Chapter 2. Basic computer Architecture

2.1. SIMPLE AS POSSIBLE SAP-1

SAP-1 is the first stage in the evolution towards modern computer. The main purpose of SAP is to introduce all the crucial ideas behind computer operations. But even a simple computer like SAP covers many advanced concepts.

The architecture is 8 bits and comprises of 16 X 8 memory i.e. 16 memory location with 8 bits in each location, therefore, need 4 address lines which either comes from the PC (Program Counter which may be called instruction pointer) during computer run phase or may come from the 4 address switches during the program phase. All instructions (5 only) stores in this memory. It means SAP can't store program having more than 16 instructions.

SAP can only perform addition and subtraction and no logical operation. These arithmetic operation are performed by an adder/subtractor unit.

There is one general purpose register (B register) used to hold one operand of the arithmetic operation while another is kept by the accumulator register of the SAP-1.

In addition there are 8 LEDs, work as output unit and connected with the 8 bit output register.

All timely moment of data or activities are performed by the controller/sequencer part of the SAP-1.

2.2. ARCHITECTURE

SAP-1 is a bus organized computer. All registers outputs to the W bus are three state: this allows orderly transfer of data. All other register outputs are two-state: these output continuously drive the boxes they are connected to.

The attributes in SAP-1 computer are

- W bus - A single 8 bit bus for address and data transfer.
- 16 Bytes memory (RAM)
- Registers are accumulator and B-register each of 8 bits.
- Program counter – initializes from 0000 (0H) to 1111(FH) during program execution.
- 4 bit Memory Address Register (MAR) to store memory addresses.
- 6 cycle controller with 12-bit microinstruction word (control word)

- 8 bit accumulator
- 8 bit B register.
- 8 bit Adder/Subtractor for addition and subtraction instructions
- 8 bit output register.

BLOCK DIAGRAM

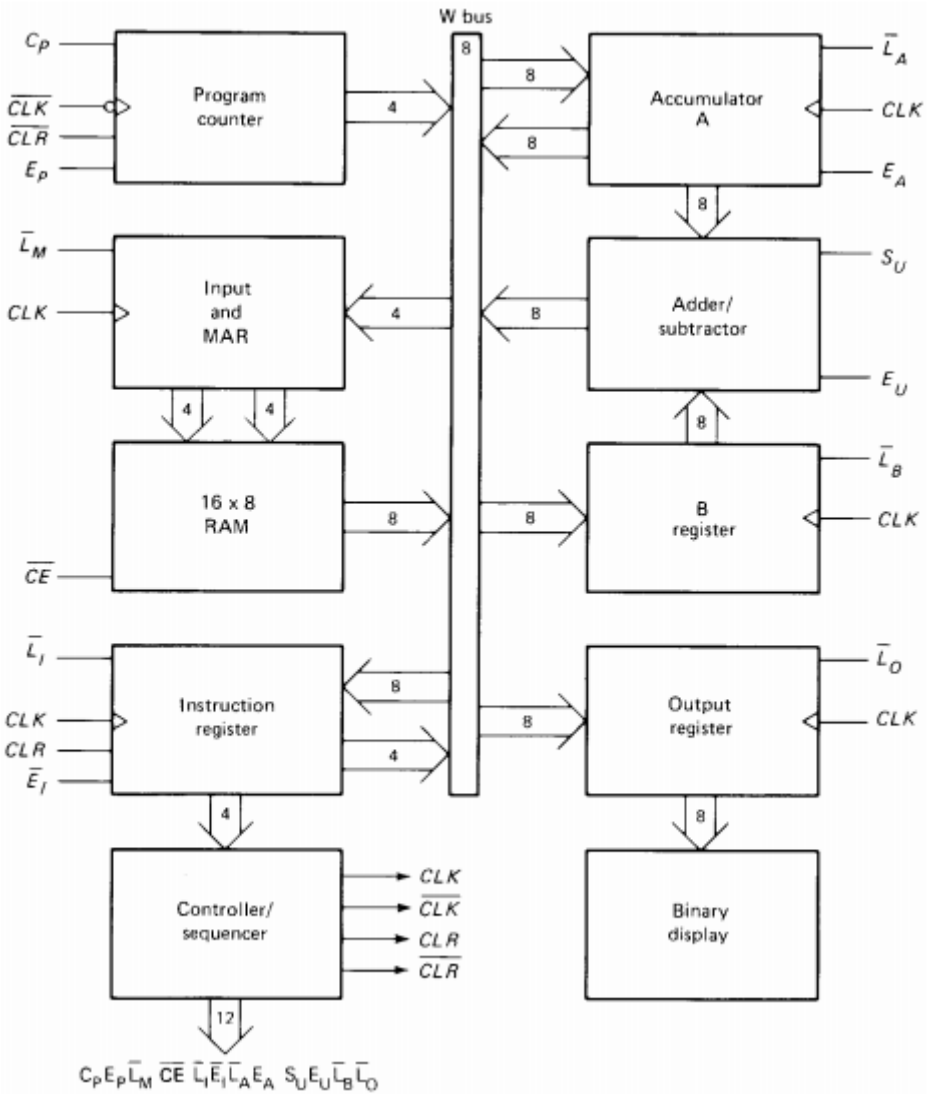


Figure 2-1: SAP-1 Architecture

Summary

Control unit: PC, IR, and Controller-sequencer.

ALU: Accumulator, Adder subtractor, B- Register.

Memory: MAR, RAM

I/O: I/P programming switches, the output port, and the binary display

2.3. COMPONENTS

The components of SAP-1 are: Program counter, Input MAR, RAM, Instruction Register, Controller Sequencer, Accumulator, The Adder-Subtractor, B register, Output Regiser, Binary Display. Following section describes them in detail.

PROGRAM COUNTER

The program counter is stored at the beginning of the memory with the first instruction at the binary address 0000, the second instruction at address 0001, the third 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.

INPUT AND MEMORY ADDRESS REGISTER (MAR)

Input and MAR block is located just below the program counter. It includes the address and data switch registers. Switch registers are part of the input unit which allows sending 4 address bits and 8 data bits to the RAM. Instruction and data words are written into RAM before a computer run.

MAR (Memory Address Register) is the part of the SAP-1 memory unit. Address in program counter (PC) is latched into MAR during a computer run. A bit latter, the MAR applies this 4 bits address to the RAM where a read operation is performed.

RAM

The RAM is SAP-1 is a 16×8 static TTL RAM. By 16×8 it means, we can store 16 words of 8 bits each. Programming of RAM is done by means of address and data switch registers. This allows us to store a program and data in the memory before a computer run.

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

INSTRUCTION REGISTER

The instruction register is the part of control unit and is 8 bit register. 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 IR are split into two nibbles. The upper nibble is a two-state output that goes directly to the block labeled "Controlled-Sequencer". The lower nibble is a three state output read onto the W bus when needed.

CONTROLLER-SEQUENCER

This block of SAP-1 is responsible for generating all the required control signals. The controller sequencer generates 12-bit word called control word. The 12 wires carrying this word can be referred as control bus.

