COMPUTER!

Build this minicomputer yourself. Add it to the TV Typewriter for a complete computer system of your own

by JONATHAN A. TITUS

The Radio-Electronics Mark-8 Minicomputer is a complete minicomputer which may be used for a number of purposes, including data acquisition, data manipulation and control of experiments. It may also be used to send data to a larger computer or to a terminal such as the Radio-Electronics TV Typewriter, (September 1973) and it is easily interfaced with a keyboard. The keyboards do not have to be ASCII encoded since the minicomputer itself can convert the input code to an equivalent ASCII code for output. This Minicomputer is not a glorified calculator and it is not intended just for educational use. It can be interfaced to a calculator (a possible future project if readers are interested) to perform complex mathematical routines, and it may also be used as a teaching tool.

The heart of the Mark-8 Minicomputer is an Intel 8008 microprocessor IC that contains all of the arithmetic registers, subroutine registers and most of the control logic necessary to interface the microprocessor with semiconductor memories as well as input and output registers. Standard TTL type IC's are used throughout and commonly available 1101, 1101A and 1101A1 type memories are used for the central storage. The microprocessor with its associated logic will be refered to as the central processor unit, or CPU.

The central processor unit is an 8-bit parallel processor. A string of eight binary bits, D_7 through D_0, is used to indicate the instruction data or memory locations. Rather than repeat, "eight bits of binary data", we refer to the eight bits as a byte. As you will note, some of the instructions take up to three bytes of data and they are, therefore, called three-byte instructions. The computer takes 20 μs to execute each byte of these instructions, so the time to execute any of the basic instructions may vary from 20 to 60 μs. The time that the computer takes to execute one byte of the instruction is called the computer's cycle time. Most minicomputers have a cycle time that is about ten times faster than the Mark-8, but this will not restrict the use of this Minicomputer in most situations.

The Intel 8008 microprocessor provides us with some sophisticated features, only found on larger, more costly computers. These include a pointer register, interrupt pointers and a stack register for multiple subroutines.

The Mark-8 is programmed in assembly or machine language, the basic language of all computers which consists of 1's and 0's grouped into bytes. While it may seem cumbersome at first, this is one of the most flexible ways to program while keeping down the cost of added storage or memory. The use of just the 1's and 0's to represent the binary numbers can become tedious after a short while. It becomes much easier to convert the binary numbers to their octal equivalent and use these direct equivalents instead.

There are 48 program instructions to use in programs on the Mark-8. Each program must consist of an orderly, logical chain of steps in successive memory locations. If data or program steps are not loaded in the correct order, the program won't work correctly and is said to have a bug in it. Those not familiar with the basic operations of a computer and the various number systems used will find *Computer Architecture*, by Caxton Foster, Van Nostrand-Reinhold, New York, New, York 1970, $12.50 an easy to read and understand introduction that should be read before attempting to build or use the Mark-8.

The basic Minicomputer consists of six modules:

1. Main CPU module.
2. Memory Address/Manual Control module.
3. Input Multiplexer module.
4. Memory module.
5. Output module.
6. Readout module.

These modules provide the experimenter with the basic minicomputer configuration. Two 8-bit input ports are provided for getting data into the computer and four 8-bit output ports are provided to output data to external devices. The memory module can accommodate up to 1024 bytes or words of storage, although only 256 words are required to start. Manual controls are provided for the user and a readout of some of the important registers is provided on the Readout module.

Six different modules

The Central Processor Unit (CPU) module contains the microprocessor IC and the extra circuitry used to interface with the rest of the computer. It is important to note that the 8008 microprocessor has been fabricated as an MOS circuit and the outputs will only drive one low-power circuit of the 74L series. Each output is buffered with a 74L04 inverter before it is used. The main, 8-line input/output bus, or I/O bus is also buffered by two 7404 circuits to give the TTL signals a high fan-out.

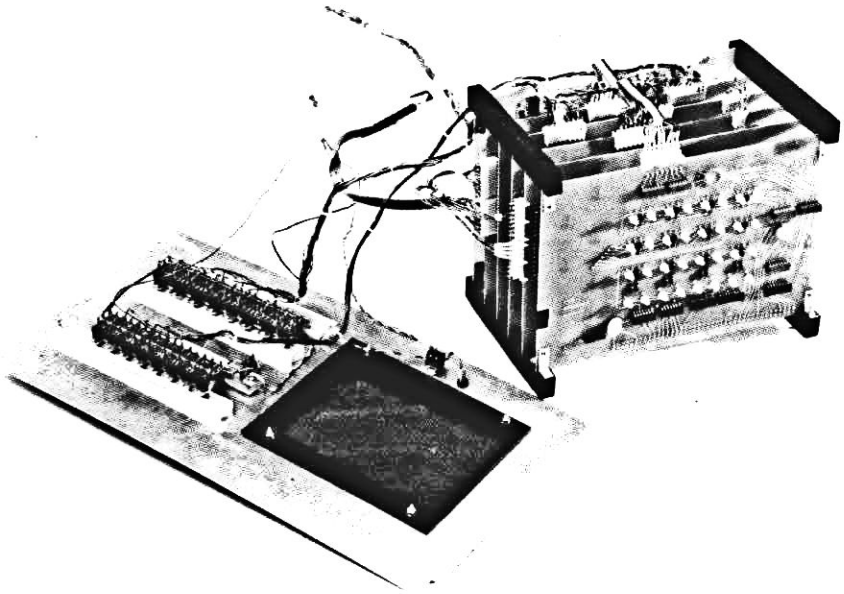
The computer is controlled by a 2-phase clock supplied by a crystal oscillator which controls the pulse widths and frequency. The clock and the synchronization signal supplied by the microprocessor are used to control some of the logical operations of the computer interface circuits. The synchronization signal synchronizes the operation of the very fast TTL circuits and the slower, clocked, MOS circuits in the microprocessor. The microprocessor also has three, state-output signals, S_0, S_1, and S_2 which are used to drive a decoder. The eight possible states are then used to control other functions in the interface logic. A complete description of the generation and use of these state outputs is included in the Intel User's Manual

Since the CPU uses a parallel 8-bit I/O bus for input and output of data there must be some control of when the bus is sending data from the CPU to an external device or when it is taking data in. Two lines are present on the CPU module, $\overline{IN}$ and $\overline{OUT}$. These lines are used by the other modules to regulate the flow of data in the correct direction at the correct time. The control of the $\overline{IN}$ and $\overline{OUT}$ lines is governed by the additional logic on the CPU module.

The Memory Address/Manual Control module is used to hold data which is to be used as the memory address. Two 8-bit latches are provided since the computer will use one set of eight bits for a memory address and the other set of eight bits for control functions. Since the microprocessor can directly address up to 16,424 words of memory, commonly noted as 16K, we will need 14 binary bits for the complete address. The complete memory address of any location is given by a 16-bit binary number: X X B_A B_A B_A B_A B_A B_A / B_B B_B B_B B_B B_B B_B B_B B_B, where the X's represent bits that are not used. The computer specifies any address by first sending out the B_B bits to one of the eight-bit latches, followed by the six B_A bits and two X bits. Control of the correct latch is supplied from the CPU module.

The B_A bits have the most significance or value in the complete digit, while the B_B bits have the least significance. This is like comparing $1000 and $1. The further to the left the digit, in any numbering system, the more value it has. For this reason the B_A bits are called the most significant or the HI part of the address, while the B_B bits are called the least significant or LO part of the address. Both the HI and LO address latches are made up of SN74193 programmable coun-

COMPUTER WITH ASCII KEYBOARD makes a complete working computer system. You can use the computer without the keyboard, but it is more difficult.



THE WORKING HEART of the computer is relatively simple. The six primary circuit boards and the front-panel controls are shown here. If additional memory is needed, more circuits boards are required.

ters, since the address held in them may be incremented, by counting up by one. The usefulness of this will be seen later. The HI and LO latches are also used for temporary data storage when they are not being used to store a memory address.

The manual control portion of this module allows us to program the computer and to control its operation from an operator's console. We are able to externally address any memory location and deposit data or instructions in it. We may also return to any location and check the data stored there. Controls are also provided to allow us to single-step the computer through a program, one instruction at a time and to interrupt the computer while it is executing a program. These controls will be described in detail later.

The Data Input Multiplexer module controls the flow of all data into the computer. All data going into the computer is placed on the I/O bus during the IN cycle signalled by the $\overline{\text{IN}}$ signal. Since data may be coming in from a number of different experiments or sources, we must have some means of selecting which data is fed into the CPU. Two basic multiplexers are used for this precise gating of data. The two 8263 quad, three-line to one-line multiplexers control which of three sets of input lines are selected. Note that two sets of these input lines are input ports 0 and 1. These are the two external data input ports. The third set of data input lines comes from the memory. Data or instructions in the memory, all go through the multiplexer and into the CPU.

This multiplexer is followed by a second set of multiplexers, 8267's. These are quad, two-line to one-line multiplexers with open-collector outputs which are compatible with the computer bus structure. This multiplexer switches between the data selected at the previous multiplexer and data from the Interrupt Instruction Port. The use of the Interrupt Instruction Port will be covered in the Interrupt section. This second multiplexer may also be in an off or unselected state which is used when data is not to be sent to the CPU module. Control lines SL_0 and SL_1 are sent directly from the CPU interface logic.

Remember that when the HI address is not being used to store a memory address, it is used for control signals. During an IN or OUT cycle these control signals are decoded and used to select the proper input or output lines for the I/O bus. The Multiplexer module decodes the control bits B, C, D, and D_{Enable} and OR's them with $\overline{\text{IN}}$ to select the proper external data input port. When the computer is instructed to get some data from memory it automatically selects the memory input section of the multiplexer. The INPUT instruction is only used when you wish to input data from some external source such as a digital voltmeter or keyboard, through one of the two input ports.

The Memory module uses the widely available 1101 type of semiconductor, integrated circuit memory. The 1101 random access memory or RAM is organized as a 256 x 1-bit memory, so eight of the 1101 type memories are used to give us 256, eight-bit words. This is the minimum configuration necessary for the operation of the Mark-8. Each memory module can hold 32 of the 1101 memories for a total of 1024 or 1K words of storage. Up to four Memory modules may be used with the Mark-8, giving us a maximum 4K of storage space. More than enough for most applications.

Each of the 256 words are addressed by the eight bits from the LO address latch. Since $2^8 = 256$ we can only address 256 words using the LO address alone. Each memory also has an enable line so we may select blocks of 256 words, using this line. The HI address is, therefore, used and decoded with a standard decoder and the decoded outputs are used to enable or select the blocks. You do not have to be concerned about the particular block where data has been stored, just use the complete 14-bit address, since the memory does the complete decoding.

Each of the addressed memory locations may store one 8-bit word or byte of information. For 2 or 3-byte program steps, two or three successive memory locations are used for storage.

The 1101 type memories are volatile semiconductor memories and information stored in them will be altered or lost if the power is shut off. If you want to save a program, leave the power on.

A chart in the construction section shows how the memory jumpers are wired for each of the four possible boards. Boards must be added in numerical sequence; 1, 2, 3, and 4. Blocks of memory must be added in units of 256 words in the A, B, C, and D sequence, to prevent gaps in the memory.

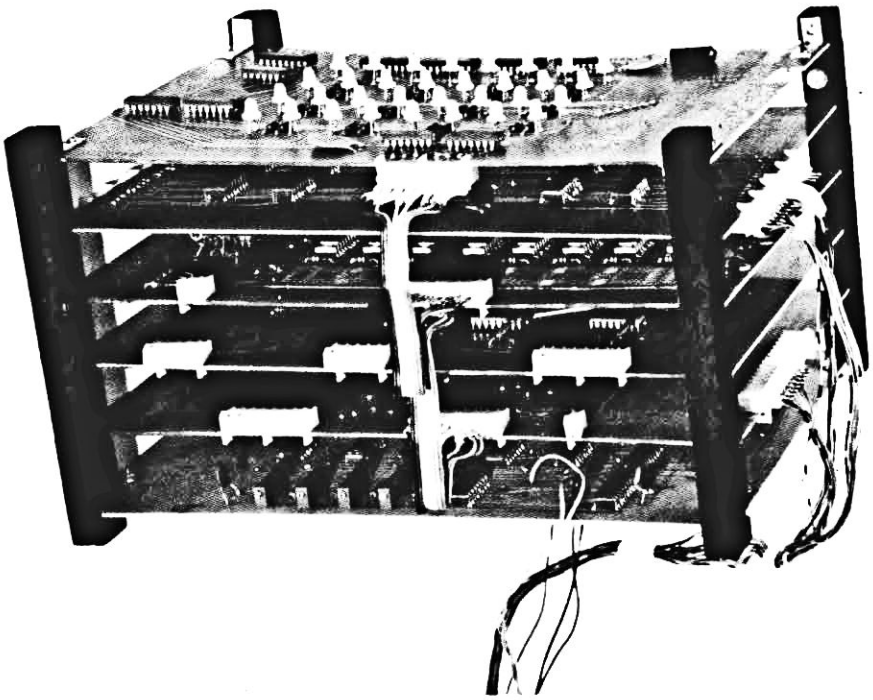
A read/write or R/W line is provided on the module so that data may either be read from, or written into a selected memory location. The CPU and the Manual Control module both control this line so that data may be entered under computer control or

so that we may insert our program data into the memory prior to use by the computer.

The eight data-output lines from the memory are sent to the CPU I/O bus through the Input Multiplexer module. When we ask for data from the memory with an LrM type of instruction (see Intel User's Manual), the CPU senses that the memory data is needed and it sets the input multiplexer so that the data is placed on the I/O bus at the proper time.

The Output Latch module is used to send data from the computer to some external device or instrument, such as a teletype or perhaps the **Radio-Electronics** TV Typewriter (**Radio-Electronics,** September 1973). Four output latches are provided on the Output Latch module and two of these modules may be used with the Mark-8. The second module may, however, only use three of the output latches.

Note that data is sent from the LO address latch to each output port and that these connections are in parallel. The computer decides which latch is activated according to the OUTPUT instruction that we have in our program. Here, again, the HI address latch holds the control bits B, C, and D which are decoded and NORed with $\overline{OUT}$ to activate the selected eight bit output port or latch. NOTE: The OUTPUT instruction in the Intel User's Manual has two RR bits shown in it. These bits must be set to RR =



PRINTED-CIRCUIT BOARD ASSEMBLY is a stack of six 2-sided boards. Molex connectors and cables are used to interconnect the boards and to connect the boards to the front-panel controls.

PARTS LIST

All resistors are ¼ Watt, 10%

CPU BOARD

C1—33-pF disc
C2 thru C6—0.1-μF disc
IC1, IC4, IC6, IC7, IC9, IC13, IC17, IC19— 7400
IC2, IC3, IC14—7476 Dual JK flip-flop
IC5, IC11, IC16, IC20, IC21—7404
IC8, IC12—7474 dual D flip-flop
IC22, IC23, IC25—74L04 hex inverter, low power
IC10, IC18—7410
IC15—7420
IC24—8008 Intel microprocessor
IC26—7442 decoder
R1, R2—220 ohms
R3—560 ohms
R4—1800 ohms
R5, R6, R7, R8, R17—1000 ohms
R9 thru R16—22,000 ohms
XTAL 1—4000.000-KHz crystal type EX ($3.95 from International Crystal, 10 N. Lee Street, Oklahoma City, OK)
Misc—PC Board, No. 24 wire, solder

INPUT MULTIPLEXER BOARD

C1, C2, C4—0.1-μF disc
C2—1.0-μF 10 V electrolytic
IC1, IC2—8263 multiplexer (Signetics)
IC3—7400
IC4, IC5—8267 multiplexer (Signetics)
IC6—7402
IC7—7442 decoder
P1, P2, P3, P4—Molex type 09-52-3081 connectors
R1—1000 ohms
Misc—PC board, No. 24 wire, solder

ADDRESS LATCH BOARD

C1 thru C6—0.01-μF disc ceramic
C7—680-pF disc
IC1, IC2—74123 dual monostable
IC3, IC4, IC5, IC6, IC7—7400
IC8, IC9, IC10, IC11—74193 programmable counter
P1, P2, P3—Molex Type 09-52-3081 connectors
R1 thru R3—10,000 ohms
R4-22,000 ohms
R5 thru R16—1000 ohms
Misc—PC board, 324 wire, solder

MEMORY BOARD

C1, C2, C3—0.1-μF disc ceramic
IC1 thru IC8—1101, 1101A or 1101A1 memory circuits, 256 x 1
IC9 thru IC32—Same as above, but optional with builder
IC33—7442 decoder
IC34—7400
P1, P2—Molex type 09-52-3081 connector
R1 thru R11, R20, R21—1000 ohms
R12 thru R19—10,000 ohms
Misc—RC board, No. 24 wire, solder

OUTPUT LATCH BOARD

C1, C2, C3,—0.1-μF disc
IC1 thru IC8—7475 quad latch
IC9, IC10—7404
IC11—7402
IC12—7442
P1, P2, P3, P4—Molex type 09-52-3081 connector
Misc—PC board, No, 24 wire, solder

LED REGISTER DISPLAY BOARD

C1—100-μF electrolytic
C2, C3, C4—0.1-μF disc
D1 thru D32—MV-50, MV-5020 or equivalent Red, visible LED's
IC1 thru IC6—7404
IC7, IC8—7475 quad latch
IC9—7442 decoder
IC10—7402
P1—Molex type 09-52-3081 connector
R1 thru R32—220 ohms
Misc—PC board, No. 24 wire, solder

CONTROL PANEL

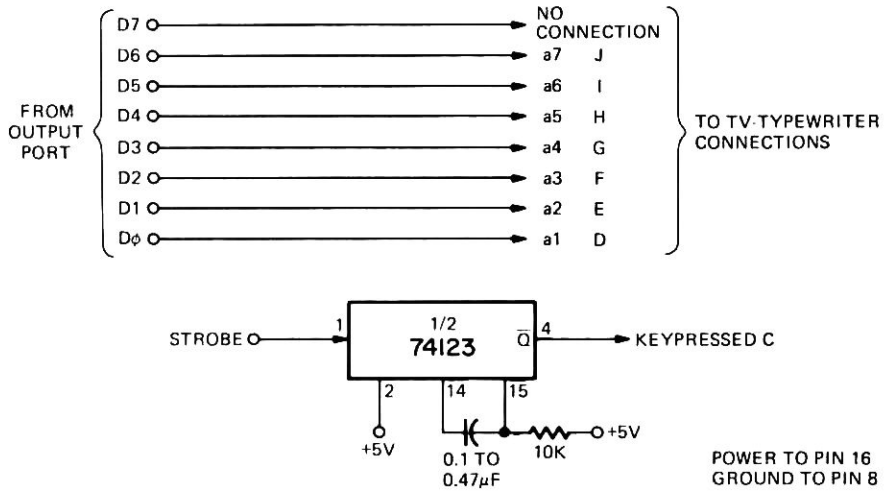
D1—MV-5020 or equivalent red, visible LED
R1—220 ohms
S1 thru S11—spdt switches, rocker or toggle
S13 thru S17—spdt momentary, spring return, rocker or toggle
PS—Power supply, logic power supply available from Precision Systems, P.O. Box 6, Murray Hill, NJ 07974. +5 volts/8.5A and −12 volts/2.0A, adjustable to −9 volts. Also other voltages available. See text.
Misc—Metal case, red plastic filter, line cord, hardware, hook-up wire, solder.

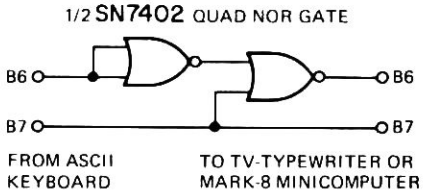
The microprocessor integrated circuit is available from Intel Corporation, 3065 Bowers Avenue, Santa Clara, CA 95051 at a cost of $120.00.

A complete set of circuit boards is available for the Mark-8 Minicomputer from Techniques Inc., 235 Jackson Stree, Englewood, N.J. 07631. Prices include shipping charges inside the United States.

Complete set of six boards (1 of each)	**$47.50**
CPU Board	**7.50**
Address Latch Board	**10.50**
Input Multiplexer Board	**9.50**
1K Memory Board	**8.45**
LED Register Display Board	**8.45**
Output Ports Board	**8.50**

Techniques had 100 sets of boards in stock when this issue went on sale. When these boards are sold, there will be a 6 to 8-week delay before additional boards become available.





HOOKUP THE MARK-8 COMPUTER TO YOUR TV TYPEWRITER using the circuits shown above and to the left. Wiring to the TV typewriter is just direct connections (above). The IC monostable (left) stretches the pulse width. Together, the TV Typewriter (Radio-Electronics, September 1973) and the Mark-8 make a powerful computer package.

Mark-8 Minicomputer to TV-Typewriter interface

The Mark-8 Minicomputer may be used with the TV-Typewriter to display computer generated information. The interface uses either the A or B Output Port strapped to the specific output code, 1-7, that you select. The A and B output ports have strobe lines which are pulsed during the output cycle. These two lines are found above the B output lines and below the A output lines on the printed circuit board. These strobe lines provide us with the Keypressed signal required to enter data into the TV Typewriter. A monostable is attached to this strobe line to stretch the pulse width and the 10 μF capacitor used for debouncing is removed from the TV-Typewriter. This is C17 shown in Fig. 8 of the TV-Typewriter booklet.

01 for proper data output. OUT = 01 01M MM1. The MMM bits are set to the binary equivalent of the decoder state selected for that particular output port. For example 01 010 111 would output data at output port 3, since 011 = MMM = 3.

The LED Register Display module provides you with a visual indication of the contents of the HI and LO address latches and the memory data in the selected memory location indicated by that address. Output port 0 is also located on the Readout module and it may be used in programming to give a visual output of a byte of data. Each of the output registers is represented by eight LED indicators, 1 = ON, 0 = OFF. As the data held in each register changes, so do the indicators. Data to be displayed at output port 0 must be sent with an OUT instruction 01 010 001 or 121₈.

Since the HI address latch is used for some control functions and the LO address latch may also be used for temporary storage of data going to the output ports, at various times in programs the data in these registers will change from a memory address to these control and output data and then back to an address. Checking this data visually in these registers during the debugging of a program is very helpful.

The power supply requirements of the Mark-8 are +5 Vdc at 3 amps and −9 Vdc at 1.5 amps. Since regulation at these high current levels is critical we suggest that the power supply or supplies are purchased. There are many good power supplies on the surplus market that may be used with the Mark-8. The type used with the prototype is listed in the complete parts List. A substitute, available from Wortek, 5971 Reseda Blvd., Tarzana, Calif. 91356 will work as well. Order part numbers PRS-1 and PRS-3, each $25.00 **R-E**

* For more detailed data on the Microprocessor IC write to Intel Corp., 3065 Bowers Ave., Santa Clara, Calif. 95051 - ask for a copy of "*8008, 8-Bit Parallel Central Processor Unit-Users Manual*. This manual was offered free at the time this article went to press.

SOFTWARE EXAMPLE

Data in the A register is output to the TV-Typewriter as a complete ASCII character. The computer then enters a short timing loop so that it can not go faster than data may be entered to the TV-Typewriter memory.

000	006		LDAI	/Load A with data
001	177		177	/Data = 177 = ASCII "?"
002	106		JSUN OUTPUT	/Jump to OUTPUT subroutine
003	040			
004	000			
005	000		HALT	/Stop, end of program
040	123	OUTPUT,	OUT1	/Data from A to output port 1
041	026		LDCI	/Load C Immediate
042	004		004	/Data
043	031	LOOP,	DECD	/Decrement D
044	110		JPFZ, LOOP	/Jump on a false zero flag to LOOP
045	043			
046	000			
047	021		DECC	/Decrement C
050	110		JPFZ, LOOP	/Jump on a false zero flag to LOOP
051	043			
052	000			
053	007		RTUN	/Unconditional return to main program

Construction

The Mark-8 Minicomputer consists of printed circuit boards for each of the six modules, a main chassis for mounting the PC boards and the controls and a power supply. The addition of a keyboard and and an alphanumeric readout is optional. The printed circuits are double sided, but the holes have not been plated through to keep the cost down. It will be necessary to be sure that both sides of the components are soldered to the board. There are also a number of holes that do not have any pins or components going through them. Short pieces of wire are inserted into these holes and soldered on both sides of the board to give a continuous current path. Use a low-wattage soldering iron and be sure that pins and components are soldered to both sides of the board. Check the parts orientation for each module in the following diagrams.

It is best to use Molex-type IC connectors for the 8008 microprocessor integrated circuit since some tests will be performed before it is placed in the circuit. Do not put the circuit in its socket before the instructions tell you to do so. Be sure to solder the socket pins on both sides of the PC board.

The Output Latches have a set of jumpers which allow the user to choose which set of lettered (A,B,C,D) outputs are assigned what code. If no preference is shown, jumper A = 1, B = 2, C = 3, D = 4 and so on for the three latches on the additional Output module, if it is used.

The Memory module also has a number of jumpers that are used to select various options. On the first or only board install the A jumpers below the SN7442 decoder and install all of the resistors. On following boards install the B jumpers and only resistors R17 through R21.

There is a 0 and a 1 jumper between the SN7442 decoder and the SN7400 NAND gate IC. Install the 0 jumper on the first two boards and the 1 jumper on the last two boards. **NEVER** install both jumpers.

Install the 1101 type memories in rows of eight IC's starting from the bottom row of eight and going to the top. Additional memories are added in blocks of eight IC's. The bottom row on the first board is the minimum configuration for a working computer. The blocks of memory are selected by the jumpers above the SN7442 decoder. The four pads above the eight numbered ones are connected to each block of eight IC's. Select, from left to right, on boards 1 and 3 codes 0, 1, 2, and 3. On boards 2 and 4 select codes 4, 5, 6 and 7. The jumper combinations for the four possible boards are summarized in the following chart.

