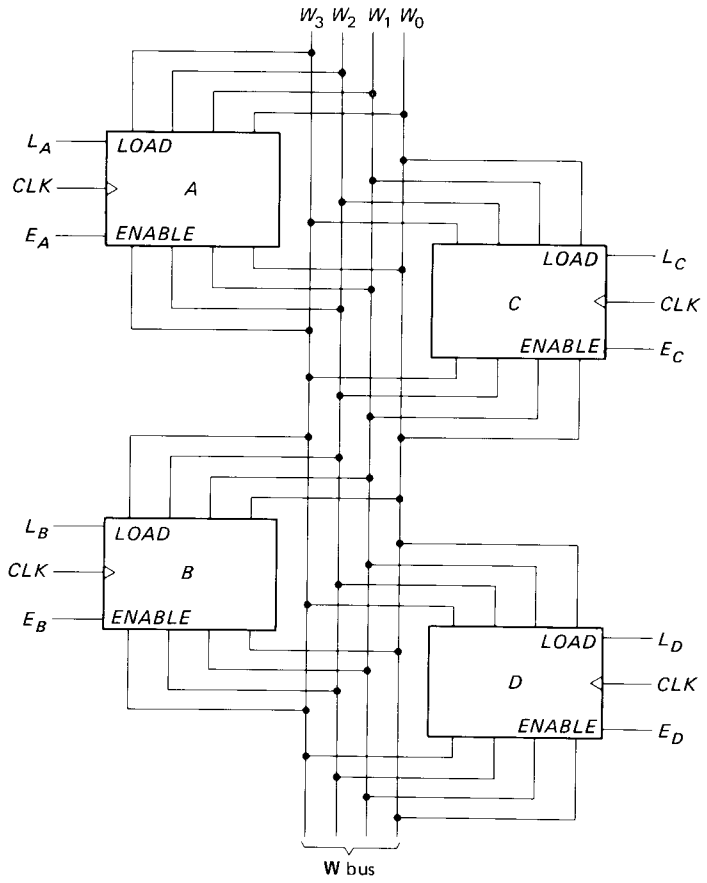


Fig. 8-34

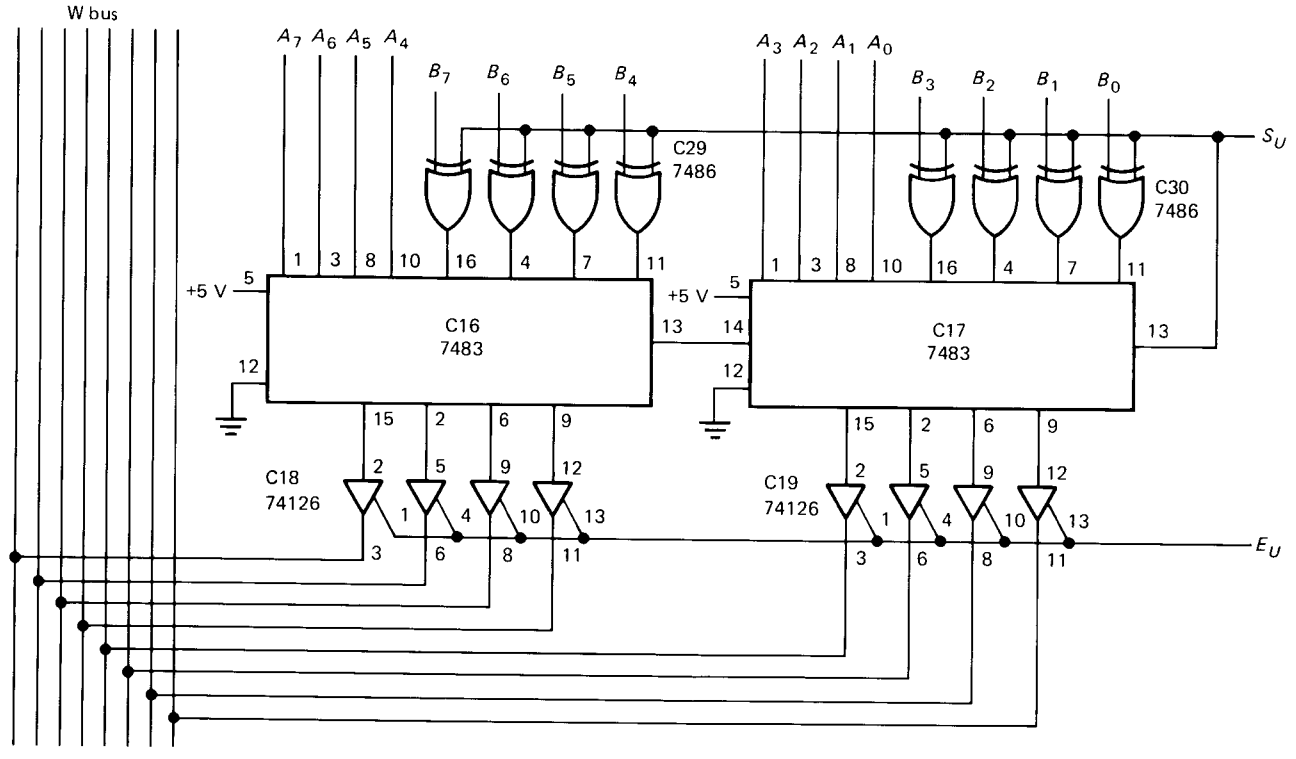
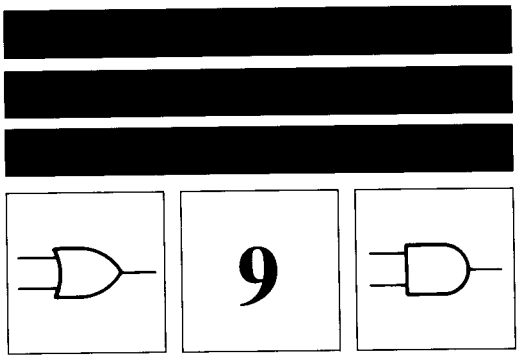


Fig. 8-35



MEMORIES

The *memory* of a computer is where the program and data are stored before the calculations begin. During a computer run, the control section may store partial answers in the memory, similar to the way we use paper to record our work. The memory is therefore one of the most active parts of a computer, storing not only the program and data but processed data as well.

The memory is equivalent to thousands of registers, each storing a binary word. The latest generation of computers relies on semiconductor memories because they are less expensive and easier to work with than core memories. A typical microcomputer has a semiconductor memory with up to 655,360 memory locations, each capable of storing 1 byte of information.

9-1 ROMS

A *read-only memory* (ROM) is the simplest kind of memory. It is equivalent to a group of registers, each permanently storing a word. By applying control signals, we can *read* the word in any memory location. ("Read" means to make the contents of the memory location appear at the output terminals of the ROM.)

Diode ROM

Figure 9-1 shows one way to build a ROM. Each horizontal row is a register or memory location. The R_0 register

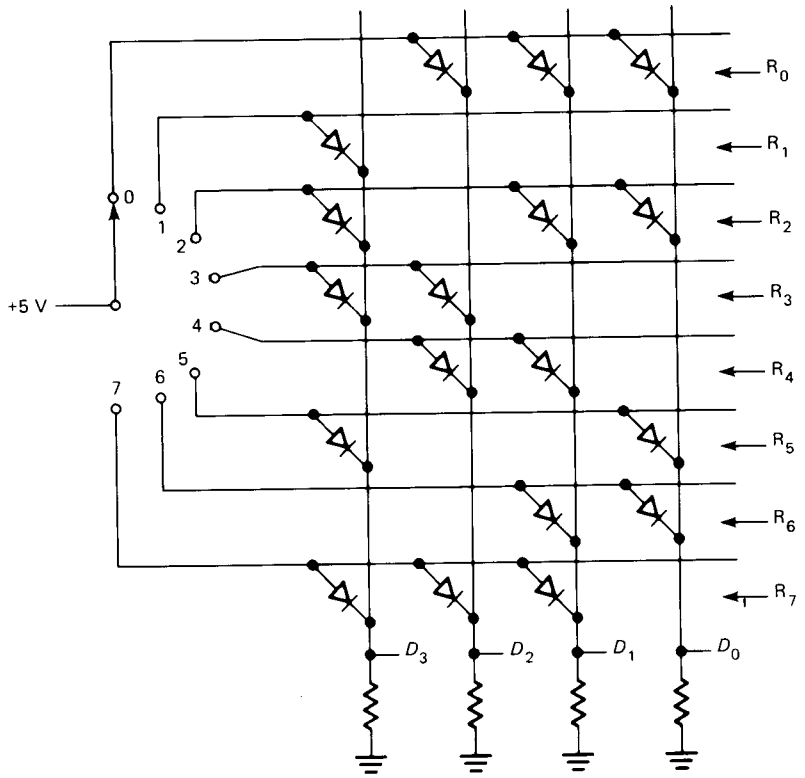


Fig. 9-1 Simple diode ROM.

TABLE 9-1. DIODE ROM

Register	Address	Word
R_0	0	0111
R_1	1	1000
R_2	2	1011
R_3	3	1100
R_4	4	0110
R_5	5	1001
R_6	6	0011
R_7	7	1110

contains three diodes, the R_1 register has one diode, and so on. The output of the ROM is the word

$$\mathbf{D} = D_3D_2D_1D_0$$

In switch position 0, a high voltage turns on the diodes in the R_0 register; all other diodes are off. This means that a high output appears at D_2, D_1, and D_0. Therefore, the word stored at memory location 0 is

$$\mathbf{D} = 0111$$

What happens if the switch is moved to position 1? The diode in the R_1 register conducts, forcing D_3 to go high. Because all other diodes are off, the output from the ROM becomes

$$\mathbf{D} = 1000$$

So the contents of memory location 1 are 1000.

As you move the switch to other positions, you will read the contents of the other memory locations. Table 9-1 shows these contents, which you can check by analyzing Fig. 9-1.

With discrete circuits we can change the contents of a memory location by adding or removing diodes. With integrated circuits, the manufacturer stores the words at the time of fabrication. In either case, the words are permanently stored once the diodes are wired in place.

Addresses

The *address* and *contents* of a memory location are two different things. As shown in Table 9-1, the address of a memory location is the same as the subscript of the register storing the word. This is why register 0 has an address of 0 and contents of 0111; register 1 has an address of 1 and contents of 1000; register 2 has an address of 2 and contents of 1011; and so on.

The idea of addresses applies to ROMs of any size. For example, a ROM with 256 memory locations has decimal addresses running from 0 to 255. A ROM with 1,024 memory locations has decimal addresses from 0 to 1,023.

On-Chip Decoding

Rather than switch-select the memory location, as shown in Fig. 9-1, IC manufacturers use *on-chip decoding*. Figure 9-2 gives you the idea. The three input pins (A_2, A_1, and A_0) supply the binary address of the stored word. Then a 1-of-8 decoder produces a high output to one of the registers.

For instance, if

$$\mathbf{ADDRESS} = A_2A_1A_0 = 100$$

the 1-of-8 decoder applies a high voltage to the R_4 register, and the ROM output is

$$\mathbf{D} = 0110$$

If you change the address word to

$$\mathbf{ADDRESS} = 110$$

you will read the contents of memory location 6, which is

$$\mathbf{D} = 0011$$

The circuit of Fig. 9-2 is a 32-bit ROM organized as 8 words of 4 bits each. It has three address (input) lines and four data (output) lines. This is a very small ROM compared with commercially available ROMs.

Number of Address Lines

With on-chip decoding, n address lines can select 2^n memory locations. For instance, we need 3 address lines in Fig.9-2 to access 8 memory locations. Similarly, 4 address lines can access 16 memory locations, 8 address lines can access 256 memory locations, and so on.

9-2 PROMS AND EPROMS

With a ROM, you have to send a list of data to be stored in the different memory locations to the manufacturer, who then produces a *mask* (a photographic template of the circuit) used in mass production of your ROMs. In fabricating ROMs the manufacturer may use bipolar transistors or MOSFETs. But the idea is still basically the same; the transistors or MOSFETs act like the diodes of Fig. 9-2.

Programmable

A *programmable* ROM (PROM) is different. It allows the user to store the data. An instrument called a *PROM programmer* does the storing by "burning in." (Fusible links at the bit locations can be burned open by high currents.) With a PROM programmer, the user can burn in the program and data. Once this has been done, the programming is permanent. In other words, the stored contents cannot be erased.

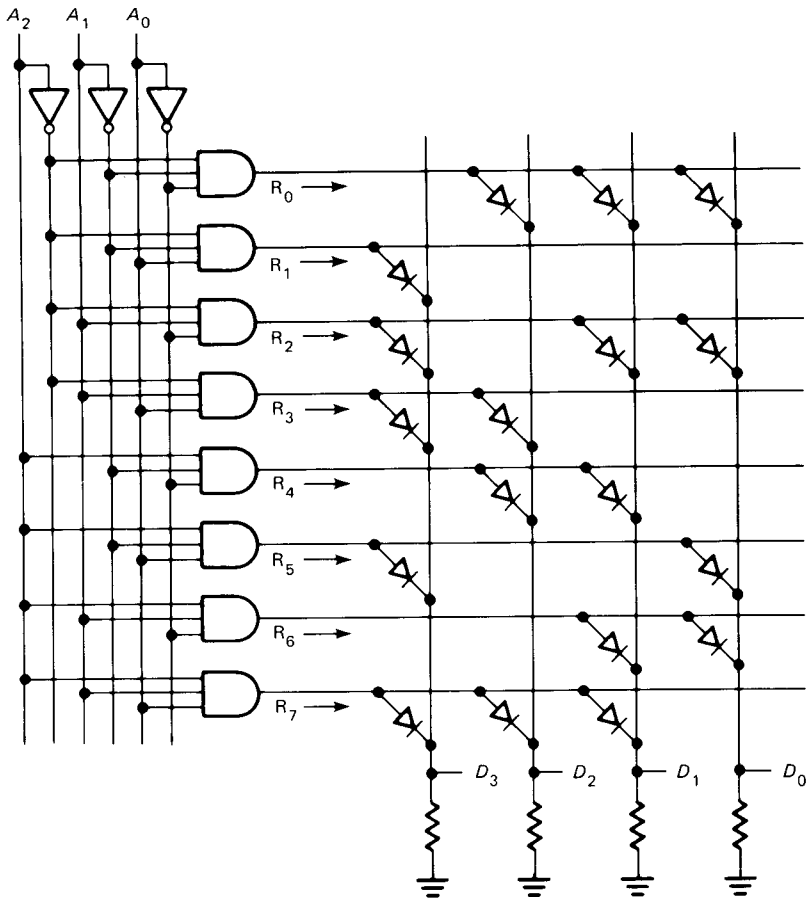


Fig. 9-2 ROM with on-chip decoding.

Erasable

The *erasable PROM* (EPROM) uses MOSFETs. Data is stored with a PROM programmer. Later, data can be erased with ultraviolet light. The light passes through a window in the IC package to the chip, where it releases stored charges. The effect is to wipe out the stored contents. In other words, the EPROM is ultraviolet-light-erasable and electrically reprogrammable.

The EPROM is helpful in design and development. The user can erase and store until the program and data are perfected. Then the program and data can be sent to an IC manufacturer who makes a ROM mask for mass production.

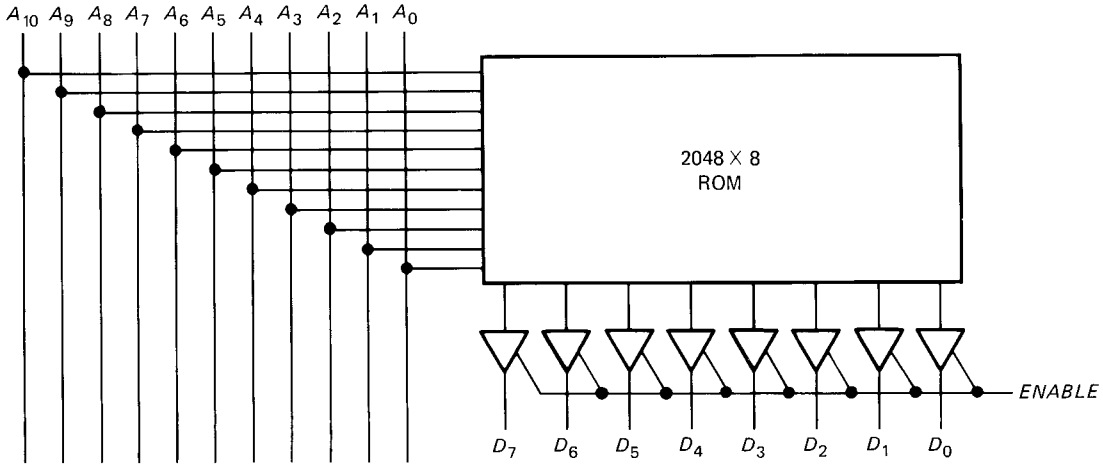


Fig. 9-3. Three-state ROM.

EEPROM

Another type of reprogrammable ROM device is the EEPROM (Electrically Erasable Programmable Read Only Memory), which is nonvolatile like EPROM but does not require ultraviolet light to be erased. It can be completely erased or have certain bytes changed, using electrical pulses. Individual bytes (or any number of bytes) can be changed using a programmer designed for use with EEPROMs. Individual bytes can also be changed by the host circuit after the EEPROM has been installed.

EEPROM is useful when data being gathered by the circuit must be stored by the system. Writing to EEPROM is slower than writing to RAM, so it cannot be used in high-speed circuits.

Unlimited READ cycles are possible; however, EEPROM will eventually wear out from repeated ERASE cycles. Since the life of typical EEPROMS allows thousands of erase cycles, this is usually not a problem.

There are matching EEPROM replacements for most EPROMs. The EEPROM uses an 8 digit in the part number whereas EPROM uses a 7 digit. For example, the 2816 EEPROM can replace the 2716 EPROM.

Manufactured Devices

With large-scale integration, manufacturers can fabricate ROMs, PROMs, and EPROMs that store thousands of words. For instance, the 8355 is a 16,384-bit ROM organized as 2,048 words of 8 bits each. It has 11 address lines and 8 data lines.

As another example, the 2764 is 65,536-bit EPROM organized as 8,192 words of 8 bits each. It has 13 address lines and 8 data lines.

Access Time

The *access time* of a memory is the time it takes to read a stored word after applying address bits. Since bipolar transistors are faster than MOSFETs, bipolar memories have faster access times than MOS memories. For instance, the 3636 is a bipolar PROM with an access time of 80 ns; the 2716 is a MOS EPROM with an access time of 450 ns. You have to pay for the speed; a bipolar memory is more expensive than a MOS memory, so it's up to the designer to decide which type to use in a specific application.

Three-State Memories

By adding three-state switches to the data lines of a memory we can get a three-state output. As an example, Fig. 9-3 shows a 16,384-bit ROM organized as 2,048 words of 8 bits each. It has 11 address lines and 8 data lines. A low *ENABLE* opens all switches and floats the output lines. On the other hand, a high *ENABLE* allows the addressed word to reach the final output.

Most of the commercially available ROMs, PROMs, and EPROMs have three-state outputs. In other words, they have built-in three-state switches that allow you to connect or disconnect the output lines from a data bus. More will be said about this later.

Nonvolatile Memory

ROMs, PROMs, and EPROMs are *nonvolatile memories.* This means that they retain the stored data even when the power to the device is shut off. Not all memories are like this, as will be explained in Sec. 9-3.

EXAMPLE 9-1

A 16 × 8 ROM stores these words in its first four locations:

$\mathbf{R}_0$ = 1110 0010 $\quad$ $\mathbf{R}_2$ = 0011 1100
$\mathbf{R}_1$ = 0101 0111 $\quad$ $\mathbf{R}_3$ = 1011 1111

Express the stored contents in hexadecimal notation.

SOLUTION

In hexadecimal shorthand, the stored contents are

$\mathbf{R}_0$ = E2H $\quad$ $\mathbf{R}_2$ = 3CH
$\mathbf{R}_1$ = 57H $\quad$ $\mathbf{R}_3$ = BFH

9-3 RAMS

A *random-access memory* (RAM), or a *read-write* memory, is equal to a group of addressable registers. After supplying an address, you can read the stored contents of the memory location or write new contents into the memory location.

Core RAMs

The core RAM was the workhorse of earlier computers. It has the advantage of being nonvolatile; even though you shut off the power, a core RAM continues to store data. The disadvantage of core RAMs is that they are expensive and harder to work with than semiconductor memories.

Semiconductor RAMs

Semiconductor RAMs may be *static* or *dynamic*. The static RAM uses bipolar or MOS flip-flops; data is retained indefinitely as long as power is applied to the flip-flops. On the other hand, a dynamic RAM uses MOSFETs and capacitors that store data. Because the capacitor charge leaks off, the stored data must be *refreshed* (recharged) every few milliseconds. In either case, the RAMs are volatile; turn off the power and you lose the stored data.

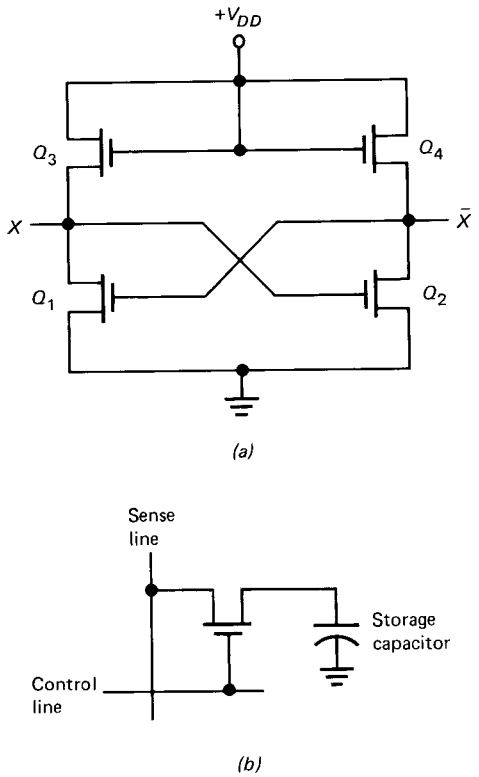


Fig. 9-4 (*a*) Static cell; (*b*) dynamic cell.

