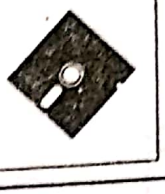
PART 2

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 × 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.

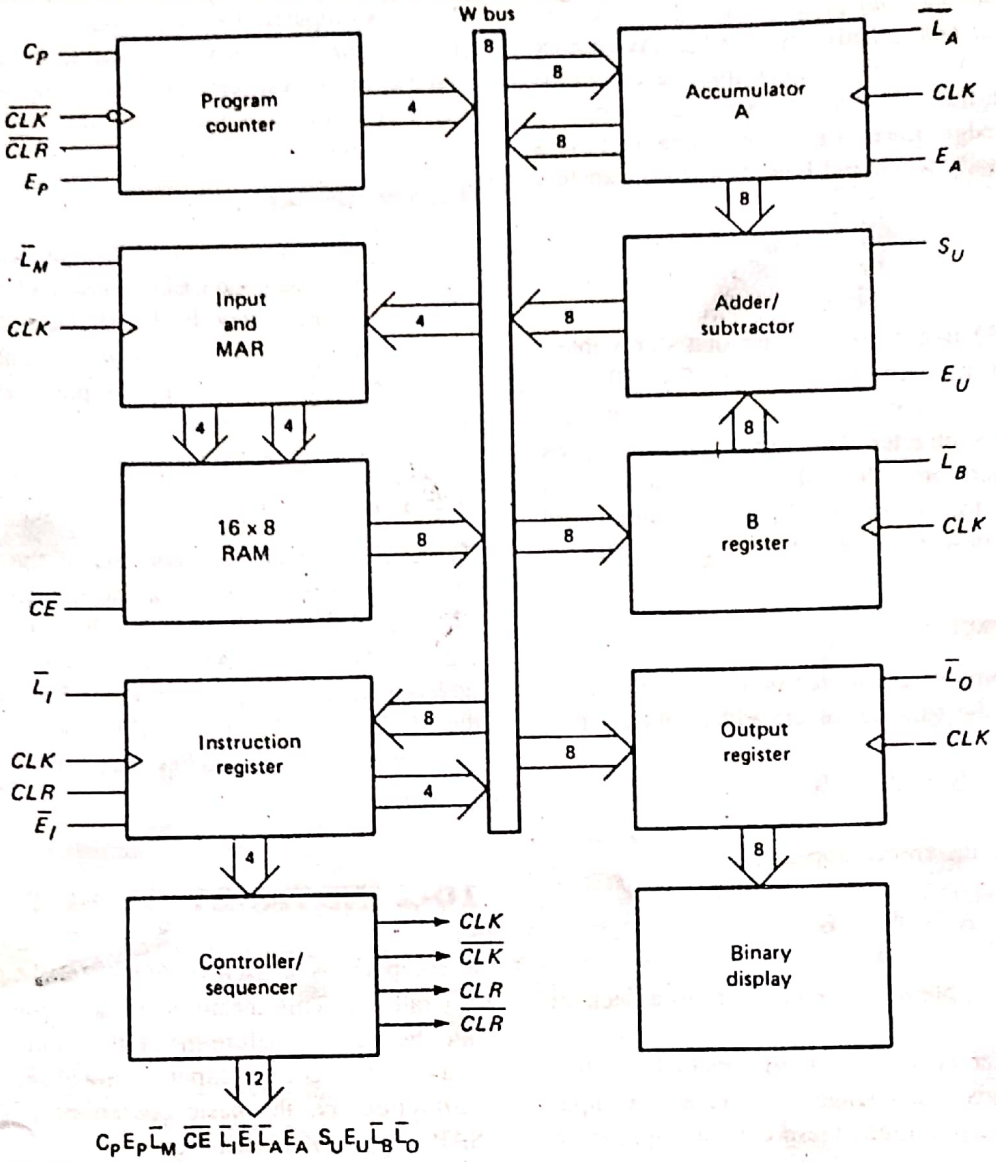


Fig. 10-1 SAP-1 architecture.

