Dr. Apurva Muley (Guest Lecturer)

School of Studies in Physics, Vikram University, Ujjain

Lecture for M. Sc. Physics IV Semester students

Paper-IV: Microprocessor

Unit-3 SAP-I

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

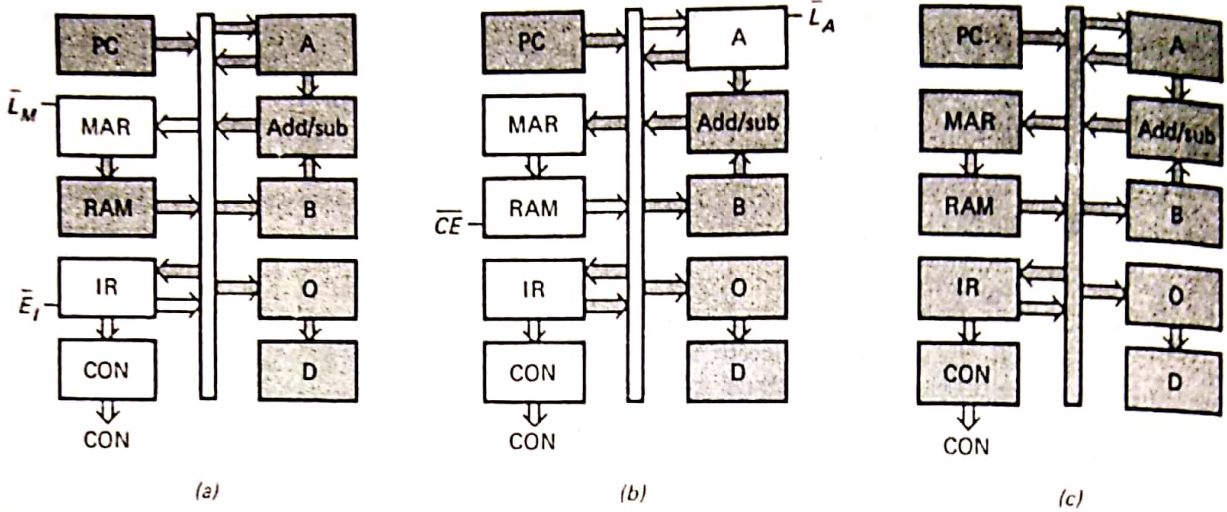

Fig. 10-4 LDA routine: (*a*) T_4 state; (*b*) T_5 state; (*c*) T_6 state.

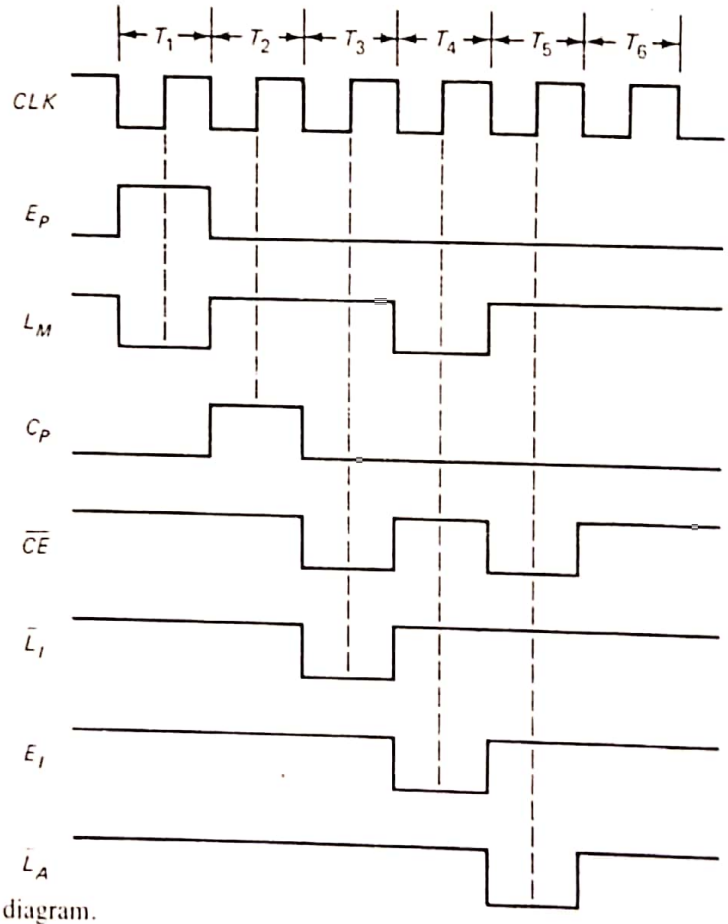

Fig. 10-5 Fetch and LDA timing diagram.

