PAIA T.M.

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.

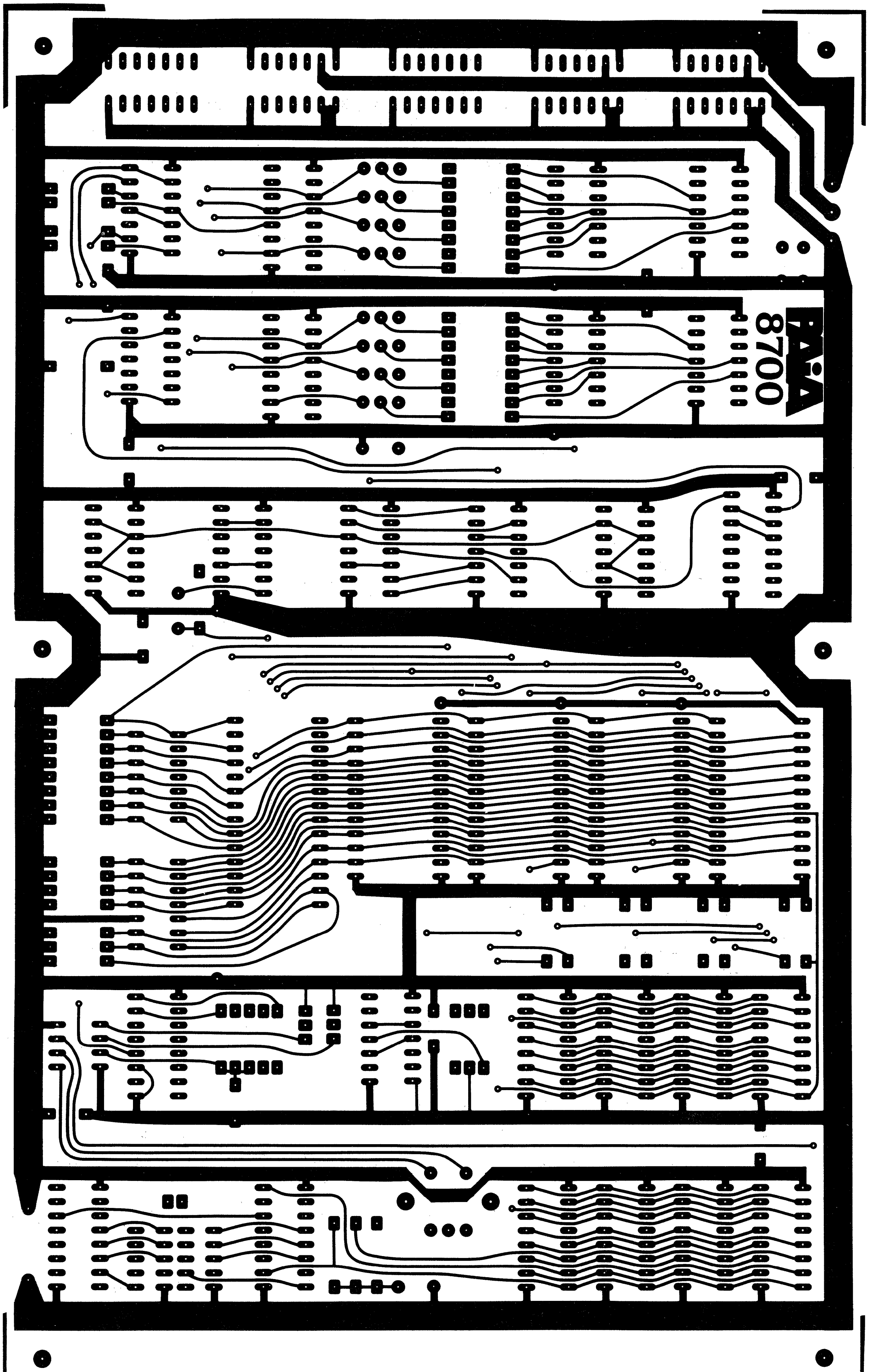


FIGURE 1 - CIRCUIT BOARD FOIL PATTERN (component side)

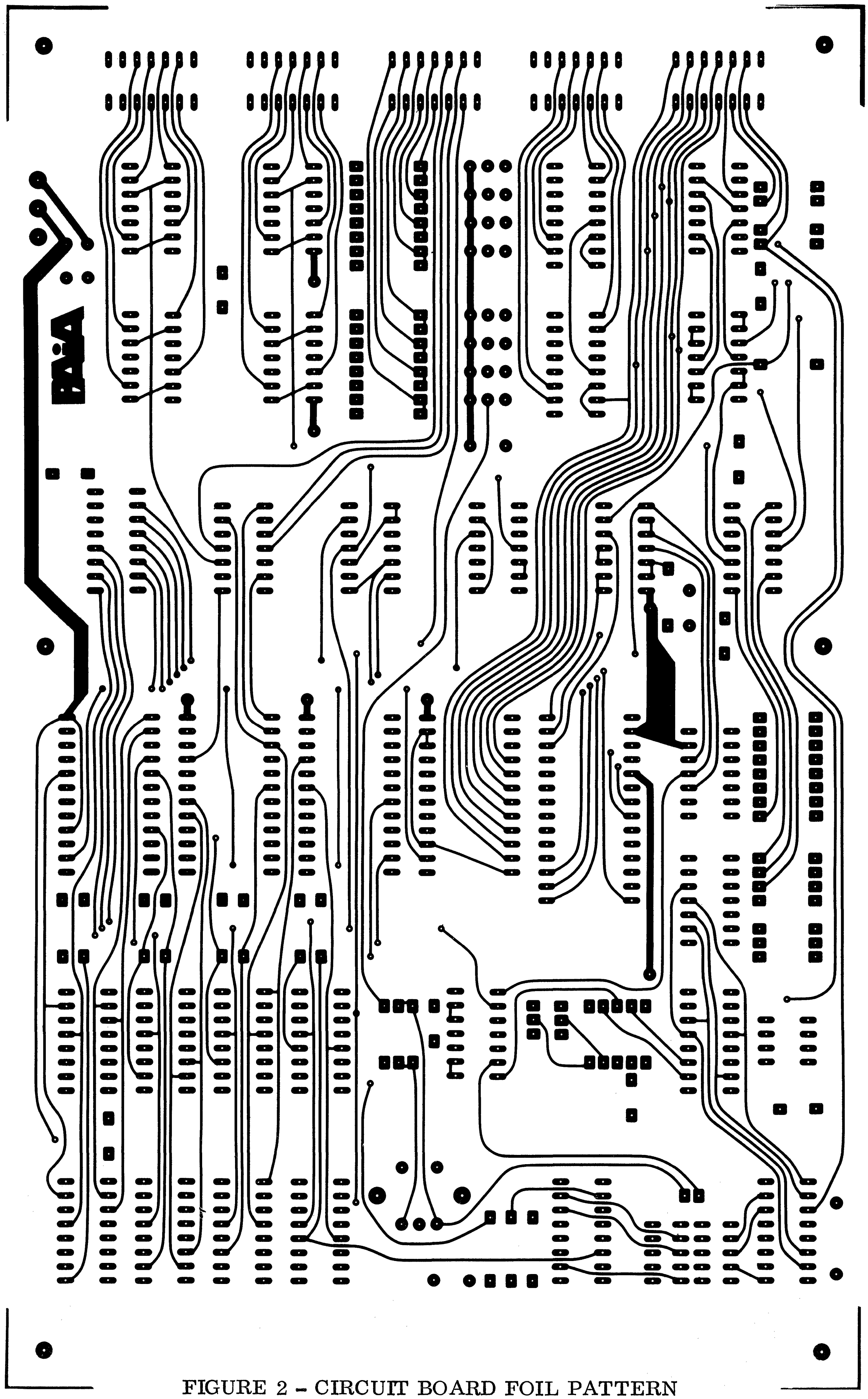


FIGURE 2 – CIRCUIT BOARD FOIL PATTERN

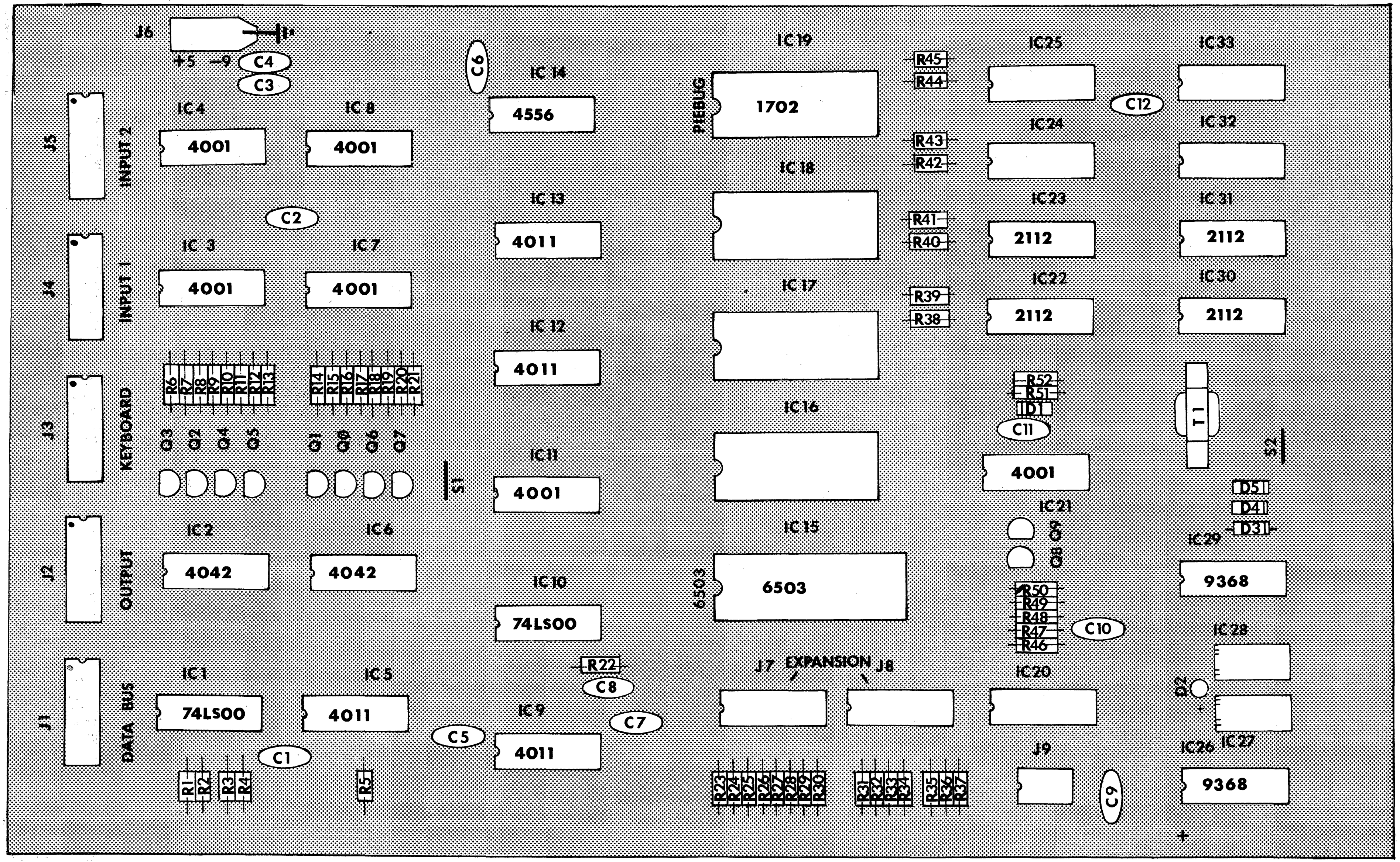


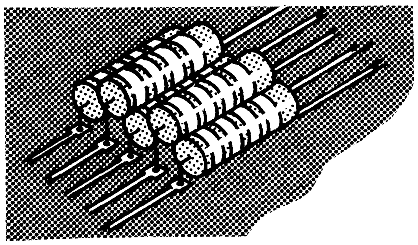
FIGURE 3

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^{o} 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:

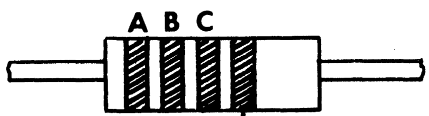
HAVE YOU INSPECTED THE BOARD? **It might just save you a lot of trouble.**



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.

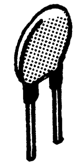
Installation of all resistors within a given group before any of the resistors in the group are soldered in place is highly recommended.



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.

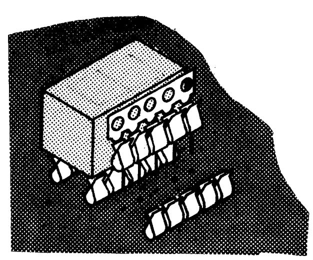
( ) Q0 ( ) Q1 ( ) Q2 ( ) Q3
( ) Q4 ( ) Q5 ( ) Q6 ( ) Q7

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.

( ) IC1 74LS00
( ) IC2 4042
( ) IC3 4001
( ) IC4 4001
( ) IC5 4011
( ) IC6 4042
( ) IC7 4001
( ) IC8 4001
( ) IC9 4011
( ) IC10 74LS02
( ) IC11 4001
( ) IC12 4011
( ) IC13 4011
( ) IC14 4556
( ) IC15 6503
( ) IC21 4001
( ) IC26 9368
( ) IC29 9368
( ) IC27 FND 357 Display
( ) 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

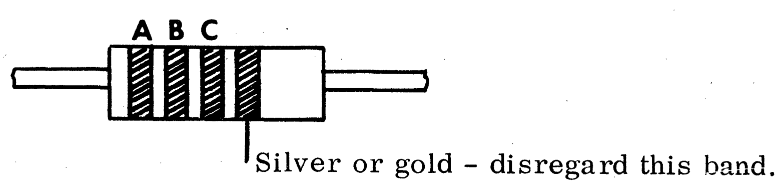
Prepare for assembly by thoroughly cleaning the exposed copper circuitry above colored keyboard area. Use steel wool and/or scouring cleanser. DO NOT USE PRE-SOAPED STEEL WOOL PAD. Use particular care to avoid scratching the printed keyboard area. Rinse and dry the board completely before beginning assembly.

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.

