



8700
COMPUTER/CONTROLLER
ASSEMBLY
AND
USING MANUAL

8700 ASSEMBLY

The PAIA 8700 COMPUTER/CONTROLLER is assembled on the double-sided, plated through-hole, etched circuit board provided. Unlike other PAIA circuit boards, this board has all conductive traces pre-tinned for easy solderability and does not require scrubbing before assembly.

Also unlike many other PAIA circuit boards, the 8700 board is complex; and on complex boards, unintended conducting paths between conductors (particularly where a conductor passes between pins on an IC) are not unheard of. While all reasonable quality control precautions have been taken, it is a wise assembler who will spend several minutes closely examining the circuit board for these unintentional bridges. Prints of the circuit board artwork have been provided for this purpose in figures (1) and (2). Bridges (particularly on the component side of the board) will be particularly difficult to find once sockets and other components are in place.

Because of the close proximity of some conductors to one another, extreme care should be exercised during soldering to prevent unintentional solder bridges during assembly. The likelihood of assembly-caused bridges has been lessened by laying out the board with an absolute minimum number of conductors passing between IC pins on the soldered side of the board, but care is nevertheless advised.

Use a clean, low-wattage iron for soldering (40 watts max.). While most temperature-sensitive components (with the exception of discrete transistors) are mounted in sockets, excessive temperature can weaken or destroy the bond between the conducting copper and the fibre-glass board material.

All sockets and other components are mounted on the side of the board with the silk-screened parts placement designators and soldered from the opposite side ONLY. DO NOT SOLDER COMPONENTS ON BOTH SIDES OF THE BOARD.

NOT ALL HOLES ON THE CIRCUIT BOARD WILL HAVE A PART ASSOCIATED WITH THEM. Many of the holes are conductive pass-throughs from one side of the board to the other while others are holes reserved for mounting optional components. Some manufacturers recommend filling through-board holes with solder to insure that a conductive path is established from one side of the board to the other. If you elect to do this, make sure that you know which holes are which. It is for all practical purposes impossible to mount a component in a plated-through hole that has been filled with solder.

NOT ALL PART NUMBERS ARE USED ON THIS CIRCUIT BOARD, some part numbers (e.g. R4) are reserved for future expansion.

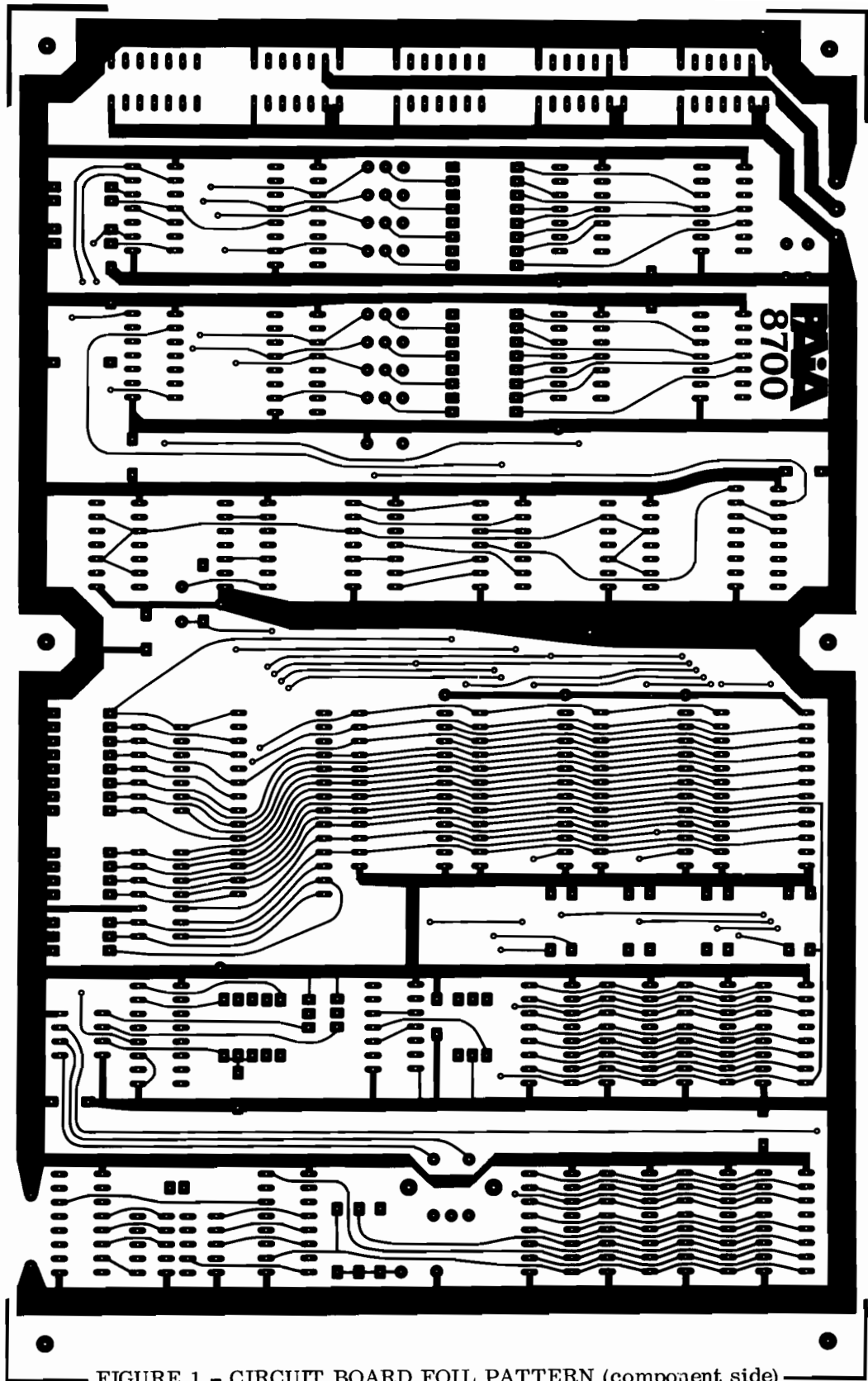


FIGURE 1 - CIRCUIT BOARD FOIL PATTERN (component side)