Static RAM

Figure 9-4*a* shows one of the flip-flops used in a static MOS RAM. Q_1 and Q_2 act like switches. Q_3 and Q_4 are active loads, meaning that they behave like resistors. The circuit action is similar to the transistor latch discussed in Sec. 7-1. Either Q_1 conducts and Q_2 is cut off or vice versa. A static RAM will contain thousands of flip-flops like this, one for each stored bit. As long as power is applied, the flip-flop remains latched and can store the bit indefinitely.

Dynamic RAM

Figure 9-4*b* shows one of the memory elements (called *cells*) in a dynamic RAM. When the *sense* and *control* lines go high, the MOSFET conducts and charges the capacitor. When the sense and control lines go low, the MOSFET opens and the capacitor retains its charge. In this way, it can store 1 bit. A dynamic RAM may contain thousands of memory cells like Fig. 9-4*b*. Since only a single MOSFET and capacitor are needed, the dynamic RAM contains more memory cells than a comparable static RAM. In other words, a dynamic RAM has more memory locations than a static RAM of the same physical size.

The disadvantage of the dynamic RAM is the need to refresh the capacitor charge every few milliseconds. This complicates the design problem because more circuitry is needed. In short, it's much simpler to work with static RAMs than dynamic RAMs. The remainder of this book emphasizes static RAMs.

Three-State RAMs

Many of the commercially available RAMs, either static or dynamic, have three-state outputs. In other words, the manufacturer includes three-state switches on the chip so that you can connect or disconnect the output lines of the RAM from a data bus.

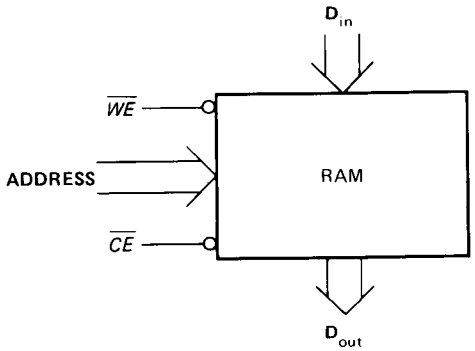


Fig. 9-5 Static RAM with inverted control inputs.

Figure 9-5 shows a static RAM and typical input signals. The **ADDRESS** bits select the memory location; control signals $\overline{WE}$ and $\overline{CE}$ select a write, read, or do nothing operation. $\overline{WE}$ is known as the *write-enable signal,* and $\overline{CE}$ is called the *chip-enable signal.* Notice that the control inputs are active low.

Table 9-2 summarizes the operation of the static RAM. Here's what happens. A low $\overline{CE}$ and low $\overline{WE}$ produce a write operation. This means that the input data $\mathbf{D}_{in}$ is stored in the addressed memory location. The three-state output data lines are floating during this write operation.

When $\overline{CE}$ is low and $\overline{WE}$ is high, we get a read operation. The contents of the addressed memory location appear on the data output lines because the internal three-state switches are closed at this time.

The final possibility is $\overline{CE}$ high. This is a holding pattern where nothing happens. Internal data at all memory locations is frozen or unchanged. Notice that the output data lines are floating.

TABLE 9-2. STATIC RAM

$\overline{CE}$	$\overline{WE}$	Operation	Output
0	0	Write	Floating
0	1	Read	Connected
1	X	Hold	Floating

Bubble Memories

A *bubble memory* sandwiches a thin film of magnetic material between two permanent bias magnets. Logical 1s and 0s are represented by magnetic bubbles in this thin film. The details of how a bubble memory works are too complicated to go into here. What is worth knowing is that bubble memories are nonvolatile and capable of storing huge amounts of data. For instance, the INTEL 7110 is a bubble memory that can store approximately 1 million bits. One disadvantage is they have slow access times.

EXAMPLE 9-2

Figure 9-6 shows the pin configuration of a 74189, a Schottky TTL static RAM with three-state outputs. This 64-bit RAM is organized as 16 words of 4 bits each. It has an access time of 35 ns. What are the different pin functions?

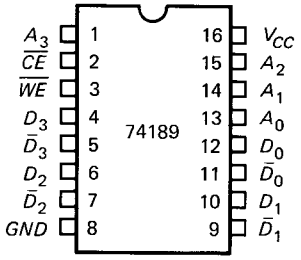


Fig. 9-6 Pinout for 74189.

SOLUTION

To begin with, 4 address bits can access $2^4 = 16$ words. This is why the 74189 needs 4 address bits to select the desired memory location.

The **ADDRESS** bits go to pin 1 (A_3), pin 15 (A_2), pin 14 (A_1), and pin 13 (A_0). The data inputs are pin 4 (D_3), pin 6 (D_2), pin 10 (D_1), and pin 12 (D_0). Because of the TTL design, the data is stored as the complement of the input bits. This is why the data outputs are pin 5 ($\overline{D}_3$), pin 7 ($\overline{D}_2$), pin 9 ($\overline{D}_1$), and pin 11 ($\overline{D}_0$).

The chip enable is pin 2, and the write enable is pin 3. These control signals work as previously described. $\overline{CE}$ and $\overline{WE}$ must be low for a write operation; $\overline{CE}$ must be low and $\overline{WE}$ high for a read, and $\overline{CE}$ must be high to do nothing.

Pin 16 gets the supply voltage, which is +5 V, and pin 8 is grounded.

9-4 A SMALL TTL MEMORY

Figure 9-7 shows a modified version of the SAP-1 memory. Two 74189s (see Appendix 4) are used to get a 16 × 8 memory. This means that we can store 16 words of 8 bits each. The bubbles on the output data pins (pins 5, 7, 9, 11) remind us that the stored data bits are the complements of the input data bits.

Addressing the Memory

The address bits come from an address-switch register (A_3, A_2, A_1, A_0). By setting the switches we can input any address from 0000 to 1111. As noted at the bottom of Fig. 9-7, an up address switch is equal to a 1. Therefore, the address with all switches up is 1111.

Setting Up Data

The data inputs come from the two other switch registers. The upper input nibble is D_7, D_6, D_5, and D_4. The lower input nibble is D_3, D_2, D_1, and D_0. By setting the data switches we can input any data word from 0000 0000 to 1111 1111, equivalent to 00H to FFH. The note at the bottom of Fig. 9-7 indicates that an up data switch produces an input 0 or an output 1. In other words, a data switch must be up to store a 1.

Programming the Memory

To *program the memory* (this means to store instruction and data words), the RUN-PROG switch must be in the PROG position. This grounds pin 2 ($\overline{CE}$) of each 74189. When the READ-WRITE switch is thrown to WRITE, pin 3 ($\overline{WE}$) is grounded and the complement of the input data word is written into the addressed memory location.

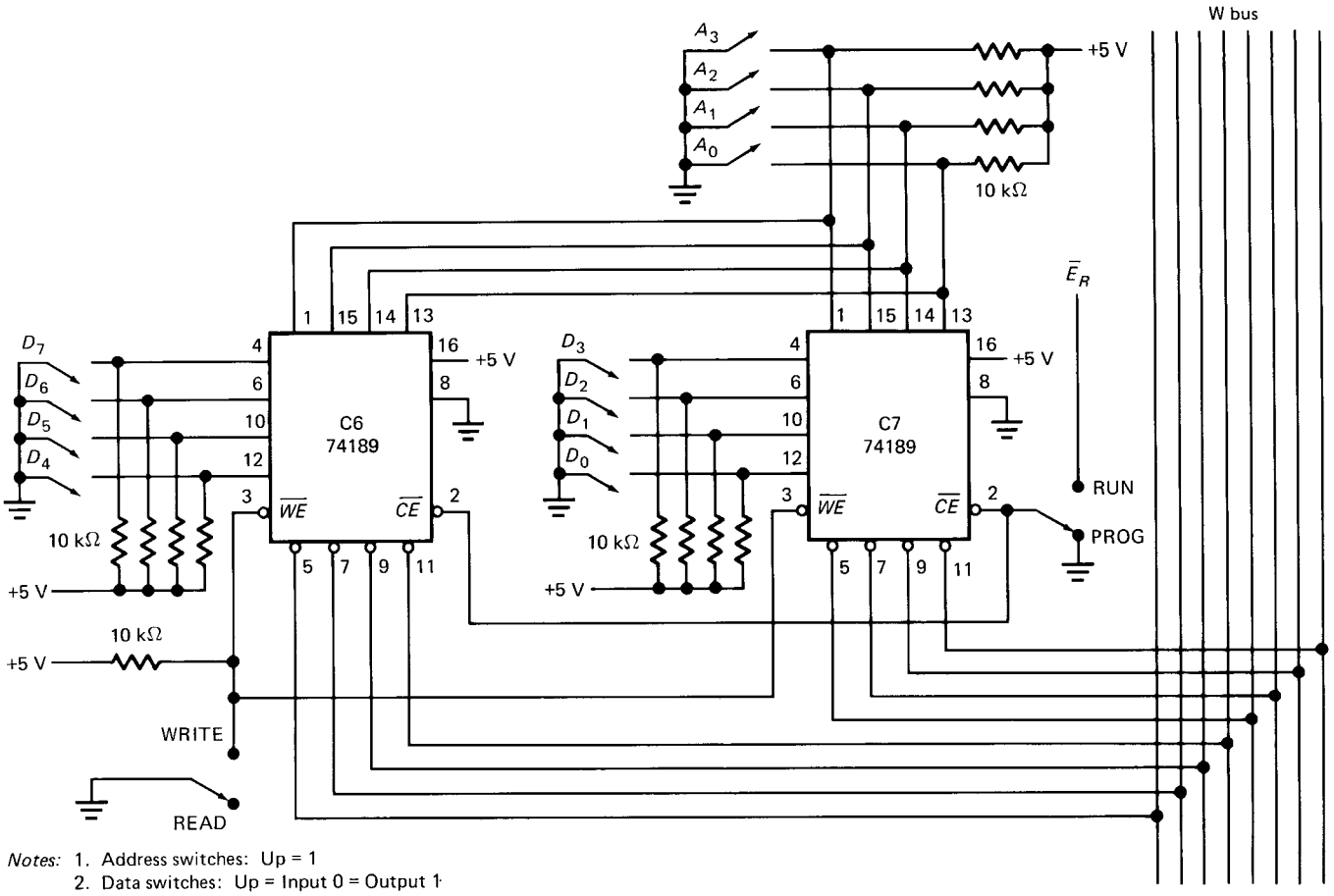
For instance, suppose we want to store the following words:

Address	Data
0000	0000 1111
0001	0010 1110
0010	0001 1101
0011	1110 1000

Begin by placing the RUN-PROG switch in the PROG position. To store the first data word at address 0000, set the switches as follows:

Address	Data
DDDD	DDDD UUUU

where D stands for down and U for up. When the READ-WRITE switch is thrown to WRITE, 0000 1111 is written into memory location 0000. The READ-WRITE switch is then returned to READ in preparation for the next WRITE operation.



Notes: 1. Address switches: Up = 1
2. Data switches: Up = Input 0 = Output 1

Fig. 9-7 Modified SAP-1 read-write memory.

To load the second word at address 0001, set the address and data switches as follows:

Address	Data
DDDU	DDUD UUUD

When the READ-WRITE switch is thrown to WRITE, the data word 0010 1110 is stored at memory location 0001.

Continuing like this, we can program the memory with the remaining words.

The SAP-1 memory is slightly different from Fig. 9-7 and will be discussed in Chap. 10. What we have discussed here, however, gives you an example of how a program and data can be entered into a memory before a computer run.

9-5 HEXADECIMAL ADDRESSES

During a computer run, the CPU sends binary addresses to the memory, where read or write operations occur. These address words may contain 16 or more bits. There's no need for us to get bogged down with long strings of binary numbers. We can chunk those 0s and 1s into neat strings of hexadecimal numbers. Using hexadecimal shorthand is standard in microprocessor work.

Typical microcomputers have an address bus with 16 address lines. The words on this bus have the binary format of

ADDRESS = XXXX XXXX XXXX XXXX

For convenience, we can chunk this into its equivalent hexadecimal form. For instance, instead of writing

ADDRESS = 0101 1110 0111 1100

we can write

ADDRESS = 5E7CH

The 16 address lines can access 2^{16} memory locations, equivalent to 65,536 words. The hexadecimal addresses are from 0000H to FFFFH. In microcomputers using 8-bit microprocessors, 1 byte is stored in each memory location. Figure 9-8 illustrates how to visualize such a memory. The first memory location has an address of 0000H, the second memory location an address of 0001H, the third an address

of 0002H, and so on. Moving toward higher memory, we eventually reach FFFDH, FFFEH, and FFFFH.

Notice that 1 byte is stored in each memory location. This is common in products using an 8-bit microprocessor like the Z80 and 6808. In other words, it is common for 8-bit microprocessor–based products to have a maximum memory of 64K (1K = 1,024 bytes).

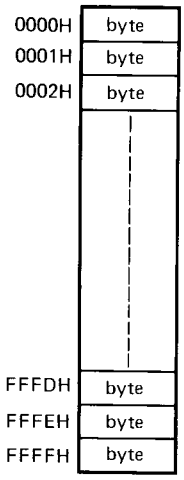


Fig. 9-8 Memory layout.

GLOSSARY

access time The time it takes to read the contents of a memory location after it has been addressed.

address A way of specifying the location of data in memory, similar to a house address.

dynamic memory A memory that relies on a MOSFET switch to charge a capacitor. This memory is highly volatile because not only must the power be kept on, but the capacitor charge must also be refreshed every few milliseconds.

EPROM Erasable programmable read-only memory, a device that is ultraviolet-erasable and electrically reprogrammable.

nonvolatile A type of memory in which the stored data is not lost when the power is turned off.

PROM Programmable read-only memory. With a PROM programmer, you can burn in your own programs and data.

RAM Random-access memory. It is also called a read-write memory because you can read the contents of a memory location or write new contents into it.

ROM Read-only memory. (ROM rhymes with Mom.) This device provides nonvolatile storage of programs and data. You can access any memory location by supplying its address.

static RAM A volatile memory using bipolar or MOSFET flip-flops. It is easy to work with. Refreshing data is unnecessary. You simply supply address and control bits for a read or write operation.

volatile A type of memory in which data stored in the memory is lost when the power is turned off.

SELF-TESTING REVIEW

Read each of the following and provide the missing words. Answers appear at the beginning of the next question.

1. The memory of a computer is where the _____ and _____ are stored before the calculations begin. During a computer run, partial answers may also be stored in the _____.
2. (*program, data, memory*) A read-only memory or _____ is equivalent to a group of memory locations, each permanently storing a word. The _____ is the only one who can store programs and data in a ROM.
3. (*ROM, manufacturer*) The _____ and contents of a memory location are two different things. Because the address is in binary form, the manufacturer uses on-chip decoding to access the memory location. With on-chip decoding, n address lines can access _____ memory locations.
4. (*address, 2^n*) The PROM allows users to store their own programs and data. An instrument called a PROM _____ does the storing or burning in. Once this is done, the programming is permanent.
5. (*programmer*) The _____ is ultraviolet-light-erasable and electrically programmable. This allows the user to erase and store until programs and data are perfected.
6. (*EPROM*) The _____ time of a memory is the

time it takes to read the contents of a memory location. Bipolar memories are faster than _____ memories but more expensive.

7. (*access, MOS*) ROMs, PROMs, and EPROMs are _____ memories. This means that they retain stored data even though the power is turned off. Core RAMs are also _____, but they are becoming obsolete.
8. (*nonvolatile, nonvolatile*) Semiconductor RAM memories may be static or _____. Both are volatile. The first type uses bipolar or MOS flip-flops, which means that data is stored as long as power is applied. The second type uses MOSFETs and capacitors to store data, which must be _____ every few milliseconds.
9. (*dynamic, refreshed*) The memory cell of a dynamic RAM is simpler and smaller than the memory cell of a _____ RAM. Because of this, the dynamic RAM can contains more memory cells than a _____ RAM of the same chip size.
10. (*static, static*) The _____ bits of a static RAM select the memory location. The write enable ($\overline{WE}$) and chip enable ($\overline{CE}$) select a write, read, or do-nothing. When $\overline{WE}$ and $\overline{CE}$ are both low, you get a _____ operation. When $\overline{WE}$ is high and $\overline{CE}$ is low, you get a _____ operation. $\overline{CE}$ high is the inactive state.
11. (*address, write, read*) During a computer run, the CPU sends binary addresses to the _____, where read or write operations occur. Typical microcomputers have an address bus with _____ bits.
12. (*memory, 16*) An address bus with 16 bits can access a maximum of 65,536 memory locations. The hexadecimal addresses of these memory locations are from 0000H to FFFFH. First-generation microcomputers store 1 byte in each memory location, which implies a maximum memory of 64K.

PROBLEMS

9-1. How many memory locations can 14 address bits access?

9-2. The 2708 is an 8,192-bit EPROM organized as a 1,024 × 8 memory. How many address pins does it have?

9-3. The 2732 is a 4,096 × 8 EPROM. How many address lines does it have?

9-4. An 8156 is a 2,048-bit static RAM with 256 words of 8 bits each. How many address lines does this RAM have?

9-5. Use U (up) and D (down) to program the TTL memory of Fig. 9-9 with the following data:

Address	Data
0000	1000 1001
0001	0111 1100
0010	0011 0110
0011	0010 0011
0100	0001 0111
0101	0101 1111
0110	1110 1101
0111	1111 1000

Show your answer by converting each 0 to a D and each 1 to a U.

9-6. The following data is to be programmed into the TTL memory of Fig. 9-9:

Address	Data
0H	EEH
1H	5CH
2H	26H
3H	6AH
4H	FDH
5H	15H
6H	94H
7H	C3H

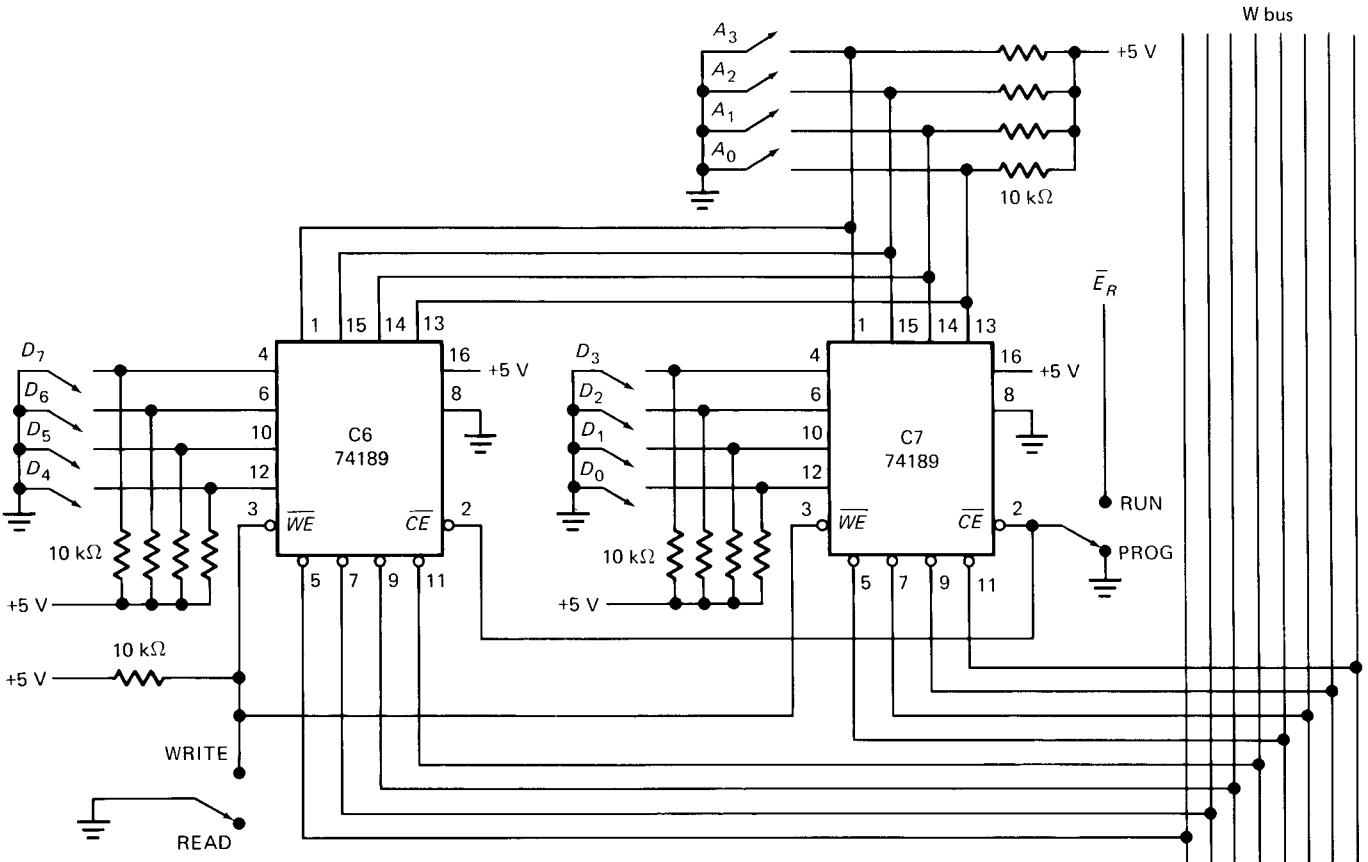
Convert these hexadecimal addresses and contents to ups (U) and downs (D) as described in Sec. 9-4.

9-7. Address 2000H contains the byte 3FH. What is the decimal equivalent of 3FH?

9-8. In a 32K memory, the hexadecimal addresses are from 0000H to 7FFFH. What is the decimal equivalent of the highest address?

9-9. What is the highest address in a 48K memory? Express the answer in hexadecimal and decimal form.

9-10. A byte is stored at hexadecimal location 6F9EH. What is the decimal address? (Use Appendix 2.)



Notes: 1. Address switches: Up = 1
2. Data switches: Up = Input 0 = Output 1

Fig. 9-9

9-11. Here is some data stored in a memory:

Address	Data
8E00H	2FH
8E01H	D4H
8E02H	CFH
8E03H	6EH
8E04H	53H
8E05H	7AH

a. What is the decimal equivalent of each stored byte? (Use Appendix 2.)
b. What is the decimal equivalent of the highest address?