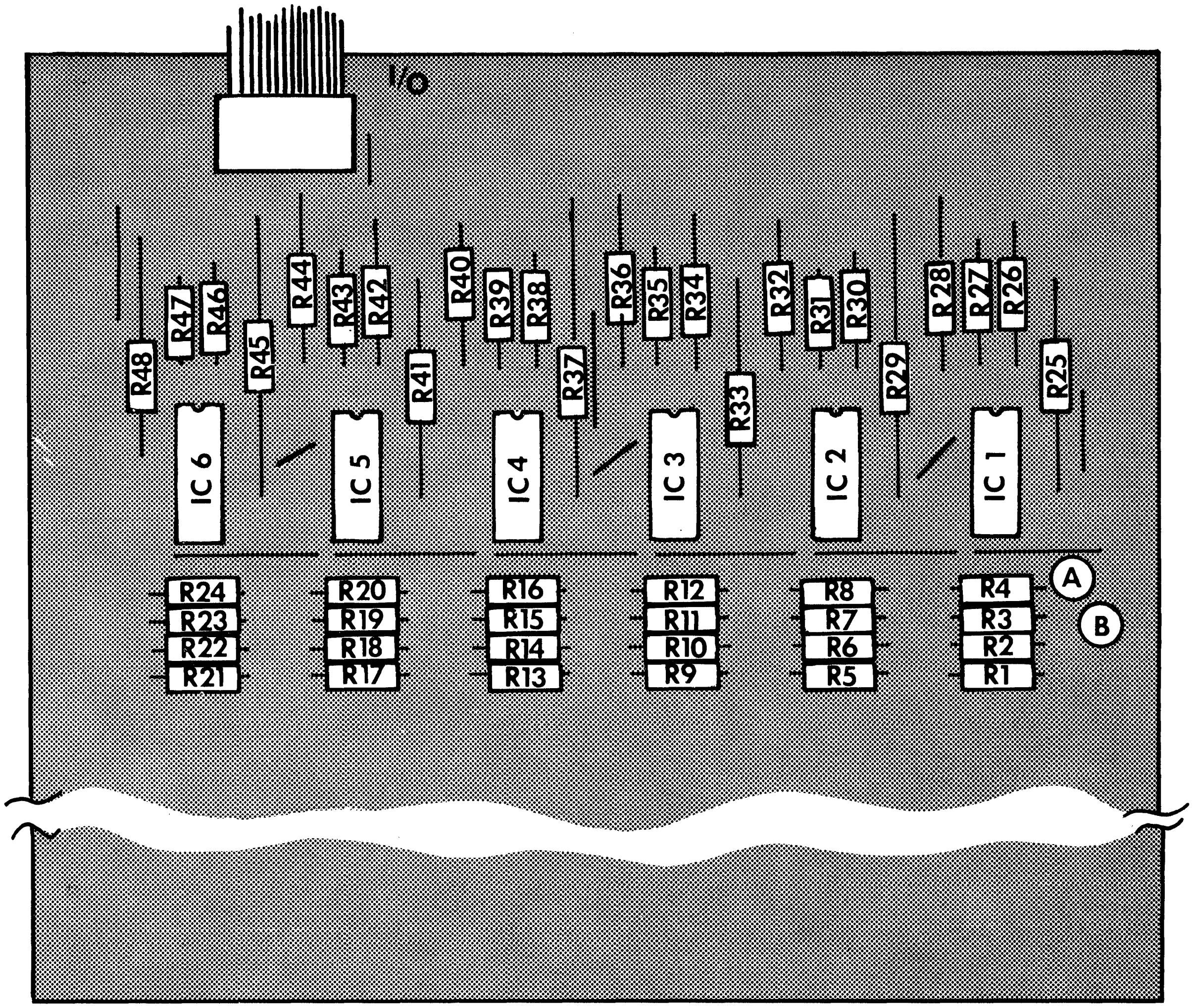


Figure 4

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^{o} 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.

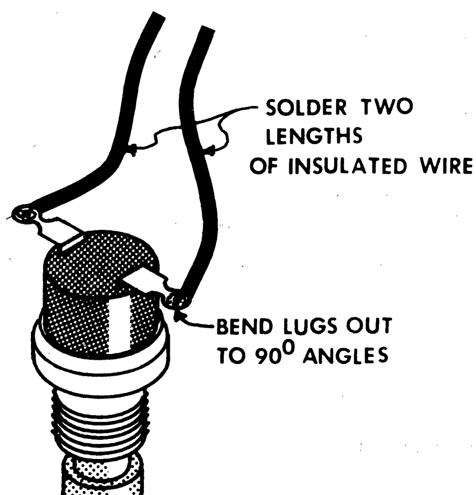


Figure 5

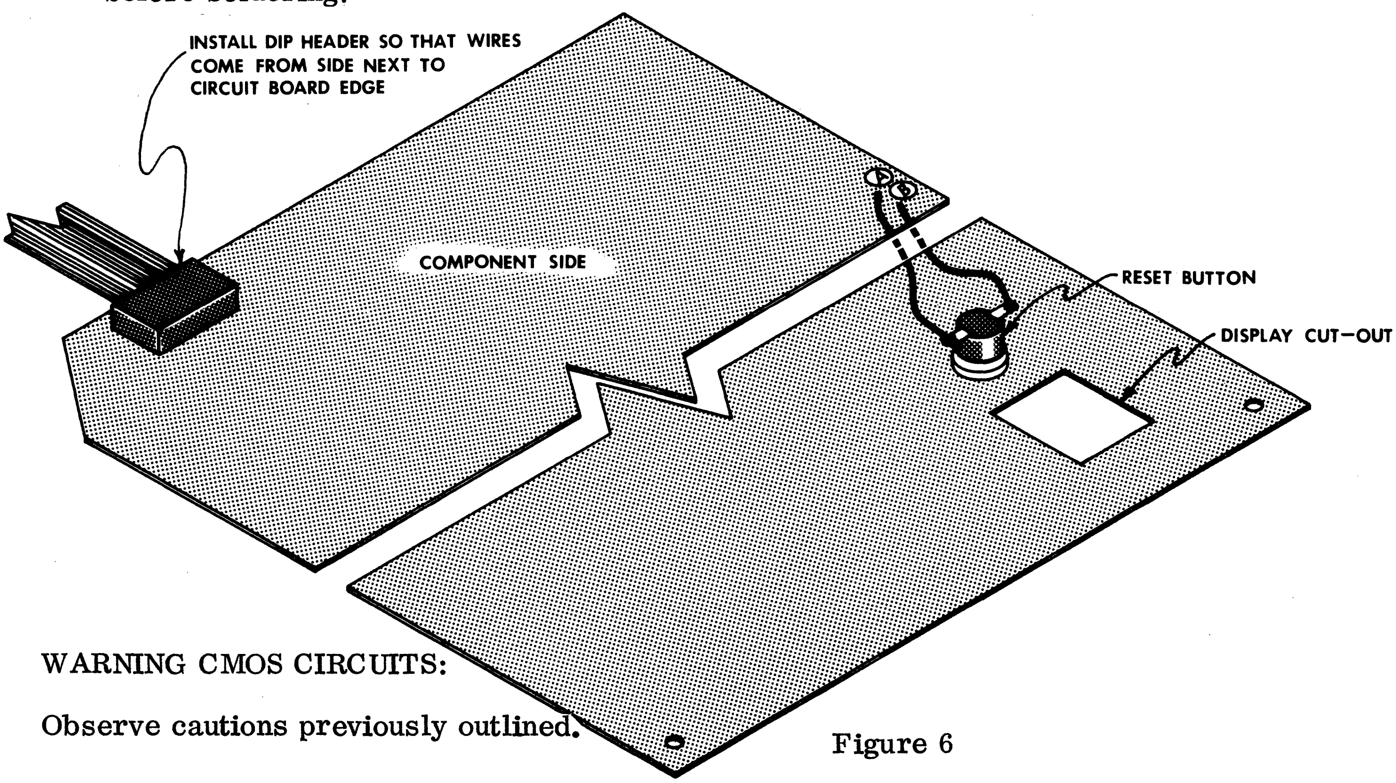
( ) Using the hardware supplied, mount the RESET button in the circular hole directly above the rectangular display cut-out. NOTE that the pushbutton mounts from the component side of the board so that the actuating stud protrudes from the side of the board printed with the keyboard designations.

( ) 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 <u>very clean</u> 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/0 cable with the 14 pin socket J3 (the middle socket of the five along the rear edge of the CPU board).

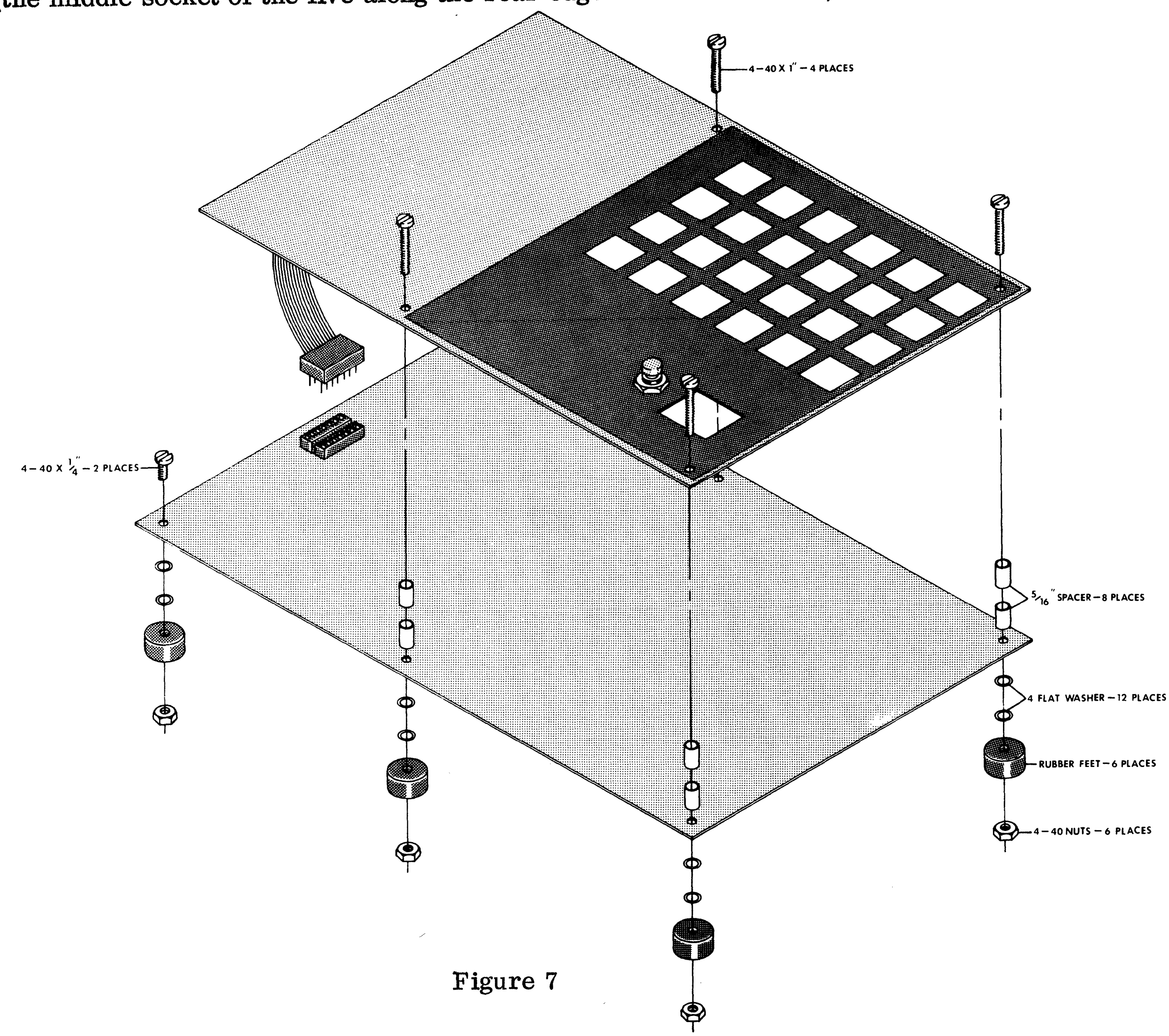


Figure 7

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

NOTES

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., 0123456789ABCDEF). 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 <u>not</u> 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 (Touch the 0 key four times)	DISPLAY shows:	00
DISPLAY (Touch the DISP key)		xx (x-don't know, don't care)

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	DISPLAY shows: 00
RUN	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:		DISPLAY shows:
RESET		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 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:	DISPLAY shows:
0-0-0-9	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 bever 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 seperate 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 <u>your</u> 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 - RAM on Video board
0800-08FF	I/0
0900-09FF	CASSETTE (IC20, IC21) S9
0A00-0~~B~~AFF	UNOCCUPIED - I/o ports on video board
0B00 - 0BFF	PROM - Video routine
0C00-0CFF	PROM (IC16) - 1702-A Programmer - Operating software, uses existing ports
0D00-0DFF	PROM (IC17) - Music
0E00-0EFF	PROM (IC18) CASSETTE OPTION
0F00-0FFF	PROM (IC19) MONITOR

I/O Breakdown

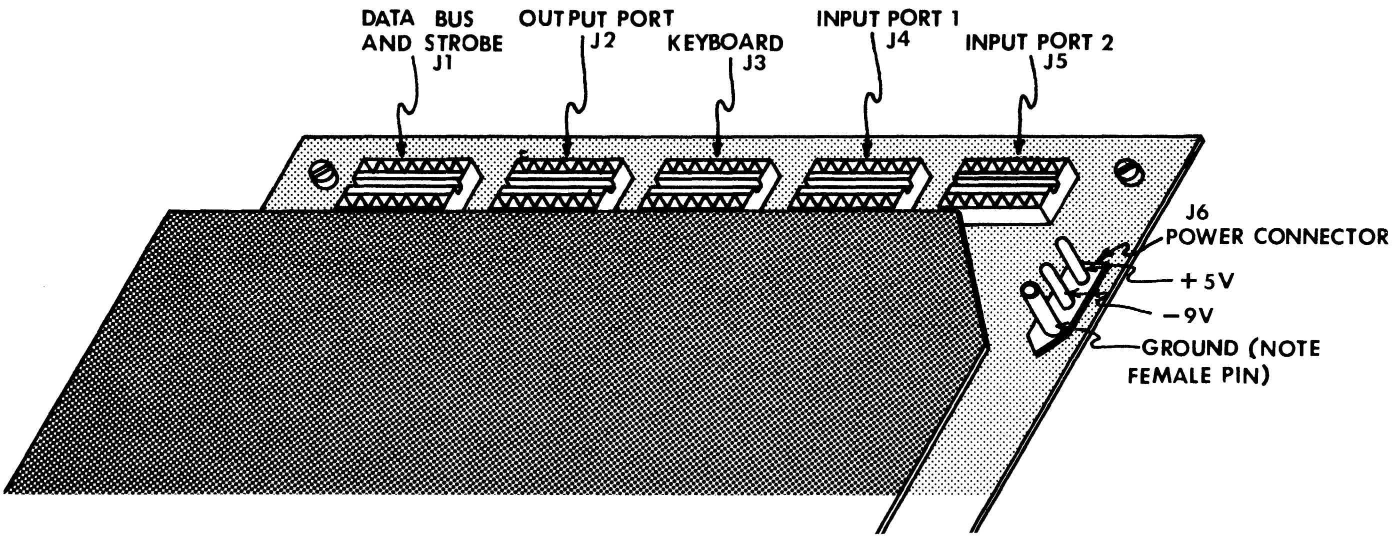
0801	KEYBOARD (IC1, IC2)	KEYS 0-7
0802	KEYBOARD (IC3, IC4)	KEYS 8-F
0804	KEYBOARD (IC5, IC6)	CONTROL KEYS
0808	INPUT PORT 2 (IC4, IC8) J5	
0810	INPUT PORT 1 (IC3, IC7) J4	
0820	DISPLAY (IC26, IC27, IC28, IC29)	
0840	OUTPUT PORT (IC2, IC6) J2	
0880	STROBE (IC1) J1	

XA0X – output
A1X – "
A2X – "

XA3X – input

XA4X – interrupt mask port
XA5X – " reset "
XA6X – " status "
XA7X – Video port

System Analysis



OUTPUT PORT (J2)

OUTPUT PORT ADDRESS - x840

+5 — O 1	14 O — GND
−9 — O	O — GND
BIT7 — O	O — BIT6
BIT5 — O	O — BIT4
BIT3 — O	O — BIT2
BIT1 — O	O — BITØ
+5 — O 7	8 O — GND

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.