MEMORY JUMPER CHART

Board	Resistors	A or B Jumper	0 or 1 Jumper	Block Code Jumpers
1	ALL	A	0	Figure 1
2	R1–R4 & R21	B	0	Figure 2
3	R1–R4 & R21	B	1	Figure 1
4	R1–R4 & R21	B	1	Figure 2

MEMORY BLOCK CODE JUMPERS

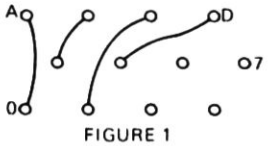


FIGURE 1

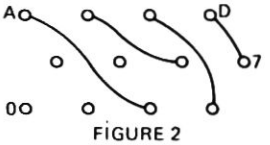
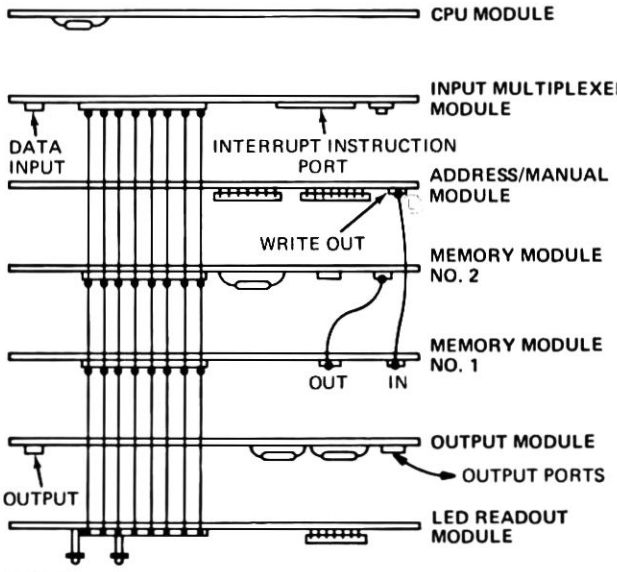


FIGURE 2

The second, third and fourth memory boards derive their Read/Write signal from the first board. When adding extra boards, after the jumpers have been set and the boards soldered into the proper sequence, connect the OUT on the top of the first board to the IN on the second, then the OUT from the second to the IN of the third, and so on. The WRITE OUT on the top of the Address/Manual module goes to the IN on the first board of the four boards. If only one board is used, wire it as if it were board number one.

All six circuit boards are connected by a series of up to 41 wires which are laced through the bottoms of the boards. All but a few are parallel connections. This allows us to "fan" the boards open for testing after they have been connected. We also avoid the use of expensive and special connectors. A few connections are made on the board tops, but these are easily made with the Molex connectors in the parts list. These connectors may also be used on the sides of the Input and Output modules where the input and output lines are connected. The following diagram, Fig. 3, shows the top-of-board connections. Be sure to observe the correct board orientation and sequence when wiring the boards together. The LED's on the readout module must face outwards and the components on the following boards face in the same direction. When mounting the Mark-8 on or in a chassis, remember to allow enough room for possible future module expansion as well as for additional modules which may be published in the future. Also allow about 3/4 to 1 inch between boards for convection cooling. The boards have been purposely made a large size to ease the job of trouble shooting and following PC conductor paths.



NOTE:
THIS DIAGRAM SHOWS THE PARTS ORIENTATION ON THE MODULES AND SOME OF THE BOARD-TO-BOARD CONNECTIONS. SEE THE INDIVIDUAL BOARD DRAWINGS FOR A COMPLETE DRAWING OF THE PART LOCATIONS AND CONNECTIONS.

FIGURE 3

The power supply used with the prototype was a surplus computer power supply available from Precision Systems, P.O. Box 6, Murray Hill, N.J. 07974, for $35.00 postpaid. This power supply provides +5 Vdc at 8.5 amps, +12 Vdc at 2.5 amps, -12 Vdc at 2 amps. The -12-volt supply is easily changed to -9 volts, using the variable resistor trimmer on the side of the power supply. Unregulated voltages of 6 Vdc, 30 Vdc and 180 Vdc are also provided.

Controls

Since we are working with an 8-bit computer, eight switches are provided on the Interrupt Instruction Port. This is called the Switch Register or SR and it is one way to get data into the computer under manual control. You will notice on the Input Multiplexer module schematic diagram that the SL_0 signal is gated with a Jam signal. When the Jam is at ground, this forces the SL_0 signal to also go to ground. When this happens, the 8267 multiplexers are held in the state which allows the data present at the Interrupt Instruction Port to be placed on the I/O bus, going to the memory and to the HI and LO address latches. When the INTERRUPT/JAM switch is returned to the normal INTERRUPT position, control of the SL_0 line is taken over by the CPU control logic. The JAM control allows us to jam data onto the I/O bus. This Jam mode is useful only when the computer is not operating, but before we start the computer the INTERRUPT/JAM control switch must be in the normal INTERRUPT position.

The computer is normally in the RUN state and it only halts when it reaches a halt or HLT type instruction in our program. If we wished to see how a program worked at slow speed so that we could follow it, it would be necessary to slow down the computer. The Mark-8 has a RUN/SINGLE STEP switch which allows us to either run the program at the normal computer speed of 50,000 steps per second, or a step at a time. In the RUN mode the computer operates at its own speed, determined by the clock. In the SINGLE STEP mode the computer is pulsed each time we press the SINGLE STEP switch, causing a complete computer cycle to take place. In the SINGLE STEP mode it is much easier to debug a program and find our programming errors. We can switch between the RUN or SINGLE STEP modes without affecting the normal operation. We only slow down the execution of the program.

The use of the next four controls lets us enter data into the com-

puter and check data already stored in memory, before starting the computer program. In this way computer programs are stored and then used. When using any one of these next four controls, the INTERRUPT/JAM switch must be in the JAM position and the RUN/SINGLE STEP switch must be in the SINGLE STEP position.

The LOAD ADDRESS-HI or LAH switch allows us to load the HI address latch with the number currently set on the switches in the switch register. We may change the number as often as we like, just by changing the number set on the switch register and actuating the LAH switch. The LOAD ADDRESS-LO or LAL switch is operated in exactly the same way, entering the number set on the switch register to the LO address latch. Using the eight switch register switches and these two controls we can manually load any address in the address latches. You will notice the new address appear on the LED indicators in the HI and LO address readouts when we do the checkout. As soon as an address has been loaded the memory data LED's will indicate the current contents of the location we have just addressed. We can address all possible 16,000 storage locations in the memory from the switch register, but it is important to note that it is only realistic to try and address memory locations that actually exist as implemented with the 1101 memories.

The DEPOSIT switch allows us to deposit data from the switch register to the memory location that we have just addressed. Once the HI and LO addresses have been loaded and checked we set the switch register to the value of the data to be entered into that location. Pressing DEPOSIT causes the computer to write the data word set in the switch register in the memory location we have selected. The memory address is automatically incremented when we actuate the DEPOSIT switch. This is done by using the SN74193 programmable counters as the address latches. By automatically incrementing the address we have gone to the next memory location without having to reload the next successive memory address. We may now deposit data in the next memory location and the next and so on just by setting the data in the switch register and actuating DEPOSIT. The address steps to the next location automatically. In this way blocks of data are easily stored in successive locations.

Another control, EXAMINE, is also provided. Once we have loaded an address in the HI and LO address latches using LAH and LAL we see the contents of that location in the memory data LED readouts. Actuating EXAMINE steps the memory address to the next location without altering the data stored there. We may examine consecutive locations just by depressing the EXAMINE switch. This allows us to check programs or data without altering the data present. Suppose that we depress EXAMINE and we find that the data is incorrect and we would like to change it. Set the new data in the switch register and actuate DEPOSIT. This changes the data in that particular location. We may now continue to examine data in the next consecutive locations by continuing to actuate the EXAMINE switch. Any of our memory locations may be examined and data may be stored in any of them. Since the new data writes over the old data it is not necessary to clear out a memory location before it is used.

Remember that these controls may only be used if the computer is in both the JAM and SINGLE STEP modes.

Interrupt

The INTERRUPT has been mentioned, but little has been said about it. The INTERRUPT is used to do just that to the computer. We can interrupt the present program and cause the computer to temporarily do some other task and then return to the program. It is also possible to make the computer leave a stopped state with the interrupt.

Suppose that we want to get a character from a parallel output ASCII keyboard and display it on the eight LED's of output port 0. The needed program could be written as:

Address		Instruction		Comments*
HI	LO	Binary	Octal	
00	000	01 001 001	111	—Input data from port 0 to CPU register A
00	001	01 010 001	121	—Output data from register A to output port 0
00	002	11 111 111	377	—Halt, wait to be interrupted
00	003	01 000 100	104	—Unconditional jump back to:
00	004	00 000 000	000	—LO Address = 000
00	005	00 000 000	000	—HI Address = 000

*For a complete description of instructions see Intel User's Manual

Assuming that the computer has been started the program gets an ASCII character at input port 0 and places it in the CPU register A. It then gets the character from register A and sends it to output port 0. The computer then halts. There is a signal line on the keyboard similar to 'key pressed' or 'data ready' which is used to signal the computer that there is new, valid data ready at input port 0. This pulse or signal is used to strobe the External Interrupt Input with a logic level 1, causing an interrupt of the processor to take place. How does the processor know what it is to do next? The Interrupt Instruction Port is used to send an instruction to the computer which indicates to the computer what its next action should be. In this case we have entered a continue or no-operation, 11 000 000 or 300 on the switch register. This tells the computer to start up again and to execute the next instruction. The next instruction is an unconditional jump, a 3-byte instruction, which indicates that the program should now go to location HI = 00 and LO = 000, or the original start of the program where it will get the next character. The 300 must have been set on the switch register prior to trying to input data.

Any instruction may be set on the switch register to be fed into the Interrupt Instruction Port when the computer is interrupted. Multiple part instructions such as the unconditional jump instruction could also be entered, but it would take some extra interface logic to do this. Multiple byte instructions are often entered when the computer is in the Single Step mode. Usually only single byte instructions are entered. The 300 instruction, continue or no-operation, a halt or HLT and the restart or RST instructions are usually the only ones entered while the computer is running at its normal speed.

We have assumed that the program just described had already been started. How are programs started? You may have wondered, since there is no START pushbutton on the operators console. With the computer program entered in memory and checked as previously described, using the JAM mode in the SINGLE STEP mode of operation, we return to the INTERRUPT mode and we enter 104 on the switches in the switch register. We then depress the INTERRUPT switch and then the SINGLE STEP switch. We then enter the low address of the start of the program on the switches and actuate SINGLE STEP, followed by the high address and the actuation of SINGLE STEP. We have now entered a multiple byte instruction. If we now switch to the RUN mode, the program will run. Notice that we have only actuated the INTERRUPT switch once, just after entering the 104 on the switch register. This is one way in which we start a program. We enter an unconditional jump instruction followed by the low and then the high address of the start of the program. We can easily stop a program by entering a 377 on the switch register and interrupting the computer with the Interrupt switch. We start up again by entering a 300 on the switch register and actuating Interrupt.

The Restart or RST Instruction is one of the most useful instructions that may be entered from the switch register when we interrupt the computer. It is used to start programs and it is extremely useful when using the computer with external devices. The Restart instruction is a pointer type of instruction that points the computer to a particular location without a multiple byte jump instruction. If we look at the RST instruction in the instruction set we see three bits labeled A: 00 AAA 101. The A's are set to the starting address of our program and we assume that all other bits in the address are zero; HI = 00 000 000, LO = 00 AAA 000. Note that we may only point to eight particular locations with the RST instruction as shown in the following table:

AAA	Restart Instruction	Pointed to	
		HI	LO
000	005	00	000
001	015	00	010
010	025	00	020
011	035	00	030
100	045	00	040
101	055	00	050
110	065	00	060
111	075	00	070

Assuming that the ASCII input/output program still starts at address 00 000 we could put the computer in the INTERRUPT and RUN modes, enter 005 on the switch register and actuate the INTERRUPT switch. We must then change the switch register setting to 300 for the program's use. We have used the Restart instruction to point the computer to one particular location. If our program started at 00 020, we could have used the 025 restart instruction to point the computer to 00 020.

Suppose, however, that our program does not start at any of these

locations, but at 00 150 instead. We would still like to be able to start the program without the multiple byte jump instruction having to be entered at the switch register at the start. In this case we go to one of the pointer addresses, say 00 030 and we place a jump instruction in that location, followed by the low and high addresses of our program's start. This will cause the computer to jump to our program whenever it reaches 00 030. The data stored now looks like:

00 030	104	Jump to:
00 031	150	LO Address
00 032	000	HI Address
"		
"		

We can now enter a 035 in the switch register and with the computer in the INTERRUPT and RUN modes we actuate INTERRUPT and cause the computer to go to location 00 030. Once it reaches 00 030 the instructions there cause it to jump to 00 150, the start of our program. We have now been able to point the computer to any location desired with a single restart instruction. This type of pointing after the computer is interrupted is usually found on large, expensive computers. It is very useful when taking in data from a number of sources, as we will see next.

Suppose that we have our ASCII keyboard connected to input port 0 and we have a digital counter connected to input port 1. Each must be able to input data to the computer. The counter will input data once a second, while the keyboard will enter data once in a while. The counter data is taken care of by a program at 00 040 and the ASCII keyboard's data is taken care of by a program at 00 020. Each input device uses the interrupt line to signal the computer that it has data ready to input, but the computer needs some way of distinguishing where the data is coming from. As each device causes an interrupt, it could also apply its coded pointer address to the Interrupt Instruction Port in the form of a coded Restart instruction. The computer would then be pointed to the correct program to take care of the different data present at the input port. The counter would apply 045 and the ASCII keyboard would apply 025 to the Interrupt Instruction Port. This could be done as shown in the following diagram, where the input lines have replaces the switches. Circuits to prevent the coincidence of the two signals would be needed, but the concept is still the same. This is a very easy method of allowing the computer to be pointed to the correct program to take care of input data from an external device.

Check-out and start-up

1. Recheck the wiring of the switches and the modules. Be sure that all components and IC's are soldered on both sides of the PC boards. Check to be sure that wires have been soldered through the holes from side to side where there are no components. At this point, the External Interrupt jumper on the Memory Address/Manual module should be in place, and no external device should be connected to the Mark-8.

2. Before connecting the power supply to the computer, turn it on and recheck the voltage levels. With the power OFF, connect the power supply to the computer. Place the control switches in the JAM and STEP modes. Set all switches in the switch register to 0. Be sure that the 8008 microprocessor IC is not in the circuit at this time.

3. Since the Mark-8 uses a crystal clock, no clock adjustments are necessary. With the 8008 IC still out of the circuit, turn the power on. Check the pins of the 8008 socket. Voltages should read between ground and +5 volts. Only one pin, pin 1, should be at -9 volts. If others are at -9 volts, check for solder bridges. Turn the power OFF and insert the 8008 IC.

4. Turn the power on. The 5-volt and 9-volt LED indicators should light. If they do not this could indicate either a poor connection or a short in the wiring. Some of the LED's on the Readout module may also be lit at this time. With the switch register set at all zeros, depress LAH and LAL. Now the LED's in both the HI and LO address registers should indicate all zeros. If no change occurs, check the connections of the control switches. If all the LED's now indicate ones, repeat the above procedure with the switches set to all ones instead. If the LED's now indicate all zeros, the power connections to the switch register are reversed. If the LED's stay on, check the connections to the INTERRUPT/JAM switch and the connections on the Input Multiplex module and the Memory Address/Manual module.

Place one switch at a time in the 1 state and depress LAH and LAL. The switch setting should be transferred to both the HI and LO LED readout registers. If it is transferred to one and not the other, check the connections to the SN74193 programmable counters on the Memory Address/Manual module. If it is transferred to neither, check the switch connections to the Interrupt Instruction Port. If it is transferred to the wrong bit position, recheck the connections to the Interrupt Instruction Port.

INTERRUPT ENCODING DIAGRAM

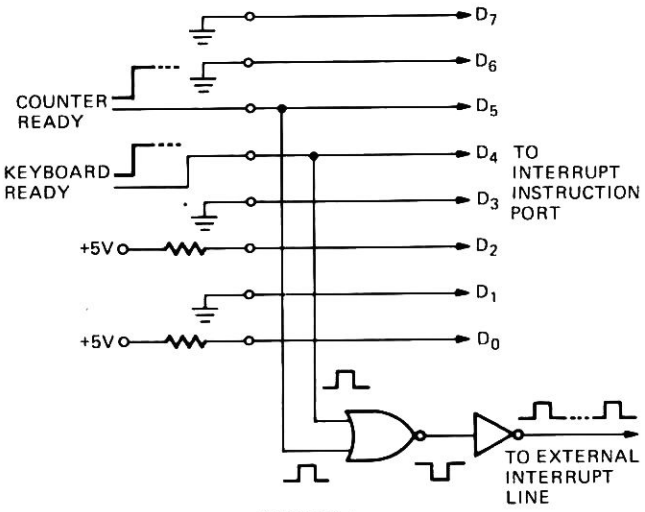


FIGURE 4

Once these checks have been made, try loading a number of different addresses to familiarize yourself with how these addresses are loaded. Then load HI = 00 and LO = 000 and use the EXAMINE switch to examine the next few locations. The memory data should change to indicate changes in data. Return to HI = 00 and LO = 000 and note on a piece of paper the data currently stored in the next seven locations. Reload the starting address and now deposit some data in the locations. The data may be random or all the same, but be sure to note it also on your paper. After the data has been deposited go back and check to see if it is really there. If it is not, check to be sure that the memory address advances each time the DEPOSIT or EXAMINE is actuated. Also check the connection from Write Out on the top of the Address/Manual module to the top of the Memory module. If data is still not being deposited, check the solder connections and jumpers on the Memory module.

We are now ready to try the complete computer. Turn the power back on and with the following switch settings, load the program below. Set INTERRUPT/JAM to JAM and set RUN/SINGLE STEP to SINGLE STEP.

```
Load LO = 000
Load HI  = 000
Deposit  = 104
Deposit  = 000
Deposit  = 000
```

This short program is used to test the operation of the microprocessor. After it has been loaded, set the switch register to 005, return from the JAM mode to the INTERRUPT mode, actuate INTERRUPT, and then actuate SINGLE STEP. Now as you actuate the SINGLE STEP switch, the program should run through and then jump back to the start at location 00 000. If the program does not run turn off the power and check the connections to the CPU module and be sure that all components and wires are soldered through the board on both sides.

The next program will test the computer in a more dynamic mode. It will increment the contents of a register and will then display it on the LED output for output port 0. The program gives the mnemonic as well as the octal and binary instruction equivalents. Here again, load the HI and LO starting addresses, enter the first data word, depress DEPOSIT, enter the next data word, depress DEPOSIT, etc., until all of the program instructions have been entered.

All the numbers in the program (program is located at the top of the next page) are in octal notation, so remember that there are NOT any memory locations between 00 007 and 00 010.

When you have finished depositing the data, set the switch register to 005, switch JAM to INTERRUPT, depress INTERRUPT, depress SINGLE STEP and then switch to the RUN mode. The computer should now be counting or incrementing the output port 0, making it look like a binary counter. Enter 377 or 000 on the switch register

REGISTER INCREMENT TEST PROGRAM

This program starts at location 00 000 and it is used to test the dynamic operation of the Mark-8 Minicomputer.

Address	Mnemonic	Octal	Comments
00 000	LDCI	026	—Load C Immediate
00 001		000	—Data to be loaded
00 002	LDAI	006	—Load A Immediate
00 003		000	—Data to be loaded
00 004	LDBA	310	—Transfer A to B
00 005	INCB	010	—Increment B by +1
00 006	LDAB	301	—Transfer B to A
00 007	OUTϕ	121	—Output A to port 0
00 010	LDBA	310	—Return A to B
00 011	LDAC	302	—Transfer C to A
00 012	ADDI	004	—Add Immediate
00 013		001	—Data to be added
00 014	JPTC	140	—Jump if Carry is True
00 015		005	—LO Address of jump
00 016		000	—HI Address of jump
00 017	JPUN	104	—Unconditional jump
00 020		012	—LO Address of jump
00 021		000	—HI Address of jump

and actuate INTERRUPT. The program should stop. Enter 300 on the switch register and actuate INTERRUPT. The program should again start to execute.

If all of the checks have been made and the computer operated all of them correctly, the construction and debugging of the MARK-8 Minicomputer has been completed. You may now use the computer in any application you wish.

Data output

We have already covered the input of data into the Mark-8 computer, but little has been said about the output. The four output ports are located on the Output module, allowing us to attach up to four different devices to the Mark-8 that will use the data from the computer. Eight bits of data are sent from the CPU to the four groups of latches where the data may be held. We designate the appropriate latch with the OUT command and our jumpers determine which latch is activated by the computer. As we have previously seen, the computer has an instruction that has three M bits in it which are set to the binary equivalent of the output port we wish to send the data to. An instruction 01 010 011 would cause the eight bits of data to be latched at output port 1, since the underlined part of the instruction is 001 = 1. We may also add extra output ports to the minicomputer by adding an extra Output module. Three of the sets of latches on the additional module may be used. Remember that we must use the jumpers on these modules to select the code which will activate each output port.

Although eight bits are present at the output of the port, sometimes we only need one or two bits to turn something on or off. The following example (Fig. 5) shows how we could use an output port in this way to turn a lamp on and off:

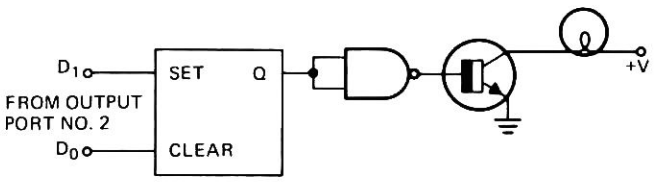


FIGURE 5

If we output an XX XXX X01 on output port 2 we would turn the lamp off, while an XX XXX X10 would turn the lamp on. The other six 'X' bits may be any value since we do not use them. They may be wired to some other device, but using this arrangement we can only control up to four switches with each output port. A more flexible arrangement uses two SN7442 decoders on the output ports. These decode the output signals from the computer into two distinct groups which may be combined to give us control of 64 devices. This is shown in the next diagram (Fig. 6):

With this circuit as shown we could latch out 021 on output port 2 and get a signal from NOR gate I-1. Latching out a 020 would activate NOR gate I-2. These gates could then be used to pulse a flip-flop, a stepper motor or to open and close a valve. The decoders alone can give us the 64 possible combinations, but with a few extra gates

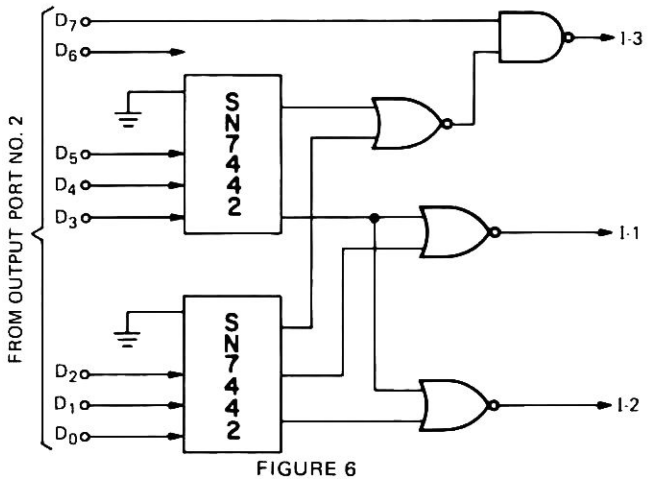


FIGURE 6

we may also use bits D_6 and D_7 to give us 256 possible combinations. The output at I-3 occurs only when we latch out 243 on the output port.

This decoder arrangement is very flexible and it is used when we need to perform some task that does not require all eight bits of data. We have converted one of our output ports to a decoded output and we still have six possible output ports to output eight bits of data to some other instrument such as the TV Typewriter.

Sequencing through each of the outputs shown in the schematic diagram could be accomplished with the following sample program:

CODE	OCTAL	COMMENTS
LDAI	006	—Load register A with the data
	021	—Data to be loaded
OUT2	123	—Output register A to port 2
LDAI	006	—Load register A with the data:
	020	—Data to be loaded
OUT2	123	—Output register A to port 2
LDAI	006	—Load register A with data:
	243	—Data to be loaded
OUT2	123	—Output register A to port 2
HLT	377	—Halt the program

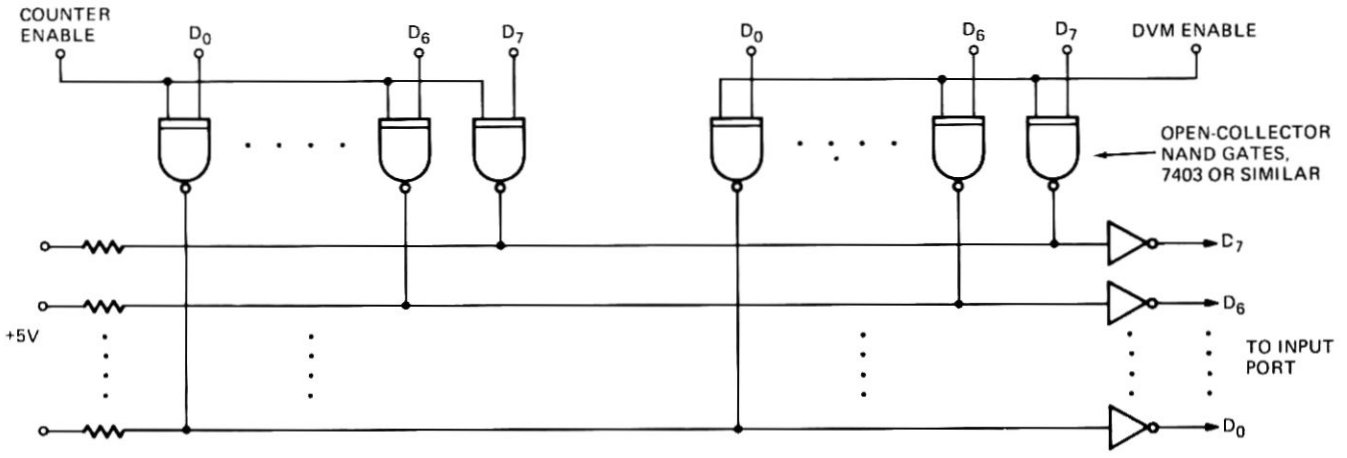
Input

Unfortunately, the minicomputer has only three input ports, one of which has been dedicated to getting the memory data to the CPU. Perhaps the other one has been assigned to a keyboard, leaving only one input port for data input. This input could be expanded using additional multiplexers, but this would add considerable interface logic to our basic computer. It is much easier to use the remaining input port with an 8-bit data bus, similar to our main I/O bus. We could then gate selected data onto the bus as it was needed. Open collector or tri-state gates are used on the bus.