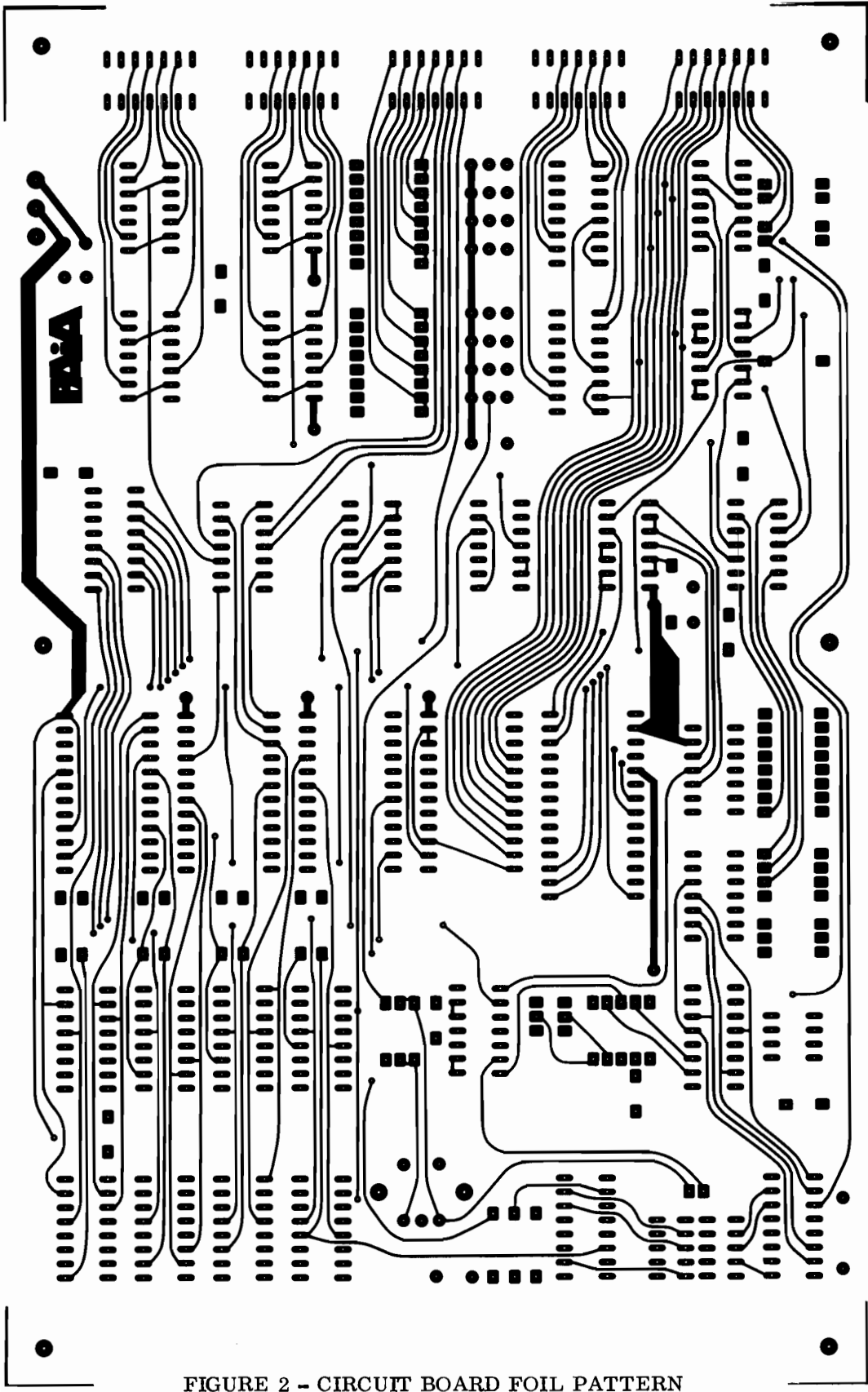


FIGURE 2 - CIRCUIT BOARD FOIL PATTERN

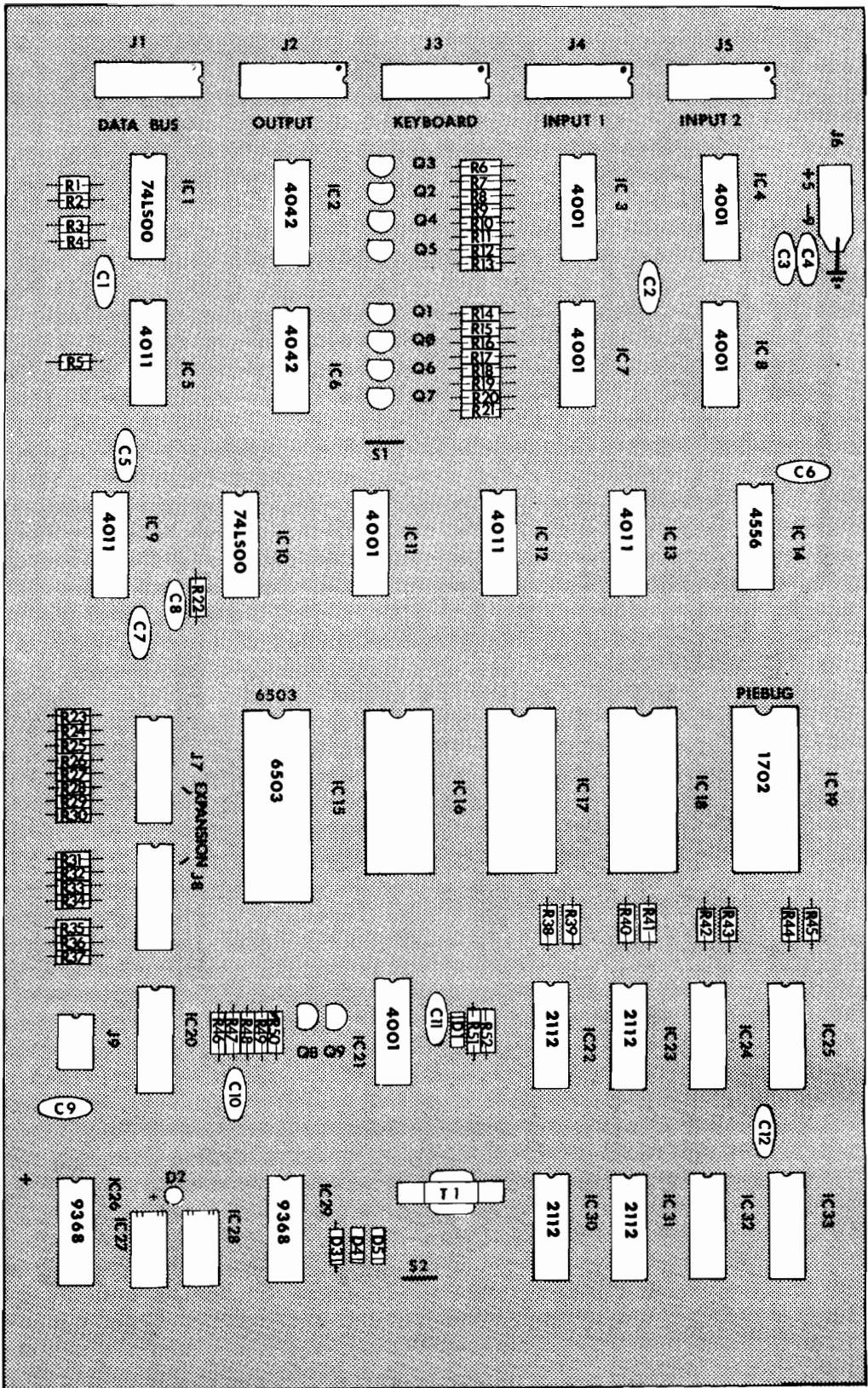
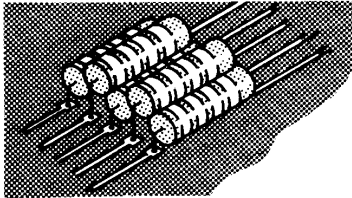


FIGURE 3
8700 COMPUTER/CONTROLLER PARTS PLACEMENT

When mounting components such as resistors, diodes and capacitors, the leads of the part should be passed through the mounting hole and then bent to a slight angle to hold the part in place for soldering. DO NOT "cinch" the leads directly against the board (bend to a 90° angle). This technique while great for the government (and others who are in the habit of throwing away things that don't work) provides only marginal additional mechanical strength and makes removing malfunctioning components extra-ordinarily difficult. AND REMEMBER... pre-tinned boards require very little additional solder.

With all of these DOs and DON'Ts out of the way, we begin:

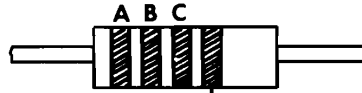
HAVE YOU INSPECTED THE BOARD? It might just save you a lot of trouble.



Using parts placement designators and the parts placement drawing in figure 3 as guides, solder the following resistors in place. Notice that many of these resistors are close together and consequently may need to be "stacked" as shown to the left.

Note that resistors are non-polarized components and that either lead may be placed in either hole without affecting performance.

Installation of all resistors within a given group before any of the resistors in the group are soldered in place is highly recommended.



Silver or gold - disregard this band.

PART NUMBER(s)	VALUE	COLOR CODE A-B-C
() R1 - R3 (3 parts).....	3300 ohms.....	orange-orange-red
() R5.....	3300 ohms.....	orange-orange-red
() R6 - R 13 (8 parts)....	27K.....	red-violet-orange
() R14 - R21 (8 parts)....	27K.....	red-violet-orange
() R22.....	27K.....	red-violet-orange
() R23 - R25 (3 parts)....	27K.....	red-violet-orange
() R26 - R30 (5 parts)....	10K.....	brown-black-orange
() R31 - R37 (7 parts)....	10K.....	brown-black-orange
() R38 - R45 (8 parts)....	10K.....	brown-black-orange

Install the following ceramic disk capacitors. Like resistors, these components are non-polarized and either lead may be installed in either of the holes provided.

- () C1-C7 (7 parts)..... .05 mfd disk
- () C9..... .05 mfd disk
- () C11, C12 (2 parts)..... .05 mfd disk
- () C8..... 33 pfd. disk



Install the Integrated Circuit sockets. Note that four different socket sizes have been supplied; 14 pin, 16 pin, 24 pin, and 28 pin. DO NOT INSTALL ANY OF THE INTEGRATED CIRCUITS AT THIS TIME!

When installing the sockets; note that there is a small notch at one end, between the rows of pins. This notch should correspond to the notch on the circuit board graphics for convenient reference later on.

Install the 14 pin sockets (17 supplied) in the following locations

- J1 J2 J3 J4
- J5 IC1 IC3 IC4
- IC5 IC7 IC8 IC9
- IC10 IC11 IC12 IC13
- IC21

Install the 16 pin sockets (9 supplied) in the following locations

- IC2 IC6 IC14 IC22
- IC23 IC26 IC29 IC30
- IC31

Install the 24 pin socket supplied at the location of IC19

Install the 28 pin socket supplied at the location of IC15

Install the 3 pin power connector at the location indicated as J6. Note that this connector is keyed by the shape of its base and must be installed properly. (see figure on page 27)

Install the 8 discrete transistors. Note that the transistors are keyed by the flat on the side of their cases and must be installed properly for proper operation. Because of later mechanical assemblies, it is also important that the transistors seat as closely as possible to the board. The tops of the transistors should be no more than 3/8" above the surface of the board.

- Q0 Q1 Q2 Q3
- Q4 Q5 Q6 Q7

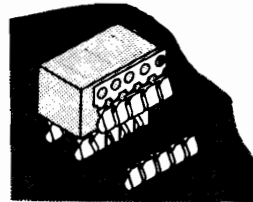
Install the three 1n914 diodes provided. Like the transistors, these parts are polarized and must be installed so that the banded end of the diode corresponds to the band indicated on the circuit board graphics.

- D3 D4 D5

Like the IC's, the seven-segment displays are socketed, but since the pins on the displays are not to standard tolerances, Molex pins must be used to mount these parts. The molex pins are tied together at the top by a metal strip referred to as a "carrier", and to be perfectly correct the carrier should be on the outside of the two strips that will constitute the socket.

The molex pins are supplied in a continuous strip and must be cut into lengths of 5 pins each prior to installation on the circuit board.

Install and solder the four rows of molex pins at the IC27 and IC28 locations. Snap off the carrier strip after the pins are soldered in place.



We are now ready to begin installing the Integrated Circuits, but first a brief explanation of where we're headed. The chances are good that with careful assembly the 8700 Computer/Controller will be ready to operate when power is first applied. Nevertheless, it is a good idea to go through the "power-up" procedure that we will outline. The procedure entails the use of an oscilloscope and should be used by anyone with access to one of these devices.

If you absolutely cannot get a scope to use, you may skip this procedure, but for those who can use it, it will give you confidence in certain sections of the computer and simplify trouble-shooting procedures in the event that there is a failure when the unit is fully assembled.