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.

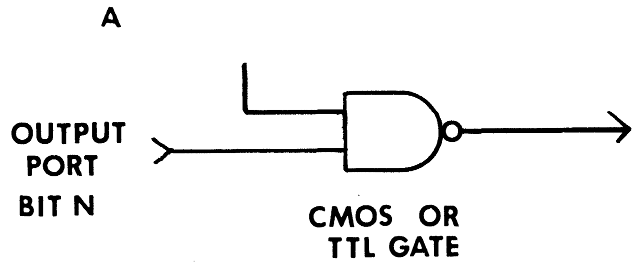
NOTICE TWO THINGS:

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

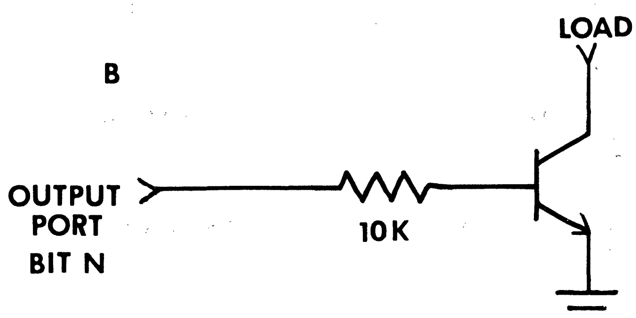
The easiest situation is interfacing the output port to CMOS logic, which is simply a matter of tying the output port pin to the input of the CMOS load. Like this:



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, ($\overline{\text{OUTPORT}}$) is present on pin #10 of the DATA BUSS and-STROBE connector (J1)

PORT #1 (J4) - ADDRESS x810
PORT #2 (J5) - ADDRESS x808

+5	1	14	GND
-9			GND
BIT7			BIT6
BIT5			BIT4
BIT3			BIT2
BIT1			BITØ
+5	7	8	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.

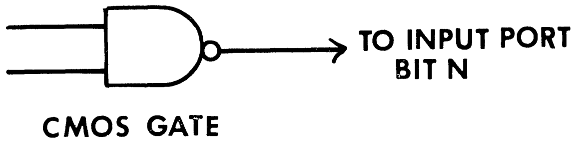
NOTICE ONE THING

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.

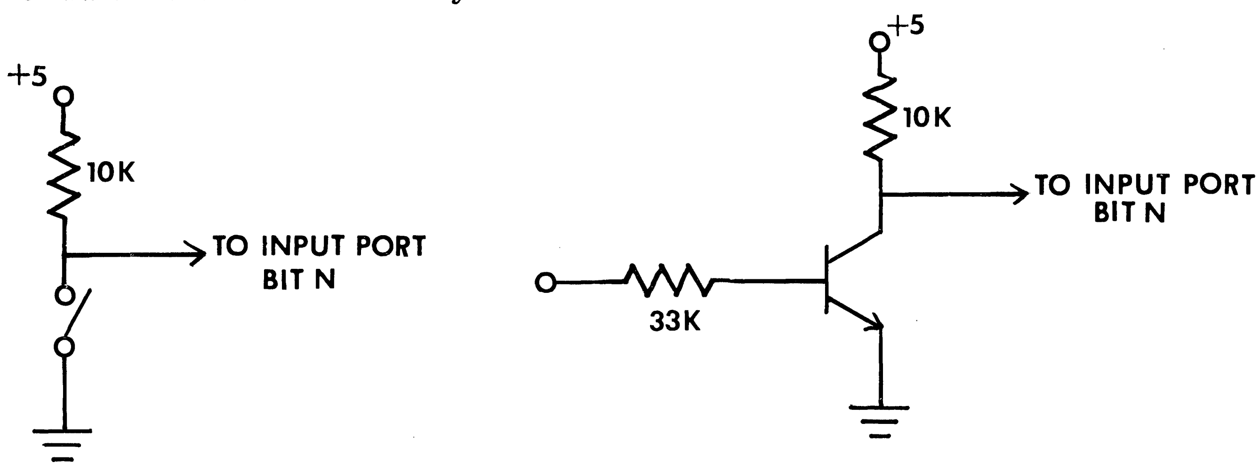
HARDWARE INTERFACING

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



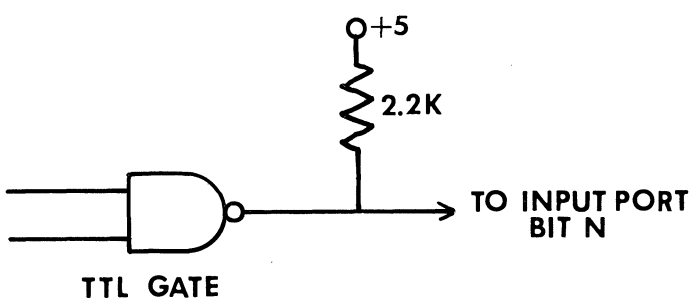
or switches or transistors may be used:



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

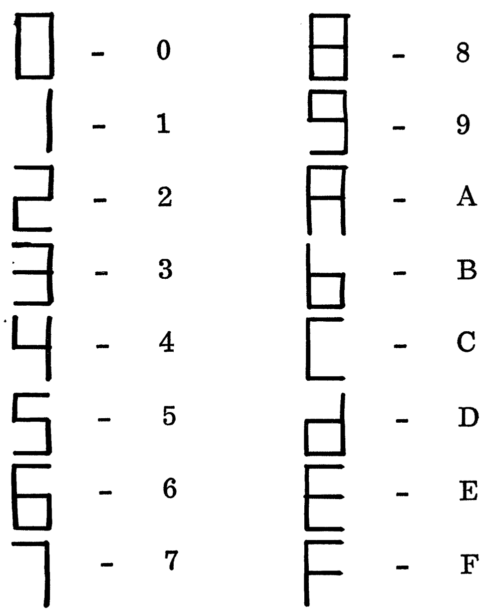
If the output of a TTL gate is being used to drive the input port, a pull-up resistor to supply must be provided:

C



DISPLAY

DISPLAY ADDRESS - x820



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.

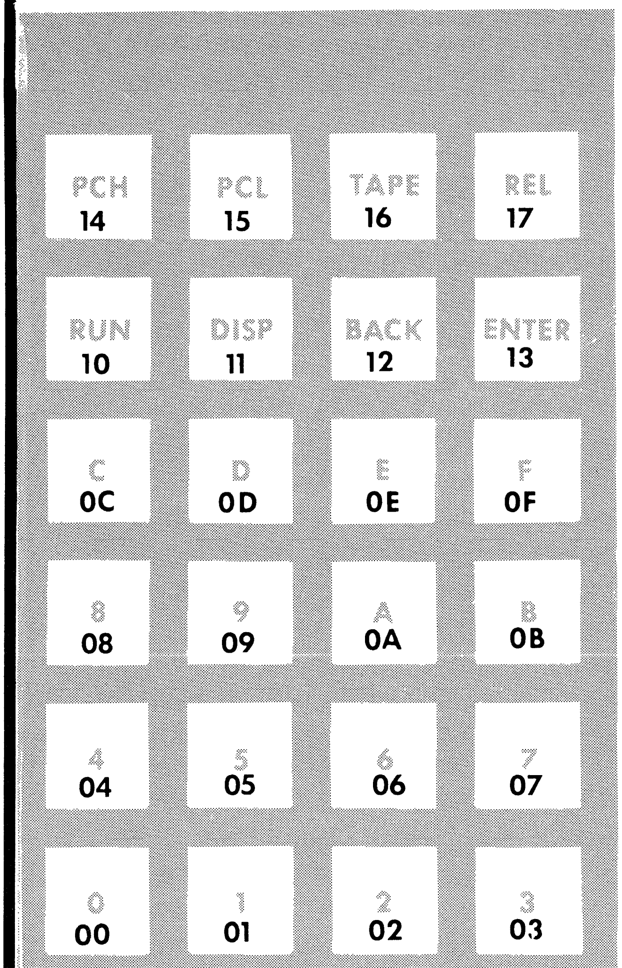
Like other peripheral ports, the displays are memory mapped and any instruction that writes to memory will operate them.

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.)

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.)

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 <u>still</u> 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, <u>not</u> 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)

$\overline{\text{STROBE}}$ - Address x880
$\overline{\text{DISPLAY}}$ - Address x820
$\overline{\text{OUTPORT}}$- Address x840
$\overline{\text{CASSETTE}}$-Address x9xx

DB7—	O 1	14 O	—DB6
DB5—	O	O	—DB4
DB3—	O	O	—DB2
DB1—	O	O	—DBØ
$\overline{\text{STROBE}}$—	O	O	—$\overline{\text{OUTPORT}}$
$\overline{\text{DISPLAY}}$—	O	O	—$\overline{\text{CASSETTE}}$
+5—	O 7	8 O	—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 ($\overline{\text{STROBE}}$, 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

$\overline{\text{STROBE}}$ - 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

$\overline{\text{DISPLAY}}$-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).

$\overline{\text{OUTPORT}}$-The low-true select line for the output port which lives at address x840 activates on write operations only.

$\overline{\text{CASSETTE}}$-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

$\overline{IRQ}$ — O[1]	J7	O — $\overline{RES}$
$\overline{NMI}$ — O		O — GND
ABØ — O		O — Ø2
AB1 — O		O — Ø2·R/W
AB2 — O		O — RAM R/W
AB3 — O		O — DBØ
AB4 — O		O — DB1

AB5 — O[1]	J8	O — DB2
AB6 — O		O — DB3
AB7 — O		O — DB4
AB8 — O		O — DB5
+5 — O		O — DB6
AB11 — O		O — DB7
AB9 — O		O — AB10

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

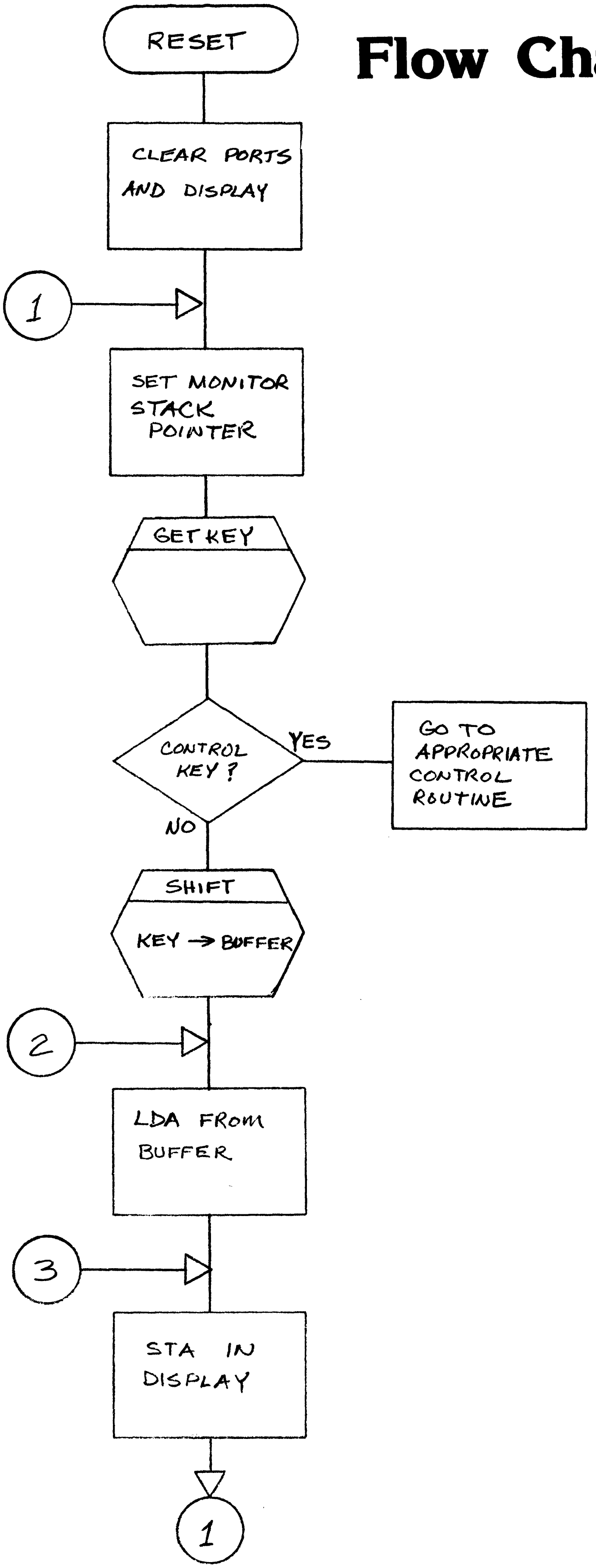
+5 — O[1]	O — RELAY 2
−9 — O	O — RELAY 1
EAR — O	O — MIC
EAR — O	O — GND

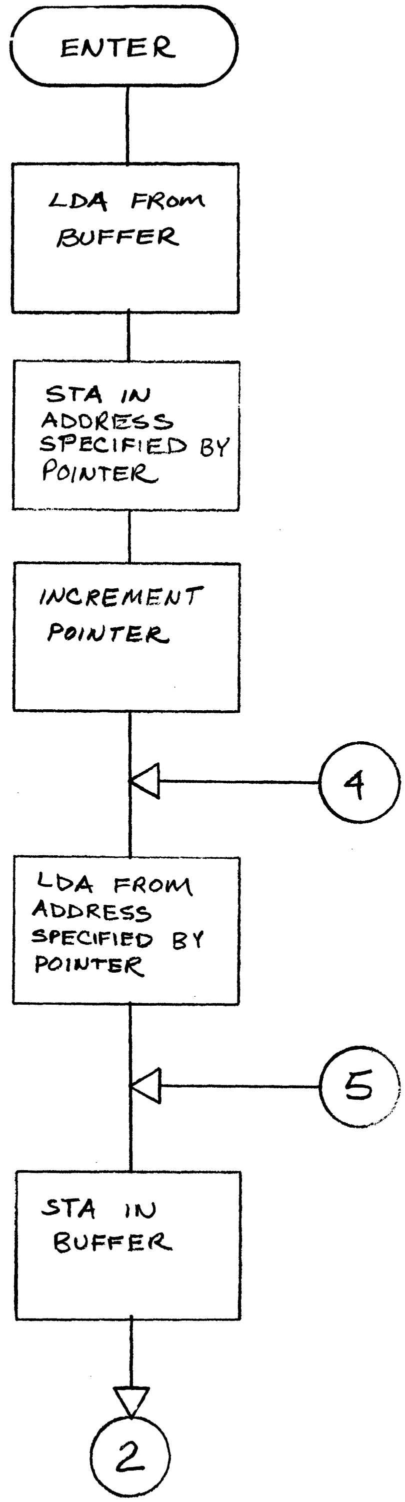
The cassette connector is used in conjunction with the CS-87 option to provide program and data-saving and loading from cassette recorder (see CS-87 Cassette option manual for operating details).