9-12. Suppose there are four different memories with the following capacities:

Memory A = 16K
Memory B = 32K
Memory C = 48K
Memory D = 64K

All memories start with hexadecimal address 0000H.

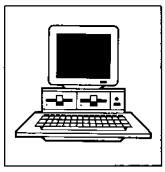
a. How many bytes can memory C store? Express the answer in decimal.
b. What is the highest decimal address in memory A?
c. We want to store a byte at address C300H. Which memory must we use?
d. What is the highest hexadecimal address for each memory?

9-13. What kind of memory can be programmed and then erased with ultraviolet light, so that it can be reprogrammed?

9-14. What kind of memory can be programmed and then erased with electrical pulses, so that it can be reprogrammed?

9-15. What kind of nonvolatile memory can have individual bytes reprogrammed without erasing the entire chip?

PART 2
SAP
(SIMPLE-AS-POSSIBLE) COMPUTERS



10

SAP-1

The SAP (Simple-As-Possible) computer has been designed for you, the beginner. The main purpose of SAP is to introduce all the crucial ideas behind computer operation without burying you in unnecessary detail. But even a simple computer like SAP covers many advanced concepts. To avoid bombarding you with too much all at once, we will examine three different generations of the SAP computer.

SAP-1 is the first stage in the evolution toward modern computers. Although primitive, SAP-1 is a big step for a beginner. So, dig into this chapter; master SAP-1, its architecture, its programming, and its circuits. Then you will be ready for SAP-2.

10-1 ARCHITECTURE

Figure 10-1 shows the *architecture* (structure) of SAP-1, a bus-organized computer. All register outputs to the W bus are three-state; this allows orderly transfer of data. All other register outputs are two-state; these outputs continuously drive the boxes they are connected to.

The layout of Fig. 10-1 emphasizes the registers used in SAP-1. For this reason, no attempt has been made to keep all control circuits in one block called the control unit, all input-output circuits in another block called the I/O unit, etc.

Many of the registers of Fig. 10-1 are already familiar from earlier examples and discussions. What follows is a brief description of each box; detailed explanations come later.

Program Counter

The program is stored at the beginning of the memory with the first instruction at binary address 0000, the second instruction at address 0001, the third at address 0010, and so on. The *program counter,* which is part of the control unit, counts from 0000 to 1111. Its job is to send to the memory the address of the next instruction to be fetched and executed. It does this as follows.

The program counter is reset to 0000 before each computer run. When the computer run begins, the program counter sends address 0000 to the memory. The program counter is then incremented to get 0001. After the first instruction is fetched and executed, the program counter sends address 0001 to the memory. Again the program counter is incremented. After the second instruction is fetched and executed, the program counter sends address 0010 to the memory. In this way, the program counter is keeping track of the next instruction to be fetched and executed.

The program counter is like someone pointing a finger at a list of instructions, saying do this first, do this second, do this third, etc. This is why the program counter is sometimes called a *pointer;* it points to an address in memory where something important is being stored.

Input and MAR

Below the program counter is the *input* and *MAR* block. It includes the address and data switch registers discussed in Sec. 9-4. These switch registers, which are part of the input unit, allow you to send 4 address bits and 8 data bits to the RAM. As you recall, instruction and data words are written into the RAM before a computer run.

The *memory address register* (MAR) is part of the SAP-1 memory. During a computer run, the address in the program counter is latched into the MAR. A bit later, the MAR applies this 4-bit address to the RAM, where a read operation is performed.

The RAM

The *RAM* is a 16 $\times$ 8 static TTL RAM. As discussed in Sec. 9-4, you can program the RAM by means of the address and data switch registers. This allows you to store a program and data in the memory before a computer run.

During a computer run, the RAM receives 4-bit addresses from the MAR and a read operation is performed. In this way, the instruction or data word stored in the RAM is placed on the W bus for use in some other part of the computer.

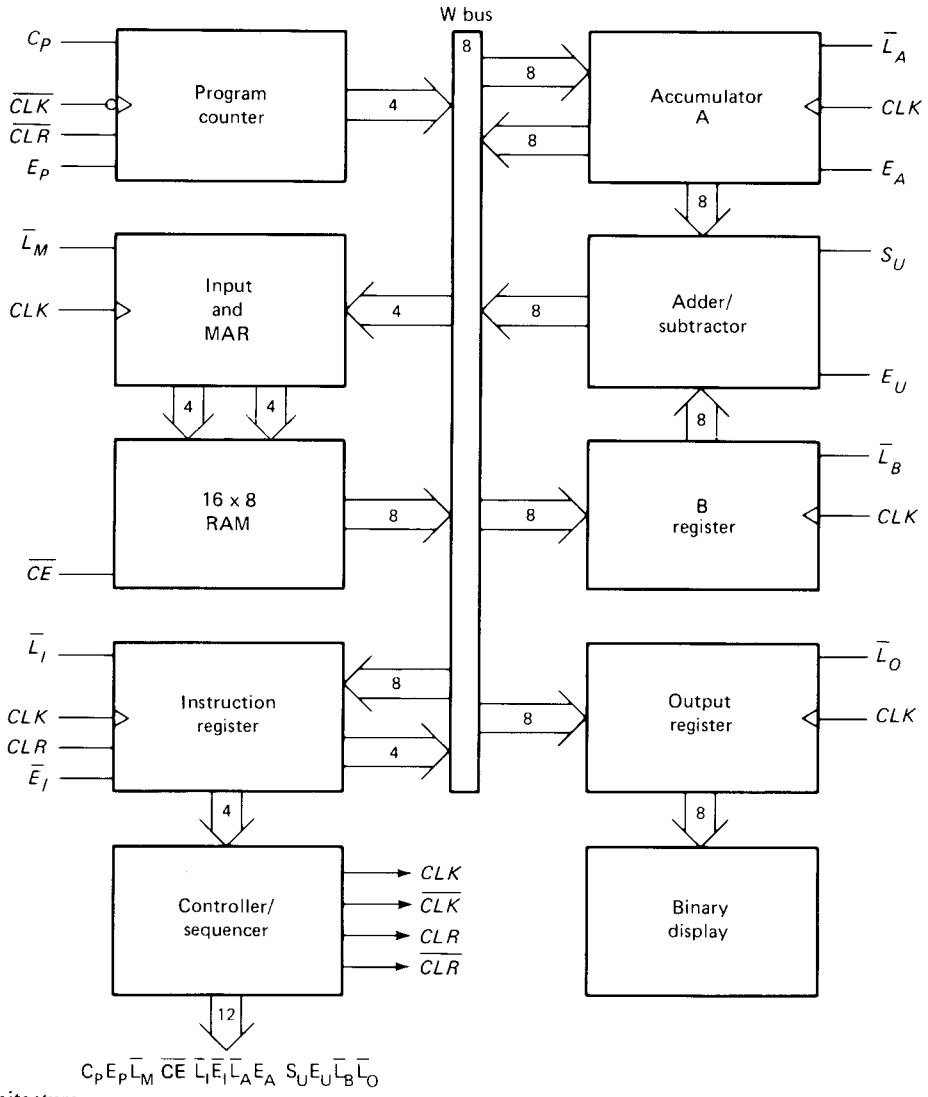


Fig. 10-1 SAP-1 architecture.

Instruction Register

The *instruction register* is part of the control unit. To fetch an instruction from the memory the computer does a memory read operation. This places the contents of the addressed memory location on the W bus. At the same time, the instruction register is set up for loading on the next positive clock edge.

The contents of the instruction register are split into two nibbles. The upper nibble is a two-state output that goes directly to the block labeled "Controller-sequencer." The lower nibble is a three-state output that is read onto the W bus when needed.

Controller-Sequencer

The lower left block contains the *controller-sequencer*. Before each computer run, a $\overline{CLR}$ signal is sent to the program counter and a *CLR* signal to the instruction register. This resets the program counter to 0000 and wipes out the last instruction in the instruction register.

A clock signal *CLK* is sent to all buffer registers; this synchronizes the operation of the computer, ensuring that things happen when they are supposed to happen. In other words, all register transfers occur on the positive edge of a common *CLK* signal. Notice that a $\overline{CLK}$ signal also goes to the program counter.

The 12 bits that come out of the controller-sequencer form a word controlling the rest of the computer (like a supervisor telling others what to do.) The 12 wires carrying the control word are called the *control bus*.

The control word has the format of

$$\mathbf{CON} = C_P E_P \overline{L}_M \overline{CE}\ \ \overline{L}_I \overline{E}_I \overline{L}_A E_A\ \ S_U E_U \overline{L}_B \overline{L}_O$$

This word determines how the registers will react to the next positive *CLK* edge. For instance, a high E_P and a low

$\overline{L}_M$ mean that the contents of the program counter are latched into the MAR on the next positive clock edge. As another example, a low $\overline{CE}$ and a low $\overline{L}_A$ mean that the addressed RAM word will be transferred to the accumulator on the next positive clock edge. Later, we will examine the timing diagrams to see exactly when and how these data transfers take place.

Accumulator

The *accumulator* (A) is a buffer register that stores intermediate answers during a computer run. In Fig. 10-1 the accumulator has two outputs. The two-state output goes directly to the adder-subtracter. The three-state output goes to the W bus. Therefore, the 8-bit accumulator word continuously drives the adder-subtracter; the same word appears on the W bus when E_A is high.

The Adder-Subtracter

SAP-1 uses a 2's-complement *adder-subtracter*. When S_U is low in Fig. 10-1, the sum out of the adder-subtracter is

$$\mathbf{S = A + B}$$

When S_U is high, the difference appears:

$$\mathbf{A = A + B'}$$

(Recall that the 2's complement is equivalent to a decimal sign change.)

The adder-subtracter is *asynchronous* (unclocked); this means that its contents can change as soon as the input words change. When E_U is high, these contents appear on the W bus.

B Register

The *B register* is another buffer register. It is used in arithmetic operations. A low $\overline{L}_B$ and positive clock edge load the word on the W bus into the B register. The two-state output of the B register drives the adder-subtracter, supplying the number to be added or subtracted from the contents of the accumulator.

Output Register

Example 8-1 discussed the output register. At the end of a computer run, the accumulator contains the answer to the problem being solved. At this point, we need to transfer the answer to the outside world. This is where the *output register* is used. When E_A is high and $\overline{L}_0$ is low, the next positive clock edge loads the accumulator word into the output register.

The output register is often called an *output port* because processed data can leave the computer through this register. In microcomputers the output ports are connected to *interface circuits* that drive peripheral devices like printers, cathode-ray tubes, teletypewriters, and so forth. (An interface circuit prepares the data to drive each device.)

Binary Display

The *binary display* is a row of eight light-emitting diodes (LEDs). Because each LED connects to one flip-flop of the output port, the binary display shows us the contents of the output port. Therefore, after we've transferred an answer from the accumulator to the output port, we can see the answer in binary form.

Summary

The SAP-1 control unit consists of the program counter, the instruction register, and the controller-sequencer that produces the control word, the clear signals, and the clock signals. The SAP-1 ALU consists of an accumulator, an adder-subtracter, and a B register. The SAP-1 memory has the MAR and a 16 × 8 RAM. The I/O unit includes the input programming switches, the output port, and the binary display.

10-2 INSTRUCTION SET

A computer is a useless pile of hardware until someone programs it. This means loading step-by-step instructions into the memory before the start of a computer run. Before you can program a computer, however, you must learn its *instruction set,* the basic operations it can perform. The SAP-1 instruction set follows.

LDA

As described in Chap. 9, the words in the memory can be symbolized by $\mathbf{R}_0$, $\mathbf{R}_1$, $\mathbf{R}_2$, etc. This means that $\mathbf{R}_0$ is stored at address 0H, $\mathbf{R}_1$ at address 1H, $\mathbf{R}_2$ at address 2H, and so on.

LDA stands for "load the accumulator." A complete LDA instruction includes the hexadecimal address of the data to be loaded. LDA 8H, for example, means "load the accumulator with the contents of memory location 8H." Therefore, given

$$\mathbf{R}_8 = 1111\ 0000$$

the execution of LDA 8H results in

$$\mathbf{A} = 1111\ 0000$$

Similarly, LDA AH means "load the accumulator with the contents of memory location AH," LDA FH means "load the accumulator with the contents of memory location FH," and so on.

ADD

ADD is another SAP-1 instruction. A complete ADD instruction includes the address of the word to be added. For instance, ADD 9H means "add the contents of memory location 9H to the accumulator contents"; the sum replaces the original contents of the accumulator.

Here's an example. Suppose decimal 2 is in the accumulator and decimal 3 is in memory location 9H. Then

$$\mathbf{A} = 0000\,0010$$
$$\mathbf{R}_9 = 0000\,0011$$

During the execution of ADD 9H, the following things happen. First, $\mathbf{R}_9$ is loaded into the B register to get

$$\mathbf{B} = 0000\,0011$$

and almost instantly the adder-subtracter forms the sum of **A** and **B**

$$\mathbf{SUM} = 0000\,0101$$

Second, this sum is loaded into the accumulator to get

$$\mathbf{A} = 0000\,0101$$

The foregoing routine is used for all ADD instructions; the addressed RAM word goes to the B register and the adder-subtracter output to the accumulator. This is why the execution of ADD 9H adds $\mathbf{R}_9$ to the accumulator contents, the execution of ADD FH adds $\mathbf{R}_F$ to the accumulator contents, and so on.

SUB

SUB is another SAP-1 instruction. A complete SUB instruction includes the address of the word to be subtracted. For example, SUB CH means "subtract the contents of memory location CH from the contents of the accumulator"; the difference out of the adder-subtracter then replaces the original contents of the accumulator.

For a concrete example, assume that decimal 7 is in the accumulator and decimal 3 is in memory location CH. Then

$$\mathbf{A} = 0000\,0111$$
$$\mathbf{R}_C = 0000\,0011$$

The execution of SUB CH takes place as follows. First, $\mathbf{R}_C$ is loaded into the B register to get

$$\mathbf{B} = 0000\,0011$$

and almost instantly the adder-subtracter forms the difference of **A** and **B**:

$$\mathbf{DIFF} = 0000\,0100$$

Second, this difference is loaded into the accumulator and

$$\mathbf{A} = 0000\,0100$$

The foregoing routine applies to all SUB instructions; the addressed RAM word goes to the B register and the adder-subtracter output to the accumulator. This is why the execution of SUB CH subtracts $\mathbf{R}_C$ from the contents of the accumulator, the execution of SUB EH subtracts $\mathbf{R}_E$ from the accumulator, and so on.

OUT

The instruction OUT tells the SAP-1 computer to transfer the accumulator contents to the output port. After OUT has been executed, you can see the answer to the problem being solved.

OUT is complete by itself; that is, you do not have to include an address when using OUT because the instruction does not involve data in the memory.

HLT

HLT stands for halt. This instruction tells the computer to stop processing data. HLT marks the end of a program, similar to the way a period marks the end of a sentence. You must use a HLT instruction at the end of every SAP-1 program; otherwise, you get computer trash (meaningless answers caused by runaway processing).

HLT is complete by itself; you do not have to include a RAM word when using HLT because this instruction does not involve the memory.

Memory-Reference Instructions

LDA, ADD, and SUB are called *memory-reference instructions* because they use data stored in the memory. OUT and HLT, on the other hand, are not memory-reference instructions because they do not involve data stored in the memory.

Mnemonics

LDA, ADD, SUB, OUT, and HLT are the instruction set for SAP-1. Abbreviated instructions like these are called *mnemonics* (memory aids). Mnemonics are popular in computer work because they remind you of the operation that will take place when the instruction is executed. Table 10-1 summarizes the SAP-1 instruction set.

The 8080 and 8085

The 8080 was the first widely used microprocessor. It has 72 instructions. The 8085 is an enhanced version of the 8080 with essentially the same instruction set. To make SAP practical, the SAP instructions will be upward com-

TABLE 10-1. SAP-1 INSTRUCTION SET

Mnemonic	Operation
LDA	Load RAM data into accumulator
ADD	Add RAM data to accumulator
SUB	Subtract RAM data from accumulator
OUT	Load accumulator data into output register
HLT	Stop processing

patible with the 8080/8085 instruction set. In other words, the SAP-1 instructions LDA, ADD, SUB, OUT, and HLT are 8080/8085 instructions. Likewise, the SAP-2 and SAP-3 instructions will be part of the 8080/8085 instruction set. Learning SAP instructions is getting you ready for the 8080 and 8085, two widely used microprocessors.

EXAMPLE 10-1

Here's a SAP-1 program in mnemonic form:

Address	Mnemonics
0H	LDA 9H
1H	ADD AH
2H	ADD BH
3H	SUB CH
4H	OUT
5H	HLT

The data in higher memory is

Address	Data
6H	FFH
7H	FFH
8H	FFH
9H	01H
AH	02H
BH	03H
CH	04H
DH	FFH
EH	FFH
FH	FFH

What does each instruction do?

SOLUTION

The program is in the low memory, located at addresses 0H to 5H. The first instruction loads the accumulator with the contents of memory location 9H, and so the accumulator contents become

$$\mathbf{A} = 01H$$

The second instruction adds the contents of memory location AH to the accumulator contents to get a new accumulator total of

$$\mathbf{A} = 01H + 02H = 03H$$

Similarly, the third instruction add the contents of memory location BH

$$\mathbf{A} = 03H + 03H = 06H$$

The SUB instruction subtracts the contents of memory location CH to get

$$\mathbf{A} = 06H - 04H = 02H$$

The OUT instruction loads the accumulator contents into the output port: therefore, the binary display shows

0000 0010

The HLT instruction stops the data processing.

10-3 PROGRAMMING SAP-1

To load instruction and data words into the SAP-1 memory we have to use some kind of code that the computer can interpret. Table 10-2 shows the code used in SAP-1. The number 0000 stands for LDA, 0001 for ADD, 0010 for SUB, 1110 for OUT, and 1111 for HLT. Because this code tells the computer which operation to perform, it is called an *operation code* (op code).

As discussed earlier, the address and data switches of Fig. 9-7 allow you to program the SAP-1 memory. By design, these switches produce a 1 in the up position (U)

TABLE 10-2. SAP-1 OP CODE

Mnemonic	Op code
LDA	0000
ADD	0001
SUB	0010
OUT	1110
HLT	1111

and a 0 in the down position (D). When programming the data switches with an instruction, the op code goes into the upper nibble, and the *operand* (the rest of the instruction) into the lower nibble.

For instance, suppose we want to store the following instructions:

Address	Instruction
0H	LDA FH
1H	ADD EH
2H	HLT

First, convert each instruction to binary as follows:

LDA FH = 0000 1111
ADD EH = 0001 1110
HLT = 1111 XXXX

In the first instruction, 0000 is the op code for LDA, and 1111 is the binary equivalent of FH. In the second instruction, 0001 is the op code for ADD, and 1110 is the binary equivalent of EH. In the third instruction, 1111 is the op code for HLT, and XXXX are don't cares because the HLT is not a memory-reference instruction.

Next, set up the address and data switches as follows:

Address	Data
DDDD	DDDD UUUU
DDDU	DDDU UUUD
DDUD	UUUU XXXX

After each address and data word is set, you press the write button. Since D stores a binary 0 and U stores a binary 1, the first three memory locations now have these contents:

Address	Contents
0000	0000 1111
0001	0001 1110
0010	1111 XXXX

A final point. *Assembly language* involves working with mnemonics when writing a program. *Machine language* involves working with strings of 0s and 1s. The following examples bring out the distinction between the two languages.

EXAMPLE 10-2

Translate the program of Example 10-1 into SAP-1 machine language.

SOLUTION

Here is the program of Example 10-1:

Address	Instruction
0H	LDA 9H
1H	ADD AH
2H	ADD BH
3H	SUB CH
4H	OUT
5H	HLT

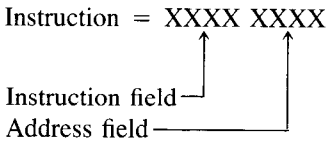
This program is in assembly language as it now stands. To get it into machine language, we translate it to 0s and 1s as follows:

Address	Instruction
0000	0000 1001
0001	0001 1010
0010	0001 1011
0011	0010 1100
0100	1110 XXXX
0101	1111 XXXX

Now the program is in machine language.

Any program like the foregoing that's written in machine language is called an *object program*. The original program with mnemonics is called a *source program*. In SAP-1 the operator translates the source program into an object program when programming the address and data switches.

A final point. The four MSBs of a SAP-1 machine-language instruction specify the operation, and the four LSBs give the address. Sometimes we refer to the MSBs as the *instruction field* and to the LSBs as the *address field*. Symbolically,



EXAMPLE 10-3

How would you program SAP-1 to solve this arithmetic problem?

$$16 + 20 + 24 - 32$$

The numbers are in decimal form.

SOLUTION

One way is to use the program of the preceding example, storing the data (16, 20, 24, 32) in memory locations 9H

to CH. With Appendix 2, you can convert the decimal data into hexadecimal data to get this assembly-language version:

Address	Contents
0H	LDA 9H
1H	ADD AH
2H	ADD BH
3H	SUB CH
4H	OUT
5H	HLT
6H	XX
7H	XX
8H	XX
9H	10H
AH	14H
BH	18H
CH	20H

The machine-language version is

Address	Contents
0000	0000 1001
0001	0001 1010
0010	0001 1011
0011	0010 1100
0100	1110 XXXX
0101	1111 XXXX
0110	XXXX XXXX
0111	XXXX XXXX
1000	XXXX XXXX
1001	0001 0000
1010	0001 0100
1011	0001 1000
1100	0010 0000

Notice that the program is stored ahead of the data. In other words, the program is in low memory and the data in high memory. This is essential in SAP-1 because the program counter points to address 0000 for the first instruction, 0001 for the second instruction, and so forth.

EXAMPLE 10-4

Chunk the program and data of the preceding example by converting to hexadecimal shorthand.

SOLUTION

Address	Contents
0H	09H
1H	1AH
2H	1BH
3H	2CH
4H	EXH
5H	FXH
6H	XXH
7H	XXH
8H	XXH
9H	10H
AH	14H
BH	18H
CH	20H

This version of the program and data is still considered machine language.

Incidentally, negative data is loaded in 2's-complement form. For example, −03H is entered as FDH.