Open the integrated circuit package and install the following integrated circuits in their respective sockets. Notice that the orientation of the ICs is keyed by a semi-circular notch at one end of the device, and that the position of this notch should correspond with the notch that is part of the circuit board graphics.

WARNING CMOS CIRCUITS

Some of the integrated circuits used in this kit are Complementary Metallic Oxide Semiconductors (CMOS). While state of the art internal protection is provided, these circuits are still susceptible to damage from STATIC ELECTRICITY. You should not experience any difficulties if you observe the following precautions.

- 1) The circuits are supplied to you inserted in blocks of conductive foam. Leave them in these blocks until you are ready to install the part.
- 2) Do not install the parts in sequence other than that called for in the instructions.
- 3) Do not wear synthetic (e.g. nylon) clothing while handling these parts.

Install the following ICs in their sockets. NOTE: FND 357 displays are keyed by a series of small grooves on their top edge.

<input type="checkbox"/> IC1 74LS00	<input type="checkbox"/> IC2 4042	<input type="checkbox"/> IC3 4001
<input type="checkbox"/> IC4 4001	<input type="checkbox"/> IC5 4011	<input type="checkbox"/> IC6 4042
<input type="checkbox"/> IC7 4001	<input type="checkbox"/> IC8 4001	<input type="checkbox"/> IC9 4011
<input type="checkbox"/> IC10 74LS02	<input type="checkbox"/> IC11 4001	<input type="checkbox"/> IC12 4011
<input type="checkbox"/> IC13 4011	<input type="checkbox"/> IC14 4556	<input type="checkbox"/> IC15 6503
<input type="checkbox"/> IC21 4001	<input type="checkbox"/> IC26 9368	<input type="checkbox"/> IC29 9368
<input type="checkbox"/> IC27 FND 357 Display	<input type="checkbox"/> IC28 FND 357	

This should leave you with 5 ICs that have not been installed; four 2112 RAMs and one 1702A PIEBUG monitor PROM.

Using a section of resistor clipping, form and install the jumper indicated as S2 on the parts placement diagram. Leave a generous loop in this jumper as it will be cut open later.

The jumper that was installed above enables a test feature of the 8700, described in the "Self Test" section of this manual, you should at this point skip to that section and perform the tests outlined there. Return to this point for final assembly when the procedure outlined has been completed.

() Clip the jumper installed as S2 in a previous step into two sections and spread the sections apart so they do not touch, but so that they may be re-soldered if needed.

() Using a section of excess resistor lead, form and install the jumper indicated as S1 on the circuit board graphics. (This jumper enables "normal" operation of the system, and must be in place for the unit to function properly.

() Install the remaining Integrated Circuits in their respective sockets (observe orientation markings).

() IC22 2112 () IC23 2112 () IC30 2112
() IC31 2112 () IC19 1702A PROM

This completes assembly of the 8700 CPU board. Proceed to assembly of the 8700A active keyboard.

8700/A KEYBOARD ASSEMBLY

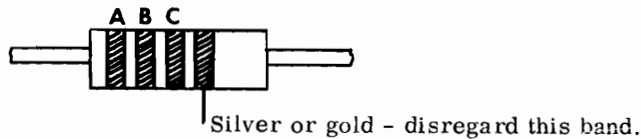
Prepare for assembly by thoroughly cleaning the exposed copper circuitry above colored keyboard area. Use steel wool and/or scouring cleanser. DO NOT USE PRE-SOAPED STEEL WOOL PAD. Use particular care to avoid scratching the printed keyboard area. Rinse and dry the board completely before beginning assembly.

A WORD OF ADVICE - Do not clean the circuit board until you are ready to assemble and test this unit. When assembly is complete and the unit verified as being operational, a coat of artist's spray fixative (available at most artist's/ engineers supply stores; e.g. "Blair Spray Fix") will keep the copper bright and shiny and prevent oxidation.

DO NOT try to protect the copper with any oil-based sprays as these may entrain moisture or otherwise become conductive.

NOTE that there are no sockets used in the 8700/A.

And finally, just so there is no confusion, the parts are mounted on the side of the board marked "IC1", "R1", etc.



Begin assembly by soldering all resistors in place as per the parts placement designators printed on the circuit board and the detail figure 4.

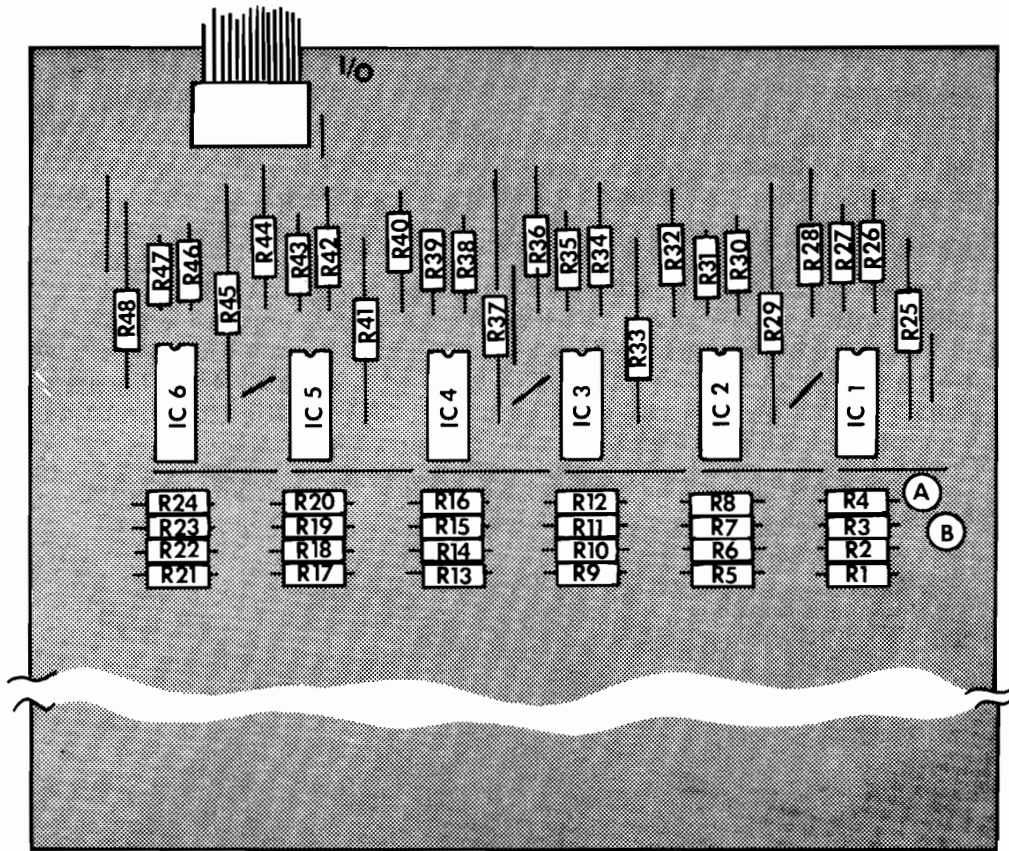


Figure 4

DESIGNATION	VALUE	COLOR CODE A-B-C
() R1 - R16 (16 resistors)	82K	grey-red-orange
() R17 - R20 (4 resistors)	100K	brown-black-yellow
() R21 - R24 (4 resistors)	150K	brown-green-yellow
() R25-R48 (24 resistors)	27K	red-violet-orange

There are 13 solid wire jumpers used on this board. Using the solid wire provided, form and install these jumpers.

- () Form and install 13 jumpers. Count them.
- () Locate the RESET push-button (S1) and prepare it for installation by using a pair of needle-nose pliers to carefully bend its two solder lugs out to 90° angles as shown in detail figure 5.
- () Cut the length of insulated wire provided into two equal 5-inch lengths, strip 1/4 inch of insulation from each end of each wire and twist and tin the exposed ends. Solder one end of each of the lengths to the lugs of the switch.