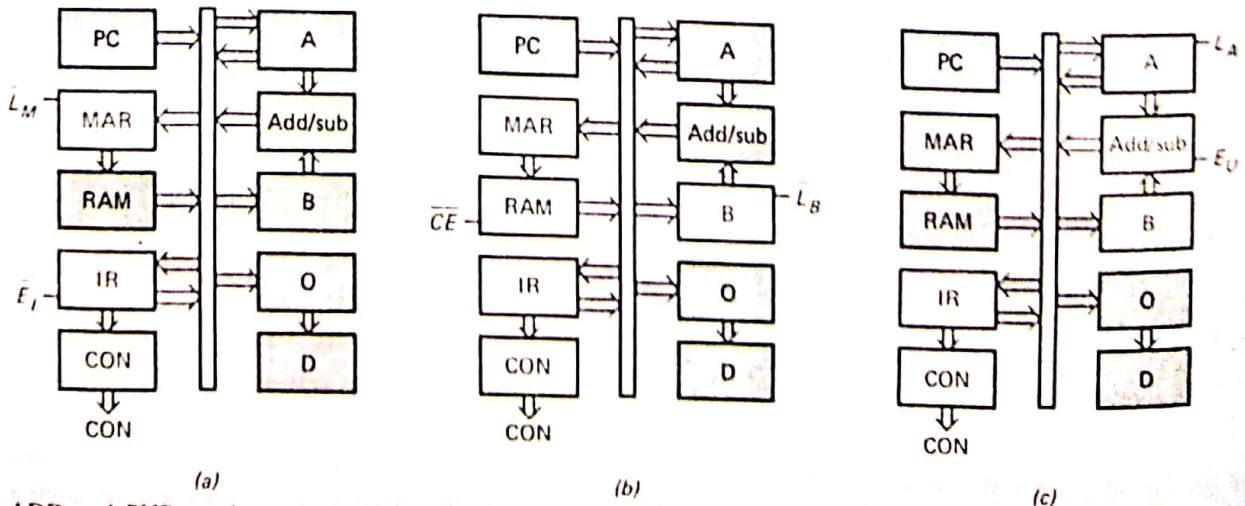

Fig. 10-6 ADD and SUB routines: (*a*) T_4 state; (*b*) T_5 state; (*c*) T_6 state.

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.

