Unit-1

Introduction:

Microprocessor:-

A microprocessor is a multi-purpose, programmable, clock-driven, register based, electronic device that reads binary instructions from a storage device called memory. It accepts binary data as input and process data according to those instructions and provides result as output.

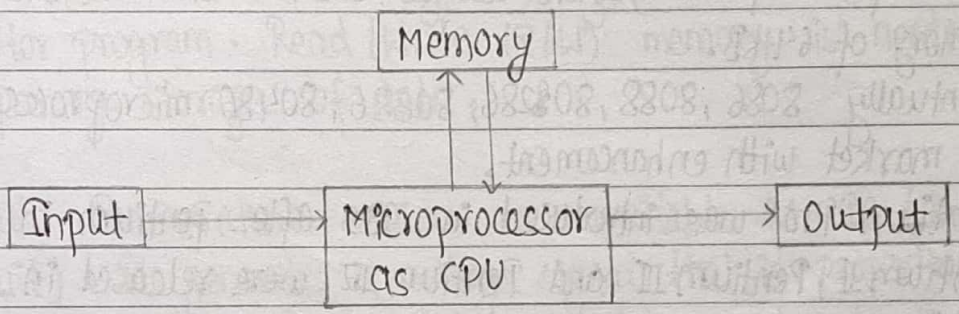


Fig: Block diagram of a computer with microprocessor as CPU.

We can also view the microprocessor as a primary component of a computer. The block diagram shows that the computer has four components:

i) Input Unit
ii) Output Unit
iii) Memory
iv) CPU

Evolution of microprocessor:

- In 1969, Intel engineer Ted Hoff introduced a 64 bit bipolar RAM chip.
- Intel released first 4 bit microprocessor as 4004 with LSI technology which consists of 2300 transistor, 640 bytes of memory and 808 KHz Clock.
- Gordan Moore; Co-founder of Intel corporation predicted that the no. of transistors per IC would double every 18 months. This is called Moore's law.
- Intel 4004 was quickly replaced by 8 bit microprocessor 8008.
- In 1976, 8085 microprocessor was introduced with 6500 transistor, clock speed of 5MHz, 16 bit address bus, 8 bit data bus and memory of 64 KB.
- Eventually 8086, 8088, 80286, 80386, 80486 microprocessor came in market with enhancement.
- Pentium Pro was introduced in 1995 after pentium in 1993.
- Pentium II, Pentium III and Pentium IV were released in 1997, 1999 and 2000 respectively. Now we have core generation with multiple core processor within a single chip.

Application: Microprocessor Controlled Temperature System (MCTS):

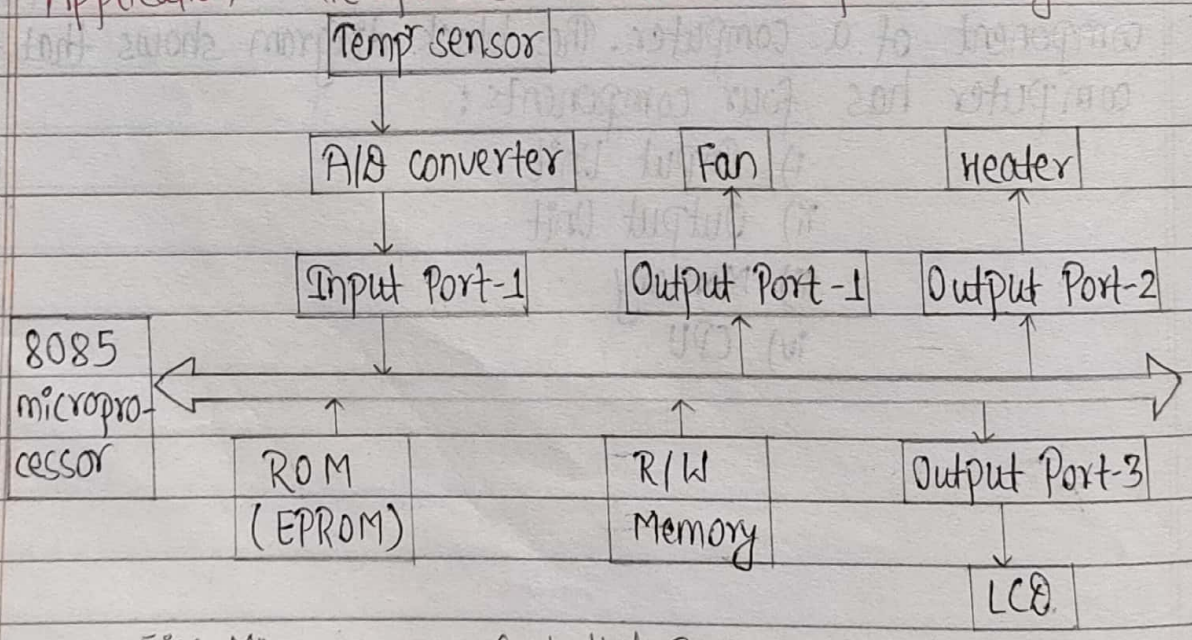


Fig: Microprocessor Controlled Temperature System

This system is expected to read the temperature in a room, display the temperature at a Liquid Crystal Display (LCD) panel, turn on a fan if the temperature is above a set point, and turn on a heater if the temperature is below a set point.

Different components are:

I) Microprocessor: The processor will read the binary instructions from memory and execute those instructions continuously.

II) Memory: ROM (Read Only Memory) will be used to store the monitor program. Read/Write (R/W) memory is needed for temporary storage of data.

III) Input: Temperature sensors are available as IC (Integrated Circuit). It generates a voltage signal that is proportional to the temperature. Voltage is analog signal so analog to digital converter (ADC) is required to convert the signal into digital bits.

IV) Output: Figure shows 3 output devices, fan, heater, LCD. Fan (Port-1) is turned on by the processor when temperature reaches higher limit. Heater (Port-2) is turned on by the microprocessor when the temperature reaches a lower limit. LCD (Port-3) is used to display temperature.

Imp. Basic Organization of Microprocessor:

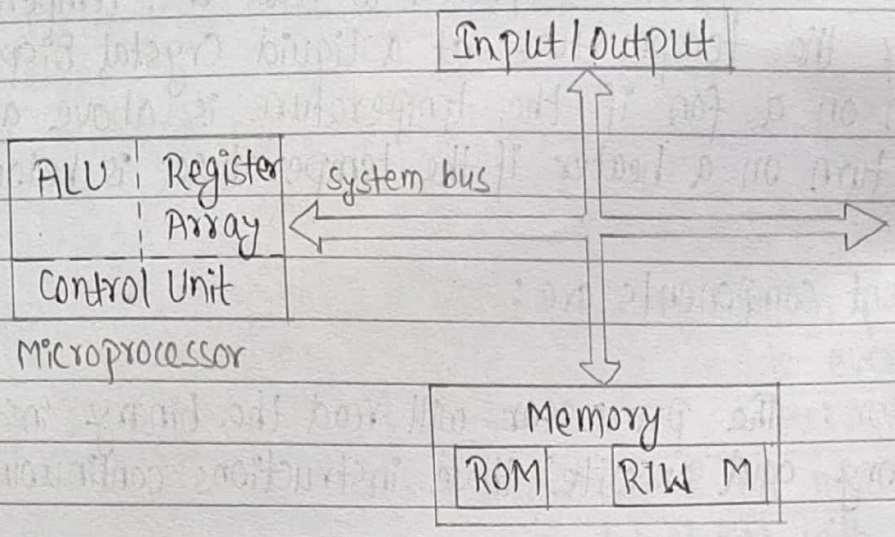


Fig: Microprocessor based system with bus architecture

Figure shows a simplified microprocessor based system. It includes 4 components: microprocessor, input/output, memory and system bus.

1) Microprocessor:

Microprocessor is a clock driven semi-conductor device consisting of electronic logic circuit. It can be divided into three segments:

- ALU (Arithmetic & Logic Unit)
- Register Array and,
- Control Unit (CU)

- ALU :- This is the area of microprocessor where various computing functions are performed on data. ALU perform arithmetic operations as addition and subtraction, and logical operations as OR, AND, exclusive OR, etc.

- **Register Array :-**
  It consists of various registers identified by letters such as B, C, D, E, H and L. These registers are primarily used to store data temporarily during the execution of a program.

- **Control Unit :-**
  The control Unit (CU) provides the necessary timing and control signals to all the operations in the micro computer.

ii) **Memory:**
To execute programs, the microprocessor reads instructions and data from memory and performs the computing operations in its ALU sections.
- ROM is used to store programs that do not need alternations. Program stored in the ROM can only be read.
- R/W memory is also known as user memory. The information stored in this memory can be easily read and altered.

iii) **Input/Output:**
I/O communicates with the outside world. Input devices includes keyboard, switch, analog to digital converters, etc. Output devices includes devices such as LED (Light Emitting Diode), a cathode Ray Tube (CRT), a printer, plotter, digital to analog converter, etc.

iv) **System bus:** The system bus is a communication path between the microprocessor and peripherals (I/O). It is a group of wires to carry bits. In 8085 microprocessor:
- 16 bit unidirectional address bus is used to send out memory address.
- 8 bit bidirectional data bus is used to transfer data.
- 8 bit control bus for timing signals.

Unit-2

Basic Computer Architecture

SAP-1 Architecture:

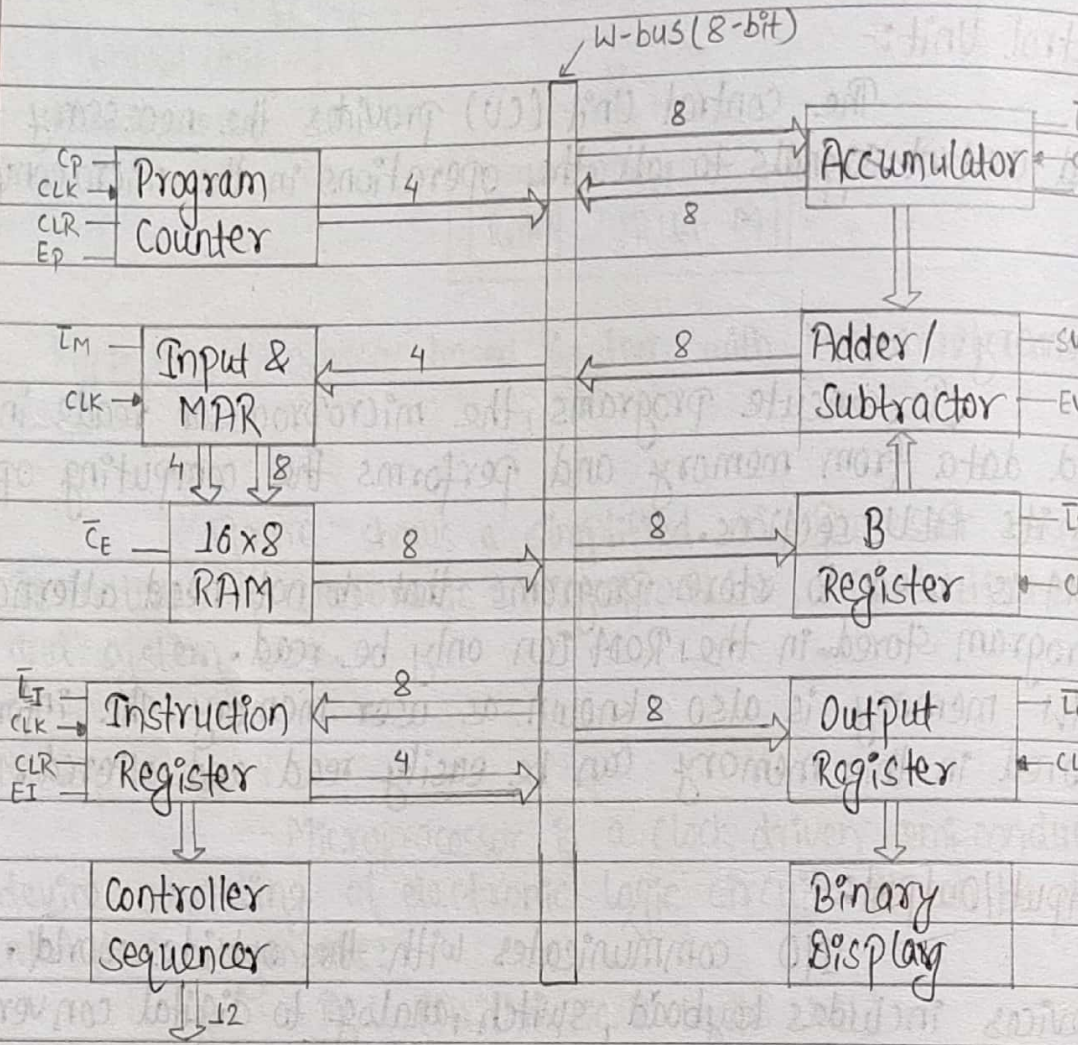


Fig: SAP-1 architecture

Figure shows the architecture of SAP-1 (Simple as possible-1), a bus organized computer. SAP-1 computer has been designed for the beginners. The main purpose of SAP-1 is to introduce all the crucial ideas behind computer operation without burying or in unnecessary details.

Each block are explained below:

- W-bus (8-bit):

All register outputs to the W-bus. It is 8-bit with three states which allows orderly transfer of data.

- Program counter (4-bit):

The program is stored at the beginning of the memory with the first instruction at the binary address 0000, 2nd at 0001, 3rd at 0010 and so on. PC sends to the memory at the address of the next instruction to be fetched and executed.

- Input & MAR:

Below the PC is the input and MAR block. Memory address register (MAR) allows to send 4 address bit and 8 data bit to the RAM. During a computer run, the address in the PC is latched (passed) into the MAR. Later the MAR applies this 4 bit address to the RAM, where a read operation is performed.

- RAM (16x8 memory):

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

- 8-bit Instruction Register (IR):

It is part of the control unit. After memory read operation the contents of the addressed memory location are placed on the W-bus. Contents of IR are split into two nibbles. Upper nibble goes to controller sequencer and lower nibble is read onto the W-bus when needed.

- **8-bit accumulator:**

The accumulator (A) is a buffer register that store intermediate answers during a computer run. It has two output; one goes to adder / subtractor and another goes to W-bus.

- **8-bit B register:**

B register is another buffer register. It is used in arithmetic operation. A low L_B and positive clock edge load the word on the W-bus into the B-register.

- **8-bit adder/subtractor:**

SAP-1 uses a 2's complement adder/subtractor. When S_u is low, the sum out of the adder/subtractor is

$$A = A + B$$

When S_u is high, the difference appear as:

$$A = A + B$$

When E_u is high, those contents appears on the W-bus.

- **8-bit output register:**

The accumulator contains the answer of the problem being solved. Output register are used to transfer the answer to the outside world. When E_A is high and L_o is low, the next positive clock edge loads the accumulator word into the output register. Output register is often called an output port.

SAP-1 instructions:

1. LDA :-

LDA stands for Load the accumulator. It includes the hexadecimal address of the data to be loaded.

Eg: LDA 8H

It means load the accumulator with the contents of memory location 8H.

If 8H contains, R_8 = 11110000

Execution of LDA 8H result in

A = 11110000

2. ADD :-

It includes the address of the word to be added.

Eg: ADD 9H

It means add the contents of memory location 9H to the accumulator contents and store the result in accumulator.

If A = 00000010 (decimal 2)

9H (R_9) = 00000011 (decimal 3)

First R_9 is loaded in register is

i.e. B = 00000011

Finally sum = 00000101

Sum is loaded in accumulator.

i.e. A = 00000101

3. SUB :-

A complete SUB instruction includes the address of the word to be subtracted.

Eg: SUB CH

It means subtract the contents of memory location CH from the contents of the accumulator and replace the original contents of accumulator with the result.

A = 00000111 (decimal 7)
R_C i.e. B = 00000011 (decimal 3)
finally A = 00000100 (after SUB CH)

4. OUT:-
OUT transfer the accumulator contents to the output port.
Eg: OUT
It is complete by itself. So, address is not required.

5. HLT:-
HLT stands for halt. It tells the computer to stop processing data. We must use a HLT instruction at the end of every SAP-1 program.
Eg: HLT
It doesnot require address.

Example (Program)

Address	Mnemonics	Meaning
0H	LDA 9H	Load accumulator with content of 9H
01H	ADD AH	Add accumulator with content of AH
2H	ADD BH	Add accumulator with content of BH
3H	SUB CH	Subtract accumulator with content of CH
4H	OUT	Load accumulator content to output port.
5H	HLT	Halt the program.

Address	Data
9H	01H
AH	02H
BH	03H
CH	04H

i) LDA 9H

A = 01H

ii) ADD AH

A = 01+02 = 03H

iii) ADD BH

A = 03+03 = 06H

iv) SUB CH

A = 06-04 = 02H

v) OUT:

Load accumulator contents to output port or binary display i.e. 00000010.

Fetch cycle:

The control unit generates the control words that fetch and execute each instruction while each instruction is fetch and executed. The computer passes through different timing states (T states) periods during which register contents change.

Ring counter:

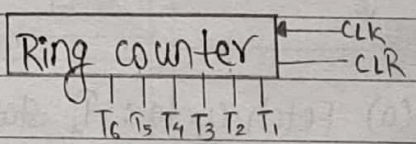


Fig symbolizes the ring counter which has an output of $T = T_6 T_5 T_4 T_3 T_2 T_1$

Symbol of ring counter:

At the beginning of a computer run the ring word is $T_1 = 000001$.

Successive clock pulse ring word is

$T_2 = 000010$
$T_3 = 000100$
$T_4 = 001000$
$T_5 = 010000$
$T_6 = 100000$

Then, the ring counter resets to 000001 and cycle repeats.