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

$$\text{CON} = C_P E_P \overline{L_M} \overline{CE}\ \ \overline{L_I}\overline{E_I}\overline{L_A}E_A\ \ S_U E_U \overline{L_B}\overline{L_O}$$

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

$$S = A + B$$

When S_U is high, the difference appears:

$$A = A + B'$$

(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 $\overline{L_O}$ 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

$$A = 1111\ 0000$$

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

$$A = 01H$$

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

$$A = 01H + 02H = 03H$$

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

$$A = 03H + 03H = 06H$$

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

$$A = 06H - 04H = 02H$$

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

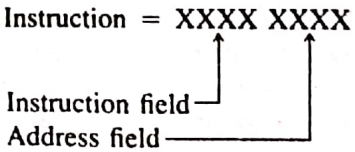
This program is in assembly language as it now stands. To get it into machine language, we translate it to 0s and 1s as follows:

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

$$T = T_6T_5T_4T_3T_2T_1$$

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

$$T = 000001$$

Successive clock pulses produce ring words of

$$T = 000010$$
$$T = 000100$$
$$T = 001000$$
$$T = 010000$$
$$T = 100000$$

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

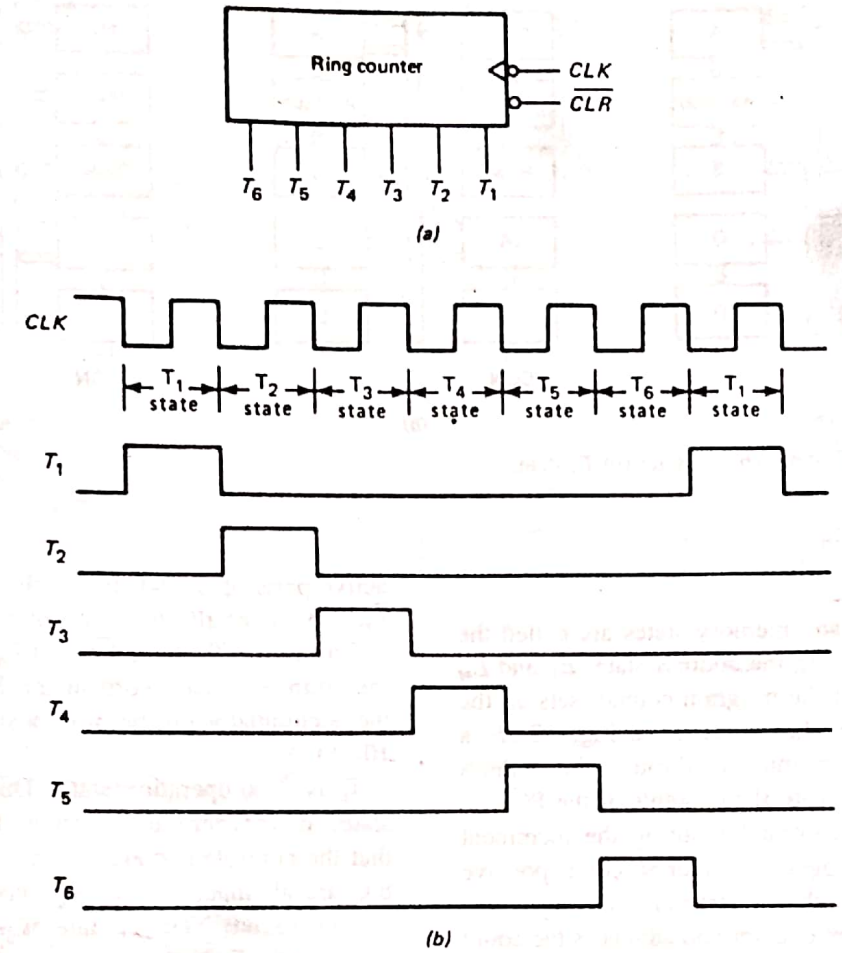


Fig. 10-2 Ring counter: (*a*) symbol; (*b*) clock and timing signals.