10-4 FETCH CYCLE

The *control unit* is the key to a computer's automatic operation. The control unit generates the control words that fetch and execute each instruction. While each instruction is fetched and executed, the computer passes through different *timing states* (T states), periods during which register contents change. Let's find out more about these T states.

Ring Counter

Earlier, we discussed the SAP-1 ring counter (see Fig. 8-16 for the schematic diagram). Figure 10-2*a* symbolizes the ring counter, which has an output of

$$\mathbf{T} = T_6T_5T_4T_3T_2T_1$$

At the beginning of a computer run, the ring word is

$$\mathbf{T} = 000001$$

Successive clock pulses produce ring words of

$$\mathbf{T} = 000010$$
$$\mathbf{T} = 000100$$
$$\mathbf{T} = 001000$$
$$\mathbf{T} = 010000$$
$$\mathbf{T} = 100000$$

Then, the ring counter resets to 000001, and the cycle repeats. Each ring word represents one T state.

Figure 10-2*b* shows the timing pulses out of the ring counter. The initial state T_1 starts with a negative clock edge and ends with the next negative clock edge. During this T state, the T_1 bit out of the ring counter is high.

During the next state, T_2 is high; the following state has a high T_3; then a high T_4; and so on. As you can see, the

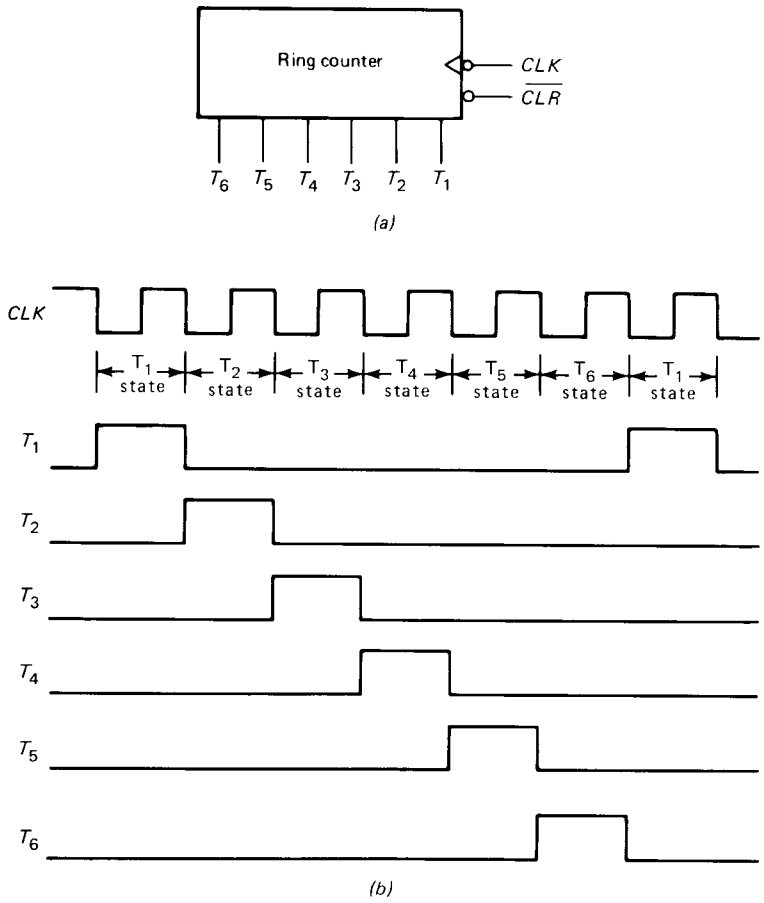


Fig. 10-2 Ring counter: (*a*) symbol; (*b*) clock and timing signals.

ring counter produces six T states. Each instruction is fetched and executed during these six T states.

Notice that a positive *CLK* edge occurs midway through each T state. The importance of this will be brought out later.

Address State

The T_1 state is called the *address state* because the address in the program counter (PC) is transferred to the memory address register (MAR) during this state. Figure 10-3*a* shows the computer sections that are active during this state (active parts are light; inactive parts are dark).

During the address state, E_P and $\overline{L}_M$ are active; all other control bits are inactive. This means that the controller-sequencer is sending out a control word of

$$\begin{aligned}\textbf{CON} &= C_P E_P \overline{L}_M \overline{CE} \quad \overline{L}_I \overline{E}_I \overline{L}_A E_A \quad S_U E_U \overline{L}_B \overline{L}_O \\ &= 0\ 1\ 0\ 1 \quad 1\ 1\ 1\ 0 \quad 0\ 0\ 1\ 1\end{aligned}$$

during this state.

Increment State

Figure 10-3*b* shows the active parts of SAP-1 during the T_2 state. This state is called the *increment state* because the program counter is incremented. During the increment state, the controller-sequencer is producing a control word of

$$\begin{aligned}\textbf{CON} &= C_P E_P \overline{L}_M \overline{CE} \quad \overline{L}_I \overline{E}_I \overline{L}_A E_A \quad S_U E_U \overline{L}_B \overline{L}_O \\ &= 1\ 0\ 1\ 1 \quad 1\ 1\ 1\ 0 \quad 0\ 0\ 1\ 1\end{aligned}$$

As you see, the C_P bit is active.

Memory State

The T_3 state is called the *memory state* because the addressed RAM instruction is transferred from the memory to the instruction register. Figure 10-3*c* shows the active parts of SAP-1 during the memory state. The only active control bits during this state are $\overline{CE}$ and $\overline{L}_I$, and the word out of the controller-sequencer is

$$\begin{aligned}\textbf{CON} &= C_P E_P \overline{L}_M \overline{CE} \quad \overline{L}_I \overline{E}_I \overline{L}_A E_A \quad S_U E_U \overline{L}_B \overline{L}_O \\ &= 0\ 0\ 1\ 0 \quad 0\ 1\ 1\ 0 \quad 0\ 0\ 1\ 1\end{aligned}$$

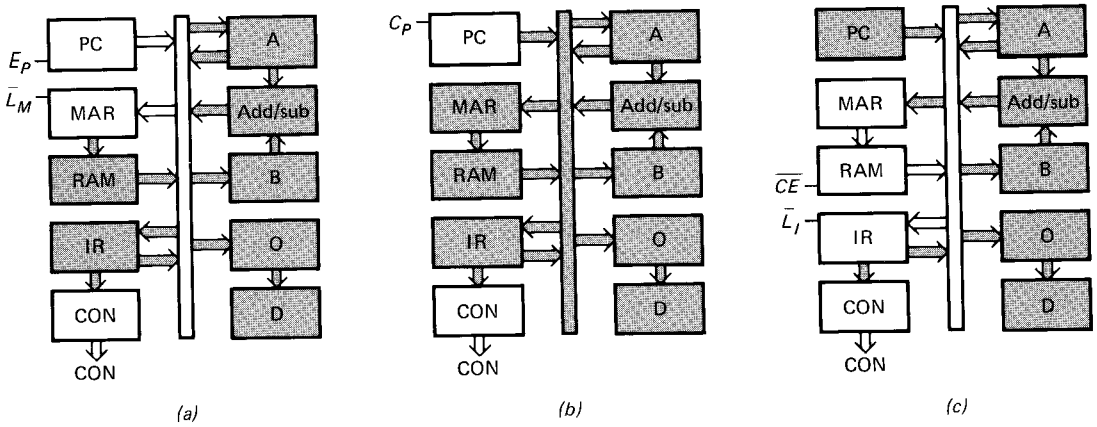


Fig. 10-3 Fetch cycle: (*a*) T_1 state; (*b*) T_2 state; (*c*) T_3 state.

Fetch Cycle

The address, increment, and memory states are called the *fetch cycle* of SAP-1. During the address state, E_P and $\overline{L}_M$ are active; this means that the program counter sets up the MAR via the W bus. As shown earlier in Fig. 10-2*b*, a positive clock edge occurs midway through the address state; this loads the MAR with the contents of the PC.

C_P is the only active control bit during the increment state. This sets up the program counter to count positive clock edges. Halfway through the increment state, a positive clock edge hits the program counter and advances the count by 1.

During the memory state, $\overline{CE}$ and $\overline{L}_I$ are active. Therefore, the addressed RAM word sets up the instruction register via the W bus. Midway through the memory state, a positive clock edge loads the instruction register with the addressed RAM word.

10-5 EXECUTION CYCLE

The next three states (T_4, T_5, and T_6) are the execution cycle of SAP-1. The register transfers during the execution cycle depend on the particular instruction being executed. For instance, LDA 9H requires different register transfers than ADD BH. What follows are the *control routines* for different SAP-1 instructions.

LDA Routine

For a concrete discussion, let's assume that the instruction register has been loaded with LDA 9H:

$$\mathbf{IR} = 0000\ 1001$$

During the T_4 state, the instruction field 0000 goes to the controller-sequencer, where it is decoded; the address field 1001 is loaded into the MAR. Figure 10-4*a* shows the active parts of SAP-1 during the T_4 state. Note that $\overline{E}_I$ and $\overline{L}_M$ are active; all other control bits are inactive.

During the T_5 state, $\overline{CE}$ and $\overline{L}_A$ go low. This means that the addressed data word in the RAM will be loaded into the accumulator on the next positive clock edge (see Fig. 10-4*b*).

T_6 is a no-operation state. During this third execution state, all registers are inactive (Fig. 10-4*c*). This means that the controller-sequencer is sending out a word whose bits are all inactive. *Nop* (pronounced *no op*) stands for "no operation." The T_6 state of the LDA routine is a nop.

Figure 10-5 shows the timing diagram for the fetch and LDA routines. During the T_1 state, E_P and $\overline{L}_M$ are active; the positive clock edge midway through this state will transfer the address in the program counter to the MAR. During the T_2 state, C_P is active and the program counter is incremented on the positive clock edge. During the T_3 state, $\overline{CE}$ and $\overline{L}_I$ are active; when the positive clock edge occurs, the addressed RAM word is transferred to the instruction register. The LDA execution starts with the T_4 state, where $\overline{L}_M$ and $\overline{E}_I$ are active; on the positive clock edge the address field in the instruction register is transferred to the MAR. During the T_5 state, $\overline{CE}$ and $\overline{L}_A$ are active; this means that the addressed RAM data word is transferred to the accumulator on the positive clock edge. As you know, the T_6 state of the LDA routine is a nop.

ADD Routine

Suppose at the end of the fetch cycle the instruction register contains ADD BH:

$$\mathbf{IR} = 0001\ 1011$$

During the T_4 state the instruction field goes to the controller-sequencer and the address field to the MAR (see Fig. 10-6*a*). During this state $\overline{E}_I$ and $\overline{L}_M$ are active.

Control bits $\overline{CE}$ and $\overline{L}_B$ are active during the T_5 state. This allows the addressed RAM word to set up the B

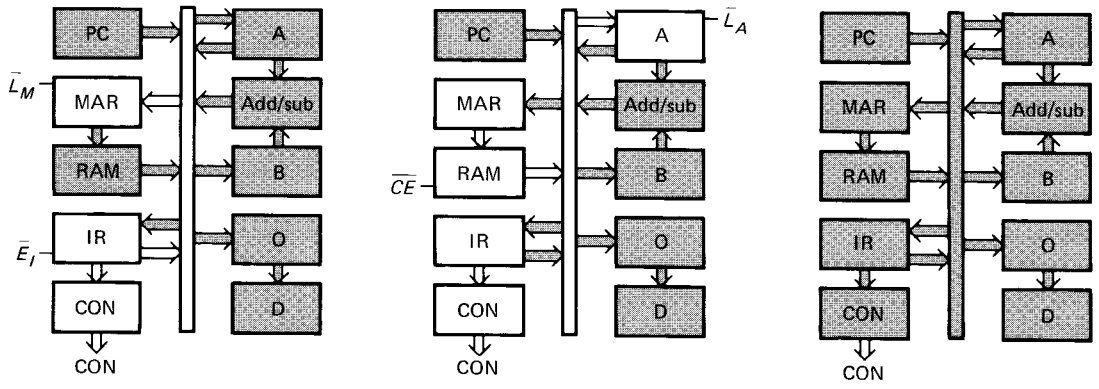


Fig. 10-4 LDA routine: (*a*) T_4 state; (*b*) T_5 state; (*c*) T_6 state.

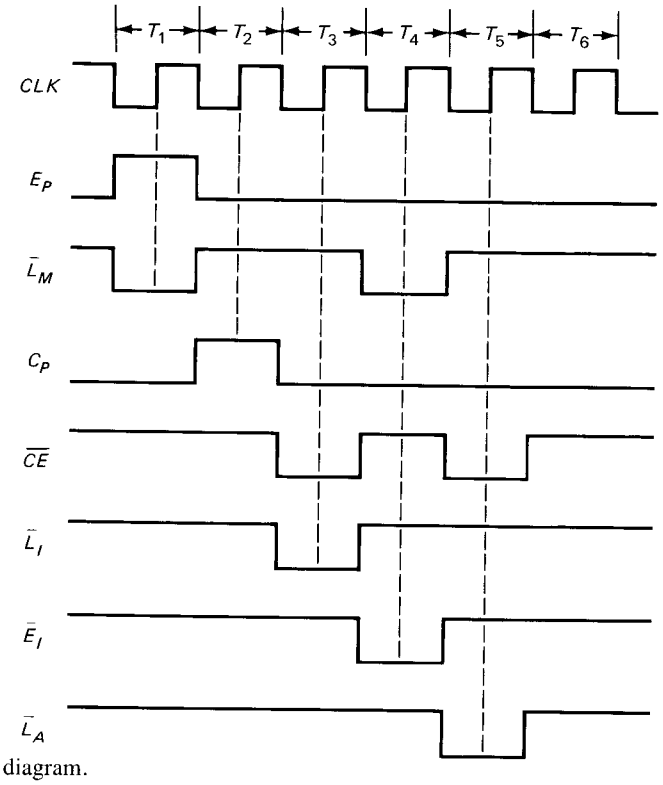


Fig. 10-5 Fetch and LDA timing diagram.

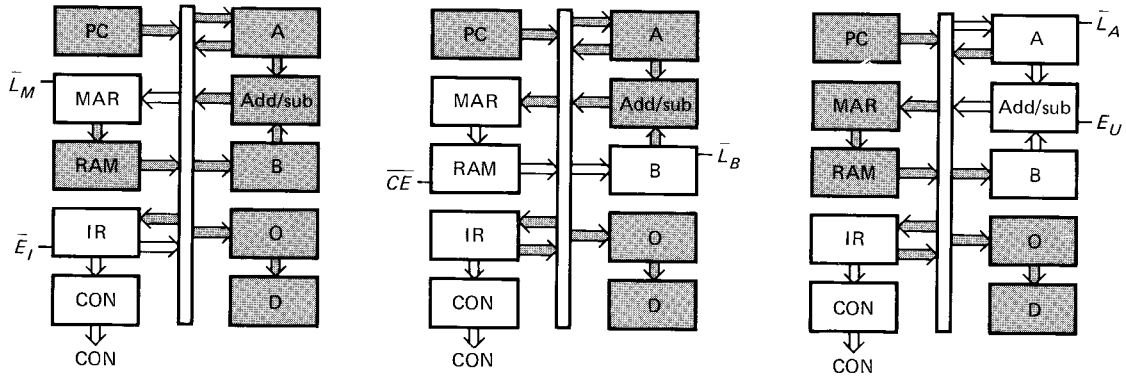


Fig. 10-6 ADD and SUB routines: (*a*) T_4 state; (*b*) T_5 state; (*c*) T_6 state.

register (Fig. 10-6*b*). As usual, loading takes place midway through the state when the positive clock edge hits the *CLK* input of the B register.

During the T_6 state, E_U and $\overline{L}_A$ are active; therefore, the adder-subtracter sets up the accumulator (Fig. 10-6*c*). Halfway through this state, the positive clock edge loads the sum into the accumulator.

Incidentally, setup time and propagation delay time prevent racing of the accumulator during this final execution state. When the positive clock edge hits in Fig. 10-6*c*, the accumulator contents change, forcing the adder-subtracter contents to change. The new contents return to the accumulator input, but the new contents don't get there until two propagation delays after the positive clock edge (one for the accumulator and one for the adder-subtracter). By then it's too late to set up the accumulator. This prevents accumulator racing (loading more than once on the same clock edge).

Figure 10-7 shows the timing diagram for the fetch and ADD routines. The fetch routine is the same as before: the T_1 state loads the PC address into the MAR; the T_2 state increments the program counter; the T_3 state sends the addressed instruction to the instruction register.

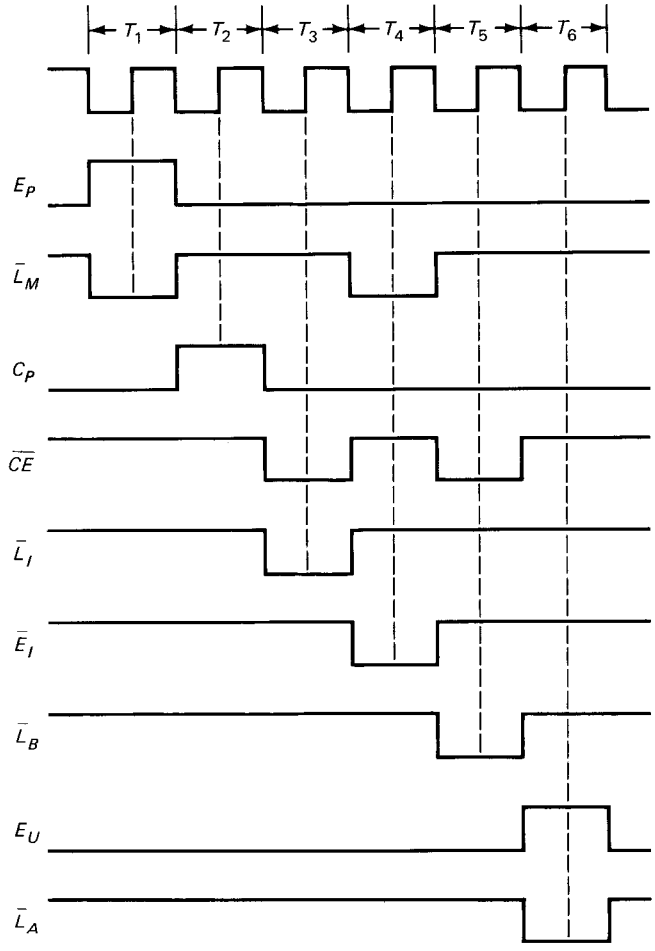


Fig. 10-7 Fetch and ADD timing diagram.

During the T_4 state, $\overline{E}_I$ and $\overline{L}_M$ are active; on the next positive clock edge, the address field in the instruction register goes to the MAR. During the T_5 state, $\overline{CE}$ and $\overline{L}_B$ are active; therefore, the addressed RAM word is loaded into the B register midway through the state. During the T_6 state, E_U and $\overline{L}_A$ are active; when the positive clock edge hits, the sum out of the adder-subtracter is stored in the accumulator.

SUB Routine

The SUB routine is similar to the ADD routine. Figure 10-6*a* and *b* show the active parts of SAP-1 during the T_4 and T_5 states. During the T_6 state, a high S_U is sent to the adder-subtracter of Fig. 10-6*c*. The timing diagram is almost identical to Fig. 10-7. Visualize S_U low during the T_1 to T_5 states and S_U high during the T_6 state.

OUT Routine

Suppose the instruction register contains the OUT instruction at the end of a fetch cycle. Then

$$\mathbf{IR} = 1110\ XXXX$$

The instruction field goes to the controller-sequencer for decoding. Then the controller-sequencer sends out the control word needed to load the accumulator contents into the output register.

Figure 10-8 shows the active sections of SAP-1 during the execution of an OUT instruction. Since E_A and $\overline{L}_O$ are active, the next positive clock edge loads the accumulator contents into the output register during the T_4 state. The T_5 and T_6 states are nops.

Figure 10-9 is the timing diagram for the fetch and OUT routines. Again, the fetch cycle is same: address state, increment state, and memory state. During the T_4 state, E_A and $\overline{L}_O$ are active; this transfers the accumulator word to the output register when the positive clock edge occurs.

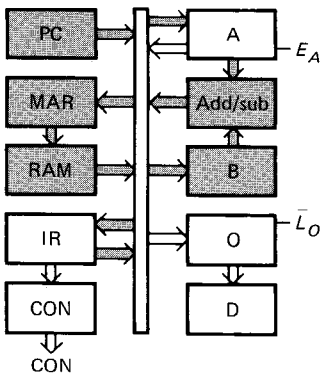


Fig. 10-8 T_4 state of OUT instruction.

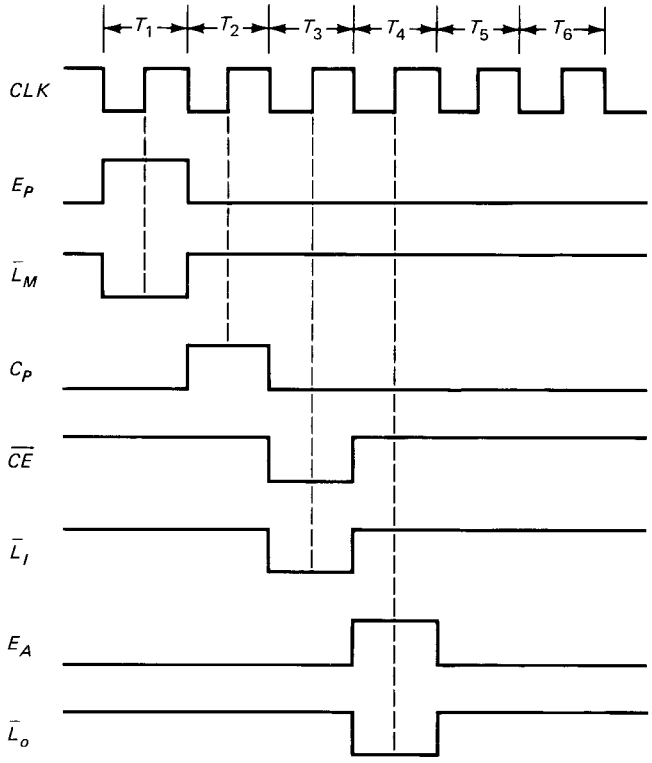


Fig. 10-9 Fetch and OUT timing diagram.