The format of the control word is

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

Where,

C_p ➔Count Pulse	E_p ➔Enable Pulse	$\overline{L}_M$ ➔Load MAR
$\overline{CE}$ ➔Chip Enable	$\overline{L}_I$ ➔Load IR	$\overline{E}_I$ ➔Enable IR
$\overline{L}_A$ ➔Load Accumulator	E_A ➔Enable Accumulator	S_U ➔Subtract Unit
E_U ➔Enable Unit	$\overline{L}_B$ ➔Load B-Register	$\overline{L}_O$ ➔Load O/P Register

The control word determines how the registers will react to the next positive clock edge. Before each computer run, a $\overline{CLR}$ is sent to the program counter (PC) and CLR to the instruction register (IR). This resets the PC to 0000 and erases the last instruction in IR.

A clock signal is sent to all buffer registers which synchronizes the operation of computer, ensuring that things happen when they are supposed to happen. In other word, all register transfers occur on the positive edge of a common CLK signal.

For an instance, when $\overline{CE}$ is low and $\overline{LA}$ is low controller-sequences determines that RAM's data is loaded to the Accumulator.

ACCUMULATOR (A)

The accumulator is an 8-bit register that is a part of arithmetic/logic unit (ALU). This register is used to store 8-bit data and to perform arithmetic and logical operations. The data stored in the accumulator is used for operation in Adder/ Subtractor block and the result is again stored to the accumulator. The accumulator is also identified as register A.

From the block diagram, it is clear that accumulator has two outputs: the two state output goes directly to the adder-subtractor and the three state output goes to the 'W' bus when E_A is high.

ADDER-SUBTRACTOR UNIT

Adder/Subtractor block perform arithmetic operation in Microprocessor. This block takes inputs form Accumulator and/or other registers (B). The result after the operation is stored in Accumulator. Every arithmetic operation performed in this block is with the logic of addition. After performing the operation result is sent to the 8-bit W-bus.

SAP-1 uses a 2's-complement adder subtractor. When S_u is low, the sum out of the adder-subtractor is; $S=A+B$ When S_u is high, the difference appears: $S=A+\overline{B}+1$.

B- REGISTER

The B register is another 8 bit 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-subtractor, supplying the number to be adder or subtracted from the contents of the accumulator.

OUTPUT REGISTER

At the end of the computer run, the accumulator contains the answer to the problem being solved. It is also called as output port because the processed data can leave the computer from this register. When E_A is high and $\overline{L}_O$ is low, the next positive clock pulse loads the content of accumulator to output register.

BINARY DISPLAY

The binary display is a row of eight light emitting diodes. Because each LED connects to one flip flop of the output port, the binary display shows us the contents of the output port. ON state of LED represents binary 1 and OFF state represents binary 0. Therefore, after we've transferred an answer from the accumulator to the output port, we can see the answer in binary form.

2.4. SAP-1 INSTRUCTION

SAP-1 consists of 5 instructions set. These are LDA, ADD, SUB, OUT and HLT. Abbreviated instructions like these are called *mnemonics* which indicate the operation that will take place when the instruction is executed.

TABLE 2-1: SAP-1 INSTRUCTION SET

Mnemonics	OP Code	Operation
LDA	0000	Load RAM data into accumulator
ADD	0001	Add RAM data to accumulator
SUB	0010	Subtract RAM data from accumulator
OUT	1110	Load accumulator data into output register
HLT	1111	Stop processing

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

LDA

Purpose: to load the accumulator with the content of given memory location.

Binary Code (Op-Code): 0000

Syntax: **LDA** XH; X= 0H, 1H… FH

Example: LDA 8H, means load the accumulator with the contents of memory location 8H.

ADD

Purpose: to add the content of given memory location to the content of accumulator.

Binary Code: 0001

Syntax: ADD XH; X=0H, 1H… FH

Example: ADD 9H means add the contents of memory location 9H to the accumulator contents. The sum replaces the contents of the accumulator. During the execution of ADD 9H, the content of memory location (at address 9H) is transferred to the register B and then content of B register is added to the accumulator by adder-subtractor unit whose content is loaded into accumulator. Symbolically, A←A+B

SUB

Purpose: to subtract the content of given memory location from the content of accumulator and store the result in accumulator.

Binary Code: 0010

Syntax: SUB XH; X=0H, 1H…FH

Example: SUB CH means "subtract the contents of memory location CH from the contents of the accumulator"; the difference out of the adder-subtractor then replaces the original contents of the accumulator. Symbolically: (A←A-B)

OUT

Purpose: to transfer the content of accumulator to the output register. After, OUT has been executed, we can see the result in the binary display.

Syntax: OUT.

Note: It is complete instruction (non-memory reference instruction). It does not contain any address field.

Binary Code: 1110

HLT (halt)

Purpose: to tell the computer to stop processing data. HLT marks the end of a program. We must use a HLT instruction at the end of every SAP-1 program.

Syntax: HLT

Note: It also does not contain address field being a complete instruction.

Binary code: 1111

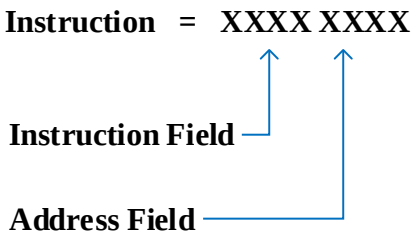
HOME WORK:

How do you program to solve this arithmetic problem using SAP-I instruction set? 16+24+9-23; all the numbers in decimal.

2.5. INSTRUCTION FORMAT OF SAP-1

The four MSBs of a SAP-1 machine language instruction specify the operation and the four LSBs gives the address.

We refer to the MSBs as the instruction field and to the LSBs as the address field. Symbolically,



2.6. INSTRUCTION CYCLE

Instruction cycle is defined as the total T-states needed to fetch and execute an instructions. In SAP-1, each instruction cycle consists fixed six T-state, i.e. six T-state are reserved for each instruction. Not all the instructions require all the six T-states for execution. The unused T-states are marked as 'No Operation (NOP)' cycle

T= T6T5T4T3T2T1

Six T states are called machine cycle for SAP-1. A ring counter is used to generate a T-state at every falling edge of clock pulse. The ring counter is reset at every 6^{th} T-state.

The six T-state is divided into two cycles:

(i) Fetch Cycle: T1 (Address State),T2 (Increment State), T3 (Memory State)

(ii) Execution Cycle: T4, T5, T6, but the task of each steps depends on the instruction

2.7. FETCH CYCLE

In fetch cycle, the instruction is fetched (read) from memory which contains initial three states T1 (Address State), T2 (Increment State), and T3 (Memory State). The fetch cycle is same for all instruction of SAP-1.

T1: Address State

The T1 state is called the address state because the address in the program counter (PC) is transferred to the memory address register (MAR).

