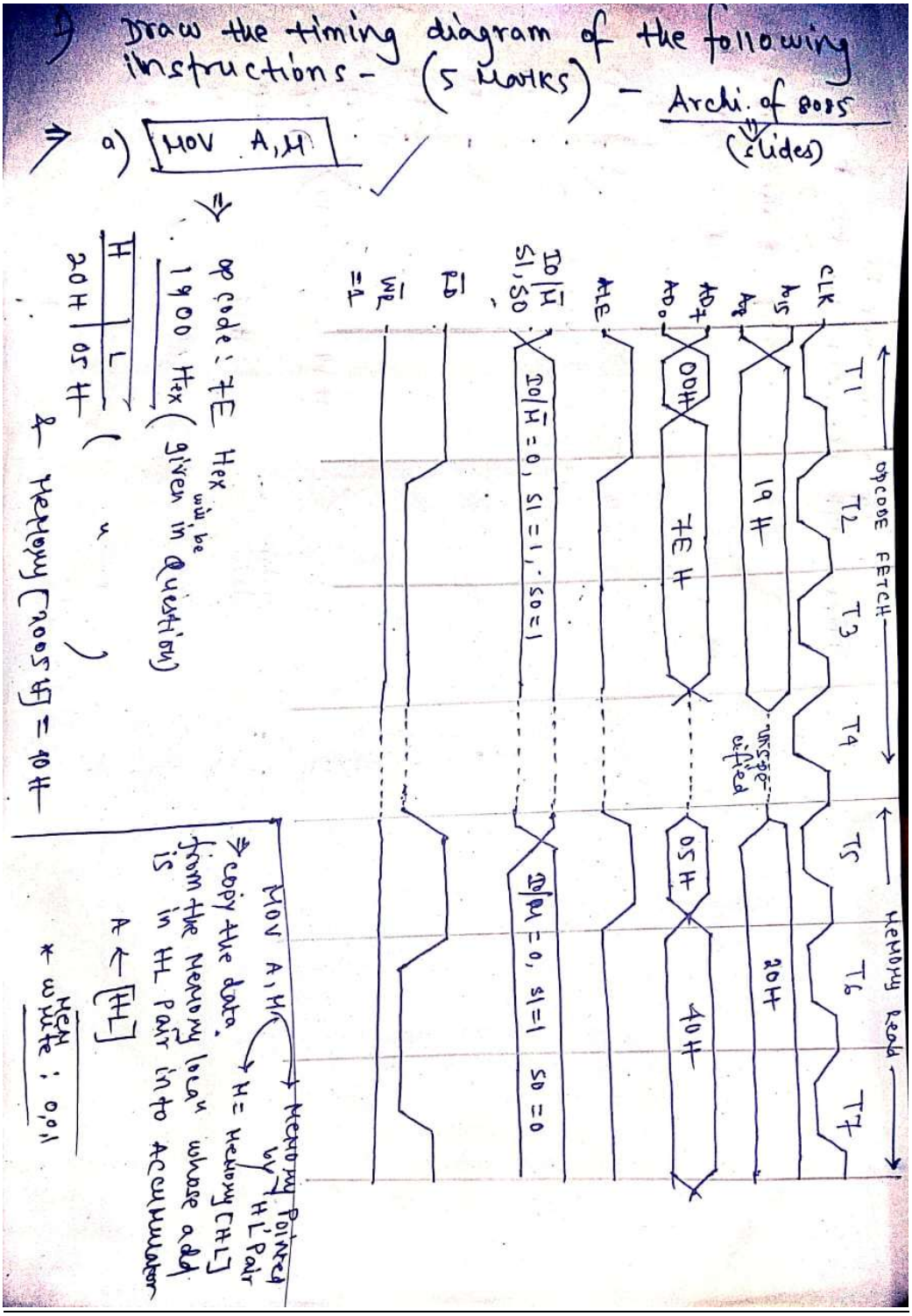
Microprocessor – Suggestions and PYQ – Questions and Answer