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). In figure, active parts are left blank and inactive parts are dotted as show:

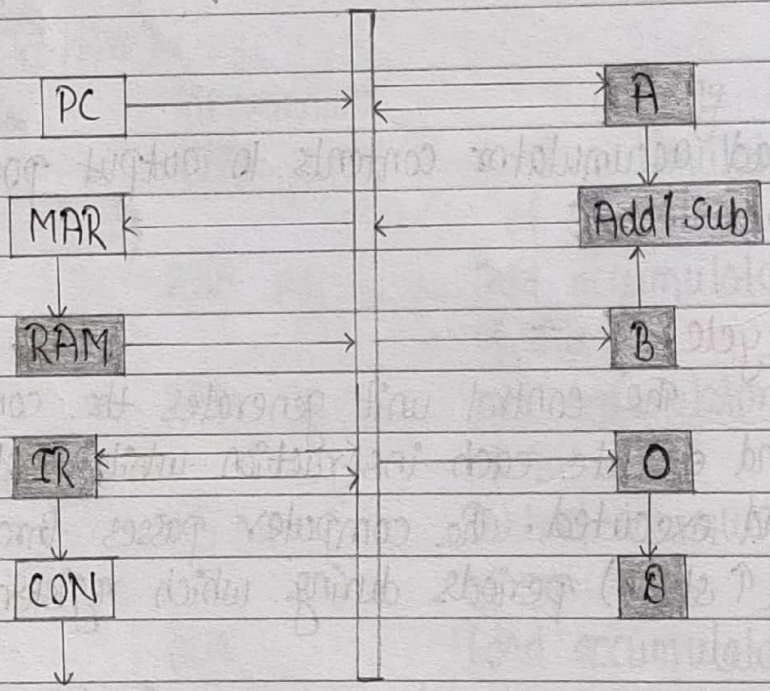
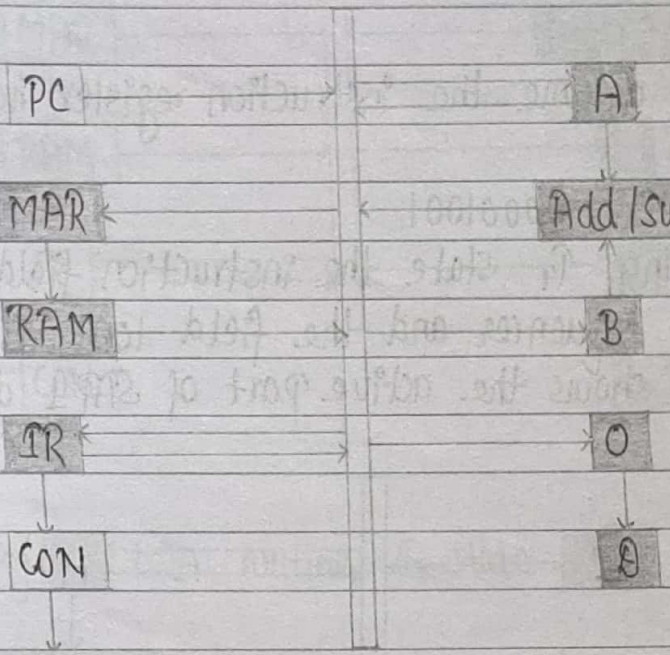


Fig (a) Fetch cycle: T_1 state

Increment state:

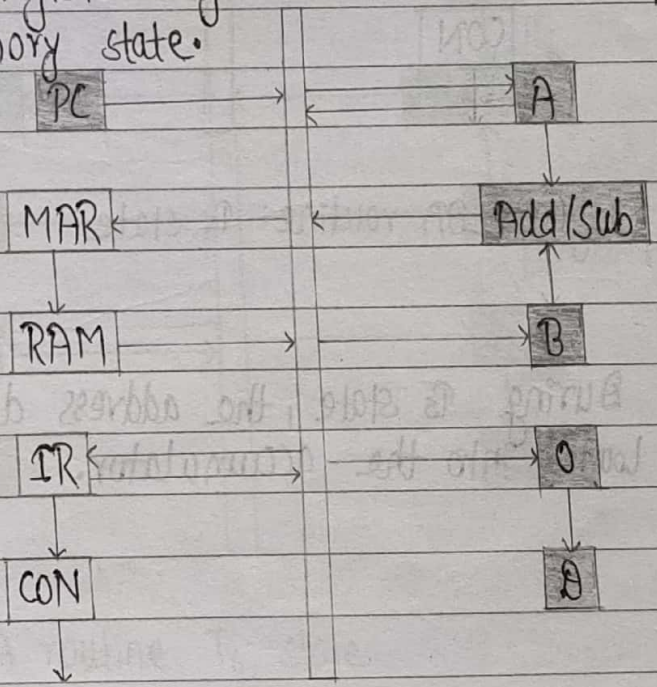
Fig(b) shows the active part of SAP 1 during the T_2 state. This state is called increment state because the PC is incremented.



Fig(b) Fetch cycle: T_2 state

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. Fig(c) shows the active part of SAP 1 during the memory state.



Fig(c) Fetch cycle: T_3 state

Execution cycle:

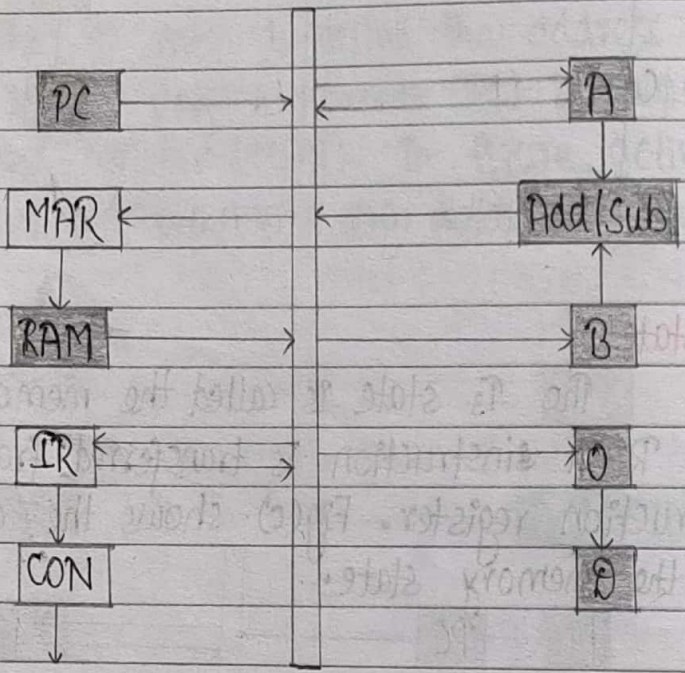
The next three states (T_4, T_5, T_6) are the execution cycle of SAP 1.

LDA Routine:

Let's assume the instruction register has been loaded with LDA 9H.

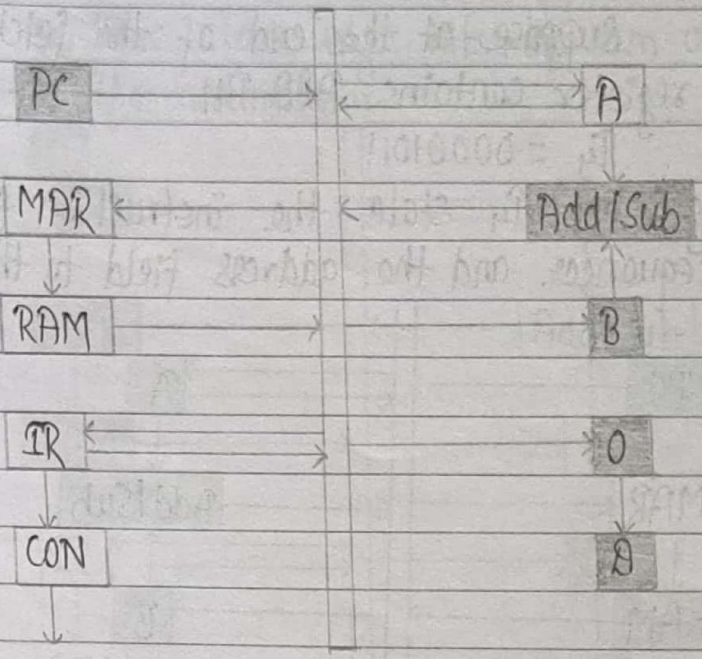
IR = 00001001

During T_4 state the instruction field 0000 goes to the controller sequencer and the field 1001 is loaded into the MAR. Fig(a) shows the active part of SAP 1 during the T_4 state.



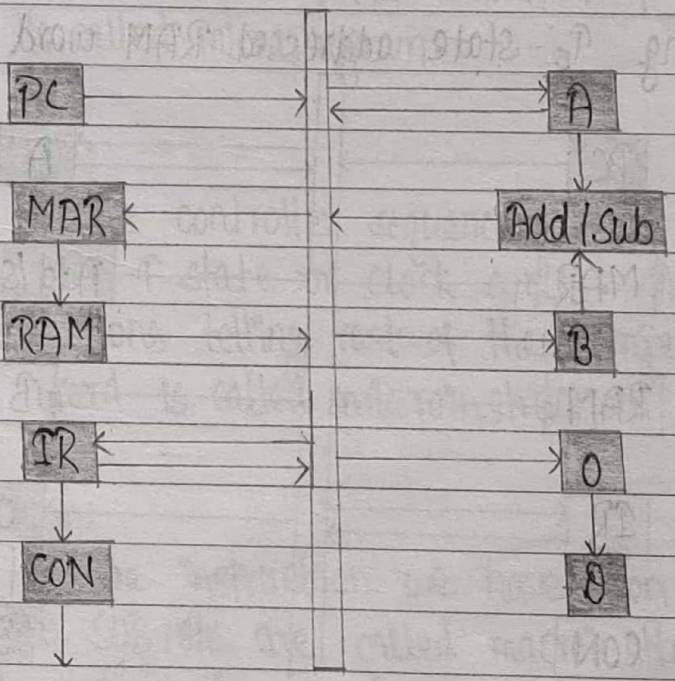
Fig(a) LDA routine: T_4 state

During T_5 state, the address data word in the RAM will be loaded into the accumulator.



Fig(b) LDA routine T_5 state

The T_6 state of LDA routine is a no operation state. During this state, all register are inactive.



Fig(c) LDA routine T_6 state

Add / SUB instruction:

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

T_4 = 00001011

During the T_4 state the instruction field goes to the controller sequencer and the address field to the MAR.

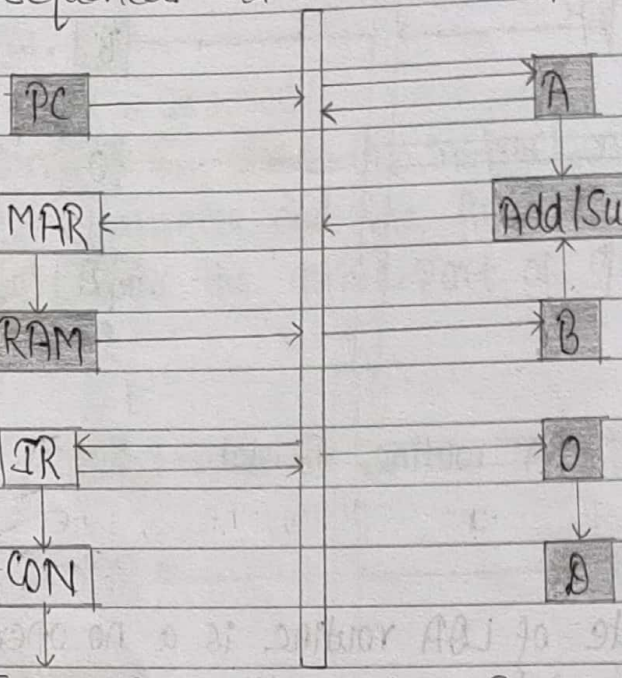


Fig (a): ADD / SUB routine T_4 state

During T_5 state addressed RAM word setup the register.

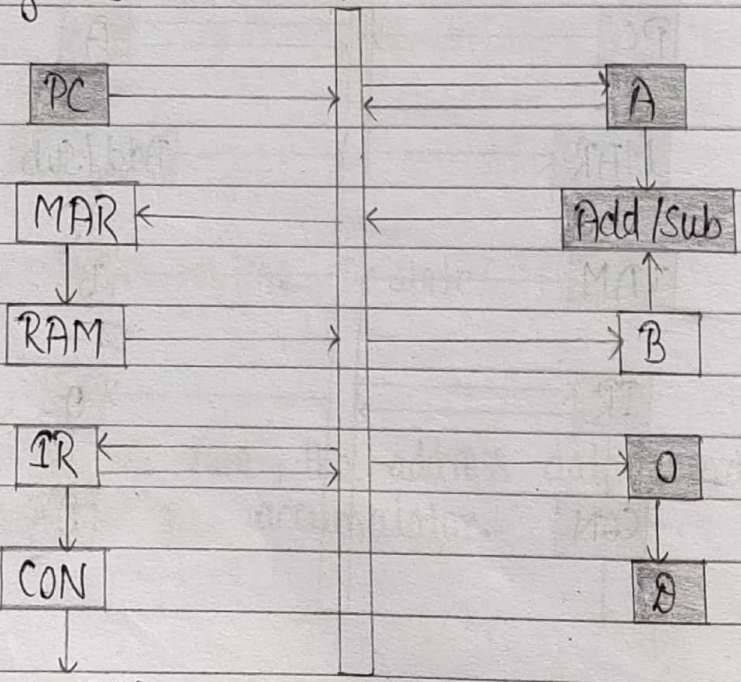


Fig (b): ADD and SUB routine T_5 state

During T_4, T_5 states adder/subtractor setup the accumulator. When the positive clock edge hits, the sum out of adder/subtractor is stored in the accumulator.

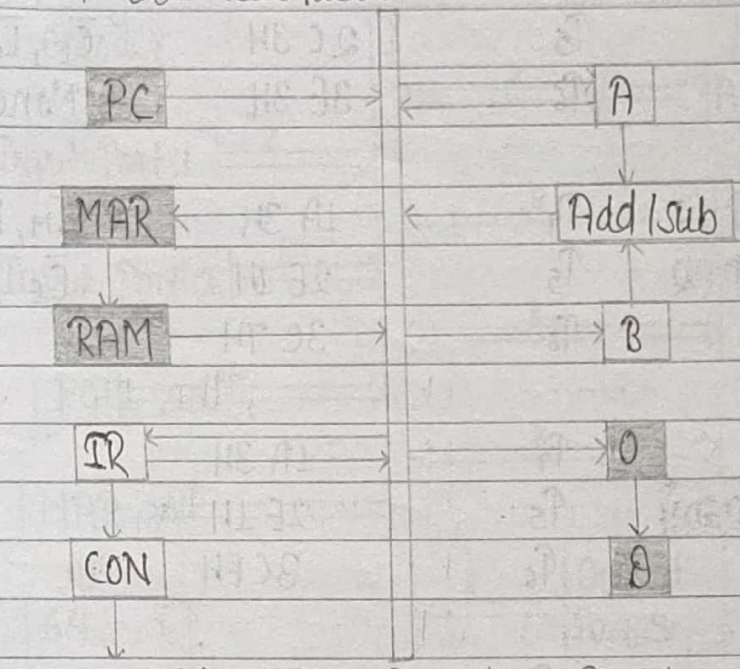


Fig (c): ADD and SUB routine T_6 state

Microprogram:

Summarization of the execution of SAP 1 instruction in a table is called microprogram.

Microinstruction:

The controller sequencer sends out control words, one during each T state or clock cycles. Those control words are like directions telling rest of the computer what to do. Each control word is called microinstruction.

Macroinstruction:

The instruction we have been programming with like LDA, ADD, SUB, etc are called macroinstructions. Each SAP 1 macroinstruction is made up of 3 microinstruction.

Macro	State	CON	Active
LDA	T_4	1A 3H	$\overline{L_M}$, $\overline{E_I}$
	T_5	2C 3H	$\overline{C_E}$, $\overline{L_A}$
	T_6	3E 3H	None
ADD	T_4	1A 3H	$\overline{L_M}$, $\overline{E_I}$
	T_5	2E 1H	$\overline{C_E}$, $\overline{L_B}$
	T_6	3C 7H	
SUB	T_4	1A 3H	
	T_5	2E 1H	
	T_6	3CFH	
OUT	T_4	3E 2H	
	T_5	3E 3H	
	T_6	3E 3H	

SAP-2 Architecture:

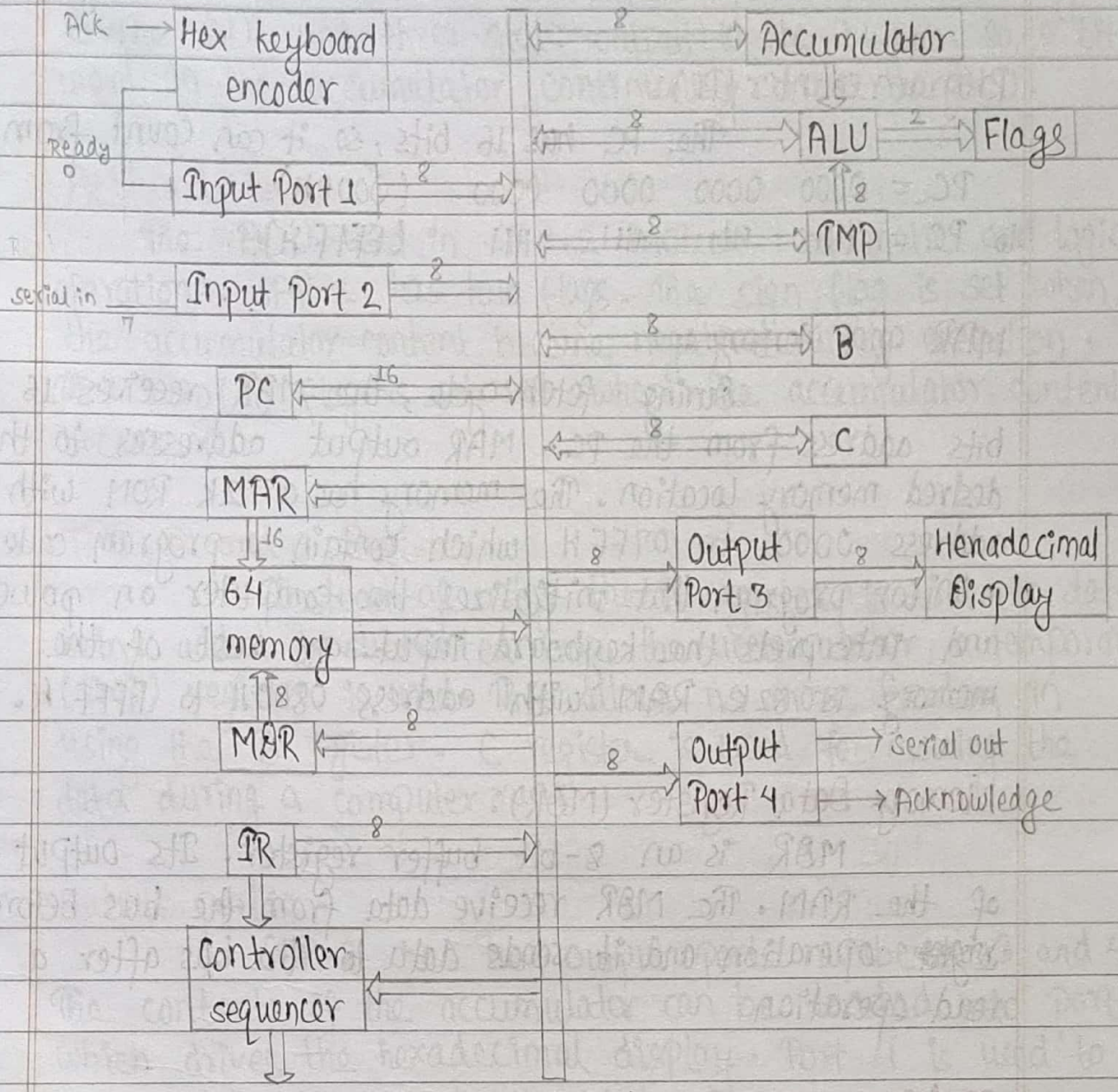


Fig: SAP-2 Architecture

SAP-2 is the next step in the evolution toward modern computer because it included jump instructions. These new instruction force the computer to repeat or skip part of a program. A brief description of each block is given here:

Input ports:

SAP-2 has 2 input ports numbered 1 and 2. A

hexadecimal keyboard encoder is connected to port 1. It allows us to enter hexadecimal instructions and data through port 1.