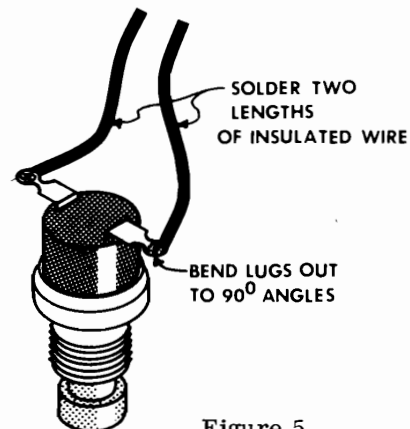


Figure 5

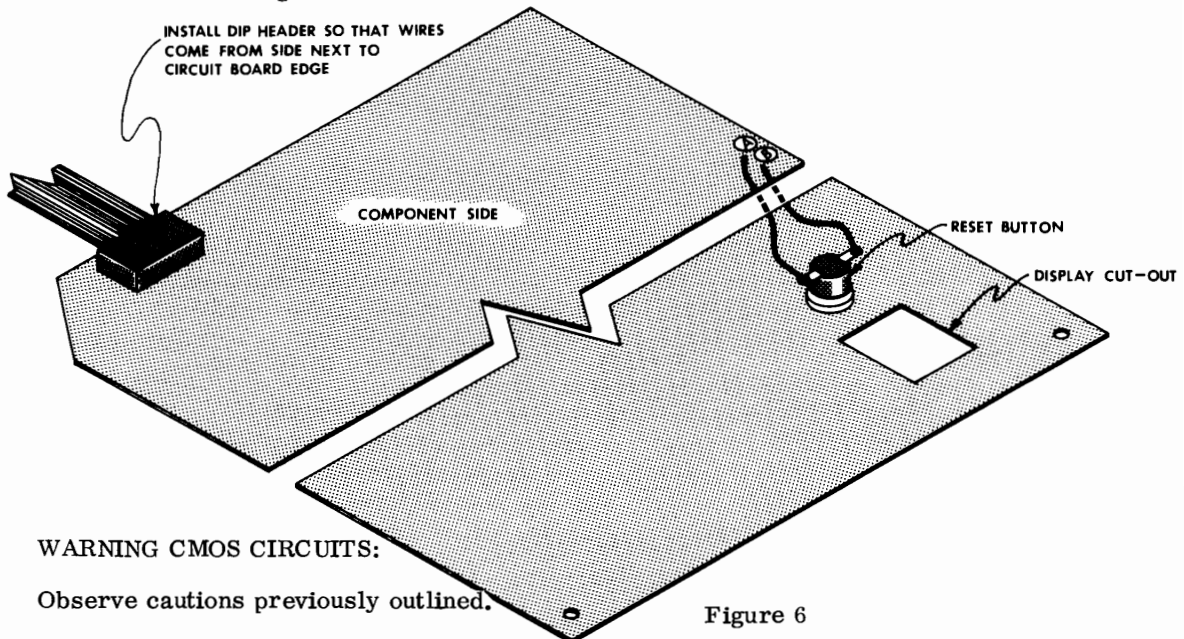
() Using the hardware supplied, mount the RESET button in the circular hole directly above the rectangular display cut-out. NOTE that the pushbutton mounts from the component side of the board so that the actuating stud protrudes from the side of the board printed with the keyboard designations.

() Solder one of the two wires connected to the RESET button to the circuit board point labeled "A" and the other to the circuit board point labeled "B".

Install the 14 lead, DIP header terminated I/O connector as follows:

() From the component side of the board, push the 14 pins of the keyboard I/O cable header (either end may be used) into the 14 holes provided at the circuit board location marked "I/O". While either end of the jumper may be used here, the header MUST be installed so that the wires coming from it point TOWARDS THE NEAREST EDGE OF THE CIRCUIT BOARD as shown in detail figure 6.

() Carefully solder all 14 pins of the header in place. Excessive heat at this operation can melt the header. Make sure that the copper is very clean before soldering.



WARNING CMOS CIRCUITS:

Observe cautions previously outlined.

Figure 6

A three-wire grounded soldering iron is ideal but if you don't have one, your present iron may be used by allowing it to heat, then UNPLUGGING it during the soldering operation. Before soldering and after unplugging touch the tip of the iron momentarily to the ground screw of an electrical outlet or other source of ground to drain the static charges.

Install the six 4001 CMOS NOR gate packages IC-1 through IC-6.

DESIGNATION	TYPE
() IC-1 to IC-6	CA4001B

THIS COMPLETES ASSEMBLY OF THE PAIA 8700/A KEYBOARD.

FINAL ASSEMBLY-

() Using the hardware illustrated, mount the 8700A active keyboard above the 8700 CPU board. Note that two 5/16" spacers are used on each of the 1" machine screws that hold the keyboard above the processor, and that the displays are visible through the rectangular cut-out above the RESET switch.

ALSO check that the solder lugs on the RESET switch (S3 on the 8700A) do not contact any of the components on the CPU board. If necessary, loosen the switch and re-orient.

() Using the hardware illustrated, mount the two remaining rubber feet at the rear edge of the 8700 board.

() Mate the 14 pin header of the keyboard I/O cable with the 14 pin socket J3 (the middle socket of the five along the rear edge of the CPU board).

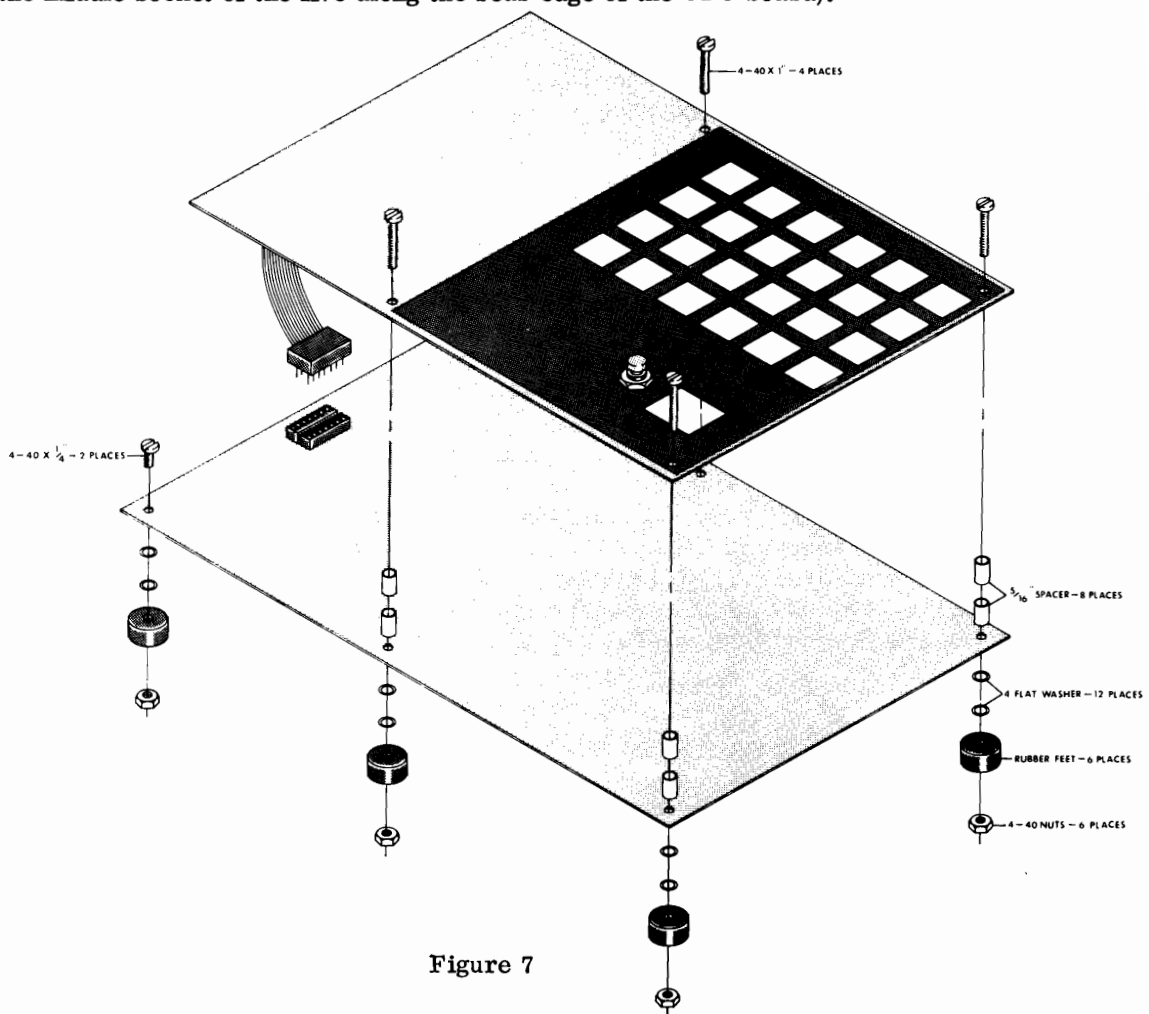


Figure 7

THIS COMPLETES ASSEMBLY OF THE PAIA 8700 COMPUTER/CONTROLLER.
Check out the system using the Testing and Preliminary Familiarization section which follows.

TESTING & FAMILIARIZATION

THE PAIA MONITOR (PIEBUG)

Now that you have your computer assembled the next step is obviously to try it out. To do that you will have to know a little bit about the monitor program. We will assume that you know little or nothing about computers and attempt to explain why there is a monitor program in the first place.

You can think of your computer as a machine that follows your instructions to the letter. That's really all that any computer is. The group of instructions you give it to do a specific job is called a program. A person that writes a set of instructions (program) for a computer is commonly called a "computer programmer". There are lots of computers in the world, consequently there are lots of computer programmers. You are about to become one!

In general, a computer by itself is useless. There is no way to feed instructions into it or get results out of it. Although it has the ability to follow your directions it must rely on external equipment or devices for input and output operations. The external equipment and devices fall into a category known as "peripherals" and include such things as printers, CRT terminals, teletype terminals, tape drives, card readers, and so on. On small computers you may find peripherals such as cassette recorders, A/D and D/A converters, relays to control external events, etc.

The PAIA 8700 Computer/Controller has two peripherals that come with it; a keyboard and display. The keyboard has 24 "touch-pad" keys. Each key is activated by simply touching it with your finger, there is no key movement. If you have the CS-87 Cassette option each keystroke is accompanied by the muted "beep" of the audible feed-back circuitry. Eight of the keys are for control functions while the other sixteen represent the hexadecimal number set. Hexadecimal is a number set that fits computers very well but contains sixteen symbols instead of ten like you are used to working with now. The symbols used in the hex (short for hexadecimal) set are 0 through 9 and A through F (i. e. , 0 1 2 3 4 5 6 7 8 9 A B C D E F). If you don't know hex it will be fairly easy for you to learn since you are already familiar with all the symbols.