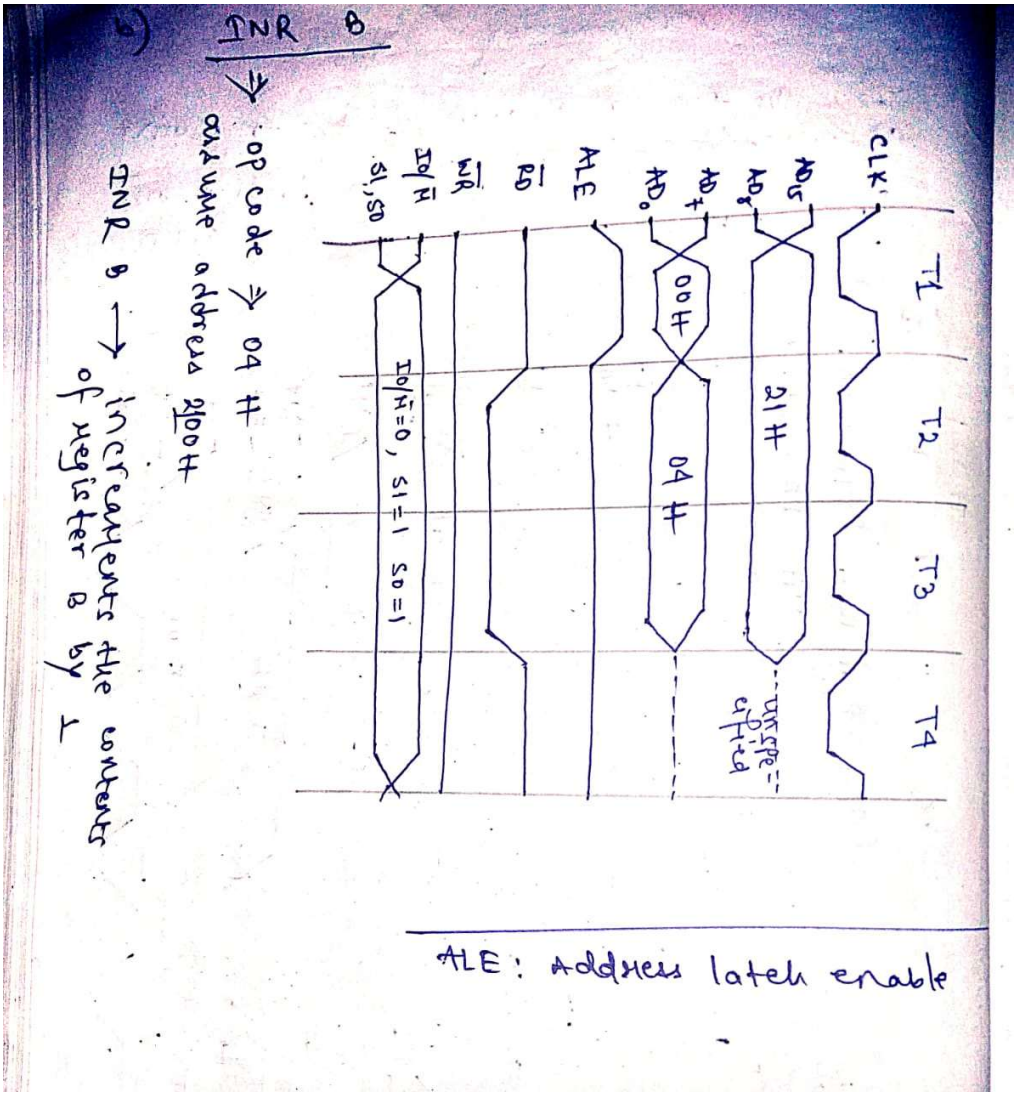
i. ***Instructions:***

1. Draw the timing Diagram of the following Instructions : *(Archi. Of 8085 up)* (5 marks)

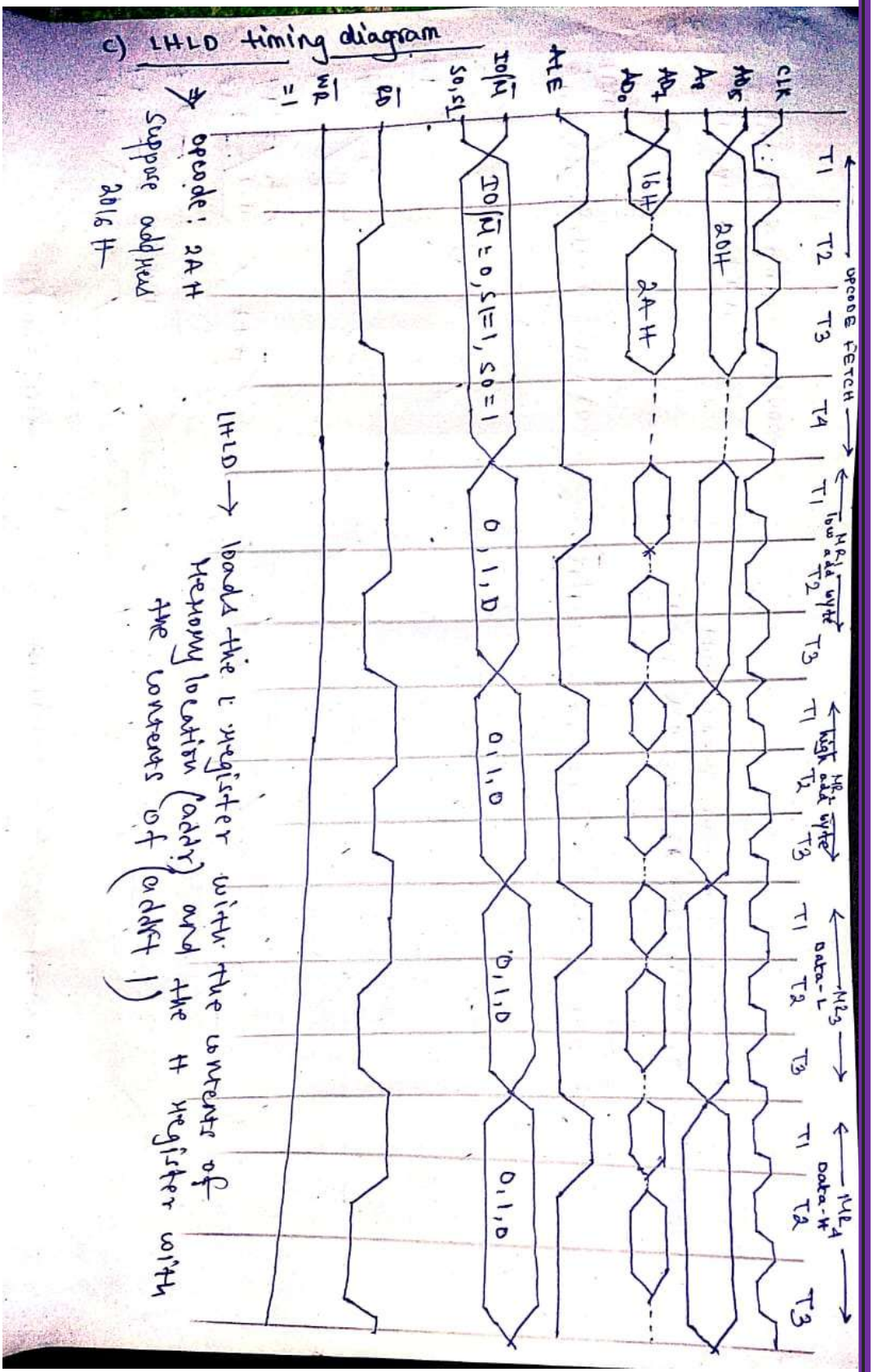
a. **MOV A,M :**



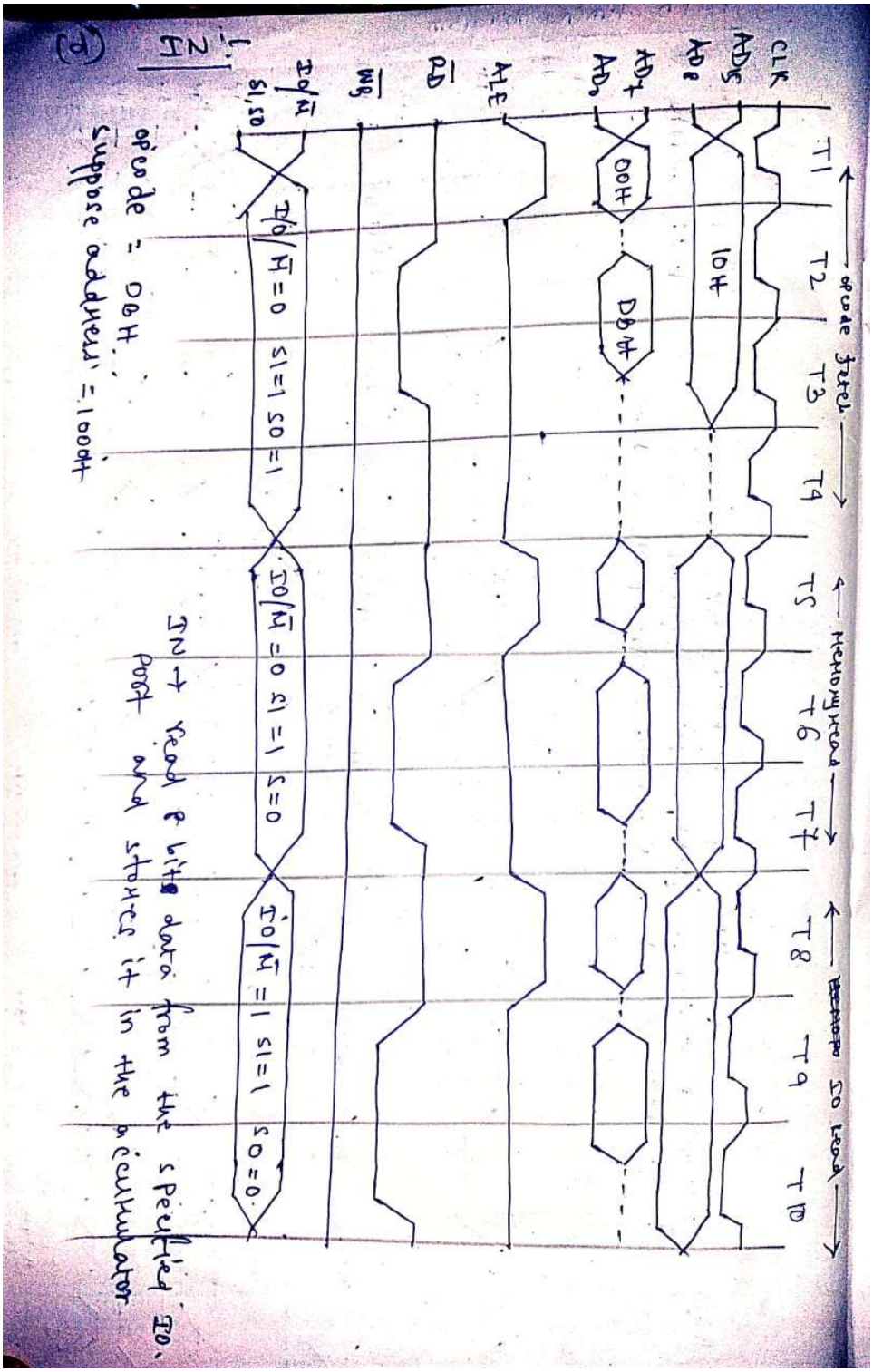
b. **INR B:**


INR B
op code ⇒ 04 H
assume address 2100 H
INR B → increments the contents of register B by 1
CLK
AD7
AD6
AD0
ALE
RD
WR
IO/M
S1,S0
T1
T2
T3
T4
00 H
21 H
04 H
IO/M=0, S1=1 S0=1
ALE: Address latch enable

c. **LHLD**:


c) LHLD timing diagram
CLK
A15
A0
AD7
AD0
ALE
M/IO
S0, S1
RD
WR
IO/M = 0, S1=1, S0=1
0, 1, D
0, 1, D
0, 1, D
0, 1, D
T1
T2
T3
T4
T1
T2
T3
T1
T2
T3
T1
T2
T3
OPCODE FETCH
Low address byte
High address byte
Data-L
Data-H
20H
2AH
16H
LHLD → loads the L register with the contents of memory location (addr) and the H register with the contents of (addr+1)
Opcode: 2AH
Suppose address 2016H

d. **IN :**

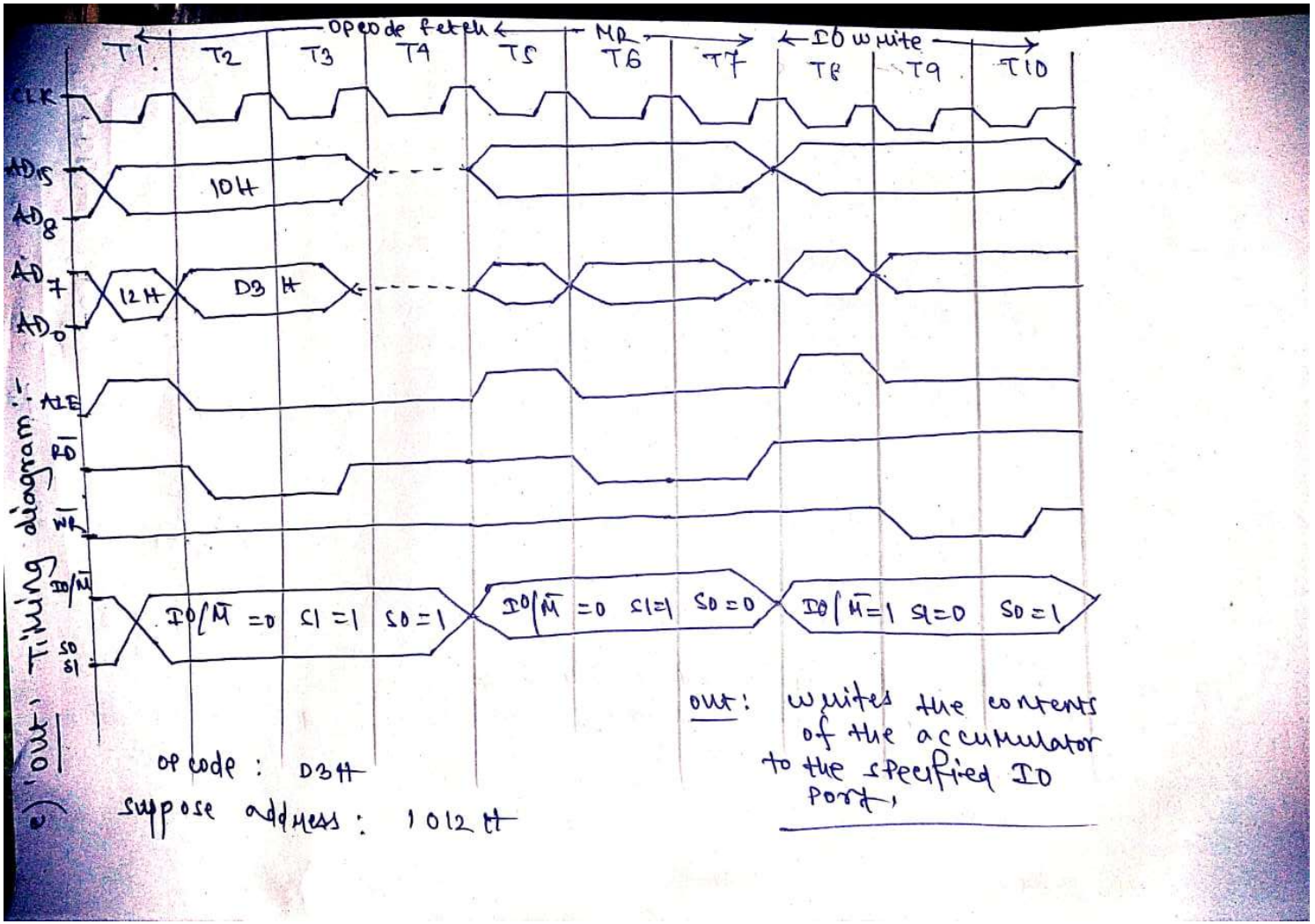


IN → read a byte data from the specified I/O port and stores it in the accumulator.

opcode = DBH.

suppose address = 10H.

e. **OUT:**



e) 'OUT' Timing diagram :-

opcode : D3H

suppose address : 1012 H

OUT : writes the contents of the accumulator to the specified IO port.