In one of our previous examples, a counter was connected to the computer's remaining input port, but now let's assume that we have also added a digital voltmeter and it, too, will send eight bits of data to the computer. The following bus circuit (Fig. 7) shows how this data would be sent to the computer:

ENABLE STATE		DATA TO CPU FROM
Counter	DVM	
0	0	NONE
0	1	DVM
1	0	Counter
1	1	Not Allowed

We can selectively enable the set of open collector NAND gates (SN7401 or SN7403 types) that gate the data onto the data bus that is connected to our input port. We may select either the counter or the DVM for data transfer to the CPU. In most cases, however, the data will be sent to the computer faster than we can switch between instruments with a manual switch. There must be some method for the computer to select which data is sent to the CPU. We have seen how the computer may be pointed to a program, using the Interrupt Instruction Port and a Restart (RST) instruction. We use this method to distinguish which instrument has data to input to the CPU. Since the computer can now distinguish between the DVM and the counter all we need is some way for the computer to switch between the



8-BIT DATA INPUT BUS
FIGURE 7

Enable lines on the sets of open-collector gates attached to the data bus.

Decoders on an output port could be used to turn a lamp on and off, so why not use this same concept to allow the computer to switch between the two sets of gates. The output of NOR gate I-1 could go to the DVM Enable line and the output of NOR gate I-2 could go to the counter Enable line. To enable each we would use the following steps in our data input programs: (Data input instructions have also been included.)

DVM INPUT PROGRAM		
LDAI	006	—Load register A with data:
	020	—Data to be loaded. 020 is latched
OUT	123	out at this point, activating the NOR gate output, placing the DVM data on the input data bus.
INP1	103	—Input data from port 1 to reg. A

COUNTER INPUT PROGRAM		
LDAI	006	—Load A with data:
	021	—Data to be loaded
OUT	123	—Output data from reg A to port 2
INP	103	—Input data from port 1 to reg A

Now, whenever 020 is output at port 2, we select the DVM and when 021 is output at port 2 we select the counter to input its data on the input bus. We could, therefore, call the DVM device 020 and the counter device 021. We could also use the extra function, the output from I-3 to reset the counter after the data has been taken in by the CPU.

Other input devices could easily be added to the input data bus, just by adding extra sets of eight open-collector NAND gates to the bus. Extra control functions could also be added to the decoded output port by adding NOR and NAND gates as needed.

The input and output type of programming just illustrated shows the power of the Mark-8 minicomputer. It can input and output eight bits of digital data and it can control external events such as on and off control for switches, opening and closing valves and enabling other devices.

Programming

Programming is a skill that can only be learned well by doing. A good point to start learning about programming is by disecting some programs or parts of some programs. The programming examples in the Intel User's Manual are useful for this. Try and understand the BCD-to-Binary and the Binary-to-BCD routines in the Bootstrap Loader program. The register increment program given as a test program earlier can also be disected to understand how it works. The program could have been written to do the incrementing and outputting in other ways. Can you write a short program to do this? Try to substitute other instructions while understanding how the program operates. Try writing short programs that use other, new instructions. These short programs will introduce you to the power of the Mark-8. Try also to use the subroutine pointer register and some of the four flags in your programs.

All of your programs must be entered into the computer through the switch register using the manual controls. If you have a numeric or an ASCII keyboard, try and write a program that will allow you to enter program data into the computer from the keyboard. This will certainly simplify the task of programming.

There are seven general-purpose registers in the CPU—A, B, C, D, E, H and L. These are used for temporary storage of eight bits of data when programming. The H and L registers are special since they are used to point to memory locations during the execution of a program. The H-register contains the HI bits of the memory location and the L register contains the LO bits of the memory location. These registers must be loaded with an address prior to the execution of an LMr, LrM or LMI type of instruction. We have used the LAI instruction before in our examples.

The following example shows how these two registers are used. We wish to load a memory location, 01 023 with data, 137. We first must load H and L registers with the address and then load the data into that selected address.

LDHI	056	—Load register H with data:
	001	—Data to be loaded (HI address)
LDLI	066	—Load register L with data:
	023	—Data to be loaded (LO address)
LDMI	076	—Load memory address pointed to with data:
	137	—Data to be loaded to memory

The H and L registers are internal to the 8008 microprocessor, as are the other temporary registers. They are NOT the HI and LO address latches that we have talked about before. When we are programming we always must specify a memory location with the address in the H and L registers before anything is done to that location.

Other basics of the programming instructions are included in the Intel User's Manual. When looking at some of the programming examples in the User's Manual, note that some of the instruction codes are in DECIMAL and not octal notation that has been used throughout this article. Use the octal codes as much as possible since the binary-to-octal and octal-to-binary conversion is a snap.

Octal number system

Before you attempt to understand the octal number system, you should understand the binary number system used extensively in computers. If you need to review the binary system, see *Computer Architecture*, by Caxton Foster, mentioned previously.

The Mark-8 is an 8-bit minicomputer, and each of its eight bits can exist in only one of two possible states, 1 or 0. You have probably noticed in the Intel User's Manual that the instructions are extensively noted in straight binary format, such as 01 011 110. This is an excellent way to become familiar with the instructions, but it leaves a great deal to be desired when communicating between programmers and computers. We have, therefore, decided to use the OCTAL numbering system to represent the program instructions and data as presented in this article.

The octal numbering system breaks the eight bits of the Mark-8 up into sets of 2, 3 and 3 bits each. Each set has a value assigned for each bit as shown below:

Value =	2	1	4	2	1	4	2	1
Bit =	D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
	Set 3		Set 2			Set 1		

We can now convert any 8-bit binary number to an octal number by placing the values over the numbered bits and adding the value if a 1 is present and adding zero if a 0 is present. Thus to convert the binary number 10 111 101 to octal we would have Set 3 = 2, Set 2 = 7, and Set 1 = 5. Our octal number would then be 275. You will note that the largest binary number we can have for our eight bits is 11 111 111 which converts to 377.

Cover the right hand side of this column and test your conversion skill with the following numbers:

Convert to octal	
11 101 001	351
00 000 000	000
00 011 111	037
Convert to binary	
352	11 101 010
273	10 111 011
105	01 000 101

MARK-8 MINICOMPUTER EXPERIMENTS

1. These experiments build upon the information presented in the main article. If you are in doubt about a point that has been discussed, go back and review it before starting the experiments.

2. These experiments use the minimum memory configuration of 256 eight-bit words. More memory may be used, but it is not necessary for these experiments.

3. A keyboard or an output device such as the TV typewriter is *not* necessary for these experiments. If an ASCII keyboard is connected to the minicomputer, connect it to input port O.

4. Most of the programs to be used in the experiments start at address HI=000 and LO=000, so the memory board must have the jumpers in place for the minimum configuration as noted in the main article. Programs are started unless noted otherwise by loading 005 in the switch register (SR), placing the JAM/INTERRUPT switch in the INTERRUPT mode and the RUN/STEP switch in the RUN mode. Depressing the momentary INTERRUPT pushbutton or switch will then vector the computer to HI=000 and LO=000 to start the program. This will be noted in the experiments as 005/Interrupt.

5. Only a few supplies are needed to do these experiments, but the user is encouraged to build upon these experiments to learn more about the computer operations.

- 1 Breadboard (EL Instruments IC breadboard or equivalent)
- 4 Red LED's, MV-5020 or equivalent
- 4 470-ohm 1/4-watt resistors
- 1 SN7404 hex inverter
- 3 SN7400 quad NAND gates
- 1 SN7400 (nonfunctional)
- 1 Datel 98BI digital-to-analog converter (optional)

Experiment No. 1 DATA INPUT

Connect eight wires from input port No. 1 to the breadboard, and load the following program into the computer:

```
HI=000   LO=000   103   Input data from port 1 to A
                  121   Output data from A to port 0
                  000   Halt
```

Start the program with a 005/Interrupt. Connect some of the wires to +5 volts and some to ground. Depress the INTERRUPT pushbutton and the data set on the lines at the breadboard should be transferred to the LED's on the front corresponding to port 0. Change the data set on the lines to something different and again depress the Interrupt pushbutton. Does the data change? Is it equivalent to the data now on the eight input lines?

Change the Program to:

```
103
121
104     Unconditional jump to:
000     LO 000
000     HI 000
```

This will continuously run the program once you start it with a 005/Interrupt. Data is continuously taken from the input port, held in register A and then output to the LED's on port 0. As you change the data on the eight input lines, the LED's will also change. Place the RUN/STEP switch in the STEP mode and depress the momentary Step switch to step the computer through the program. Change the input data while doing this. Can you see when the data comes into the computer and when it is sent to port 0?

This program will only input one word at a time and it is only temporarily stored in register A. We often wish to store the data in memory for use at some later time. This is done with the following program which takes in an 8-bit data word and stores it. We can later go back and check the memory location to confirm that the data has indeed been stored there.

```
056     Load H Immediate
000     H=000
066     Load L Immediate
100     L=100
103     Input data to A from port 1
121     Output from A to port 0
370     Load data from A to memory
060     Increment L (L=L+1)
000     Halt
104     Unconditional jump
004
000
```

In this program we load registers H=000 and L=100 to act as the address for memory storage, so our data file will start at address 100. Set some new data on the eight input lines and start the program with a 005/Interrupt. This will start the program and the first data word will be stored in location 100. Change the SR setting to 300 (Continue) and now each time the INTERRUPT pushbutton is depressed the eight bits set on the breadboard will be stored in sequential memory locations, 101, 102, 103, etc. Store about 10 different words of data and then stop the computer, a 000/Interrupt will do this, and then load HI=000 and LO=100 and use the EXAMINE key to check your data. Does it correspond to the data that you set on the breadboard? To restart the program, use a 005/Interrupt. This will also re-initialize the storage address back to 100.

Data that has been stored could have come from a keyboard, a digital voltmeter (DVM) a digital clock or even a digital combination lock. This experiment demonstarted how a computer inputs data and also how data files are built up for future use.

Experiment No. 2 DATA OUTPUT

We have already used a data output statement in the programs in Experiment No. 1 on data input. Data has been sent to output port 0 which drives a row of eight LED's for data display. This visual data can give us an indication of the computer's operation or a display of data for a visual check. Output ports 1–4 are used to output TTL signals to external devices for our use in real operations.

Connect four output lines from output port 3, bits D_0, D_1, D_2, and D_3, to the breadboard and the circuit shown in Fig. 8.

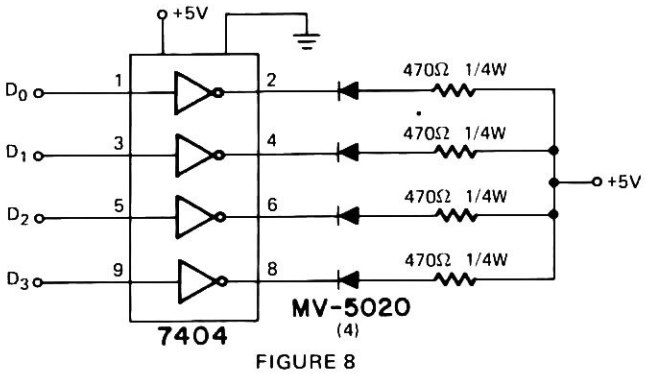


FIGURE 8

When the output data is logic 1, the corresponding LED will be ON and if the data is logic 0, the LED will be OFF. Load the following program into the computer and start it with a 005/Interrupt;

```
006     Load A Immediate
000     Data
127     Output to port 3
000     Halt
```

This loads A with 000 and outputs this to the LED's on port 3 at the breadboard. Go back and change the data in location 001 in the program to something different and restart with a 005/Interrupt. Remember that only bits 0 to 3 are used so changes in the other four bits will not be seen on the LED's. Confirm that your new data is sent to the four LED's. Now change the program to:

```
103     Input from port 1
127     Output to port 3
000     Halt
```

Start with a 005/Interrupt and the data from the input lines will be output to the four LED's on output port 3. We have used the input lines to get data and we have then sent it out to the LED's for a visual indication. Change the data on bits 0 to 3 of the input lines and confirm that the data is sent to output port 3 when you actuate the INTERRUPT pushbutton. If we now change the program to:

```
103
127
104
000
000
```

we find that when we start with 005/Interrupt, the program will continue to run and as we change the data on the input lines, the LED's will also change. So far, the computer has not interacted with the data, but has only acted to transfer it from one place to another. Let us imagine that our four input lines, D_0, D_1, D_2, and D_3 are connected to a binary encoder that encodes the level of liquid in a tank. Binary 0000 = empty and 1111 = full. We wish to have the computer control the level in the tank so that it will never overflow. We will program the computer to turn on all four LED's when the level reaches the point which gives a binary 1110 code. This will signal the operator to turn off the fill valve. We could also connect one logic output to a relay to turn the valve off if we have an electrical valve. How do we get the computer to detect a 1110 input code? The use of the compare instruction is the easiest. The compare instruction is explained in the Intel Users Manual.

Our new program looks like:

```
006     Load A Immediate
000     Data
127     Output to port 3 (Clears LED's)
103     Input data from port 1
074     Compare with:
016     016₈ = 1110₂
110     Jump if result ≠ 0
003
000
006     If = 0, Load A Immediate
377
127     Output to port 3
000     Halt
```

This program may be loaded and tried. The program will continuously input data from input port 1 and check it with 1110 or 016 octal. If the result is equal to 0, the computer outputs 377 to port 3 and this lights all the LED's. If the result is not equal to 0, the computer jumps back and inputs and checks the next input word. Remember that the unused input lines to input port 1 must be grounded for this program to work.

This experiment has shown you how the computer outputs data and how the computer can make a decision based upon the input data.

We would really like to make the complete process as automatic as possible. Try writing a program to turn the lights off again when the input code is binary 0001. This will allow the computer to refill the tank when it is almost empty.

Experiment No. 3 NAND GATE TESTER

It would be very difficult for both IC manufacturers and users to test large numbers of integrated circuits by hand, so some sort of an automatic tester must be used. In this experiment we will use our concepts of input and output of data and computer decision making to do actual tests on NAND gates to determine if they are bad or good.

The program listed in this experiment will only test an SN7400 quad NAND gate package as being good if all four of the individual NAND gates are operational. Packages with one or more bad gates will be rejected as bad. Wire the test circuit on the breadboard as shown in Fig. 9. A NAND gate is inserted in the breadboard and the computer is started with a 005/Interrupt. The package is removed and the next one inserted and the computer is again started with a 005/Interrupt to test this next package. The result of the test is shown in the LED's on output port 0 as noted in the program.

In this program, our notation will be a bit more sophistcated. We will start to understand the program steps in terms of the mnemonics associated with each step. We will also start to assign names to various program steps to simplify our notation, thus JPUN START means to execute an unconditional jump to the location designated with START. A program step like JPUN START+2 means to jump

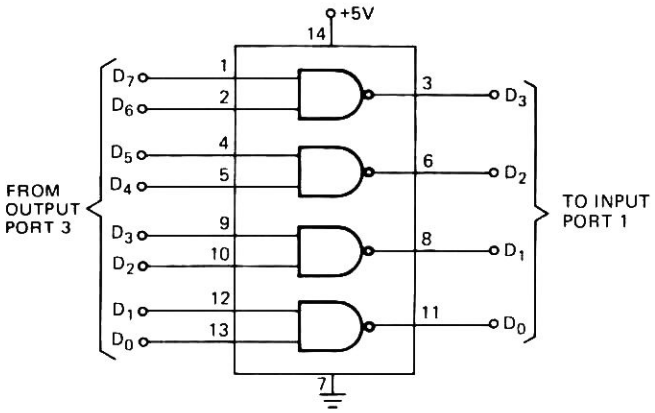


FIGURE 9

to the address that is equal to the address of START with 2_8 added to it.

```
000 016     LDBI          030 047
001 377                   031 000
002 026     LDCI          032 303     LDAD
003 252                   033 106     JSUN Test
004 036     LDDI          034 047
005 125                   035 000
006 046     LDEI          036 304     LDAE
007 000                   037 106     JSUN Test
010 066     LDLI          040 047
011 361                   041 000
012 250     XORA          042 006     LDAI
013 301     LDAB          043 001
014 127     OUT3          044 104     JPUN Bad+2
015 103     INP1          045 055
016 074     COMI          046 000
017 000                   047 127 Test, OUT3
020 150     JPTZ Next     050 103     INP1
021 026                   051 206     ADDL
022 000                   052 043     RTTC
023 104     JPUN Bad      053 006 Bad, LDAI
024 053                   054 200
025 000                   055 121     OUTϕ
026 302 Next, LDAC        056 000     HALT
027 106     JSUN Test
```

Once you have loaded the program, test a NAND gate and see if it is good or bad. If it is good, the right-most LED in output port 0 will be lit. If it is bad, the left-most LED will be lit.

Try and follow the logic of the program. Test patterns are sent to the NAND gate and then checked against what they should be if the gates are indeed good. This can also be done for other circuit elements such as flip-flops, counters and other types of gates. Try to rewrite the test portions of this program so that it will test NOR gates.

Experiment No. 4 RUNNING TWO PROGRAMS AT THE SAME TIME

It is often necessary for the computer to do two things at the same time. We may require the minicomputer to take in data once a second while it is also doing some calculations or controlling another experiment. This experiment shows how the computer is used to run a main program and to also acquire data at the same time. We will use the basic register increment program that we used during the computer checkout as our main running program and we will use a shorter program to take in data and store it in memory whenever we actuate the INTERRUPT pushbutton. The register increment program is listed below:

```
250     Exclusive OR A with A (Clears A)
310     Load A → B
320     Load A → C
350     Load A → H
066     Load L Immediate
100
010     Increment B
301     Load B → A
121     Load A → Output 0
310     Load A → B
302     Load C → A
```

```
004        Add Immediate
001
140        Jump on a True Carry
006
000
104        Jump Unconditional
013
000
```

This program loads register H = 000 and register L = 100 as address pointers for our data storage file. The program then will execute the register increment portion of the sequence.

The next part of the program is located in a different section of the memory and it starts at HI = 000 and LO = 050. This program will temporarily store the A register in E, input data from input port 1, store it in memory, return A from E and return to the main program. Now load:

```
000 050    340    Load A → E
           103    Input 1 → A
           370    Load A → Memory
           060    Increment L
           304    Load E → A
           007    Unconditional Return
```

Start the program with a 005/Interrupt and then change the switch register setting to 055. Now, each time the interrupt is actuated, the data from the input port 1 input lines will be stored in memory, starting at location 100. Make a note of your data set at the input lines and actuate INTERRUPT. Change the data, note it, and again actuate the INTERRUPT. After entering a few data words, look closely at the incrementing register, output port 0. Do you see any change in the rate of operation when you actuate the interrupt? The computer execution of our short subroutine is very fast and we do not see any visible change. Calculate how long it takes the computer to run through the short routine starting at location 000 050.

This experiment has demonstarted how we can perform two, or even more tasks with the computer without any visible change in the computer operation. This type of programming is called Interrupt Programming since we interrupt the regular program, perform some other task and then return to the main program.

Using the same main program, can you change the subroutine at 000 050 to output data from locations 100 and on up to the LED's on output port 3? Hint: Only two instructions need to be changed. We will, of course, only display the four bits, 0 to 3. Does the data output agree with the data you just stored?

Experiment No. 5 PROGRAMMABLE FUNCTION GENERATOR

An inexpensive, 8-bit digital-to-analog converter (D/A) such as the Datel type DAC 98BI ($9.95, Datel Systems, Inc., 1020 Turnpike St., Canton, Mass. 02021) may be used to convert each of the possible 2^8 or 256 input conditions to an analog voltage proportional to the value of the binary input. As the input conditions change, the output voltage also changes and this may be used to give us complex waveforms under computer control. Programs are given to generate a positive ramp, a negative ramp, a triangular wave and a complex waveform designed by the user. Each of these programs may be treated as a subroutine by removing the jump statements that cause the program to repeat itself and inserting a RTUN or return statement. The individual statements may then be called as subroutines to generate a very complex set of functions, such as three triangular waves, two positive ramps and a user defined wave. This can also be done continuously or in a burst, by making the program that calls the subroutines jump back and restart itself or by halting it after one complete cycle through the program.

Positive Ramp Generator

```
000    010    INCB
001    301    LDAB
002    127    OUT3
003    104    JPUN
004    000
005    000
```

Negative Ramp Generator

Replace the first instruction with 011, DECB

Triangular Waveform Generator

```
000    010    UP, INCB
001    150        JPTZ DOWN
002    011
003    000
004    301        LDAB
005    127        OUT3
006    104        JPUN UP
007    000
010    000
011    011    DOWN, DECB
012    150        JPTZ UP
013    000
014    000
015    301        LDAB
016    127        OUT3
017    104        JPUN DOWN
020    011
021    000
```

A staircase waveform is generated by incrementing in large steps or by adding a fairly large number to itself a number of times. This is shown in the following staricase generator program:

```
000    006    START, LDAI
001    000
002    004        ADDI
003    040
004    127        OUT3
005    104        JPUN START
006    000
007    000
```

The memory may also be used to store binary values that are to be output to the D/A converter. In all the previous experiments we have only used the registers for data storage. Load the following program into the computer and execute it with a 005/Interrupt.

```
000    056    START, LDHI
001    000
002    307        LDAM
003    127        OUT3
004    010        INCL
005    104        JPUN START
006    000
007    000
```

The output voltages now observed on your scope will correspond to the voltages given by the binary values stored in the 256 memory locations from 00 000 to 00 377. Stop the program and load 000 in locations from 100 to 130. Restart the program with a 005/Interrupt. Do you observe these values as a voltage output on the scope? Try to load some different values and observe them in the same way. Can you get this program to output a triangular wave from the data stored in memory? Other complex waveforms may also be generated by using the memory in this way. We call this a "look-up table" since the computer looks up each value and outputs it to the D/A converter.

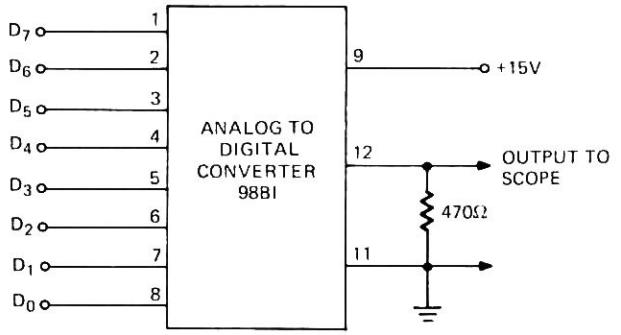


FIGURE 10

Experiment No. 6 CODE CONVERSION

Unfortunately, not all communication codes are the same. Radio amateurs and some commercial communication systems use a five bit binary code called Baudot. Most of the readily available character generators for use in graphic terminals such as the TV typewriter use the ASCII code system. Let's assume that we are receiving Baudot code and we want to convert it to ASCII for use with the TV Typewriter. We will input the five bits of Baudot code in parallel at input port 1 and we will output the equivalent ASCII code on output port

0 for visual check. Some of the equivalent Baudot and ASCII codes are given below. We only need a few for this illustration.

	BAUDOT	ASCII
A	030	301
B	023	302
C	016	303
D	022	304
E	020	305

As you can see,,there is no easy correspondence between the two codes, so the computer will have to do some of the conversion for us. There are two methods that could be used to convert the codes. The first one would input the character and then compare it with all possible characters and when a match was found it would output the correct code. This type of approach takes a great deal of memory, about 46 memory locations just to convert the five letters above, and it is also slow since our comparisons may be at the bottom of the list and all the other comparisons must be made first.

The second approach uses the look up table method that we used in Experiment No. 5 to store values in the memory. Since the Baudot code has only five bits, we will neglect shifts in this example, we know that there are only $11\ 111_2$ different characters with octal values between 000 and 037. We will use the octal value of the Baudot character as a memory address, thus using computer memory addresses 000 to 037. Now each input character has its own memory address that is associated with it. We now insert in each of the memory locations the ASCII equivalent of the Baudot character which serves as the address. If we were to look as a section of memory we would see:

Memory address	Data
016	303
.	.
.	.
020	305
021	332
022	304
023	302
.	.
.	.
030	301

Now we treat each input character as if it were a memory address. We get the equivalent character out of memory and output it to output port 0. The following short program will do this:

```
000   056   LDHI
001   000
002   103   INP1
003   360   LDLA
004   307   LDAM
005   121   OUTϕ
006   000   HALT
```

Now the conversion program requires 32 storage locations for the look-up table and only seven program steps to do the conversion. We can convert all 32 characters with only 39 program steps. With the compare-type program it took more than that just to convert five characters. Using the input port 1 lines, try the above program with either the five Baudot code words listed or write your own code.