Program counter (PC):

The PC has 16 bits, so it can count from

PC = 0000 0000 0000 0000 (0000H)

to PC = 1111 1111 1111 1111 (FFFFH)

MAR and Memory:

During fetch cycle, the MAR receives 16 bits address from the PC. MAR output addresses to the desired memory location. The memory has a 2k ROM with address 0000 to 07FFH which contain a program called monitor program that initializes the computer on power up and interprets the keyboard input. The rest of the memory is 62k RAM with address 0800H to FFFFH.

Memory Data Register (MDR):

MDR is an 8-bit buffer register. Its output sets of the RAM. The MDR receive data from the bus before a write operation and it sends data to the bus after a read operation.

IR (Instruction Register):

We use 8 bit for the opcode rather than 4-bit as in SAP-1. So, it can accomodate upto 256 instructions but it has only 42 instruction.

Controller Sequencer (CON):

The CON produces the control words or micro instructions that coordinates and direct the rest of the computer.

Accumulator:

The two states output of the accumulator goes to the ALU and three state output to the W-bus. So, 8 bit word in the accumulator continuously drives the ALU.

ALU and Flags:

The ALU used in SAP-2 include arithmetic and logic operations. SAP 2 has two flags. The sign flag is set when the accumulator content become negative during execution. The zero(0) flag is set ~~set~~ when the accumulator content become 0.

TMP, B and C Register:

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. C register is used for moving the data during a computer run.

Output ports:

SAP-2 has two output ports numbered 3 and 4. The contents of the accumulator can be loaded into port 3 which drives the hexadecimal display. Port 4 is used to give serial output and acknowledgement.

Instruction sets:

1. Memory reference instructions:

i) LDA:

LDA has the same meaning as before i.e. load the accumulator with the addressed memory data.

Eg: LDA 2000H

It means to load the accumulator with the contents of memory location 2000H.

ii) STA:

STA is a mnemonic to store the accumulator content in memory location.

Eg: STA 7FFFH

It means to store the accumulator contents at memory location 7FFF H.

iii) MVI:

MVI is the mnemonics for move immediate. It tells the computer to load a designated register with the byte that immediately follows the opcode.

immediate data for read data.

Eg: MVI A, 37H

- It tells the computer to load the accumulator with 37H.
- We can use MVI with A, B and C register.

2. Register Instructions:

iv) MOV:

Mov is the mnemonic for move. It tells the computer to move data from one register to another.

Eg: Mov A, B

It moves the data in the register B to accumulator.

v) ADD and SUB:

ADD stands for addition of the data in the designated register to the accumulator.

Eg: ADD B

SUB means subtract the data in the register from the accumulator.

Eg: SUB C

vi) INR and DCR:

- INR is the mnemonic for increment.
- It tells the computer to increment the designated register.
- DCR is the mnemonic for decrement and it instructs the computer to decrement the register.

Eg: INR A
DCR B

3. JUMP and CALL instruction:

SAP-2 has 4 jump instructions. These instructions can change the program sequence.

i) ~~vii)~~ JMP:

It tells the computer to get the next instruction from the designated memory location. Every JMP instruction includes an address that is loaded in the PC.

Eg: JMP 3000H

ii) ~~viii)~~ JM:

SAP 2 has two flags called the sign flag and the zero flag. Sign flag is set if result on accumulator is negative. JM is a mnemonic for jump if minus. The computer

execution will jump to a designated address if and only if the sign flag is set.

Eg: JM 30FF H

ix) JZ:

zero flag is set if the result on accumulator is zero. JZ is the mnemonic for jump if zero. It tells the computer to jump to the designated address only if the zero flag is set. JZ

Eg: JZ 4000H

x) JNZ:

JNZ stands for jump if not zero. We get a jump when the zero flag is clear and no jump when it is set.

Eg: JNZ 6000H

xi) CALL and RET:

A sub routine is a program stored in a the memory for possible use in another program. CALL is the mnemonic for calling the sub routine. Every CALL instruction must include the starting address of the desired sub routine.

RET stands for return. It is used at the end of every sub routine to tell the computer to go back to the original program.

Program:

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.

⇒

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

2. Show the mnemonic for adding 23 and 45 (decimal). The answer is to be stored at memory location 5600H. Also the answer incremented by 1 is to be stored in the C register

⇒

```
MVI A, 17H   (23 in decimal)
MVI B, 2DH   (45 in decimal)
ADD B
STA 5600H
INR A
MOV C,A
HLT.
```

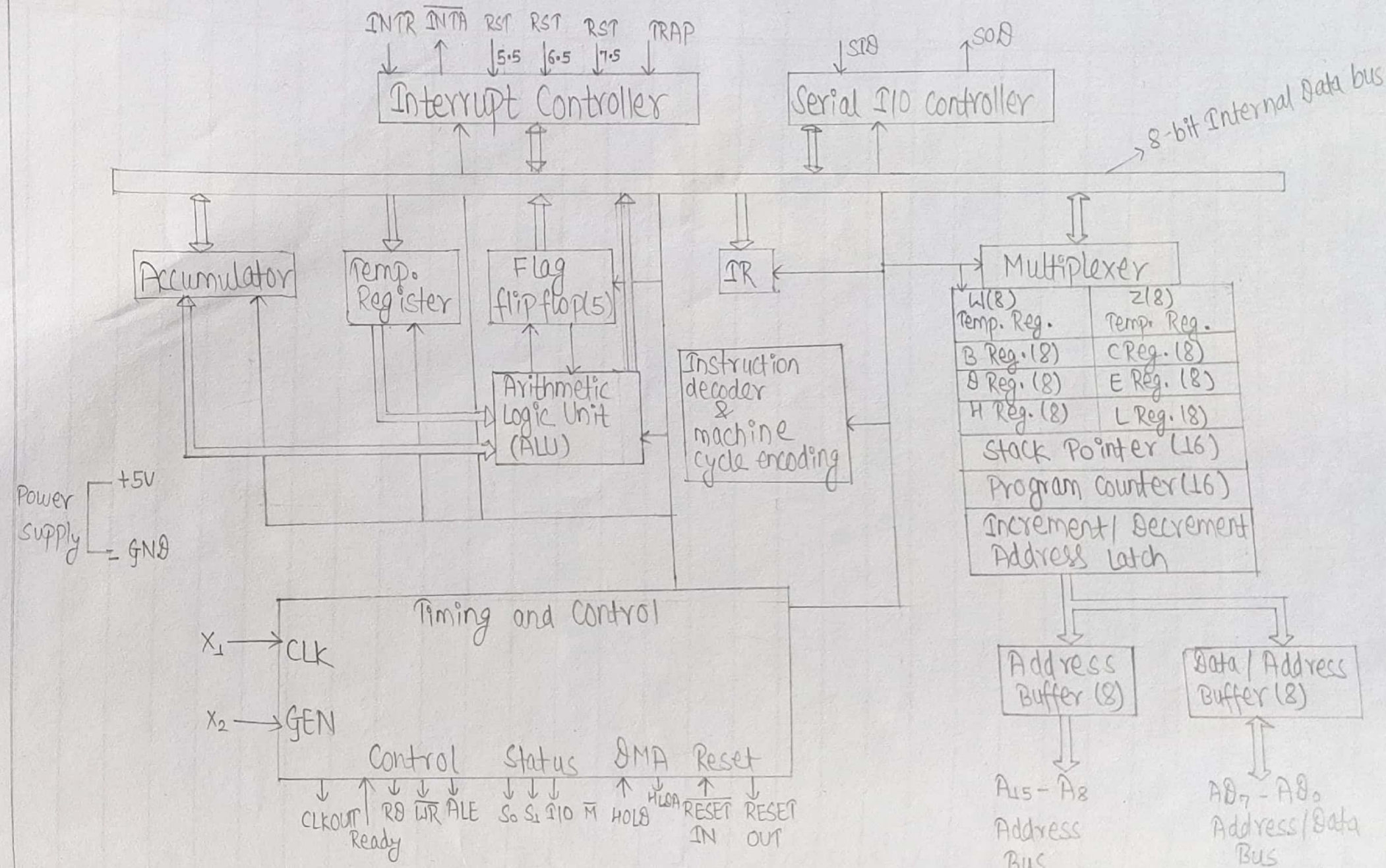


Fig: Block diagram of intel 8085 microprocessor

Temporary Register:

- It is used to hold data during arithmetic and logic operation.
- The next input for the ALU comes from temporary register.
- Temporary register receives the data to be processed by ALU from general purpose register.

Accumulator:

- It is an 8-bit register.
- It stores the data to be processed by ALU and the result of the operation.
- Accumulator is connected to 8-bit internal data bus.

General Purpose Register:

- In 8085 microprocessor there are six 8-bit general purpose register. They are B, C, D, E, H and L register.
- Two temporary register W and Z are used internally and are not available to the programmer.
- They can be used individually or combined as a pair to perform 16-bit operation.
- The HL-register pair is usually used as 16-bit memory pointer.

Stack Pointer (SP):

- Stack pointer is a 16-bit register used as a memory pointer.
- It maintain the memory of last byte entered into the stack.
- Stack is position of RAM.

Program Counter:

- This is 16-bit register which deals with sequencing the execution of instructions.
- It points the address of next instruction to be executed.

Incrementer/Decrementer:

- It increments or decrements one from the contents of PC or stack pointer.

ALU:

- The ALU carries out the arithmetic and logic operation on 8 bit words.
- As shown the contents of the accumulator and temporary register are input to the ALU.
- It performs arithmetic operations like addition, subtraction and logic operation like AND, OR, XOR. ALU result is stored back to accumulator.

Flag:

- Flag register is a group of 5 individual flip flops.
- The contents of flag register will change 0 or 1 after the execution of arithmetic and logic operation.

i) S (sign) flag:

If bit D_7 of result is 1, the sign flag is set i.e. the no is negative. If it is 0, the number will be considered positive.

ii) Z (zero) flag:

Z (zero) flag is set if the ALU operation result is zero.

iii) AC (Auxiliary carry) Flag:
When a carry is generated by digit D_3 and passed onto digit D_4 then AC flag is set.

iv) Parity flag:
If the result has an even no. of 1's, the flag is set.

v) CY (carry) flag:
If result has a carry, the carry flag is set.

D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
S	Z		AC		P		CY

Instruction register and decoder:
- IR and decoder is a 8-bit register.
- When the instruction is fetched from memory it is loaded in IR.
- The instruction decoder decodes the contents of instruction register.
- It also determines the operation to be followed in executing the entire instruction and drives the timing and control unit accordingly.

Timing and Control unit:
- Timing and control unit of microprocessor consists of oscillator and controller sequencer.
- The oscillator generates 2 phase clock signal clk and $\overline{CLK}$ that synchronize all register.
- It is microprogrammed and consists of ROM that stores all the micro-routines needed for executing the instructions.

Interrupt Control:

- Sometimes it is important to interrupt the execution of main program to answer the request from input/output device.
- For example: an input/output device may send an interrupt signal to interrupt control unit to indicate data is ready for input.
- The computer temporarily stop the execution of main program input the data and returns to the main program.

Serial I/O control:

- Sometimes, I/O devices work with serial data instead of parallel.
- The serial input of data is done through pin 5 and serial output through pin 4.

Address/data buffer and address:

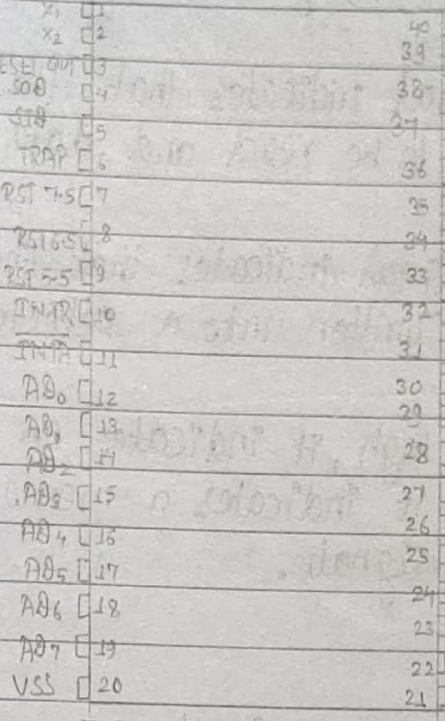
- The address of PC and stack pointer is loaded into address buffer and address/data buffer.
- The o/p of these buffer drives the external address/data bus.

Pin configuration:

The 8085 is an 8-bit general purpose microprocessor capable of addressing 64 K of memory. It has 40 pins, requires +5 V single power supply and can operate with a 3MHz single phase clock.

Fig (a) shows the logic pin out of the 8085 microprocessor. All the signals can be classified into six groups.

SOD - Serial Output Data
SID - Serial Input Data



Fig(a): Pin configuration

i) Address bus
ii) Data bus
iii) Control and status signal
iv) Power supply and frequency signal
v) Externally initiated signals
vi) Serial I/O ports.

Address bus and data bus:

8085 has 16 pins that are used as the addressed bus. The eight pins A_{15} - A_8 are unidirectional higher order address bus. The signal lines AD_7 - AD_0 are used for dual purpose i.e. they are used as the low order address bus as well as the data bus. So, AD_7 - AD_0 is multiplexed address/data bus.

Control and status signals:

These include two control signals $\overline{RD}$ and $\overline{WR}$, three status signals IO/$\overline{M}$, S_1, S_0 and one special signal ALE.

- ALE (Address Latch Enable) generates a separate set of eight address lines A7-A0.
- $\overline{RD}$ (Read) control signal indicates that the selected I/O or memory device is to be read and data are available on the data bus.
- $\overline{WR}$ (Write) control signal indicates that the data on the data bus are to be written into a selected memory or I/O location.
- I/O $\overline{M}$: When it is high, it indicates an I/O operation and when it is low, it indicates a memory operation.
- S_1 and S_0 are status signals.

Power supply and frequency signals:

V_{cc} : +5V power supply
V_{ss} : Ground reference
$X_1 X_2$: for frequency setting usually 3 MHz.
CLK (OUT): used as the system clock for other devices.

Externally initiated signals including interrupts:

INTR (Input) : Interrupt request
INTA (Output): Interrupt acknowledgement
RST 7.5 (Input) : Restart interrupt
RST 6.5
RST 5.5
TRAP (Input) : Non-maskable interrupt
HOLD (Input): Request to hold address and data buses by DMA (Direct Memory Address)
HLDA (Output) : Hold acknowledgement
READY : Used to delay read and write cycle.
$\overline{RESET\ IN}$: To reset up i.e. set PC to zero.
RESET OUT : Used to restart other device.

Serial I/O ports:

SID : Serial input data

SOD : Serial output data

Instruction word size:

The 8085 instruction set is classified into three groups according to word size or byte size.

i) 1-byte instruction

Task	Opcode	Operand
- Copy the contents of the accumulator in register C.	MOV	C, A
- Add the contents of register B to the contents of accumulator.	ADD	B
- Invert each bit in the accumulator	CMA	

ii) 2-byte instruction

Task	Opcode	Operand
- Load 8-bit data byte in the accumulator	MVI	A, 32H
- Load 8-bit data byte in register B	MVI	B, F2H

iii) 3-byte instruction

P.T.O

Task	Opcode	Operand
- Load the contents of memory 2050H into accumulator.	LDA	2050H
- Transfer the program sequence to memory location 2085H.	JMP	2085H

Imp:

Addressing Modes:

Instructions are commands to the microprocessor to perform a certain task. The instructions consist of op-code and data called operand. The way operands are choosen during program execution is dependent on the addressing mode.

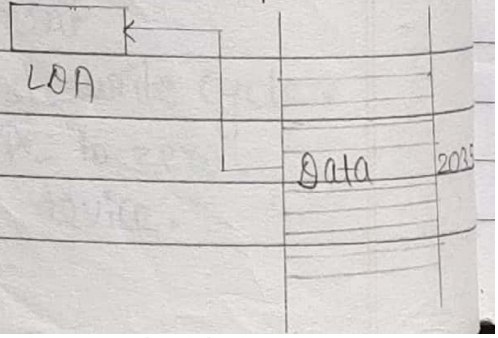
Different addressing modes are:

- Direct addressing
- Register indirect addressing
- Register direct addressing
- Implied direct addressing
- Immediate direct addressing

1, Direct Addressing:

This addressing mode is called direct because the effective address of the operand is specified directly in the instruction. Instructions using this mode may contain 2 or 3 bytes, with first byte as the op-code followed by 1 or 2 byte of the address of data.

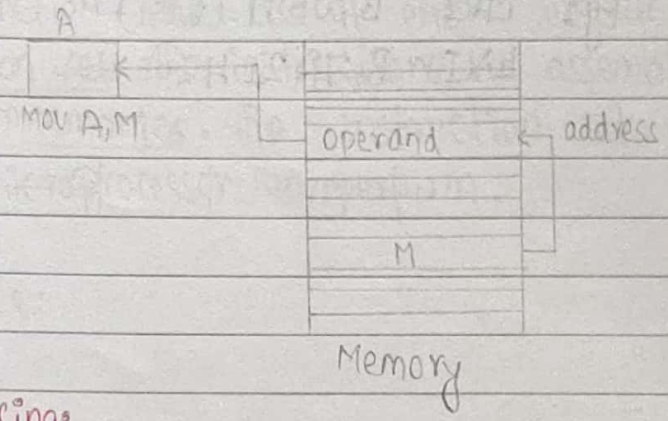
Eg: LDA 2035H
IN FCH



2. Register Indirect Addressing:

The address part of the instruction specifies the memory location whose contents is the address of operand. In 8085, whenever the instruction uses HL pointer the address is called indirect addressing.

Eg: MOV A,M.

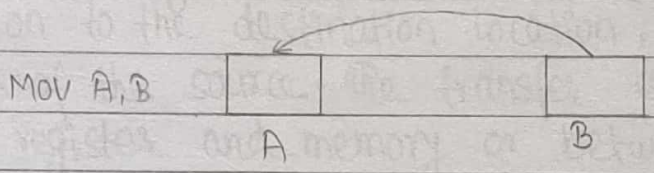


3. Register Direct Addressing:

Register direct addressing mode means that a register is the source of an operand for an instruction. It is similar to direct addressing.