2. Explain the function of (2 marks)

a. **ADD :**

Adds the content of a register or memory to the accumulator.
Result is stored in the accumulator and flags are affected.

ADD B

Adds content of register B to Accumulator (A)
Example:
A = 05H, B = 03H → A = 08H

b. STAX:

Stores the content of the accumulator into the memory location whose address is in a register pair (BC or DE).

STAX B

Stores A into memory whose address is in BC
Example:
BC = 2000H, A = 4AH → M[2000H] = 4AH

c. SHLD:

Stores the content of the HL register pair into two consecutive memory locations.
Lower byte (L) → specified address, higher byte (H) → next address.

SHLD 2500H

Stores L at 2500H and H at 2501H
Example:
HL = 1234H
M[2500H] = 34H
M[2501H] = 12H

d. LHLD

Loads the HL register pair with data from two consecutive memory locations.
Lower byte → L register, higher byte → H register.

LHLD 2500H

Loads L from 2500H and H from 2501H
Example:
M[2500H] = 78H
M[2501H] = 56H
HL = 5678H

e. LDAX

Loads the accumulator with data from the memory location whose address is in a register pair (BC or DE).

LDAX D

Loads A from memory pointed by DE

Example:

DE = 3000H

M[3000H] = 9AH → A = 9AH

f. LXI

Loads a 16-bit immediate data into a register pair (BC, DE, HL) or Stack Pointer (SP).

LXI H, 4000H

Loads 4000H into HL register pair

H = 40H, L = 00H

g. DAA

Decimal Adjust Accumulator.

Adjusts the accumulator content to obtain a correct **BCD result** after addition.

ADD B

DAA

Example:
A = 09H, B = 08H
After ADD → A = 11H
After DAA → A = 17H (BCD corrected)

h. CMP

Compares the content of a register or memory with the accumulator.
Accumulator is not changed; only flags are affected.

CMP C

Example:
A = 05H, C = 05H
Result: Zero flag = 1 (A unchanged)

i. CALL

Calls a subroutine.
The address of the next instruction is pushed onto the stack and control transfers to the subroutine.

CALL 2050H

Calls subroutine at **2050H**
Return address is pushed onto stack.

j. RET

Returns from a subroutine.
The return address is popped from the stack and execution resumes from that address.

RET
Returns to the address stored on the stack after subroutine execution

k. XCHG

Exchanges the contents of the HL register pair with the DE register pair.

XCHG
Example:
HL = 1234H
DE = 5678H
After execution:
HL = 5678H
DE = 1234H

l. LDA

Loads the accumulator with data from a specified memory address.

LDA 3000H

Loads **A** from memory location 3000H
Example:
M[3000H] = 4FH → A = 4FH

m. INR

Increments the content of a register or memory by 1.
Flags are affected except the Carry flag.

INR B

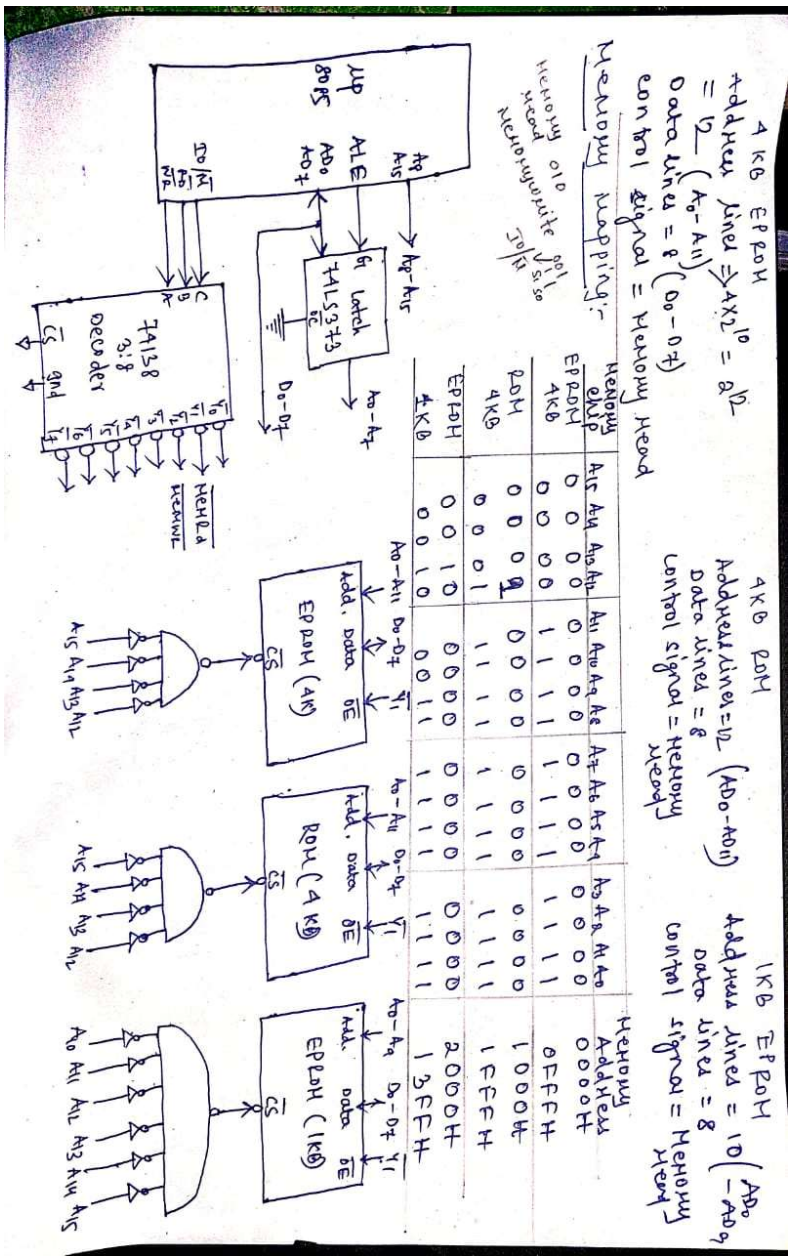
Example:

B = 09H → B = 0AH

Carry flag not affected

ii. *Memory Interface:*

1. Interface 4 KB EPROM , 4KB ROM and 1KB EPROM memory device on 8085 microprocessor in absolute decoding method. (3 Marks each)



2. CALL and RET are any instructions of 8085 microprocessor that uses subroutine , comment on this statement.

The statement is correct.
CALL and RET are 8085 instructions used to implement subroutines. CALL transfers control to a subroutine and stores the return address on the stack, while RET retrieves the return address from the stack and resumes execution of the main program.

3. PUSH , POP are only instructions of 8085 that uses stack , comment on this statement.

This statement is **false**, because **several other 8085 instructions also use the stack**, not only PUSH and POP.
PUSH and POP are explicit stack manipulation instructions in 8085, but CALL, RET, RST, and interrupt operations also use the stack implicitly.

iii. *Interrupt:*

1. What is interrupt?

An **interrupt** is a **signal or event** that **temporarily stops the normal execution of a program** and transfers control to a **special routine called an Interrupt Service Routine (ISR)** to handle that event.

After the ISR is completed, the processor **returns to the main program** from where it was interrupted.

2. Explain different types of interrupt.

a. Based on Source:

1 Hardware interrupts

- **Generated by external devices**
- **Examples:**
  - **TRAP (non-maskable, highest priority)**
  - **RST 7.5**
  - **RST 6.5**
  - **RST 5.5**
  - **INTR**

2 Software interrupts

- **Generated by program instructions**
- **Examples:**
  - **RST 0 to RST 7**

b. Based on Vectoring

1) **Vectored Interrupt**
   The interrupt has a fixed, predefined memory address (vector address) where the ISR is located.

Examples (8085):

- **TRAP → 0024H**
- **RST 7.5 → 003CH**
- **RST 6.5 → 0034H**
- **RST 5.5 → 002CH**
- **RST 0 – RST 7 (software interrupts)**

---

2) **Non-Vectored Interrupt**
   The ISR address is not fixed; it must be supplied externally.

Example (8085):

- **INTR**

---

c. Based on Masking Capability

1) **Maskable Interrupt**
   Interrupts that can be enabled or disabled (masked) by software.

Examples (8085):

- **RST 7.5**
- **RST 6.5**
- **RST 5.5**
- **INTR**

---

2) **Non-Maskable Interrupt**
   Interrupts that cannot be disabled by software.

Example (8085): TRAP

3. WAP to enable all interrupt except RST 5.5

```
MVI A, 09H    ; Mask RST 5.5, unmask RST 6.5 & 7.5
SIM           ; Set interrupt masks
EI            ; Enable all interrupts globally
HLT
```

This program enables all interrupts except RST 5.5 by masking RST 5.5 using SIM and enabling interrupts using EI.

4. WAP to enable RST 7.5 , RST 6.5 and disable RST 5.5

```
MVI A, 09H    ; Enable RST 7.5 & 6.5, disable RST 5.5
SIM           ; Set interrupt masks
EI            ; Enable interrupts globally
HLT
```

This program enables RST 7.5 and RST 6.5 while disabling RST 5.5 using the SIM instruction.