HLT

HLT does not require a control routine because no registers are involved in the execution of an HLT instruction. When the IR contains

IR = 1111 XXXX

the instruction field 1111 signals the controller-sequencer to stop processing data. The controller-sequencer stops the computer by turning off the clock (circuitry discussed later).

Machine Cycle and Instruction Cycle

SAP-1 has six T states (three fetch and three execute). These six states are called a *machine cycle* (see Fig. 10-10*a*). It takes one machine cycle to fetch and execute each instruction. The SAP-1 clock has a frequency of 1 kHz, equivalent to a period of 1 ms. Therefore, it takes 6 ms for a SAP-1 machine cycle.

SAP-2 is slightly different because some of its instructions take more than one machine cycle to fetch and execute. Figure 10-10*b* shows the timing for an instruction that requires two machine cycles. The first three T states are the fetch cycle; however, the execution cycle requires the next nine T states. This is because a two-machine-cycle instruction is more complicated and needs those extra T states to complete the execution.

The number of T states needed to fetch and execute an instruction is called the *instruction cycle*. In SAP-1 the instruction cycle equals the machine cycle. In SAP-2 and other microcomputers the instruction cycle may equal two or more machine cycles, as shown in Fig. 10-10*b*.

The instruction cycles for the 8080 and 8085 take from one to five machine cycles (more on this later).

EXAMPLE 10-5

The 8080/8085 programming manual says that it takes thirteen T states to fetch and execute the LDA instruction.

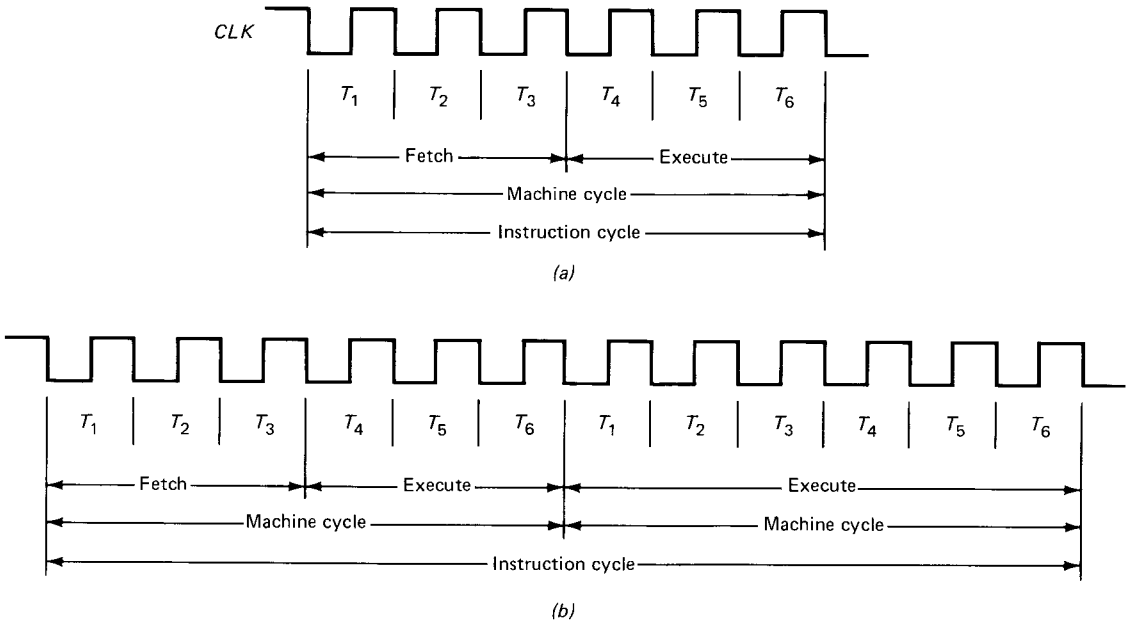


Fig. 10-10 (*a*) SAP-1 instruction cycle; (*b*) instruction cycle with two machine cycles.

If the system clock has a frequency of 2.5 MHz, how long is an instruction cycle?

SOLUTION

The period of the clock is

$$T = \frac{1}{f} = \frac{1}{2.5\text{ MHz}} = 400\text{ ns}$$

Therefore, each T state lasts 400 ns. Since it takes thirteen T states to fetch and execute the LDA instruction, the instruction cycle lasts for

$$13 \times 400\text{ ns} = 5{,}200\text{ ns} = 5.2\ \mu\text{s}$$

EXAMPLE 10-6

Figure 10-11 shows the six T states of SAP-1. The positive clock edge occurs halfway through each state. Why is this important?

SOLUTION

SAP-1 is a *bus-organized computer* (the common type nowadays). This allows its registers to communicate via the W bus. But reliable loading of a register takes place only when the setup and hold times are satisfied. Waiting half a cycle before loading the register satisfies the setup time; waiting half a cycle after loading satisfies the hold time. This is why the positive clock edge is designed to strike the registers halfway through each T state (Fig. 10-11).

There's another reason for waiting half a cycle before loading a register. When the *ENABLE* input of the sending register goes active, the contents of this register are suddenly dumped on the W bus. Stray capacitance and lead inductance prevent the bus lines from reaching their correct voltage levels immediately. In other words, we get transients on the W bus and have to wait for them to die out to ensure valid data at the time of loading. The half-cycle delay before clocking allows the data to settle before loading.

10-6 THE SAP-1 MICROPROGRAM

We will soon be analyzing the schematic diagram of the SAP-1 computer, but first we need to summarize the execution of SAP-1 instructions in a neat table called a *microprogram*.

Microinstructions

The controller-sequencer sends out control words, one during each T state or clock cycle. These words are like directions telling the rest of the computer what to do. Because it produces a small step in the data processing, each control word is called a *microinstruction*. When looking at the SAP-1 block diagram (Fig. 10-1), we can visualize a steady stream of microinstructions flowing out of the controller-sequencer to the other SAP-1 circuits.

Macroinstructions

The instructions we have been programming with (LDA, ADD, SUB, . . .) are sometimes called *macroinstructions* to distinguish them from microinstructions. Each SAP-1 macroinstruction is made up of three microinstructions. For example, the LDA macroinstruction consists of the microinstructions in Table 10-3. To simplify the appearance of these microinstructions, we can use hexadecimal chunking as shown in Table 10-4.

Table 10-5 shows the SAP-1 microprogram, a listing of each macroinstruction and the microinstructions needed to carry it out. This table summarizes the execute routines for the SAP-1 instructions. A similar table can be used with more advanced instruction sets.

10-7 THE SAP-1 SCHEMATIC DIAGRAM

In this section we examine the complete schematic diagram for SAP-1. Figures 10-12 to 10-15 show all the chips, wires, and signals. You should refer to these figures throughout the following discussion. Appendix 4 gives additional details for some of the more complicated chips.

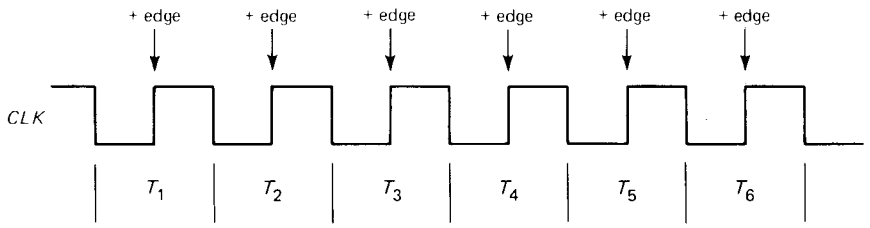


Fig. 10-11 Positive clock edges occur midway through T states.

TABLE 10-3

Macro	State	$C_P E_P \overline{L}_M \overline{CE}$	$\overline{L}_I \overline{E}_I \overline{L}_A E_A$	$S_U E_U \overline{L}_B \overline{L}_O$	Active
LDA	T_4	0 0 0 1	1 0 1 0	0 0 1 1	$\overline{L}_M$, $\overline{E}_I$
	T_5	0 0 1 0	1 1 0 0	0 0 1 1	$\overline{CE}$, $\overline{L}_A$
	T_6	0 0 1 1	1 1 1 0	0 0 1 1	None

TABLE 10-4

Macro	State	CON	Active
LDA	T_4	1A3H	$\overline{L}_M$, $\overline{E}_I$
	T_5	2C3H	$\overline{CE}$, $\overline{L}_A$
	T_6	3E3H	None

TABLE 10-5. SAP-1 MICROPROGRAM†

Macro	State	CON	Active
LDA	T_4	1A3H	$\overline{L}_M$, $\overline{E}_I$
	T_5	2C3H	$\overline{CE}$, $\overline{L}_A$
	T_6	3E3H	None
ADD	T_4	1A3H	$\overline{L}_M$, $\overline{E}_I$
	T_5	2E1H	$\overline{CE}$, $\overline{L}_B$
	T_6	3C7H	$\overline{L}_A$, E_U
SUB	T_4	1A3H	$\overline{L}_M$, $\overline{E}_I$
	T_5	2E1H	$\overline{CE}$, $\overline{L}_B$
	T_6	3CFH	$\overline{L}_A$, S_U, E_U
OUT	T_4	3F2H	E_A, $\overline{L}_O$
	T_5	3E3H	None
	T_6	3E3H	None

† CON = $C_P E_P \overline{L}_M \overline{CE}$ $\overline{L}_I \overline{E}_I \overline{L}_A E_A$ $S_U E_U \overline{L}_B \overline{L}_O$.

Program Counter

Chips C1, C2, and C3 of Fig. 10-12 are the *program counter*. Chip C1, a 74LS107, is a dual *JK* master-slave flip-flop, that produces the upper 2 address bits. Chip C2, another 74LS107, produces the lower 2 address bits. Chip C3 is a 74LS126, a quad three-state normally open switch; it gives the program counter a three-state output.

At the start of a computer run, a low $\overline{CLR}$ resets the program counter to 0000. During the T_1 state, a high E_P places the address on the W bus. During the T_2 state, a high C_P is applied to the program counter; midway through this state, the negative $\overline{CLK}$ edge (equivalent to positive *CLK* edge) increments the program counter.

The program counter is inactive during the T_3 to T_6 states.

MAR

Chip C4, a 74LS173, is a 4-bit buffer register; it serves as the MAR. Notice that pins 1 and 2 are grounded; this converts the three-state output to a two-state output. In other words, the output of the MAR is not connected to the W bus, and so there's no need to use the three-state output.

2-to-1 Multiplexer

Chip C5 is a 74LS157, a 2-to-1 nibble *multiplexer*. The left nibble (pins 14, 11, 5, 2) comes from the address switch register (S_1). The right nibble (pins 13, 10, 6, 3) comes from the MAR. The RUN-PROG switch (S_2) selects the nibble to reach to the output of C5. When S_2 is in the PROG position, the nibble out of the address switch register is selected. On the other hand, when S_2 is the RUN position, the output of the MAR is selected.

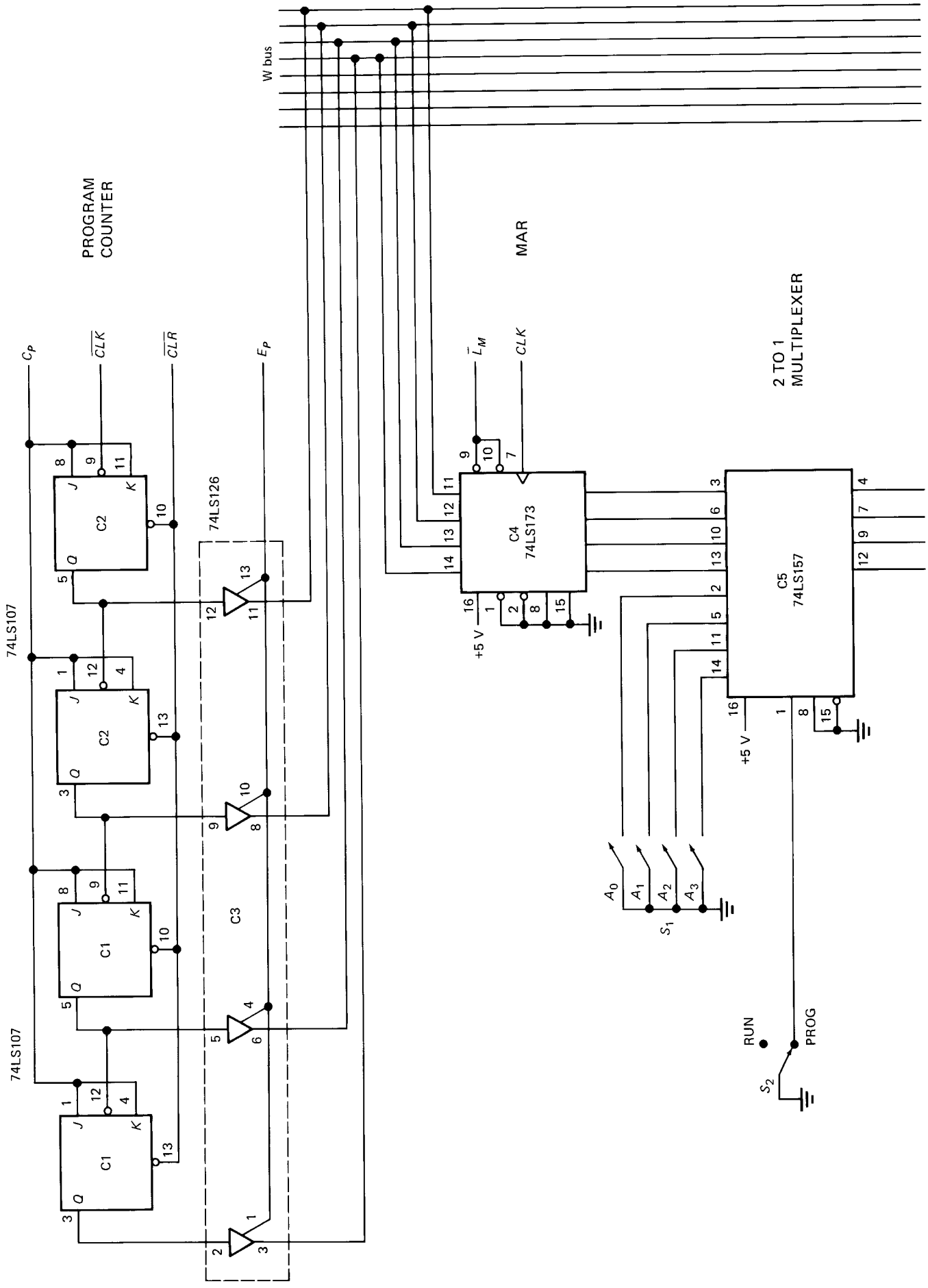
16 × 8 RAM

Chips C6 and C7 are 74189s. Each chip is a 16 × 4 static RAM. Together, they give us a 16 × 8 *read-write memory*. S_3 is the data switch register (8 bits), and S_4 is the read-write switch (a push-button switch). To program the memory, S_2 is put in the PROG position; this takes the $\overline{CE}$ input low (pin 2). The address and data switches are then set to the correct address and data words. A momentary push of the read-write switch takes $\overline{WE}$ low (pin 3) and loads the memory.

After the program and data are in memory, the RUN-PROG switch (S_2) is put in the RUN position in preparation for the computer run.

Instruction Register

Chips C8 and C9 are 74LS173s. Each chip is a 4-bit three-state buffer register. The two chips are the *instruction register*. Grounding pins 1 and 2 of C8 converts the three-state output to a two-state output, $I_7 I_6 I_5 I_4$. This nibble goes to the instruction decoder in the controller-sequencer. Signal $\overline{E}_I$ controls the output of C9, the lower nibble in the instruction register. When $\overline{E}_I$ is low, this nibble is placed on the W bus.


W bus
PROGRAM COUNTER
MAR
2 TO 1 MULTIPLEXER
C_P
$\overline{CLK}$
$\overline{CLR}$
E_P
$\overline{L}_M$
CLK
74LS107
74LS126
C1
C2
C3
C4 74LS173
C5 74LS157
+5 V
A_0
A_1
A_2
A_3
S_1
S_2
RUN
PROG

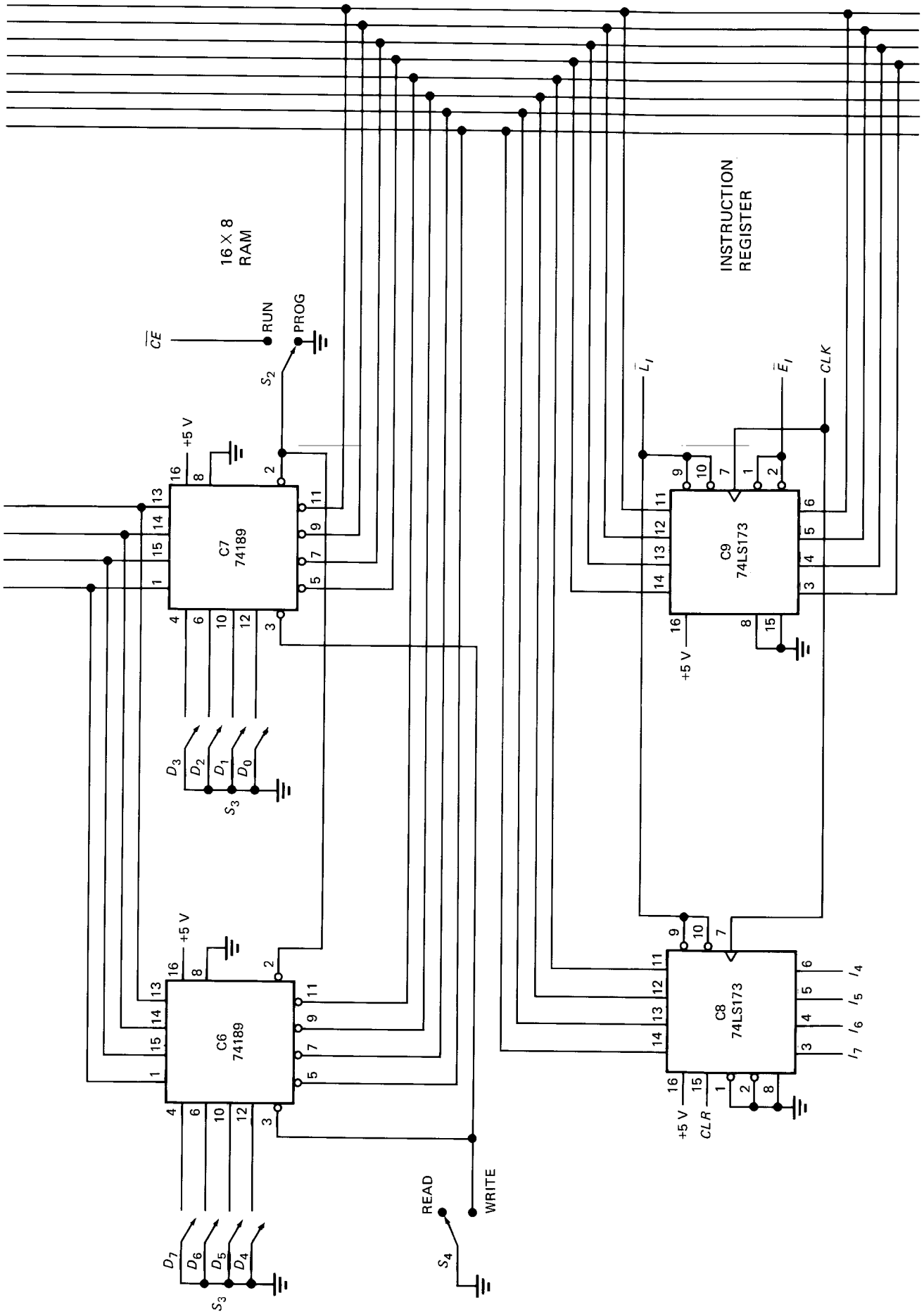
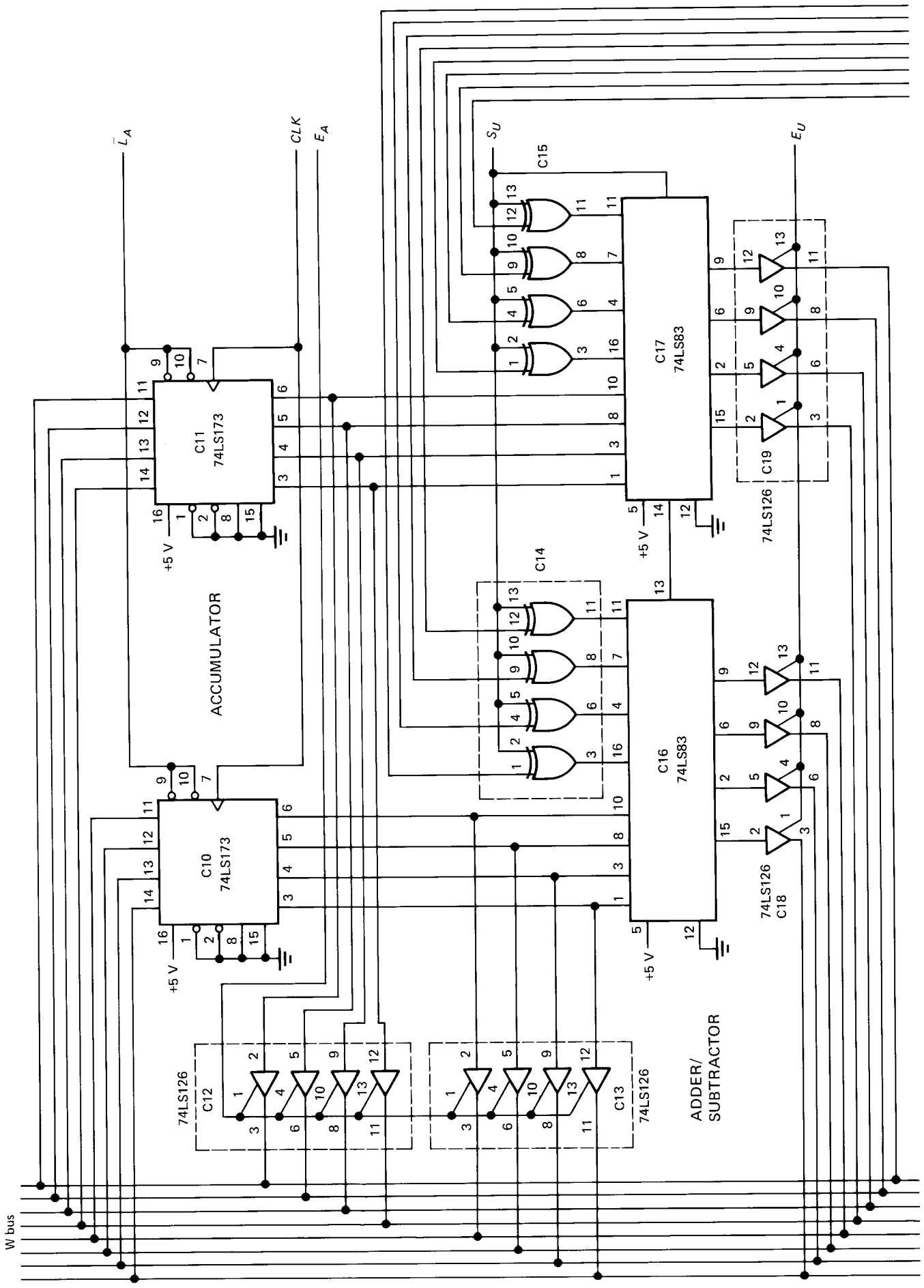


Fig. 10-12 SAP-1 program counter, memory, and instruction register.