For eg: MOV A,B

ADD B



4. Implied or Inherant Addressing:

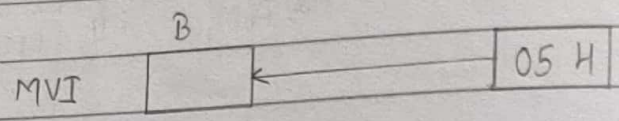
The instruction of this mode do not have operands.

Eg: EI (Enable Interrupt)
STC (Set the carry flag)
NOP (No operation)

5. Immediate Addressing:

This is the simplest form of addressing. When it executes the instruction will operate on immediate hexadecimal number. The operand may be 8 or 16 bit data.

Eg: MVI B, 05 H (8 bit)
LXI B, 7A21 H (16 bit)



Unit -5

Assembly Language Programming

Instruction set:

An instruction is a binary pattern designed to perform a specific function. The list of entire instruction is called the instruction set. The instructions in assembly language are called mnemonics. The instruction set determines what function the microprocessor can perform.

8085 instruction types:

- Data transfer group
- Arithmetic group
- Logical group
- Branching group
- Miscellaneous group

Data transfer group instruction:

This group of instruction copy data from a source location to the destination location, without modifying the contents of the source. The transfer of data may be between the register and memory or between an I/O device and accumulator. The data transfer instructions do not affect the flags.

	Opcode	Operand	Description
1)	MOV	R_d, R_s	Move - 1 byte instruction - copy data from source register R_s to destination register R_d

	Opcode	Operand	Description
2)	MVI	R, 8-bit	- Move immediate - 2 byte instruction. - load the 8 bit of the second byte into the register specified.
3)	OUT	8-bit port address	- output to port - 2 byte instruction - sends the content of accumulator (A) to the output port specified in the second byte.
4)	IN	8-bit port address	- Input from port - accepts data from the input port into the accumulation.
5)	HLT		- Halt - 1 byte instruction - processor stops execution and enters wait state.
6)	NOP		- No operation - 1 byte instruction. - No operation is performed generally used to increase processing time.

Example:

1. Load the accumulator A with the data byte 82H and save the data in register B.

sol^n:

Instruction	Description
MVI A, 82H	Load the accumulator with 82H.
MOV B, A	Copy the data from accumulator to register B
HLT	Halt

2. Write instruction to read eight ON/OFF switches connected to the input port with the address 00H and turn on the devices connected to the output port with the address 01H.

sol^n:

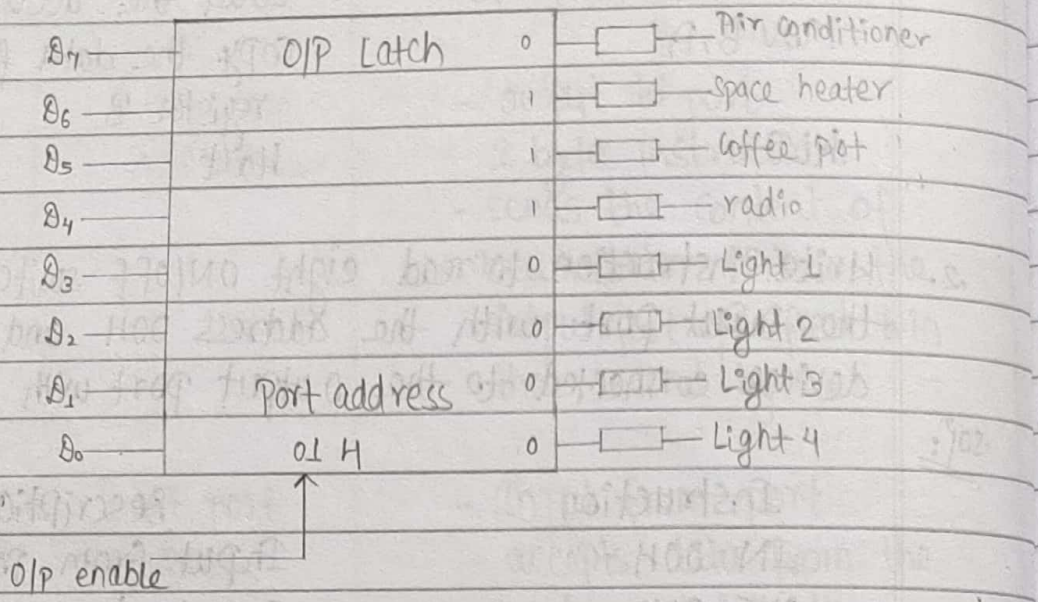
Instruction	Description
IN 00H	Input from port address given
OUT 01H	Output to port address given
HLT	Halt.

3. Load the hexadecimal number 37H in register B, and display the number at the output port labelled PORT 1.

sol^n:

Instruction	Description
MVI B, 37H	Load the register B with 37H.
MOV A, B	Copy the data to accumulator.
OUT PORT 1	Display at PORT 1.
HLT	Halt.

4. A microcomputer controls appliances in your house. It has O/P port 01H and various units are connected to the bit D_7 to D_0 as shown in figure. You want to turn on the radio, coffee pot and the space heater. Write appropriate instruction for the computer assuming R/W memory begins at 2000H.



Soln:

To turn on radio, coffee pot and the space heater set D_4, D_5 and D_6 at logic-1 and other bits at logic 0.

The output port requires 70H as;

D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
0	1	1	1	0	0	0	0
7				0			

It can be sent to the port by loading the accumulator with 70H.

P.T.O.

Program

Memory address	Instruction	Comments
2000	MVI A, 70H	Load the accumulator with bits required to turn on the devices.
2002	OUT 01 H	Send the bit pattern to the port 01H and turn on the devices.
2004	HLT	End the program

Arithematic operation:

	Opcode	Operand	Description
1)	ADD	R	- Add - 1 byte instruction - Add the contents of register R to the contents of accumulator
2)	ADI	8-bit	- Add immediate - 2 byte instruction - Add the second byte to the contents of accumulator.
3)	SUB	R	- Subtract - 1 byte instruction - subtract the contents of R from accumulator.
4)	SUI	8-bit	- Subtract immediate - 2 byte instruction - subtract the second byte from the contents of the accumulator.

	Opcode	Operand	Description
5)	INR	R	- Increment - 1 byte instruction - Increment the contents of register R by 1.
6)	DCR	R	- Decrement - 1 byte instruction - Decrement the contents of register R by 1.

Example 1:
The content of the accumulator is 93H and the content of register C is B7H. Add both contents.

Sol:

Program:

```
MVI A, 93H
MVI C, B7H
ADD C
HLT
```

	D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
A: 93H	1	0	0	1	0	0	1	1
C: B7H	1	0	1	1	0	1	1	1
	1	0	1	0	0	1	0	1 0

Sum (A) = 1 0 1 0 0 1 0 1 0

flag status → S=0, Z=0, CY=1

Example 2:
Add the member 35H directly to the sum in the previous example when the CY flag is set.

Sol^n:

Instruction: ADI 35H

```
A:   01001010
ADI: 00110101
     ---------
     01111111
```

Sum (A) = 01111111 (7F)
flag status S=0, Z=0, CY=0

Example 3:
Write a program that
i) Load the no. 8BH in register B.
ii) Load the no. 67H in register C.
iii) Increment the contents of register C by 1.
iv) Add the contents of register C and B and display the sum at the output port 1.

Sol^n:

Instruction:	Comment
MVI B, 8BH	Load the number in register B.
MVI C, 67H	Load the number in register C.
INR C	Increment C
MOV A, C	Copy the contents of C register to accumulator.
ADD B	Add the contents of accumulator with register B.
OUT port 1	Display the result at Port 1.
HLT	End the program

Finding result:

B = 8BH

C = 67H

INRC = 68H

MOV A, C A = 68H

ADD B

A = 68 = 0110 1000

B = 8B = 1000 1011

1111 0011

F 3

S = 0 , Z = 0 , CY = 0 , AC = 1

Example 4:

Write a program to

i) Load the number 30H in register B and 39 H in register C

ii) Subtract 39H from 30H

iii) Display the answer from PORT 1. R/W memory in your system begins at 2000H.

Sol?

Memory	Instruction	Comment
2000	MVI B, 30H	Load the number in register B
2002	MVI C, 39H	Load the number in register C
2004	MOV A, B	Copy the contents of register B in A
2005	SUB C	
2006	OUT PORT 1	
2008	HLT	

Finding result:

39 = 00111001

1's complement = 11000110

2's complement = 11000110 + 1

= 11000111

Add 30H = 11000111
00110000
[0] 11110111
CY F 7

Complement carry CY = 1
S = 1, Z = 0.

The number F7H is 2's complement of magnitude (30H - 39H) = -9H.
The instruction OUT displays F7H at PORT 1.

Logical operations:

The 8085 instruction set includes logic function as AND, OR, EXOR and NOT.

Opcode	Operand	Description
ANA	R	- Logically AND with accumulator - 1 byte instruction.
ANI	8-bit	- AND immediate with accumulator - 2 byte instruction
ORA	R	- logically OR with accumulator - 1 byte instruction
ORI	8-bit	- OR immediate with accumulator - 2 byte instruction

Opcode	Operand	Description
XRA	R	- logically exclusive OR with accumulator. - 1 byte instruction
XRI	8-bit	- Exclusive OR immediate - 2 byte instruction.
CMA		- complement the accumulator - 1 byte instruction.

Assume register B holds 93H and the accumulator holds 15H. Illustrate the result of the instruction.

i) ORA B
ii) XRA B
iii) CMA

Sol?

i) ORA B:

B = 10010011
A = 00010101
ORA B = 10010111
(9 7)

Accumulator (A) = 97H

ii) XRA B

B = 10010011
00010101
XRA B = 10000110
(8 6)

iii) CMA

A = 00010101

CMA = 1110 1010

E A

Branch operation:

The branch instructions are the most powerful instructions because they allow the microprocessor to change the sequence of a program either unconditionally or under certain test conditions. Jump instructions are classified into two categories:

i) Unconditional jump and

ii) Conditional jump

i) Unconditional jump:

The unconditional jump instruction enable the programmer to set up continuous loop.

Opcode	Operand	Description
JMP	16 bit	Jump - 3 byte instruction - 2nd & 3rd byte specify the 16-bit memory address

Eg:- To instruct the µp to go to memory location 2000H.

Sol:

Mnemonics

JMP 2000H

ii) Conditional jump:

Conditional jump instruction allow the microprocessor to make decision based on certain conditions indicated by the flags.

Opcode	Operand	Descriptions
JC	16-bit	Jump on carry (if CY=1)
JNC	16-bit	Jump on no carry (if CY=0)
JZ	16-bit	Jump on zero (if Z=1)
JNZ	16-bit	Jump on no zero (if Z=0)
JP	16-bit	Jump on plus (if D_7=0 & S=0)
JM	16-bit	Jump on minus (if D_7=1 & S=1)
JPE	16-bit	Jump on even parity (P=1)
JPO	16-bit	Jump on odd parity (P=0)

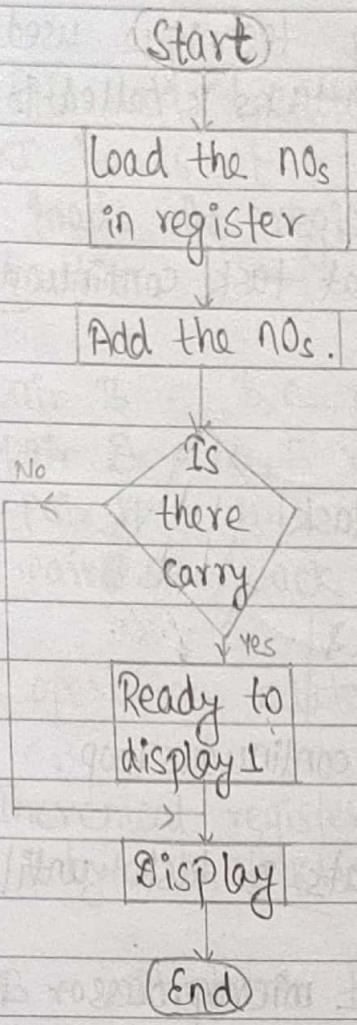
Example:

Load the hexadecimal number 98H and A7H in register B and E respectively and add the numbers. If the sum is greater than FFH, display 01H at output port 0, otherwise display the sum.

→

Algorithm:

Step 1: Load the numbers in register
Step 2: Add the number
Step 3: Check the sum
Step : Is the sum > FFH?
Step 4: If yes get ready to display 1.
If NO go to step5 to display the sum.
Step 5: Display
Step 6: Exit

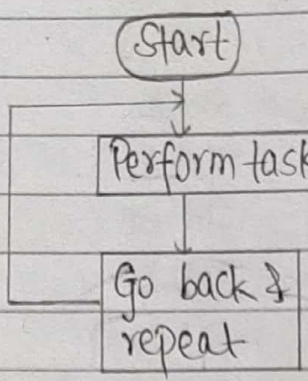


Memory	Label	Mnemonics
2000	START	MVI D, 98H
2002		MVI E, A7H
2004		MOV A, D
2005		ADD E
2006		JNC DISPLAY
2009		MVI A, 01H
200B	DISPLAY	OUT 00H
200D		HLT

Looping:-

The programming technique used to instruct the microprocessor to repeat tasks is called looping. Loops can be classified as:-

i) **Continuous loop:** Repeat task continuously.



Flowchart of continuous loop.

ii) **Conditional loop:** Repeats a task until certain data conditions are meet.

Eg:- How does the microprocessor repeat a task five times?

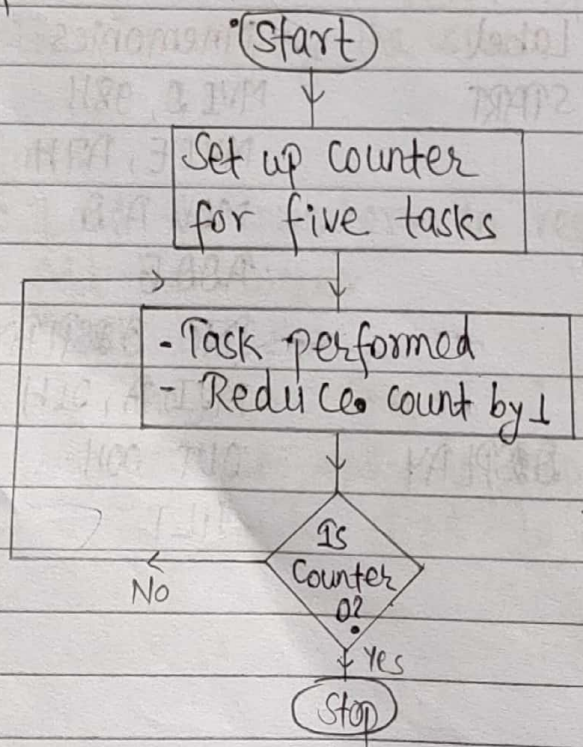


Fig: Flowchart of conditional loop.

Additional data transfer and 16-bit arithmetic instruction:

LXI: 16 bit data transfer to register pair.

Eg: LXI Rp, 16 bit

⇒ It loads the register pair. LXI is 3 byte instruction.

Register pair B = B, C register
Register pair D = D, E register
Register pair H = H, L register
Register pair SP = stack pointer register.

Arithmetic operation related to 16-bit or register pair:

1) **INX Rp:** - Increment register pair by 1.
   - 1 byte instruction

   Eg:
   INX B - It treats the content of two registers
   INX D as one 16-bit number and increase the content by 1.

2) **DCX Rp:** - Decrement register pair.
   Eg:- DCX B.

Example:
Write instruction to load the 16-bit number 2050H in the register pair HL using LXI and MVI opcode and explain the difference between the two instructions.

→

Mnemonics	Comment
LXI H, 2050H	Load 50H in L register and 20H in H register.

Now using MVI	Comment
MVI H, 20H	Load 20H in register H
MVI L, 50H	Load 50H in register L.

Example:

Write the instruction to load the number 2050H in the register pair BC. Increment the no. using the instruction INX B and illustrate whether the INX B instruction is equivalent to the instruction INR B and INR C.

→

Condition 1:

```
LXI B, 2050H
INX B
HLT
```

- Load register pair BC with 2050

20	50
B	C

- After increment

20	51
B	C

Condition 2:

```
LXI B, 2050H
INR B
INR C
HLT
```

- B will increase to 21
- C will increase to 51

21	51
B	C

So, we have to use only INR C to be equivalent with INX B as in example.

Data transfer (copy from memory to microprocessor):

1) MOV R,M:
- Move from memory to register
- 1 byte instruction
- memory location is specified by the contents of the HL register.

2) LDAX B/D:
- Load accumulator indirect
- 1 byte instruction
- Memory location is specified by the contents of the register BC or DE

3) LDA 16 bit:
- Load accumulator direct
- 3 byte instruction

Data transfer from the microprocessor to memory or directly into memory:

1) MOV M,R:
- Move from register to memory
- 1 byte instruction

2) STAX B/D:
- Store accumulator indirect
- 1 byte instruction
- To copy data from accumulator into the memory location specified by BC or DE register.

3) STA 16-bit:
- Store accumulator direct
- 3 byte instruction to copy data from accumulator into memory location given by 16-bit operand.

4) MVI M, 8-bit :
- Load 8-bit data to memory.
- Memory location is specified by the contents of HL register.

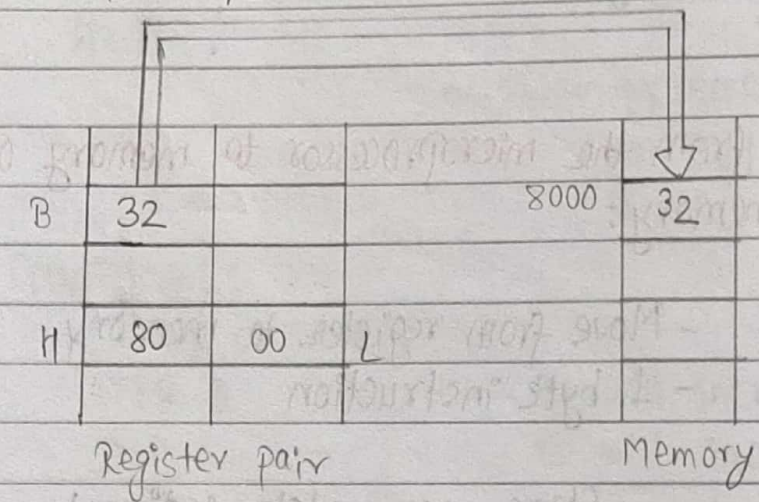
Example:

Register B contains 32H. Illustrate the instruction MOV and STAX to copy the contents of register B into memory location 8000 H using indirect addressing.