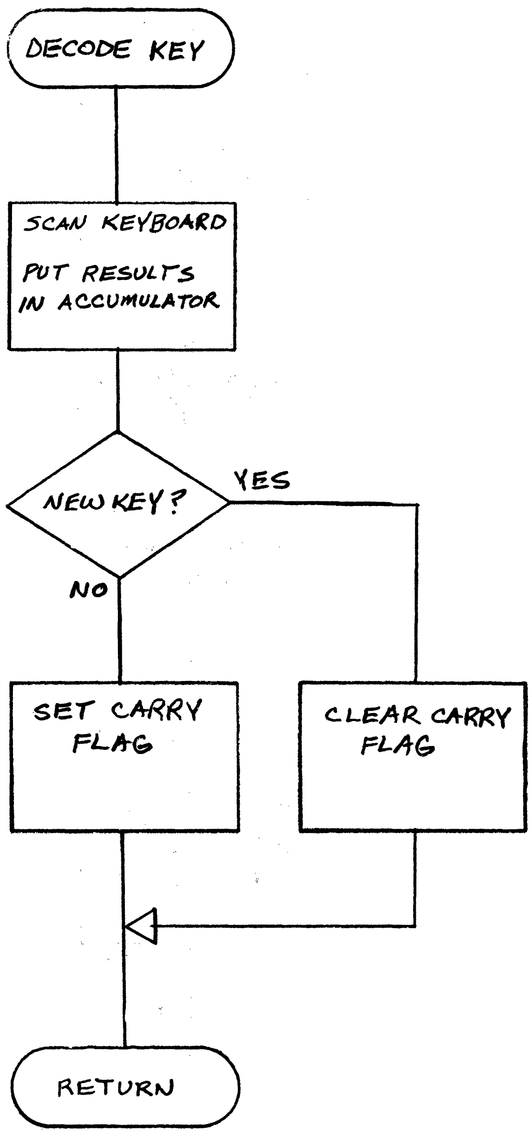
Additionally, this port and its corresponding components provides for a keyboard "beeper" which indicates activation of the control keys of the 8700/A Active Keyboard.

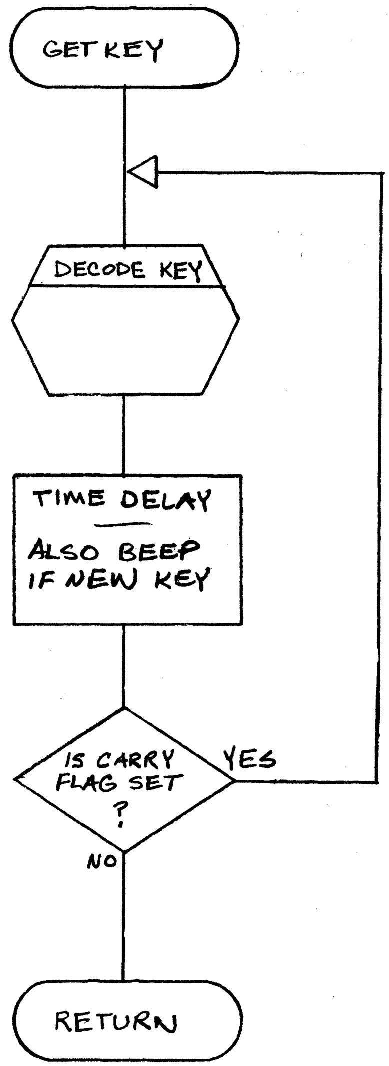
NOTES

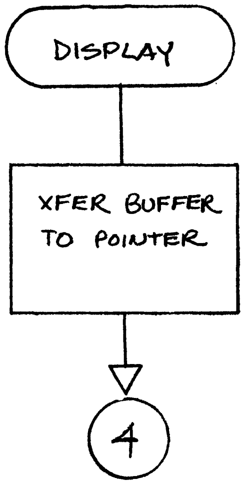
Flow Charts & Monitor Listing

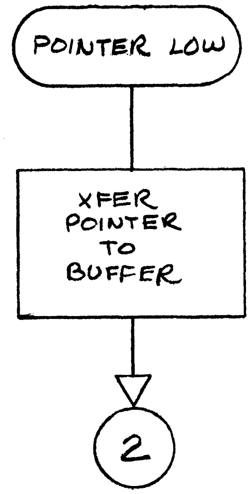


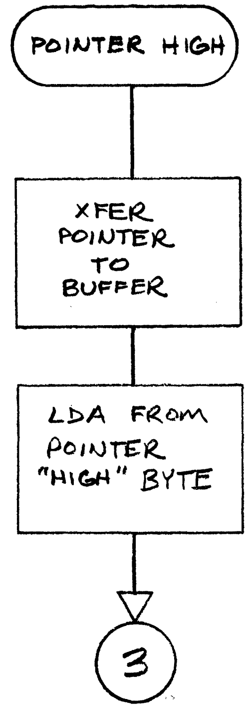


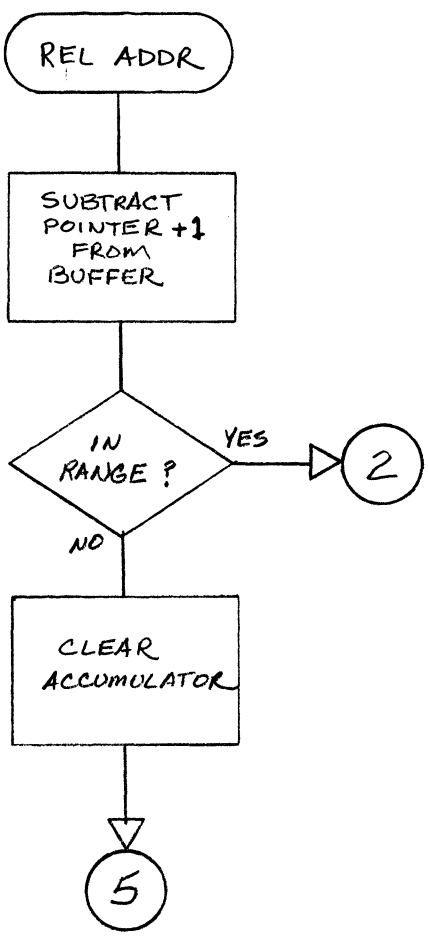

DECODE KEY
SCAN KEYBOARD
PUT RESULTS IN ACCUMULATOR
NEW KEY?
YES
NO
SET CARRY FLAG
CLEAR CARRY FLAG
RETURN

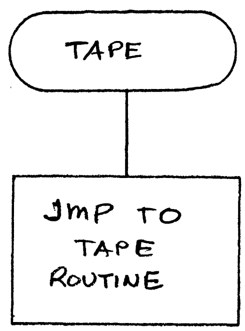

GET KEY
DECODE KEY
TIME DELAY
ALSO BEEP IF NEW KEY
IS CARRY FLAG SET ?
YES
NO
RETURN

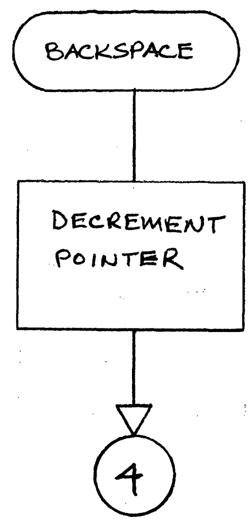

DISPLAY
XFER BUFFER TO POINTER
4

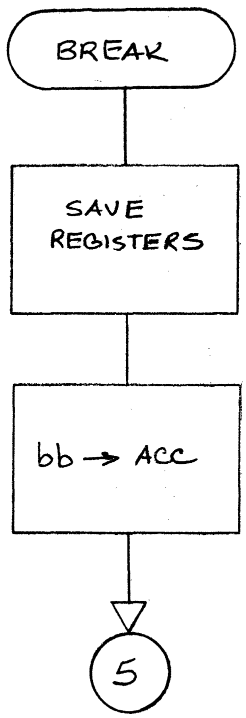

POINTER LOW
XFER POINTER TO BUFFER
2

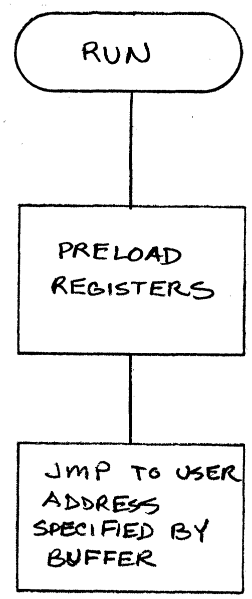

POINTER HIGH
XFER POINTER TO BUFFER
LDA FROM POINTER "HIGH" BYTE
3

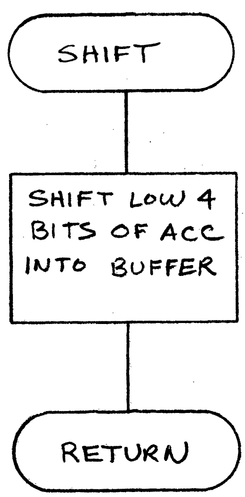

REL ADDR
SUBTRACT POINTER +1 FROM BUFFER
IN RANGE ?
YES
2
NO
CLEAR ACCUMULATOR
5


TAPE
JMP TO TAPE ROUTINE


BACKSPACE
DECREMENT POINTER
4


BREAK
SAVE REGISTERS
bb → ACC
5


RUN
PRELOAD REGISTERS
JMP TO USER ADDRESS SPECIFIED BY BUFFER


SHIFT
SHIFT LOW 4 BITS OF ACC INTO BUFFER
RETURN

PAiA INTERACTIVE EDITOR DEBUGGER (PIEBUG)