5. WAP to clear / set all the flags of 8085 microprocessor without affecting the content of accumulator.

A. WAP to CLEAR all flags without affecting Accumulator

```
MOV B, A      ; Save accumulator
XRA A         ; Clears CY, AC, S, P (Z = 1)
MOV A, B      ; Restore accumulator (MOV does not affect flags)
HLT
```

B. WAP to SET all flags without affecting Accumulator

```
MOV B, A      ; Save accumulator
MVI A, FFH
ADD A         ; Generates carry and affects flags
MOV A, B      ; Restore accumulator
HLT
```

iv. *Program:*

1. 6 Byte BCD addition.

6-Byte BCD Addition (8085):

Assumption:

- 1st number: 2000H–2005H
- 2nd number: 2006H–200BH
- Result: 2010H onwards

Address	Instruction	Remarks
XXXX	LXI H,2000H	HL → 1st BCD number
XXXX	LXI D,2006H	DE → 2nd BCD number
XXXX	LXI B,2010H	BC → result location
XXXX	MVI C,06H	Counter = 6 bytes
XXXX	STC	
XXXX	CMC	Clear carry
XXXX	LOOP MOV A,M	Get byte from 1st number
XXXX	LDAX D	Get byte from 2nd number
XXXX	ADC M	Add with carry
XXXX	DAA	BCD adjust
XXXX	STAX B	Store result byte
XXXX	INX H	Next source byte
XXXX	INX D	
XXXX	INX B	
XXXX	DCR C	Decrement counter
XXXX	JNZ LOOP	Repeat
XXXX	HLT	Stop

2. 2 Byte BCD subtraction.

Assumption:

- Minuend: 2000H–2001H
- Subtrahend: 2002H–2003H
- Result: 2010H–2011H

```
Address   Instruction     Remarks
-----------------------------------------------
XXXX      LXI H,2000H     HL → minuend
XXXX      LXI D,2002H     DE → subtrahend
XXXX      LXI B,2010H     BC → result
XXXX      STC
XXXX      CMC             Clear borrow

XXXX      MOV A,M         Load LSB of minuend
XXXX      SUB M           Subtract LSB
XXXX      DAA             BCD adjust
XXXX      STAX B          Store LSB result

XXXX      INX H
XXXX      INX D
XXXX      INX B

XXXX      MOV A,M         Load MSB
XXXX      SBB M           Subtract with borrow
XXXX      DAA
XXXX      STAX B          Store MSB result

XXXX      HLT
```

3. 10 Byte Hex number addition.

Assumption

- 1st number: 2000H–2009H
- 2nd number: 200AH–2013H
- Result: 2020H onwards

```
Address   Instruction     Remarks
-----------------------------------------------
XXXX      LXI H,2000H     HL → 1st HEX number
XXXX      LXI D,200AH     DE → 2nd HEX number
XXXX      LXI B,2020H     BC → result
XXXX      MVI C,0AH       Counter = 10 bytes
XXXX      STC
```

```
XXXX    CMC          Clear carry

XXXX LOOP MOV A,M    Load byte from 1st number
XXXX    LDAX D       Load byte from 2nd number
XXXX    ADC M        Add with carry
XXXX    STAX B       Store result
XXXX    INX H
XXXX    INX D
XXXX    INX B
XXXX    DCR C
XXXX    JNZ LOOP

XXXX    HLT
```

4. Six 8 bit data stored of memory location 6070 H. Transfer the entire block of data to new memory location starting from 6050H.

Problem

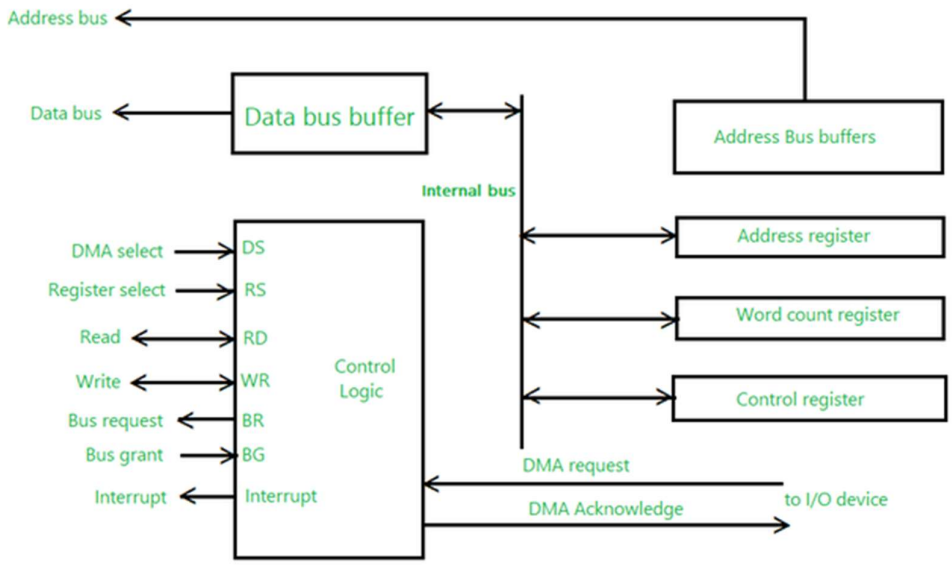
- Source: **6070H–6075H**
- Destination: **6050H–6055H**

```
Address   Instruction     Remarks
-------------------------------------------
XXXX      LXI H,6070H     HL → source address
XXXX      LXI D,6050H     DE → destination
XXXX      MVI C,06H       Counter = 6 bytes

XXXX LOOP MOV A,M         Read source byte
XXXX      STAX D          Store at destination
XXXX      INX H           Next source
XXXX      INX D           Next destination
XXXX      DCR C           Decrement counter
XXXX      JNZ LOOP        Repeat

XXXX      HLT
```

5. Draw and explain the DMA operation.



DMA Operation Steps:

The process generally involves a dedicated hardware component called a DMA Controller (DMAC) (e.g., the Intel 8257 or 8237 for the 8085 microprocessor).

1. **Initialization: The CPU prepares the DMA controller by providing the starting memory address, the number of bytes to transfer (word count), and the direction (Read or Write).**
2. **Request (DRQ): When an I/O device is ready to send or receive data, it sends a DMA Request (DRQ) signal to the DMA controller.**
3. **Hold Request (HOLD): The DMA controller sends a HOLD signal to the CPU, asking for control of the system buses (Address, Data, and Control buses).**
4. **Hold Acknowledgment (HLDA): The CPU finishes its current bus cycle, puts its bus pins in a high-impedance (tri-state) mode to "disconnect" from them, and sends a HLDA signal back to the controller.**
5. **Data Transfer: The DMA controller now acts as the master. It places the memory address on the bus and manages the data flow directly between the device and RAM.**
6. **Release: Once the transfer is complete (word count reaches zero), the DMAC drops the HOLD signal, and the CPU resumes control of the buses.**

v. 8085 Architecture:

1. What is microprocessor ?

   A **microprocessor** is a **single IC (Integrated Circuit)** that **fetches, decodes, and executes instructions** to perform arithmetic, logical, and control operations.

2. **Why is the data bus bidirectional?** ***(2 Marks)***

   The data bus is **bidirectional** because the microprocessor must **send data to memory/I/O devices** during write operations and **receive data from memory/I/O devices** during read operations.

3. What is vonn-neuman architecture?

   Von Neumann architecture is a computer design where **instructions and data share a single memory and a single set of buses**.

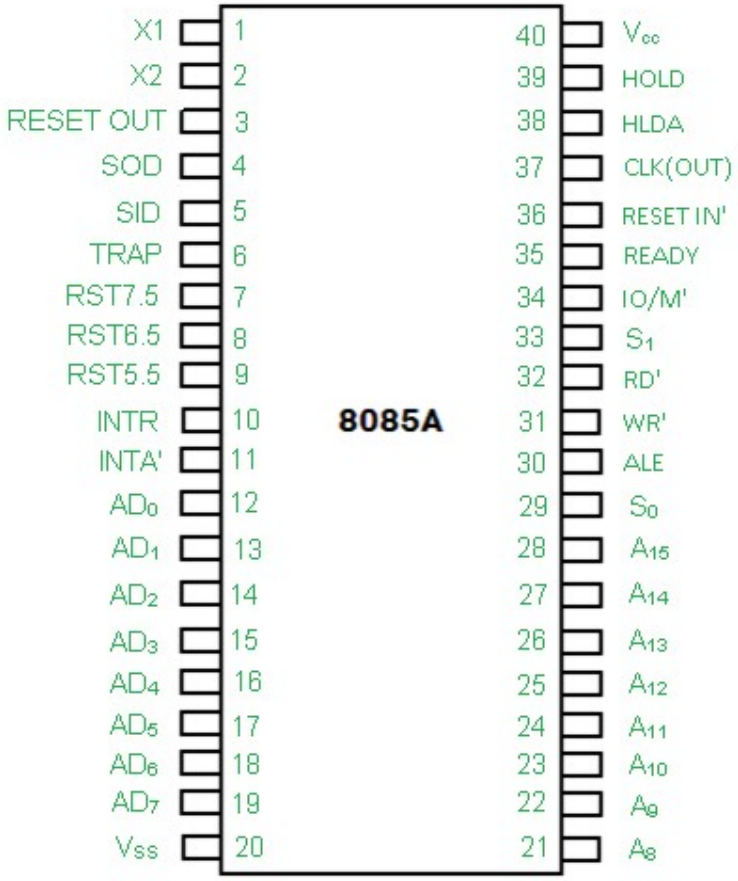
4. What are the difference between 8086 and 8088 microprocessor?