Sol:

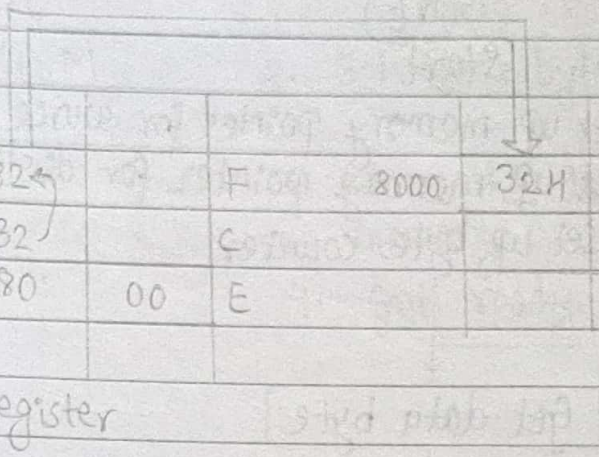
Mnemonics	Comments
LXI H, 8000H	Load 00 in L register and 80 in H register.

MOV M, B



STAX copies the contents of the accumulator into memory. Therefore, it is necessary to copy B into A. STAX works on BC or DE register only.

```
LXI B, 8000H
MOV A, B
STAX B
```

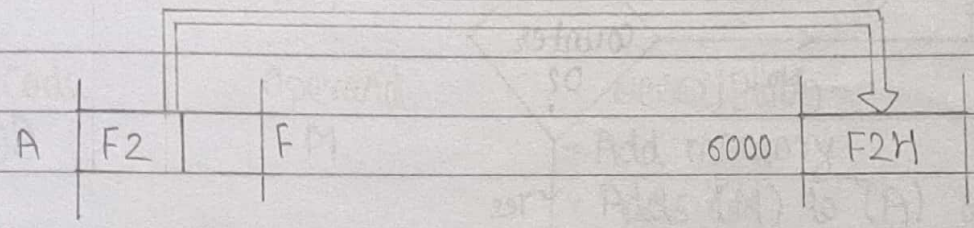


Example:

The accumulator contains F2H. Copy the content of accumulator into memory location 6000H using direct addressing.

Sol:

```
MVI A, F2H
STA 6000H
```

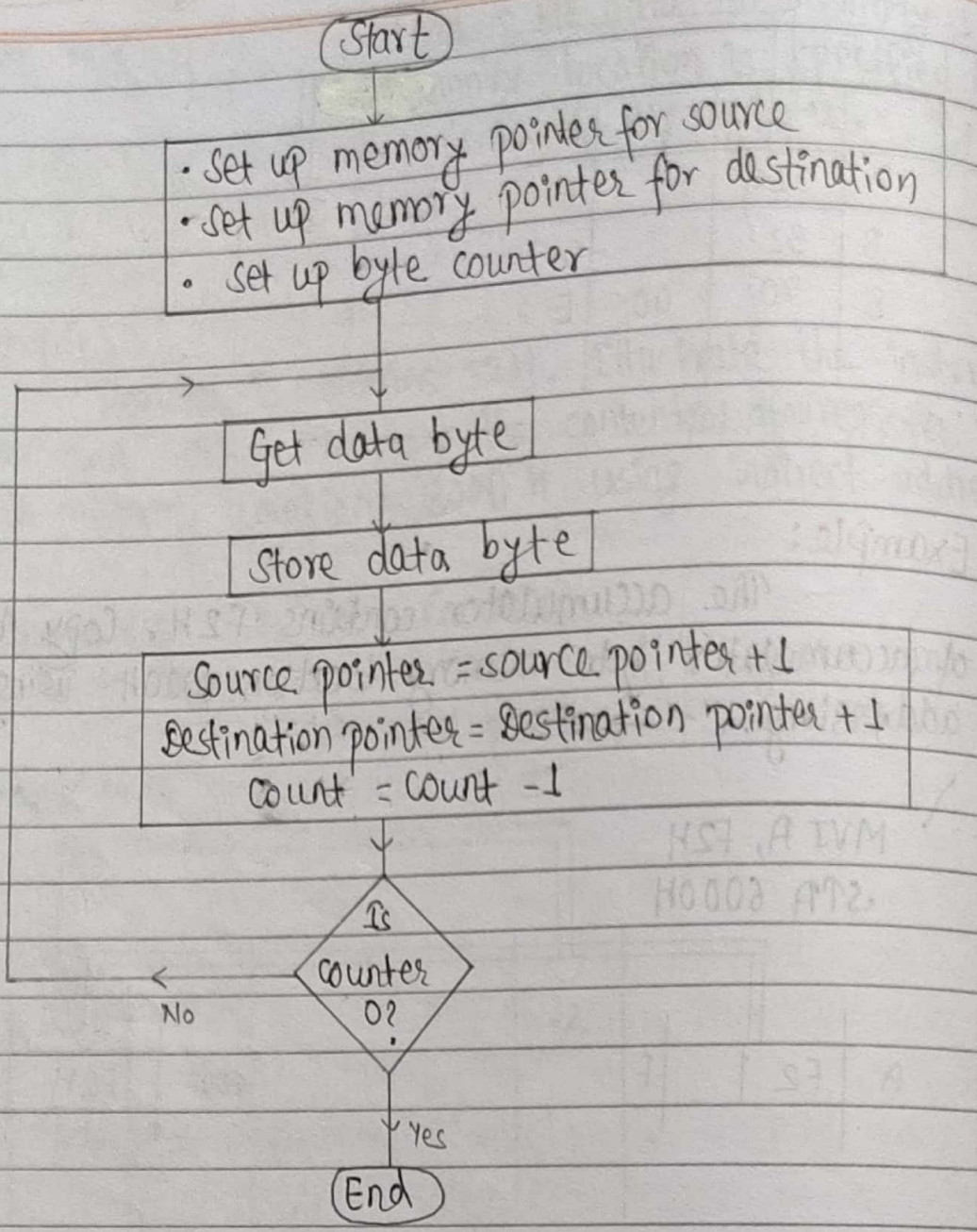


Imp... **Program:**

Sixteen bytes of data are stored in memory locations at XX50H to XX5FH. Transfer the entire block of data to new memory locations starting at XX70H.

Data (H): 37, A2, F2, 82, 57, 5A, 7F, 8A, E5, 8B, A7, C2, B8, 10, 19, 98.

Sol:



Mnemonics	Steps
LXI H, XX50H	1. Set up HL as a pointer for the source memory.
LXI D, XX70H	Set up DE as a pointer for the destination memory.
MVI B, 10H	Set up byte counter

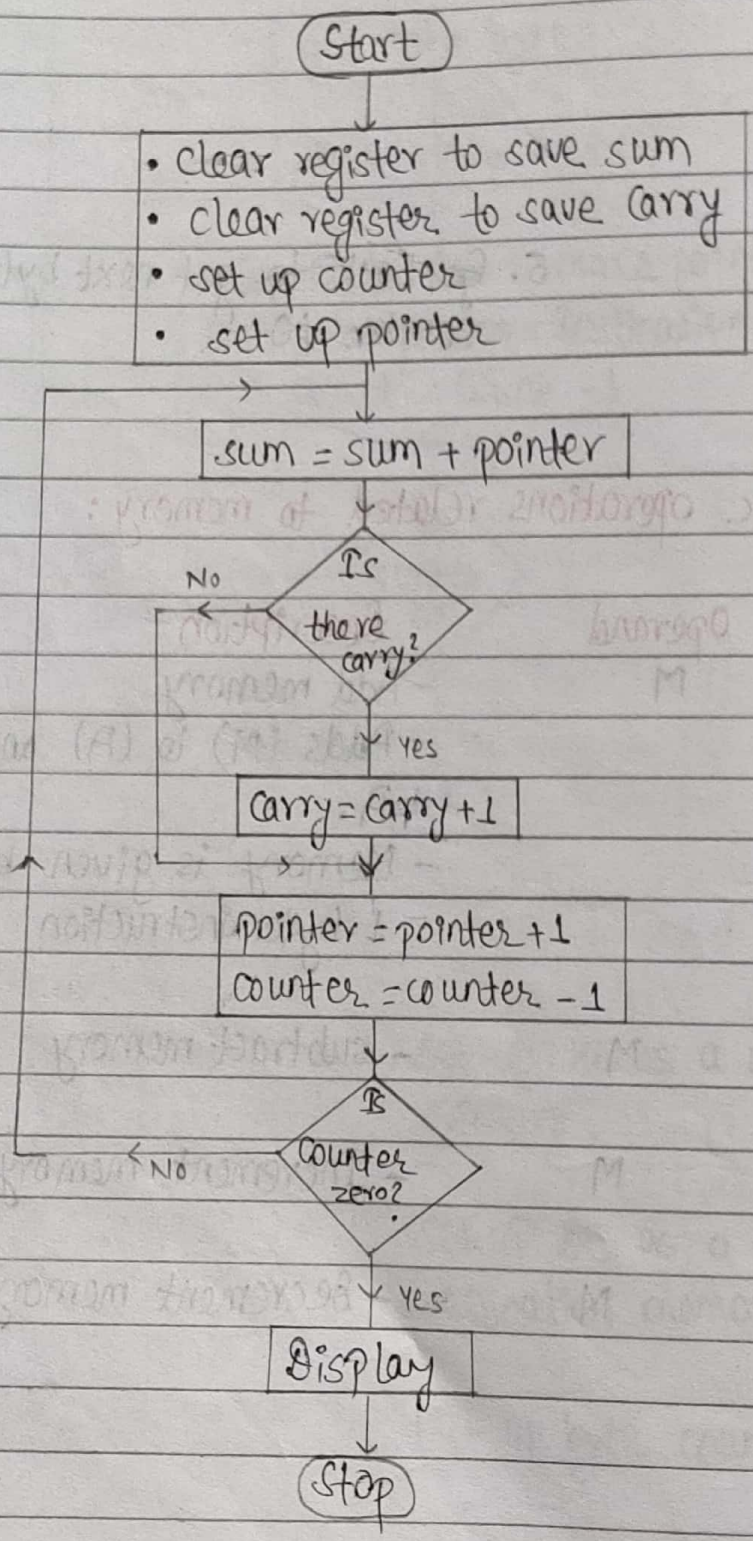
Mnemonics	Steps
NEXT: MOV A, M	2. Get data byte from the source memory.
STAX D	3. Store the data byte in destination memory.
INX H	4. Get ready to transfer next byte.
INX D	
DCR B	
JNZ NEXT	5. Go back to get next byte if byte counter ≠0
HLT	

Arithmetic operations related to memory:

Opcode	Operand	Description
ADD	M	- Add memory - Adds (M) to (A) and stores in A. - Memory is given by HL register - 1 byte instruction
SUB	M	- subtract memory
INR	M	- Increment memory by 1.
DCR	M	- Decrement memory by 1.

Imp:- Program:-

6 byte of data are stored in memory location starting at 2050H. Add all the data bytes. Use register B to save any carries generated while adding the data bytes. Display the entire sum at two output ports.



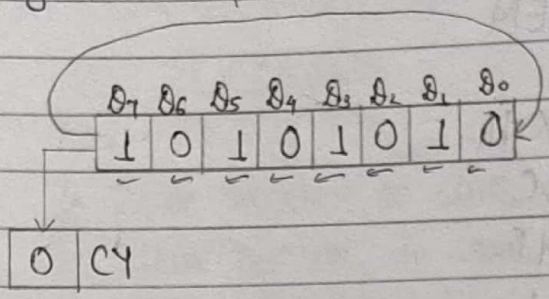
Program:

```
        XRA  A
        MOV  B,A
        MVI  C,06H
        LXI  H,2050H
NXTBYT : ADD  M
        JNC  NXTMEM
        INR  B
NXTMEM : INX  H
        DCR  C
        JNZ  NXTBYT
        OUT  PORT1
        MOV  A,B
        OUT  PORT2.
        HLT.
```

Logical operation:

1) Rotate Accumulator left (RLC):

- Each bit is shifted to adjacent left position. Bit D_7 becomes D_0.
- CY flag is modified according to bit D_7.

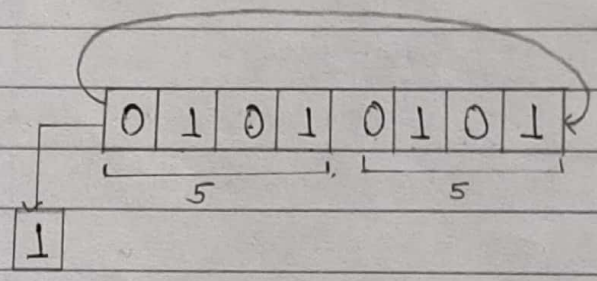


Example:

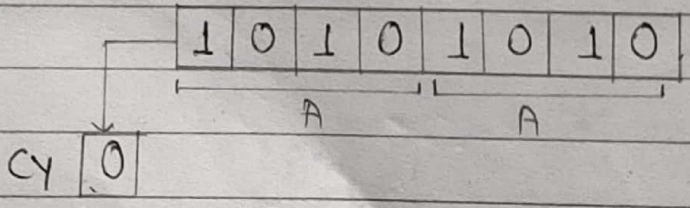
Assume the accumulator contents are AAH and CY = 0. Illustrate RLC instruction if execution is done two times.

Sol:

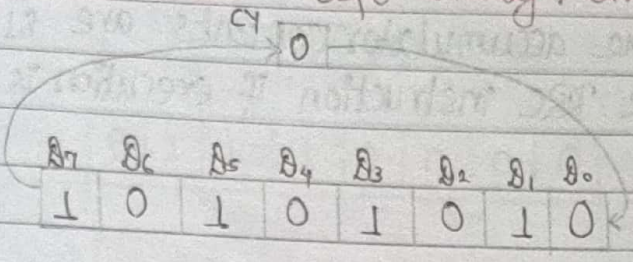
After 1st RLC:



After 2nd RLC:



2) Rotate Accumulator Left through Carry (RAL):

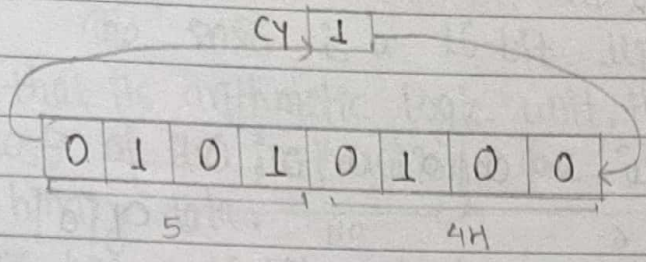


Example:

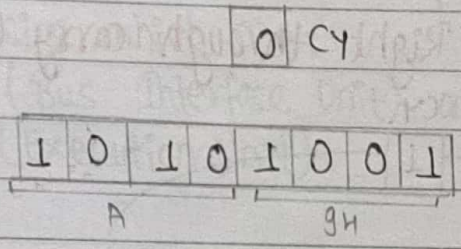
Assume the accumulator contents are AAH and CY=0. Illustrate RAL instruction if execution is done two times.

Sol:

After 1st RAL:

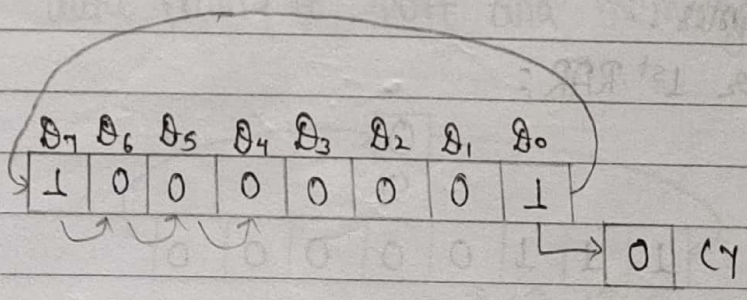


After 2nd RAL:



3) Rotate Accumulator Right (RRC):

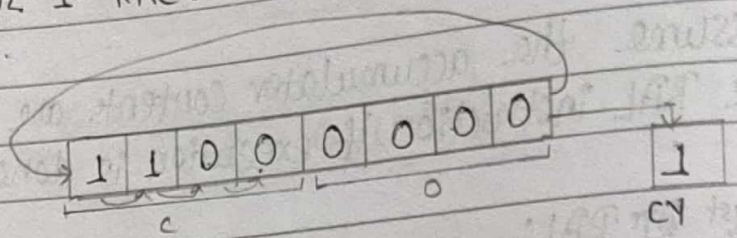
- Each bit is shifted right to the adjacent position. Bit D0 becomes D7.
- Carry flag is modified according to bit D0.



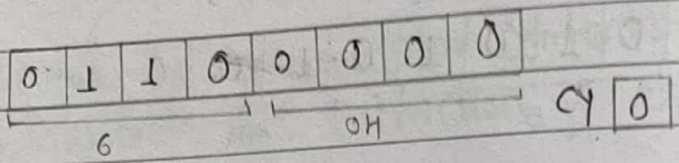
Example:
Assume the accumulator contents are 81 H and CY = 0. Illustrate RRC instruction if execution is done two times.

Sol:

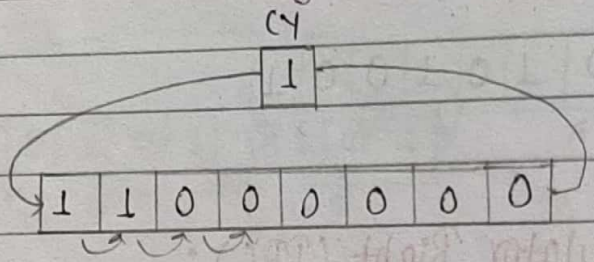
After 1st RRC:



After 2nd RRC:



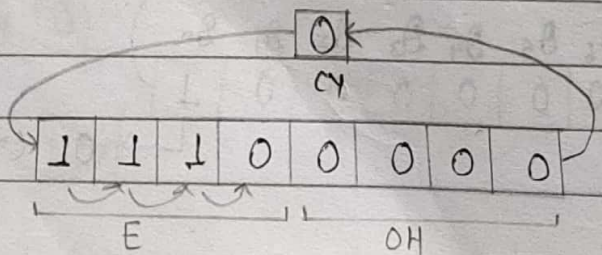
4) Rotate Accumulator Right through carry (RAR):



Example:
Assume the accumulator contents are C0 H and CY = 1. Illustrate RAR instruction if execution is done two times.

Sol:

After 1st RAR:



After 2nd RAR:

0	CY

0	1	1	1	0	0	0	0

7 ... 0H

8086 microprocessor (16-bit up):

The 8086 is a 16-bit up. The term 16 bit implies that its arithmetic logic unit, its internal registers and most of its instructions are intended to work with 16-bit binary data.

- The 8086 has a 16-bit data bus and 20 bit address bus.
- 8086 CPU is divided into 2 function parts to speed up the processing which are:
  * BIU (Bus Interface Unit) and
  * EU (Execution Unit)

BIU:

- It handles all transfer of data and address on the buses for the execution
- sends out address
- fetch instructions from memory
- Read/Write data from/to port and memory.