Monitor Listing

```
0200      ;
0200      ;
0200      ;
0200      ;
0200      ;
0200      ;
0200      ;    ************************************
0200      ;    *                                  *
0200      ;    *  PIEBUG              VERSION 1.0 *
0200      ;    *  PAIA INTERACTIVE EDITOR-DEBUGGER *
0200      ;    *  WRITTEN BY ROGER WALTON         *
0200      ;    *  COPYRIGHT 1977 BY PAIA          *
0200      ;    *    ELECTRONICS, INC.             *
0200      ;    *                                  *
0200      ;    ************************************
0200      ;
0200      ;
0200      ;
0200             *=$0F00
0F00      ;
0F00      KEY    =$0800          ;BASE ADDR OF KEY PORTS
0F00      TEMP   =$EE            ;TEMPORARY STORAGE
0F00      LASTKE =$F8            ;PREVIOUS KEY DECODED
0F00      BUFFER =$F0            ;KEY ENTRY BUFFER
0F00      DISP   =$0820          ;LED DISPLAY
0F00      MSTACK =$ED            ;MONITOR STACK POINTER
0F00      PNTER  =$F6            ;16 BIT ADDR POINTER
0F00      TAPE1  =$0E00          ;START OF TAPE SYSTEM
0F00      CASS   =$0900          ;CASSETTE PORT
0F00      ;
0F00      ;
0F00      ACC    =$F9            ;REG STORAGE
0F00      YREG   =$FA            ;     "
0F00      XREG   =$FB            ;     "
0F00      PC     =$FC            ;     "
0F00      STACKP =$FE            ;     "
0F00      PREG   =$FF            ;REG STORAGE
0F00      ;
0F00      ;
```

```
0F00                  ;      DECODE KEY SUBROUTINE
0F00                  ;      THIS SUB SCANS THE ENTIRE KEYBOARD AND
0F00                  ;      RETURNS WITH DECODED KEY VALUE IN A AND Y.
0F00                  ;      CARRY IS CLEAR IF NEW KEY.  X IS
0F00                  ;      DESTROYED.  $18 IS "NO KEY" CODE.
0F00                  ;
0F00   A0 00          DECODE LDY #0          ;CLEAR RESULT REG
0F02   A2 21                 LDX #$21        ;X IS PORT REG
0F04   A9 01          LOOP   LDA #1
0F06   85 EE                 STA TEMP        ;SET UP MASK
0F08   BD 00 08       NEXT   LDA KEY,X       ;READ CURRENT KEY PORT
0F0B   25 EE                 AND TEMP        ;USE MASK TO SELECT KEY
0F0D   D0 0A                 BNE RESULT      ;BRANCH IF KEY DOWN
0F0F   C8                    INY             ;SET RESULT TO NEXT KEY
0F10   06 EE                 ASL TEMP        ;SHIFT MASK TO NEXT KEY
0F12   90 F4                 BCC NEXT        ;BR IF MORE KEYS ON PORT
0F14   8A                    TXA
0F15   0A                    ASL A           ;SELECT NEXT PORT
0F16   AA                    TAX
0F17   90 EB                 BCC LOOP        ;BRANCH IF NOT LAST PORT
0F19   C4 F8          RESULT CPY LASTKE      ;CLEAR CARRY IF NEW KEY
0F1B   84 F8                 STY LASTKE      ;UPDATE LASTKEY
0F1D   98                    TYA             ;MOVE KEY TO ACC
0F1E   60                    RTS             ;RETURN
0F1F                  ;
0F1F                  ;
0F1F                  ;
0F1F                  ;      GETKEY SUBROUTINE
0F1F                  ;      THIS SUB WAITS FOR A NEW KEY TO BE
0F1F                  ;      TOUCHED AND THEN RETURNS WITH THE
0F1F                  ;      KEY VALUE IN THE ACCUMULATOR.
0F1F                  ;      X AND Y ARE CLEARED.
0F1F                  ;
0F1F                  ;      BEEP SUBROUTINE (EMBEDDED IN GETKEY SUB)
0F1F                  ;      THIS SUB PRODUCES A SHORT BEEP AT
0F1F                  ;      THE CASSETTE PORT.  CARRY MUST BE
0F1F                  ;      CLEAR BEFORE ENTERING.  X AND Y
0F1F                  ;      ARE CLEARED.
0F1F                  ;
0F1F   20 00 0F       GETKEY JSR DECODE      ;GET A KEY
0F22   A2 14          BEEP   LDX #20         ;ENTER HERE FOR BEEP SUB
0F24   A0 3F          NXTX   LDY #$3F
0F26   B0 03          DELAY  BCS DLY         ;SKIP TONE IF CARRY SET
0F28   8C 00 09              STY CASS        ;GENERATE TONE
0F2B   88             DLY    DEY             ;DELAY
0F2C   D0 F8                 BNE DELAY
0F2E   CA                    DEX             ;DELAY SOME MORE
0F2F   D0 F3                 BNE NXTX        ;NEXT X
0F31   B0 EC                 BCS GETKEY      ;BRANCH IF NOT NEW KEY
0F33   60                    RTS             ;RETURN
0F34                  ;
0F34                  ;
0F34                  ;
0F34                  ;
```

```
0F34                    ;       SHIFT BUFFER SUBROUTINE
0F34                    ;       THIS SUB SHIFTS THE LOWER 4 BITS OF
0F34                    ;       THE ACCUMULATOR INTO THE LEAST
0F34                    ;       SIGNIFICANT POSITION OF BUFFER.  THE
0F34                    ;       ENTIRE BUFFER IS SHIFTED 4 TIMES AND
0F34                    ;       THE MOST SIGNIFICANT 4 BITS ARE LOST.
0F34                    ;       X AND Y ARE CLEARED.  IF ON RETURN,
0F34                    ;       A SINGLE "ROL A" IS PERFORMED,
0F34                    ;       THE LOWER 4 BITS OF THE ACCUMULATOR
0F34                    ;       WILL CONTAIN THE 4 BITS THAT WERE
0F34                    ;       SHIFTED OUT OF BUFFER.
0F34                    ;
0F34  0A                SHIFT  ASL A              ;SHIFT KEY INFORMATION
0F35  0A                       ASL A              ;TO UPPER 4 BITS OF ACC
0F36  0A                       ASL A
0F37  0A                       ASL A
0F38  A0 04                    LDY #4
0F3A  2A                ROTATE ROL A              ;SHIFT BIT TO CARRY
0F3B  A2 FA                    LDX #$FA           ;WRAP AROUND TO $F0
0F3D  36 F6             ROTNXT ROL BUFFER+6,X     ;CARRY TO BUFFER TO CARRY
0F3F  E8                       INX                ;AND SO ON
0F40  D0 FB                    BNE ROTNXT         ;UNTIL END OF BUFFER
0F42  88                       DEY                ;DONE 4 BITS?
0F43  D0 F5                    BNE ROTATE         ;BRANCH IF NOT
0F45  60                       RTS                ;RETURN
0F46                    ;
0F46                    ;
0F46                    ;       RESET ENTRY POINT
0F46                    ;
0F46  A9 00             RESET  LDA #0
0F48  8D E0 08                 STA $08E0          ;CLEAR DISPLAY AND PORTS
0F4B  F0 08                    BEQ COMAND         ;BRANCH ALWAYS
0F4D                    ;
0F4D                    ;
0F4D                    ;
0F4D  20 34 0F          SHFTD  JSR SHIFT          ;SHIFT KEY INTO BUFFER
0F50  A5 F0             DSPBUF LDA BUFFER         ;GET BUFFER
0F52  8D 20 08          SEE    STA DISP           ;UPDATE DISPLAY
0F55                    ;
0F55  A6 ED             COMAND LDX MSTACK
0F57  9A                       TXS                ;SET MONITOR STACK
0F58  20 1F 0F                 JSR GETKEY         ;WAIT FOR KEY
0F5B  C9 10                    CMP #$10           ;IS IT CONTROL KEY
0F5D  90 EE                    BCC SHFTD          ;BRANCH IF NOT
0F5F  A8                       TAY                ;CONTROL KEY INTO Y
0F60  BE E2 0F                 LDX TABLE-16,Y     ;GET COMMAND ADDR LOW
0F63  86 EE                    STX TEMP           ;SAVE IT
0F65  A2 FF                    LDX #$FF           ;GET COMMAND ADDR HIGH
0F67  86 EF                    STX TEMP+1         ;ASSEMBLE COMMAND ADDR
0F69  E8                       INX                ;CLR X
0F6A  6C EE 00                 JMP (TEMP)         ;EXECUTE COMMAND
0F6D                    ;
0F6D                    ;
```

```
1560  0F6D  18           PHIGH  CLC
1570  0F6E  A5 F6        PLOW   LDA PNTER         ;MOVE POINTER TO BUFFER
1580  0F70  85 F0               STA BUFFER
1590  0F72  A5 F7               LDA PNTER+1
1600  0F74  85 F1               STA BUFFER+1
1610  0F76  B0 D8               BCS DSPBUF        ;BRANCH IF POINTER LOW
1620  0F78  90 D8               BCC SEE           ;BRANCH IF POINTER HIGH
1630  0F7A               ;
1640  0F7A               ;
1650  0F7A  A5 F0        DISPLA LDA BUFFER        ;MOVE BUFFER TO POINTER
1660  0F7C  85 F6               STA PNTER
1670  0F7E  A5 F1               LDA BUFFER+1
1680  0F80  85 F7               STA PNTER+1
1690  0F82  B0 14               BCS LOAD          ;BRANCH ALWAYS
1700  0F84               ;
1710  0F84               ;
1720  0F84  A5 F6        BACKSP LDA PNTER         ;DEC 16 BIT POINTER
1730  0F86  D0 02               BNE SKIP          ;BRANCH IF NO BORROW
1740  0F88  C6 F7               DEC PNTER+1
1750  0F8A  C6 F6        SKIP   DEC PNTER
1760  0F8C  B0 0A               BCS LOAD          ;BRANCH ALWAYS
1770  0F8E               ;
1780  0F8E               ;
1790  0F8E  A5 F0        ENTER  LDA BUFFER        ;GET BYTE IN BUFFER
1800  0F90  81 F6               STA (PNTER,X)     ;STORE IT IN ACTIVE CELL
1810  0F92  E6 F6               INC PNTER         ;INC 16 BIT POINTER
1820  0F94  D0 02               BNE LOAD          ;BRANCH IF NO CARRY
1830  0F96  E6 F7               INC PNTER+1
1840  0F98  A1 F6        LOAD   LDA (PNTER,X)     ;GET BYTE IN ACTIVE CELL
1850  0F9A  85 F0        STABUF STA BUFFER        ;STORE IT IN BUFFER
1860  0F9C  B0 B2               BCS DSPBUF        ;BRANCH ALWAYS
1870  0F9E               ;
1880  0F9E               ;
1890  0F9E  D8           RELADR CLD
1900  0F9F  18                  CLC               ;THIS ADDS 1 TO POINTER
1910  0FA0  A5 F0               LDA BUFFER        ;GET BUFFER LOW
1920  0FA2  E5 F6               SBC PNTER         ;SUBTRACT POINTER LOW + 1
1930  0FA4  85 F0               STA BUFFER        ;SAVE RESULTS
1940  0FA6  A5 F1               LDA BUFFER+1      ;GET BUFFER HIGH
1950  0FA8  E5 F7               SBC PNTER+1       ;SUBTRACT POINTER HIGH
1960  0FAA  A8                  TAY               ;SAVE RESULTS IN Y
1970  0FAB  A5 F0               LDA BUFFER        ;GET RESULTS LOW
1980  0FAD  B0 08               BCS POS           ;BR IF TOTAL RESULT POS
1990  0FAF  10 0A               BPL BAD           ;BR IF RESULT LOW POS
2000  0FB1  C8                  INY               ;INC RESULT HIGH
2010  0FB2  98           CHK    TYA               ;CHECK RESULT HIGH
2020  0FB3  D0 06               BNE BAD           ;BR IF NOT ZERO
2030  0FB5  F0 99               BEQ DSPBUF        ;BR ALWAYS, DISP REL ADDR
2040  0FB7  30 02        POS    BMI BAD           ;BR IF RESULT LOW NEG
2050  0FB9  10 F7               BPL CHK           ;BR ALWAYS
2060  0FBB  8A           BAD    TXA               ;CLEAR ACC
2070  0FBC  38                  SEC
2080  0FBD  B0 DB               BCS STABUF        ;BRANCH ALWAYS
2090  0FBF               ;
2100  0FBF               ;
2110  0FBF  EA                  NOP
2120  0FC0               ;
```

```
0FC0                 ;
0FC0                 ;
0FC0                 ;      BREAK ROUTINE ENTRY POINT
0FC0                 ;
0FC0  85 F9          BREAK  STA ACC          ;SAVE ACCUMULATOR
0FC2  84 FA                 STY YREG         ;SAVE Y
0FC4  86 FB                 STX XREG         ;SAVE X
0FC6  68                    PLA              ;GET STATUS REG
0FC7  85 FF                 STA PREG         ;SAVE IT
0FC9  68                    PLA              ;GET PC LOW
0FCA  D8                    CLD
0FCB  38                    SEC
0FCC  E9 02                 SBC #2           ;CORRECT PC LOW
0FCE  85 FC                 STA PC           ;SAVE IT
0FD0  68                    PLA              ;GET PC HIGH
0FD1  E9 00                 SBC #0           ;SUBTRACT CARRY
0FD3  85 FD                 STA PC+1         ;SAVE IT
0FD5  BA                    TSX              ;GET USER STACK POINTER
0FD6  86 FE                 STX STACKP       ;SAVE IT
0FD8  A9 BB                 LDA #$BB         ;BREAK INDICATION
0FDA  B0 BE                 BCS STABUF       ;BRANCH ALWAYS
0FDC                 ;
0FDC                 ;
0FDC  A6 FE          RUN    LDX STACKP       ;GET USER STACK POINTER
0FDE  9A                    TXS              ;INIT STACK
0FDF  A5 F1                 LDA BUFFER+1     ;GET PC HIGH
0FE1  48                    PHA              ;PUT IT ON STACK
0FE2  A5 F0                 LDA BUFFER       ;GET PC LOW
0FE4  48                    PHA              ;PUT IT ON STACK
0FE5  A5 FF                 LDA PREG         ;GET STATUS REG
0FE7  48                    PHA              ;PUT IT ON STACK
0FE8  A6 FB                 LDX XREG         ;RESTORE X
0FEA  A4 FA                 LDY YREG         ;RESTORE Y
0FEC  A5 F9                 LDA ACC          ;RESTORE ACCUMULATOR
0FEE  40                    RTI              ;RESTORE PC & STATUS REG
0FEF                 ;                         FROM STACK AND EXECUTE
0FEF                 ;                         USER'S PROGRAM
0FEF                 ;
0FEF                 ;
0FEF  4C 00 0E       TAPE   JMP TAPE1        ;EXECUTE TAPE OPTION
0FF2                 ;
0FF2                 ;
```

```
0FF2                  ;      COMMAND ADDRESS TABLE
0FF2                  ;      STORES LOW BYTE ONLY OF ENTRY
0FF2                  ;      ADDRESS FOR EACH COMMAND
0FF2                  ;
0FF2  DC 0F           TABLE   .WORD RUN
0FF4                          *=*-1
0FF3  7A 0F                   .WORD DISPLA
0FF5                          *=*-1
0FF4  84 0F                   .WORD BACKSP
0FF6                          *=*-1
0FF5  8E 0F                   .WORD ENTER
0FF7                          *=*-1
0FF6  6D 0F                   .WORD PHIGH
0FF8                          *=*-1
0FF7  6E 0F                   .WORD PLOW
0FF9                          *=*-1
0FF8  EF 0F                   .WORD TAPE
0FFA                          *=*-1
0FF9  9E 0F                   .WORD RELADR
0FFB                          *=*-1
0FFA                  ;
0FFA                  ;
0FFA  03 00                   .WORD $0003        ;NMI VECTOR
0FFC  46 0F                   .WORD RESET        ;RESET VECTOR
0FFE  00 00                   .WORD $0000        ;IRQ VECTOR
1000                  ;
1000                  ;
1000                          .END
```

ERRORS = 0000

SYMBOL TABLE

RESULT	0F19	DLY	0F2B	COMAND	0F55	LOAD	0F98
SKIP	0F8A	POS	0FB7	BAD	0FBB	TABLE	0FF2
KEY	0800	TEMP	00EE	LASTKE	00F8	BUFFER	00F0
DISP	0820	MSTACK	00ED	PNTER	00F6	TAPE1	0E00
CASS	0900	ACC	00F9	YREG	00FA	XREG	00FB
PC	00FC	STACKP	00FE	PREG	00FF	DECODE	0F00
LOOP	0F04	NEXT	0F08	GETKEY	0F1F	BEEP	0F22
NXTX	0F24	DELAY	0F26	SHIFT	0F34	ROTATE	0F3A
ROTNXT	0F3D	RESET	0F46	SHFTD	0F4D	DSPBUF	0F50
SEE	0F52	PHIGH	0F6D	PLOW	0F6E	DISPLA	0F7A
BACKSP	0F84	ENTER	0F8E	STABUF	0F9A	RELADR	0F9E
CHK	0FB2	BREAK	0FC0	RUN	0FDC	TAPE	0FEF