Feature	8086	8088
Data bus width	**16-bit**	**8-bit**
External architecture	16-bit	8-bit
Instruction queue size	**6 bytes**	**4 bytes**
Speed	Faster	Slower (compared to 8086)
Cost	More expensive	Cheaper
Pin configuration	More complex	Simpler
Used with memory	16-bit memory	8-bit memory

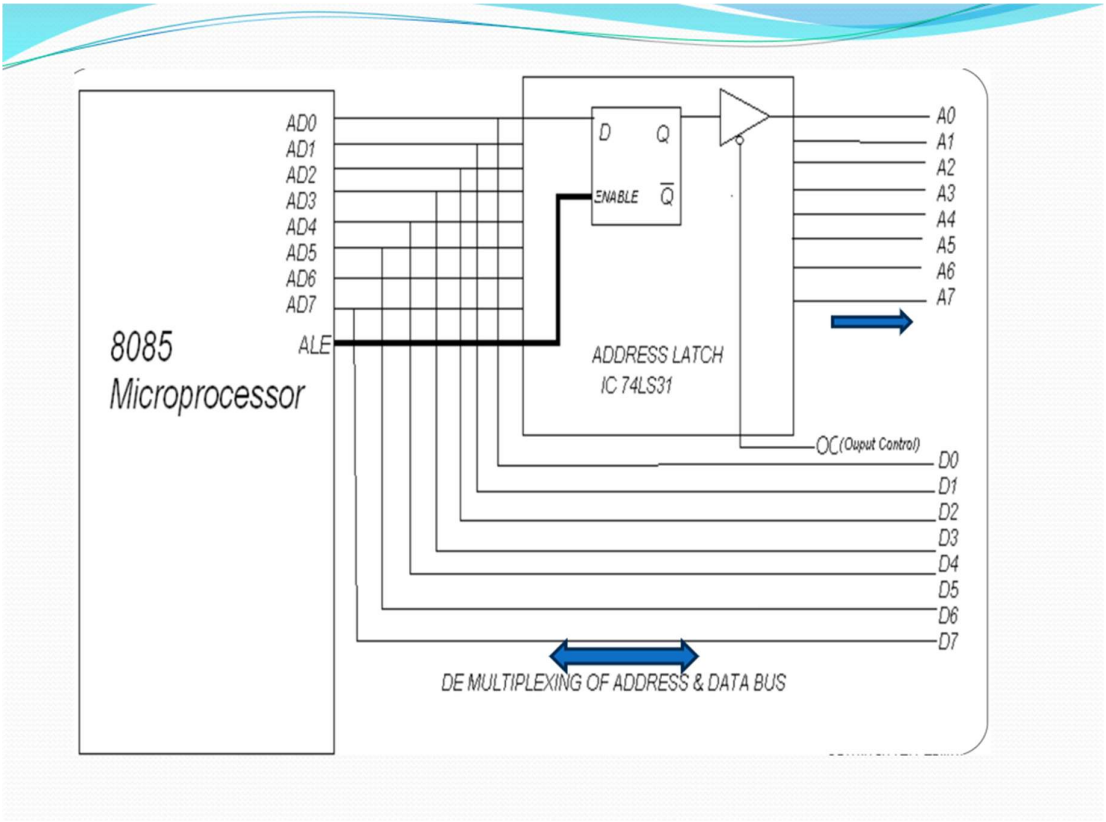
5. Sketch and explain the various pin of 8085 microprocessor.

   The **8085 microprocessor** has a **40-pin IC**. Its pins are divided into functional groups:

- **Address/Data Bus:**
  AD0–AD7 are multiplexed lines used for lower address and data; **A8–A15** carry higher address bits.
- **Control & Status Signals:**
  ALE, $\overline{RD}$, $\overline{WR}$, IO/$\overline{M}$, S0, S1 control and indicate memory and I/O operations.
- **Interrupt Signals:**
  TRAP, RST 7.5, RST 6.5, RST 5.5, INTR, $\overline{INTA}$ handle hardware interrupts.
- **Serial I/O:**
  SID (serial input) and **SOD** (serial output).
- **DMA Signals:**
  HOLD and **HLDA** support direct memory access.
- **Clock & Reset:**
  X1, X2, CLK OUT, RESET $\overline{IN}$, RESET OUT manage timing and initialization.
- **Power Supply:**
  VCC (+5V) and **VSS (GND)**.



6. Explain the demultiplexing of the address and data multiplex bus with the help of proper diagram.



In **8085**, the lower address bus **AD0–AD7 (Multiplexed Lines)** is **multiplexed** with the data bus to reduce pin count.

To separate (demultiplex) them, an **external latch (74LS31)** is used.

- During **T1**, **AD0–AD7(Multiplexed Lines) carry address (A0–A7)** and **ALE = 1**.
- The latch stores this address.
- After T1, **ALE = 0**, and **AD0–AD7 are used as data lines**.
- The latched output provides a **stable address** while data transfer occurs.

7. Explain the function of ALE pin of 8085 microprocessor.

ALE(Address Latch Enable)

- It is an output signal used to give information of AD_0-AD_7 contents.
- This is a positive going pulse generated any time the 8085 begins its operation.
- When pulse goes high it indicates that AD_0-AD_7 lines are address.
- When it is low it indicates that the contents are data.
- This signal is used primarily to latch the lower order address from the multiplexed bus and generate a separate set of 8 bit address lines.

8. Describe the different addressing mode of 8085/8086 microprocessor.

Addressing Modes of 8085 Microprocessor: (5)

Addressing mode defines **how an operand (data) is specified in an instruction**.

1. Immediate Addressing Mode:

- Data is given **within the instruction**
- Fast execution

Example:

MVI A, 25H

ADI 05H

---

2. Register Addressing Mode:

- Operand is stored in a **register**

Example:

MOV A, B

ADD C

---

3. Direct Addressing Mode:

- **16-bit memory address** is directly specified

Example:

LDA 2050H
STA 3050H
LHLD 2500H

---

4. Register Indirect Addressing Mode:

- Address of data is stored in a **register pair**

Example:

MOV A, M ; M → memory pointed by HL
LDAX B ; memory pointed by BC
STAX D ; memory pointed by DE

---

5. Implied (Implicit) Addressing Mode:

- Operand is **implied**, not explicitly mentioned

Example:

CMA → Complement Accumulator
RAL → Rotate Accumulator left
RRC → Rotate Accumulator right

Addressing Modes of 8086 Microprocessor :(8)

8086 supports **more advanced addressing modes**.

---

1. Immediate Addressing Mode:

- Constant data is part of instruction

Example:

MOV AX, 1234H
ADD AL, 05H

---

2. Register Addressing Mode:

- Operand is in a register

Example:

MOV AX, BX
ADD CX, DX

---

3. Direct Addressing Mode:

- Offset address is directly given

Example:

MOV AX, [1234H]

---

4. Register Indirect Addressing Mode:

- Address stored in **SI, DI, BX, or BP**

Example:

MOV AX, [BX]

MOV AL, [SI]

---

5. Based Addressing Mode:

- Effective address = **Base register + displacement (Offset)**

Example:

MOV AX, [BX+10H]

---

6. Indexed Addressing Mode:

- Effective address = **Index register + displacement (Offset)**

Example:

MOV AX, [SI+05H]

---

7. Based Indexed Addressing Mode:

- Effective address = **Base register + Index register**

Example:

MOV AX, [BX+SI]

8.Implied Addressing Mode:

- Operand is implied

Example:

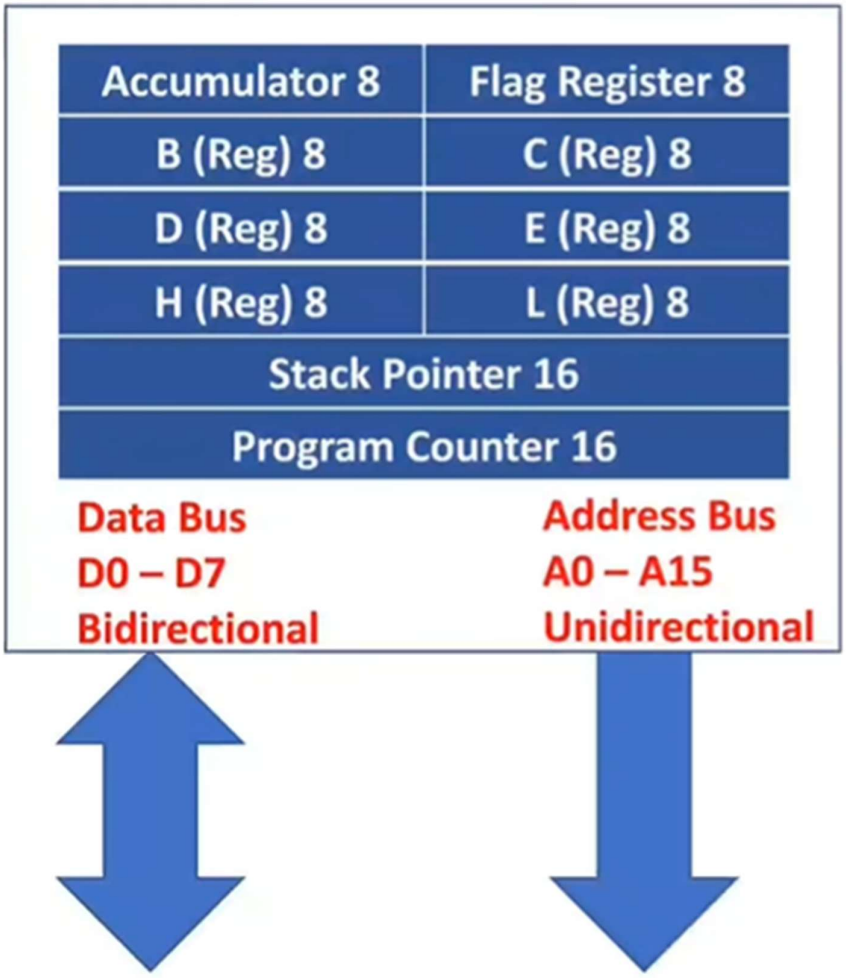
CLC

STC

DAA

9. Draw and explain software and hardware model of 8085microprocessor.

Software Model:



Accumulator (A – 8 bit): Main register used for arithmetic and logical operations.