The longer program with the compare instructions is often called a command decoder type program. It is useful to input a keyboard character and perform some action in many programs. The command decoder decodes the key and jumps the program to a subroutine or other program to perform some other action.

Experiment No. 7 DATA INPUT AND DATA DISPLAY

In some experiments we wish to take data with an instrument, an ohmmeter, a voltmeter or a thermometer. But we wish to take the data at a rate which is faster than we can possibly write it down for later use. Let us assume that we have a digital voltmeter (DVM) and we are using it to measure voltage changes in an experiment. We would like to take 10 readings a second for 10 seconds. This is quite a bit faster than we can possibly record the data by hand, so we use the Mark-8. We will assume that our DVM converts at the rate of 10 readings per second so an external timer will not be needed.

The voltage changes will be significant enough so that we will be able to use the most significant two and a half digits of the DVM for data input to the Mark-8. Unfortunately, the 2½ digits use nine binary bits of information, four each for the BCD digits and one for the half digit which may be either a 1 or a 0. The maximum count is, however, 199 at the DVM which is less than the 256 possible states for an 8-bit binary word in the computer. If we can convert the nine BDC input lines to binary we will save storage space, and the resulting binary code will be compatible with our D/A converter that will be used to display the data on our scope.

To interface the DVM to the Mark-8 we make the connections as shown in the following figure. The nine data lines from the DVM go to two input ports and the strobe line from the DVM, signalling the end of a conversion, is used to interrupt the computer. The outputs are the same as those used in the function generator experiment.

The program at the end of this experiment will control the data acquisition by the Mark-8. Since we want 10 readings per second for 10 seconds, 100 readings must be taken. At the end of this time, the Mark-8 will jump to a display program to display the data as analog voltages on the oscilloscope. Comments have been included in the program to make the operation clear for the user.

The data will not be displayed until all 100 samples have been taken. Could you modify the basic program so that the data is constantly displayed as it enters the computer? Hints: 1) first clear out the storage registers and 2) let the interrupt from the DVM vector to the data input and conversion program. Treat the display program as the main operating program.

DVM Data Input and Display Program

```
000  056  BCDBIN,  LDHI            /Initialize H address pointer
001  000            000
002  066            LDLI            /Initialize L address pointer
003  233            233
004  046            LDEI            /Set up data word counter = 100
005  144            144             /144₈ = 100₁₀
006  026  RESTART,  LDCI            /Clear data storage accumulator
007  000            000
010  103            INPϕ            /Input the DVM half-digit (MSD)
011  074            COMI            /Compare MSD with 0
012  000            000
013  150            JPTZ I  INPUT   /Jump on a true 0
014  022            022
015  000            000
016  302            LDAC            /Not = 0, program jumps here
017  004            ADDI            /Add immediate 100₁₀ to A
020  144            144
021  320            LDCA            /Restore data to C
022  103  INPUT,    INP1            /Input 2 other digits
023  310            LDBA            /Store in B
024  044            ANDI            /Mask out w. 017 to get LSD
025  017            017
026  202            ADDC            /Add LSD to C
027  320            LDCA            /Store result in C
030  301            LDAB            /Get data back from B
031  044            ANDI            /Mask out w. 360 to get middle digit
032  360            360
033  150            JPTZ    STORE   /Middle digit = 0, store binary data
034  052            052
035  000            000
036  012            RART            /Data was ≠ 0, so rotate 4 times right
037  012            RART            /to shift it to the least significant
040  012            RART            /digit position
041  012            RART
042  330            LDDA            /Store rotated data in D
043  302            LDAC            /Get C back to A
044  004  TENADD,   ADDI            /Add 10 to C for each digit in D
045  012            012             /10₁₀ = 12₈
046  031            DECD            /Decrease digits in D by 1
047  110            JPFZ    TENADD  /Digits still left, loop thru again
050  044            044
051  000            000
052  372  STORE,    LDMC            /Load C to memory
053  060            INCL            /Increment memory address by 1
054  041            DECE            /Decrement 100 points counter by 1
055  150            JPTZ    DSPLA   /If E = 0, display data
056  120            120
057  000            000
060  000            HALT            /Halt and wait for next DVM interrupt
061  104            JPUN    RESTART /Restart data input after interrupt
062  006            006
063  000            000
120  056  DSPLA,    LDHI            /Load H address pointer
121  000            000
122  066  LOOP,     LDLI            /Load L address pointer
123  233            233
124  307  OUT,      LDAM            /Get first Binary data word to A
125  127            OUT3            /Output data to D/A converter
126  060            INCL            /Increment memory address by 1
127  150            JPTZ    LOOP    /If L = 0, reinitialize
130  122            122
131  000            000
132  104            JPUN    OUT     /If L ≠ 0, output next data word
133  124            124
134  000            000
```

Experiment No. 8 COMPUTER GAMES

The Mark-8 Minicomputer may be programmed to play games since it can rapidly make decisions based upon new input data. One simple game that can be played on the Mark-8 is a variation of the

old "take-away" type of game, Nim. Nim is played by specifying a set number of sticks to be used, toothpicks, pennies or any convenient indicator may be used, the number being 22, 23 or 24. Once the the number of sticks have been set down, players remove 1, 2, 3 or 4 sticks until only one is left. The player forced to take the last stick is the loser.

The program given at the end of this experiment will consistently win at Nim. After loading the program into the computer, arrange the number of sticks you have chosen and indicate to the computer the number you will be playing with by using input lines D_2, D_1 and D_0. The number of sticks is entered as 22 = 2 (010), 23 = 3 (011) and 24 = 4 (100) where the binary number is entered into the computer. Be sure that all the other input lines have been grounded. Start the game with a 005/Interrupt and the computer will indicate on the output port ϕ LED's the number of sticks it will take away, again in binary notation. Now load 300 on the switches in the switch register and after you have entered your next move on the input lines, depress Interrupt and the computer will indicate its next move. Continue until only one stick is left.

This is a simple game type program that makes decisions based upon the number you entered. Can you find the two decisions in the program? The computer assumes that you are an honest player. Suppose that you restart the program and you cheat. What does the computer do? It has not been programmed to detect a cheat in the game. Could you add to the basic program to detect a cheat and halt the program?

Programs could also be written to play more complex games such as Tic-Tac-Toe, but since more decisions must be made, more memory space would be required.

Nim Program

```
000   026        LDCI
001   004
002   103        INP1
003   024        SUBI
004   001
005   121        OUTφ
006   000  INPUT, HALT
007   103        INP1
010   310        LDBA
011   006        LDAI
012   005
013   221        SUBB
014   121        OUTφ
015   021        DECC
016   110        JPFC  INPUT
017   006
020   000
021   000        HALT Game ends here
```

Other applications of the Mark-8 minicomputer

We have seen in the experiments how the Mark-8 can input and output data and how it can make decisions based upon input data and stored data. In the text of the article we have also seen some of the unique features available for the user to make using the Mark-8 easy and fun.

As you probably realize by now, there are many more applications and experiments that could be done with the Mark-8. This section will describe one application and will list some that you might be interested in trying.

Let us develop a sophisticated security system around the Mark-8. We will assume that we wish to protect a home or an apartment and that we have three doors and four windows to monitor. We would also like to use the computer to turn some of the house lights on and off in a sequence to simulate to those outsiders, the presence of someone at home. We must also sound the alarm if power fails.

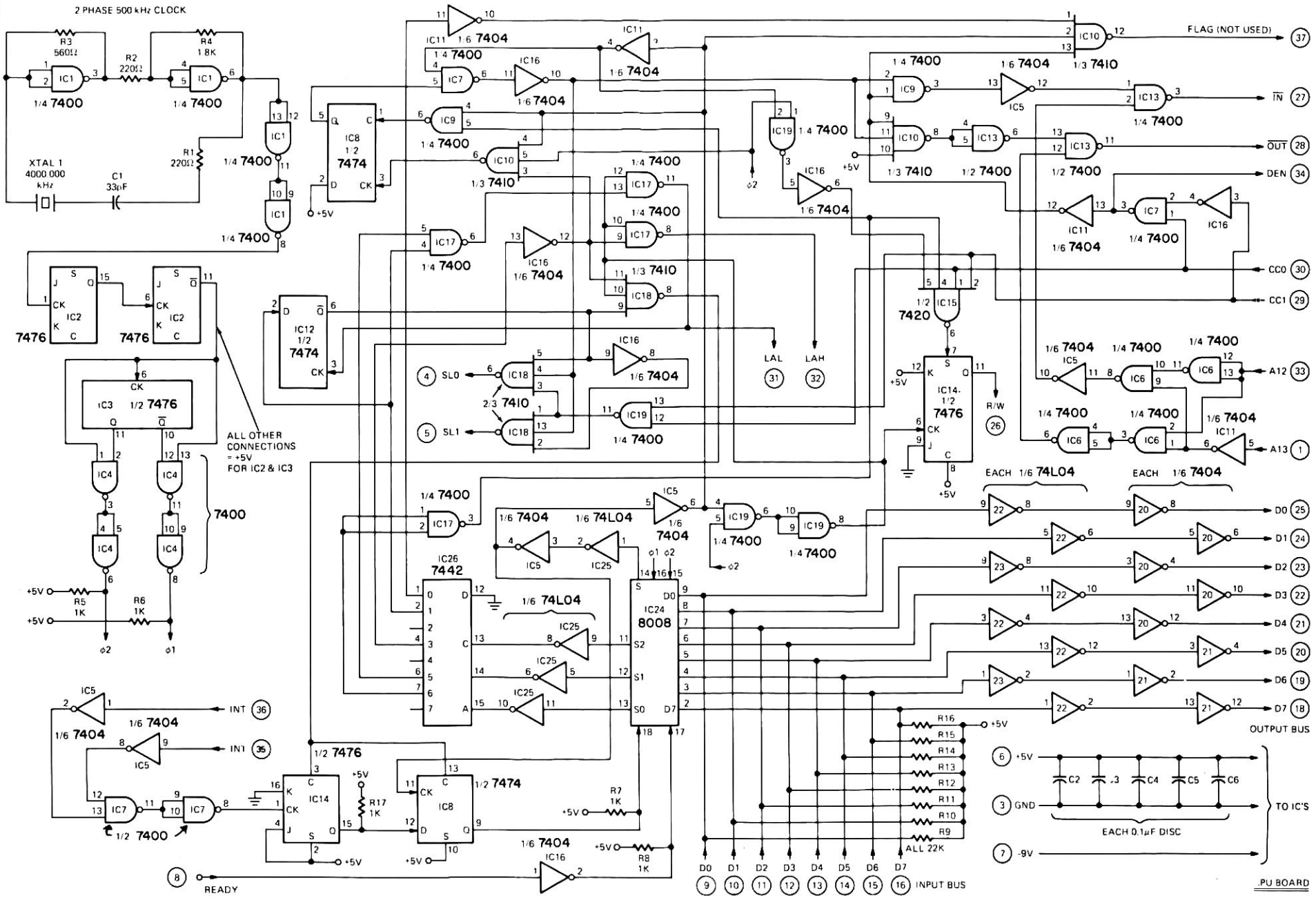
Seven switch closures, one for each door and window are connected to seven input lines on input port 1. The eighth line is connected to a 1-Hz clock that is used by the computer as a timer pulse. Five output lines from output port 4 are connected to solid-state relays to turn house lights on and off according to the bits present at the output. We use a low-power electromechanical relay on one of the remaining output lines to actuate the alarm circuit which is battery powered. The computer monitors each of the switches on a continual basis and it also senses the 1-Hz clock. After 1800 clock pulses, or every ½ hour, the computer outputs a preset pattern to the solid-state relays to change the lights that are on and also those that are off.

The program to do this is not very complicated and we could have simplified it by placing all the switch closures in parallel to the computer. This would leave six open input lines that might be used for a digital keyboard input to make the computer also act like a digital combination lock.

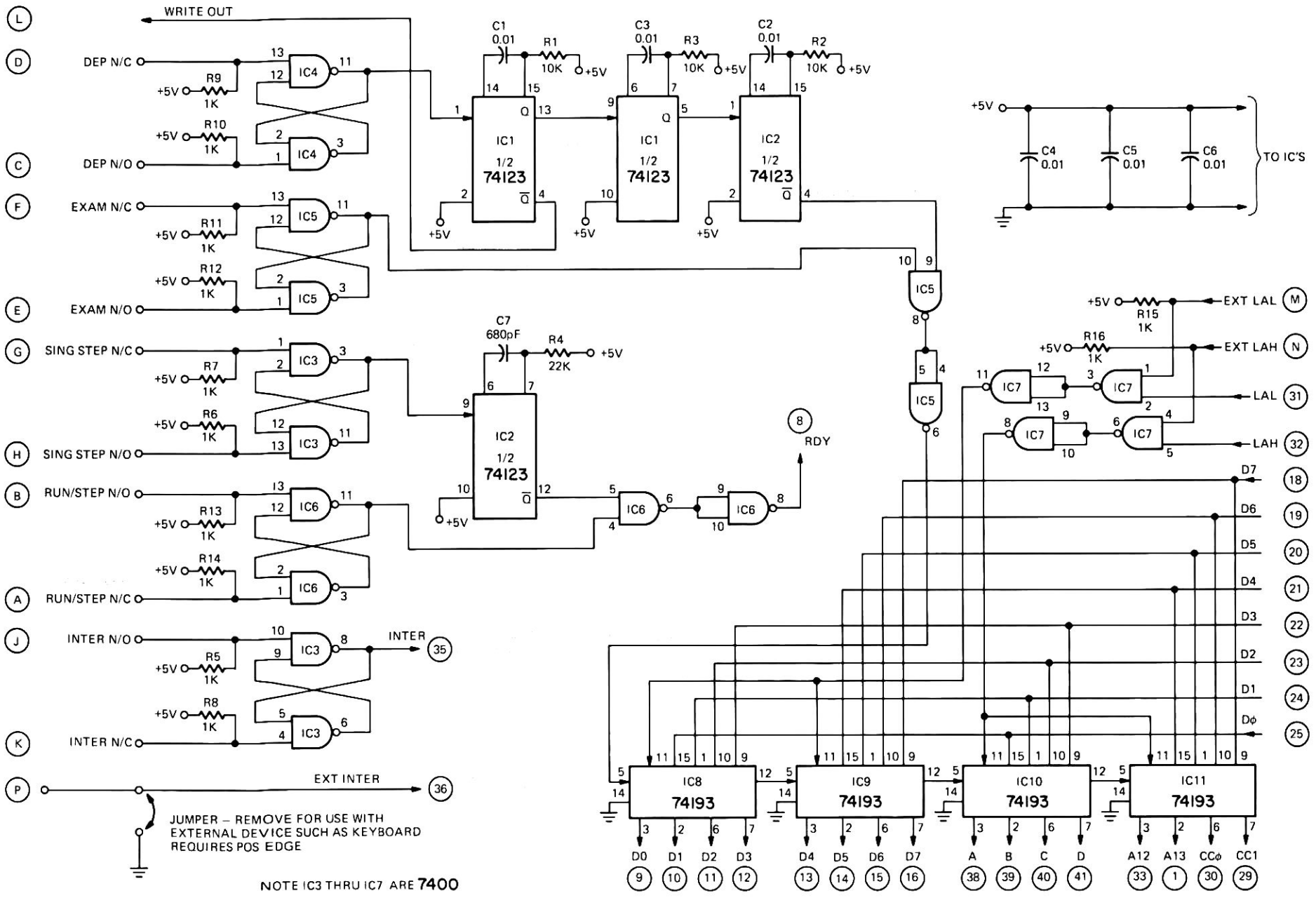
Some other possible applications include:

1. **Minicomputer-calculator interface** to allow complex functions such as log, trig functions and roots to be programmed on regular four-function calculators.
2. **Completely automated dark-room system.** Control of times, temperatures and solution valves.
3. **Educational teaching unit** for beginning programmers.
4. **Laboratory experiment control** and data acquisition system.
5. **Model train controller** and scheduler.