MAR ←[PC]

During this state EP and $\overline{L}_M$ control signals are active and all other control bits are inactive. Thus the controller sequencer sends the word:

CON= C_P E_P L'_M C'_E L'_I E'_I L'_A E_A S_U E_U L'_B L'_O

0 1 0 1 1 1 1 0 0 0 1 1

= 5E3H

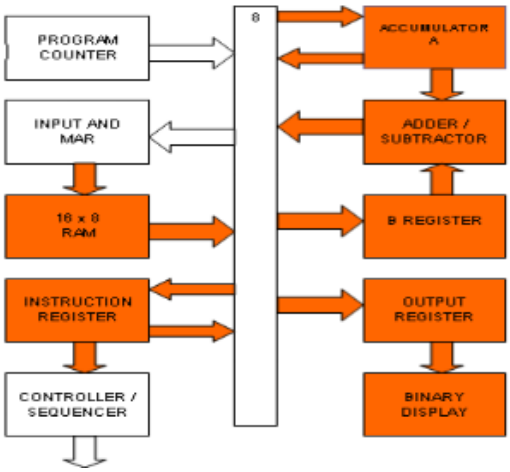


Figure 2-2: Fetch Cycle- T1 Cycle

T2: Increment State

T2 state is called increment state as during T2 state the program counter gets increment by counting the pulse. PC←PC+1

At this state only C_P bit is active so control word would be

CON=	C_P	E_P	L'M	C'_E	L'I	E'_I	L'A	E_A	S_U	E_U	L'B	L'O
	1	0	1	1	1	1	1	0	0	0	1	1

=BE3H

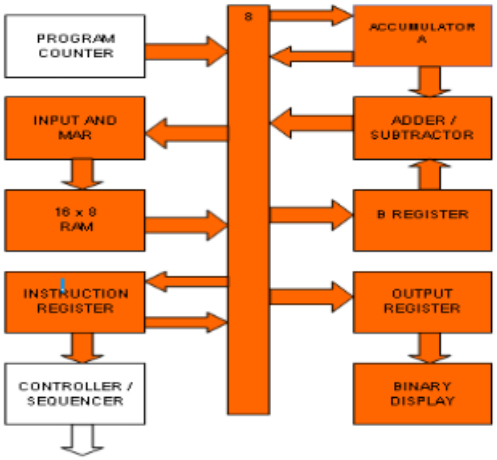


Figure 2-3: Fetch Cycle T2 State

T3: Memory State

The T3 state is called the memory state because the addressed RAM instruction is transferred from the memory to the instruction register. The active control bits during this state are $\overline{CE}$ and $\overline{L}_I$, and the word out of the controller-sequencer is

CON=	C_P	E_P	L'_M	C'_E	L'_I	E'_I	L'_A	E_A	S_U	E_U	L'_B	L'_O
	0	0	1	**0**	**0**	1	1	0	0	0	1	1

= 263H

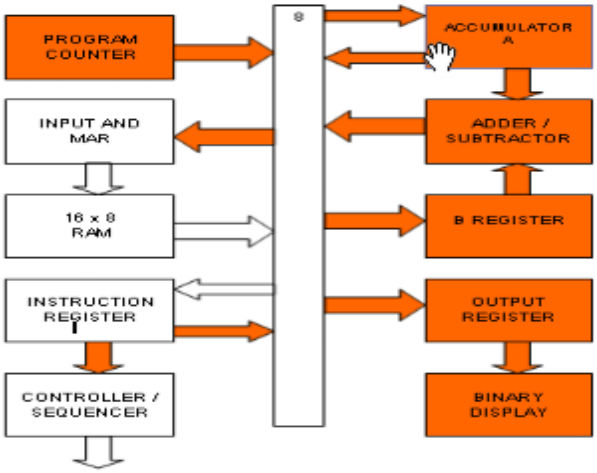


Figure 2-4: Fetch Cycle- T3 state

2.7. EXECUTION CYCLE

After fetching of an instruction, the actual operand to be performed is completed in execution cycle. The upper three states (T4, T5, and T6) are used in execution cycle and the register transfer during this cycle are instruction dependent. Therefore this cycle differs for each instruction.

LDA routine

T4: memory address is sent from IR to MAR.

T5: data from memory is fetched and send to ACC.

T6: do nothing!

Let's assume that the instruction register (IR) has been loaded with LDA AH

IR = 0000 1010

T4 state

During the T4 state, the instruction field 0000 goes to the controller-sequencer, where it is decoded: the address field 1010 is loaded into the MAR. Symbolically,

Controller-sequencer← op-code (0000 in this example).

MAR←address-field of IR (1010 in this example).

To perform this operation $\overline{E}_I$ and $\overline{L}_M$ are active; all other control bits are inactive. The control word will be

CON=	C_P	E_P	L'_M	C'_E	L'_I	E'_I	L'_A	E_A	S_U	E_U	L'_B	L'_O
	0	0	**0**	1	1	**0**	1	0	0	0	1	1

= 1A3H

T5 state

During the T5 state data word in the RAM will be loaded into the accumulator on the next positive clock edge. Symbolically,

A ←[RAM] (content of address 1010 is loaded to A)

To perform the above operation, $\overline{CE}$ and $\overline{L}_A$ are active. Therefore the addressed RAM word sets up the instruction register via the W bus. The control word is:

CON=	C_P	E_P	L'_M	C'_E	L'_I	E'_I	L'_A	E_A	S_U	E_U	L'_B	L'_O
	0	0	1	**0**	1	1	**0**	0	0	0	1	1

= 2C3H

T6 state

T6 is a non-operation state. During this third execution state, all registers are inactive. This means that the controller-sequencer is sending out a word whose bits are all inactive. The T6 state of the LDA routine is a *nop* (no operation).

CON=	C_P	E_P	L'_M	C'_E	L'_I	E'_I	L'_A	E_A	S_U	E_U	L'_B	L'_O
	0	0	1	1	1	1	1	0	0	0	1	1

= 3E3H

Therefore, over all machine cycle for LDA routine can be summarized as

T-State	Operation	Active Signals	Control word
T1	MAR←PC	E_P, $\overline{L}_M$	5E3H
T2	PC←PC+1	C_P	BE3H
T3	IR←[RAM]	$\overline{CE}$, $\overline{L}_I$	263H
T4	MAR←[Address field]	$\overline{E}_I$,$\overline{L}_M$	1A3H
T5	A←[RAM]	$\overline{CE}$,$\overline{L}_A$	2C3H
T6	NOP	……..	3E3H

The timing diagram for LDA routing is as shown in fig. 2-5.