Obviously the purpose of the keyboard/display is to get programs and information into and out of your PAIA computer. However, to do this task the computer must have the instructions (program) to tell it how to perform. That is the purpose of the monitor program. It instructs the computer on how to interpret the information from the keyboard and what information is to be sent to the display. The basic use of the monitor is in loading and examining the contents of memory using the keyboard and display. That gives you the ability to enter a program into the computer from the keyboard, try it out, and change it if necessary. The Monitor will perform other functions to aid you in your feat of using the computer and those functions will be explained as you read on.

ENTERING A PROGRAM INTO THE COMPUTER

The following is a sample program that we will use as an example:

ADDR	CODE	LABEL	INSTRUCTION	COMMENTS
0000	A9 00	BEGIN	LDA #0	;CLEAR ACCUMULATOR
0002	8D 20 08	REPEAT	STA \$0820	;DISPLAY ACC
0005	A0 00		LDY #0	;CLR Y
0007	A2 50		LDX #\$50	;SPEED SETTING (IN HEX)
0009	C8	LOOP	INY	;DELAY LOOP
000A	D0 FD		BNE LOOP	;BRANCH UNTIL Y=0
000C	CA		DEX	;CHECK SPEED
000D	D0 FA		BNE LOOP	;BRANCH UNTIL X=0
000F	F8		SED	;SET DECIMAL MODE
0010	18		CLC	;CLR CARRY
0011	69 01		ADC #1	;ADD 1 TO ACC
0013	4C 02 00		JMP REPEAT	;DO IT ALL AGAIN

Fig 1.

This program will make your computer count from 0 to 99 and then start over. You will be able to see it count by watching the display.

You will notice that the format of this program listing is divided into five "fields"; ADDR, CODE, LABEL, INSTRUCTION and COMMENT. Each of these fields has its own significance.

The ADDR column is the "address" in memory (more on this shortly) of the data or instruction.

The CODE column is the actual "machine language" which will be stored at the memory location specified by the ADDR field. The first two digits of the CODE field are referred to as the OP-CODE, this is the part of the code field that tells the computer which instruction, among its repertoire of many dozen, it is to execute. The pairs of digits following the op-code are called the OPERAND and in general this part of the CODE field tells the computer where and how to execute the instruction specified by the op-code. Notice that some op-codes have one pair of digits for the operand while others have two pair or none at all.

In general, a computer executes instructions in a linear manner; doing one, then the next in line, then the next, etc; but, there will be times when a program will "loop"; that is, repeat a given section of the program a number of times to obtain the required result. For the convenience of the programmer (this is not entered into the machine) the LABEL field is provided for naming specific locations or parts of the program that are to be "jumped" to out of their normal sequence. For example, the last instruction in our demo program is `JuMP REPEAT`, which means that when the computer executes this instruction it will jump back to the portion of the program marked as REPEAT in the label field (in this case, at location 0002) and continue running the program from that point on.

The INSTRUCTION field, like the LABEL field is provided as an assistance to the programmer. It is difficult (at least) to remember all of the op-codes in

the computer's repertoire, and the INSTRUCTION field provides space for a mnemonic (pronounced ne'-mon-ic - a memory aid) for the instruction that the computer is to execute. Some programmers may be able to look at the op-code A9 and remember that it is the instruction for loading the accumulator in the immediate mode, but LDA #0 (LoaD Accumulator; #, an almost universal symbol for "immediate"; and 0, the thing to be entered in the accumulator) is a whole lot easier to remember.

The COMMENT field is another aid to the programmer. In this area is written a short comment on the reason for using that instruction. Ideally, the scope of the comments used should be sufficient for a person other than the programmer to make out what it is that the program is doing (this rarely happens in practice).

As you may have concluded, the ADDR and CODE fields are the only ones that have anything to do with the numbers that you enter into the computer to make the program run.

At this point it becomes necessary to define a "byte". As we mentioned above, some of the instructions consist of two digits, some four, and some six, but all of them were in two-digit clusters. Each cluster is called a "byte" and that is the main unit of measurement we will be working with. For example: instead of saying each instruction can consist of two, four, or six digits, we say that it consists of one, two, or three bytes.

The memory of your computer is also measured in bytes. It comes with 512 bytes and an additional 512 bytes can be added by simply plugging in four more memory IC's. It takes three bytes of memory to store (hold) a three-byte-instruction. Each byte of memory has a unique address associated with it which enables the computer to pick out the particular byte it's looking for. You can easily visualize how the computer's memory is organized if you think of it as a town with only one very long street. All the houses of the town are on that one street and the only way you can locate a particular house is by its address. If you think of each house as representing one byte of memory then that's what your computer's memory looks like. Each unique address is specified using a four-digit hex number. Look under the "ADDR" column of the program listing (Fig. 1) for an example of this. Notice that some numbers are skipped in the column. Each address shown is the address of the first byte of the instruction on the same line. In the case of a two- or three-byte instruction the addresses of the additional bytes are not shown but they are counted. Count the bytes in the program and you will notice that each time you start on a new line, the count will agree with the address listed on that line until you get past nine. Remember now that we are working in hexadecimal (hex) and there are six more symbols to count after the "9" symbol. Here is an example of how to count in hex:

0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19,
1A, 1B, 1C, 1D, 1E, 1F, 20, 21, 97, 98, 99, 9A, 9B, 9C,
9D, 9E, 9F, A0, A1, A2, A8, A9, AA, AB, AC, AD, AE, AF, B0, B1,
B2, F8, F9, FA, FB, FC, FD, FE, FF, 100, 101, 102,
and so on.

Now you should be ready to enter the program from Fig. 1 into your computer. Start by applying power to the computer, then press the reset button. Arbitrarily touch some of the numbered keys and notice how the numbers shift left through the display. The display only shows the last two entries from the keyboard but the computer can remember as many as the last twelve. If anything goes wrong and the display stops responding to the keyboard, press the reset button and it should return to normal;

```
Now type: 0-0-0-0          DISPLAY shows: 00
(Touch the 0 key four times)
DISPLAY                      xx (x-don't know, don't care)
(Touch the DISP key)
```

This sets the pointer to memory location 0000, which is the address of the first byte to be entered into the memory (see ADDR column of Fig. 1.). The display will show the contents of that location. This operation lets the monitor know where in memory your program is to be stored (programs don't always start at 0000).

```
Type: A-9          DISPLAY shows: A9
      ENTER                xx
```

This enters the first byte of the program into the computer's memory, moves the pointer to the next address in memory and displays the contents of that next address. It is important to understand the concept of the pointer since it will be referred to quite often. Each time the "ENTER" key is touched, what you see in the display will be stored in the memory location specified by the pointer. The pointer will then be incremented to the next memory location and the contents of that location will be displayed.

```
Type: 0-0          DISPLAY shows: 00
      ENTER                xx
```

This enters the second byte of the program into memory. The first and second bytes of the program form the first instruction of the program which is a LDA (load accumulator) instruction (See Fig. 1).

```
Type: 8-D          DISPLAY shows: 8D
      ENTER                xx
      2-0                  20
      ENTER                xx
      0-8                  08
      ENTER                xx
```

These three bytes form the second instruction (STA-store accumulator) of the program (See Fig. 1).

Now that you have the hang of it, enter the rest of the program listing under the "CODE" column in Fig. 1 starting with A0, 00, A2, etc.

CORRECTING ERRORS

If you make a mistake in typing but catch it before touching the "ENTER" key then you can correct it by simply retyping the correct entry; the mistake will be shifted out of the display. If you have already entered the mistake in memory; then touch the "BACKSPACE" key and the mistake will reappear in the display. Now type the correct entry and then be sure to touch the "ENTER" key or the memory will still contain the mistake. Touching the "BACKSPACE" causes the monitor to decrement the pointer and then display that location.

EXAMINING THE PROGRAM

Now that you have the program in memory it's a good idea to go back and check it to make sure it was entered 100% correctly. If even one digit is wrong then the program will not operate properly. First you must let the monitor know where in memory your program is; or in more technical terms: set the pointer to the beginning of the program. To do that you must type: "0-0-0-0-DISPLAY". Always remember that the "DISPLAY" key is used to set the pointer. The display should now show the first byte of the program (A9). If it doesn't then you have done something wrong and you should start all over. If it does then you can examine the next byte by simply touching the "ENTER" key. This causes the data shown in the display (which is what was in the memory location in the first place) to be entered back into the same memory location and increments the pointer to display the next location. You can step through the program by repeatedly touching the "ENTER" key. The series of bytes seen in the display should correspond with the ones in the program ("CODE" column of Fig. 1). If you find a byte that's not correct you should retype it while it's in the display and then touch the "ENTER" key.

RUNNING THE PROGRAM

Everything should be set to run the sample program now. To execute (run) a program you must tell the computer where the starting point of the program is. In the sample program the starting point is at the beginning instruction (ADDR 0000); However, not all programs start at their beginning.

Type: 0-0-0-0
RUN

DISPLAY shows: 00
the program counting

This tells the monitor to execute a program starting at location 0000. If all is well your display should have started with 00 and should be counting its way to 99 at which time it will start over. It will take approximately 30 seconds to count from 00 to 99. If your display is not counting then something is wrong and you should go back and examine the program for errors. Notice that touching keys on the keyboard produces no results since the computer is running the sample program and not the monitor program. Keyboard control can only be regained by pressing the "RESET" button which causes the computer to return to the monitor program.

MODIFYING THE PROGRAM

You can make your computer count faster or slower by changing the speed setting at address 0008. To make it count faster, a smaller number should be substituted. For example;

Type: RESET	DISPLAY shows: 00
0-0-0-8	08
DISP	50
2-0	20
ENTER	C8
0-0-0-0	00
RUN	counting

This will cause the counting rate to increase by more than double. Notice that the operations performed were: (0-0-0-8-DISPLAY) set the pointer to location 0008; (2-0-ENTER) enter 20 in location 0008; (0-0-0-0-RUN) run the program starting at location 0000. (Note: the speed setting is in hex; therefore the largest number that can be used is "FF" and not "99".)