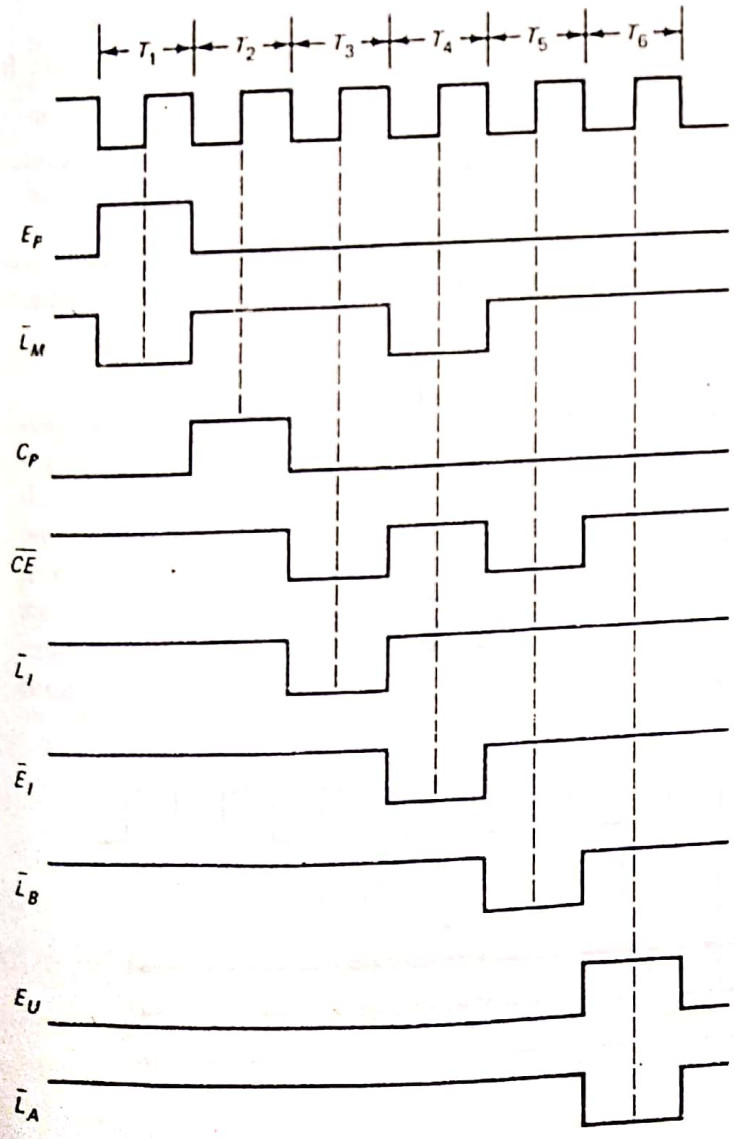

Fig. 10-7 Fetch and ADD timing diagram.

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

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

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.

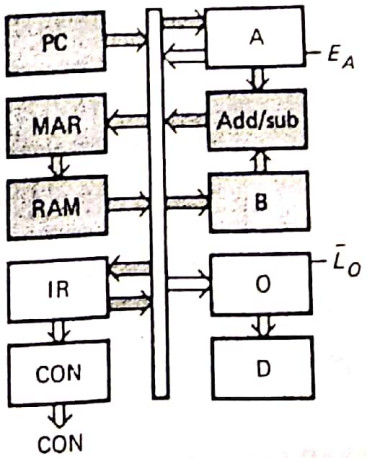

Fig. 10-8 T_4 state of OUT instruction.

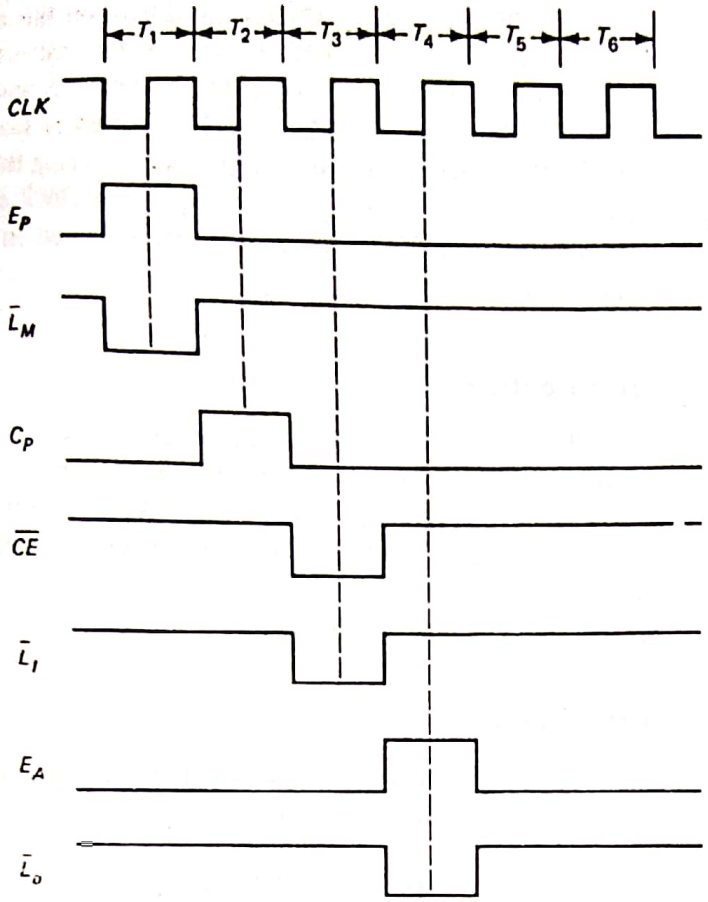

Fig. 10-9 Fetch and OUT timing diagram.