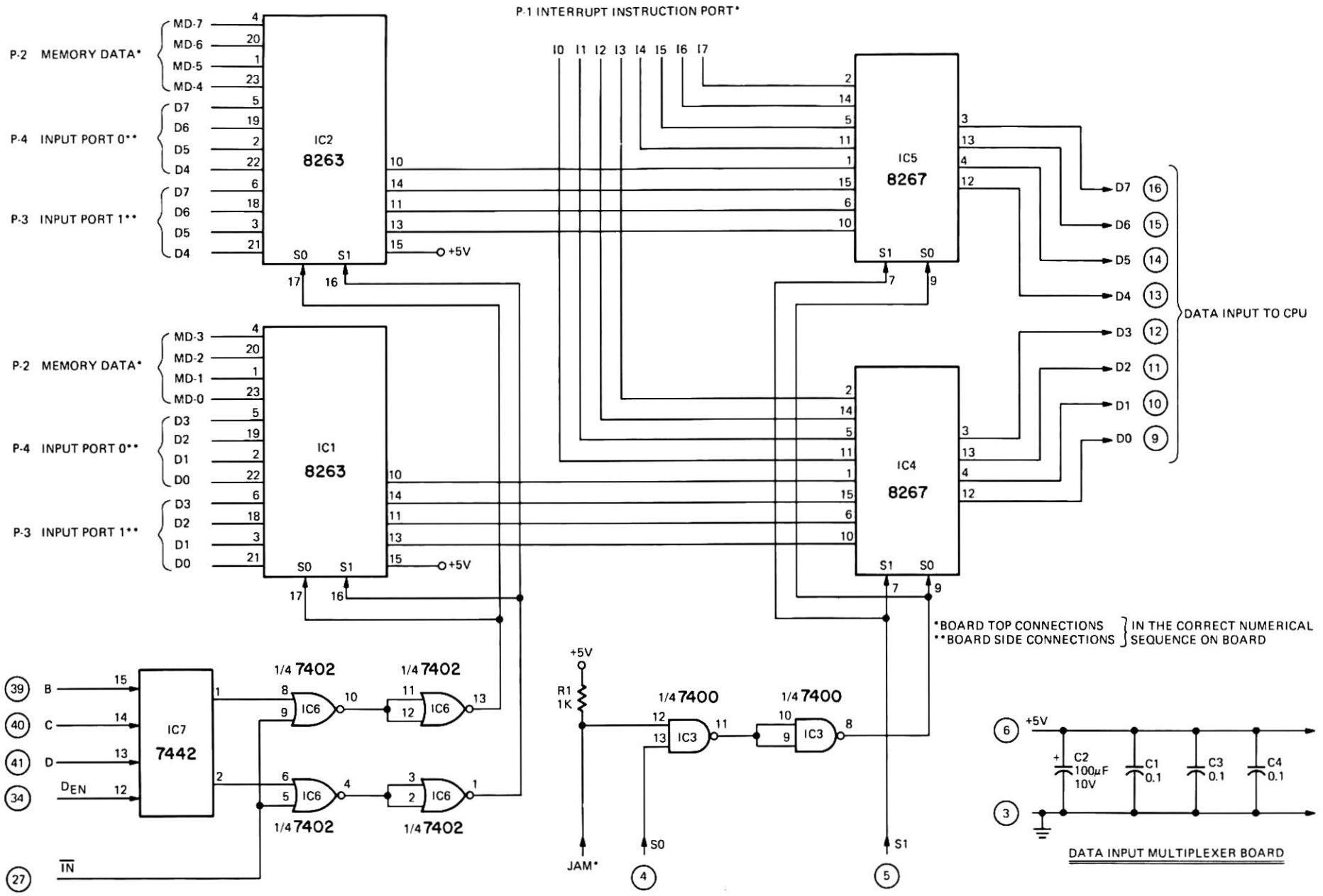
NOTES



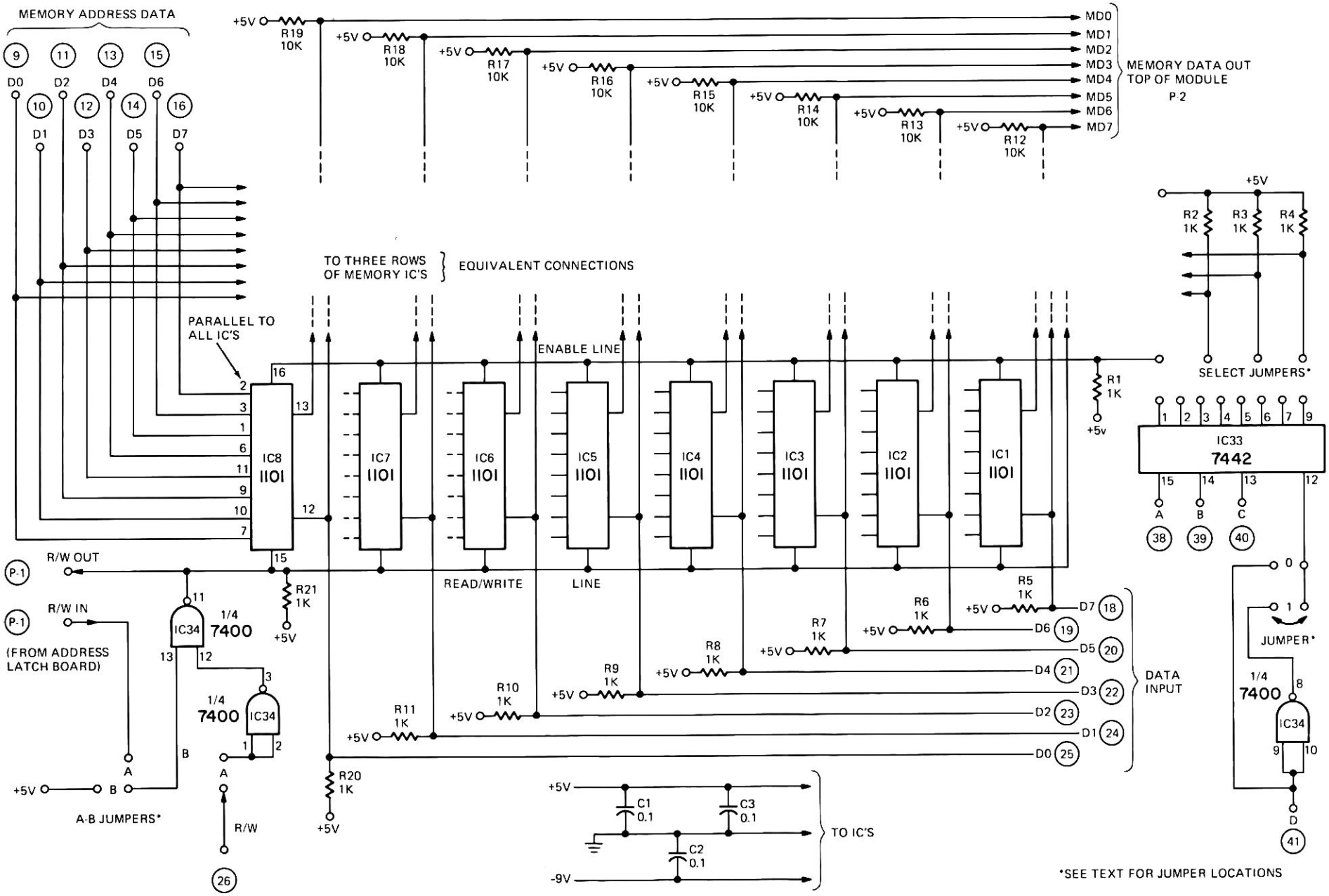
SCHEMATIC – CPU MODULE



SCHEMATIC – MEMORY ADDRESS/MANUAL CONTROL MODULE

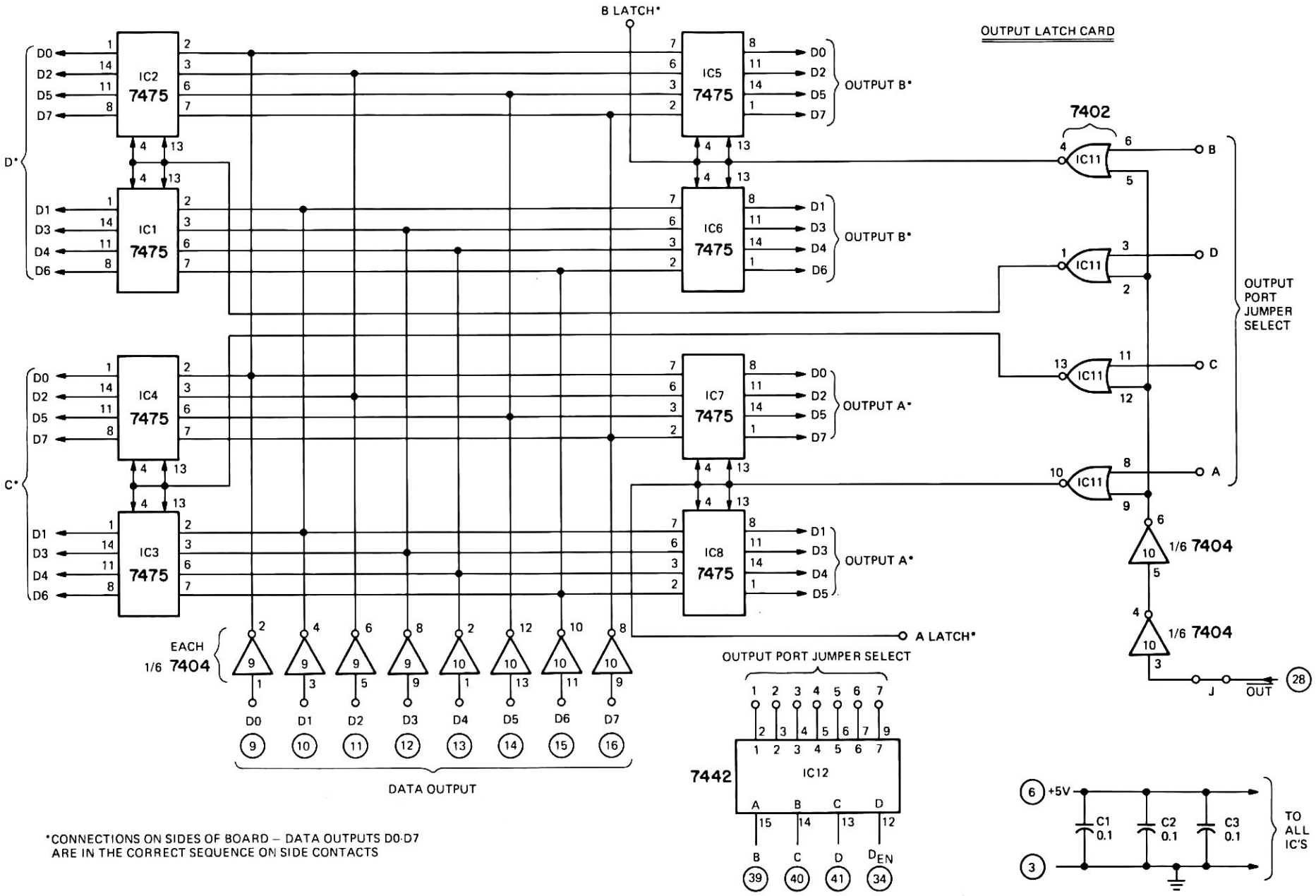


SCHEMATIC – DATA INPUT MULTIPLEXER MODULE



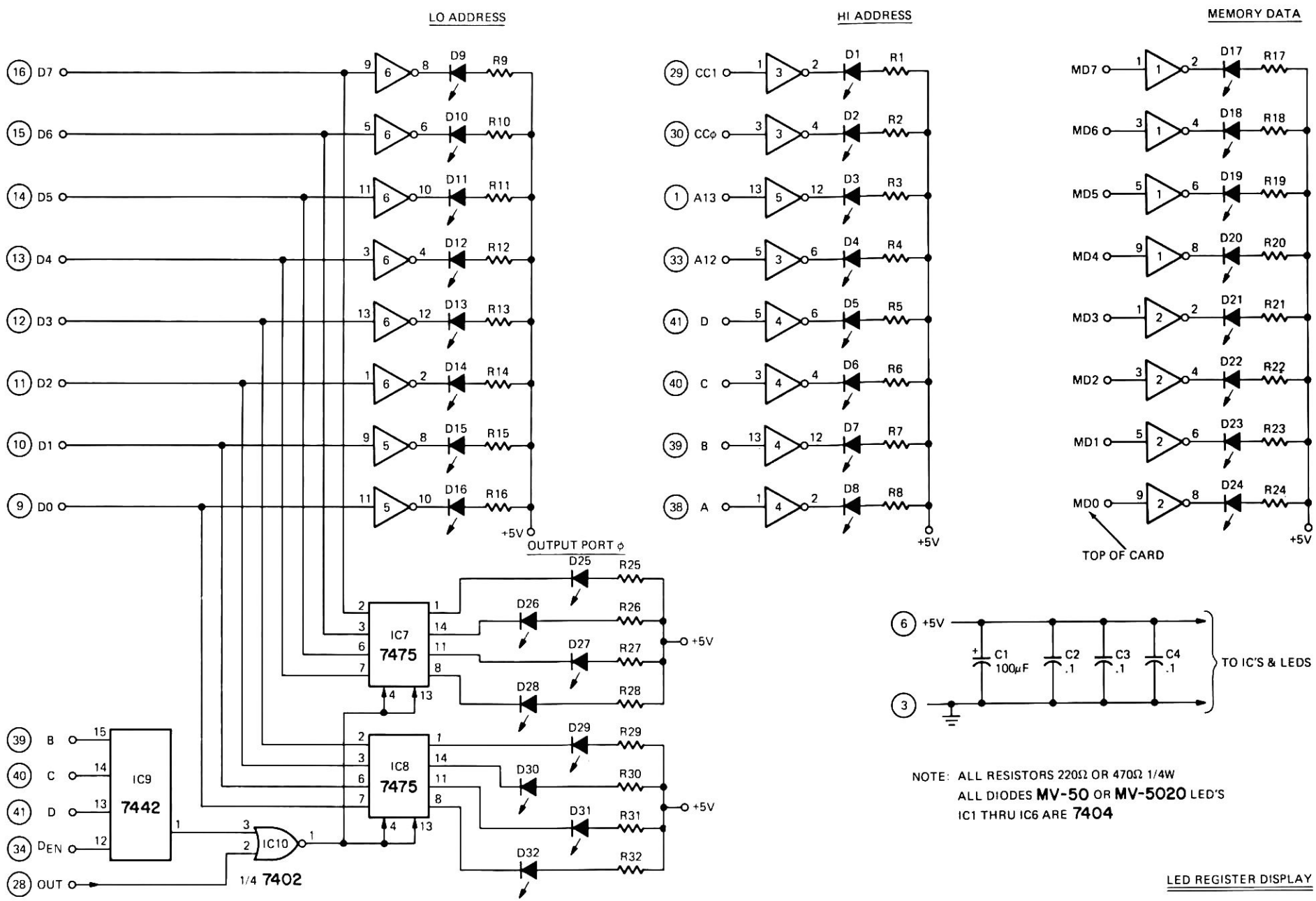
SCHEMATIC – MEMORY MODULE

*SEE TEXT FOR JUMPER LOCATIONS



*CONNECTIONS ON SIDES OF BOARD – DATA OUTPUTS D0-D7 ARE IN THE CORRECT SEQUENCE ON SIDE CONTACTS

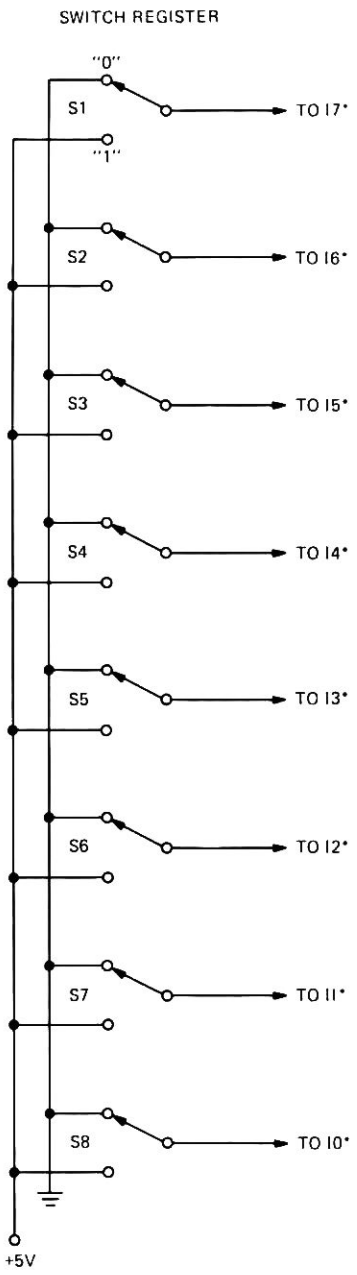
SCHEMATIC – OUTPUT LATCH MODULE

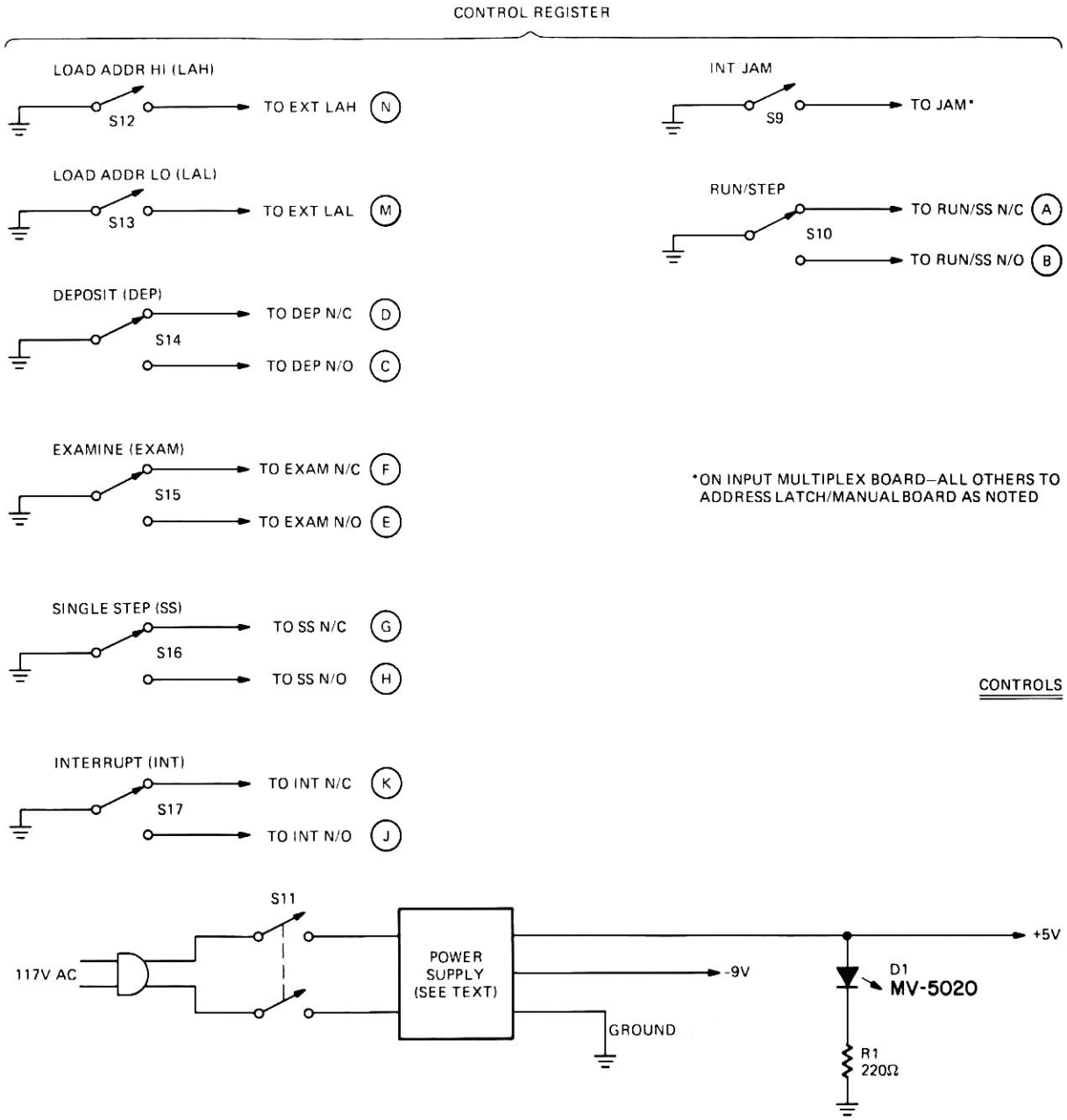


NOTE: ALL RESISTORS 220Ω OR 470Ω 1/4W
ALL DIODES **MV-50** OR **MV-5020** LED'S
IC1 THRU IC6 ARE **7404**

SCHEMATIC – LED REGISTER DISPLAY