$\overline{L}_A$
CLK
E_A
S_U
E_U
C15
C17
74LS83
C19
74LS126
C14
C16
74LS83
C11
74LS173
C10
74LS173
ACCUMULATOR
+5 V
C12
74LS126
C13
74LS126
C18
74LS126
ADDER/
SUBTRACTOR
W bus

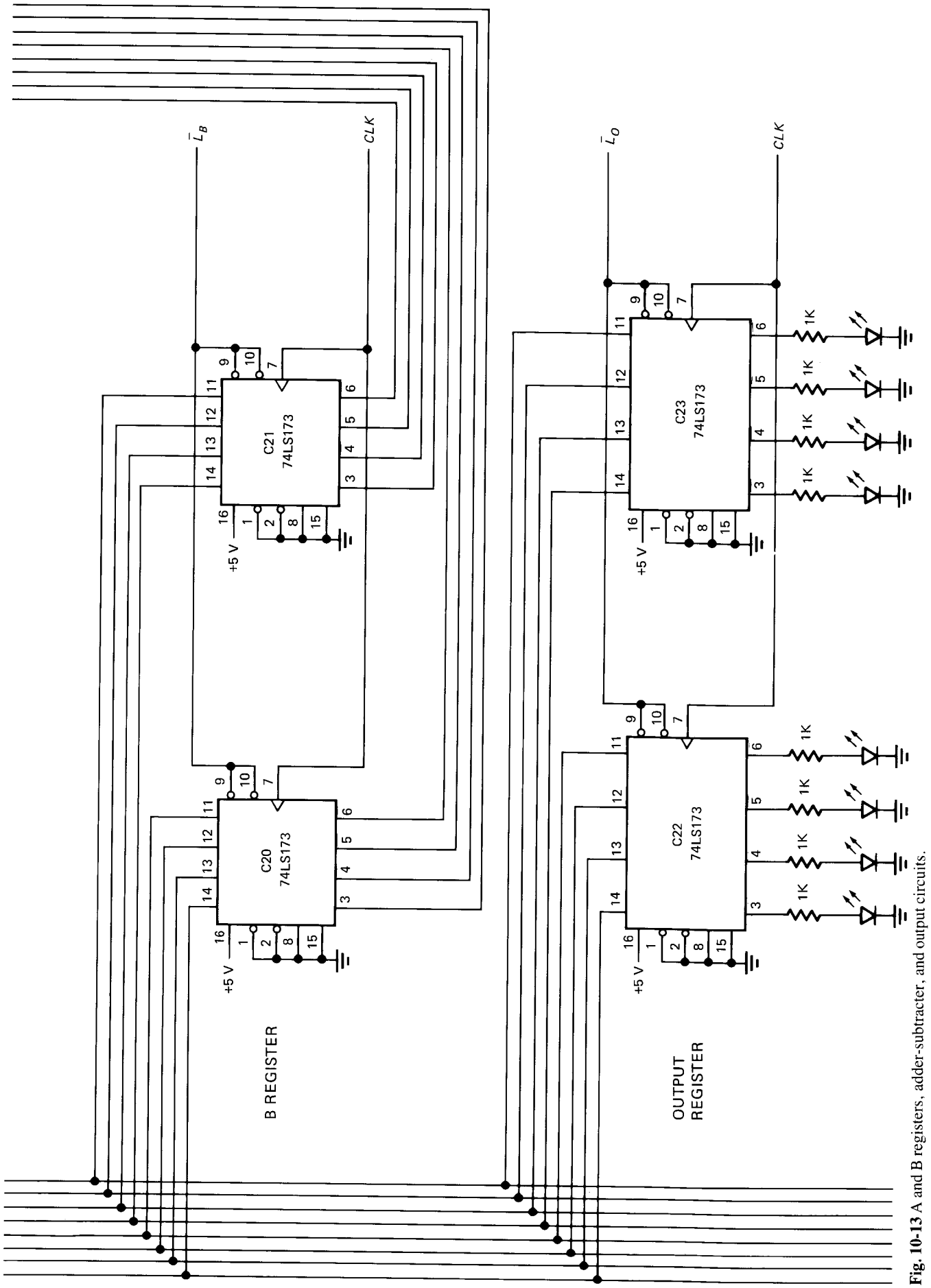


Fig. 10-13 A and B registers, adder-subtracter, and output circuits.

Accumulator

Chips C10 and C11, 74LS173s, are the *accumulator* (see Fig. 10-13). Pins 1 and 2 are grounded on both chips to produce a two-state output for the adder-subtracter. Chips C12 and C13 are 74LS126s; these three-state switches place the accumulator contents on the W bus when E_A is high.

Adder-subtracter

Chips C14 and C15 are 74LS86s. These EXCLUSIVE-OR gates are a controlled inverter. When S_U is low, the contents of the B register are transmitted. When S_U is high, the 1's complement is transmitted and a 1 is added to the LSB to form the 2's complement.

Chips C16 and C17 are 74LS83s. These 4-bit full adders combine to produce an 8-bit sum or difference. Chips C18 and C19, which are 74LS126s, convert this 8-bit answer into a three-state output for driving the W bus.

B Register and Output Register

Chips C20 and C21, which are 74LS173s, form the *B register*. It contains the data to be added or subtracted from the accumulator. Grounding pins 1 and 2 of both chips produces a two-state output for the adder-subtracter.

Chips C22 and C23 are 74LS173s and form the output register. It drives the binary display and lets us see the processed data.

Clear-Start Debouncer

In Fig. 10-14, the *clear-start debouncer* produces two outputs: *CLR* for the instruction register and $\overline{CLR}$ for the program counter and ring counter. $\overline{CLR}$ also goes to C29, the clock-start flip-flop. S_5 is a push-button switch. When depressed, it goes to the CLEAR position, generating a high *CLR* and a low $\overline{CLR}$. When S_5 is released, it returns to the START position, producing a low *CLR* and a high $\overline{CLR}$.

Notice that half of C24 is used for the clear-start debouncer and the other half for the single-step debouncer. Chip C24 is a 7400, a quad 2-input NAND gate.

Single-Step Debouncer

SAP-1 can run in either of two modes, manual or automatic. In the manual mode, you press and release S_6 to generate one clock pulse. When S_6 is depressed, *CLK* is high; when released, *CLK* is low. In other words, the *single-step debouncer* of Fig. 10-14 generates the *T* states one at a time as you press and release the button. This allows you to step through the different *T* states while troubleshooting or debugging. (Debugging means looking for errors in your program. You troubleshoot hardware and debug software.)

Manual-Auto Debouncer

Switch S_7 is a single-pole double-throw (SPDT) switch that can remain in either the MANUAL position or the AUTO position. When in MANUAL, the single-step button is active. When in AUTO, the computer runs automatically. Two of the NAND gates in C26 are used to debounce the MANUAL-AUTO switch. The other two NAND C26 gates are part of a NAND-NAND network that steers the single-step clock or the automatic clock to the final *CLK* and $\overline{CLK}$ outputs.

Clock Buffers

The output of pin 11, C26, drives the *clock buffers*. As you see in Fig. 10-14, two inverters are used to produce the final *CLK* output and one inverter to produce the $\overline{CLK}$ output. Unlike most of the other chips, C27 is standard TTL rather than a low-power Schottky (see SAP-1 Parts List, Appendix 5). Standard TTL is used because it can drive 20 low-power Schottky TTL loads, as indicated in Table 4-5.

If you check the data sheets of the 74LS107 and 74LS173 for input currents, you will be able to count the following low-power Schottky (LS) TTL loads on the clock and clear signals:

$$\begin{aligned} CLK &= 19 \text{ LS loads} \\ \overline{CLK} &= 2 \text{ LS loads} \\ CLR &= 1 \text{ LS load} \\ \overline{CLR} &= 20 \text{ LS loads} \end{aligned}$$

This means that the *CLK* and $\overline{CLK}$ signals out of C27 (standard TTL) are adequate to drive the low-power Schottky TTL loads. Also, the *CLR* and $\overline{CLR}$ signals out of C24 (standard TTL) can drive their loads.

Clock Circuits and Power Supply

Chip C28 is a 555 timer. This IC produces a rectangular 2-kHz output with a 75 percent duty cycle. As previously discussed, a *start-the-clock flip-flop* (C29) divides the signal down to 1 kHz and at the same time produces a 50 percent duty cycle.

The *power supply* consists of a full-wave bridge rectifier working into a capacitor-input filter. The dc voltage across the 1,000-μF capacitor is approximately 20 V. Chip C30, an LM340T-5, is a voltage regulator that produces a stable output of +5 V.

Instruction Decoder

Chip C31, a hex inverter, produces complements of the op-code bits, $I_7I_6I_5I_4$ (see Fig. 10-15). Then chips C32, C33, and C34 decode the op code to produce five output signals: *LDA*, *ADD*, *SUB*, *OUT*, and $\overline{HLT}$. Remember:

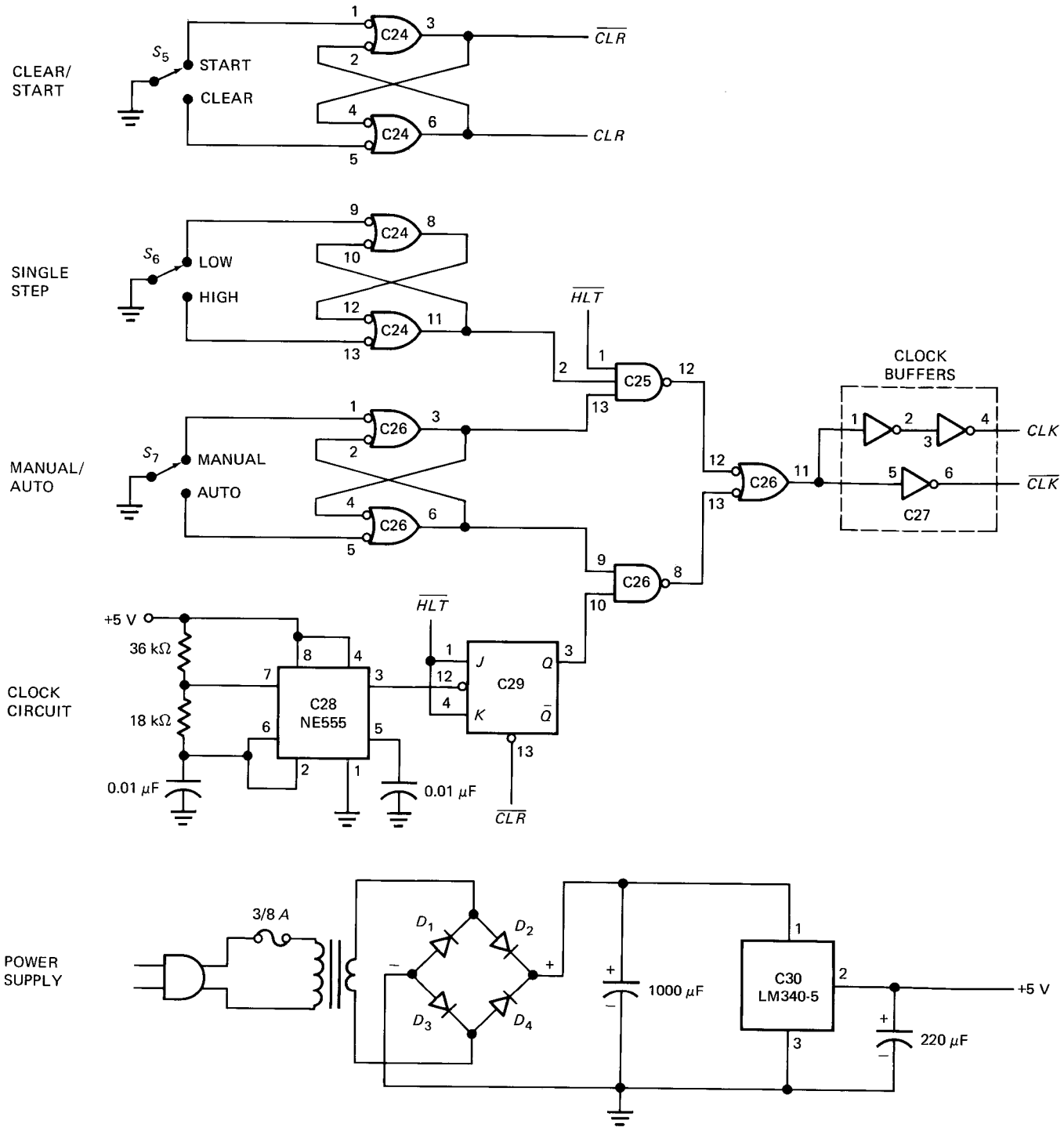


Fig. 10-14 Power supply, clock, and clear circuits.

only one of these is active at a time. ($\overline{HLT}$ is active low; all the others are active high.)

When the HLT instruction is in the *instruction register*, bits $I_7I_6I_5I_4$ are 1111 and $\overline{HLT}$ is low. This signal returns to C25 (single-step clock) and C29 (automatic clock). In either MANUAL or AUTO mode, the clock stops and the computer run ends.

Ring Counter

The ring counter, sometimes called a *state counter*, consists of three chips, C36, C37, and C38. Each of these chips is a 74LS107, a dual *JK* master-slave flip-flop. This counter is reset when the clear-start button (S_5) is pressed. The Q_0 flip-flop is inverted so that its $\overline{Q}$ output (pin 6, C38) drives

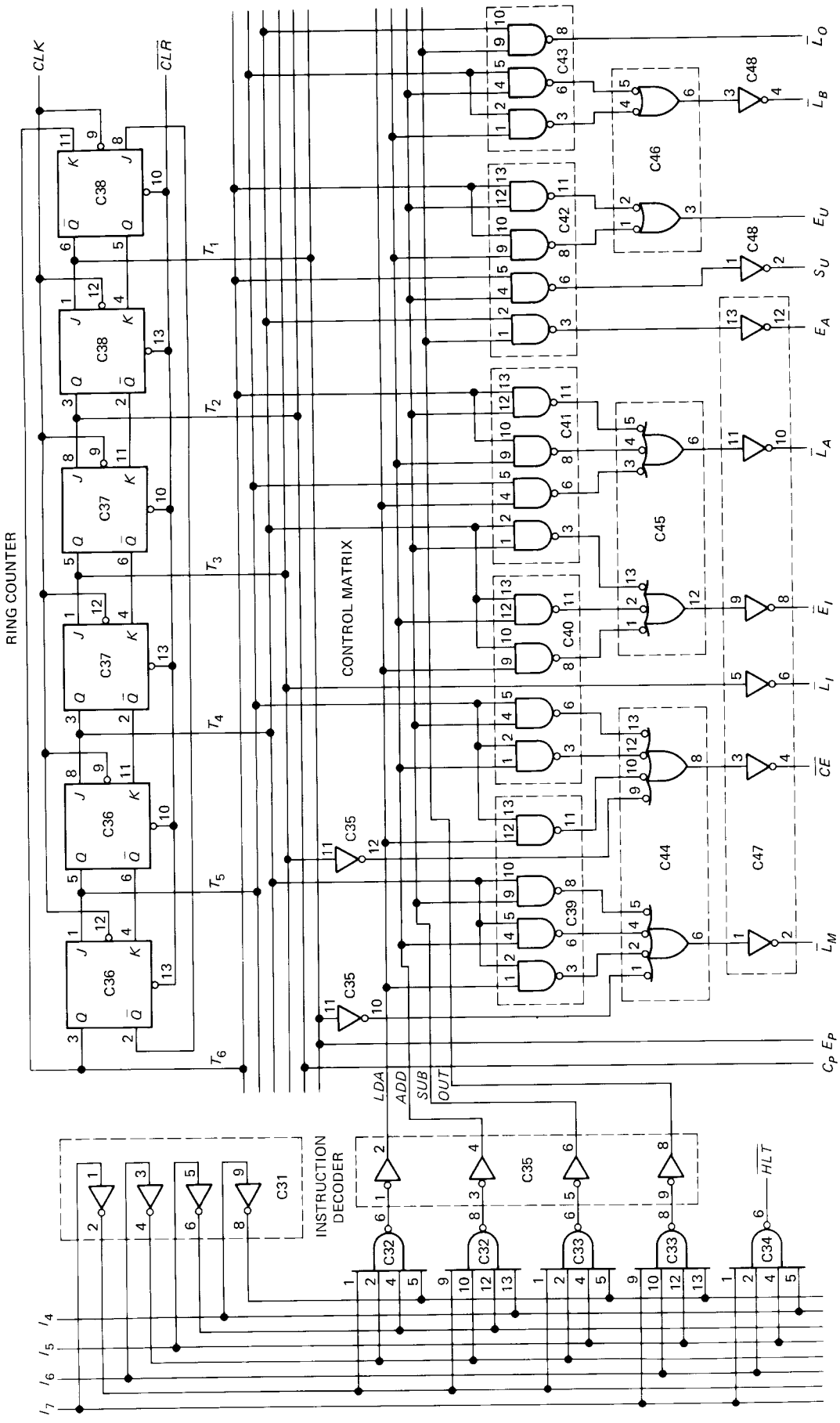


Fig. 10-15 Instruction decoder, ring counter, and control matrix.

the J input of the Q_1 flip-flop (pin 1, C38). Because of this, the T_1 output is initially high.

The *CLK* signal drives an active low input. This means that the negative edge of the *CLK* signal initiates each T state. Half a cycle later, the positive edge of the *CLK* signal produces register loading, as previously described.

Control Matrix

The *LDA*, *ADD*, *SUB*, and *OUT* signals from the instruction decoder drive the *control matrix*, C39 to C48. At the same time, the ring-counter signals, T_1 to T_6, are driving the matrix (a circuit receiving two groups of bits from different sources). The matrix produces **CON**, a 12-bit microinstruction that tells the rest of the computer what to do.

In Fig. 10-15, T_1 goes high, then T_2, then T_3, and so on. Analyze the control matrix and here is what you will find. A high T_1 produces a high E_P and a low $\overline{L}_M$ (address state); a high T_2 results in a high C_P (increment state); and a high T_3 produces a low $\overline{CE}$ and a low $\overline{L}_I$ (memory state). The first three T states, therefore, are always the fetch cycle in SAP-1. In chunked notation, the **CON** words for the fetch cycle are

State	CON	Active Bits
T_1	5E3H	E_P, $\overline{L}_M$
T_2	BE3H	C_P
T_3	263H	$\overline{CE}$, $\overline{L}_I$

During the execution states, T_4 through T_6 go high in succession. At the same time, only one of the decoded signals (*LDA* through *OUT*) is high. Because of this, the matrix automatically steers active bits to the correct output control lines.

For instance, when *LDA* is high, the only enabled 2-input NAND gates are the first, fourth, seventh, and tenth. When T_4 is high, it activates the first and seventh NAND gates, resulting in low $\overline{L}_M$ and low $\overline{E}_I$ (load MAR with address field). When T_5 is high, it activates the fourth and tenth NAND gates, producing a low $\overline{CE}$ and a low $\overline{L}_A$ (load RAM data into accumulator). When T_6 goes high, none of the control bits are active (nop).

You should analyze the action of the control matrix during the execution states of the remaining possibilities: high *ADD*, high *SUB*, and high *OUT*. Then you will agree the control matrix can generate the ADD, SUB, and OUT microinstructions shown in Table 10-5 (SAP-1 microprogram).

Operation

Before each computer run, the operator enters the program and data into the SAP-1 memory. With the program in low memory and the data in high memory, the operator presses and releases the clear button. The *CLK* and $\overline{CLK}$ signals drive the registers and counters. The microinstruction out of the controller-sequencer determines what happens on each positive *CLK* edge.

Each SAP-1 machine cycle begins with a fetch cycle. T_1 is the address state, T_2 is the increment state, and T_3 is the memory state. At the end of the fetch cycle, the instruction is stored in the instruction register. After the instruction field has been decoded, the control matrix automatically generates the correct execution routine. Upon completion of the execution cycle, the ring counter resets and the next machine cycle begins.

The data processing ends when a HLT instruction is loaded into the instruction register.

10-8 MICROPROGRAMMING

The control matrix of Fig. 10-15 is one way to generate the microinstructions needed for each execution cycle. With larger instruction sets, the control matrix becomes very complicated and requires hundreds or even thousands of gates. This is why *hardwired control* (matrix gates soldered together) forced designers to look for an alternative way to produce the control words that run a computer.

Microprogramming is the alternative. The basic idea is to store microinstructions in a ROM rather than produce them with a control matrix. This approach simplifies the problem of building a controller-sequencer.

Storing the Microprogram

By assigning addresses and including the fetch routine, we can come up with the SAP-1 microinstructions shown in Table 10-6. These microinstructions can be stored in a *control ROM* with the fetch routine at addresses 0H to 2H, the LDA routine at addresses 3H to 5H, the ADD routine at 6H to 8H, the SUB routine at 9H to BH, and the OUT routine at CH to EH.