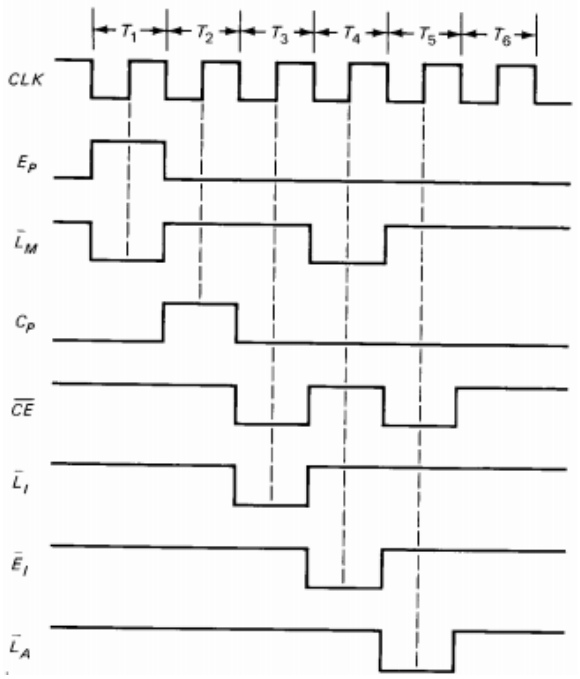
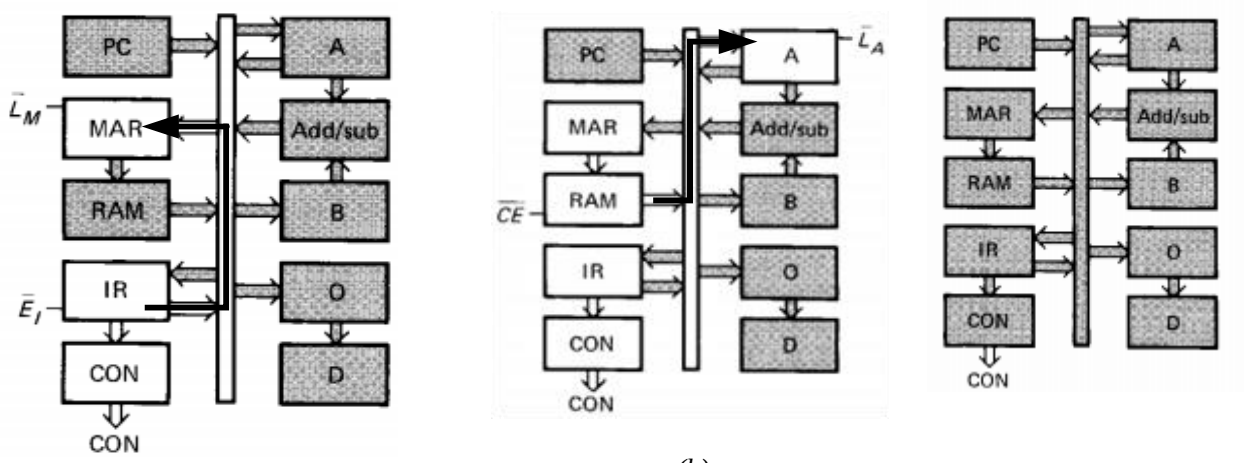


Figure 2-5: Fetch and LDA timing diagram

The working of the execution cycle is as described in fig. 2-6.



(a) *(b)* *(c)*

Figure 2-6: LDA routine: (a) T-4 state (b) T-5 state (c) T-6 state

ADD Routine

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

IR = 0001 1011

T4 state

During the T4 state the instruction field 0001 goes to the controller-sequencer and the address field to the MAR. During this state $\bar{E}_I$ and $\bar{L}_M$ are active.

MAR ←[Address field of IR]

The control word will be

CON=	C_P	E_P	L'_M	C'_E	L'_I	E'_I	L'_A	E_A	S_U	E_U	L'_B	L'_O
	0	0	**0**	1	1	**0**	1	0	0	0	1	1

= 1A3H

T5 state

During T5 state, the addressed RAM word (pointed by MAR in T4 state) is transferred to B-register. Control bit $\overline{CE}$ and $\bar{L}_B$ are active.

B ←[RAM]

Thus the control word is

CON=	C_P	E_P	L'_M	C'_E	L'_I	E'_I	L'_A	E_A	S_U	E_U	L'_B	L'_O
	0	0	1	**0**	1	1	1	0	0	0	**0**	1

= 2E1H

T6 state

During the T6 state, E_U and $\bar{L}_A$ are active; therefore, the adder-subtractor unit performs addition and then the result is transferred to the accumulator.

A←A+B

The control word is

CON=	C_P	E_P	L'_M	C'_E	L'_I	E'_I	L'_A	E_A	S_U	E_U	L'_B	L'_O
	0	0	1	1	1	1	**0**	0	0	**1**	1	1

= 3C7H

Now the complete instruction of ADD routine can be summarized as;

T-State	Operation	Active Signals	Control word
T1	MAR←PC	E_P, $\overline{L}_M$	5E3H
T2	PC←PC+1	C_P	BE3H
T3	IR←[RAM]	$\overline{CE}$, $\overline{L}_I$	263H
T4	MAR←[Address field]	$\overline{E}_I$,$\overline{L}_M$	1A3H
T5	B←[RAM]	$\overline{CE}$,$\overline{L}_B$	2E1H
T6	A←A+B	E_U , $\overline{L}_A$	3C7H

Fig. 2-7 shows the timing diagram for the ADD routines.

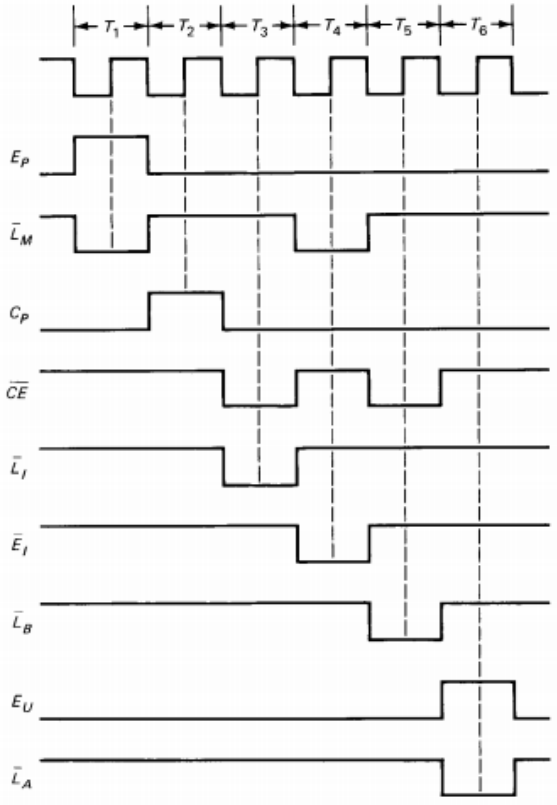


Figure 2-7: ADD timing diagram

SUB ROUTINE

The difference in instruction cycle of ADD routine and SUB routine is only that, during T6 state, S_U line becomes active in SUB routine so that the Adder/Subtractor unit performs subtraction operation using 2's complement method.

T6 state of SUB

A←A-B; active signal: S_U, E_U, $\bar{L}_A$

CON=	C_P	E_P	L'_M	C'_E	L'_I	E'_I	L'_A	E_A	S_U	E_U	L'_B	L'_O
	0	0	1	1	1	1	**0**	0	**1**	**1**	1	1

= 3CFH

The working of execution cycle of ADD and SUB routine is shown in fig. 2-8.