SCHEMATICS

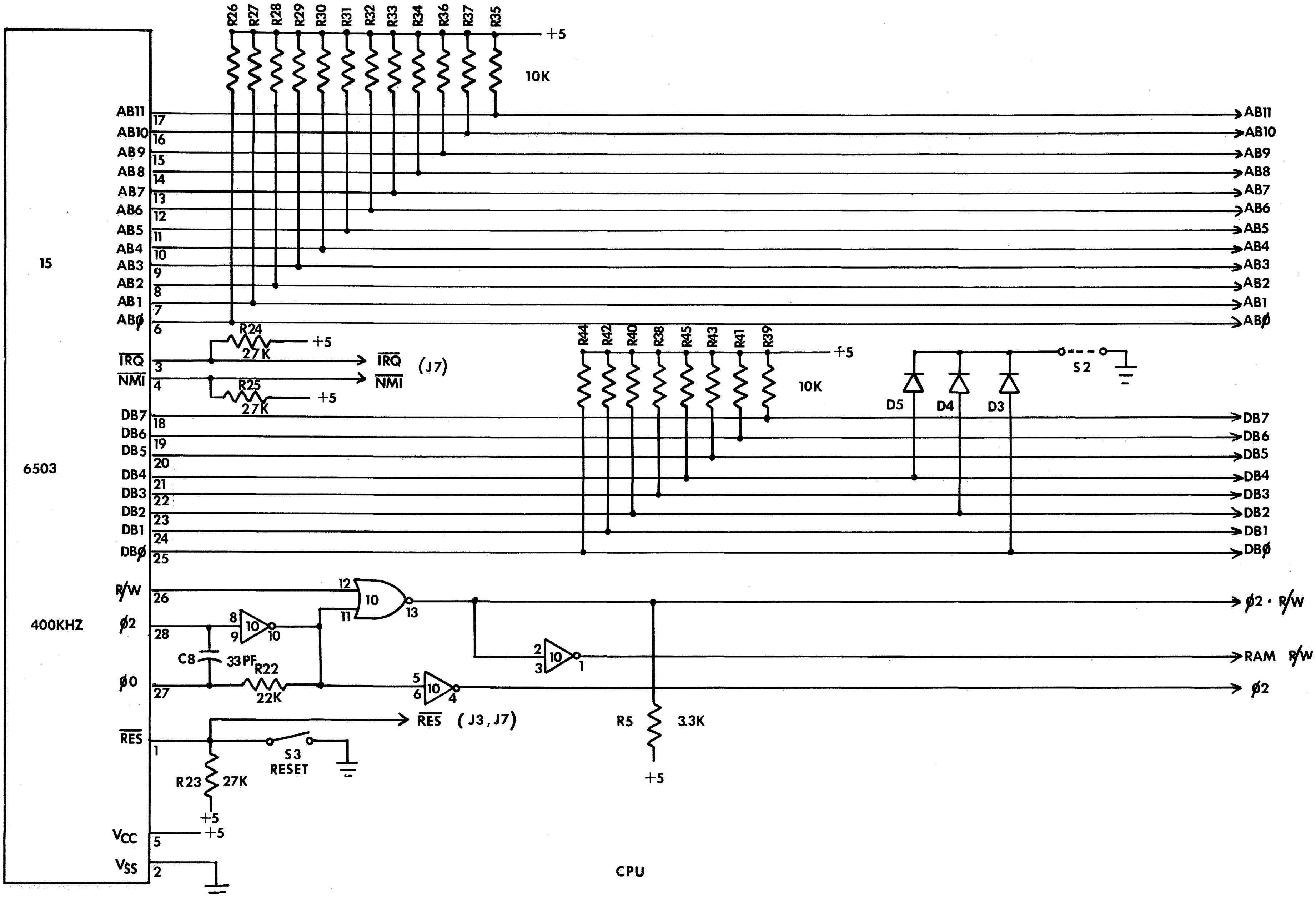
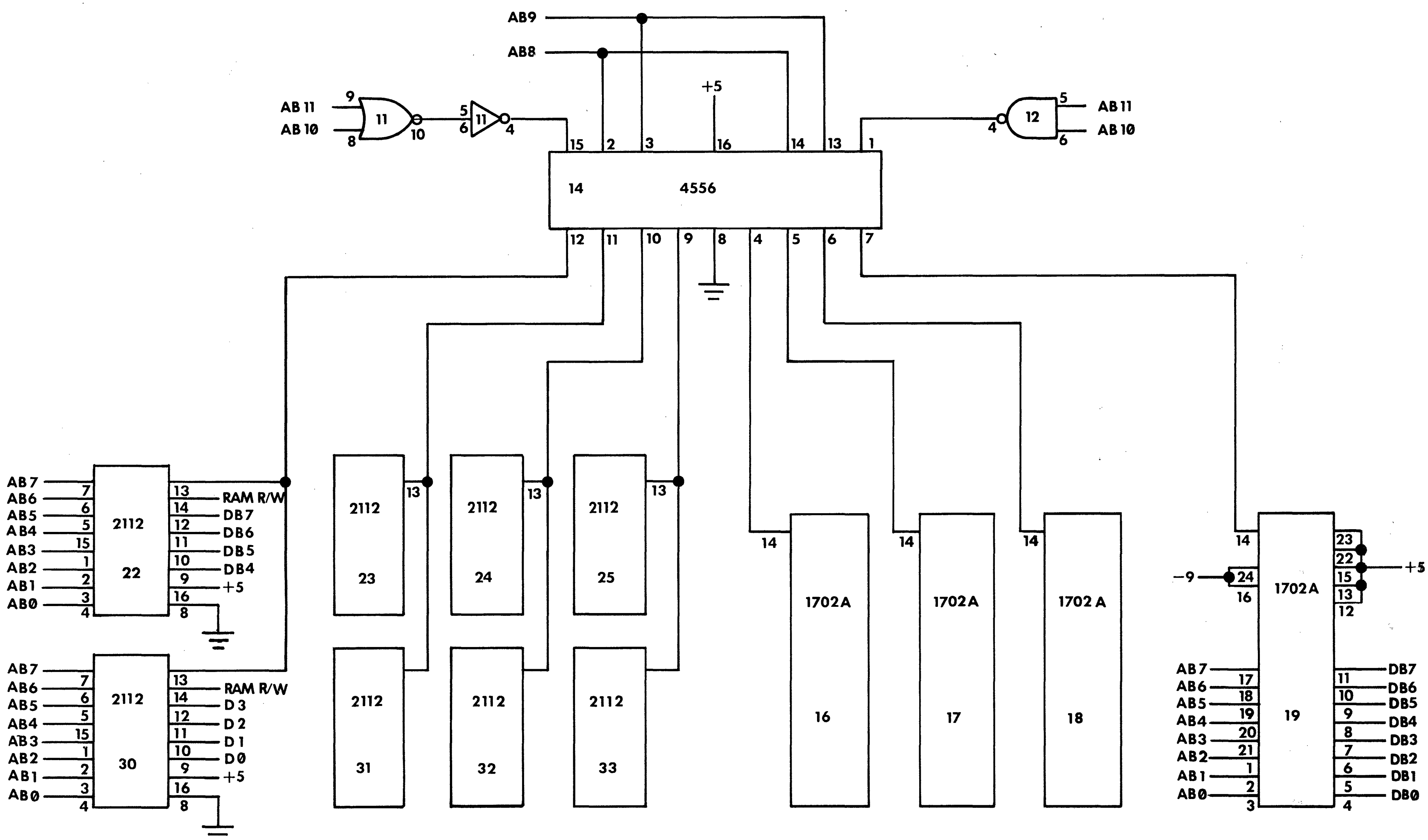

6503
15
400KHZ
AB11
AB10
AB9
AB8
AB7
AB6
AB5
AB4
AB3
AB2
AB1
ABØ
IRQ
NMI
DB7
DB6
DB5
DB4
DB3
DB2
DB1
DBØ
R/W
Ø2
Ø0
RES
VCC
VSS
R26 R27 R28 R29 R30 R31 R32 R33 R34 R36 R37 R35
+5
10K
R24 27K
R25 27K
(J7)
R44 R42 R40 R38 R45 R43 R41 R39
D5 D4 D3
S2
C8 33PF
R22 22K
Ø2 • R/W
RAM R/W
RES (J3, J7)
S3 RESET
R23 27K
R5 3.3K
CPU