To access any routine, we need to supply the correct addresses. For instance, to get the ADD routine, we need to supply addresses 6H, 7H, and 8H. To get the OUT routine, we supply addresses CH, DH, and EH. Therefore, accessing any routine requires three steps:

1. Knowing the starting address of the routine
2. Stepping through the routine addresses
3. Applying the addresses to the control ROM.

Address ROM

Figure 10-16 shows how to microprogram the SAP-1 computer. It has an *address* ROM, a *presettable* counter, and a *control* ROM. The address ROM contains the starting addresses of each routine in Table 10-6. In other words,

TABLE 10-6. SAP-1 CONTROL ROM

Address	Contents†	Routine	Active
0H	5E3H	Fetch	E_P, $\overline{L}_M$
1H	BE3H		C_P
2H	263H		$\overline{CE}$, $\overline{L}_I$
3H	1A3H	LDA	$\overline{L}_M$, $\overline{E}_I$
4H	2C3H		$\overline{CE}$, $\overline{L}_A$
5H	3E3H		None
6H	1A3H	ADD	$\overline{L}_M$, $\overline{E}_I$
7H	2E1H		$\overline{CE}$, $\overline{L}_B$
8H	3C7H		$\overline{L}_A$, E_U
9H	1A3H	SUB	$\overline{L}_M$, $\overline{E}_I$
AH	2E1H		$\overline{CE}$, $\overline{L}_B$
BH	3CFH		$\overline{L}_A$, S_U, E_U
CH	3F2H	OUT	E_A, $\overline{L}_O$
DH	3E3H		None
EH	3E3H		None
FH	X	X	Not used

† **CON** = $C_PE_P\overline{L}_M\overline{CE}$ $\overline{L}_I\overline{E}_I\overline{L}_AE_A$ $S_UE_U\overline{L}_B\overline{L}_O$.

TABLE 10-7. ADDRESS ROM

Address	Contents	Routine
0000	0011	LDA
0001	0110	ADD
0010	1001	SUB
0011	XXXX	None
0100	XXXX	None
0101	XXXX	None
0110	XXXX	None
0111	XXXX	None
1000	XXXX	None
1001	XXXX	None
1010	XXXX	None
1011	XXXX	None
1100	XXXX	None
1101	XXXX	None
1110	1100	OUT
1111	XXXX	None

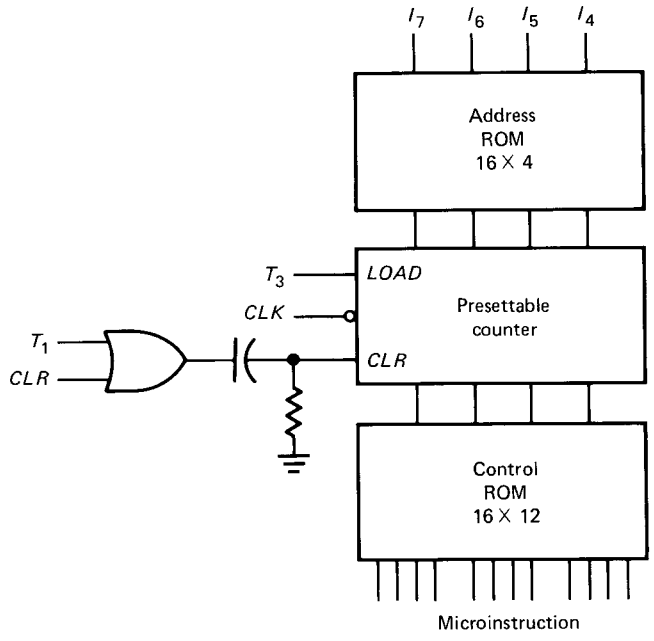


Fig. 10-16 Microprogrammed control of SAP-1.

the address ROM contains the data listed in Table 10-7. As shown, the starting address of the LDA routine is 0011, the starting address of the ADD routine is 0110, and so on.

When the op-code bits $I_7I_6I_5I_4$ drive the address ROM, the starting address is generated. For instance, if the ADD instruction is being executed, $I_7I_6I_5I_4$ is 0001. This is the input to the address ROM; the output of this ROM is 0110.

Presettable Counter

When T_3 is high, the load input of the *presettable counter* is high and the counter loads the starting address from the address ROM. During the other T states, the counter counts.

Initially, a high *CLR* signal from the clear-start debouncer is differentiated to get a narrow positive spike. This resets the counter. When the computer run begins, the counter output is 0000 during the T_1 state, 0001 during the T_2 state, and 0010 during the T_3 state. Every fetch cycle is the same because 0000, 0001, and 0010 come out of the counter during states T_1, T_2, and T_3.

The op code in the instruction register controls the execution cycle. If an ADD instruction has been fetched, the $I_7I_6I_5I_4$ bits are 0001. These op-code bits drive the address ROM, producing an output of 0110 (Table 10-7). This starting address is the input to the presettable counter. When T_3 is high, the next negative clock edge loads 0110 into the presettable counter. The counter is now preset, and counting can resume at the starting address of the ADD routine. The counter output is 0110 during the T_4 state, 0111 during the T_5 state, and 1000 during the T_6 state.

When the T_1 state begins, the leading edge of the T_1 signal is differentiated to produce a narrow positive spike which resets the counter to 0000, the starting address of the fetch routine. A new machine cycle then begins.

Control ROM

The control ROM stores the SAP-1 microinstructions. During the fetch cycle, it receives addresses 0000, 0001, and 0010. Therefore, its outputs are

5E3H
BE3H
263H

These microinstructions, listed in Table 10-6, produce the address state, increment state, and memory state.

If an ADD instruction is being executed, the control ROM receives addresses 0110, 0111, and 1000 during the execution cycle. Its outputs are

1A3H
2E1H
3C7H

These microinstructions carry out the addition as previously discussed.

For another example, suppose the OUT instruction is being executed. Then the op code is 1110 and the starting address is 1100 (Table 10-7). During the execution cycle, the counter output is 1100, 1101, and 1110. The output of the control ROM is 3F2H, 3E3H, and 3E3H (Table 10-6). This routine transfers the accumulator contents to the output port.

Variable Machine Cycle

The microinstruction 3E3H in Table 10-6 is a nop. It occurs once in the LDA routine and twice in the OUT routine. These nops are used in SAP-1 to get a *fixed machine cycle* for all instructions. In other words, each machine cycle takes exactly six T states, no matter what the instruction. In some computers a fixed machine cycle is an advantage. But when speed is important, the nops are a waste of time and can be eliminated.

One way to speed up the operation of SAP-1 is to skip any T state with a nop. By redesigning the circuit of Fig. 10-16 we can eliminate the nop states. This will shorten the machine cycle of the LDA instruction to five states (T_1, T_2, T_3, T_4, and T_5). It also shortens the machine cycle of the OUT instruction to four T states (T_1, T_2, T_3, and T_4).

Figure 10-17 shows one way to get a *variable machine cycle*. With an LDA instruction, the action is the same as before during the T_1 to T_5 states. When the T_6 state begins, the control ROM produces an output of 3E3H (the nop microinstruction). The NAND gate detects this nop instantly and produces a low output signal $\overline{NOP}$. $\overline{NOP}$ is fed back to the ring counter through an AND gate, as shown in Fig. 10-18. This resets the ring counter to the T_1 state, and a new machine cycle begins. This reduces the machine cycle of the LDA instruction from six states to five.

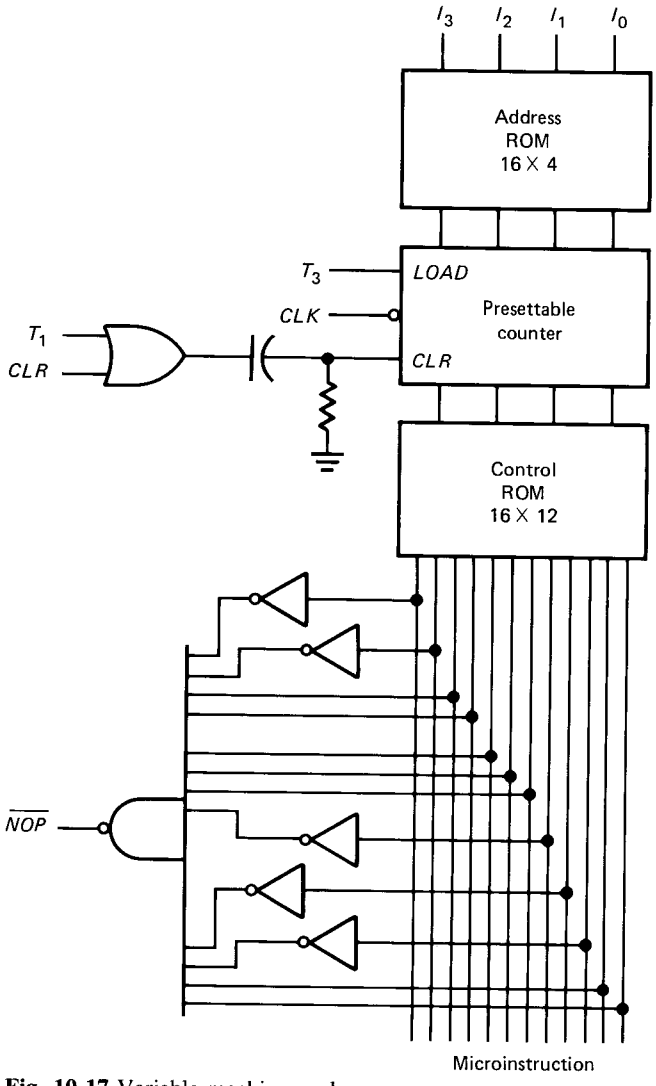


Fig. 10-17 Variable machine cycle.

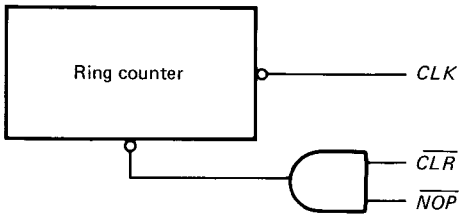


Fig. 10-18

With the OUT instruction, the first nop occurs in the T_5 state. In this case, just after the T_5 state begins, the control ROM produces an output of 3E3H, which is detected by the NAND gate. The low $\overline{NOP}$ signal then resets the ring counter to the T_1 state. In this way, we have reduced the machine cycle of the OUT instruction from six states to four.

Variable machine cycles are commonly used with microprocessors. In the 8085, for example, the machine cycles take from two to six T states because all unwanted nop states are ignored.

Advantages

One advantage of microprogramming is the elimination of the instruction decoder and control matrix; both of these become very complicated for larger instruction sets. In other words, it's a lot easier to store microinstructions in a ROM than it is to wire an instruction decoder and control matrix.

Furthermore, once you wire an instruction decoder and control matrix, the only way you can change the instruction set is by disconnecting and rewiring. This is not necessary with microprogrammed control; all you have to do is change the control ROM and the starting-address ROM. This is a big advantage if you are trying to upgrade equipment sold earlier.

Summary

In conclusion, most modern microprocessors use microprogrammed control instead of hardwired control. The microprogramming tables and circuits are more complicated than those for SAP-1, but the idea is the same. Microinstructions are stored in a control ROM and accessed by applying the address of the desired microinstruction.

GLOSSARY

address state The T_1 state. During this state, the address in the program counter is transferred to the MAR.
accumulator The place where answers to arithmetic and logic operations are accumulated. Sometimes called the A register.
assembly language The mnemonics used in writing a program.
B register An auxiliary register that stores the data to be added or subtracted from the accumulator.
fetch cycle The first part of the instruction cycle. During the fetch cycle, the address is sent to the memory, the program counter is incremented, and the instruction is transferred from the memory to the instruction register.
increment state The T_2 state. During this state, the program counter is incremented.
instruction cycle All the states needed to fetch and execute an instruction.
instruction register The register that receives the instruction from the memory.
instruction set The instructions a computer responds to.
LDA Mnemonic for load the accumulator.
machine cycle All the states generated by the ring counter.
machine language The strings of 0s and 1s used in a program.
macroinstruction One of the instructions in the instruction set.
MAR Memory address register. This register receives the address of the data to be accessed in memory. The MAR supplies this address to the memory.
memory-reference instruction An instruction that calls for a second memory operation to access data.
memory state The T_3 state. During this state, the instruction in the memory is transferred to the instruction register.
microinstruction A control word out of the controller-sequencer. The smallest step in the data processing.
nop No operation. A state during which nothing happens.
output register The register that receives processed data from the accumulator and drives the output display of SAP-1. Also called an output port.
object program A program written in machine language.
op code Operation code. That part of the instruction which tells the computer what operation to perform.
program counter A register that counts in binary. Its contents are the address of the next instruction to be fetched from the memory.
RAM Random-access memory. A better name is read-write memory. The RAM stores the program and data needed for a computer run.
source program A program written in mnemonics.

SELF-TESTING REVIEW

Read each of the following and provide the missing words. Answers appear at the beginning of the next question.

1. The _____ counter, which is part of the control unit, counts from 0000 to 1111. It sends to the memory the _____ of the next instruction.
2. (*program, address*) The MAR, or _____ register, latches the address from the program counter. A bit later, the MAR applies this address to the _____, where a read operation is performed.
3. (*memory-address, RAM*) The instruction register is

part of the control unit. The contents of the _____ register are split into two nibbles. The upper nibble goes to the _____.

4. (*instruction, controller-sequencer*) The controller-sequencer produces a 12-bit word that controls the rest of the computer. The 12 wires carrying this _____ word are called the control _____.
5. (*control, bus*) The _____ is a buffer register that stores sums or differences. Its two-state output goes to the adder-subtracter. The _____ produces the sum when S_U is low and the difference when S_U is high. The output register is sometimes called an output _____.
6. (*accumulator, adder-subtracter, port*) The SAP-1 _____ set is LDA, ADD, SUB, OUT, and HLT. LDA, ADD, and SUB are called _____ instructions because they use data stored in the memory.
7. (*instruction, memory-reference*) The 8080 was the first widely used microprocessor. The _____ is an enhanced version of the 8080 with essentially the same instruction set.
8. (*8085*) LDA, ADD, SUB, OUT, and HLT are coded as 4-bit strings of 0s and 1s. This code is called the _____ code. _____ language uses mnemonics when writing a program. _____ language uses strings of 0s and 1s.
9. (*op, Assembly, Machine*) SAP-1 has _____ *T* states, periods during which register contents change. The ring counter, or _____ counter, produces these *T* states. These six *T* states represent one machine cycle. In SAP-1 the instruction cycle has only one machine cycle. In microprocessors like the 8080 and the 8085, the _____ cycle may have from one to five machine cycles.
10. (*six, state, instruction*) The controller-sequencer sends out control words, one during each *T* state or clock cycle. Each control word is called a _____. Instructions like LDA, ADD, SUB, etc. are called _____. Each SAP-1 macroinstruction is made up of three _____.
11. (*microinstruction, macroinstructions, microinstructions*) With larger instruction sets, the control matrix becomes very complicated. This is why hardwired control is being replaced by _____. The basic idea is to store the _____ in a control ROM.
12. (*microprogramming, microinstructions*) SAP-1 uses a fixed machine cycle for all instructions. In other words, each machine cycle takes exactly six *T* states. Microprocessors like the 8085 have variable machine cycles because all unwanted nop states are eliminated.

PROBLEMS

10-1. Write a SAP-1 program using mnemonics (similar to Example 10-1) that will display the result of

$$5 + 4 - 6$$

Use addresses DH, EH, and FH for the data.

10-2. Convert the assembly language of Prob. 10-1 into SAP-1 machine language. Show the answer in binary form and in hexadecimal form.

10-3. Write an assembly-language program that performs this operation:

$$8 + 4 - 3 + 5 - 2$$

Use addresses BH to FH for the data.

10-4. Convert the program and data of Prob. 10-3 into machine language. Express the result in both binary and hexadecimal form.

10-5. Figure 10-19 shows the timing diagram for the ADD instruction. Draw the timing diagram for the SUB instruction.

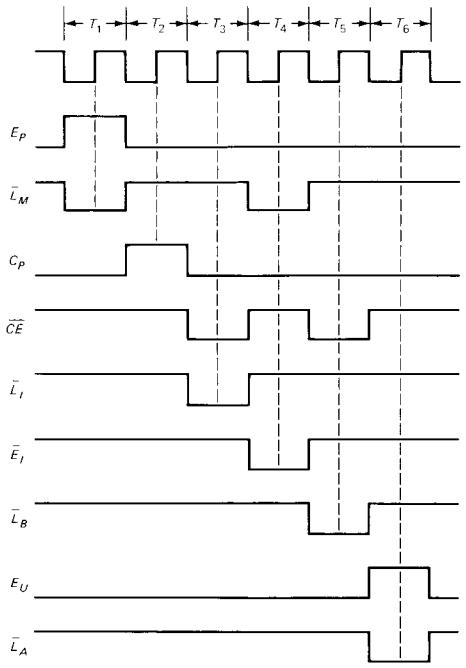
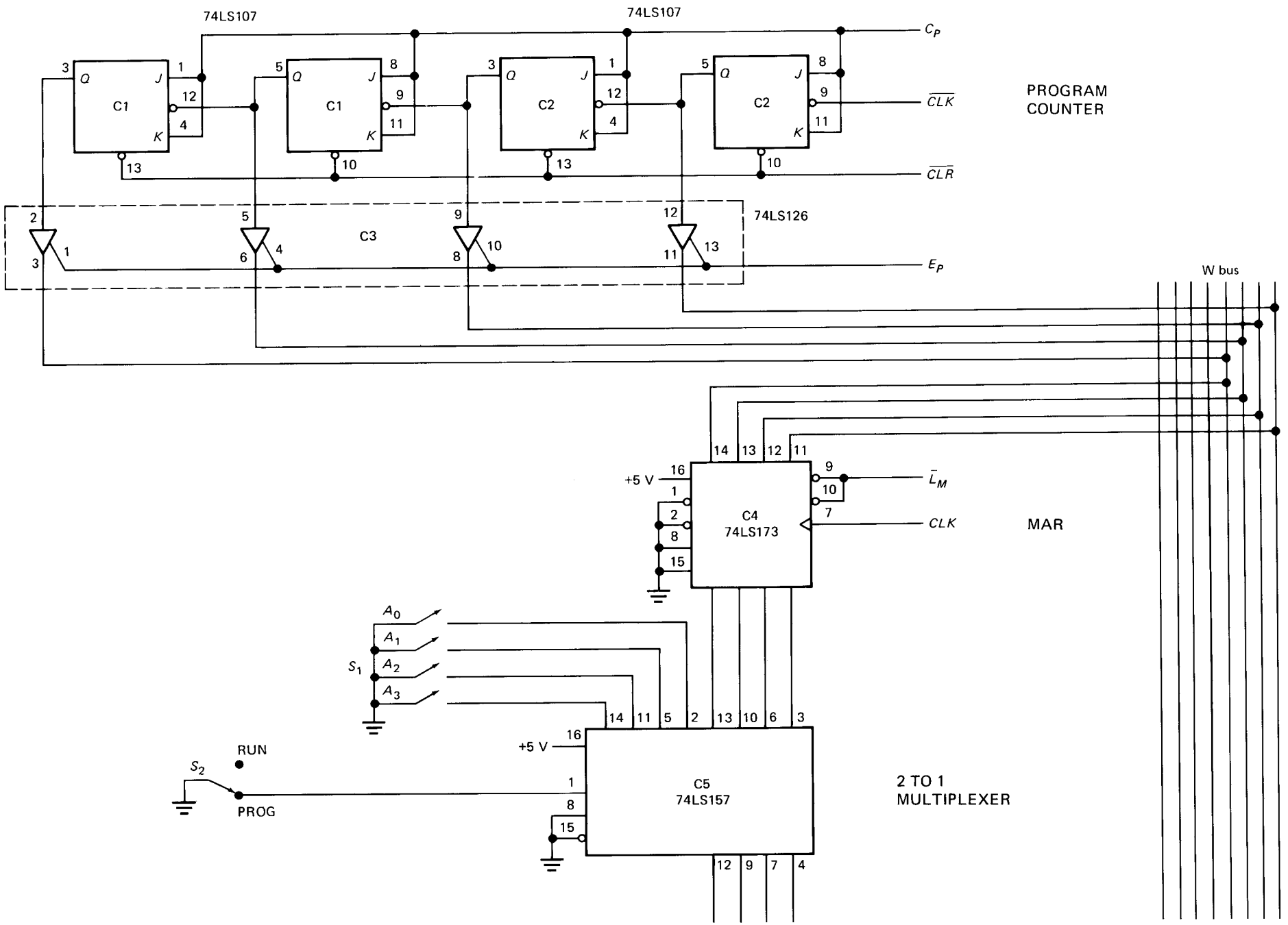


Fig. 10-19


74LS107
74LS107
C1
C1
C2
C2
C3
74LS126
C_P
$\overline{CLK}$
$\overline{CLR}$
E_P
PROGRAM
COUNTER
W bus
C4
74LS173
+5 V
$\overline{L}_M$
CLK
MAR
A_0
A_1
A_2
A_3
S_1
RUN
S_2
PROG
C5
74LS157
+5 V
2 TO 1
MULTIPLEXER