You can change the number that the program counts with by changing location 0012 (presently "01"). Try "05".

For an interesting effect restore location 0012 to "01" and then change locations 0007 and 0008 to "AA" and "EA" respectively. (The easiest way to accomplish this is as follows: 0-0-0-7-DISPLAY-A-A-ENTER-E-A-ENTER-0-0-1-2-DISPLAY-0-1-ENTER). This replaces a two-byte instruction (LDX #\$50) with two one-byte instructions ("TAX" and "NOP"). Now run the program and note that the effect produced is to count slower as the number gets larger. It is left as an exercise to the user to determine why these changes produce this effect.

If you would like for the computer to teach you how to count in hex then restore the program to normal and then change location 000F to "D8". Run the program and watch the display count up in hexadecimal (You may want to slow it down as noted above).

OTHER GOODIES IN PIEBUG

So far you have used four control keys (DISPLAY, ENTER, BACKSPACE, and RUN). Four more remain to be defined (POINTER HIGH, POINTER LOW, TAPE, and RELATIVE ADDRESS COMPUTE). Since the pointer contains four digits but the display can only show two digits, the pointer is divided into two segments: POINTER HIGH and POINTER LOW. Each contains two digits of the pointer.

POINTER HIGH (PH) AND POINTER LOW (PL)

These two keys are used to see exactly what address the pointer contains. Touching key "PH" will display the first two digits of the pointer and likewise "PL" will display the last two digits. Normal sequence is "PH-PL-DISP" which will show you
20 the pointer and then the contents of the location it's pointing to.

TAPE

If you have the cassette tape option this key can be used to save programs on tape and load them back into the computer at a later time. Details of its use are supplied with the option.

CAUTION

If your computer does not have this option and you touch this key, you may lose control of the computer and it may overwrite portions of your program with garbage and it may just eat your lunch!

RELATIVE ADDRESS COMPUTE

As you learn to write programs you will develop a need to compute relative addresses. These addresses take only two digits instead of the usual four and can be computed by hand. However, a much faster and more accurate way is to let the computer do it at the touch of a button. The monitor contains a program to compute relative addresses for you. To use it you simply enter a program as you normally would and then when you come to a branch operand, instead of typing in the operand (relative address) type in the absolute (4-digit) address of the destination and then touch the "REL" key. Instantly the correct operand will appear in the display. If the display indicates "00" then the destination was out of range. Otherwise you may enter the operand with the "ENTER" key. Part of the sample program is used for an example. Starting at location 0009;

Type: 0-0-0-9	DISPLAY shows: 09
DISP	xx
C-8	C8
ENTER	xx
D-0	D0
ENTER	xx
0-0-0-9	09
REL	FD
ENTER	

When you touched the "REL" key the display should have indicated "FD", as shown in the program.

DEBUGGING YOUR PROGRAMS

Normally a new program will never run properly the first time (this is a perfect example of Murphy's Law: If anything can go wrong, it will!). Therefore some means of determining what went wrong with your program is necessary. Most computers use a "breakpoint" for this purpose. The idea behind it is to stop the computer at some specified point in your program and display the contents of the processor's internal registers as well as any other memory locations pertinent to your program (such as those containing status information). By doing this you can compare the status of the computer against what you thought it should be at that point. If it doesn't agree then you have a clue to what is wrong and by placing the breakpoint at previous points in your program you can determine just where it is that you and your computer disagree.

Determining just where to put the first breakpoint is usually a "seat-of-the-pants" operation. If some part of your program is supposed to do a certain job and that particular job doesn't get done then that's usually a good place to start with a breakpoint. Indeed, many times you will put in a breakpoint only to find that the computer never got to that part of the program at all (indicated by the fact that it never breaks). In such a case you should put breakpoints in earlier parts of the program until you find some part of the program that the computer is running and then proceed to move the breakpoint toward the problem area until you find where you are losing the computer.

TO USE THE BREAK DEBUGGER FUNCTION IN THE MONITOR YOU MUST ENTER THESE THREE BYTES STARTING AT LOCATION 0000: 4C, C0, FF. To place a breakpoint in your program, change the opcode of the selected instruction to "00". This is the break code and it must always be substituted for an opcode and never an operand. When the computer comes to the break code it will display "BB" to indicate a break and it will save the contents of its internal registers in the following memory locations:

00F9	ACCUMULATOR
00FA	Y-REGISTER
00FB	X-REGISTER
00FC	PROGRAM COUNTER LOW
00FD	PROGRAM COUNTER HIGH
00FE	STACK POINTER
00FF	STATUS REGISTER

Control will then be returned to the monitor and you can examine and change any memory locations including the ones above. The program counter locations above will indicate where the break occurred.

If desired you can continue from where the break occurred by replacing the break code with the original instruction opcode and then running the program from that point. Each time the RUN key is touched all the registers in the processor are loaded from the above locations before executing the program (with the exception of the program counter which is loaded from the keyboard). This gives you the ability to run a program to the break, examine and change any registers or memory locations necessary, and then continue from that point. You can also start a program at some point other than the beginning by preloading the registers with the values expected at that point in the program and then running at that point.

STACK POINTER

The PIEBUG Monitor maintains two different stacks; one for the monitor and cassette routines and a separate stack for your programs. The reason for this is to keep the monitor from destroying your program stack. Preserving your stack can sometimes aid in program debugging since the monitor can then be used to examine it.

This is especially useful if your program stores data on the stack. However, you must be careful how you interpret this information since the break command itself uses three bytes of your stack.

You have control over where these two stacks are located in page one of memory (0100 - 01FF; the processor limits the stack pointer to page one). To set the initial position of the monitor stack, store the desired value in memory location 00ED; likewise your stack is set with location 00FE. The monitor and cassette routines require only ten bytes of stack space.

Note: It is not necessary to set the stack locations at all if
(1) You do not need to examine the stack during debugging
and (2) You do not write any programs in page one.

If you do write programs in page one then you must be familiar with how the stack operates, know how much room it will need, and locate it accordingly so it will not destroy your program. To save space, both stacks can be located at the same place if you do not need to examine the stack during debugging.

GENERAL NOTES

Always remember that the reset button is the panic button! When pushed, control should return to the monitor. If it doesn't then something is wrong with the computer.

The memory that you are storing programs in is called "RAM" memory. When you turn the power off it loses its mind and forgets everything it knew (such as programs and data; hence the cassette tape for saving things). So if you can't seem to make the break function work, make sure you have re-entered those three bytes starting at 0000.

RAM locations 00ED thru 00FF are reserved for use by the monitor. You should not use these locations in your programs unless you are familiar with how they affect the monitor.

QUICK REFERENCE

Definitions:

BUFFER	Memory locations (00F0 through 00F5) that the monitor uses to save the last 12 entries from the keyboard. Only the last 2 or 4 entries are used in monitor operations.
POINTER	16-bit address that indicates which byte of memory is to be affected by the next operation.
ACTIVE CELL	Memory location currently being specified by pointer.
DISPLAY	On-board two-digit led display.

Commands:

DISPLAY	Displays contents of memory location specified by the last 4 entries from the keyboard and sets the pointer to that location. Moves buffer to pointer, then moves active cell to buffer and display.
ENTER	Stores the contents of the display in the currently addressed memory location and then displays the contents of the next location. Moves buffer to active cell, increments pointer and moves new active cell to buffer and display.
BACKSPACE	Displays contents of the memory location previous to the current one and then sets the pointer to that location. Decrements pointer, then moves active cell to buffer and display.
RUN	Executes program starting at location specified by the last 4 entries from the keyboard. Loads program counter from buffer, all other processor registers from appropriate register storage (00F9 thru 00FF).
POINTER HIGH	Displays first two digits of pointer. Moves pointer to buffer, pointer high byte to display.
POINTER LOW	Displays last two digits of pointer. Moves pointer to buffer, pointer low byte to display.
TAPE	Transfers control to the tape routines (optional). Note: Use of this key without the tape option will cause loss of control.
RELATIVE	Computes relative address when active cell is a branch operand. Moves result of (buffer minus pointer+1) to buffer and display. Sets results to "00" if out of range. See Text.