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.

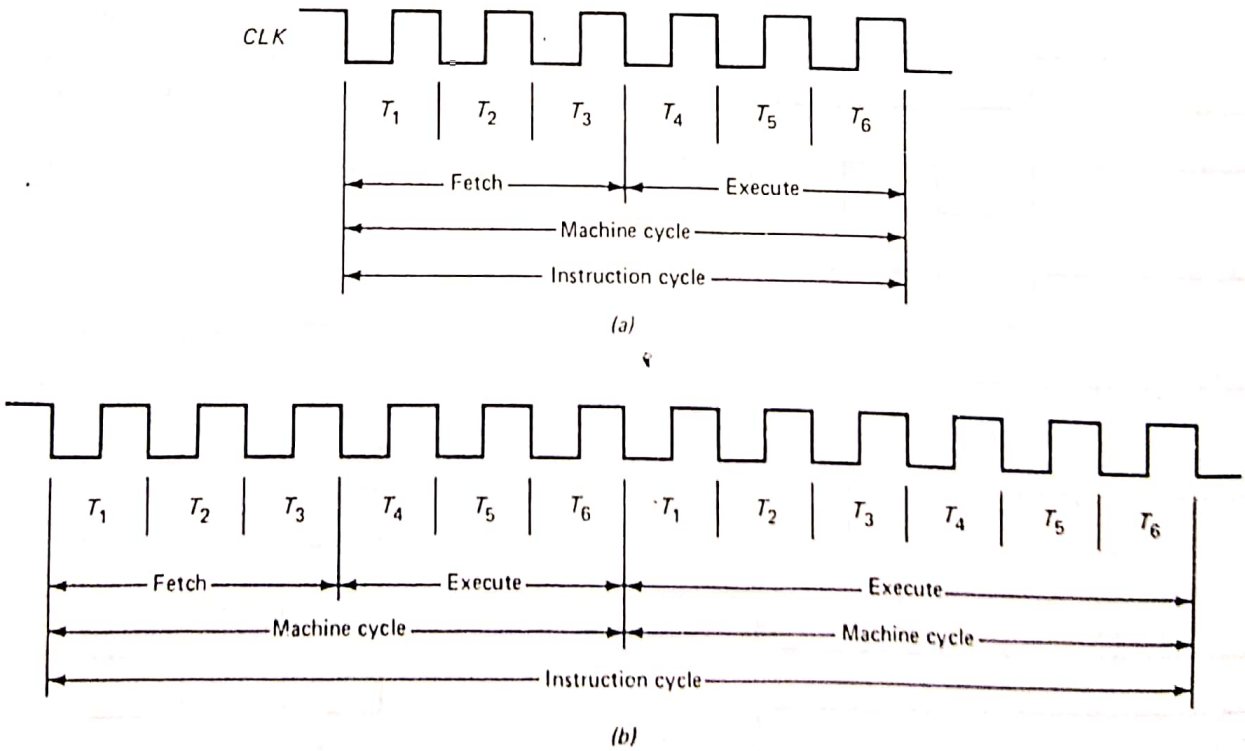

Fig. 10-10 (*a*) SAP-1 instruction cycle; (*b*) instruction cycle with two machine cycles.

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.