FIGURE 1



RAM — PROM
MEMORY

FIGURE 2

I/O DECODING

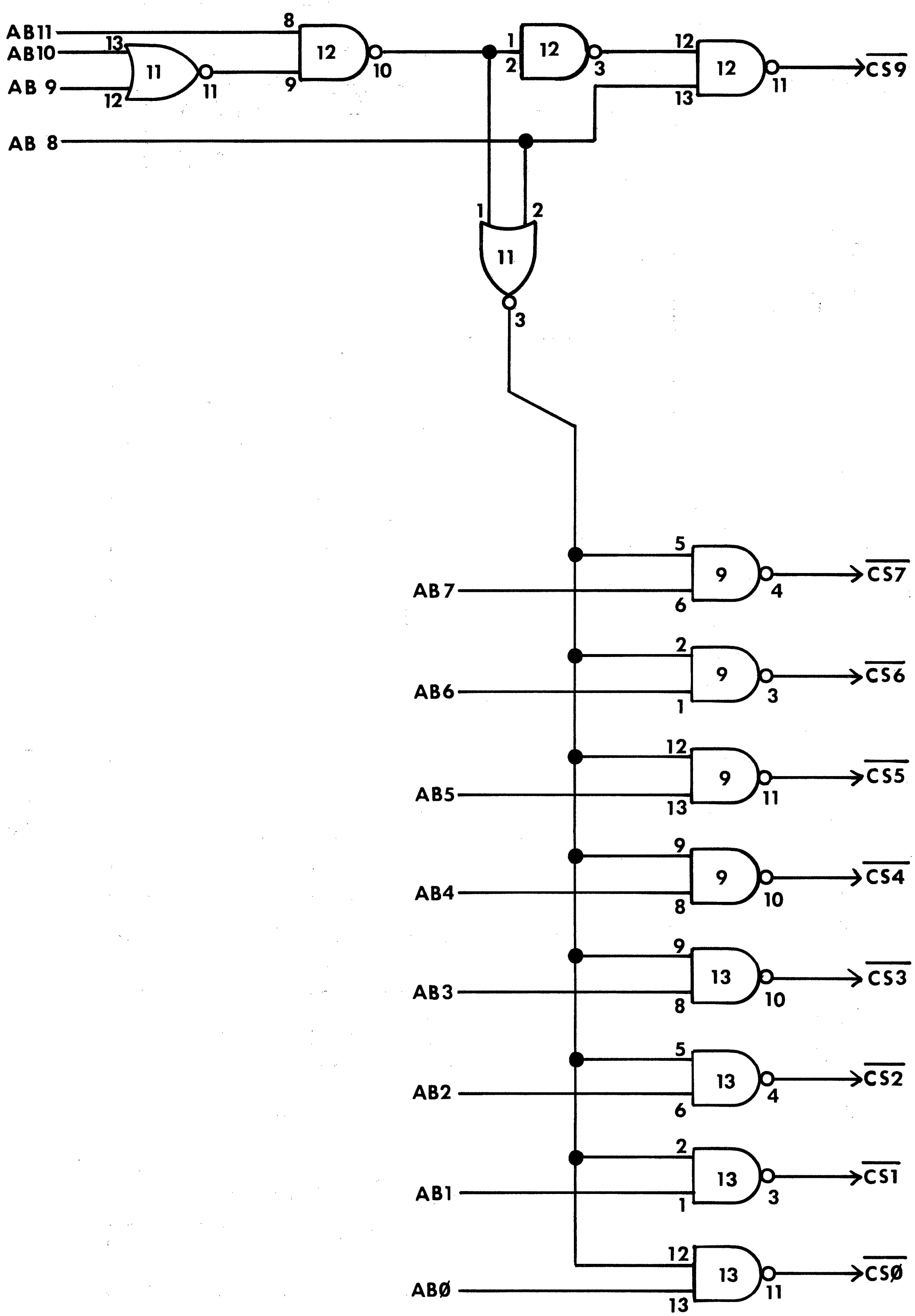


FIGURE 3

$\overline{\text{STROBE}}$ SELECT

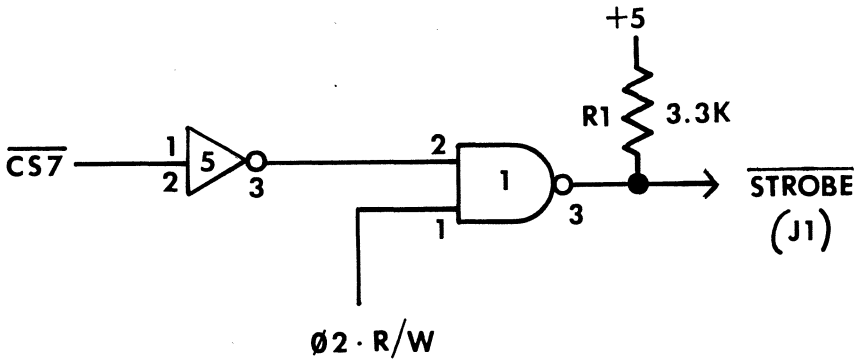


FIGURE 4

TRANSISTOR BUSS

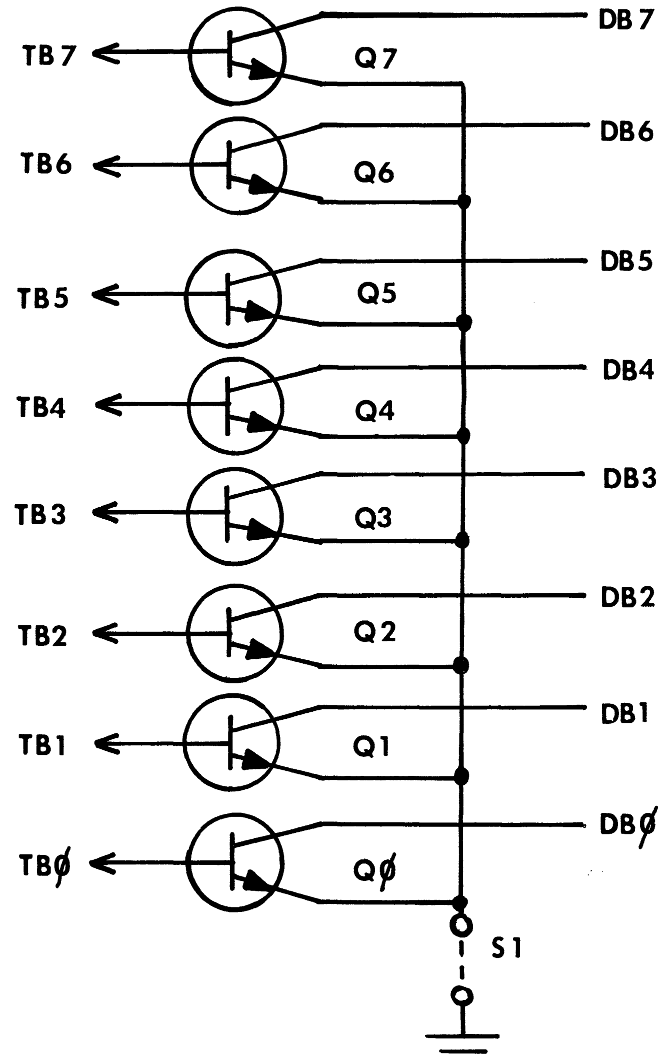
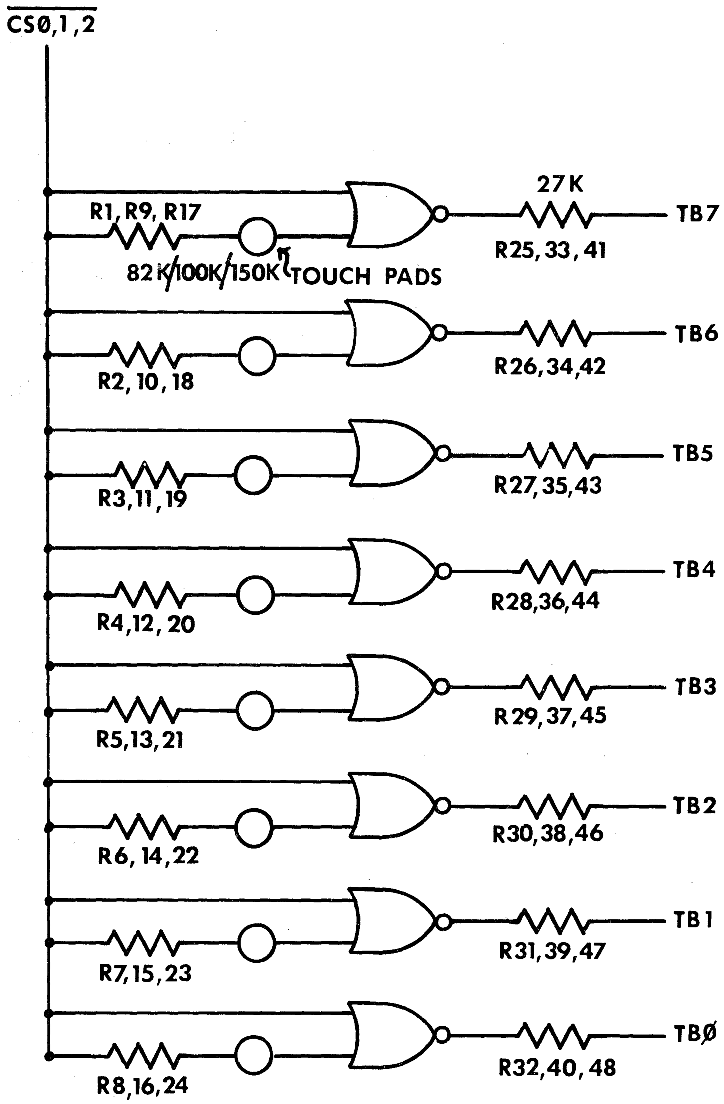
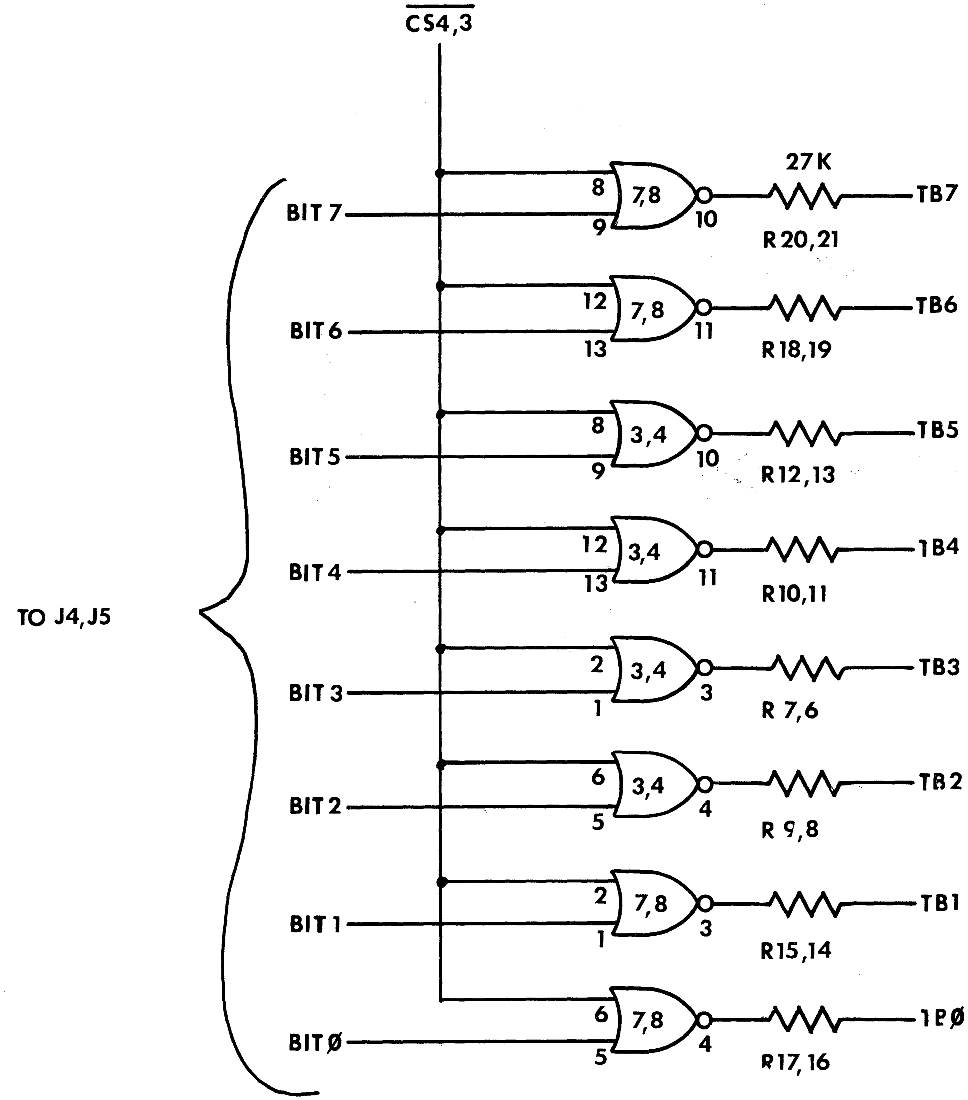


FIGURE 5



KEYBOARD

FIGURE 6



INPUT PORTS

3 AND 7, 4 AND 8
PORT 1 PORT 2

FIGURE 7

DISPLAYS

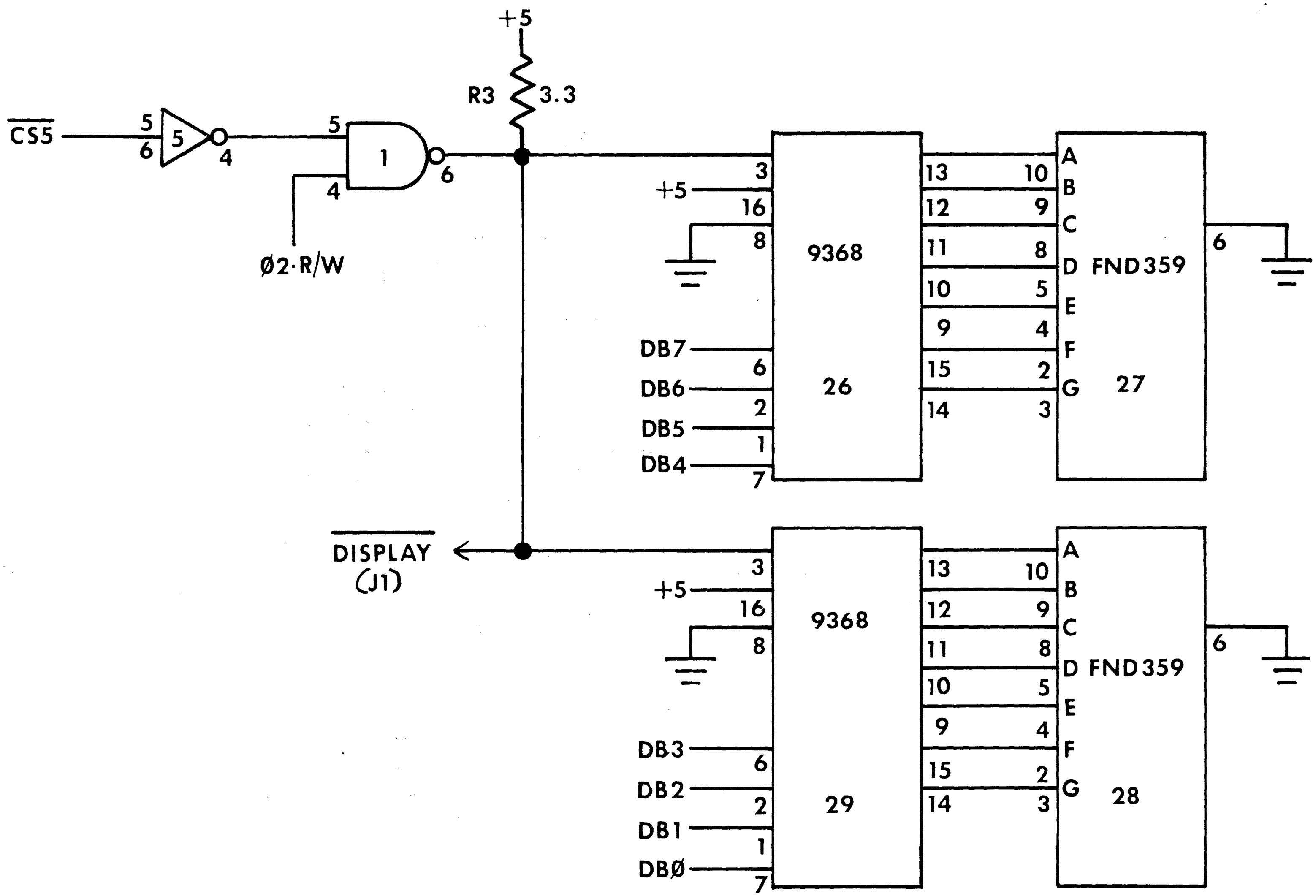
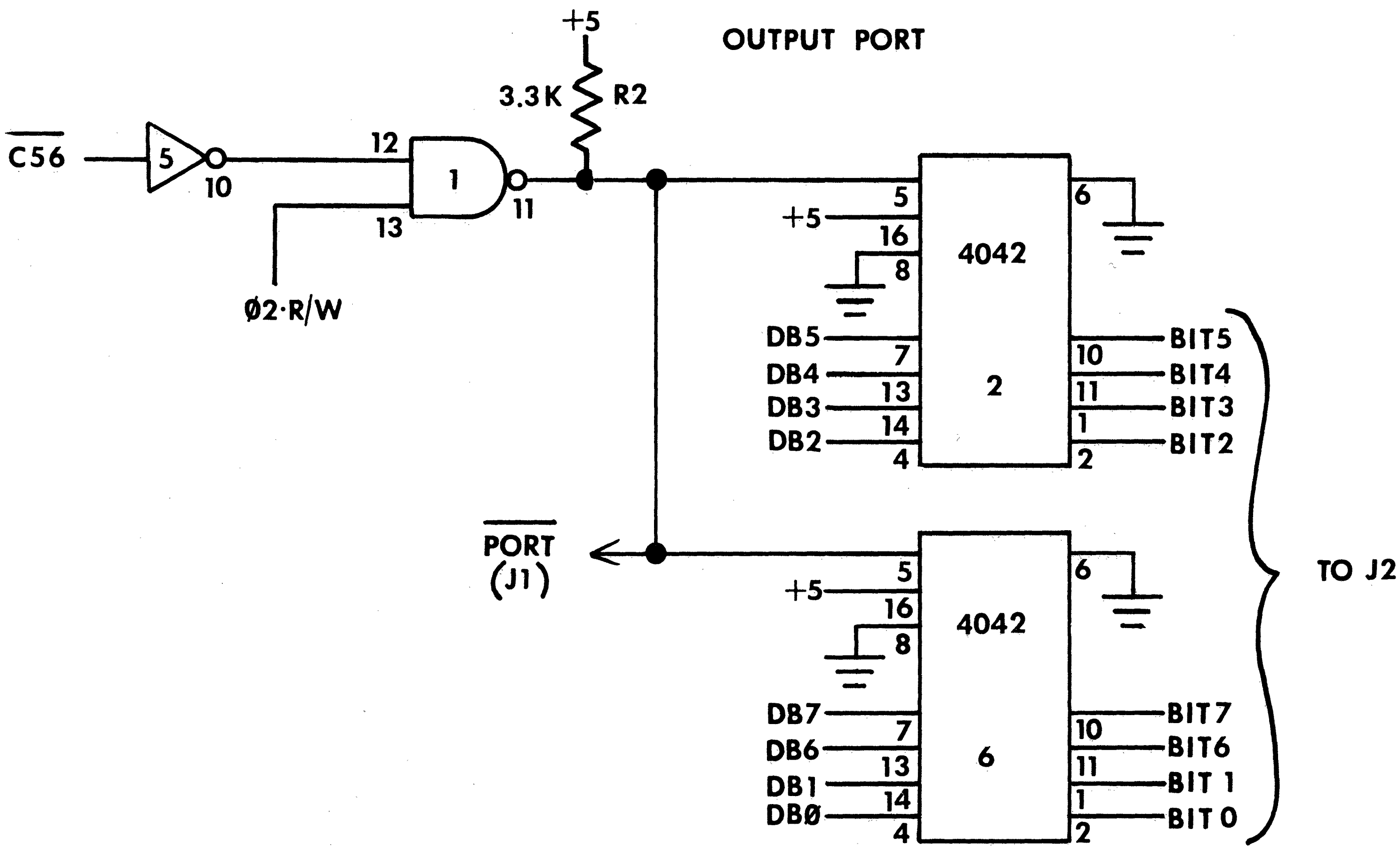


FIGURE 8



OUTPUT PORT
+5
3.3K
R2
C56
5
10
12
13
1
11
Ø2·R/W
PORT
(J1)
+5
4042
2
5
16
8
6
DB5
DB4
DB3
DB2
7
13
14
4
10
11
1
2
BIT5
BIT4
BIT3
BIT2
TO J2
+5
4042
6
DB7
DB6
DB1
DBØ
BIT7
BIT6
BIT 1
BIT 0

FIGURE 9

The PAiA 8700 Self-Test Micro-Diagnostic T.M.

There is a significant test feature built into the PAIA 8700 which, while simple in concept, provides an exceptionally powerful tool for spotting a number of potential faults associated with the Grand Buss architecture common to micro-computers. Two small circuitry details are involved in implementing this feature:

1) A means by which devices connected to the CPU's data buss may be disconnected and allowed to float.
2) A means by which a no-operation (NOP) instruction is forced onto the data buss.

Together, these two things cause a properly assembled and functioning 8700 board to operate in a very special manner.

The processor, on being reset, will fetch the first instruction from the memory location specified by the reset vector. Since all sources of data have been isolated from the data buss, the only source of instructions to the processor is from the combination of the data buss pull-up resistors R39 - R44 and the three diodes D3 - D5. The diodes clamp data buss lines D0, D2, and D4 to ground producing the binary pattern 11101010 (EA in hex) on the data buss. EA is a NOP instruction.

The processor's response to a NOP is to increment the address buss to the next address and fetch the instruction that it finds there, which is of course again a NOP. The address lines increment again and fetch the NOP, etc.

The overall result is that the address lines (all 12 of them) count in a normal binary sequence. This in turn allows for easy checking of the address lines which are operating in an easily verified and predictable manner as well as exercising all of the address decoding circuitry, making for easy checking of the various chip select lines to output ports, memory locations, etc. to see that this portion of the circuitry is operating properly.

Using the Self-Test

1) Remove all RAM (IC22 -IC25; IC30 - IC33) and ROM (IC16 - IC19) from their sockets.

2) Close the circuit board jumper S2 either by putting it in place, or, if already installed but severed, by soldering the cut ends together. This step ties the cathodes of the three diodes D3 - D5 to ground causing them to forward bias and hold the data lines D0, D2 and D4 low.