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

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

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

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

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

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

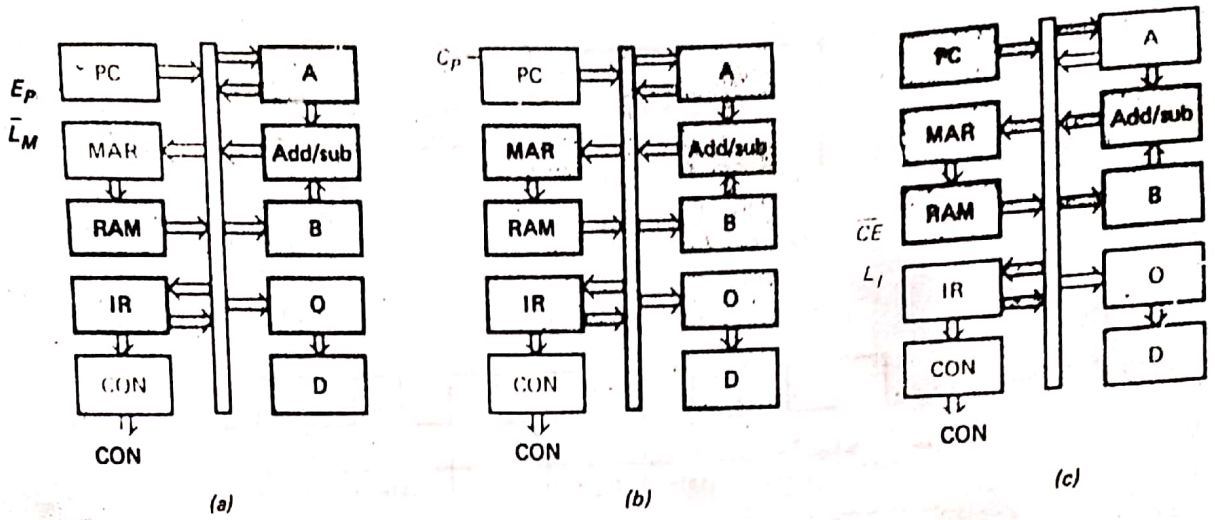


Fig. 10-3 Fetch cycle: (*a*) T_1 state; (*b*) T_2 state; (*c*) T_3 state.

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:

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

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:

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

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

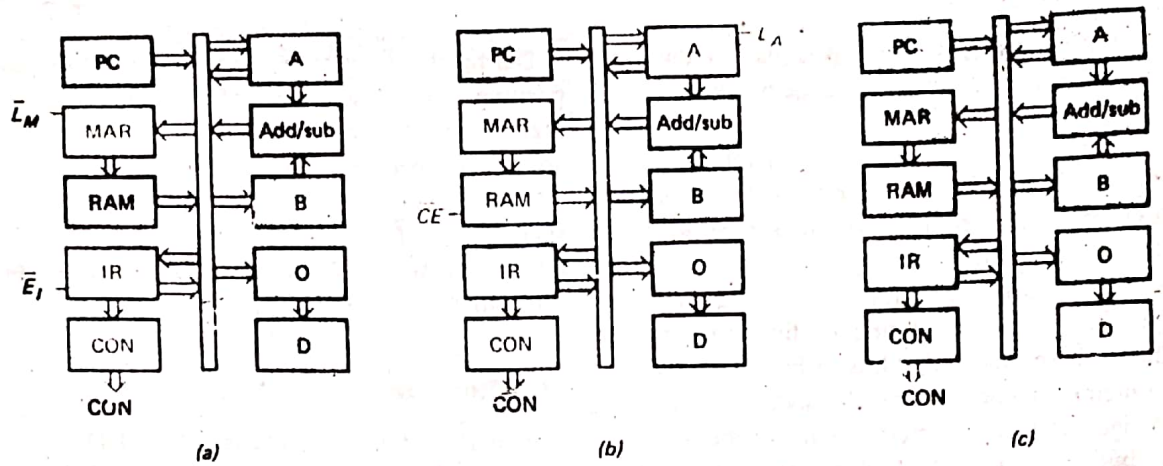


Fig. 10-4 LDA routine: (*a*) T_4 state; (*b*) T_5 state; (*c*) T_6 state.