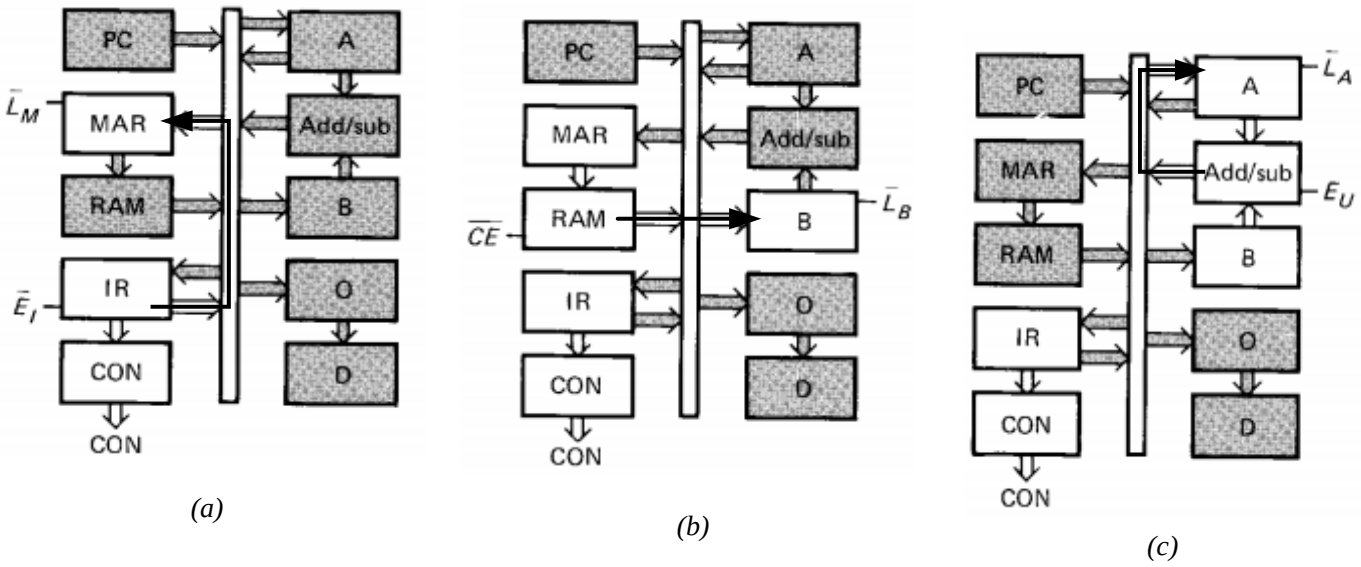


Figure 2-8: ADD and SUB routines: (a) T4 State; (b) T5 state, (c) T6 state.

OUT Routine

The OUT routine is completed during T4 state, thus, The T5 and T6 states are No-operation state (nop). Fig. 2-9 shows the timing diagram for the fetch and ADD routine and fig. 2-10 shows the active section of SAP-1 during the execution of OUT instruction (at T4).

T4 state

At this state output register gets loaded with the content of accumulator. i.e.

Output Register←A

E_A and $\bar{L}_O$ are active, the next positive clock edge loads the accumulator contents into the output register. The control word is

CON=	C_P	E_P	L'_M	C'_E	L'_I	E'_I	L'_A	E_A	S_U	E_U	L'_B	L'_O
	0	0	1	1	1	1	1	**1**	0	0	1	**0**

= 3F2H

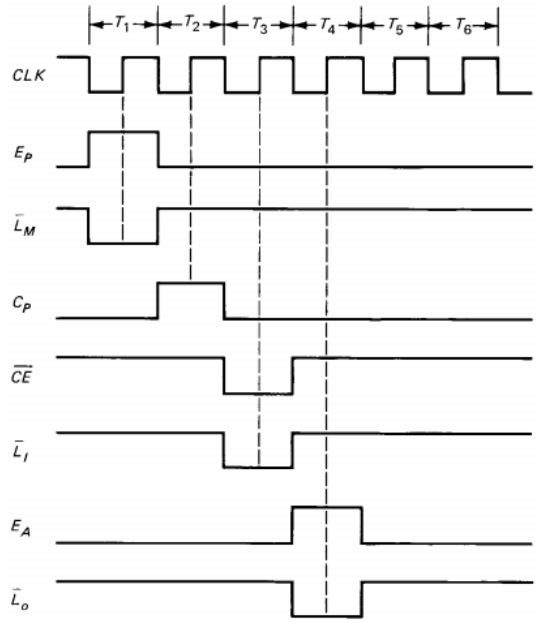


Figure 2-9: OUT timing diagram

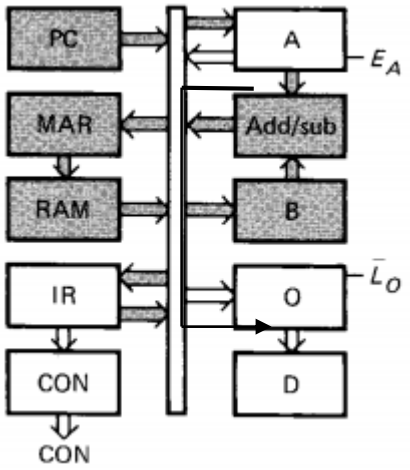


Figure 2-10: T4 state, OUT instruction

2.8. MICROINSTRUCTION

The controller-sequencer sends out control words, one during each T state which are like directions telling the rest of the computer what to do. The operation (micro-operation) performed in each clock pulse is called a microinstruction. In SAP-1 system, each instruction has six microinstructions (three for fetch cycle and three for execution cycle). As the control word tells what to do, so sometimes control word is referred to as microinstruction.

A sequence of microinstructions constitutes a *microprogram*.

MACROINSTRUCTION

The instructions we have been programming with (LDA, ADD, SUB...) are sometimes called *macroinstruction*. Each microinstruction of SAP-1 is actually made up of six micro-instructions among them first three are common to all microinstructions (fetch cycle). Therefore, sometimes it is said that SAP-1 macroinstructions contain only three micro-instruction (i.e. only of execution cycle).

T-STATE

One sub-division of an operation performed in one clock cycle is called a *T-state*. These sub-divisions are internal state synchronized with the system clock and each T-state is precisely equal to one clock period.

MACHINE CYCLE

SAP-1 has six T states (three fetch and three execute). These six states are called a *machine cycle*. 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 1ms. Therefore, it takes 6ms for a SAP-1 machine cycle.

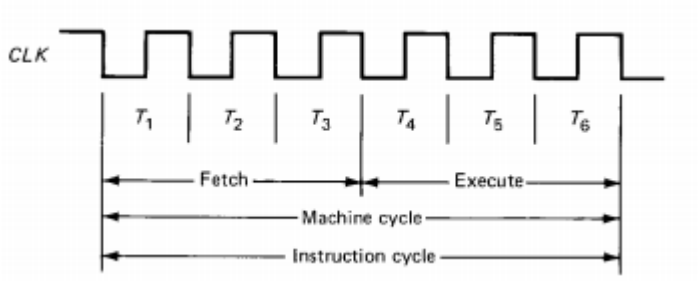


Figure 2-11: SAP-1 machine cycle

For instance LDA routine

Microinstruction

Macroinstruction	**T-State**	**Operation**	**Control word**
LDA	T1	MAR←PC	5E3H
	T2	PC←PC+1	BE3H
	T3	IR←[RAM]	263H
	T4	MAR←[Address field of IR(LSB nibble)]	1A3H
	T5	A←[RAM]	2C3H
	T6	NOP	3E3H

2.9. SAP-2 ARCHITECTURE

The attributes in SAP-2 computer are

- Bidirectional registers.
- Two input and two output port
- HEX keyboard encoder and HEX display
- 16-bit address bus (i.e. 16-bit PC, MAR)
- 8 bit data bus
- 64 kB memory (initial 2KB ROM and 62KB RAM)
- 8 bit memory data register (MDR)
- 8 bit ALU with 2 bit flags (sign and zero)
- 8 bit accumulator (A)
- 8 bit TMP, B and C registers

The architecture of SAP-2 computer system is listed above. The working principle of SAP-2 is similar to that of SAP-1 with some advancements. As in SAP-1, here all registers output to the W-bus are three state and those not connected to the bus are two state and the controller-sequencer sends all required control signals to each register. With increment of some blocks, it is obvious that the control word is also of larger than that of SAP-1.