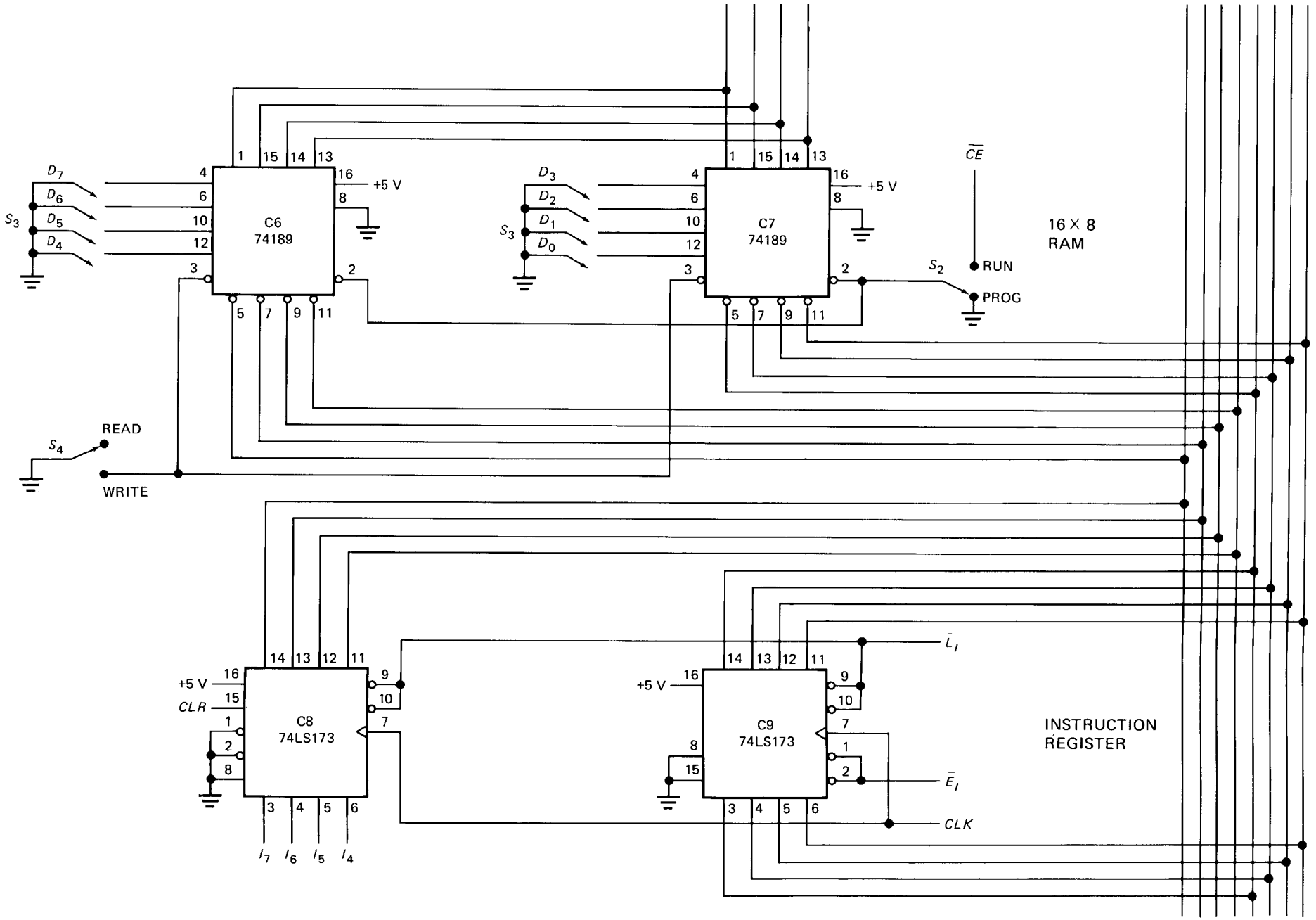
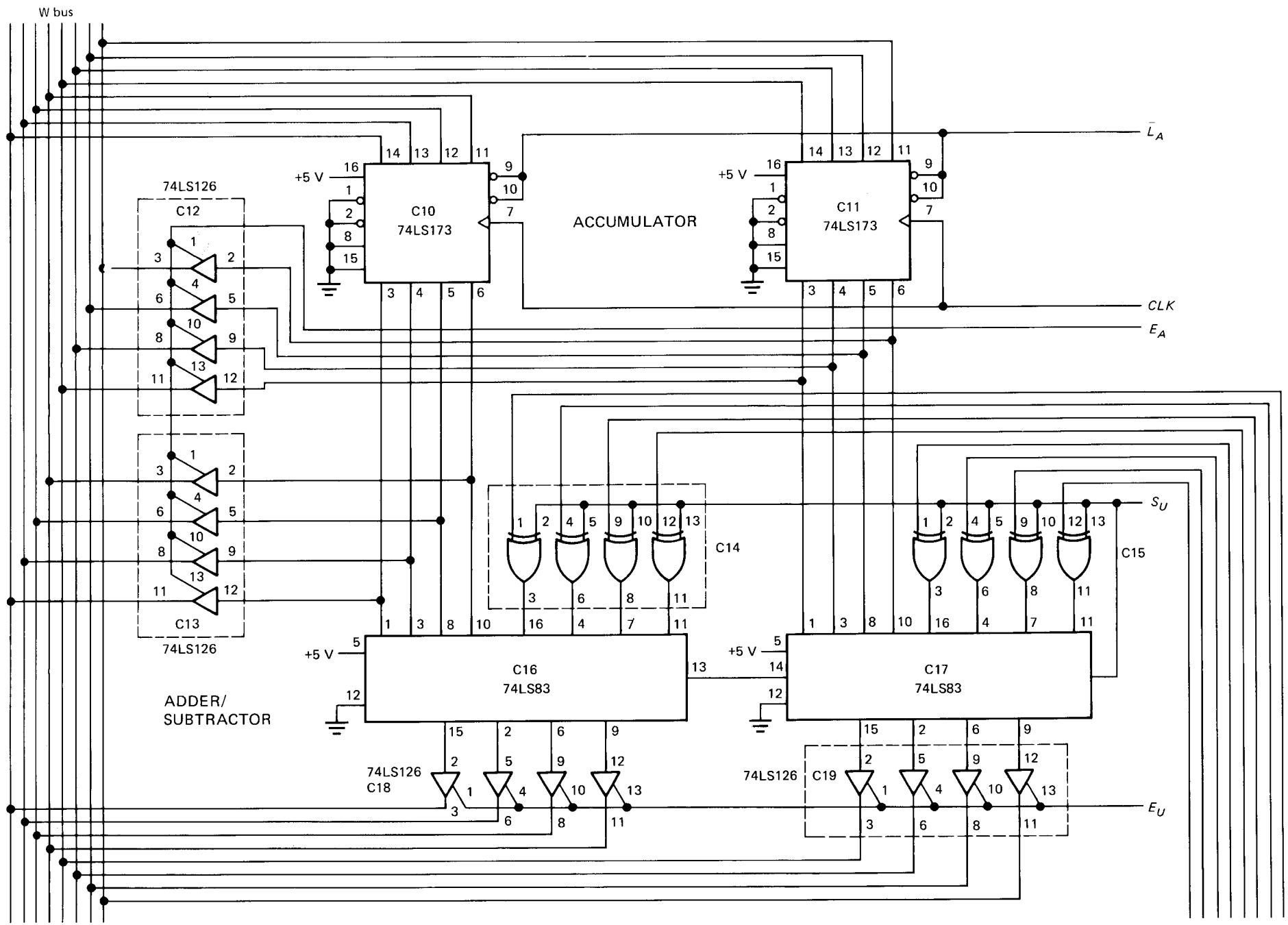

C6
74189
C7
74189
+5 V
D_7
D_6
D_5
D_4
D_3
D_2
D_1
D_0
S_3
$\overline{CE}$
RUN
PROG
S_2
16 X 8
RAM
READ
WRITE
S_4
C8
74LS173
C9
74LS173
CLR
$\overline{L}_I$
$\overline{E}_I$
CLK
INSTRUCTION
REGISTER
I_7 I_6 I_5 I_4

Fig. 10-20


W bus
74LS126
C12
C13
74LS126
ADDER/
SUBTRACTOR
+5 V
C10
74LS173
ACCUMULATOR
C11
74LS173
$\overline{L}_A$
CLK
E_A
C14
C15
S_U
C16
74LS83
C17
74LS83
74LS126
C18
74LS126
C19
E_U

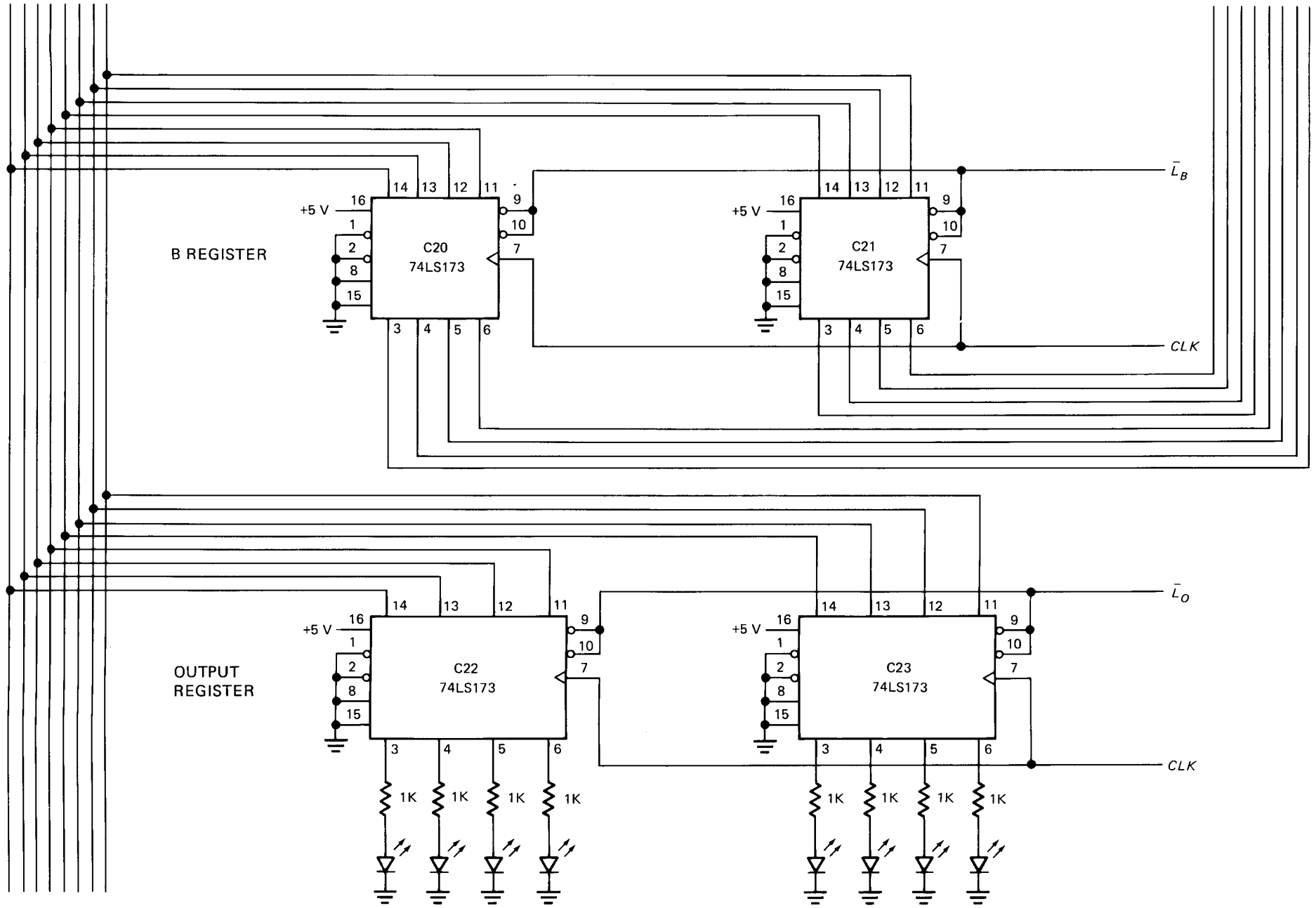

B REGISTER
C20
74LS173
C21
74LS173
+5 V
$\overline{L}_B$
CLK
OUTPUT
REGISTER
C22
74LS173
C23
74LS173
$\overline{L}_O$
1K

Fig. 10-21

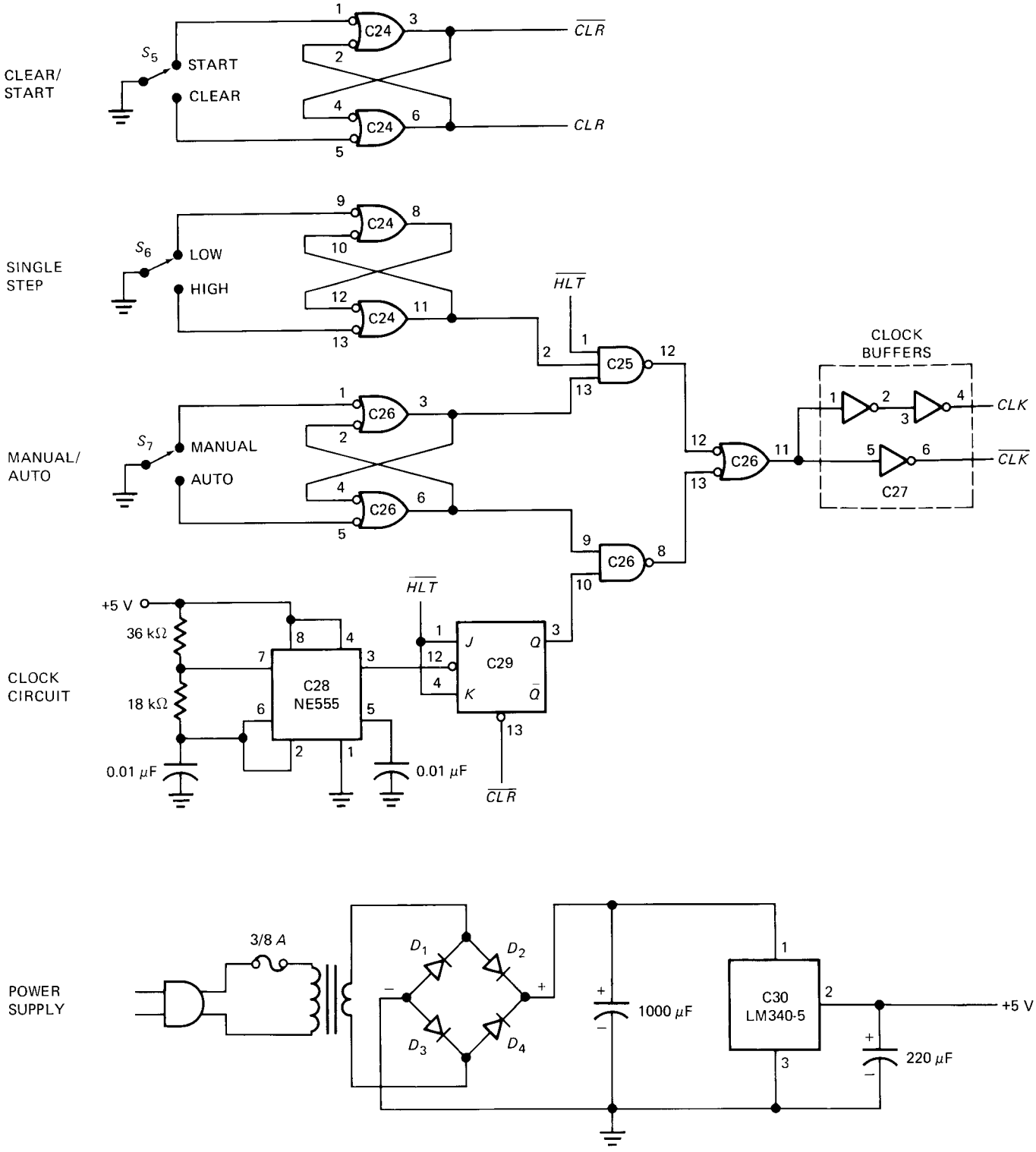

CLEAR/
START
S_5
START
CLEAR
C24
$\overline{CLR}$
CLR
SINGLE
STEP
S_6
LOW
HIGH
$\overline{HLT}$
C25
CLOCK
BUFFERS
MANUAL/
AUTO
S_7
MANUAL
AUTO
C26
C27
CLK
$\overline{CLK}$
CLOCK
CIRCUIT
+5 V
36 kΩ
18 kΩ
0.01 μF
C28
NE555
C29
J
K
Q
$\overline{Q}$
POWER
SUPPLY
3/8 A
D_1
D_2
D_3
D_4
1000 μF
C30
LM340-5
220 μF
+5 V

Fig. 10-22

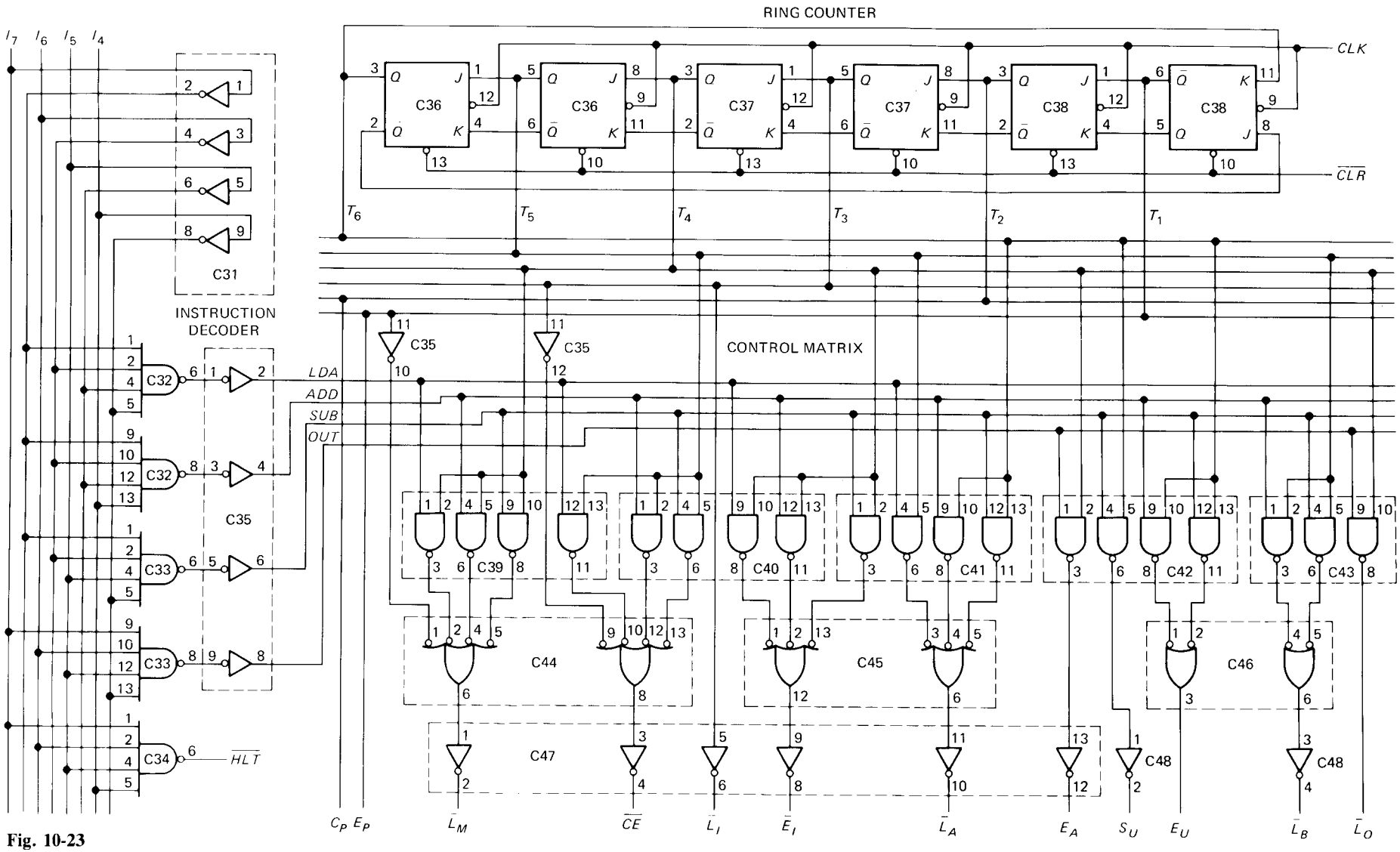

RING COUNTER
CLK
$\overline{CLR}$
C36
C37
C38
T_6
T_5
T_4
T_3
T_2
T_1
I_7 I_6 I_5 I_4
C31
INSTRUCTION DECODER
C35
CONTROL MATRIX
LDA
ADD
SUB
OUT
C32
C33
C34
$\overline{HLT}$
C39
C40
C41
C42
C43
C44
C45
C46
C47
C48
C_P E_P
$\overline{L_M}$
$\overline{CE}$
$\overline{L_I}$
$\overline{E_I}$
$\overline{L_A}$
E_A
S_U
E_U
$\overline{L_B}$
$\overline{L_O}$

Fig. 10-23

10-6. Suppose an 8085 uses a clock frequency of 3 MHz. The ADD instruction of an 8085 takes four T states to fetch and execute. How long is this?

10-7. What are the SAP-1 microinstructions for the LDA routine? For the SUB routine? Express the answers in binary and hexadecimal form.

10-8. Suppose we want to transfer the contents of the accumulator to the B register. This requires a new microinstruction. What is this microinstruction? Express your answer in hexadecimal and binary form.

10-9. Look at Fig. 10-20 and answer the following questions:

a. Are the contents of the program counter changed on the positive or negative edge of the $\overline{CLK}$ signal? At this instant, is the CLK signal on its rising or falling edge?

b. To increment the program counter, does C_P have to be low or high?

c. To clear the program counter, does $\overline{CLR}$ have to be low or high?

d. To place the contents of the program counter on the W bus, should E_P be low or high?

10-10. Refer to Fig. 10-21:

a. If $\overline{L}_A$ is high, what happens to the accumulator contents on the next positive clock edge?

b. If **A** = 0010 1100 and **B** = 1100 1110, what is on the W bus if E_A is high?

c. If **A** = 0000 1111, **B** = 0000 0001, and S_U = 1, what is on the W bus when E_U is high?

10-11. Answer the following questions for Fig. 10-22:

a. With S_5 in the CLEAR position, is the $\overline{CLR}$ output low or high?

b. With S_6 in the LOW position, is the output low or high for pin 11, C24?

c. To have a clock signal at pin 3 of C29, should $\overline{HLT}$ be low or high?

10-12. Refer to Fig. 10-23 to answer the following:

a. If $I_7I_6I_5I_4$ = 1110, only one of the output pins in C35 is high. Which pin is this? (Disregard pins 10 and 12.)

b. $\overline{CLR}$ goes low. Which is the timing signal (T_1 to T_6) that goes high?

c. LDA and T_5 are high. Is the voltage low or high at pin 6, C45?

d. ADD and T_4 are high. Is the signal low or high at pin 12, C45?