EU:
- Tell BIU where to fetch instruction or data from.
- Decodes instruction
- Executes instruction

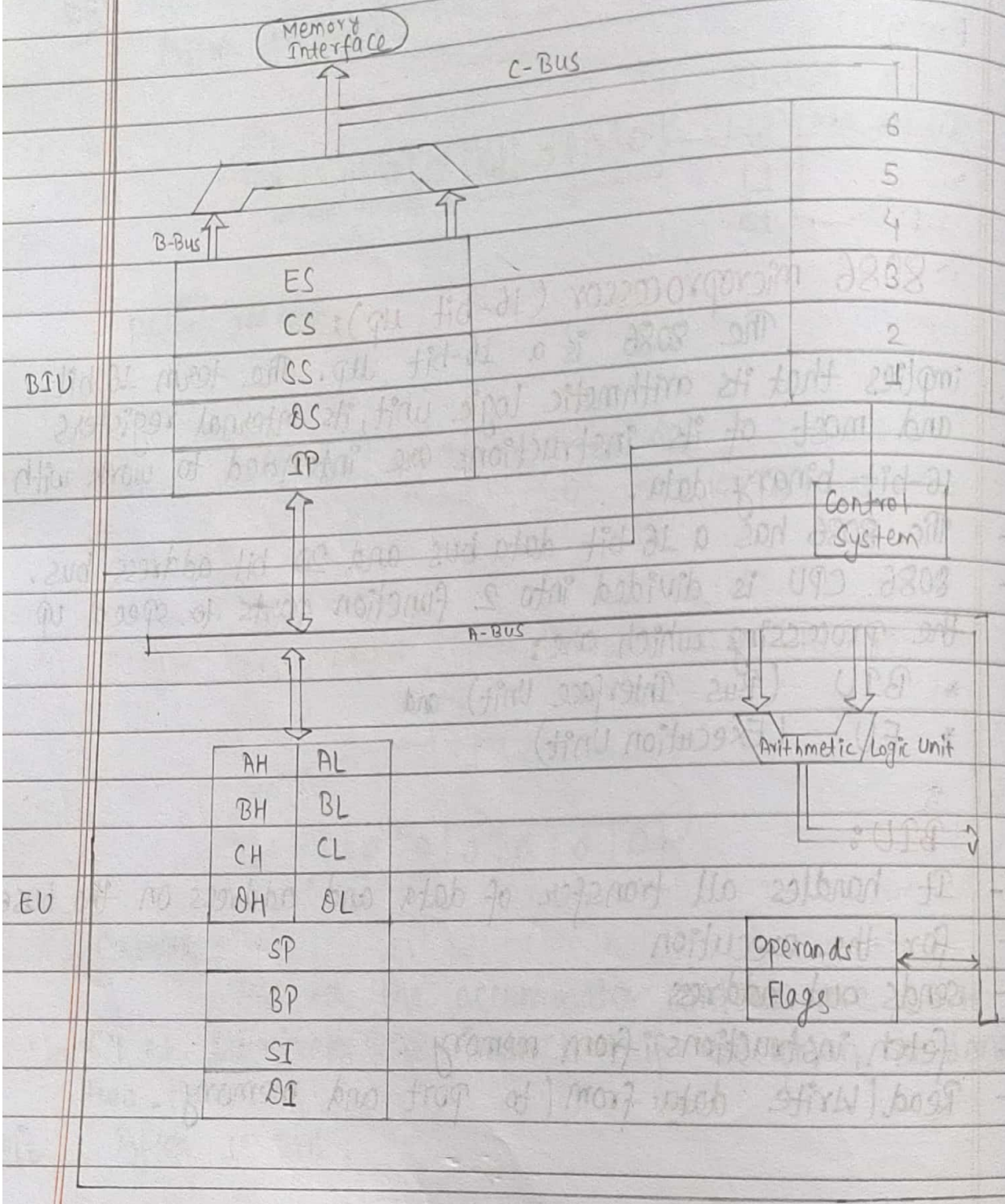


Fig:- 8086 µp architecture

ES : Extra segment
CS : Code segment
SS : Stack segment
DS : Data segment
IP : Instruction Pointer
AH : A (High)
AL : A (Low)
SP : Stack pointer
BP : Base pointer
SI : Source Index
DI : Destination Index

Instruction decoder and ALU:

Decoder in the EU translates instruction fetched from the memory into a series of action which the EU carries out. 16-bit ALU in the EU performs actions such as AND, OR, XOR, INR, DCR, etc.

Flag registers:

It is a 16-bit register. 9 bits are used as different flag, remaining bits are unused.

U	U	U	U	OF	DF	IF	TF	SF	ZF	U	AF	U	PF	U	CF

Out of 9 flags, 6 are conditional flags and 3 are control flags.

1) Conditional flags:

These are set or reset by the EU on the basis of the results of some arithmetic and logic operations.

- OF (overflow flag):
  OF is set if there is arithmetic overflow i.e. result exceeds the capacity.

- SF (sign flag):
  SF is set if the MSB of the result is 1.

- ZF (zero flag):
  ZF is set if result is 0.

- AF (Auxiliary carry flag):
  If carry from lower nibble to upper nibble.

- PF (Parity flag):
  PF is set if the result has even parity.

- CF (Carry flag):
  CF is set if there is carry from addition or burrow from subtraction.

2) Control Flags:
They are set using certain instructions and are used to control certain operations of the processor.

i) TF (Trap Flag):
For single stepping through the program.

ii) IF (Interrupt Flag):
To allow or prohibit the interrupt of a program.

iii) DF (Direction Flag):

Used with string instructions.

General Purpose Register:

- 8 general purpose registers AH, AL, BH, BL, CH, CL, DH, DL are used to store 8 bit data.
- Can form a register pair to store 16 bit.

- Acceptable register pair:
  - AH - AL pair (AX register)
  - BH - BL pair (BX register)
  - CH - CL pair (CX register)
  - DH - DL pair (DX register)

Pointer and index register:

SP (stack pointer), BP (base pointer), SI (source index), DI (destination index) are different pointer and index register.

- The two pointer register SP and BP are used to access data in the stack segment. The SP is used as offset from current segment and BP contains offset address and is used in base addressing mode.

- EU also contains a 16 bit source index (SI) and 16-bit destination index (DI) register. These can be used for temporary storage of data.

Bus Interface Unit (BIU):

QUEUE:

When bus will be free, BIU pre-fetches upto six instruction byte to be executed and places them in queue.

- This improves the overall speed of execution.

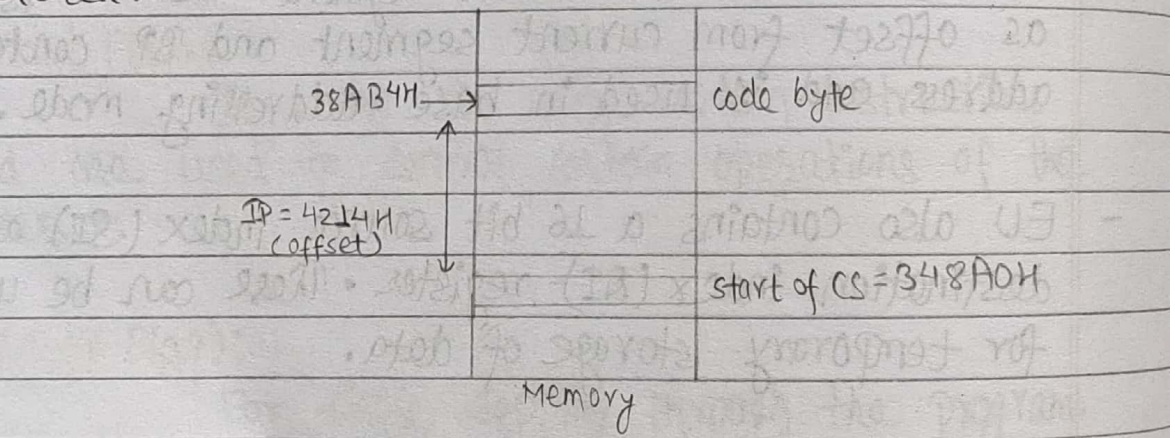
Segment Register:

BIU has four 16-bit segment register namely:

- code segment
- stack segment
- extra segment
- data segment

Code segment (CS) and instruction pointer (IP):

CS contains the base or start of the current code segment and IP contains the distance or offset from this address to the next instruction byte to be fetched.



Addition of IP to CS

Stack segment (SS) and stack Pointer (SP):

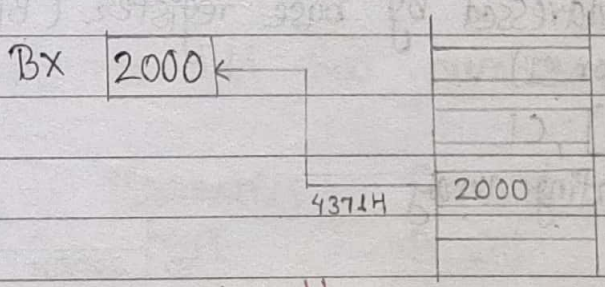
- A stack is a section of memory to store address and data while a subprogram is in progress.
- The stack segment register point to the current stack.

Addressing Modes:

1) **Immediate addressing mode:**
Operand (source) is immediate data in the instruction. Eg:- ADI 23H
- MOV CX, 234AH

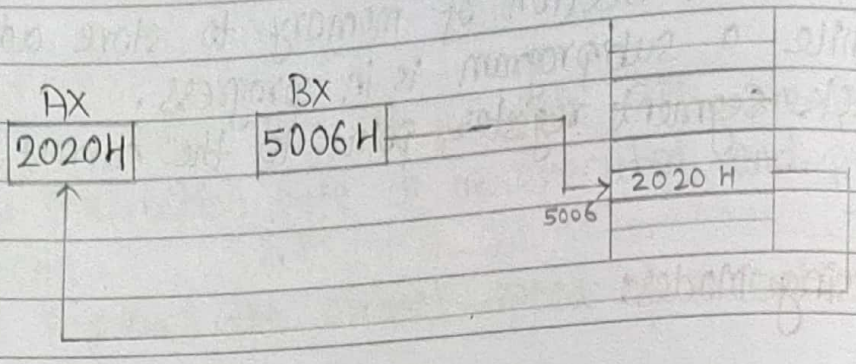
2) **Register addressing mode:**
Transfer a copy of byte or word from source register or memory location to destination register or memory location.
Eg: MOV CX, BX

3) **Direct addressing mode:**
One operand should be memory location.
Eg: MOV BX, [4371H]



4) **Register indirect addressing mode:**
Transfers a word or byte between register and memory addressed by index or base register. Index or base register are BP, BX, DI, SI.

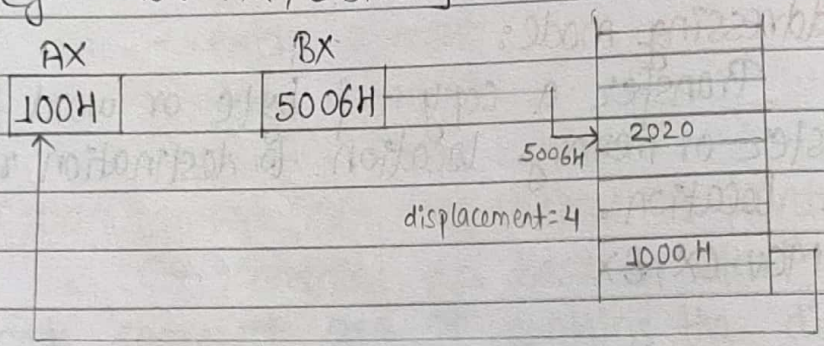
Eg: MOV AX, [BX]



5) Register relative addressing mode:

Transfer a word or byte between registers and memory location addressed by an index or base register plus displacement.

Eg: MOV AX, [BX+4]

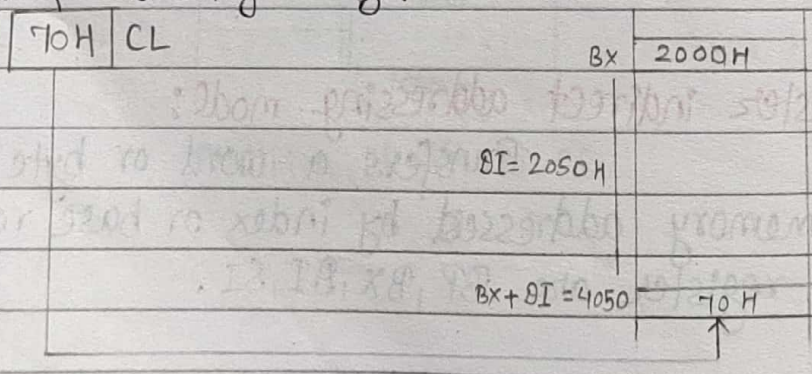


6) Base plus index addressing mode:

Transfer a word byte between register and memory location addressed by base register (BP or BX) plus index register (DI or SI).

Eg: MOV [BX+DI], CL

Used for locating array.



Machine Language:

Machine language is only one programming language that any computer can actually understand and execute. This is the lowest possible level of language in which it is possible to write a program. However, binary code is not native to humans and it is very easy for error to occur in the program.

Program memory address	binary content	context (Hex)
00100H	0000 0100	04 H

Assembly Language:

Program in assembly language are represented by certain words representing the operation of instruction. This programming gets easier. These words used to represent each instruction are called mnemonics.

Assembly language statements has four fields:

- Label field
- Instruction, mnemonics or opcode field
- operand field
- comment field.

Example to add two numbers:

Label	Mnemonics	Operand	Comment
Start	MVI	A, 10H	Move 10H to accumulator
	MVI	B, 20H	Move 20H into B register
	ADD	B	Add contents of register B with accumulator.
	HLT		Halt.

For execution these codes are converted to machine codes. This is done by assembler. An assembler translates a program written in assembly language into machine language program (object code).

A translator converts source codes to object code and then into executable formats. The process of converting source code into object code is called compilation which is done by assembler.

The process of converting object code into executable format is called linking and linker close it.

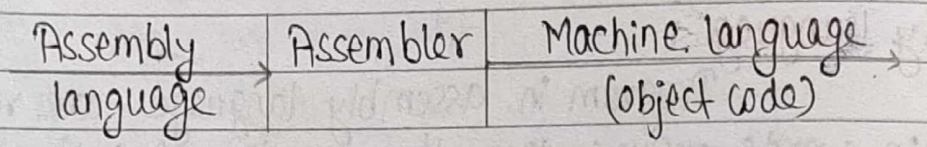


Fig:- Language translator.

Types of assembler:

1, One Pass Assembler
2, Two Pass Assembler

1, One Pass Assembler:-

The assembler reads the assembly language program once and translates the assembly language program to machine code. This assembler has the program of defining forward reference. This means that a branching instruction using an address that appears later in the program must be defined by the programmer after the program is assembled.

2, Two Pass Assembler :-

This assembler scans the assembly language program twice. In first pass, the assembler generates the tables of the symbols (label with address). On second pass, the assembler translates the assembly language program into the machine code. The two pass assembler is thus easier to use.

Macro assemblers linking assemblers and assembler directives:

→ Macro assembler is an assembly language that allows macros to be defined and used. The Microsoft Macro Assembler (MASM) is an X-86 assembler that uses the Intel syntax for MS-Dos and Microsoft Windows.

→ Assembler can be linked with two or more than two assembler called linking assembler.

→ Assembler directives, also called pseudo opcodes or pseudo operations are instructions that are executed by an assembler at assembly time, not by a CPU at run time. They can be used to manipulate presentation of a program to make it easier to read and maintain.

8086 Instruction Sets:

1, Data Transfer Instructions:

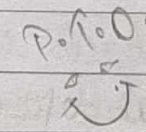
	Mnemonics	Description
1.	MOV	- Copy byte or word from specified source to specified destination.
2.	IN	- Copy a byte or word from specified port to accumulator.
3.	OUT	- Copy a byte or word from accumulator to specified port.
4.	LEA	- Load the effective address of operand into specified register.

2, Arithmetic and Logical Instructions:

i)	ADD	- Add specified byte to byte or specified word to word.
ii)	SUB	- Subtract byte from byte or word from word.
iii)	INC	- Add byte + byte + carry flag or word + word + carry flag.
iv)	DEC	- Decrement specified byte or specified word by 1

v)	MUL	- Multiply unsigned byte by byte or unsigned word by word.
vi)	DIV	- Divide unsigned word by byte or unsigned double word by word.
	(Logical instructions)	
vii)	AND	- AND each bit with the corresponding bit in byte or word.
viii)	OR	- OR each bit with the corresponding bit in byte or word.
ix)	XOR	- Exclusive OR each bit with the corresponding bit in byte or word.
x)	NOT	- Invert each bit.
xi)	CMP	- Compare two specified bytes or words.
xii)	DAA	- Decimal (BCD) adjust after addition.
xiii)	AAA	- ASCII adjust after addition.
xiv)	ROR	- Rotate bits right, LSB to MSB and to C.F.
xv)	RCR	- Rotate bits right, LSB to CF and CF to MSB

xvi)	ROL	-Rotate bits left, MSB to LSB and to C.F.
xvii)	RCL	-Rotate bits left, MSB to CF and CF to LSB.
xviii)	SHL ISAL	-Shift bits left, put zero in LSB.
xix)	SHR	-Shift bits right, put zero in MSB.



Branching Instruction:

JMP	Go to specified address to get next instruction.
CALL	Call a procedure (sub program) save return address on stack.
RET	Return from procedure to calling program
LOOP	Loop through a sequence of instruction until CX=0

Stack instruction:

PUSH	Copy specified word to top of stack
POP	Copy word from top of stack to specified location.

Imp: **Program:**

1. Write an ALP to find factorial of a number for 8086.

Sol:

```
MOV AX,05H
MOV CX,AX
Back: DEC CX
      MUL CX
      Loop Back
; result stored in AX
; to store result at 8000 H
MOV [80000],AX
HLT
```

2. The 8 data bytes are stored in memory location from E000H to E007H, write 8086 ALP to transfer the block of data to new location B001H to B008H.

Sol:

```
MOV BL,08H
MOV CX,E000H
MOV EX,B001H
LOOP: MOV DL,[CX]
      MOV [EX],DL
      DEC BL  → INX CX
      JNZ LOOP  INX EX
      HLT
```

INT 21H function (interrupt):

INT 21H is a common function. The INT 21H instruction in 8086 is a software interrupt. It is used for input/output operation.

Function 01H - Read character from standard input:

It will wait for a character to be stored read at the standard input device then echoes the character to the standard output device and return the character in AL.

Input: AH = 01H

Output: AL = character from the standard input device.

Function 02H - write character to standard output:

- It will pass the character in DL to the standard output device.

Input: AH = 02H

DL = character

Function 09H - write string to standard output device:
- Sends the characters in the string to the standard output device.

Input = AH = 09H
DS: DX = Pointer to the character string ending with '$'.

Function 0AH - Buffered input

Function 4CH - Exists program

Assembler Directives:
* .model small ; selects the size of the memory
; model usually sufficient
; max 64k code 64k data
* .stack ; size of the stack segment
* .data ; beginning of the data segment
* .code ; beginning of the code segment.