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

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

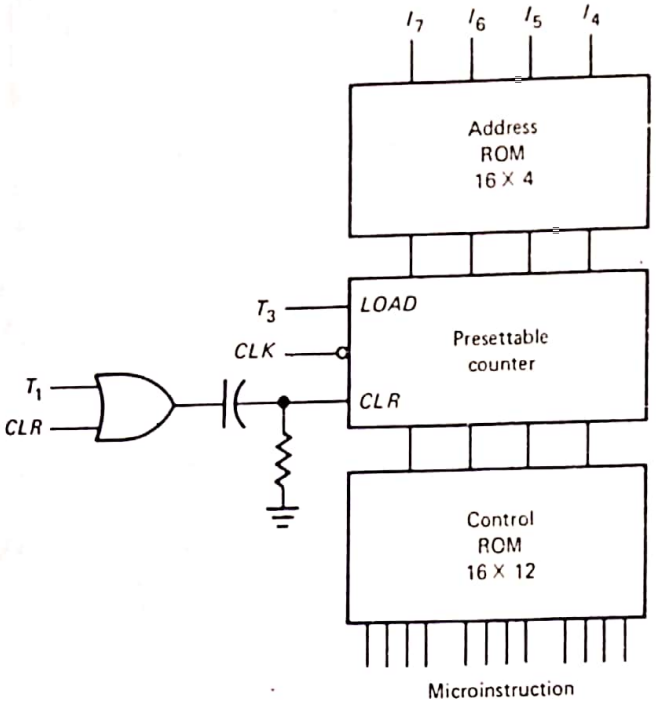

Fig. 10-16 Microprogrammed control of SAP-1.

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.

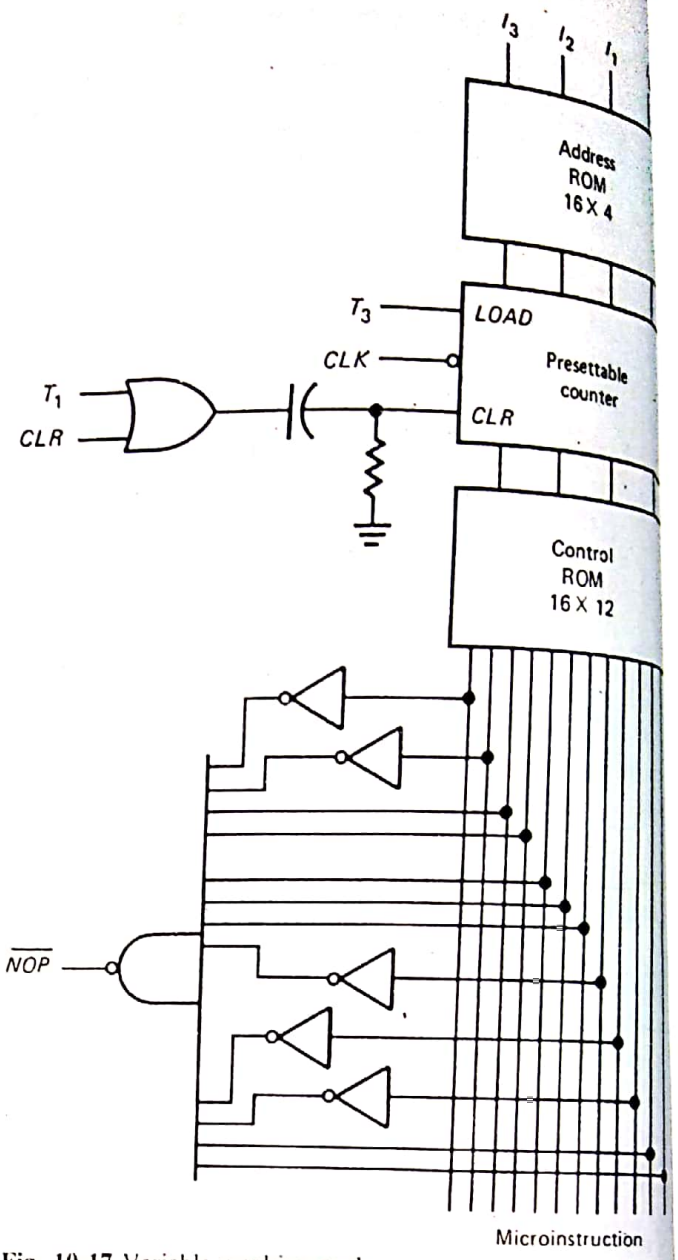

Fig. 10-17 Variable machine cycle.

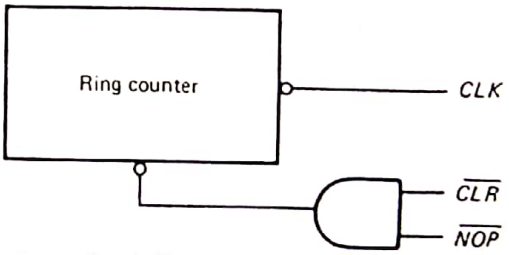

Fig. 10-18

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.