The most exciting feature of SAP-2 is that it has jump instructions which force the computer to repeat or skip part of a program.

2.10. BIDIRECTIONAL REGISTER

To reduce the wiring capacitance of SAP-2, we will run only one set of wires between each register and the bus. The input and output pins are shorted; only one group of wires is connected to the bus.

During a computer run, either LOAD or ENABLE may be active, but not both at the same time. An active LOAD means that a binary word flows from the bus to the register input; during the load operation, the output lines are floating. On the other hand, an active ENABLE means that a binary word flows from the register to the bus; in this case, the input lines are float.

The input and output lines are internally connected which not only reduces the wiring capacitance; it also reduce the number of I/O pins.

2.11. BLOCK DIAGRAM

Fig. 2-12 shows the architecture of SAP-2. Similar as SAP-1, all register output to the W bus are three-state; those not connected to the bus are two state and the controller-sequencer sends control signal to each register.

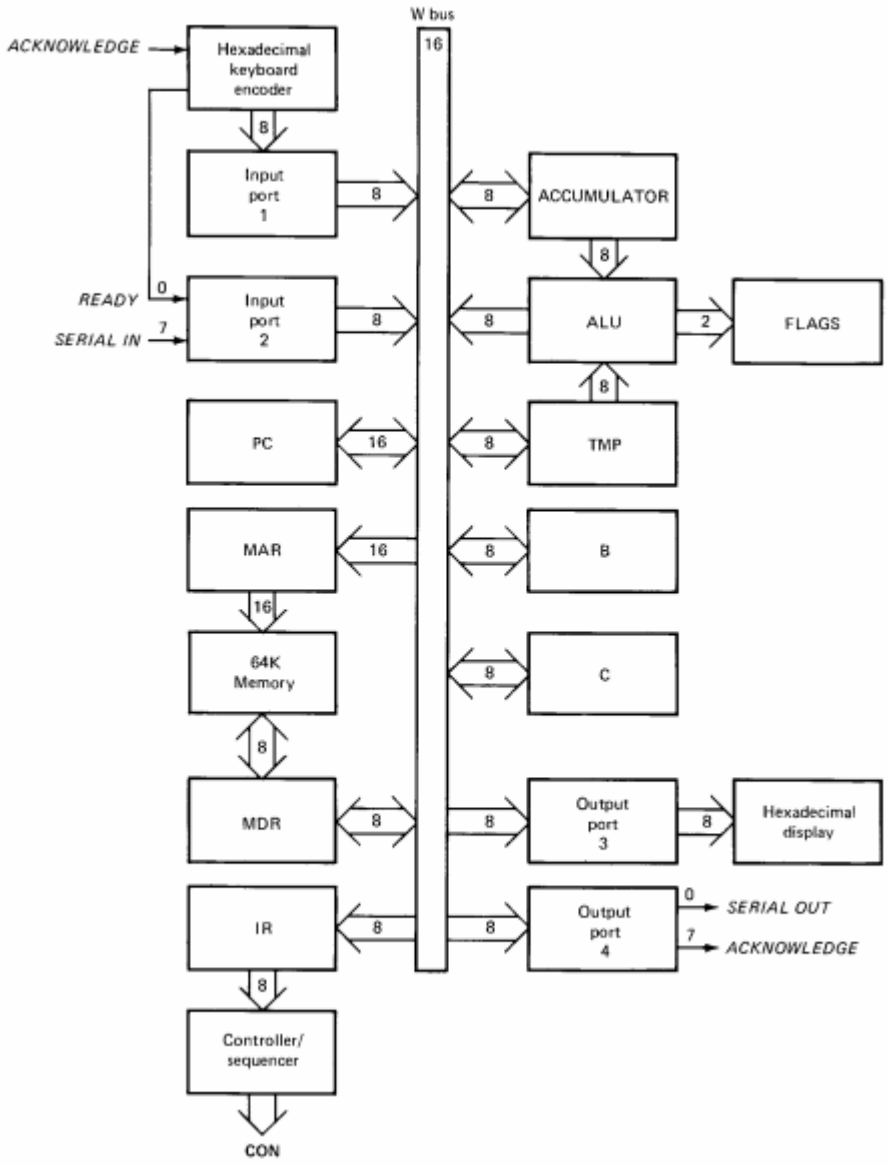


Figure 2-12: Architectural Block Diagram of SAP-2

2.12. COMPONENTS

Hexadecimal Keyboard encoder: The hexadecimal keyboard encoder receives the data from outer environment and converts it into hexadecimal form the system can understand and send them to the input port.

Input Ports: The SAP-2 contain two input ports (1 and 2) which inputs the data in the system in the most convenient way. Both the ports output 8-bit data to the W-bus. A hexadecimal keyboard encoder is connected to port 1 which allows us to enter hexadecimal instructions and data through port 1.

Port 2 is used for serial data through pin 7 and later the serial data is converted to parallel one and transferred to the W-bus.

PC: PC is the program counter that holds the address of the next instruction to be fetched. It initializes from 0000H to 1111H (decimal 0 to 65,535) during the execution.

A low $\overline{CLR}$ signal reset the PC before each computer run; so the data processing starts with the instruction stored in memory location 0000H.

MAR and Memory: MAR is the memory address register that stores the complete format of the address send by the program counter. It stores the final address of the memory word that needs some computations.

During the fetch cycle, the MAR receives 16-bit addresses from the program counter. The two-state MAR output then addresses the desired memory location. The memory contains 64 K where data and instruction reside. All the computations are performed relative to the memory. The memory has 2K ROM with addresses of 0000H to 07FFH. This ROM contains a program called a monitor that initializes the computer on power-up, interprets the keyboard inputs, etc. The rest of the memory is a 62K RAM with addresses from 0800H to FFFFH.

Memory Data Register (MDR): The MDR is an 8-bit buffer register used to hold the data to be written on the memory or data to be transferred to the W-bus from memory. MDR receives data from the bus before a write operation and it sends data to the bus after a read operation.

Instruction Register (IR): The IR is the Instruction Register that holds the instruction after fetch cycle. Because SAP-2 has more instructions than SAP-1, it use 8 bits for the op code rather than 4 in SAP-1. The 8-bit can accommodate 256 instruction (2^8), however SAP-2 has only 42 instructions. The 8-bit op code then transmitted to the controller sequencer.

Controller- sequencer: The controller-sequencer produces the control words or microinstructions that coordinate and direct the rest of the computer. Since, the SAP-2 has a bigger instruction set and hardware circuitry, the size of control word is also larger but the idea of generating control word (CON) is same. The control word or microinstruction determines how the registers react to the next positive clock edge.