3) If the jumper S1 is in place, cut it so that no connection is made and isolate the two ends from one another. Also, tie the end of the jumper designated by the arrow on the circuit board to the+5 volt power supply line.A clip lead may be used here and the best place to pick up the+5 volts is at the left end of R5. These steps isolate the data buss by breaking the emitter leads of the transistors Q0 - Q7 and assures isolation by reverse-biasing these devices.

4) Apply power to the processor. And note that since only the ROMs require the -9 volt supply, this voltage does not have to be provided for these tests. On the other hand, it won't hurt to have it there either - whichever is easier. When the power is applied the displays should immediately light with some random digits. This is of course a quick check that the +5 volt supply is active and that there is not a direct short across the supply lines somewhere. The 9368 Display Drivers will quickly become uncomfortably warm to the touch. This is normal.

5) Reset the CPU by using a clip lead or other temporary jumper to momentarily ground the RESET line. For the purposes of these tests the RESET line is most easily accessed at pin 14 of the expansion connector J7 and ground can be picked up at the circuit board jumper S2. Since some malfunctions can cause the processor to "lock-up" (recieve an instruction that causes paralysis of the address and data busses - who knows what it's up to internally) it would be wise to have your temporary RESET switch handy during the entire procedure.

6) Check the Ø2 clock signal at pin 12 of the expansion connector J7 to see that it is:

a) present
b) swinging between essentially +5 v. and ground
c) has a period of approximately 2.5 micro-seconds ± 20%
d) has a duty factor somewhere between 30% and 70% - exact duty factor is not critical

7) Check, in sequence, the 12 address lines AB0 - AB11. These lines are most easily accessed at the expansion connectors J7 and J8 as shown below:

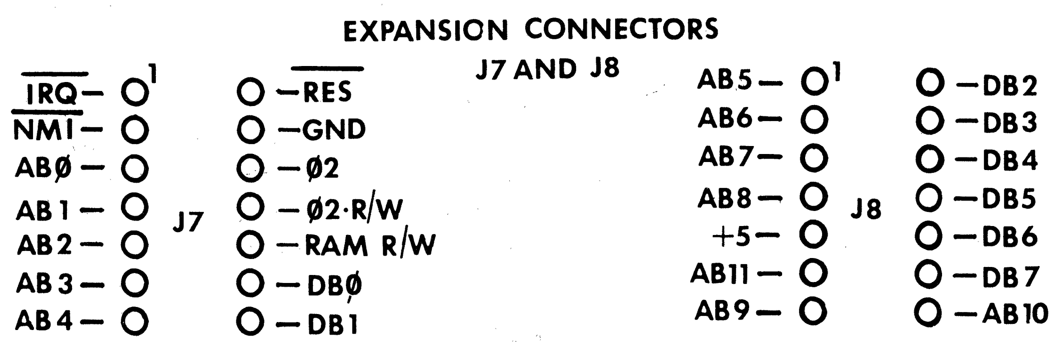


Figure 1

NOTE that AB9, AB10 and AB 11 are out of sequence at these connectors. When checking these waveforms you should observe that:

a) they are perfectly square (50% duty factor)
b) they will be slightly rounded on the rising as in figure 2:

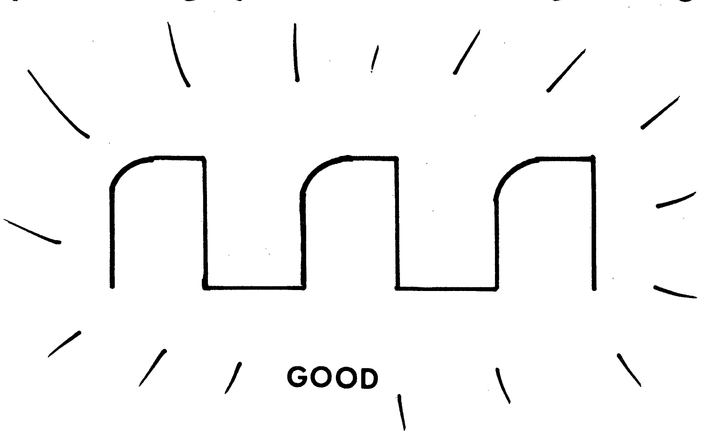
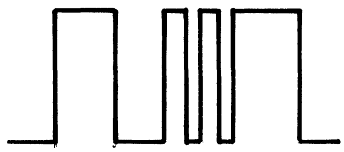


Figure 2

c) each of the lines in ascending sequence, starting with AB0, is exactly half the frequency of the preceding line in the sequence.
d) each line should swing from essentially supply to ground.
e) both the "1" state and the "0" state of the lines should be relatively free of glitches. (not more than 200 - 300 millivolts)

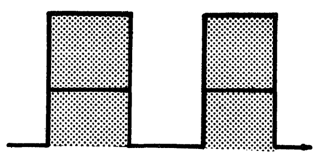
There are two problems that you will most likely spot with this test. First, that the lines do not toggle symmetrically but switch in bursts, which at low oscilloscope sweep rates will look like this:



BAD

Figure 3

which could be an indication either of a malfunctioning component or a defective conductive trace or solder joint on the circuit board.
Second, one or more of the address lines may not swing fully between supply and ground, which, again at low sweep rates, will look like this:



BAD

Figure 4

and could also be caused by a defective component but is more probably the result of a short between adjacent conductors on the circuit board. Even more specifically, this condition can most often be traced to a short between the malfunctioning address line and either a data line or one of the chip select lines. More information on these conditions can be gained with the rest of the tests.

8) Check the chip select lines individually. There are seventeen of them;

4 lines going to the RAM chips (check at pin 13 of each of the RAM locations IC22 - 25)

4 lines going to the ROM chips (pin 14 of each of the ROM locations IC16 - 19

1 line ($\overline{CS9}$) present at pin 11 of IC12

4 lines ($\overline{CS0}$, $\overline{CS1}$, $\overline{CS2}$, and $\overline{CS3}$) present at pins 11, 3, 4 and 10 respectively, of IC13

4 lines ($\overline{CS4}$, $\overline{CS5}$, $\overline{CS6}$ and $\overline{CS7}$) present at pins 10, 11, 3 and 4 respectively, of IC9

At each of these points you should see the same thing; a narrow negative pulse on the order 1 to 2 milli-seconds in duration. Like this:

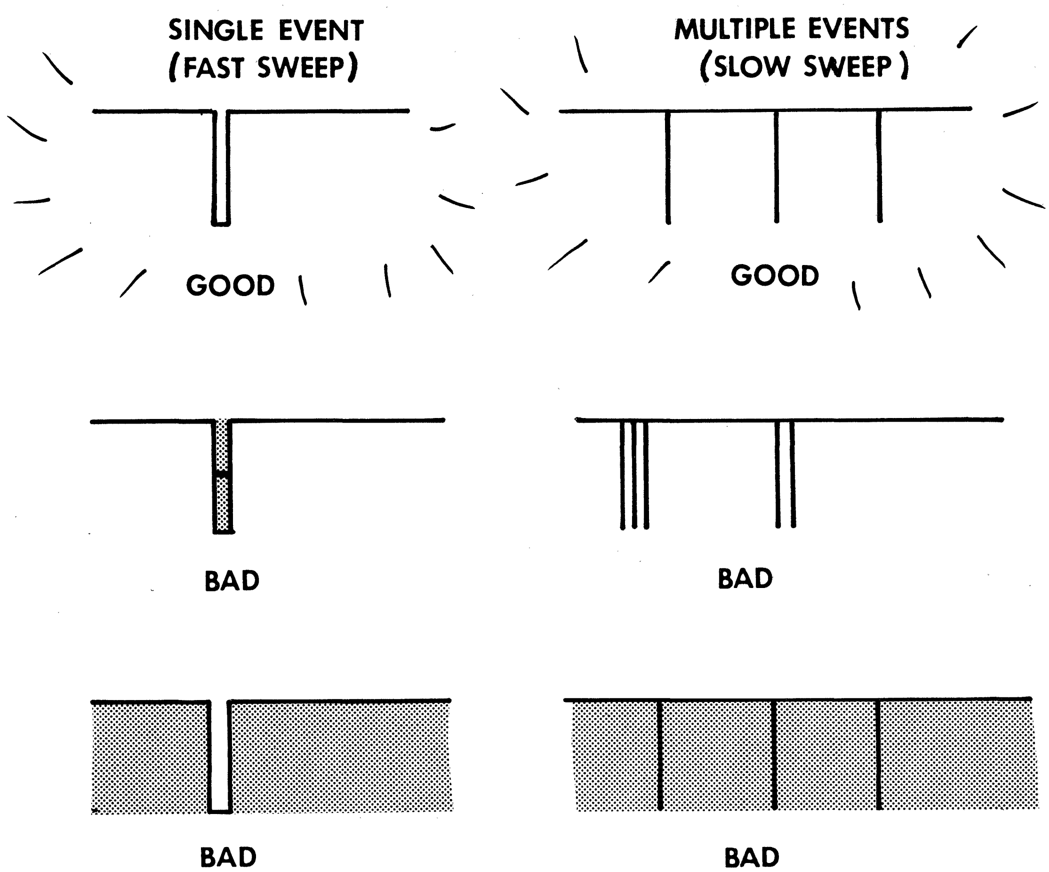


Figure 5

You should be sure that these pulses:

a) occur at a constant repetition rate
b) occur at constant time intervals
c) swing essentially from supply to ground
d) do not have faster switching events happening inside the pulse (with the exception of $\overline{CS\emptyset}$ - $\overline{CS7}$, which will have switching inside the negative-going pulse)

9) If you have checked and successfully verified that all of the points above are as they should be, this last becomes academic. Check the data buss lines which are accessible at the expansion connectors J7 and J8. These lines should be totally static (not switching at all) and should be at the logical levels which follow:

DB0 - 0
DB1 - 1
DB2 - 0
DB3 - 1
DB4 - 0
DB5 - 1
DB6 - 1
DB7 - 1

Even if the lines are static, check to make sure that they are all within 500 millivolts of either supply or ground.

Successful completion of all of the foregoing procedures is a very strong indication that the 8700 Computer is functioning properly and will continue to do so when mated with its companion keyboard.

BEFORE LEAVING THIS SECTION BE SURE TO RESTORE THE MACHINE TO ITS ORIGINAL CONFIGURATION. Put RAM and ROM back in place, solder the ends of the jumper S1 back together and cut jumper S2 being sure to fold the ends back so they do not touch each other or surrounding circuitry.

If any of these tests failed, you must begin trouble shooting. It would be nice if we could cover all of the things which can potentially die or short to one another. We can't. Trouble shooting a system of this level of complexity is most readily accomplished in much the same manner as methods employed by medical diagnosticians:

a) consider all the symptoms (complete all tests)
b) postulate a defect that would produce some or all of the observed symptoms
c) check your hypothesis
d) probably go back and try again.

WE CAN HELP and are happy to do it. If you have any difficulties with the tests in this section, write or call:

PAIA Electronics, Inc.
1020 Wilshire Blvd.
Okla. City, OK, 73116
(405) 843-9626 9:00 am - 5:00 pm CST

Please supply information relative to the results of your tests: which lines looked OK, which didn't, etc.

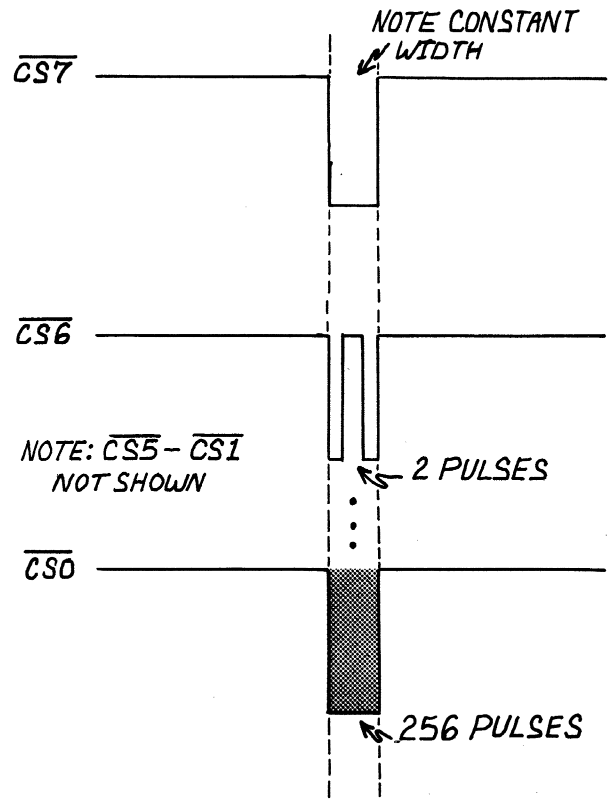
NOTES

8700 ERRATA

8700 manual erratta sheet
(we'll get it right yet)

On page 60 of the manual, in the section dealing with using the Micro-Diagnostic to check the chip select lines CSO-CS7;

Each of these chip select lines in a properly operating 8700 will be different. CS7 will be identical to the GOOD Single Event (fast sweep) waveform shown on page 60. CS6, at fast sweeps, will appear to be two very closely spaced negative-going pulses. CS5 will be 4 closely spaced pulses, and so on "up" to CS0 which will be a quick burst or 256 pulses (which will appear simply as a haze on most 'scopes). An important point is that the width in time of the group pulses will be the same for all 8 lines, whether 1 pulse or 256.



8700 CPU

PARTS LIST

Before beginning assembly of your 8700 computer controller, carefully check the parts enclosed in each parts bag against this parts check list. Every effort has been made to assure that the proper parts are included.

One each of the following parts:

8700 IC PACK	24 pin socket	28 pin socket
3 CKT power connector (see drawing page 27)	33 pf. ceramic disc. capacitor	

one strip of 20 molex pins

Three each:

1N914 diodes

Four each:

33K resistors (orange-orange-red)

Eight each:

2N5129 transistor

Nine each:

16 pin socket

Ten each:

.05 mfd. ceramic disc capacitor

Seventeen each:

14 pin socket

Twenty each:

10K 1/4 watt resistor (brown-black-orange)

27K 1/4 watt resistor (red-violet-orange)

8700A Keyboard

Parts List

Before beginning assembly of your 8700A Keyboard, carefully check the parts enclosed in the parts bag against this parts check list. Every effort has been made to assure that the parts enclosed in the parts bag correspond with this check list. Report any shortages at once to PAIA.

One each of the following Parts:

12" length of bare wire

12" length of insulated wire

Miniature SPDT pushbutton switch

IC header (see fig 6, page 12; fig 7, page 13)

Two each:

4-40 X 1/4 inch machine screws

Four each:

150K resistor (brown-green-yellow)

100K resistor (brown-black-yellow)

4-40 X 1 inch screws

Six each:

Rubber feet

4-40 nuts

CD-4001 Integrated Circuit (in foam)

Eight each:

5/16 inch spacers

Twelve each:

#4 flat washers

Sixteen each:

82K resistors (grey-red-orange)

Twenty-four each: 27K resistors (red-violet-orange)