Useful Zero Page Locations:

00ED	Monitor stack
00F0	Buffer, LSB (latest entry)
00F1	Buffer
00F2	Buffer
00F3	Buffer
00F4	Buffer
00F5	Buffer, MSB (oldest entry)
00F9	Accumulator
00FA	Y-Register
00FB	X-Register
00FC	Program counter low
00FD	Program counter high
00FE	Stack pointer (user)
00FF	Status register

Vectors:

NMI - 0003
RES - FF48
IRQ - 0000

Break Vector: Store starting at 0000; 4C, C0, FF

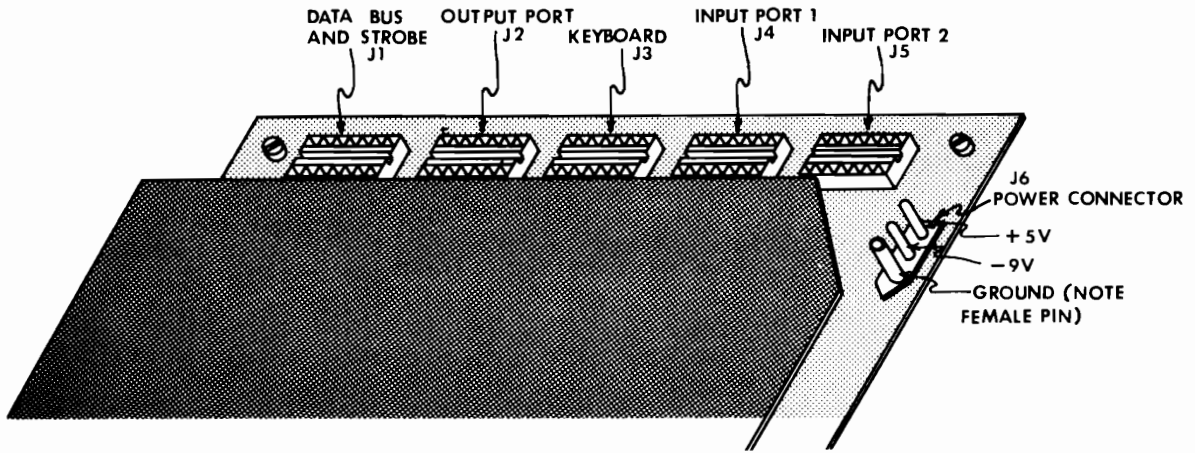
Memory Map:

0000-00FF	RAM (IC22, IC30)
0100-01FF	RAM (IC23, IC31)
0200-02FF	RAM (IC24, IC32)
0300-03FF	RAM (IC25, IC33)
0400-07FF	UNOCCUPIED
0800-08FF	I/O
0900-09FF	CASSETTE (IC20, IC21) S9
0A00-0BFF	UNOCCUPIED
0C00-0CFF	PROM (IC16)
0D00-0DFF	PROM (IC17)
0E00-0EFF	PROM (IC18) CASSETTE OPTION
0F00-0FFF	PROM (IC19) MONITOR

I/O Breakdown

0801	KEYBOARD (IC1, IC2)	KEYS 0-7
0802	KEYBOARD (IC3, IC4)	KEYS 8-F
0804	KEYBOARD (IC5, IC6)	CONTROL KEYS
0808	INPUT PORT 2 (IC4, IC8)	J5
0810	INPUT PORT 1 (IC3, IC7)	J4
0820	DISPLAY (IC26, IC27, IC28, IC29)	
0840	OUTPUT PORT (IC2, IC6)	J2
0380	STROBE (IC1)	J1

System Analysis



OUTPUT PORT (J2)
OUTPUT PORT ADDRESS - x840

The output port is a means of getting data being processed within the computer out to peripheral devices.

The eight output lines (bit 0-bit 7) are all latched and each represents a CMOS output structure.

Included at the output port connector are the system power voltages, +5 volts and -9 volts and gnd.

+5 -	○ ¹	14	○	- GND
-9 -	○		○	- GND
BIT 7 -	○		○	- BIT 6
BIT 5 -	○		○	- BIT 4
BIT 3 -	○		○	- BIT 2
BIT 1 -	○		○	- BIT 0
+5 -	○ ⁷	8	○	- GND

PROGRAMMING CONSIDERATIONS

The port is memory-mapped, so that any instruction which would ordinarily be used to write data into memory can also be used to write data to the output port.

PROGRAMMING EXAMPLE

```
0020  LOOP  E8  inx; increment count
0021          8E  stx (abs); write result to output port
          40
          A8
0024          4C  jmp LOOP; go to do next
          20
          00
```

ANALYSIS

This short program causes the bits of the output port to count in binary. Bit 0 is the least significant, bit 7 the most significant.

When running, the program increments the X index register by 1 (INX) at location 0020, the STX instruction at location 0021 causes the incremented result in the X register to be "stored" in the output port which occupies memory location x840. The JMP instruction at location 0024 causes the program to loop back to the beginning.

NOTICE TWO THINGS:

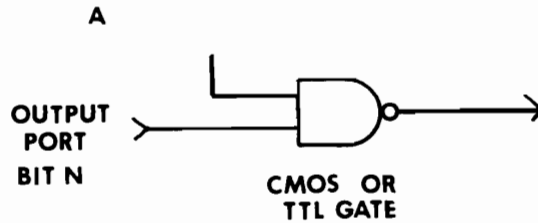
1) the location of the output port is listed as x840 where x can be any hexadecimal digit. In this example x is A, but this is arbitrary. Using an oscilloscope you can check that the output lines are counting and that x can be given any value from 0-F without affecting the operation of the program.

2) because of the pipe-lined architecture of the 650x family of processors, absolute addresses are given LEAST SIGNIFICANT BYTE FIRST. This will be confusing to first-time users of these processors but results in significantly greater processor through-put than would otherwise be possible. (See 6500 PROGRAMMING MANUAL.)

HARDWARE INTERFACING

The easiest situation is interfacing the output port to CMOS logic, which is simply

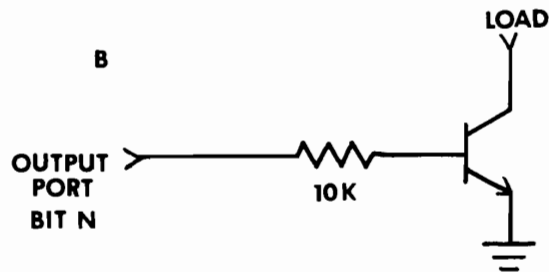
a matter of tying the output port pin to the input of the CMOS load. Like this:



Because of the static nature of these outputs, practically any number of CMOS gates can be driven. (The limiting factor is the risetime of the output as the additional capacitors that the inputs of the gates represent are added.) If you like, and if the specifications of the power supply are not exceeded, power for the peripheral device can be picked up on J2 as are the signal leads.

TTL gates are just as easily driven from the output port, but unfortunately not in unlimited quantities. To be on the safe side, stick to one regular TTL load or two LS TTL loads max.

When interfacing to a discrete transistor, a current-limiting resistor should be put in the line like this:



If needed, the activating signal that strobes new data into the output port latches, (OUTPORT) is present on pin #10 of the DATA BUSS and-STROBE connector (J1)

INPUT PORTS

PORT #1 (J4) - ADDRESS x810

PORT #2 (J5) - ADDRESS x808

+5-	O ¹	14	O	- GND
-9-	O		O	- GND
BIT7-	O		O	- BIT 6
BIT5-	O		O	- BIT 4
BIT3-	O		O	- BIT 2
BIT1-	O		O	- BIT 0
+5-	O ⁷	8	O	- GND

The input ports are means of getting data from the outside world into the computer.

Each input line represents a single CMOS input structure.

Included at the input port connectors are the system supply voltages +5 volts -9 volts and gnd.

PROGRAMMING CONSIDERATIONS

Like the output port, these input ports are memory mapped and any instruction which reads data from a memory location may be used to read the port into the processor.

PROGRAMMING EXAMPLE

```
0020 LOOP AD LDA (abs) IN#1 ;read input port
          10
          A8
0023      8D STA (abs) DSPLY ;put result in display
          20
          A8
0026      4C JMP LOOP ;do again
          20
          00
```

ANALYSIS

The instruction at location 0020 causes data which is currently being presented to the input port to be read to the processor's accumulator. The next instruction writes this same data to the display. Finally, the jump instruction at 0026 causes the program to loop and start again.

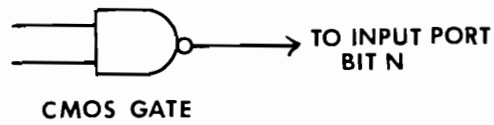
NOTICE ONE THING

Since the input port is a CMOS input, normal precautions should be taken to prevent static damage at these pins; also, if the above program is run without some device connected to the port, some means must be provided to hold the input pins of the port at either ground or supply. Otherwise, normal environmental electromagnetic fields will cause the state of the input lines to be indeterminate. 10K ohm resistors from the pins to either ground or supply (see also HARDWARE INTERFACING) will suffice.

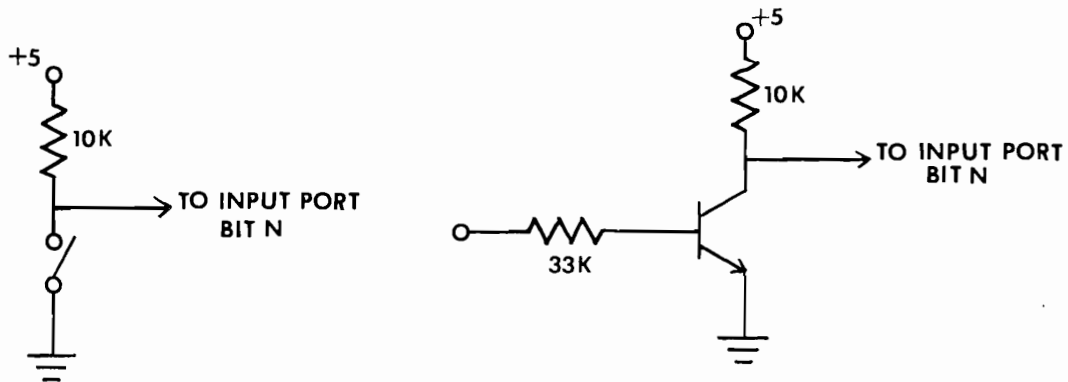
HARDWARE INTERFACING

Being a CMOS input, a variety of devices can supply data to the input ports. The output of another CMOS gate can be connected directly to the port:

A

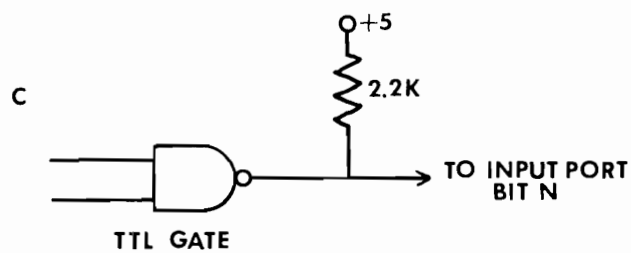


or switches or transistors may be used:



















and note that if the transistor or switch above is "on", it represents a 0 input to that pin of the port.