Example: data

```
data W    dw   213FH ; data word
data 1    db   52H   ; data byte
sum       db   ?     ; nothing stored but a
                     ; storage is assigned.
```

Example code:

```
ProgramName proc
; Every program needs name
-------- ; program statements
--------

ProgramName endP
End ProgramName
```

Hello world program

```
Title Hello world program
; This program displays "Hello, world".
.model small
.stack 100h
.data
  message db "Hello world", $
.code
  Main proc
  MOV AX, @data; address of data
  MOV DS, AX
  MOV AH, 09H
  MOV DX; offset message
  ; display message
   INT 21H
   MOV AH, 4CH
   INT 21H
   Main endP
   End Main.
```

Offset:-

The offset operator returns the distance of a label or variable from the beginning of its segment. The destination must be in 16 bits.

Example:-

```
MOV BX, offset count.
```

Program: WAP to display "Bsc Computer science and IT for 8086.

⇒

```
Title display the string
.model small
.stack 100h
.data
  string db "Bsc. computer science and IT", $
.code
  Main proc
  MOV AX, @data
  MOV DS, AX
  MOV AH, 09H
  MOV DX, offset string
  INT 21H
  MOV AH, 4CH
  INT 21H
  Main endP
  End Main
```

WAP to reverse the given string for 8086.

⇒

```
Title reverse the given string
.model small
.stack 100h
.data
  string1 db "assembly language program" $
  length dw $-string1 -1
.code
  Main proc
  MOV AX, @data
  MOV DS, AX
  MOV SI, offset string1
```

```
MOV CX, length
ADD SI,CX
Back: MOV DL,[SI]
      MOV AH,02H
      INT 21H
      DEC SI
      Loop Back
      MOV AH,4CH
      INT 21H
      Main endP
      End Main
```

Unit 6:

Basic I/O, Memory R/W and Interrupt operation

Bus structure:

A microprocessor unit performs basically four operations: memory read, memory write, I/O read, I/O write. These operations are part of communication between MPU and peripheral devices. A communication includes identifying peripheral or memory location, transfer of data and control function.

These are carried out using address bus, data bus and control bus. All the buses together are called the system bus.

In case of 8085 MPU are:

- 8 unidirectional address pin
- 8 bidirectional multiplexed address / data pins.
- 11 control output pins
- 11 control input pins

Data bus:-

The data bus provides a path for data flow between the system modules. It contains of a number of separate lines generally 8, 16, 32 or 64. The number of line is referred to as width of the data bus. A single line can only carry one bit at a time, the number of lines determine how many bits can be transmitted at a time.

The width also determines the overall system performance. An 8 bit bus would require twice the time required by 16 bit bus to transmit 16 bit

data.

8085 has 8-bit data bus where as 8086 has 16-bit data bus. Data bus are classified into two type according to the clock pulse applied as:

- Asynchronous bus
- Synchronous bus

Address bus:-

The address bus is used to designate the source and destination of data in data bus. In a computer system, each peripheral or memory location is defined by a binary number called as address. The width of the address bus determines possible memory capacity of the system. Usually higher order bits are used to select particular modules and lower order bits are used to select a memory location or I/O port within a module. 8085 have 16 bit address bus. So, memory capacity is: $2^{16} = 65536$ or 64 kb.

Control bus:-

These are group of lines used to control the data and address bus. Since this bus is shared by all the components of microprocessor system, there must be some control mechanism to distinguish between data and addres.

Some of the control signals are

- Memory read
- Memory write
- I/O read
- I/O write

- Interrupt request
- Interrupt acknowledgement

I/O strategies:-

The computer is useless without some kind of interface to the outside world. There are many different devices which we can connect to the computer system; keyboard, VDU (Visual Display Unit) and disk drives. I/O is governed by 3 basic strategies:

- Programmed I/O
- Interrupt diven I/O
- Direct Memory Access (DMA)

Programmed I/O:

In programmed I/O all data transfers between the computer system and external devices are completely controlled by the computer program. Part of the program will check if any external devices require attention and act accordingly. This process is known as polling. The programmed I/O is very cheap and easy to implement. The main disadvantage is that it is slow and inefficient (wasteful to check every I/O device).

Interrupt driven I/O:

If a polling loop is very long, it is conceivable that such devices will not be serviced in time. So, interrupt driven I/O is needed.

An extra input is provided on the

microprocessor to check the status of this input at the end of each instruction if it is asserted the program jumps to an interrupt service routine (ISR). Actual transfer of data is not accomplished any quicker than it is using programmed I/O but it is efficient i.e. can get attention faster.

Direct Memory Access (DMA):

In some cases the CPU may not be fast enough to keep up with the peripheral or it may be desirable to allow the CPU to do other useful work while the I/O is in progress. In this cases a special purpose processor called DMA controller (DMAC) can be used to transfer data between memory and I/O devices. It can perform all the operations required for data transfer in one bus cycle. This speeds up the transfer of data and reduces the number of bus cycles required to transfer a given amount of data.

Modes of DMA:

- **Cycle stealing mode:**
  DMA controller can be set up to take over the bus for each byte of data to be transferred and then return control to the CPU.

- **Burst mode:**
  In burst mode, block of data is transferred before returning bus control to the CPU.

- Transparent mode:
DMA controller only transfer data when the CPU is performing operation that doeshot use the system bus.

Advantages of DMA:
- DMA allows a peripheral device to read from or write to memory without going through the CPU.
- DMA allows for faster processing since the processor can be working on something else.
- Computer system performance is improved.

Disadvantages of DMA:
- It is costlier than other mode as we require a separate processor called DMA controller.
- In case of burst mode data transfer, the CPU is vendered inactive for relatively long period of time.

2065 IX

DMA controller 8237 interfacing on IBM PC:

The IBM PC and compatible machines use the 8237 DMA controller. It is used to transfer data between I/O ports and memory. The diagram given shows how the 8237 DMA controller interfaces to the CPU, the I/O devices and the memory.

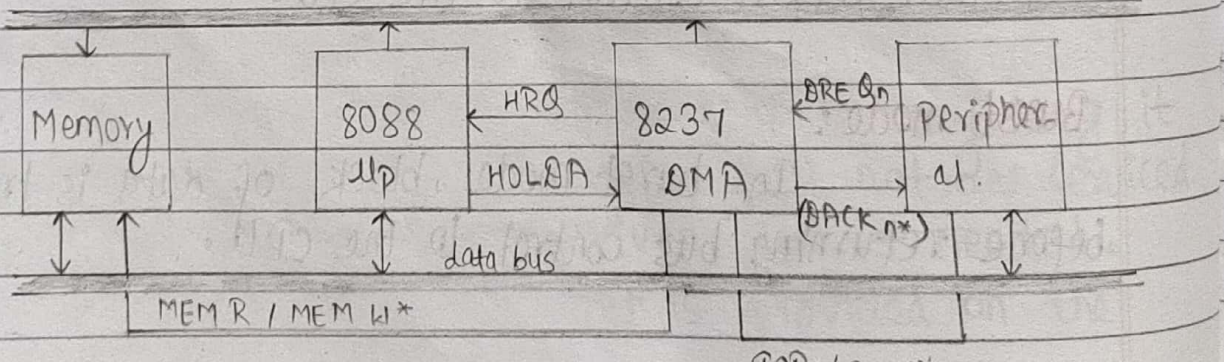


Fig:- DMA interfacing to 8088 up

The DMA controller uses hold request (HRQ) and hold acknowledge (HLDA) signals to ask the CPU to stop driving the address, data and control bus so that the DMA can use them. The DMA controller interfaces to peripherals through 4 pairs of DMA request (DRE Q_0 to DRE Q_3) and DMA acknowledge (DACK$_0$* to DACK$_3$*) lines. Once the DMA controller has control of the bus, it can also interface to memory and I/O peripherals using the address bus, data bus and memory I/O read/write control signal (MEM R*, MEM W*, IOR* and IOW*).

8237 DMA operation:

Sequence of steps a DMA transfer in cycle-stealing mode.

- Each time the peripheral is able to transfer a byte. For that its DMA request line to the DMA controller is asserted.
- DMA controller asserts the CPU hold request.
- When the CPU control circuitary is able to suspend execution it asserts the hold 'acknowledge (HLDA) signal to DMA controller and leaves address, data and control bus signals for DMA.
- DMA asserts either MEMR* plus IOW* or MEMW* plus IOR* on the control bus.
- DMA acknowledge signal by reading or writing its data to the data bus.
- Memory responds to MEMR*/MEM W* control signal which causes the data to be read/written directly from/to memory.
- DMA controller then negates hold request so that CPU can continue to execute.

Interrupt:-

An interrupt is considered to be an emergency signal that may be serviced. The µp may respond to it as soon as possible. When the µp receives an interrupt signal, it suspends the currently executing program and jump to an interrupt service routine (ISR) to respond to the incoming interrupt.

Imp.. Types of interrupt:

Vector interrupt:-

In this type of interrupt, processor knows the address of interrupt service routine.
Example: RST 7.5, RST 6.5, RST 5.5, TRAP.

Non-vector Interrupt (Polled interrupt):

In this type of interrupt, processor cannot know the address of interrupt. It should be given externally. In this, the device will have to send the address of interrupt service routine to processor for performing interrupt. Here, the interrupt controller must poll i.e. send a signal out to each device to determine which one made the request.
Example: INTR

Maskable interrupt:

An interrupt which can be disabled by software that means we can disable the interrupt by sending appropriate instruction is called a maskable interrupt.
Example: RST 7.5, RST 6.5, RST 5.5

Non-maskable interrupt:

As name suggest we cannot disable the interrupt by sending any instruction is called non-maskable interrupt. TRAP interrupt is the non-maskable interrupt for 8085. It means that if an interrupt comes via TRAP, 8085 will have to process the interrupt, we cannot mask it.

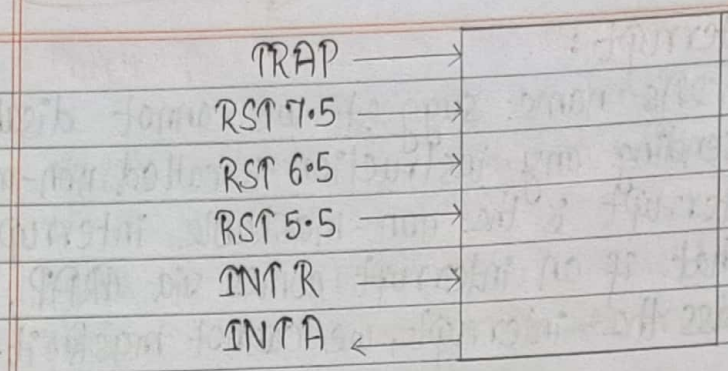
Software interrupt:

It is a instruction based interrupt which is completely controlled by software. This means programmer can use this instruction to execute interrupt in main program. There are eight software interrupt available in 8085 up.

Instruction	Hex code	Vector address
RST 0	C7	0000 H
RST 1	CF	0008 H
RST 2	D7	0010 H
RST 3	DF	0018 H
RST 4	E7	0020 H
RST 5	EF	0028 H
RST 6	F7	0030 H
RST 7	FF	0038 H

Hardware interrupt:

It is interrupt which can get the interrupt request in hardware pin of up 8085. There are six pins available for hardware interrupt. These are TRAP, RST 7.5, RST 6.5, RST 5.5, INTR, INTA.



INT A is not an interrupt pin but it is used to send acknowledgement of the interrupt request from other interrupt pin.

Interrupt vectors and the vector table:

An interrupt vector is a pointer to where ISR is stored in memory area called the interrupt vector table (IVT). IVT of 8085 is usually located in memory (0000H - 00FFH). The purpose of the IVT is to hold the vectors that redirect the µp to the right place when an interrupt arrives. RST and TRAP are automatically vector or transferred to specific location on memory page.

Interrupt	Call location
TRAP	0024H
RST 7.5	003CH
RST 6.5	0034H
RST 5.5	002CH

The TRAP has the high priority followed by RST 7.5, RST 6.5, RST 5.5 and INTR.

Intel 8259A programmable interrupt controller:

Interrupt are used to communicate a computer system with external devices such as a keyboard, a printer, system timers, hard disk controller, sound or graphic card that need immediate handling. Intel 8259 programmable interrupt controller is a circuit which controls interrupt handling.

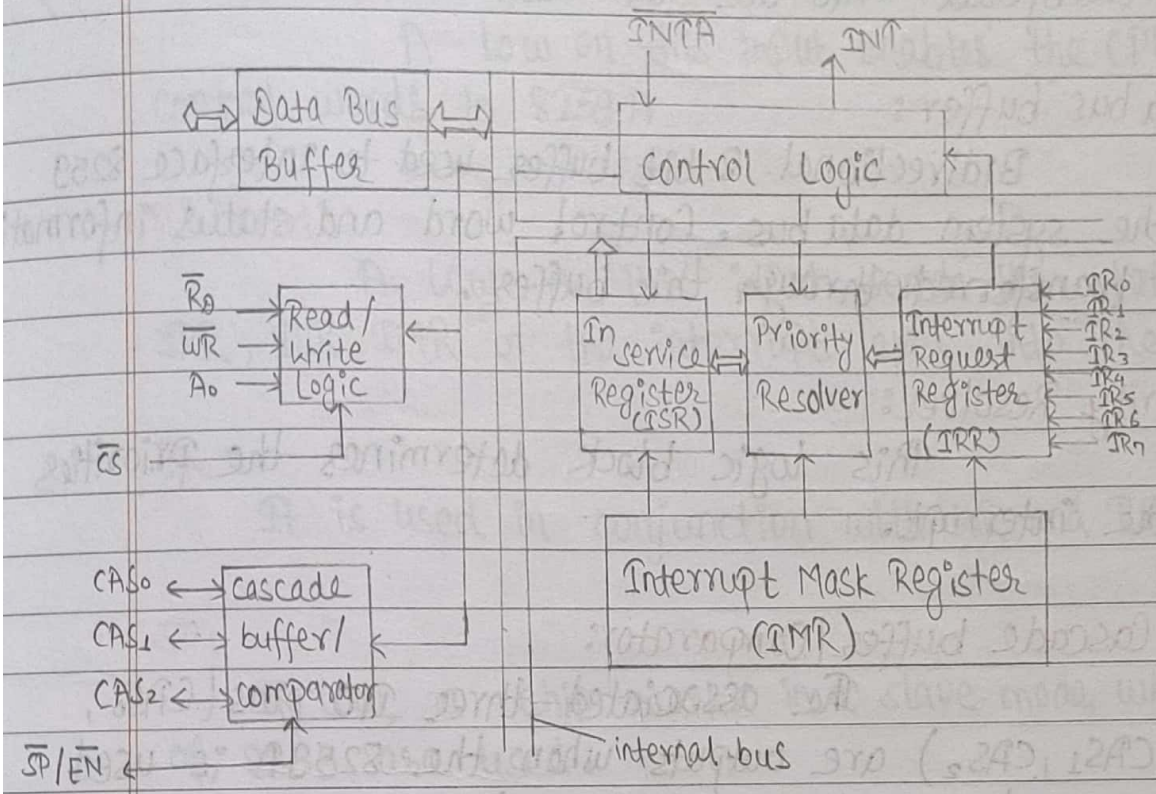


Fig: 8259A Block Diagram

Interrupt Request Register (IRR):

IRR is used to store all the interrupt levels which are requesting service.

In Service Register (ISR):

It is used to store all the interrupt levels which are being serviced.

Interrupt Mask Register (IMR):
It is used to store the bits which mask the interrupt lines to be masked.

Read/write control logic:
The function of this block is to accept command from the CPU & also allows the status of the 8259A to be transfered onto the data bus.

Data bus buffer:
Bidirectional 8-bit buffer used to interface 8259 to the system data bus. Control word and status information are transferred through this buffer.

Priority Resolver:
This logic block determines the priorities of the interrupt.

The Cascade buffer/Comparator:
The associated three I/O pins (CAS_0, CAS_1, CAS_2) are outputs when the 8259A is used as a master and are input when it is used as a slave.

INT (Interrupt):
This output goes directly to the CPU interrupt input.

INTA (Interrupt Acknowledgement):
INTA will cause the 8259A to release interrupt sub-routine address onto the data bus.

IR_0-IR_7:
Interrupt request lines.

CS (Chip select):
No reading or writing of the chip will occur unless the device is selected.

WR (write):
A low on this input enables the CPU to write control words to 8259A.

RD (Read):
A low on this input sends the status of the IRR, ISR, IMR or the interrupt level into the data bus.

A_0:
It is used in conjunction with WR and RD signals.

SP:
If low, a chip works in a slave mode when cascade of 8259 are used.

8259A operation:

- One or more of the interrupt request lines (IR_7-IR_0) are raised high, setting the corresponding IRR bits.
- The 8259A evaluates these requests, and sends an INT to the CPU.
- The CPU acknowledges the INT and responds with an INTA pulse.
- Upon receiving an INTA from the CPU, the highest priority ISR bit is reset. The 8259A will also release

a CALL instruction code into the 8-bit data bus through its D_7 to D_0 pins.
- This CALL instruction will initiate two more INTA pulses to be sent to the 8259A from the CPU group.
- These two INTA pulses allow the 8259A to release its preprogrammed subroutine address onto the data bus.
- In this way, 8259A operation is completed.

Priority modes and other features:

Commonly used priority modes are:

1) Fully nested mode:

This is a general purpose mode in which all IRs are arranged from highest to lowest, with IR_0 as the highest and IR_7 as the lowest. In addition, any IR can be assigned the highest priority in this mode.

Eg:

IR_0	IR_1	IR_2	IR_3	IR_4	IR_5	IR_6	IR_7
4	5	6	7	0	1	2	3

Here, IR_4 has the highest priority and IR_3 has the lowest priority.

2) Automatic Rotation mode:

In this mode, a device, after being serviced receives the lowest priority. Assuming that the IR_2 has just been serviced, it will receive the 7th priority as shown:

IR_0	IR_1	IR_2	IR_3	IR_4	IR_5	IR_6	IR_7
5	6	7	0	1	2	3	4

3) Specific Rotation mode:

This mode is similar to the automatic rotation mode, except that user can select any IR for the lowest priority, thus fixing all other priorities.

Other features of 8259A:

The 8259A is a complex device with various modes of operation. Some other modes are:

Interrupt Triggering:

The 8259A can accept an interrupt request with either the edge triggered mode or the level triggered mode. The mode is determined by the initialization instruction.

Interrupt Status:

The status of the 3 interrupt register (IRR, ISR and IMR) can be read and this status information can be used to make the interrupt process versatile.

Poll method:

8259A can be setup to function in a polled environment. The MPU polls the 8259A rather than each peripheral.