T_1: E_P, $\bar{L}_M$
T_2: C_P
T_3: $\overline{CE}$, $\bar{L}_I$
T_4: $\bar{E}_I$, $\bar{L}_M$
T_5: $\overline{CE}$, $\bar{L}_A$
T_6: NOP

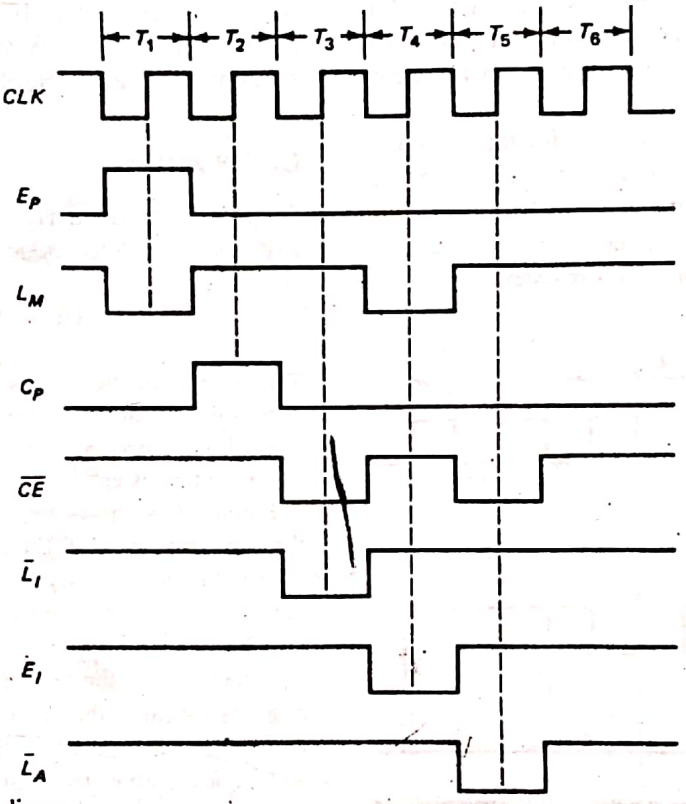


Fig. 10-5 Fetch and LDA timing diagram.

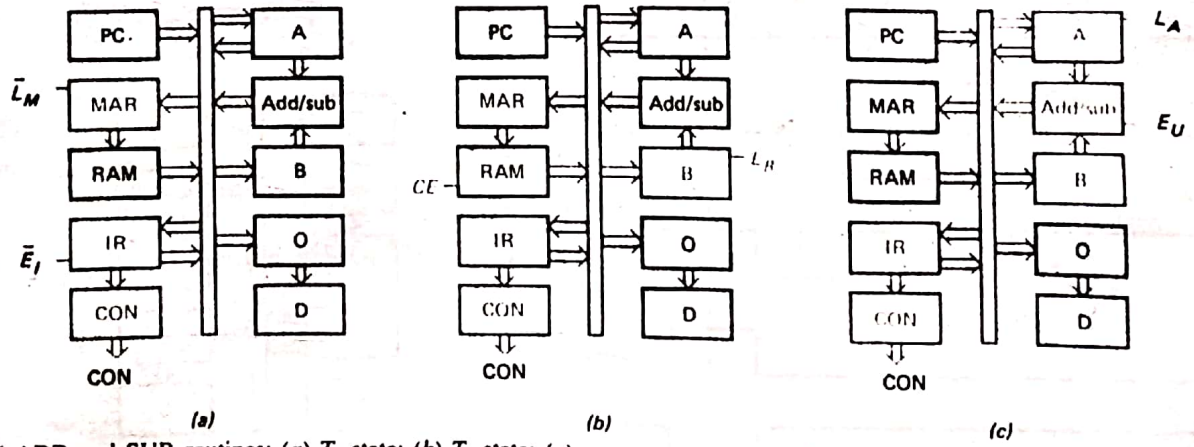


Fig. 10-6 ADD and SUB routines: (*a*) T_4 state; (*b*) T_5 state; (*c*) T_6 state.

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.