Flag Register (8 bit): Contains status flags — **S, Z, AC, P, CY** — indicating result conditions.

General Purpose Registers (8 bit): B, C, D, E, H, L used for data storage. Can be paired as **BC, DE, HL** for 16-bit operations.

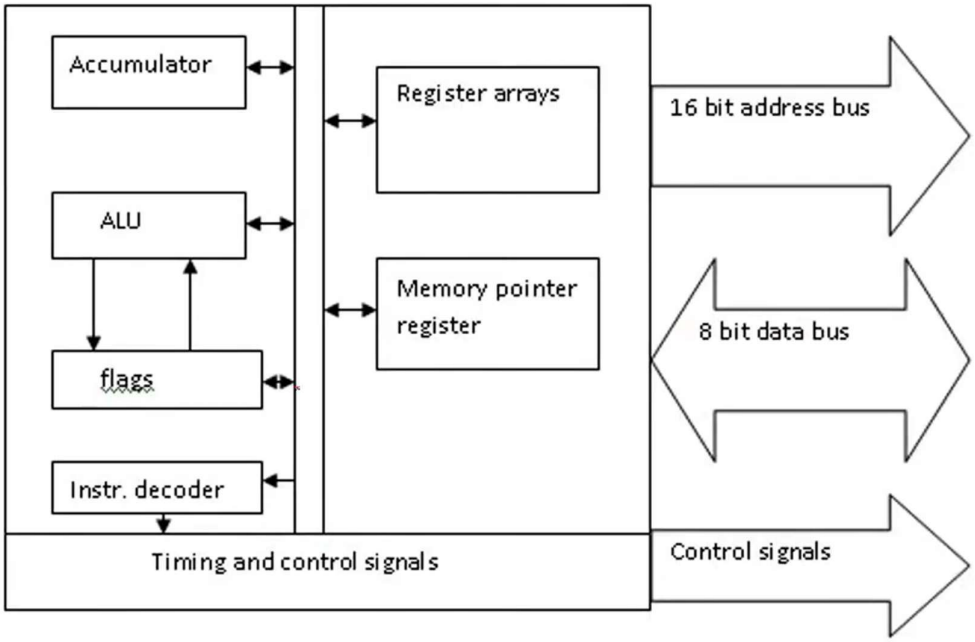
Stack Pointer (SP – 16 bit): Holds the address of the top of the stack.

Program Counter (PC – 16 bit): Stores the address of the next instruction to be executed.

Data Bus (D0–D7): 8-bit **bidirectional** bus used to transfer data.

Address Bus (A0–A15): 16-bit **unidirectional** bus used to specify memory or I/O addresses.

Hardware Model:



- **Accumulator:** Stores operands and results of arithmetic and logical operations.
- **ALU (Arithmetic Logic Unit):** Performs arithmetic and logical operations.
- **Flags:** Store status of ALU results (Sign, Zero, Auxiliary Carry, Parity, Carry).
- **Register Array:** Includes general-purpose registers used for temporary data storage.
- **Memory Pointer Register:** Holds the address of memory locations (e.g., PC, SP, HL internally).
- **Instruction Decoder:** Decodes the fetched instruction and generates required actions.
- **Timing and Control Unit:** Generates control and timing signals for all operations.
- **16-bit Address Bus:** Sends address from microprocessor to memory/I/O (unidirectional).
- **8-bit Data Bus:** Transfers data between microprocessor and memory/I/O (bidirectional).
- **Control Signals:** Manage read, write, and other operations.

10. Draw and explain the flag structure of 8085 and 8086 microprocessor.

8085:

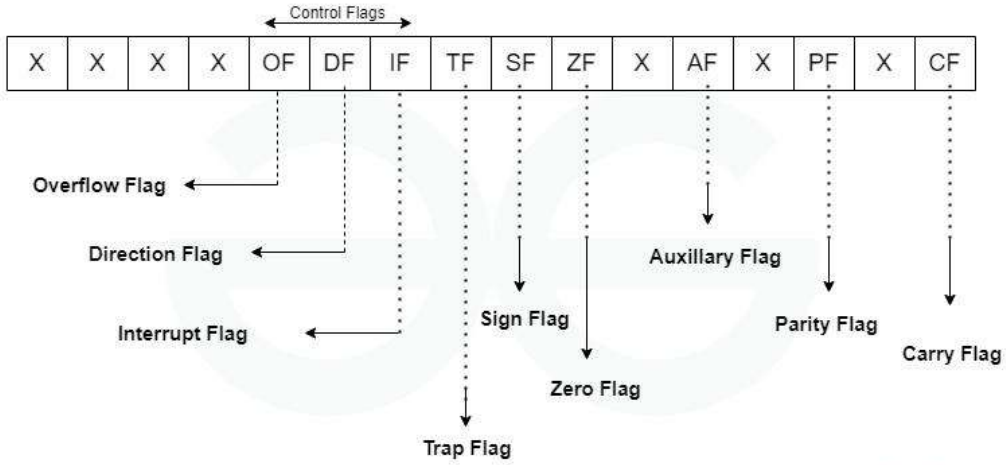


The **8085 flag register** is an **8-bit register** that shows the status of ALU operations.

- **S (Sign):** Result is negative
- **Z (Zero):** Result is zero
- **AC (Auxiliary Carry):** Carry from bit D3 to D4
- **P (Parity):** Even parity
- **CY (Carry):** Carry from MSB

Unused bits: **D5, D3, D1**

8086:



The **8086 flag register** is a **16-bit register** that indicates the status of operations and controls CPU behaviour.

Types of Flags:

1. Status Flags:

- **CF (Carry):** Carry or borrow generated
- **PF (Parity):** Even parity of result
- **AF (Auxiliary Carry):** Carry from bit 3 to bit 4
- **ZF (Zero):** Result is zero
- **SF (Sign):** MSB of result is 1
- **OF (Overflow):** Signed overflow occurs

2. Control Flags:

- **TF (Trap):** Enables single-step execution
- **IF (Interrupt):** Enables/disables maskable interrupts
- **DF (Direction):** Controls string operation direction

Unused bits are reserved.

11. Explain the instruction of 8085 microprocessor that uses auxiliary carry flag.

Instruction of 8085 that uses Auxiliary Carry (AC) flag : DAA

DAA – Decimal Adjust Accumulator

- The **DAA** instruction uses the **Auxiliary Carry (AC)** flag.
- It is used after addition of two **BCD (Binary Coded Decimal)** numbers.
- If:
  - Lower nibble of accumulator > 9 **or** AC = 1 → **6 is added** to lower nibble
  - Upper nibble > 9 **or** Carry (CY) = 1 → **6 is added** to upper nibble
- This converts the binary sum into a valid **BCD result**.

Flags affected:

- **AC, CY, Z, S, P**

Example:

MVI A, 29H
ADI 34H
DAA
Result in A = **63H** (valid BCD)

12. Explain the function of memory point register SP and PC and why they are 16 bit register.

Stack Pointer (SP) and Program Counter (PC) in 8085:

1. Stack Pointer (SP):

- **SP** is a **16-bit memory pointer register**.
- It holds the **address of the top of the stack**.
- Stack is used for:
  - Temporary data storage
  - Subroutine calls (CALL, RET)
  - Interrupt handling
- In 8085, the stack grows **downward** (SP is decremented before storing data).

2. Program Counter (PC):

- **PC** is a **16-bit memory pointer register**.
- It stores the **address of the next instruction** to be fetched and executed.
- After fetching an instruction, PC is **automatically incremented**.

Why SP and PC are 16-bit registers?

- 8085 has a **16-bit address bus**.
- It can access **64 KB memory (0000H – FFFFH)**.
- To point to **any memory location**, SP and PC must store a **16-bit address**.

vi. ***Time Delay:***

1. Write a program to generate 1 second , 10 mili second , 10 micro second time delay considering the operating freq. of 8085 microprocessor is 2 MHz or 4 MHz.

Given
8085 Microprocessor
Clock frequency:

- **2 MHz** → 1 T-state = **0.5 µs**
- **4 MHz** → 1 T-state = **0.25 µs**

Instruction T-states (used)

Instruction	T-states
NOP	4
MVI r, data	7
DCR r	4
JNZ addr	10 (jump taken)

Loop delay per iteration = **DCR + JNZ = 14 T-states**

(A) 10 µs Time Delay

---

1 For 2 MHz

Required T-states:

$$10 \div 0.5 = 20 \text{ T-states}$$

Program:

NOP
NOP
NOP
NOP
NOP

Total T-states = 5 × 4 = **20**

Delay = **10 µs**

---

2 For 4 MHz

Required T-states:

$$10 \div 0.25 = 40 \text{ T-states}$$

Program:

NOP
NOP
NOP
NOP
NOP
NOP
NOP
NOP
NOP
NOP

Total T-states = 10 × 4 = **40**
Delay = **10 µs**

(B) 10 ms Time Delay

---

1 For 2 MHz
Required T-states:

$$10{,}000 \div 0.5 = 20{,}000 \text{ T-states}$$

Loop count:

$$20{,}000 \div 14 \approx 1428$$

Program:

```
MVI B, 94H     ; Counter
LOOP1: DCR B
    JNZ LOOP1
```

Delay ≈ **10 ms**

---

2 For 4 MHz
Required T-states:

$$10{,}000 \div 0.25 = 40{,}000 \text{ T-states}$$

Loop count:

$$40{,}000 \div 14 \approx 2857$$

Program:

```
MVI B, B2H
LOOP2: DCR B
    JNZ LOOP2
```

Delay ≈ **10 ms**

(C) 1 Second Time Delay

---

1 For 2 MHz
Required T-states:

$$1{,}000{,}000 \div 0.5 = 2{,}000{,}000 \text{ T-states}$$

Using **nested loops**:
Outer loop = 100
Inner loop = 1428
Program:

```
MVI C, 64H      ; Outer loop count = 100
OUTER1: MVI B, 94H
INNER1: DCR B
        JNZ INNER1
        DCR C
        JNZ OUTER1
```
Delay ≈ **1 second**

---

2 For 4 MHz

Required T-states:

$$1{,}000{,}000 \div 0.25 = 4{,}000{,}000 \text{ T-states}$$

Outer loop = 100

Inner loop = 2857

Program:

```
MVI C, 64H
OUTER2: MVI B, B2H
INNER2: DCR B
        JNZ INNER2
        DCR C
        JNZ OUTER2
```
Delay ≈ **1 second**

2. Difference between memory mapped I/O & Peripheral mapped I/O.

Memory Mapped I/O	**Peripheral (I/O) Mapped I/O**
I/O devices are treated as **memory locations**	I/O devices have **separate I/O address space**
Uses **memory instructions** (MOV, LDA, STA etc.)	Uses **IN and OUT** instructions
16-bit address is used	**8-bit address** is used
Consumes **part of memory address space**	Does **not reduce memory space**
More instructions available for data transfer	Limited instructions (IN, OUT only)

Memory Mapped I/O	Peripheral (I/O) Mapped I/O
Slower compared to I/O mapped	Faster compared to memory mapped

Example: 8086 uses memory mapped I/O
Example: 8085 supports both methods

3. Design a 7 segment common anode LED output port with device address F5 hex and 8 bit input port containing 8 number of switches with device address F8. WAP to access the switches and display its value to LED.