Accumulator: Accumulator is also an 8-bit bidirectional buffer register used to store the result of all the mathematical operations. It is one of the operand of ADD, OUT, SUB instruction. It is also known as processor register. It has two outputs: the two state output of the accumulator goes to the ALU and the three state output goes to W-bus. Therefore, the 8-bit word in the accumulator continuously drives the ALU, but this same word appears on the bus only when E_A is active.

ALU and Flag: The ALU perform all the arithmetic and logical operation on 8-bit data. The operand for ALU operations are accumulator (A) and TMP register. Output of the operation is stored on accumulator.

SAP-2 has two flags (sign and Zero) that may setup during the ALU operations. Basically, a flag is a flip-flop that reflect the intermediate changes on the values during execution. The sign flag is set when the accumulator contents become negative during the execution of some instruction. The zero flag is set when the accumulator contents become zero. Actually, all the ALU instructions use the accumulator during the execution cycle. If the accumulator content goes negative or zero while executing any instruction, the sign or zero flag will be set.

The circuitry used for flags in SAP-2 is depicted below:

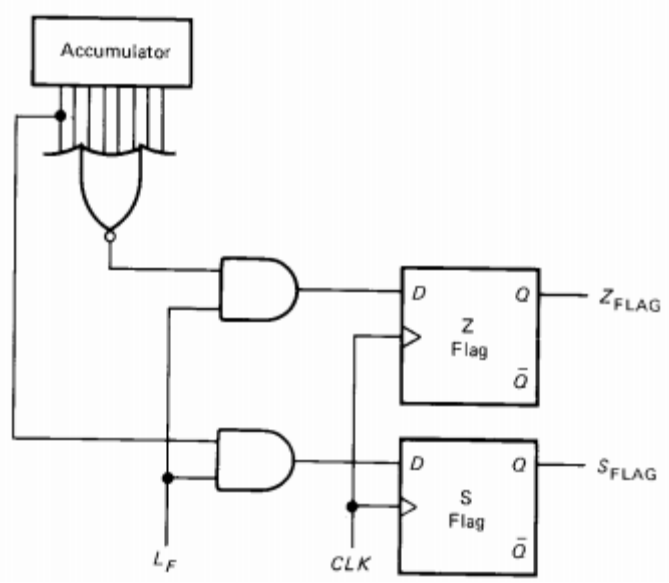


Figure 2-13: Setting the flags

Fig. 2-13 shows the circuits used in SAP-2 to set the flags. When the accumulator contents are negative, the leading bit A7 is a 1. This sign bit drives the lower AND gate. When the accumulator contents are zero, all bits are zero and output of NOR gate is a 1. This NOR output drives the upper AND gate. If gating signal L_F is high, the flags will be updated to reflect the sign and zero condition of the accumulator. This means the Z_{FLAG} will be high when the accumulator contents are zero; the S_{FLAG} will be high when the accumulator contents are negative.

Temporary register, B, C: Instead of using the B register to hold the data being added or subtracted from the accumulator, a temporary (TMP) register is used. This allows us more freedom in using the B register. SAP-2 also contains 8 bit C register which gives more flexibility in moving data during a computer run. The register B and C is accessible to the programmer.

Output Ports: SAP-2 has two output ports: port 3 and port 4. Port 3 is used for parallel output and port 4 is used for serial output. The content of the accumulator can be loaded into port 3, which drives a hexadecimal display. This allows us to see the processed data.

Hexadecimal Display: Unlike Sap-1 which has binary display, Sap-2 has a hexadecimal display to show outputs in the LEDs.

2.13. INSTRUCTION SET FOR SAP-2

1. LDA and STA

   LDA means *load the accumulator* with the addressed memory data. The only difference is that more memory locations can be accessed in SAP-2 because the address are from 0000H to FFFFH.

   Syntax: LDA XXXX; (XXXX can range from 0000H to FFFFH)
   e.g. LDA 2030H means the content of addressed data on 2030 is loaded into A. i.e. A ←[2030]

   STA is a mnemonics for *store the accumulator* which means to store the accumulator contents at specified memory location.

   Syntax: STA XXXX; (XXXX can range from 0000H to FFFFH)
   e.g. STA 7FFFH means to store the accumulator contents at memory location 7FFFH. i.e. [7FFF]←A.

2. MOV and MVI

   MOV is the mnemonics to *move* data from one register to another.
   Syntax: MOV destination register, source register
   e.g. MOV A,B means to move the data in the B register to the accumulator. i.e. A←B.
   The data can be moved between A, B, and C registers so the available MOV instruction are
   MOV A, B
   MOV A, C
   MOV B,A
   MOV B,C
   MOV C,A
   MOV C,B

   MVI is the mnemonics for *move immediate* which tells the computer to load a designated register with the 8-bit immediate data.

   Syntax: MVI Register, 8-bit data

e.g. MVI C, ABH means to move the data (ABH) in the C register.i.e. C= ABH.

You can use MVI with the A, B, and C registers. The formats for these instructions are

MVI A, byte (8-bit data)

MVI B, byte

MVI C, byte

3. ADD and SUB

ADD and SUB are the mnemonics used for addition and subtraction the data in the designated register to the accumulator.

Syntax: ADD Register

SUB Register

e.g. ADD B ➔Add the content of B register to accumulator. i.e. A=A+B

SUB C ➔Subtract the data in the designated register from the accumulator. i.e. A=A-C.

The formats for the ADD and SUB instructions are

ADD B

ADD C

SUB B

SUB C

4. INR and DCR

INR and DCR are the mnemonics for increment and decrement the designated register by 1.

Syntax: INR register

DCR register

The formats for these instructions are

INR A

INR B

INR C

DCR A

DCR B

DCR C

As an example, if

B = 56H and C= 8AH

Then the execution of INR B results in B= 57H and the execution of DCR C produces C =89H.

5. ANA and ANI

ANA (AND the accumulator) is mnemonics to perform the logical operation between content of accumulator and content of specified register where ANI (AND immediate) performs logical AND operation between accumulator content and the provided 8-bit data.

Syntax: ANA register
ANI 8-bit data

e.g. ANA B ➔means to AND the contents of the accumulator with the content of B register. i.e. A=A **AND** B.

Two ANA instructions are available in SAP-2: ANA B and ANA C.

e.g. ANI 01H ➔means to AND the content of the accumulator with the data 01H. i.e. A=A **AND** 01H.

For instance, if A= 0101 1110 and execution of **ANI C7** will AND 0101 1111 with 1100 0111 which produce the new accumulator contents of A = 0100 0110.

6. ORA and ORI

ORA is the mnemonics for *OR the accumulator* with the designated register. It perform the OR logical operation between accumulator and specified register. SAP-2 has two logical ORA instructions: ORA B and ORA C.

Syntax: ORA Register.

ORA B ➔A = A OR B

ORA C ➔ A= A OR C

ORI is the mnemonics for *OR immediate*. The accumulator contents are ORed with the byte that follows the op code.

Syntax: ORI 8-bit data

e.g. ORI 5AH ➔means to OR the content of the accumulator with the data 5AH. i.e. A=A **OR** 5AH.

7. XRA and XRI

XRA means XOR the accumulator with the designated register. The SAP-2 instruction set contains XRA B and XRA C.