If the output of a TTL gate is being used to drive the input port, a pull-up resistor to supply must be provided:



DISPLAY

DISPLAY ADDRESS - x820

	-	0		-	8
	-	1		-	9
	-	2		-	A
	-	3		-	B
	-	4		-	C
	-	5		-	D
	-	6		-	E
	-	7		-	F

The displays consist of two seven-segment displays and associated 9368-type decoders/latches/drivers. The decoder portion of the 9368 takes care of converting a single 4-bit hexadecimal digit input to the appropriate code required to operate the seven-segment displays.

These devices will display all 16 symbols in the hexadecimal character set from 0-F. NOTE that the characters B and D are both displayed as lower case characters (b and d), and that the character 6 is distinguished from the character b by the horizontal "tail" at the top of the 6.

Like other peripheral ports, the displays are memory mapped and any instruction that writes to memory will operate them.

NOTE: it is normal for the 9368 driver ICs to operate at an elevated temperature.

TYPICAL SOFTWARE

(see KEYBOARD section of system analysis for typical programming examples using the displays.)

KEYBOARD

FIRST RANK (0-7) - address x801
 SECOND " (8-15) - address x802
 THIRD " (16-23) - address x803
 DECODE SUBROUTINE address FF00

The keyboard is used by the monitor for control of user data and program entry as well as operation of the PIEBUG debugging tools, but may also be read by the user's programs employing a variety of techniques.

Because of the capacitive operating principle employed in the 8700 keyboard, this device should provide exceptionally long and trouble-free life.

(FOR EXPLANATION OF KEYBOARD
WHEN USED WITH MONITOR, SEE
PIEBUG MONITOR.)

PCH 14	PCL 15	TAPE 16	REL 17
RUN 10	DISP 11	BACK 12	ENTER 13
C 0C	D 0D	E 0E	F 0F
8 08	9 09	A 0A	B 0B
4 04	5 05	6 06	7 07
0 00	1 01	2 02	3 03

USING THE KEYBOARD AS AN INPUT TO USER'S PROGRAMS

There are two ways that the 8700's keyboard may be used to input data to a user's program.

1) Individual "ranks" of keys may be read with any of the statements that read memory locations. For example:

```

0020 LABEL  AD  LDA (abs) A801      ;read first rank
              01
              A8
0023        FO  BEQ LABEL          ;if no key, loop
              FB
  
```

causes the status of the first 8 keys on the keyboard to be read into the accumulator of the computer (instruction at location 0020). If no keys are being touched when the read operation occurs, the accumulator will be loaded with \$00. Under these circumstances, the Branch Not Equal at location 0023 will cause the processor to loop back to the top of the program and read the keyboard again. If a key is being touched when the read operation happens, the accumulator will be loaded with a number that represents the key. Each of the 8 bits in the word that is read represents a key, from \$01 (in binary 00000001) for key #0 to \$80 (in binary 10000000) for key #7.

While there are circumstances when the above procedure will suffice for inputting data, there will be times when it is most convenient to read not simply one rank of

keys, but rather the whole keyboard.

A new program to do this can of course be written, but under most conditions the effort would be redundant as this program is already a part of the PIEBUG Monitor and written as a subroutine so that it can be easily accessed from user's programs. This subroutine is named DECODE and it lives in the Monitor Prom at address FF00.

Before using this subroutine, there are a few things that you should know about it, like; when called, the routine returns with the number of the key down in both the accumulator and the Y index register, so if either of these registers contains data that will be needed after the keyboard scan, it should be either pushed to the stack or otherwise saved in memory. Similarly, though the X index register doesn't contain any key information when DECODE is exited, its contents are altered by this routine and as with the accumulator and Y register it should be saved (if needed) before entry to DECODE.

If no keys are down, the routine is exited with \$18 in A and Y and this fact can of course be used to determine if a key is down or not.

A problem that is just as important as determining that a key is down and which key it is, is to determine whether the key that is down now is the same one that was down the last time through the program. (Otherwise, what is intended as a single keyboard stroke can be interpreted as multiple switch activations, one activation for each pass through the routine). Again, external user written code could be used to perform this task; but, again, it would be redundant as DECODE already indicates whether the key that is currently down is the first activation of that key- or if the key is simply still down. It indicates this by clearing (setting to 0) the Carry Flag in the processor status register; if the key that is activated during the current scan of the keyboard is different from the key that was activated during the last scan. If the same key that was down during the last scan of the keyboard is the same one as is down during this scan, the Carry Flag will not be cleared. Note also that the carry flag is cleared only when a new key is activated, not when a key is released.

The existence of instructions to test the Carry Flag (BCS-Branch if Carry Set- and BCC-Branch if Carry Clear) make the use of this feature exceptionally easy.

A simple user program to scan the keyboard and display the key that is down could look like this:

```
0020  LOOP      20  JSR  DECODE      ;jump to monitor
                                00  ;keyboard routine
                                FF
      23      B0  BCC  LOOP      ;test for new key
                                FB
      25      8D  STA  DSP      ;if new key, put
                                20  ;in display and...
                                A8
      28      4C  JMP  LOOP      ;begin again
                                20
                                00
```


It is the op code (BO) at location 0023 and its corresponding operand at the next location that causes the program to skip the display if no key is found down. By replacing these two bytes with NOPs (EA) the program may be modified to display the key number while the key is held down and display 18 (the no-key code) when no keys are pressed.

**DATA BUSS and STROBE
CONNECTOR (J1)**

STROBE - Address x880
DISPLAY - Address x820
OUTPORT - Address x840
CASSETTE - Address x9xx

DB7 - ○ ¹	14 ○ - DB6
DB5 - ○	○ - DB4
DB3 - ○	○ - DB2
DB1 - ○	○ - DB0
<u>STROBE</u> - ○	○ - <u>OUTPORT</u>
<u>DISPLAY</u> - ○	○ - <u>CASSETTE</u>
+5 - ○ ⁷	8 ○ - GND

This connector provides direct access to the data buss as well as a selection of the system peripheral enable signals. Some of the enabling signals are activated when a single address is accessed, others when any one of a group is called for, as summarized below.

Electrical loading is an important consideration in using this connector. Five CMOS loads or one LS TTL is a safe bet, but more than that is on the questionable side. The select lines (STROBE, etc) will each drive 4 TTL loads.

The pins labeled DB0-DB7 provide access to the data buss from least significant to most significant respectively.

System +5 volts and ground appear at pins 7 and 8 respectively.

All enable signal lines are memory mapped.

PERIPHERAL ENABLE SIGNALS

STROBE - Provides a low-true signal when any of the following addresses are read from or written to:

x880	x8A0	x8C0	x8E0
x890	x8B0	x8D0	x8F0

DISPLAY - This is the select line for the 8700 displays. This line is low-true on a write operation to the address occupied by the displays (x820).

OUTPORT - The low-true select line for the output port which lives at address x840 activates on write operations only.

CASSETTE - The select line for a contiguous block of 256 addresses from locations x900 - x9FF. Activates on write operations only.

NOTE: All tape dump operations are written to address x900 and this address should be reserved for this operation only. All active addresses above x900 may be used, but if the two relay drivers are used, care must be taken during transfers so that the duty factor of the pulses is not sufficient to close the relays.

EXPANSION CONNECTORS

J7 and J8

EXPANSION CONNECTORS

J7 AND J8

$\overline{\text{IRQ}}$ - O^1	O - $\overline{\text{RES}}$
$\overline{\text{NMI}}$ - O	O - $\overline{\text{GND}}$
$\text{AB}\emptyset$ - O	O - $\emptyset 2$
$\text{AB}1$ - O	O - $\emptyset 2\text{-R/W}$
$\text{AB}2$ - O	O - RAM R/W
$\text{AB}3$ - O	O - $\text{DB}\emptyset$
$\text{AB}4$ - O	O - $\text{DB}1$

$\text{AB}5$ - O^1	O - $\text{DB}2$
$\text{AB}6$ - O	O - $\text{DB}3$
$\text{AB}7$ - O	O - $\text{DB}4$
$\text{AB}8$ - O	O - $\text{DB}5$
$+5$ - O	O - $\text{DB}6$
$\text{AB}11$ - O	O - $\text{DB}7$
$\text{AB}9$ - O	O - $\text{AB}10$

The expansion connectors J7 and J8 provide access to the DATA, ADDRESS, and CONTROL busses of the processor as shown at right.

While these connectors are reserved for future expansions by PAiA, they may be used by the experienced user for system expansion. Appropriate care must be exercised that devices connected to these points do not exceed the loading capabilities of the processor and that appropriate protection against such real-world hazards as overvoltages and transient spikes is provided.

CASSETTE CONNECTOR (J9)

CASSETTE CONNECTOR

J9

$+5$ - O^1	O - $\text{RELAY } 2$
-9 - O	O - $\text{RELAY } 1$
EAR - O	O - MIC
EAR - O	O - GND

The cassette connector is used in conjunction with the CS-87 option to provide program and data-saving and loading from cassette recorder (see CS-87 Cassette option manual for operating details).

Additionally, this port and its corresponding components provides for a keyboard "beeper" which indicates activation of the control keys of the 8700/A Active Keyboard.