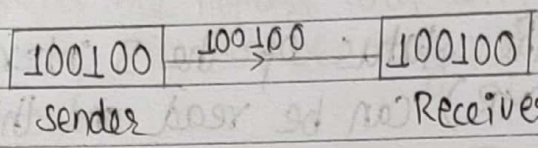
Unit 7 :

Input/Output Interfaces

Serial communication:

Serial communication involves the transmission of data from one place to another using serial devices. In serial communication, the data is transmitted bit by bit on a single line. This minimize the interconnecting wires, thereby reducing the number of the lines between drivers and receivers.

One single bit can be transmitted at a time reducing the data transfer rate as the time required to transmit increases. Only one wire is used for transmission due to which cross-talk (interference between lines) decreases.



Asynchronous Bus and Interface:

In an asynchronous bus the timing is maintained in such a way that occurance of one event on the bus follows and depends on the occurrance of previous event.

- ## ASCII code:

The ASCII stands for the American Standard Code for Information Interchange. It is an 8 bit code commonly used with microprocessor for representing alphanumeric codes. Out of 8-bits first 7 are used to represent the character while the 8-bit is used to test for errors and referred as parity bit.

- Baud rate:

The term baud rate is used to indicate the rate at which serial data is being transferred. The rate is usually expressed as Bd or bits/second.

- Start bit and stop bit:

In serial communication, the data are sent serially so to differentiate between proper data and garbage data, the data is enclosed in start bit and stop bit.

- Parity bit:

The parity bit follows the final data bit. Depending on the type of parity its value may be 0 or 1. This bit is used to check error in received data.

Synchronous Bus and Interface:

A synchronous bus has its events ted by a clock. The clock transmits a regular sequences of a 0's and 1's of equal duration. A single 1-0 transmission is called clock-cycle or bus cycle. All other devices work with the clock cycle. Synchronous interface is a widely used interface standard for industrial application between a master (controller) and a slave (sensor).

8255 programmable peripheral Interface (PPI):

The 8255A is a widely used programmable parallel I/O device. It can be programmed to transfer data under various condition from simple I/O to interrupt I/O. 24 I/O ports available in 8255 are divided into two groups (Group A and Group B). The 24 I/O ports from three 8-bit parallel ports (Port A, port B and port C). The

functions of these ports are defined by writing a control word in the control register.

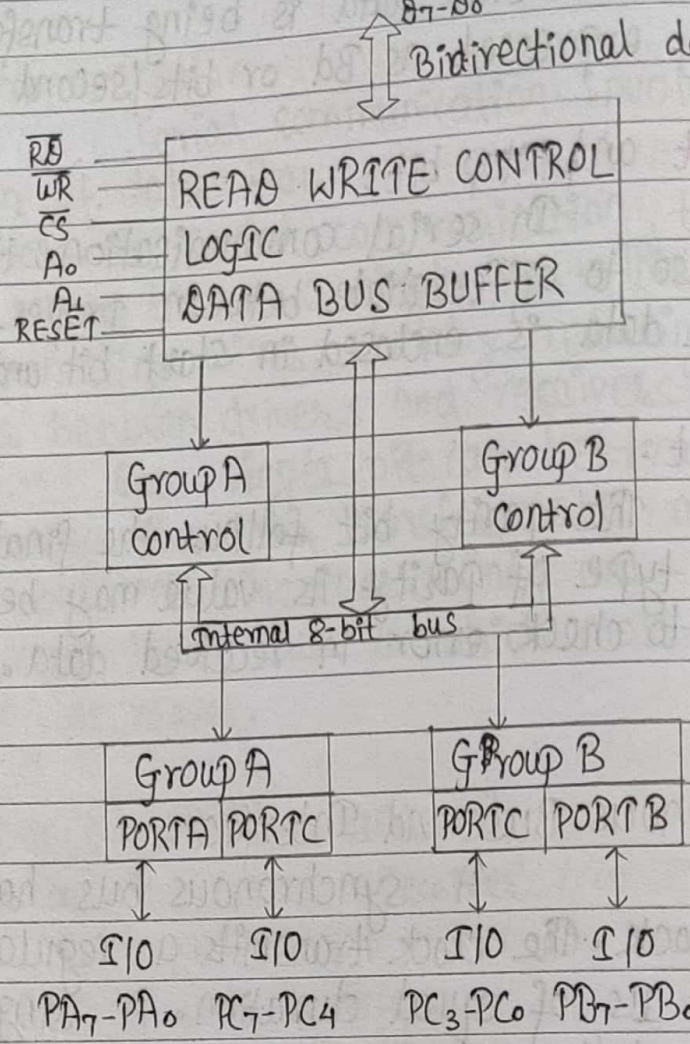


Fig: Block diagram of 8255 PPI

We see that there are 24 I/O lines. Port A and Port B can be used as 8-bit input or output port. Port C is a special port which can be used as 8-bit I/O port two 4 bit ports, as individual lines or to produce handshake signals for port A and port B.

The address input A_0 and A_1 allows us to selectively access one of the three ports or control register. The internal address for the devices are port A 00, port B

port C 10, and control register 11.

8255 operational modes and initialization:

There are 3 modes of operation that can be selected by control words.

Mode 0 : Basic I/O (Group A, Group B)
Mode 1 : Strobe I/O (Group A, Group B)
Mode 2 : Two way bus (Port A only)

Mode 0:

When we need a port for simple input or output without handshaking we initialize the port in mode 0. Ports don't have handshake or interrupt capability.

Mode 1:

When we need to use strobe input or output then we can initialize port A and port B in mode 1. In this mode, some of the pins of ports function as handshake line. In mode 1 interrupt logic is supported.

Mode 2:

Only port A can be initialized in mode 2 where it can be used for bidirectional handshake data transfer. This mean that data can be output or input on the same eight lines. If port A is used in the mode, then pins PC_4 through PC_7 are used as handshake lines for port A.

Constructing 8255 control words:

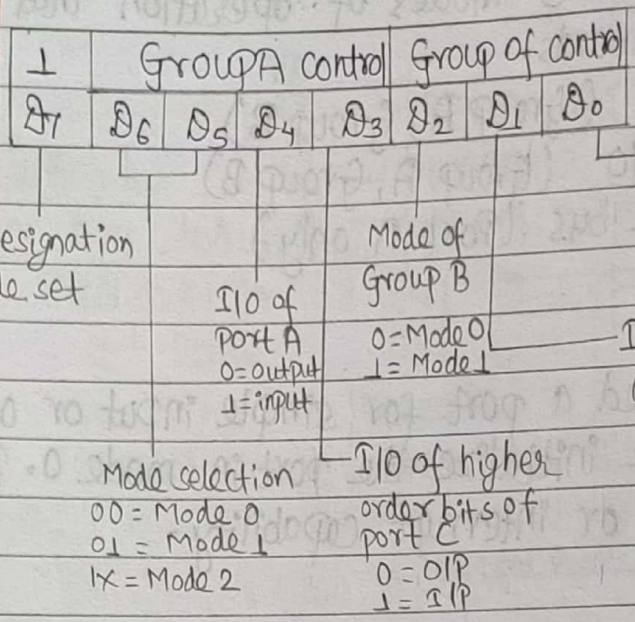


Fig: 8255 Mode set control word

Example:

Construct a control word to configure Port A and Port C_H as output port and port B and Port C_L as input ports.

Sol^n:

D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
1	0	0	0	0	0	1	1
I/O function	Port A in mode 0		Port A as output	Port C_H as output	Port B in mode 0	Port B as input	Port C_L as input

Control word
83H

Dontcare

D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0

Bit set/reset flag 0 = active

Bit set/reset 0 = reset 1 = set

0	0	0	PC_0
0	0	1	PC_1
0	1	0	PC_2
0	1	1	PC_3
1	0	0	PC_4
1	0	1	PC_5
1	1	0	PC_6
1	1	1	PC_7

Fig: 8255 PortC bit set/reset control word.

Example:

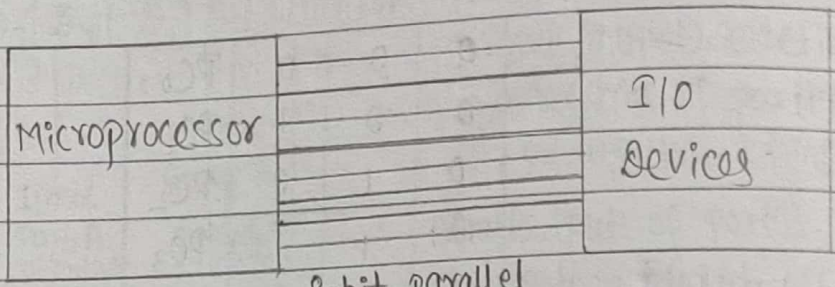
Write control word to set and reset bit PC_7 and PC_3.

Sol:

	D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0	Control word
Set bit PC_7	0	0	0	0	1	1	1	1	0FH
Reset bit PC_7	0	0	0	0	1	1	1	0	0EH
Set bit PC_3	0	0	0	0	0	1	1	1	07H
Reset bit PC_3	0	0	0	0	0	1	1	0	06H

Parallel communication:

Parallel data transfer takes place between any two devices like microprocessor and memory, microprocessor and I/O devices and memory and I/O devices.



8-bit parallel communication

Methods of parallel communication:

Simple Input/Output:-

When we need to get digital data from a simple switch such as a thermostat, into a microprocessor, all we have to do is connect the switch to an input port line and read the port. The thermostat data is always present and read. So, we can read it any time. Similarly, LED input can be connected on an output port in and output the logic level request to turn on the light. The data can be sent to LED at anytime.

Simple strobe I/O:

In many application valid data is present on an external device only at a certain time, so it must be read in at that time. An example is the ASCII encoded keyboard where a key is send on eight parallel lines. We can connect this strobe line to an input port line and poll it to determine when we can input valid data from the keyboard.

Another alternative is to connect the strobe line to an interrupt input on the processor and have an interrupt service procedure read in the data when the processor receives an interrupt. Here the transfer is time dependent.

Single handshake I/O:

The peripheral outputs some parallel data and sends an STB signal to the microprocessor. The µp detects the asserted STB signals on a polled or interrupt basis and reads in the byte of data. Then the µp sends an acknowledgement (ACK) to the peripheral to indicate that data has been read and that the peripheral can send the next byte of data.

Similarly, for an output the µp output a character to the printer and raises the STB signal to the pointer to tell "Here is a character for you." When the printer is ready, it answers back with ACK signal to tell "I get that one, send me another."

Double handshake I/O:

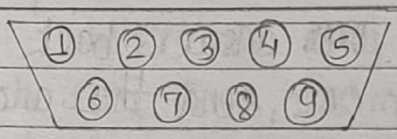
Such mode of data transfer is used when even more coordination is required between the sending systems and the receiving systems. Such mode of data transfer can be much easily understood as a conversation between two people. The sending device asserts its STB line to sak "Are you ready?". The receiving system raises its ACK line high to say, "I am ready". Then the peripheral devices send the byte of data and raise its STB line to say "Here is some valid data for you". After it has read in the data, the receiving system drops its ACK line low to say "I have the data thank you and I wait for your request to send the next byte of data."

RS-232 serial Data Standard:

The signals output of 8251 (USART) are not suitable for transmission over long distances, so these signals are converted to some other form to be transmitted. Modems were developed so that the terminals could use phone line to communicate. Modems are often referred to as data communication equipment (DCE). The terminals or computers that are sending or receiving the data are referred to as data terminal equipment (DTE).

In response to the need for signal and handshake standards between DTE and DCE, the Electronic Industries Association (EIA) developed RS-232 standard. RS-232 is a 25 pin standard but for system were many of 25 pins are not needed much, a 9 pin connector is also used.

Male RS-232 DB 9



Pin Number	Signals
1.	Carrier detect (CD)
2.	Received data (RD) from a DCE
3.	Transmitted data (TD) to a DCE
4.	Data terminal ready (DTR)
5.	Signal ground
6.	Data set ready (DSR)
7.	Request to send (RTS)
8.	Clear to send (CTS)
9.	Ring indicator (RI) from DCE

DTE and DCE interconnection:

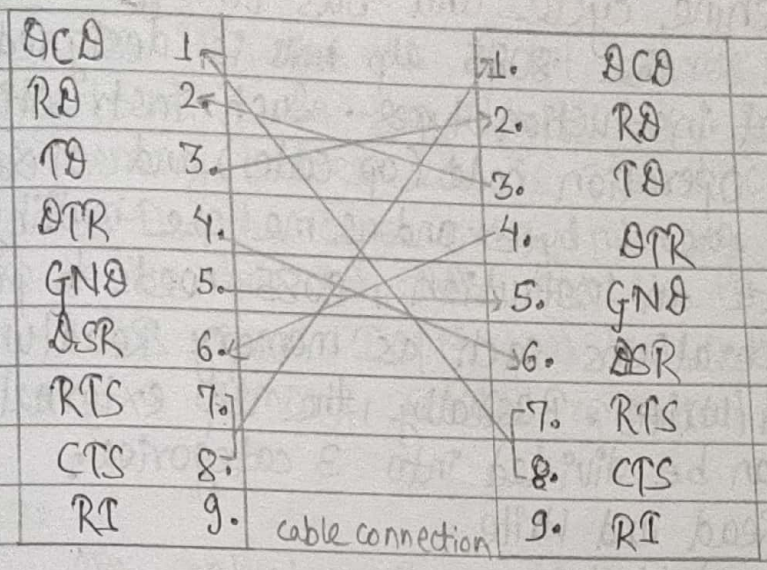


Fig: DTE & DCE interconnection (DB9)

Unit 3:

Instruction Cycle

8085 machine cycle and bus timing:

The 8085 µp is designed to execute 74 different instruction types. Such instruction has two parts: Operation code (op code) and operand. Some instruction are 1 byte and some are multibyte instructions. To execute an instruction, 8085 needs to perform various operations such as memory Read/write and I/O Read/write. Basically, the µp external communication function can be divided into 3 categories:

1. Memory Read and Write
2. I/O Read and Write
3. Request Acknowledgement

These functions are further divided into various operation called machine cycle as shown in table 4.1 (book). Each instruction consists of one or more of the machine cylce and each machine cylce is divided into T-states.

Imp Opcode fetch machine cycle:

The first operation in any instruction is opcode fetch. The µp needs to get (fetch) this machine code from the memory register where it is stored before the µp can begin to execute the instruction.

Example:-

The instruction code (MOV C,A) (4F H) (01001111) is stored in memory location 2005H. List the sequence

of events when the instruction code is fetch by the MPU with timing diagram.

Step 1:

The PC places the 16-bit address 2005H of the memory location on the address bus. At T1 the higher order memory address 20H is placed on the address line A_{15}-A_8, the lower order memory address 05H is placed on the bus AD_7-AD_0 and the ALE signal goes high. Status signal IO/$\overline{M}$ goes low, indicating that this is a memory related operation.

Step 2:

The control unit sends the memory read control signal (MEMR). So, control signal $\overline{RD}$ is sent out during the clock period T_2. The $\overline{RD}$ signals is active during two clock periods.

Step 3:

The instruction 4FH is placed on the data bus (AD_7-AD_0) and transferred to the instruction decoder of the µp.

Step 4:

The machine code 4FH is decoded by instruction decoder and contents of accumulators are copied into register C. This task is performed during the period T_4.

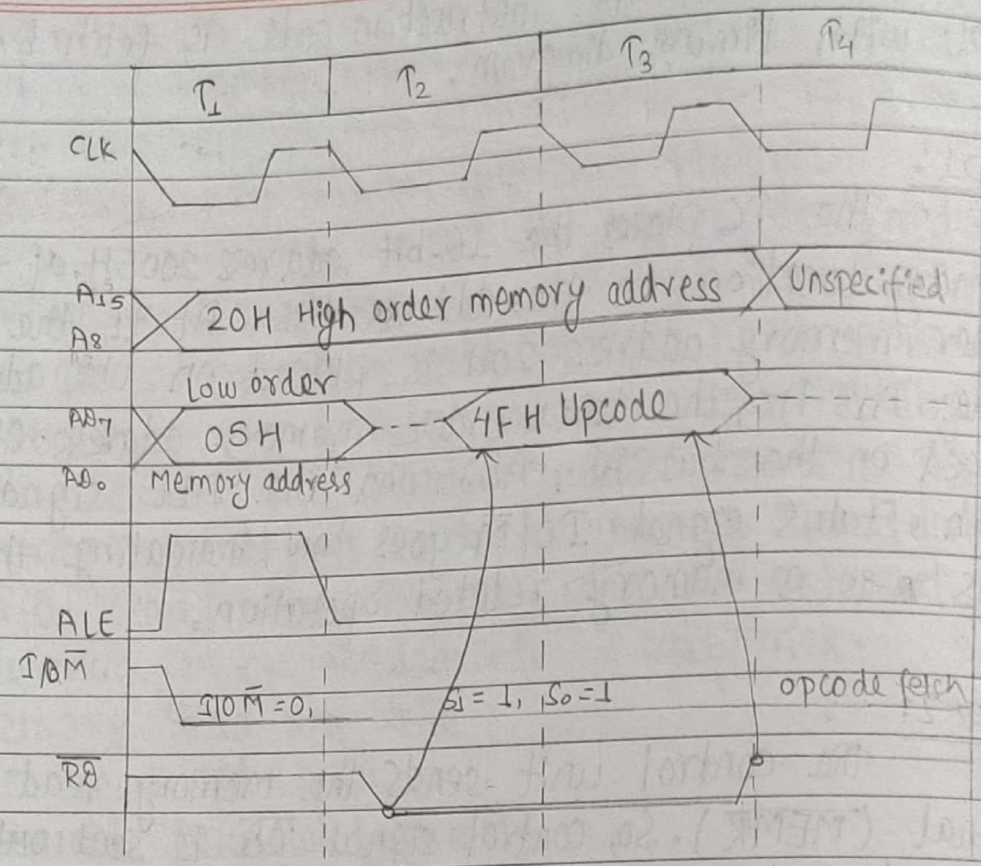


Fig:- Timing diagram of opcode fetch (MOV C,A)

Hint:-

For opcode fetch remove 20H, 05H & 4FH from the diagram.

Memory read machine cycle:

To understand the memory read machine cycle, we need to examine the execution of a 2 or 3 byte instruction because in a 1-byte instruction the machine operation is always an opcode fetch.

Example:-

Instruction	Machine code	Memory location
MVI A,32H	3E H	2000 H
	32H	2001 H

classmate
Date
Page

Steps

- The first machine cycle M_1 (opcode fetch) is identical in bus timing as with MOV except for the bus contents.
- After completion of opcode fetch cycle, 8085 places the address 2001 on the address bus and PC is implemented to 2002H.
- The second machine cycle M_2 is identified as the memory read cycle ($IO/\overline{M} = 0, S_1 = 1, S_0 = 0$) and ALE is asserted.
- At T_2 the $\overline{RD}$ signal become active and enable the memory chip and places the data byte 32H on the data bus.
- At T_3 8085 up reads and stores the bytes in the acumulator.