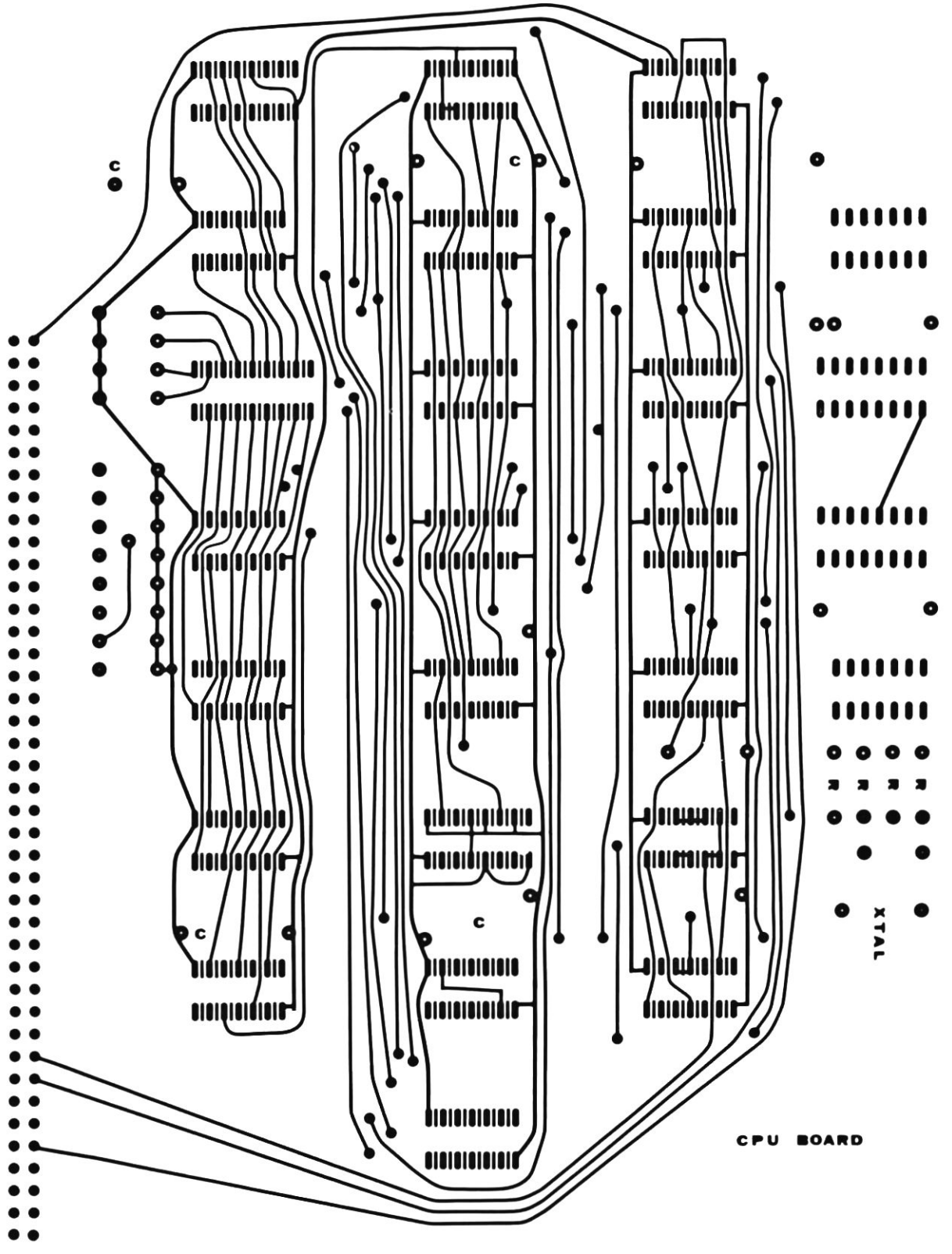
LED REGISTER DISPLAY



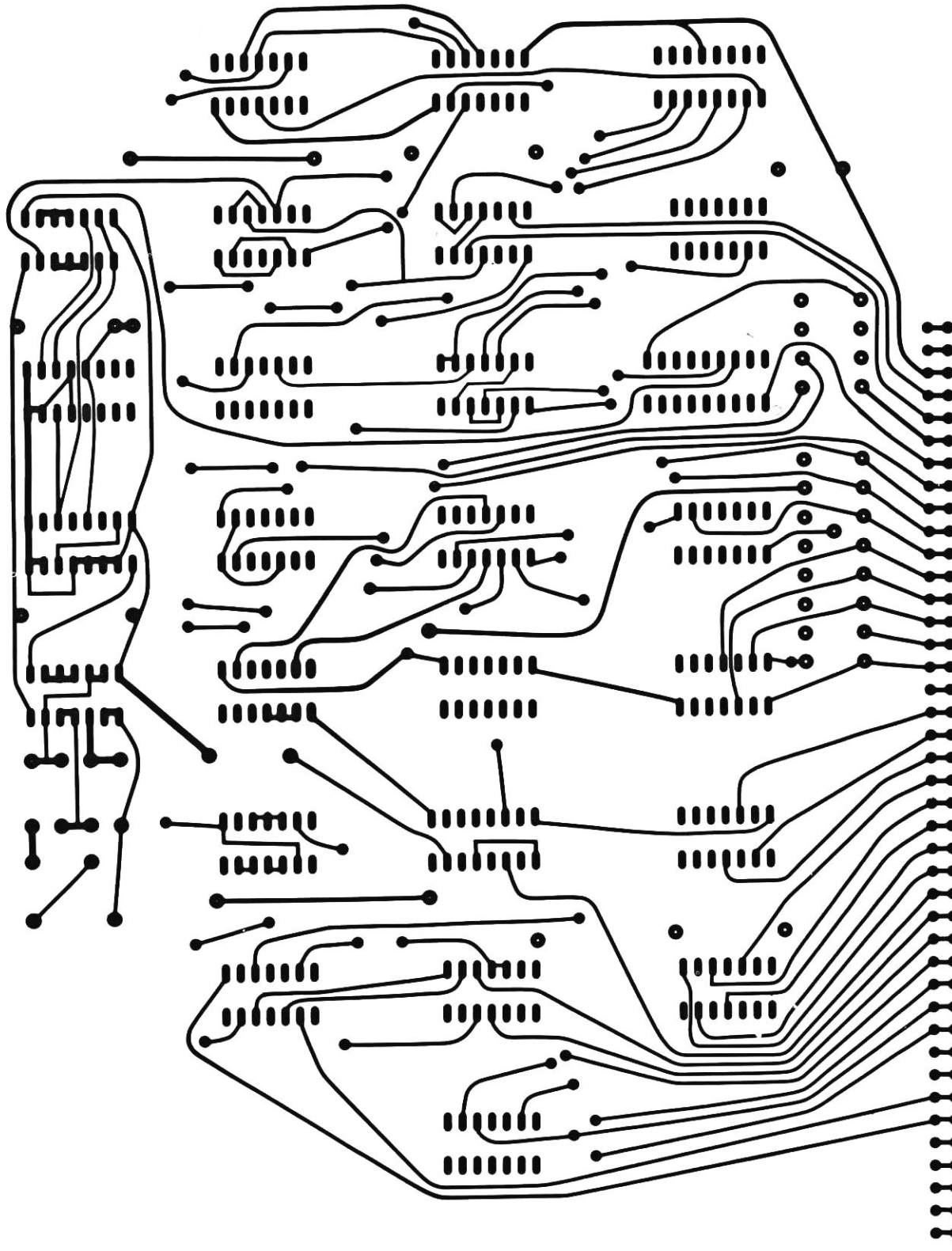


SCHEMATIC – SWITCH REGISTER

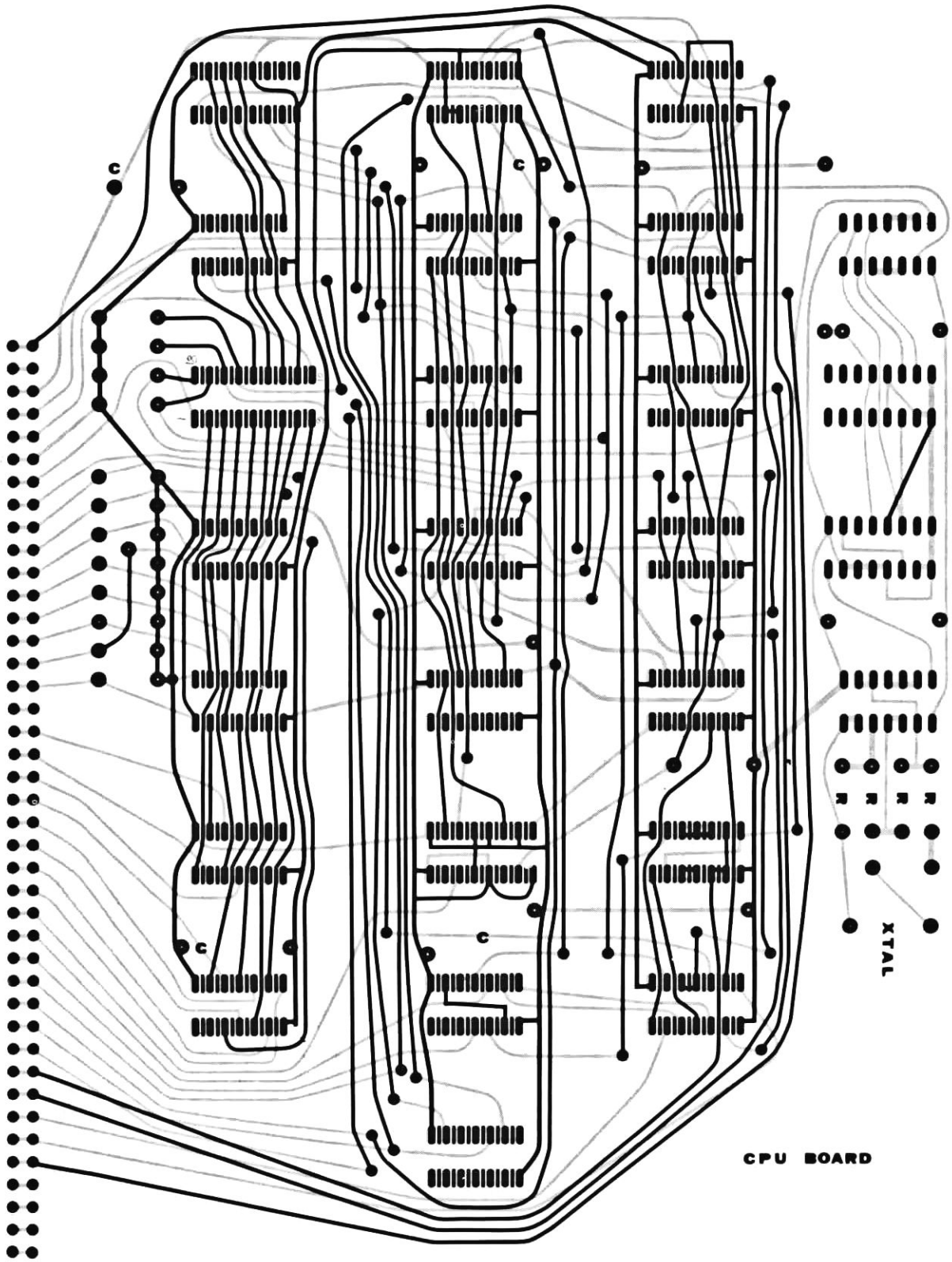
PRINTED CIRCUIT BOARD PATTERNS



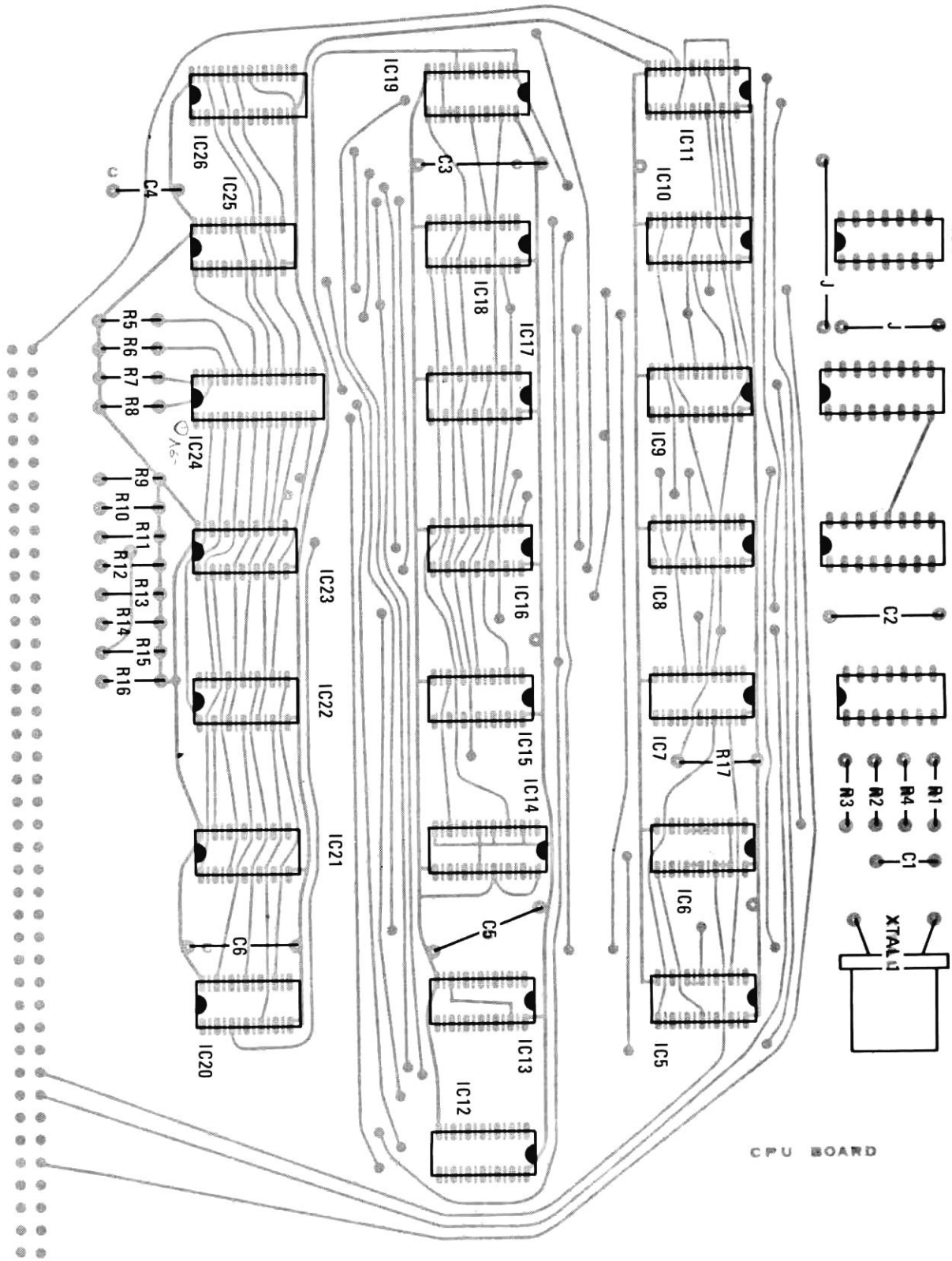
CPU BOARD – SIDE A



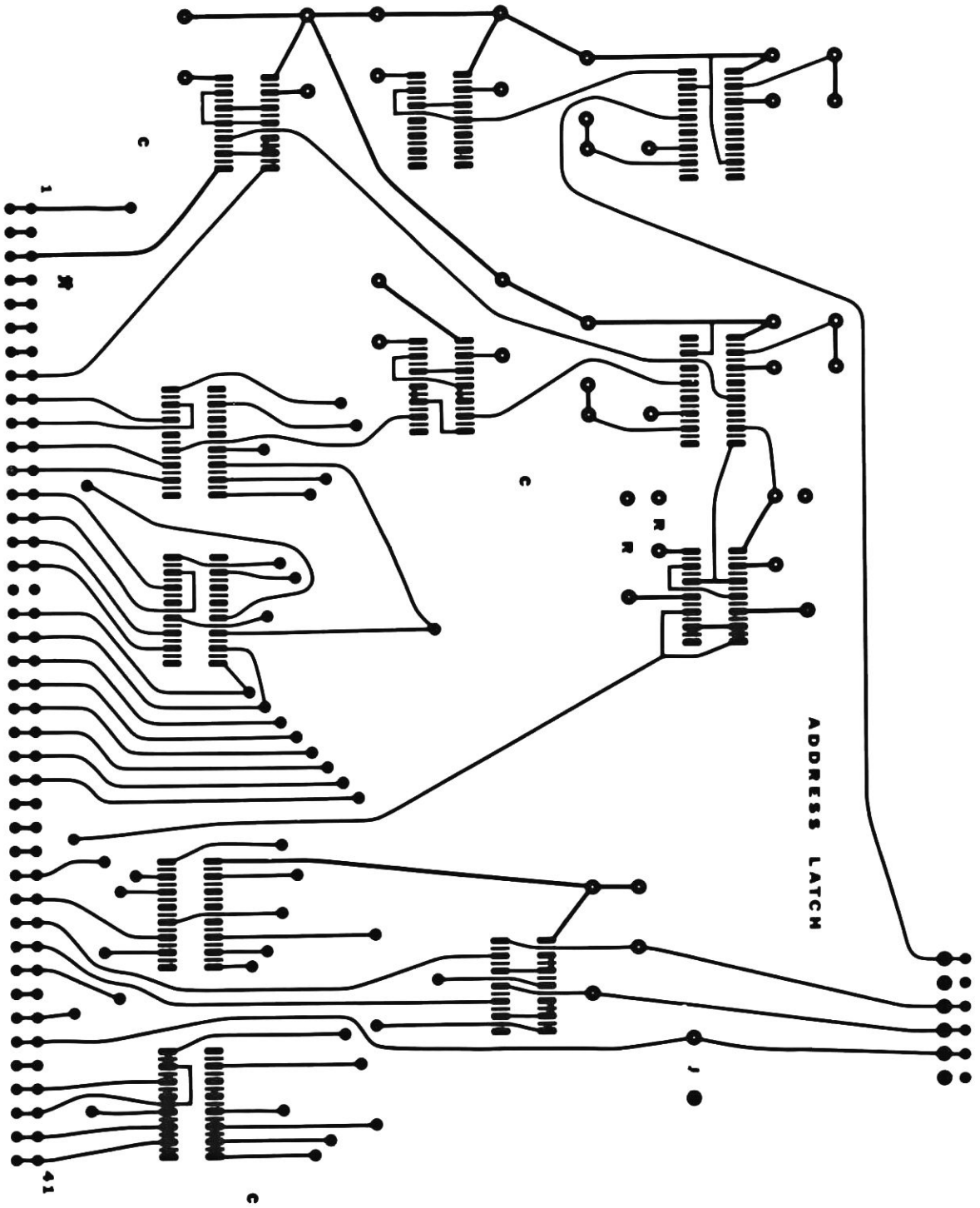
CPU BOARD – SIDE B



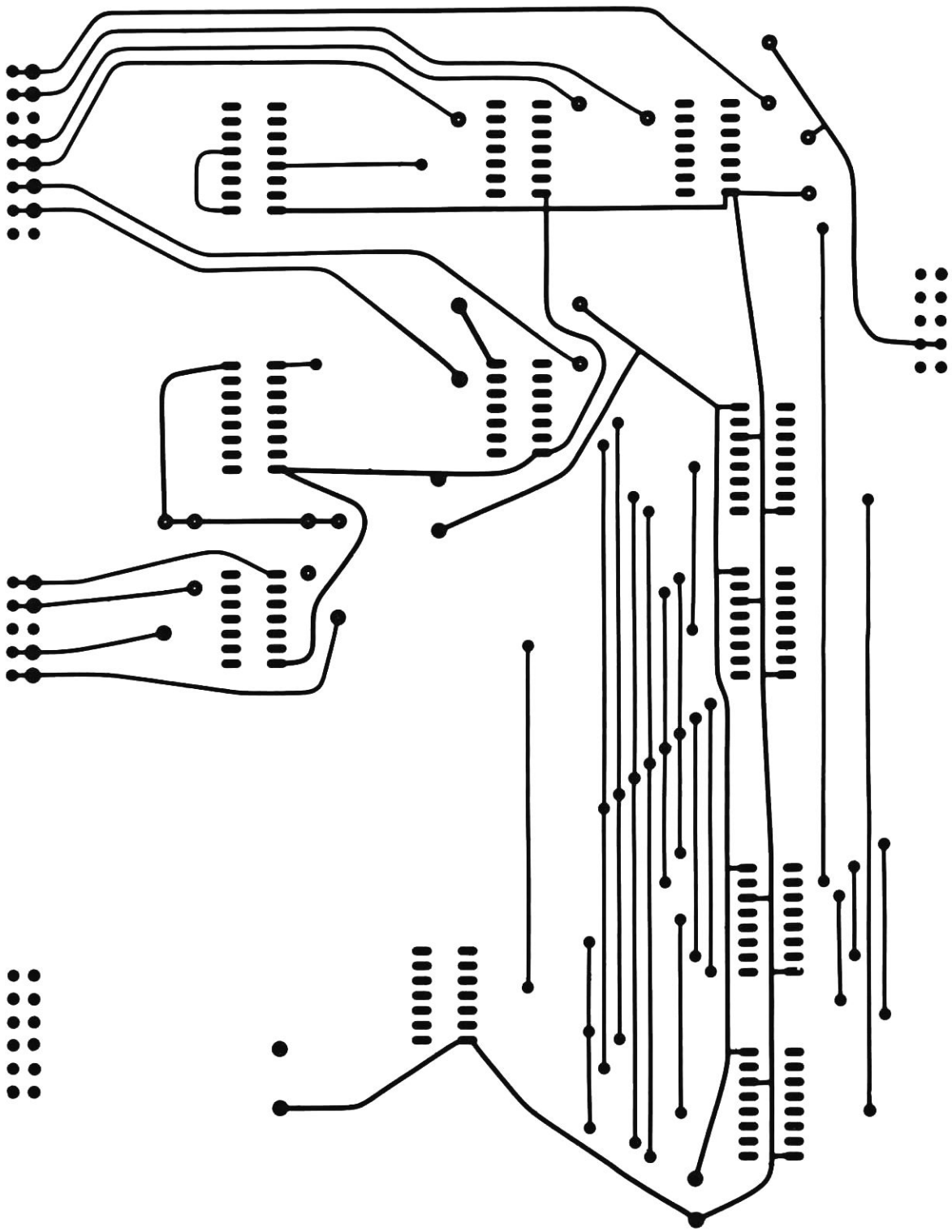
CPU BOARD — HOW FOIL PATTERNS OVERLAP



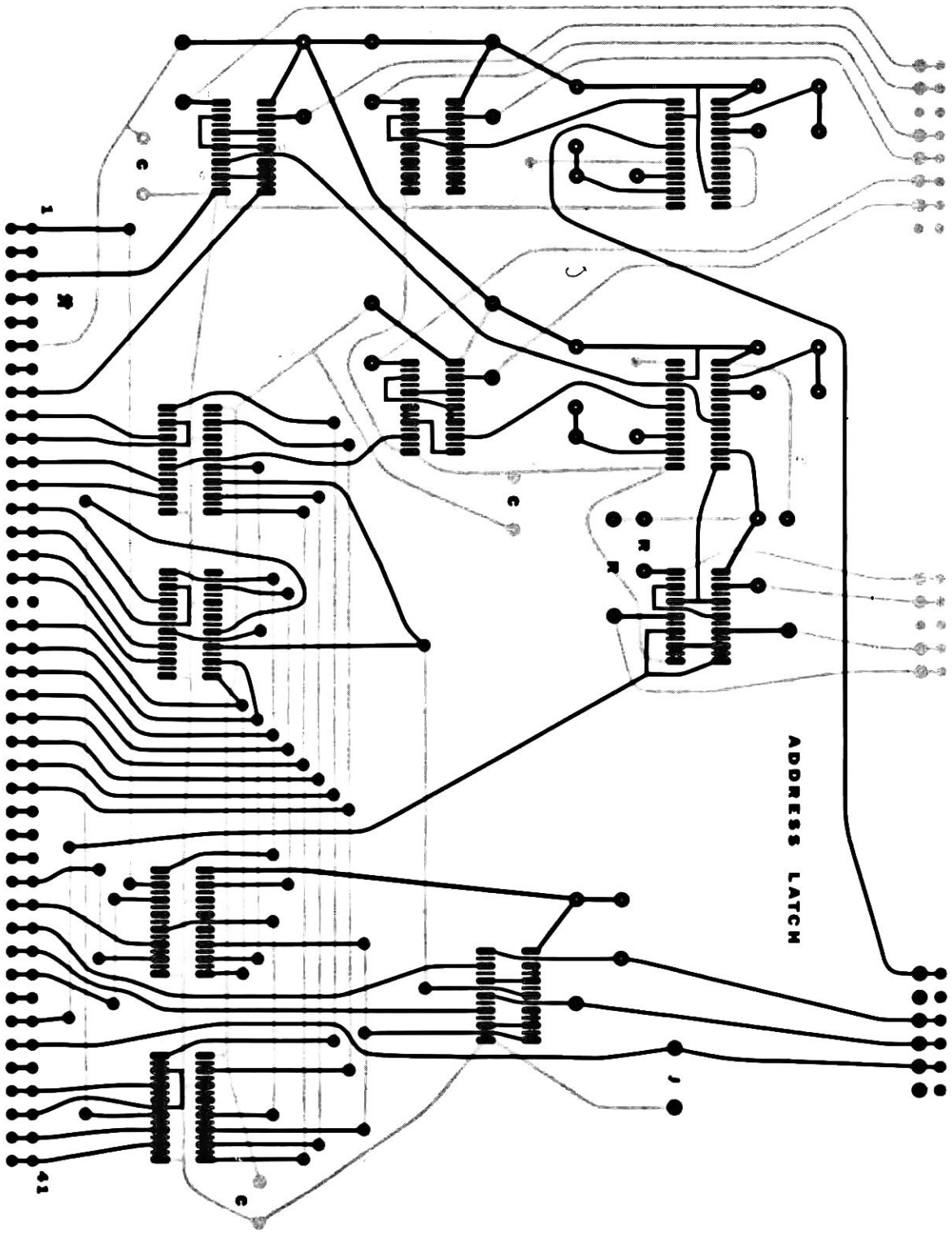
CPU BOARD – PARTS LAYOUT



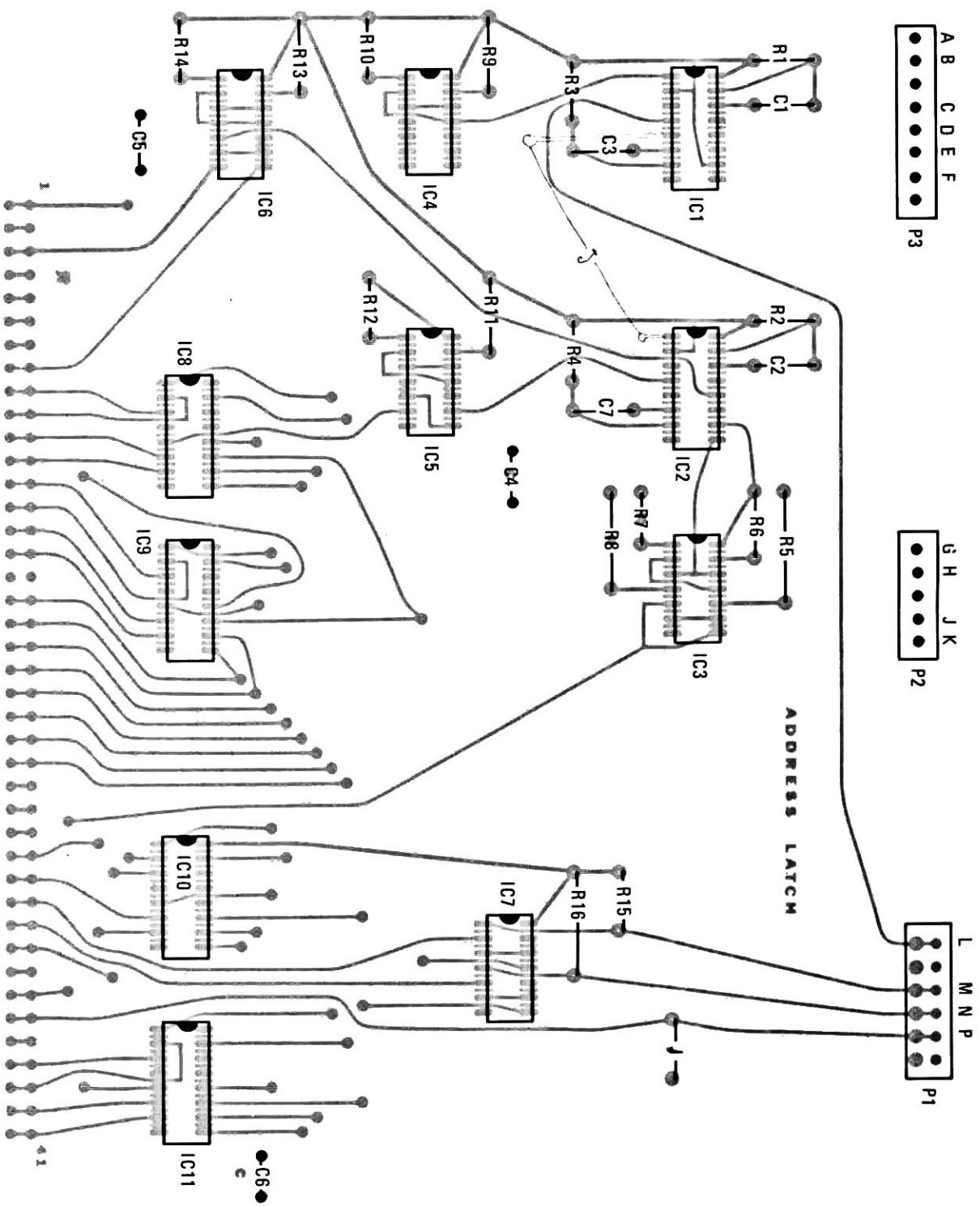
ADDRESS LATCH – SIDE A



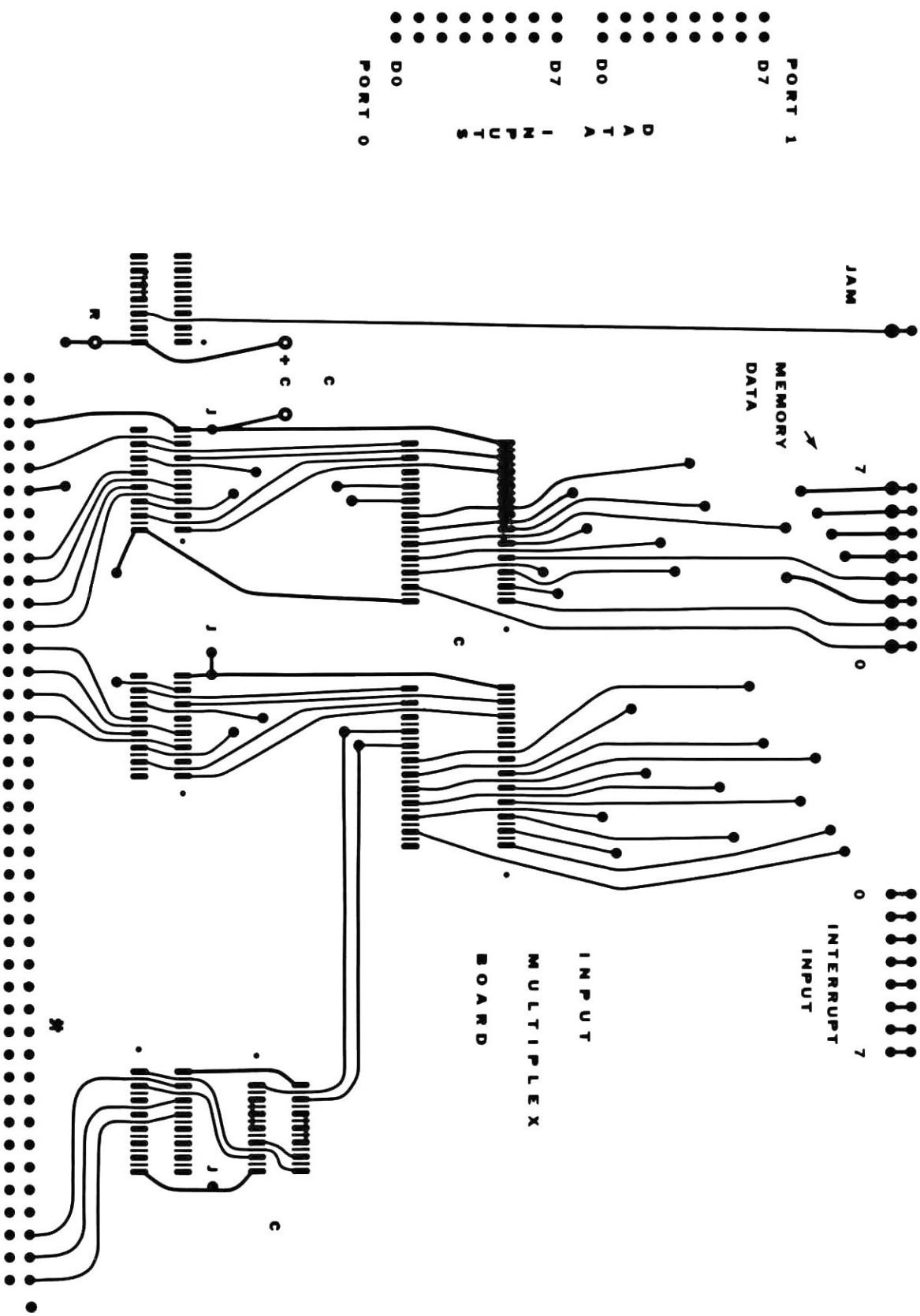
ADDRESS LATCH – SIDE B



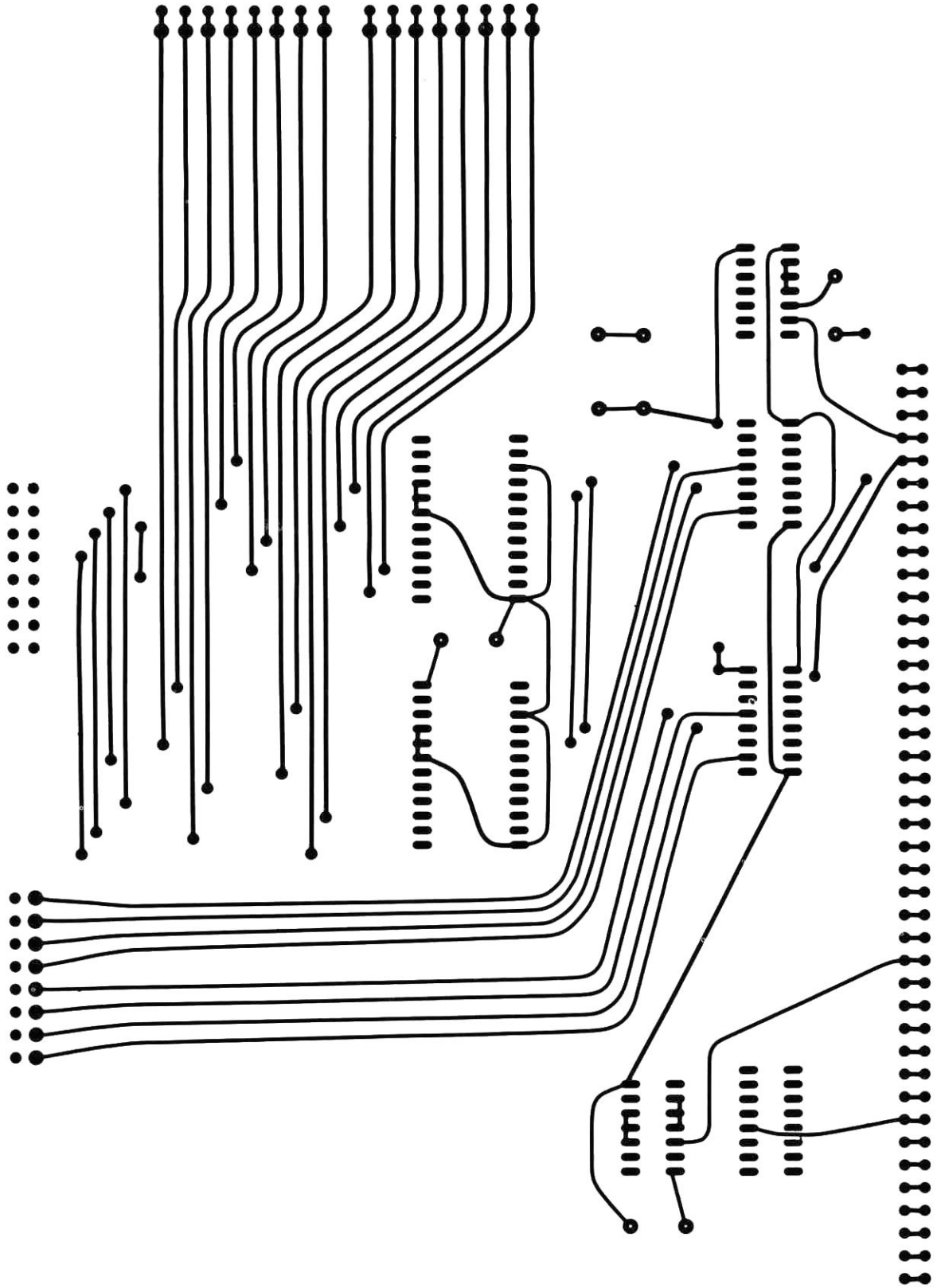
ADDRESS LATCH – HOW FOIL PATTERNS OVERLAP