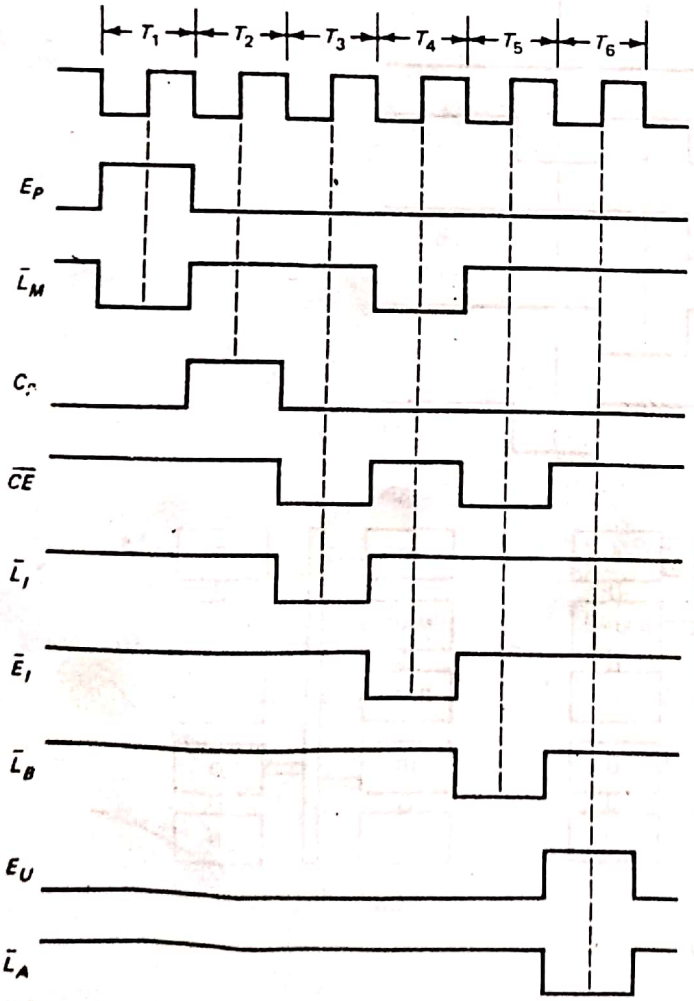


Fig. 10-7 Fetch and ADD timing diagram.

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.

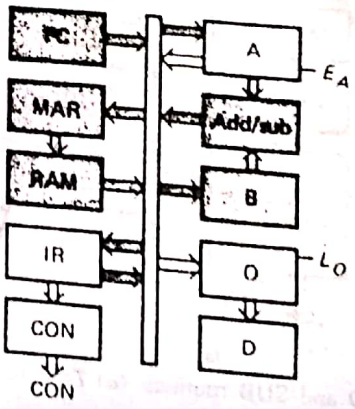


Fig. 10-8 T_4 state of OUT instruction.

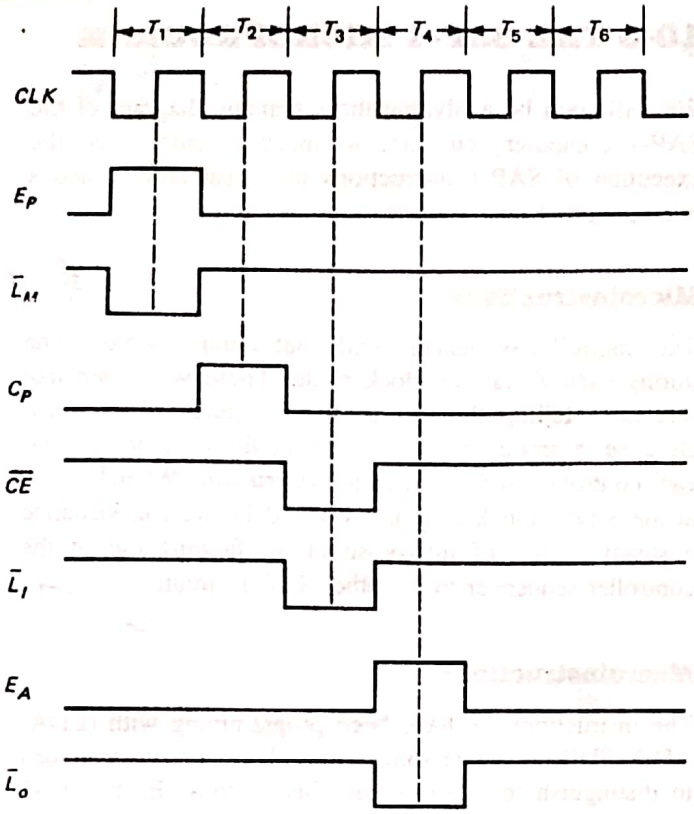


Fig. 10-9 Fetch and OUT timing diagram.