(a) Design of I/O ports

Given

- **7-segment LED (Common Anode) output port address = F5H**
- **8-bit input port (8 switches) address = F8H**
- 8085 uses **I/O mapped I/O**

---

1 7-Segment Common Anode LED (Output Port – F5H)

- Common anode → **LOW (0)** lights a segment
- Data bus D0–D6 connected to segments **a–g**
- D7 unused or DP
- Port selected when **address = F5H**
- Use **OUT instruction**

---

2 8-bit Input Port (Switches – F8H)

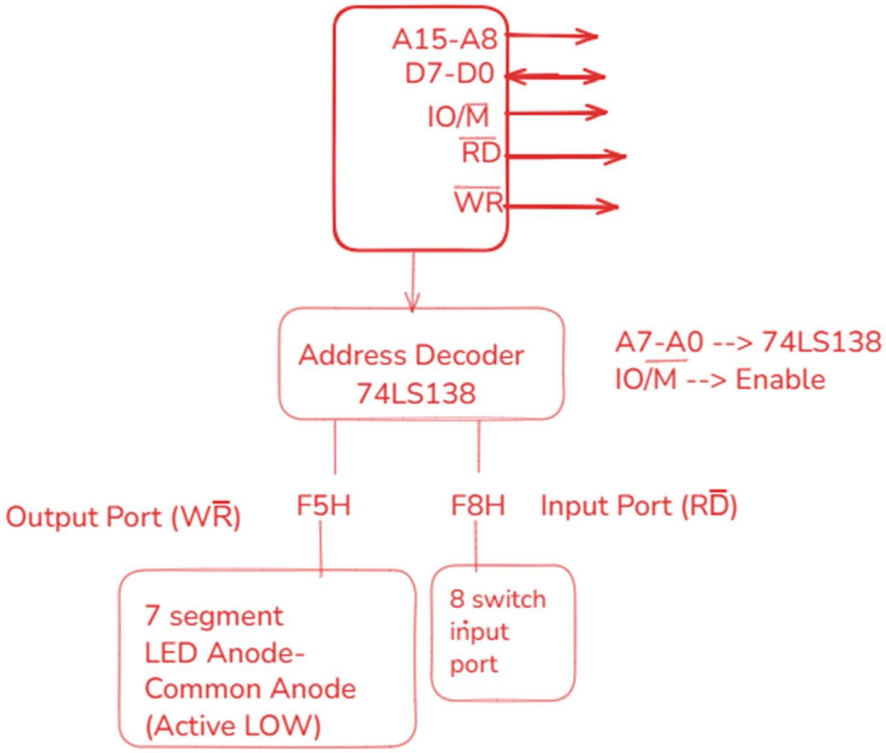
- 8 switches connected to D0–D7
- Logic:
  - Switch ON → 1
  - Switch OFF → 0
- Port selected when **address = F8H**
- Use **IN instruction**

(b) Working

1. Read switch value from input port **F8H**
2. Store data in accumulator
3. Output the same data to **7-segment LED port F5H**
4. LED displays the corresponding pattern

(c) 8085 Assembly Language Program

```
START:  IN   F8H     ; Read 8-bit data from switches
        OUT  F5H     ; Send data to 7-segment LED
        JMP  START   ; Continuous display
```



Or

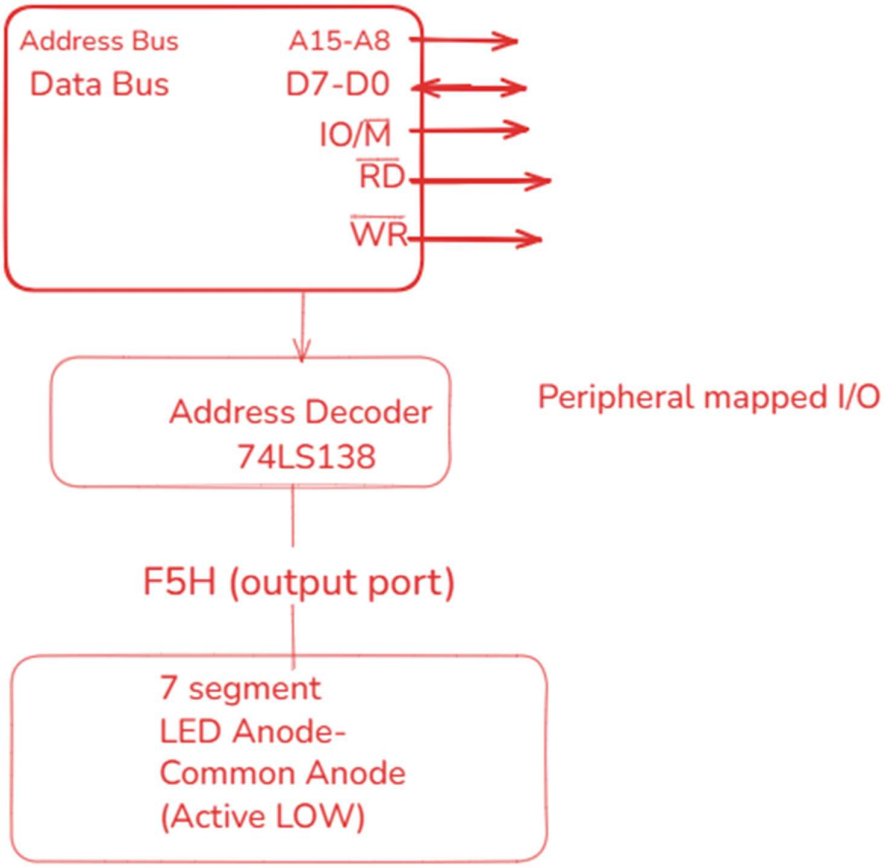
Design a 7 segment LED output port with device address F5 hex using a 3:8 decoder , 4 input NAND , NOR and a common anode 7 segment LED.
WAP to display digit 7 at the port.
Can the same port address be assigned to 2 I/O device if interface in peripheral mapped I/O ? if yes, how? If no , why?

Design of 7-segment LED output port (F5H)

- Peripheral-mapped I/O is used.
- Address **F5H** is decoded using **3:8 decoder (74LS138)**.
- Address lines **A7–A5** are connected to decoder inputs.
- **IO/$\overline{M}$ and W$\overline{R}$** are used to enable the decoder.
- Decoder output **Y7** is used as **chip select**.
- A **4-input NAND** and **NOR gate** are used for control signal generation.
- Output latch drives a **common anode 7-segment LED**.
- Segment ON when logic **0** is applied.

Hex code for digit 7 (Common Anode)

Segments ON: **a, b, c**
Segments OFF: **d, e, f, g, dp**
dp g f e d c b a
1 1 1 1 1 0 0 0 = F8H



8085 Program to display digit 7:

```
MVI A, F8H     ; Load code for digit 7
OUT F5H        ; Send to 7-segment LED port
HLT
```

Can the same port address be assigned to two I/O devices?

No.

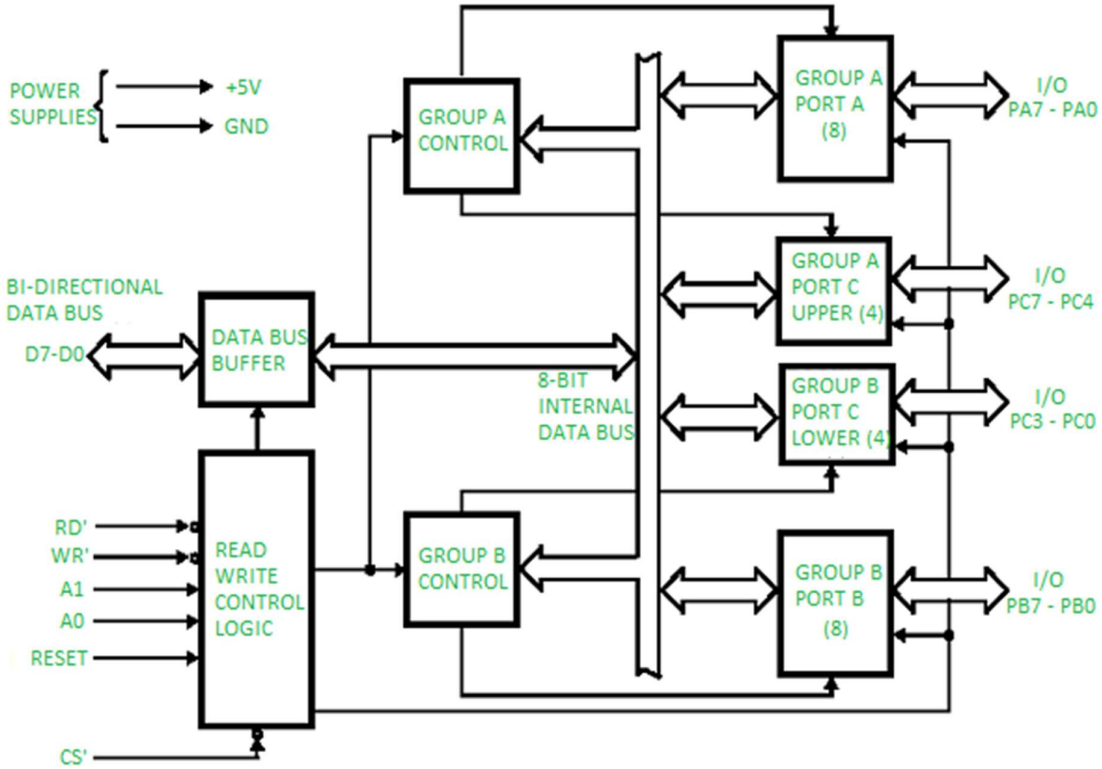
In peripheral-mapped I/O:

- One port address selects **one device only**.
- If two devices share the same address, **both respond simultaneously**.
- This causes **bus contention and incorrect operation**.