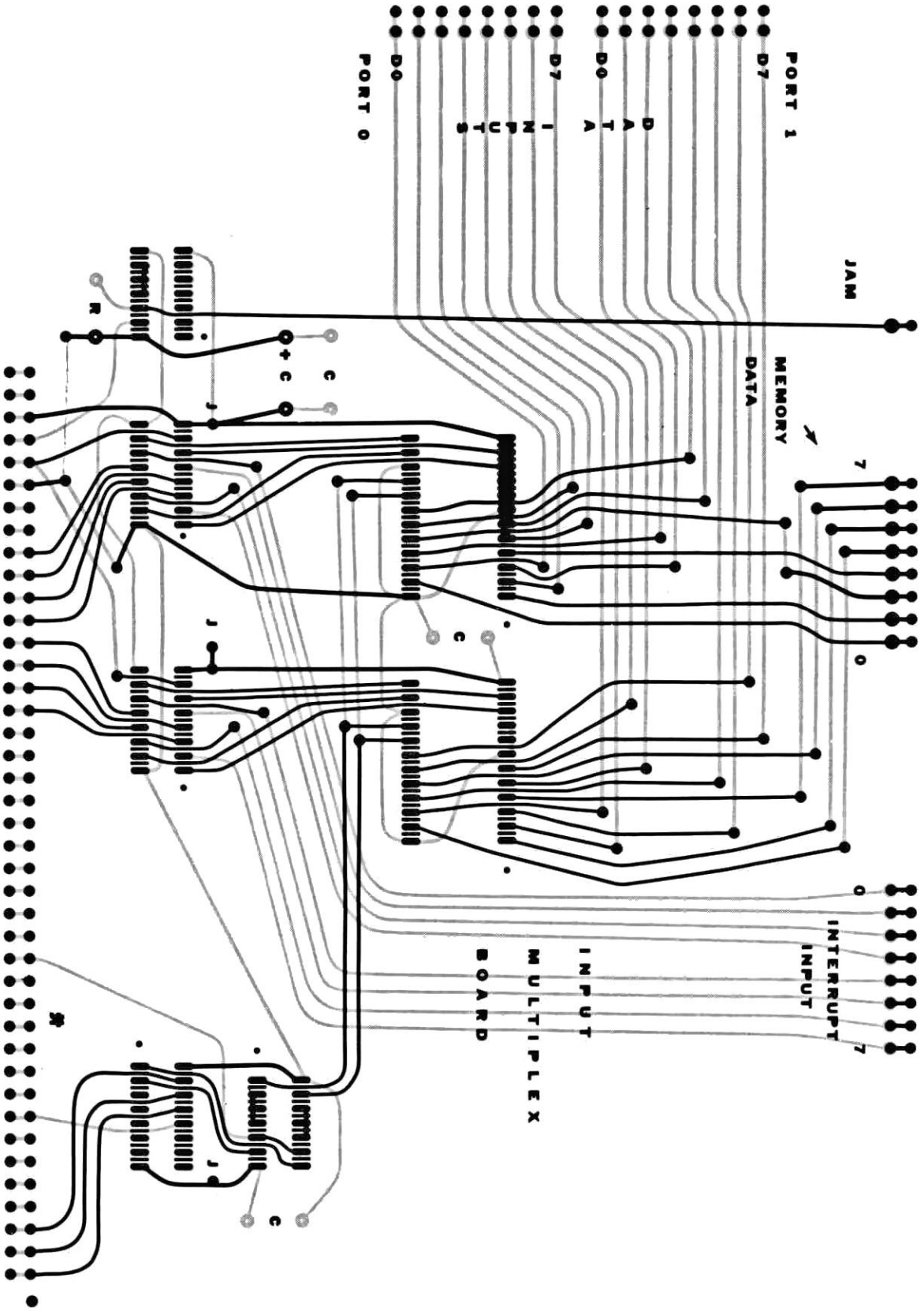
ADDRESS LATCH – PARTS LAYOUT



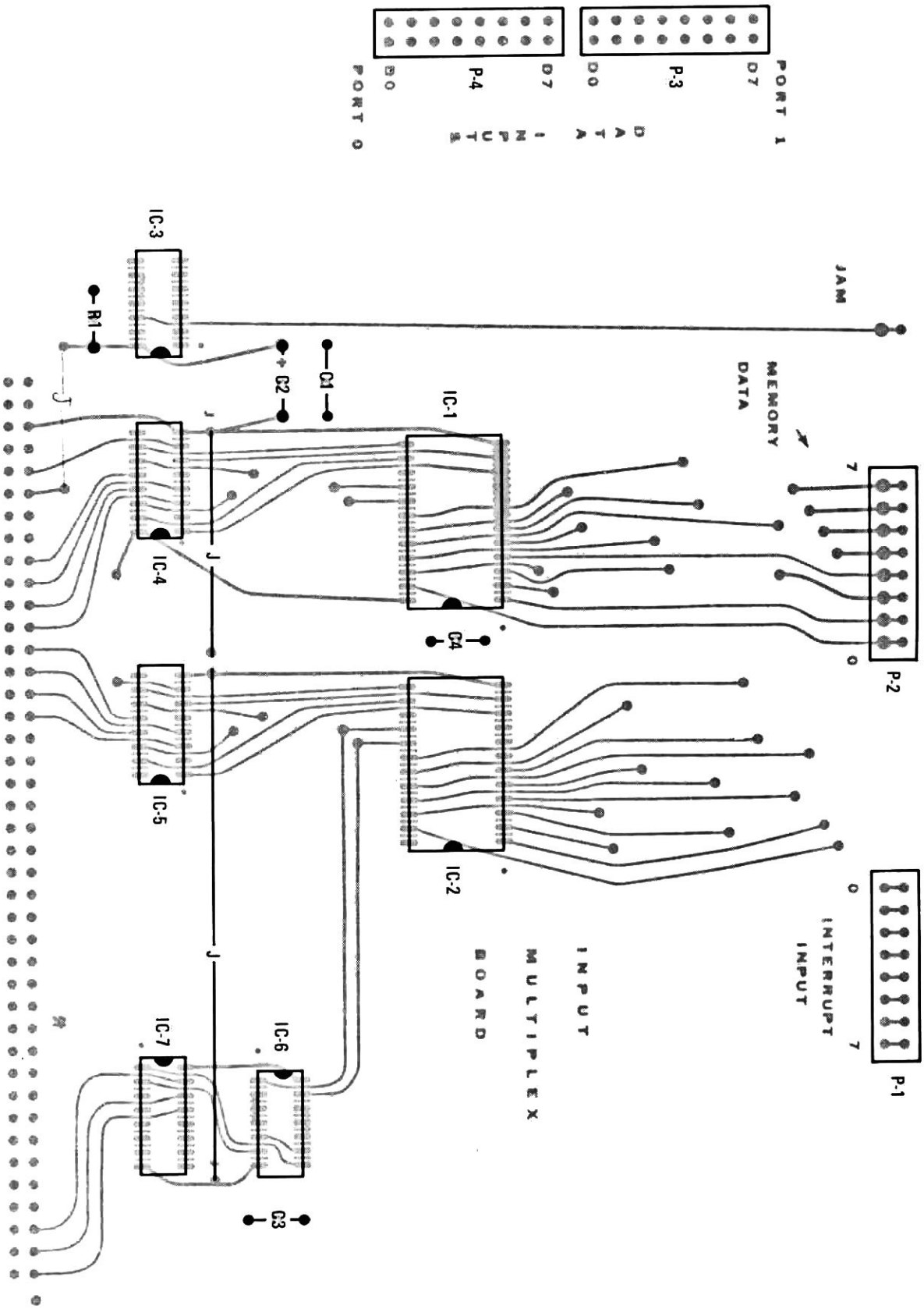
DATA INPUT MPX BOARD – SIDE A



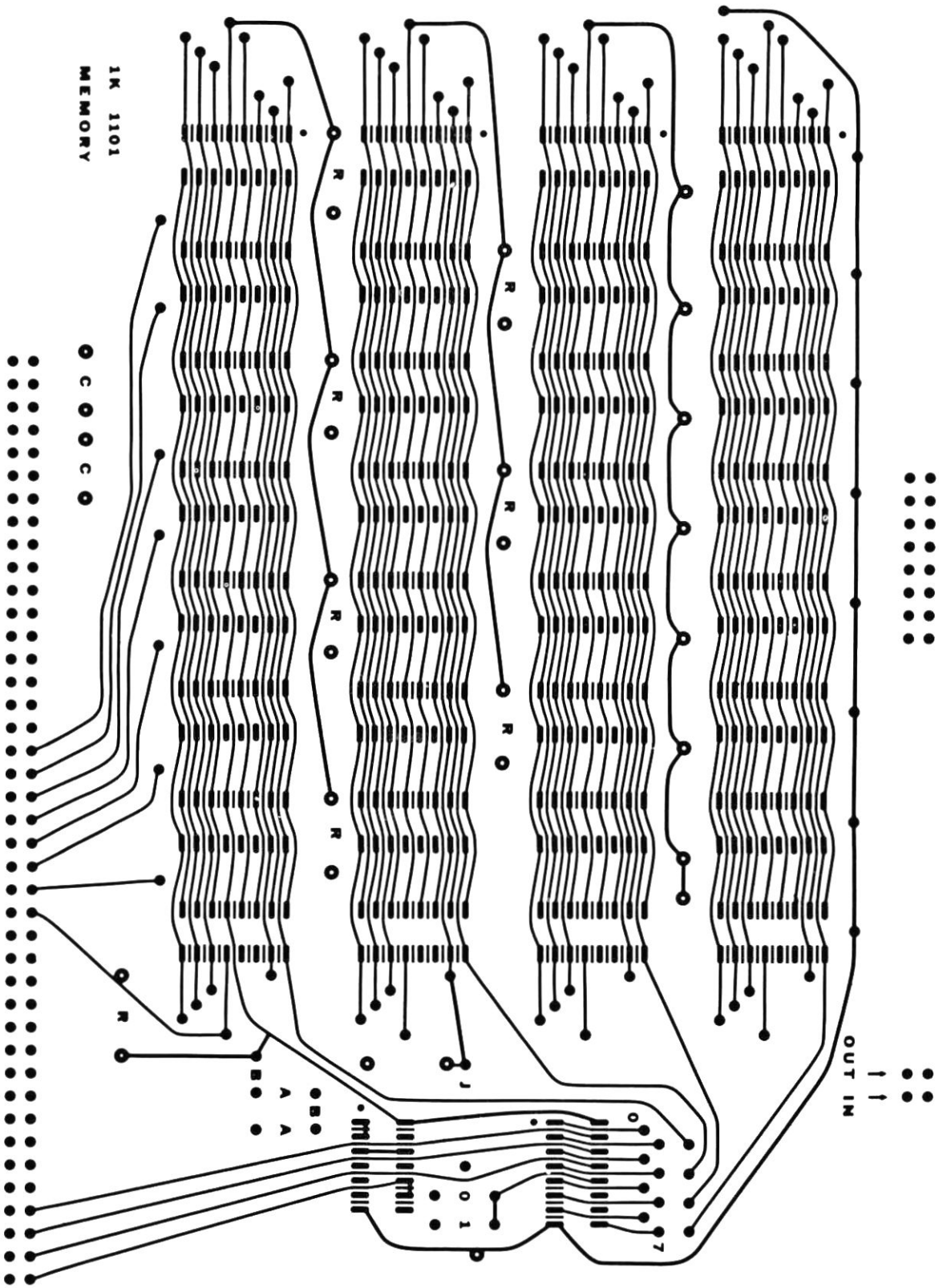
DATA INPUT MPX BOARD – SIDE B



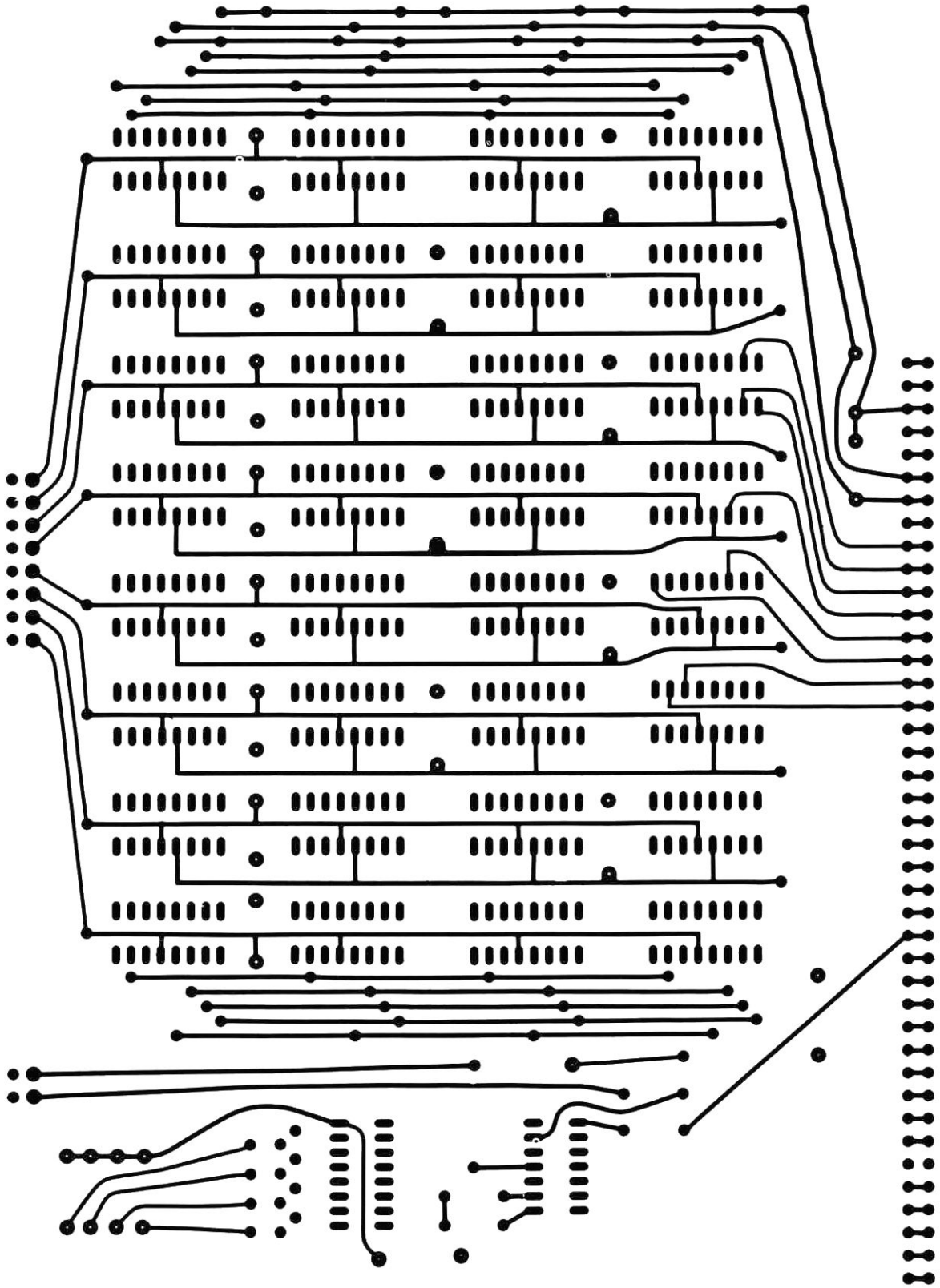
DATA INPUT MPX BOARD — HOW FOIL PATTERNS OVERLAP