HLT

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

$$\text{IR} = 1111\ \text{XXXX}$$

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.

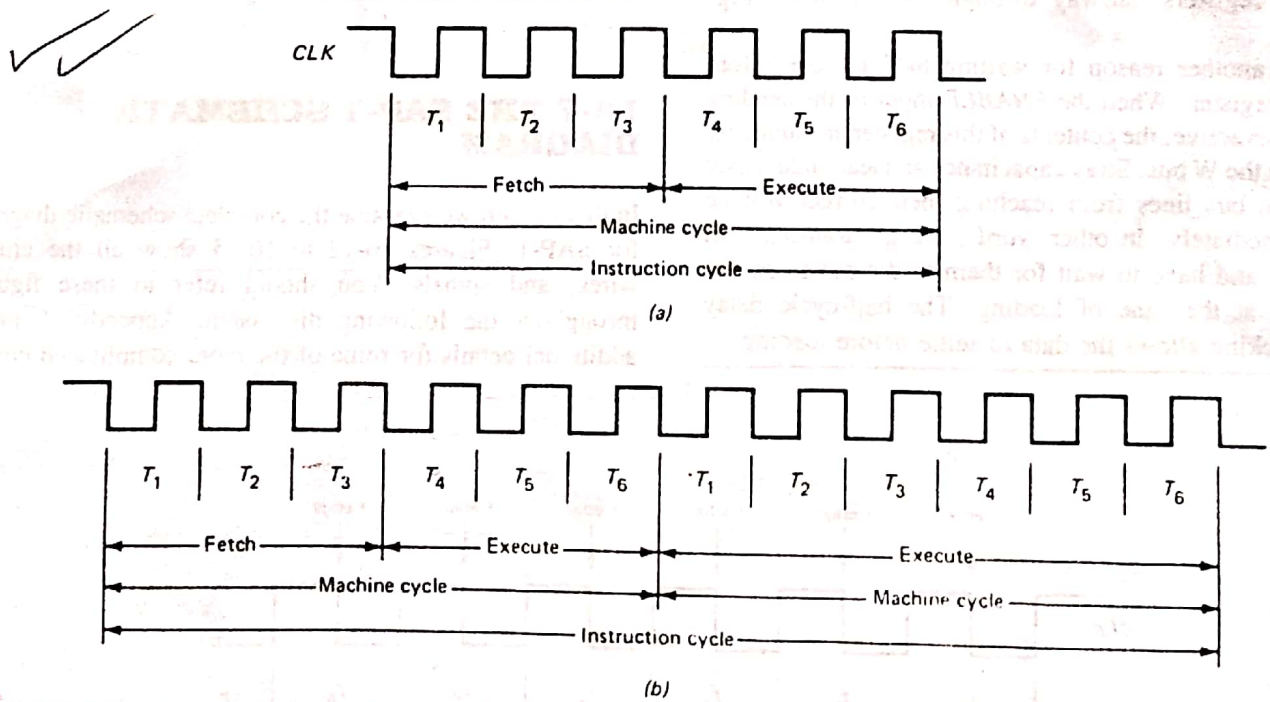


Fig. 10-10 (*a*) SAP-1 instruction cycle; (*b*) instruction cycle with two machine cycles.

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.

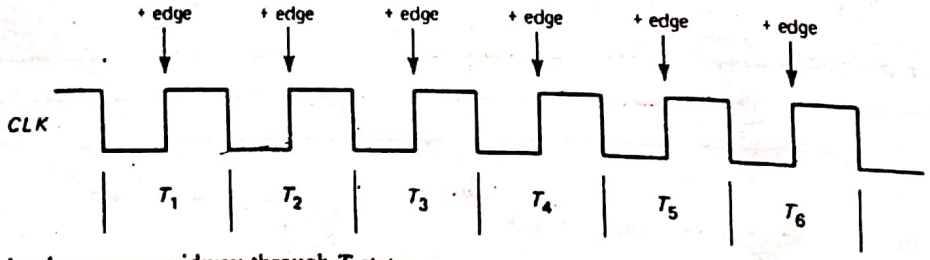


Fig. 10-11 Positive clock edges occur midway through T states.