCR
00225 82DC 8E 00F3        LDS    #STACK2
00226 82DF 7F 4001  INZ   CLR    CRA
00227 82E2 7F 4003        CLR    CRB
00228 82E5 CE 000D        LDX    #$000D
00229 82E8 FF 4000        STX    DRA
00230 82EB CE FF35        LDX    #$FF35
00231 82EE FF 4002        STX    DRB
00232                *
00233 82F1 20 AC          BRA    HALT3
00234                *
00235                *
00236                * CASSETTE INPUT ENTRY FOR FILE  *
00237                * NUMBER IN SWITCHES  *
00238                *
00239 82F3 B6 4000 TAPIN1 LDA A  DRA
00240 82F6 40             NEG A
00241 82F7 97 FF          STA A  FLAG
00242                *
00243                * CASSETTE INPUT ENTRY FOR NEXT FILE *
00244                *
00245 82F9 8E 00FE        LDS    #$FE
00246 82FC 8D 63          BSR    TAPE2
00247 82FE 20 D7          BRA    START1
00248                *
00249                *
00250                *
00251                *
00252                *
00253                *
00254                *
00255                *
00256                * *** CASSETTE INTERFACE PROGRAM
00257                *
00258                *
00259                *   OUTPUT TO TAPE ROUTINE:     *
00260                *
00261                *   START VECTOR = 82CF (F9,FA)  *
00262                *   PUT START DUMP ADR AT F0,F1  *
00263                *  END DUMP ADR AT F2,F3    *
00264                *  START TAPE; DEPRESS G/H  *
00265                *  01 IN DISPLAY = END OF TRANSMISSION *
00266                *
00267                *
00268                *
00269                *
00270                *
```

```
00271                *
00272                *
00273                *   INPUT FROM TAPE ROUTINE:        *
00274                *
00275                *   START VECTOR = 82F9
00276                *   DEPRESS G/H; START TAPE     *
00277                *
00278                *   PROGRAM STATUS IS IN DISPLAY:     *
00279                *   $80 = PPROGRAM RUNNING  *
00280                *   $81 = TAPE LEADER TONE DETECTED *
00281                *   $02 = NO MEMORY AT LOAD ADR  *
00282                *   $03 = SYNC ERROR      *
00283                *   $04 = TONE TRAILER DETECTED    *
00284                *
00285                **** OUTPUT TO TAPE ROUTINE  *****
00286                *
```

Appendix C
Replacement Parts List

Appendix C
Replacement Parts List

SCHEMATIC DESIGNATION	STOCK NUMBER	DESCRIPTION
R1, R3	06ASH0001A-123	12KΩ ¼ watt Resistor
R2, R4	06ASH0001A-471	470Ω ¼ watt Resistor
R5, R18 thru R23	06ASH0001A-222	2.2KΩ ¼ watt Resistor
R6, R7	06ASH0001A-302	3KΩ ¼ watt Resistor
R8, R10 thru R17	06ASH0001A-271	270Ω ¼ watt Resistor
R9, R24, R25	06ASH0001A-103	10KΩ ¼ watt Resistor
R26	06ASH0001A-102	1KΩ ¼ watt Resistor
C1, C2	21BSH0001A-221	220pf Dipped Mica Capacitor
C3	21BSH0003A-100	10μf 15v Electrolytic Capacitor
C4 thru C13	21BSH0002A-001	.1μf Ceramic Disc Capacitor
U1, U2	48ASH7404C-001	7404 Hex Inverter
U3	48ASH4801C-000	MPU HEPC4801
U4	48ASH4821C-000	PIA HEPC4821
U5, U6	48ASH4811C-000	RAM HEPC4811
U7	48ASH4814C-002	MS Byte RAM HEPC4814B
U8	48ASH4814C-001	LS Byte RAM HEPC4814A
U9	48ASH3807C-000	Clock Generator HEPC3807
Q1	48ASH0015S-000	NPN Transistor HEPS0015
CR1 thru CR9	48ASH2005P-000	LEDs
CR10 thru CR15	48ASH2701N-000	1N270 Germanium Diode
CR16	48ASH0050R-000	Silicon Diode HEPR0050
S1 thru S8	40BSH0001A-000	SPDT Switch
S9, S10, S11	40BSH0002A-000	SPDT Switch — Momentary
J1, J2	28BSH0001A-000	Miniature Phone Jack
F1	80ASH0001-000	1.5 Amp Fuse
	09ASH0001A-000	40 Pin I/C Socket
	09ASH0002A-000	24 Pin I/C Socket
	09ASH0003A-000	16 Pin I/C Socket
	84DSH0001A-000	Printed Circuit Board
	42ASH0001A-001	Mounting Ring for LED
	42ASH0001A-002	Mounting Bushing for LED
	77ASH0001A-000	Rubber Feet
	05ASH0001A-000	Power Cord Strain Relief Bushing
	16CSH0001A-001	Case Top
	16CSH0001A-002	Case Bottom

LIMITED WARRANTY

Motorola Inc. warrants the "kit" to be free from defects in material and workmanship and that it will conform to applicable specifications. This warranty is for a period of ninety (90) days after the date the "kit" is delivered to the original consumer only.

In the event of a defect in material or workmanship and/or nonconformity with applicable specifications during the ninety (90) day period, Motorola Inc., at its option, will either replace or repair the "kits." "Kits" thought to be defective must be returned, by mail, to Motorola Inc., HEP/MRO Operations, 705 W. 22nd Street, Tempe, Arizona 85282, along with a check or money order in the amount of $10.00 payable to Motorola Inc. to cover handling and inspection. In the event inspection reveals that the "kit" is defective or nonconforming, then Motorola Inc. shall refund the $10.00. Return postage to the consumer shall be paid by Motorola Inc.

This warranty is void if:
(a) The "kit" has been used for other than its normal and customary purpose.
(b) The "kit" has been subject to misuse, accident, neglect or damage; or
(c) Solder, other than resin core, including but not limited to acid core solder or paste are used in the assembly of the "kit."

The implied warranties, which the law imposes on the sale of this product, are expressly limited to the terms of the limited warranty extended herein. Limitations on the period of an implied warranty are not permitted in some states in which event the limitation set forth may be inapplicable.

IN NO EVENT SHALL MOTOROLA INC. BE LIABLE FOR ANY INCIDENTAL OR CONSEQUENTIAL DAMAGES FOR NONCONFORMITY, FAILURE OR DEFECT IN THE 'KIT." Motorola Inc. disclaims such liability to the full extent permitted by law. Some states do not allow the exclusion or limitation of incidental or consequential damages, so the above limitation may not apply to you.

This warranty gives you specific legal rights, and you may also have other rights which vary from state to state.

Completion of the warranty registration card enclosed with the "kit" and mailing same to Motorola Inc. at the address set forth herein is mandatory in order to obtain coverage under this warranty.

Motorola HEP/MRO Operations • 705 West 22nd Street • Tempe, Arizona 85282 • (602) 244-3208