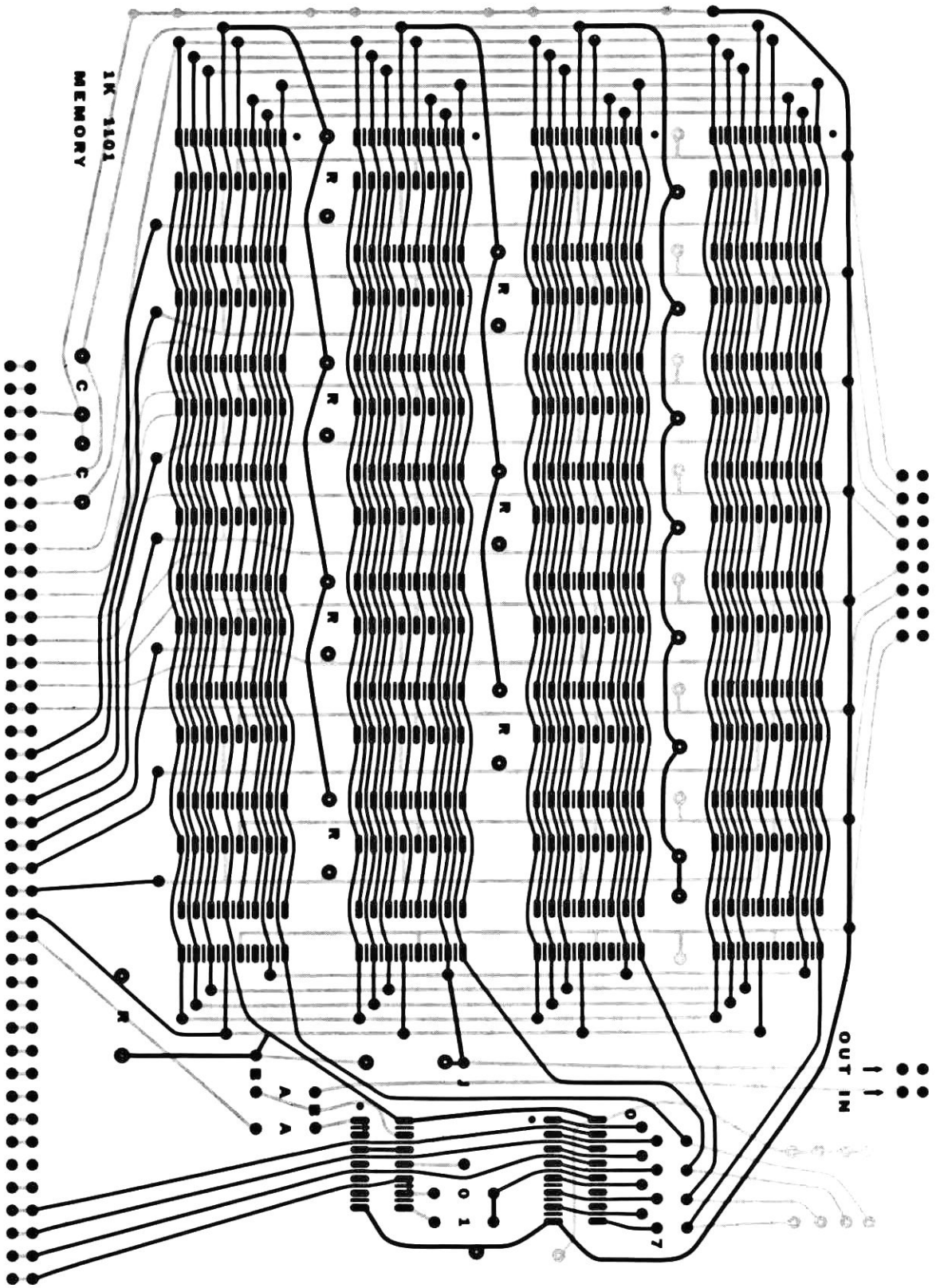
DATA INPUT MPX BOARD — PARTS LAYOUT



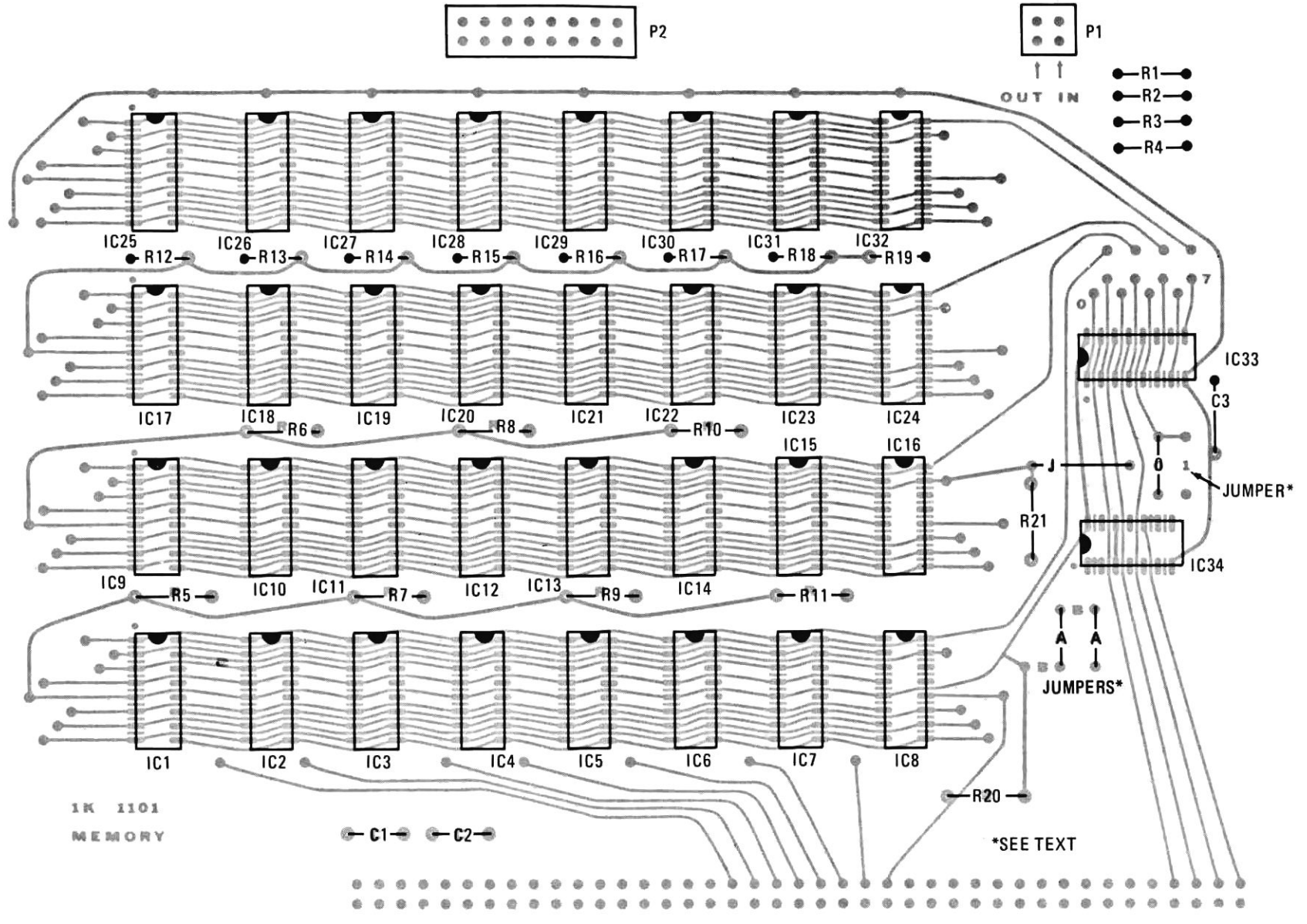
MEMORY – SIDE A



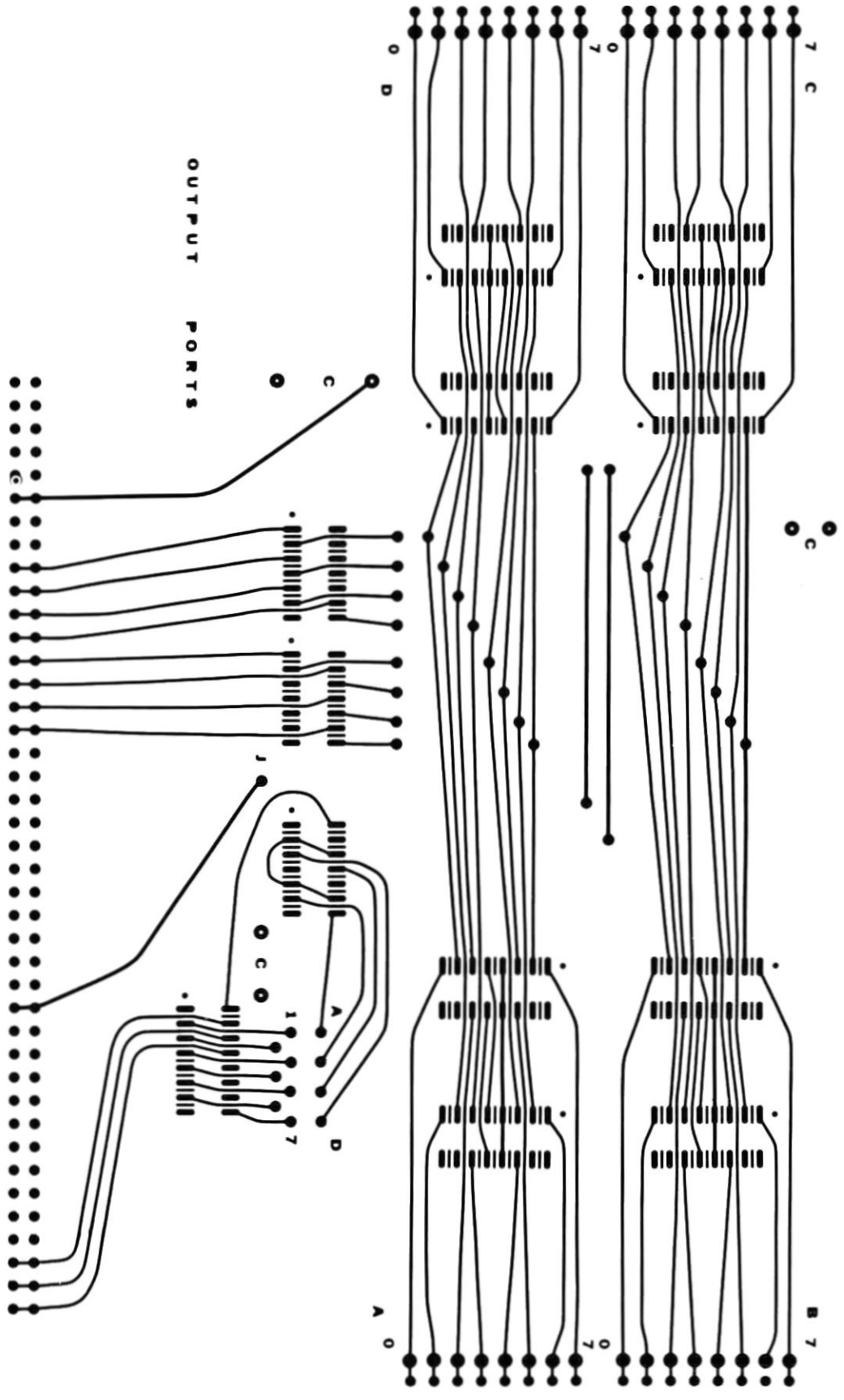
MEMORY – SIDE B



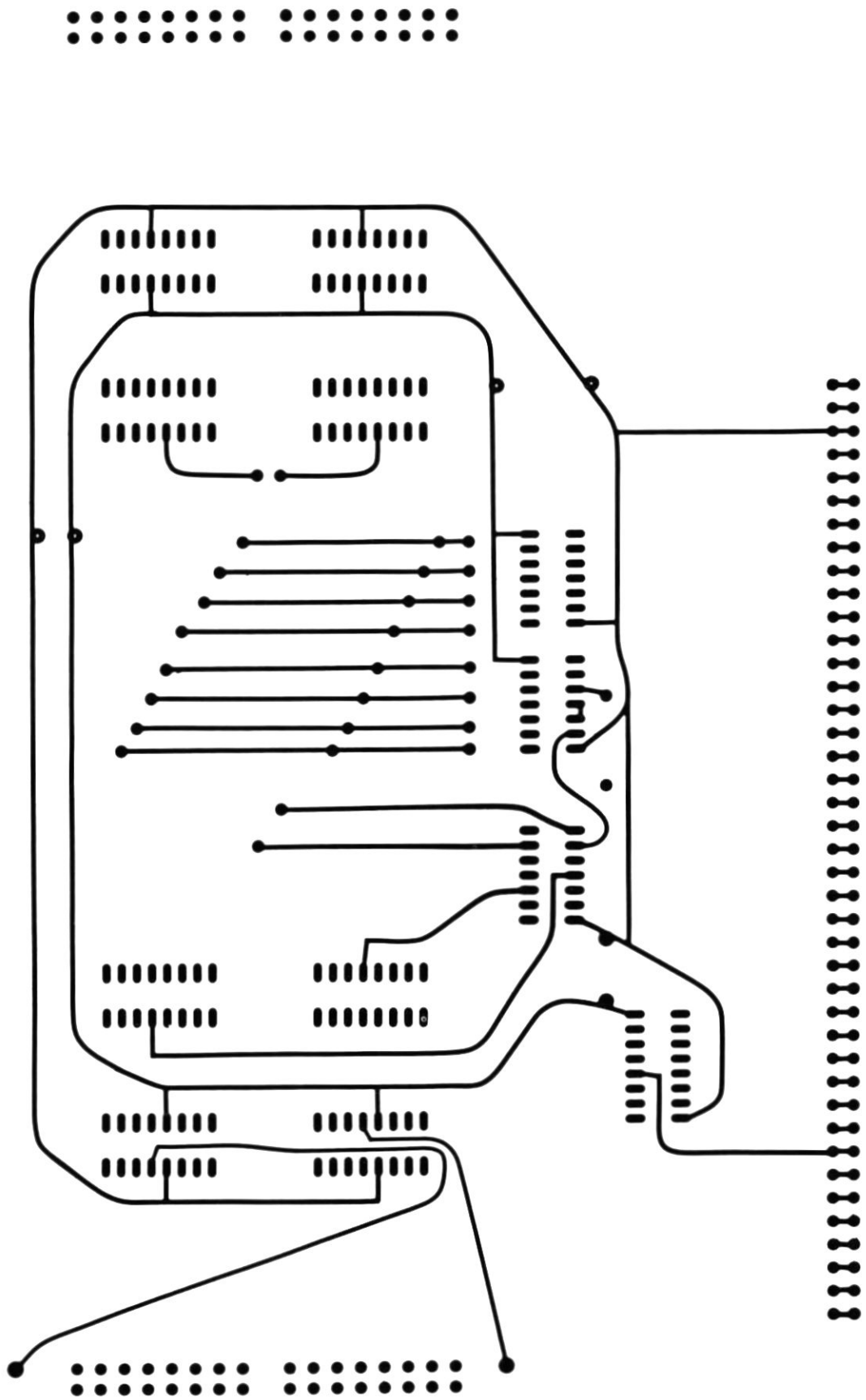
MEMORY – HOW FOIL PATTERNS OVERLAP



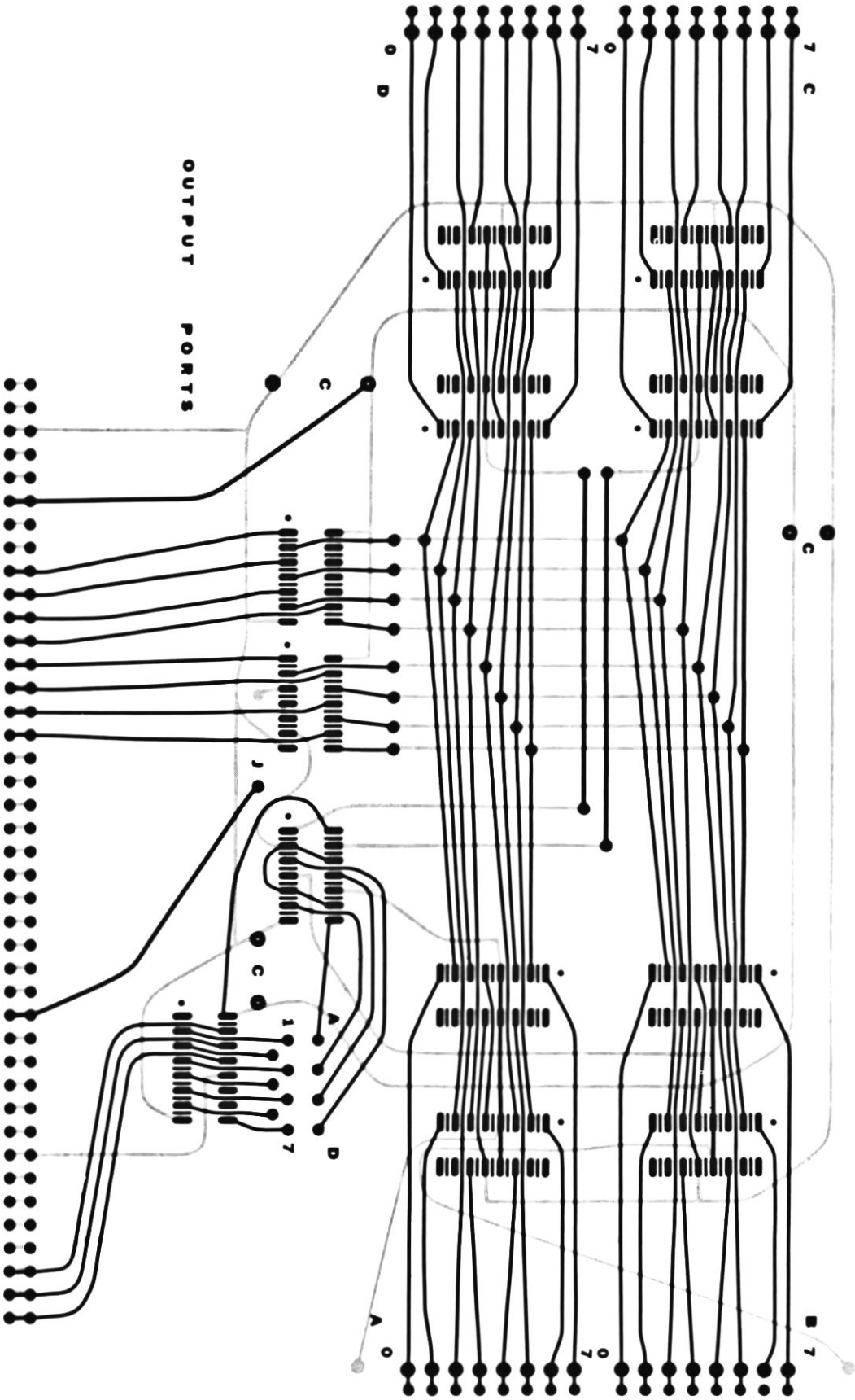
MEMORY – PARTS LAYOUT



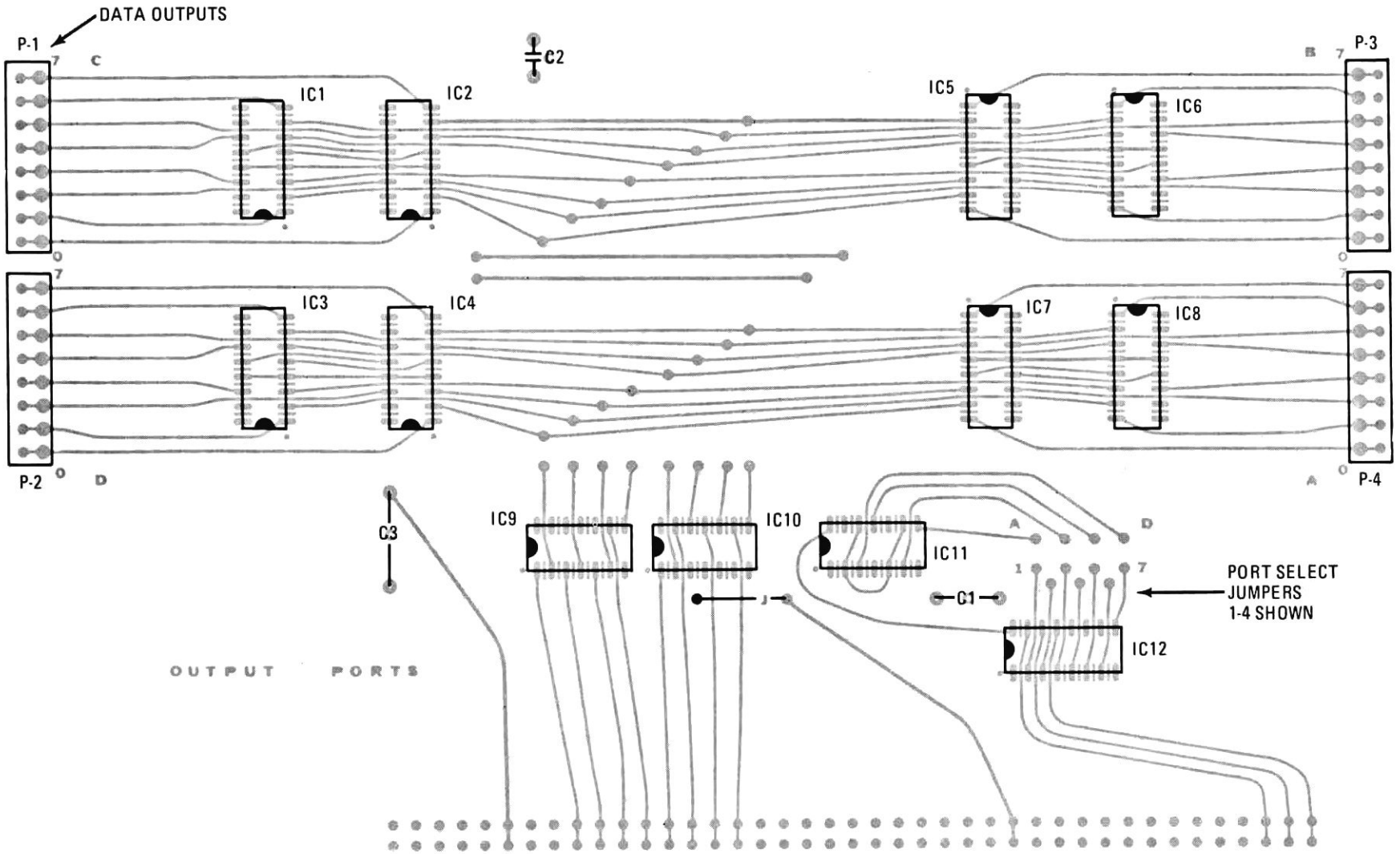
OUTPUT PORTS – SIDE A



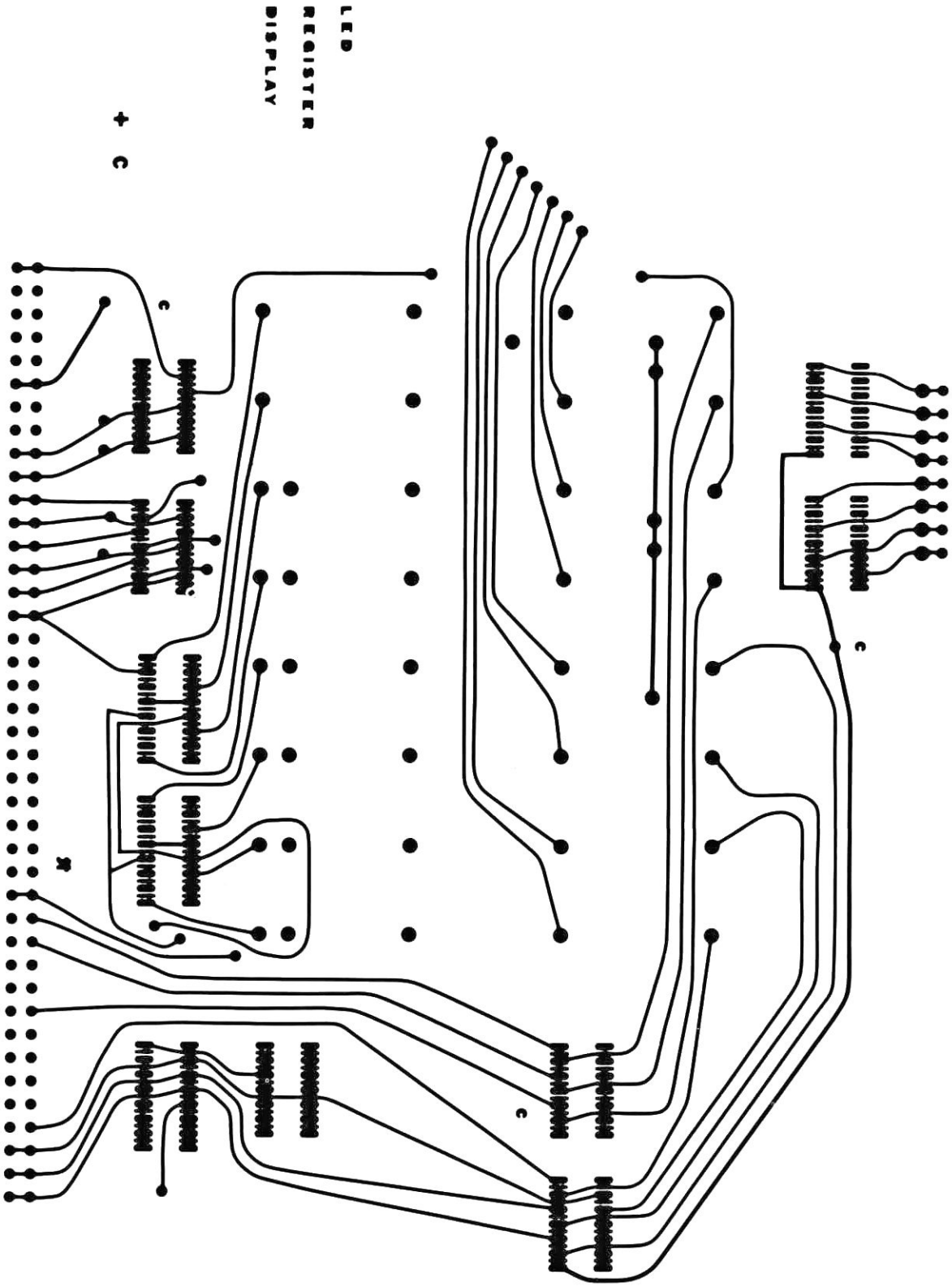
OUTPUT PORTS – SIDE B



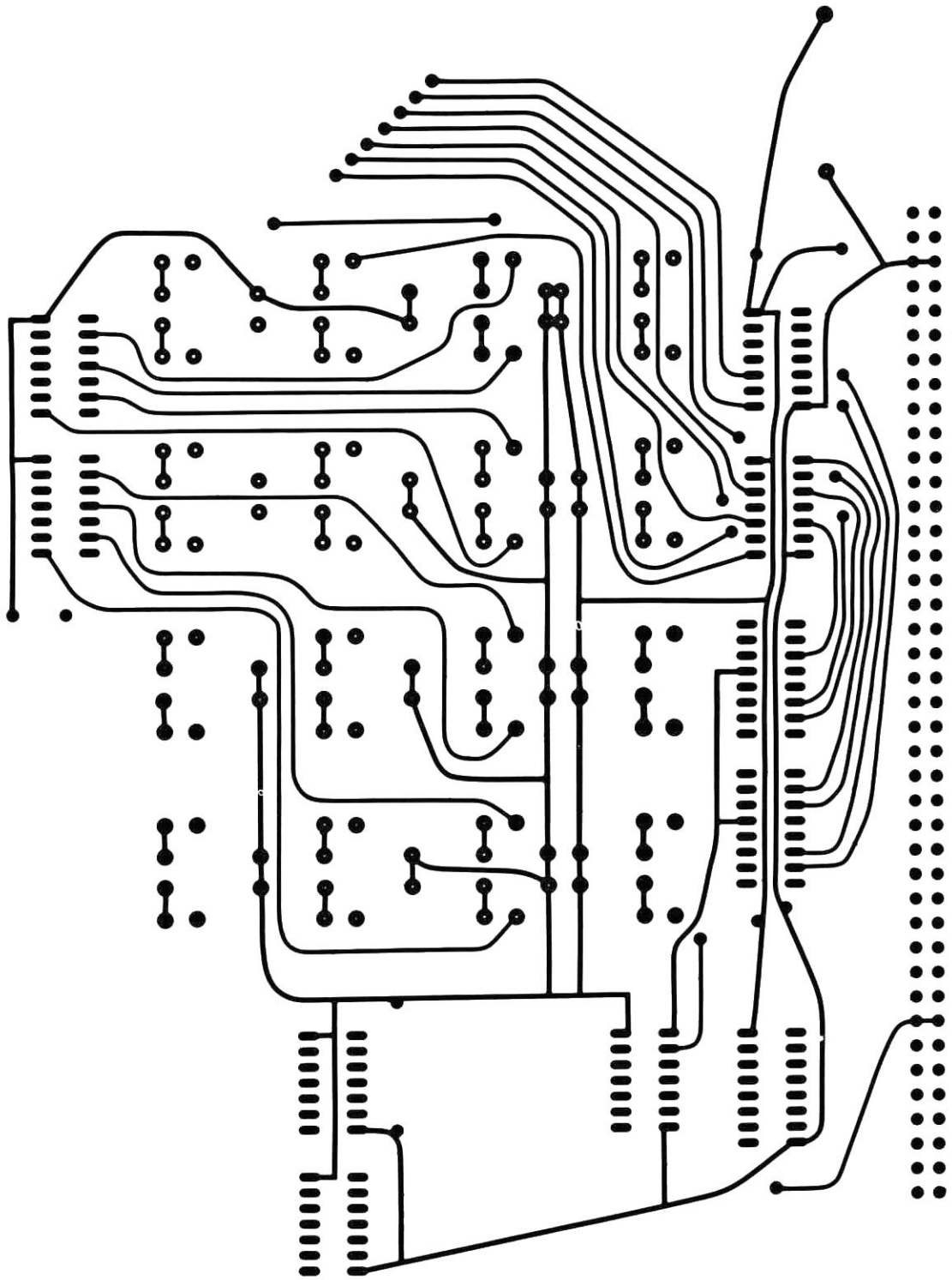
OUTPUT PORTS – HOW FOIL PATTERNS OVERLAP



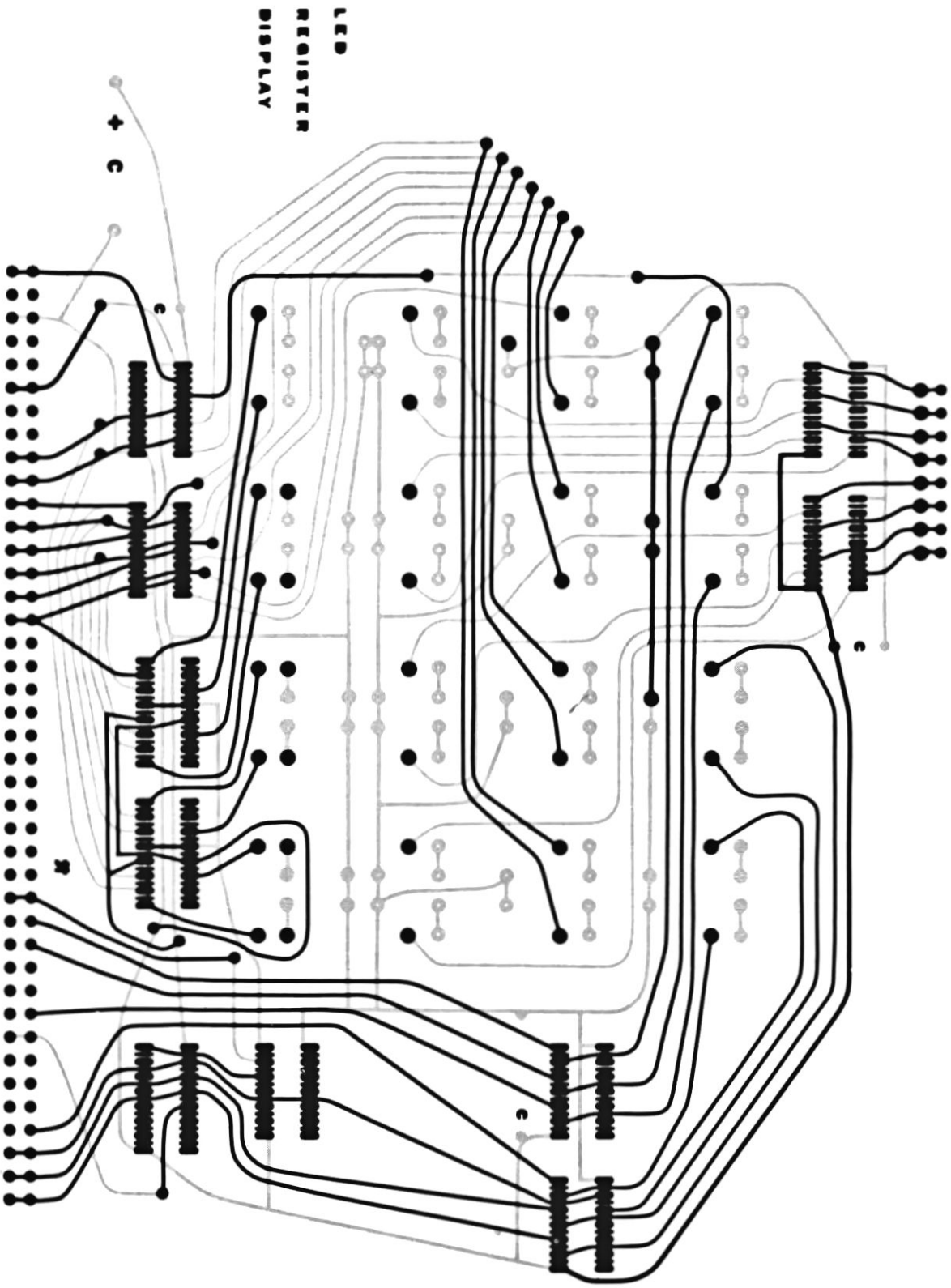
OUTPUT PORTS — PARTS LAYOUT



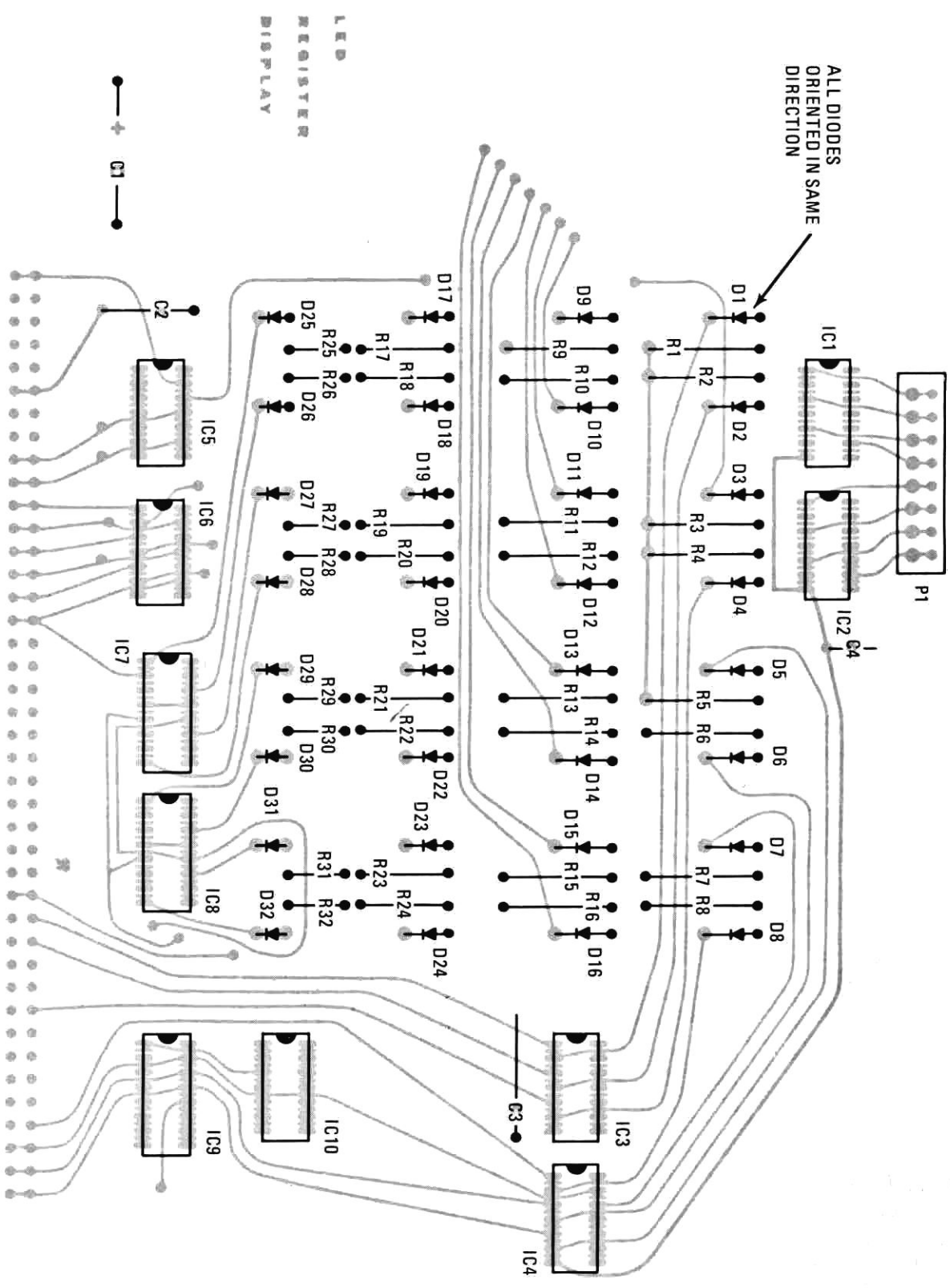
LED REGISTER DISPLAY – SIDE A



LED REGISTER DISPLAY – SIDE B



LED REGISTER DISPLAY – HOW FOIL PATTERNS OVERLAP



LED REGISTER DISPLAY — PARTS LAYOUT