Syntax: XRA Register.

e.g. XRA B ➔ A= A Ex-OR B
XRA C ➔A = A Ex-OR C.

XRI means *XOR immediate* which performs the logical Ex-OR operation between accumulator's contents and provided 8-bit data.

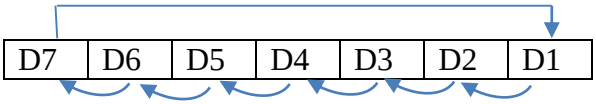
Syntax: XRI 8-bit data.

e.g. XRI A9H ➔ A= A Ex-OR A9H

8. CMA
   CMA stands for "complement the accumulator". The execution of a CMA flips each bit in the accumulator (1's complement).

9. IN and OUT
   IN is the mnemonics for *input* which is used to read the data from input port.
   Syntax: IN 8-bit port address
   e.g. IN 01H ➔ means to transfer the data in port 1 to the accumulator.
   OUT stand for *output* which is used to transfer the accumulator word into the designated output port.
   Syntax: OUT 8-bit port address
   e.g. OUT 03H ➔ means to transfer the contents of the accumulator to port 3.
   Since the output ports are numbered 3 and 4 so we have to specify which port is to be used.

10. RAL and RAR
    RAL is the mnemonics for *rotate the accumulator left*. This instruction will shift all bits to the left and move the MSB into the LSB position.



    RAR stands for *rotate the accumulator right.* The bits shift to the right, the LSB going to the MSB position.

11. NOP and HLT
    NOP stands for *no operation*. During the execution of a NOP, all T states are do nothing hence no register changes occurring during NOP.
    HLT stands for *halt* which end the data processing.

12. JUMP Instructions.
    JMP 16-bit address ➔Unconditional jump
    JZ 16-bit address
    JNZ 16-bit address ➔Conditional jump
    JM 16-bit address

    The unconditional jump instruction transfers the control of execution to the specified address whenever that instruction executes.

In case of conditional jump instruction, the jump will takes place only when the condition is true. Otherwise next instruction will be executed.

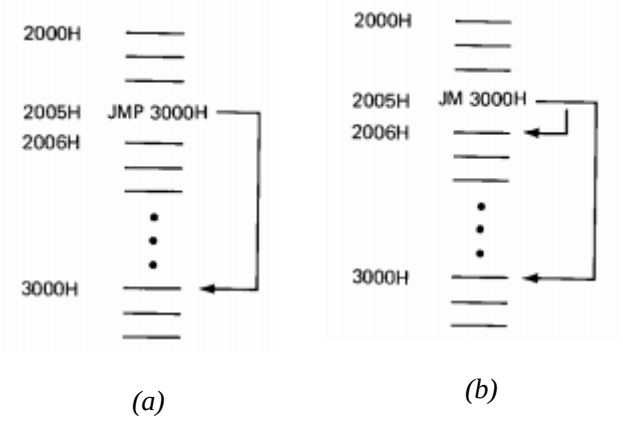


Figure 2-14: (a) Unconditional jump, (b) Conditional jump

E.g. JMP 3000H ➔JMP is mnemonics for jump which tells the computer to get the next instruction from memory location 3000H.
JM 3000H➔ JM is a mnemonics for *jump if minus*. The computer will jump to a designated address of and only of only if the sign bit is set.
JZ 3000H ➔JZ is the mnemonics for *jump if zero*; it tells the computer to jump to the designated address (3000H) only of the zero flag is set.

13. CALL and RET

CALL is the mnemonics for *call the subroutine*.
Syntax: CALL 16-bit address
e.g. CALL 5000H ➔ means to call the subroutine which has the initial instruction at address 5000H.
When a CALL is executed in the SAP-2 computer, the contents of the program counter are automatically saved in the memory locations FFFEH and FFFFH (the last two memory locations). The CALL address is then loaded into the program counter, so that execution begins with the first instruction in the subroutine.

CALL is unconditional, like JMP. Once a CALL has been fetched into the instruction register, the computer will jump to the starting address of the subroutine.

RET stands for *return*. It is used at the end of every subroutine to tell the computer to go back to the original program. After the subroutine is finished, the RET instruction causes the address in memory locations FFFEH and FFFFH to be loaded back into the program counter. This returns control to the original program.
Syntax: RET

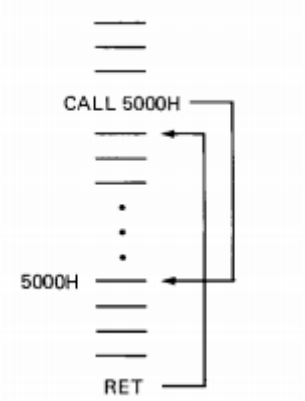


Figure 2-15: CALL and RET instruction.

2.14. FLAGS

The sign or zero flag may be set or reset during certain instructions. Table 2-1 lists the SAP-2 instructions that can affect the flags. All the instructions use the accumulator during the execution cycle. If the accumulator goes negative or zero while one of these instruction being executed, the sign or zero flag will be set.

Table 2-1: Instructions affecting flags.

Instruction	Flags Affected
ADD	S,Z
SUB	S,Z
INR	S,Z
DCR	S,Z
ANA	S,Z
ORA	S,Z
XRA	S,Z
ANI	S,Z
ORI	S,Z
XRI	S,Z

Example 1: Show the mnemonics for a program that loads the accumulator with 49H, the B register with 4AH, and the C register with 4BH; then have the program store the accumulator data at memory location 6285H

Mnemonics

```
MVI A, 49H
MVI B, 4AH
MVI C, 4BH
STA 6285H
HLT
```

Example 2: Show the mnemonics for adding 23 and 45. The answer is to be stored at memory location 5600H. Also, the answer incremented by 1 is to be stored in C register.

Mnemonics

```
MVI A, 17H
MVI B, 2DH
ADD B
STA 5600H
INR A
MOV C, A
HLT
```

Example 3

Show a program that multiplies decimal 12 and 8.		**Modify the program by using JNZ instead of a JZ.**
Solution:		Solution:
MVI A, 00H	; Clear the accumulator	MVI A, 00H
MVI B, 0CH	; Load decimal 12 into B	MVI B, 0CH
MVI C, 08H	; Preset counter with 8.	MVI C, 08H
REPEAT: ADD B	; Add decimal 12	REPEAT: ADD B
DCR C	; Decrement the counter.	DCR C
JZ DONE	; Test for zero	JNT REPEAT
JMP REPEAT	; Do it again	HLT
DONE: HLT	; Stop it.	

Example 4: The bits in a byte are numbered 7 to 0 (MSB to LSB). Show a program that can input a byte from port 2 and determine if bit 0 is a 1 or a 0. If the bit is a 1, the program is to load the accumulator with an ASCII Y (yes). If the bit is a 0, the program should the accumulator with an ASCII N (no). The yes or no answer is to be sent to output port 3.

Mnemonics

```
        IN 02H      ; get byte from port 2
        ANI 01H     ; isolate bit 0
        JNZ YES     ; jump if bit 0 is a 1
        MVI A, 4EH  ; load N into accumulator
        JMP DONE    ; skip next instruction
YES:    MVI A, 59H  ; load Y into accumulator
DONE: OUT 03H       ; send answer to port 3
        HLT
```