Therefore, **one port address must be unique**.

vii. ***8255:***

1. Draw and explain functional block diagram of 8255.



The **8255** is used to interface input/output devices with a microprocessor.

Main Functional Blocks:

1. **Data Bus Buffer**
- 8-bit bidirectional buffer.
- Transfers data between **CPU and 8255**.
- Receives control word and sends/receives I/O data.

2. **Read/Write Control Logic**
- Controls internal operations of 8255.
- Uses signals: **$\overline{CS}$, $\overline{RD}$, $\overline{WR}$, A0, A1**
- Selects **Port A, Port B, Port C, or Control Register**.

3. **Group A Control**
- Controls **Port A and Port C (PC7–PC4)**.
- Operates in **Mode 0, Mode 1, or Mode 2**.

4. **Group B Control**
- Controls **Port B and Port C (PC3–PC0)**.
- Operates in **Mode 0 or Mode 1**.

5. **Port A**
- 8-bit programmable I/O port.
- Used mainly for **data transfer**.

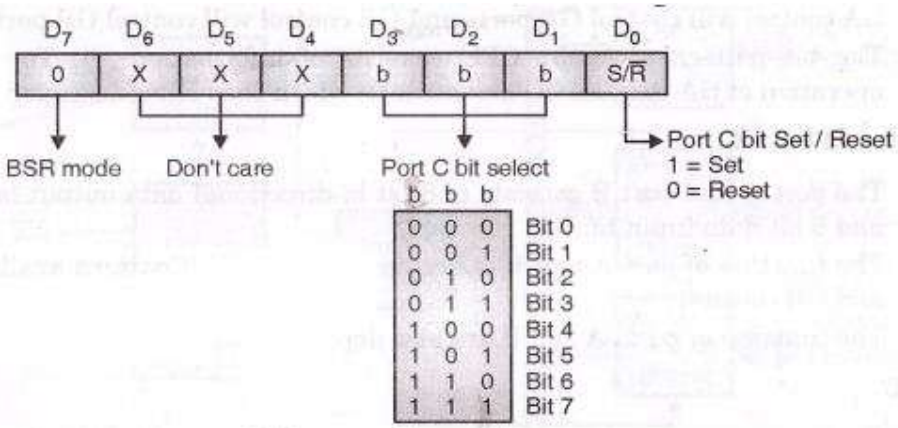
6. **Port B**

- 8-bit programmable I/O port.
- Used for **simple input/output**.

7. **Port C**

- 8-bit port divided into **upper (PC7–PC4)** and **lower (PC3–PC0)**.
- Used as:
  - I/O port
  - Control and handshake signals

2. Draw and explain the control word format of 8255.



BSR control word format

3. Write a control word for port A and port C as a input port and port B as an output port.

Given:

- **Port A → Input**
- **Port B → Output**
- **Port C → Input**
- Use **I/O mode (Mode 0)** (Default)

8255 Control Word Format (I/O Mode)

D7 D6 D5 D4 D3 D2 D1 D0

| | | | | | | | |

1 M1 M0 PA PCU MB PB PCL

Where:

- **D7 = 1** → I/O mode
- **D6–D5** → Mode of Group A
- **D4** → Port A (1 = Input, 0 = Output)
- **D3** → Port C upper (PC7–PC4)
- **D2** → Mode of Group B
- **D1** → Port B
- **D0** → Port C lower (PC3–PC0)

Setting the bits:

- **D7 = 1** → I/O mode
- **Group A Mode = 00** → Mode 0
- **Port A = Input → D4 = 1**
- **Port C (upper) = Input → D3 = 1**
- **Group B Mode = 0 → D2 = 0**
- **Port B = Output → D1 = 0**
- **Port C (lower) = Input → D0 = 1**

Control Word Bits:

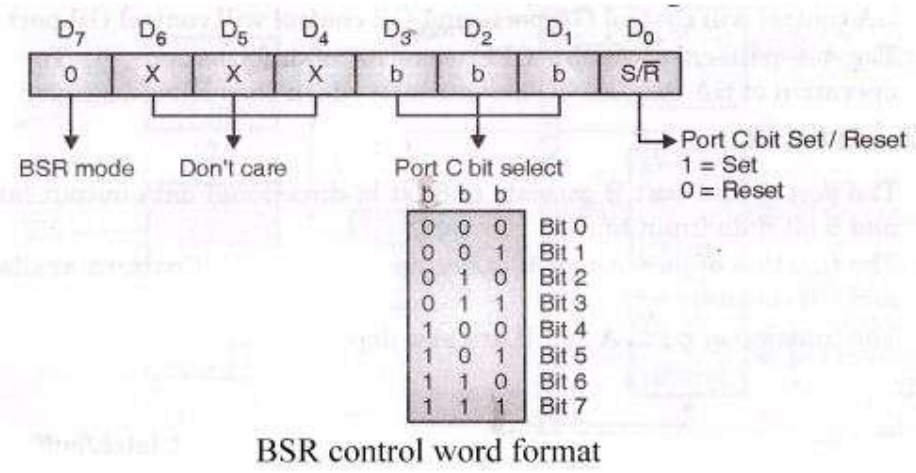
D7 D6 D5 D4 D3 D2 D1 D0

1 0 0 1 1 0 0 1

Control Word

- **Binary:** 10011001
- **Hexadecimal: 99H**

4. Explain the BSR mode of 8255.



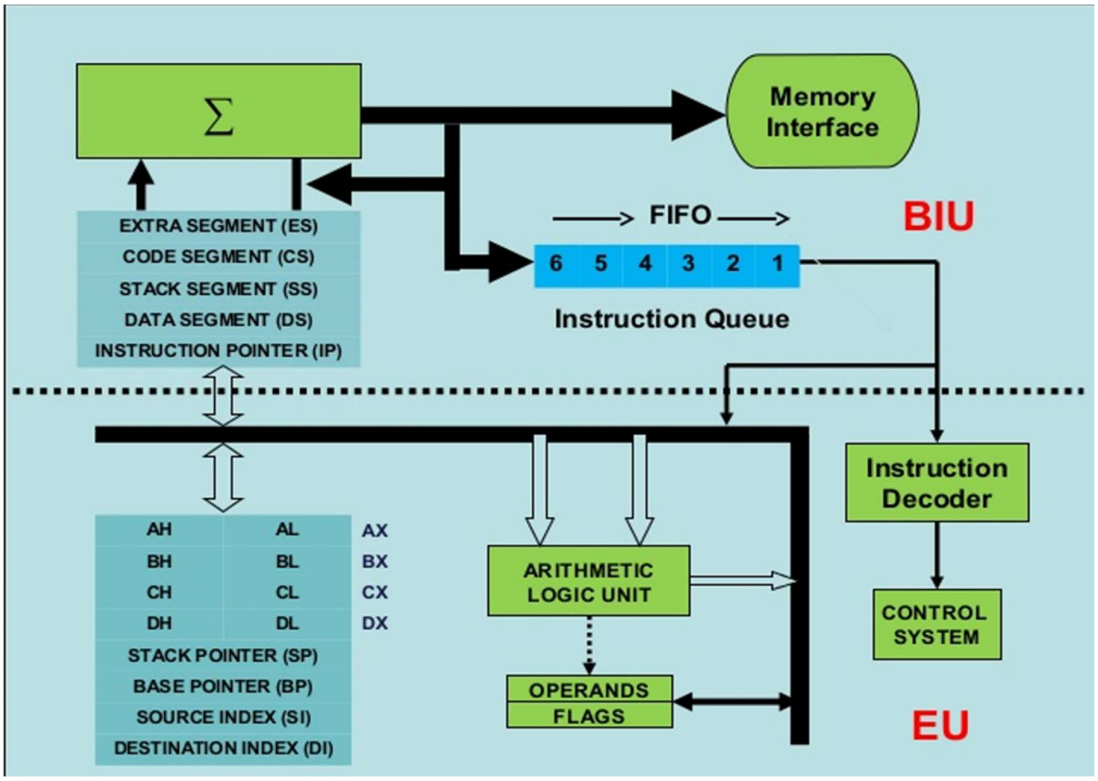
BSR control word format

- **BSR mode** stands for **Bit Set/Reset mode**.
- It is used to **set or reset individual bits of Port C**.
- In this mode, **Port A and Port B are not affected**.

- Only **Port C bits (PC0–PC7)** are controlled.
- Each bit can be **set (1)** or **reset (0)** independently.
- No effect on I/O modes of other ports.

viii. ***8086:***

1. Draw the architecture of 8086 microprocessor and discuss in behalf the BIU and EU of 8086 microprocessor.



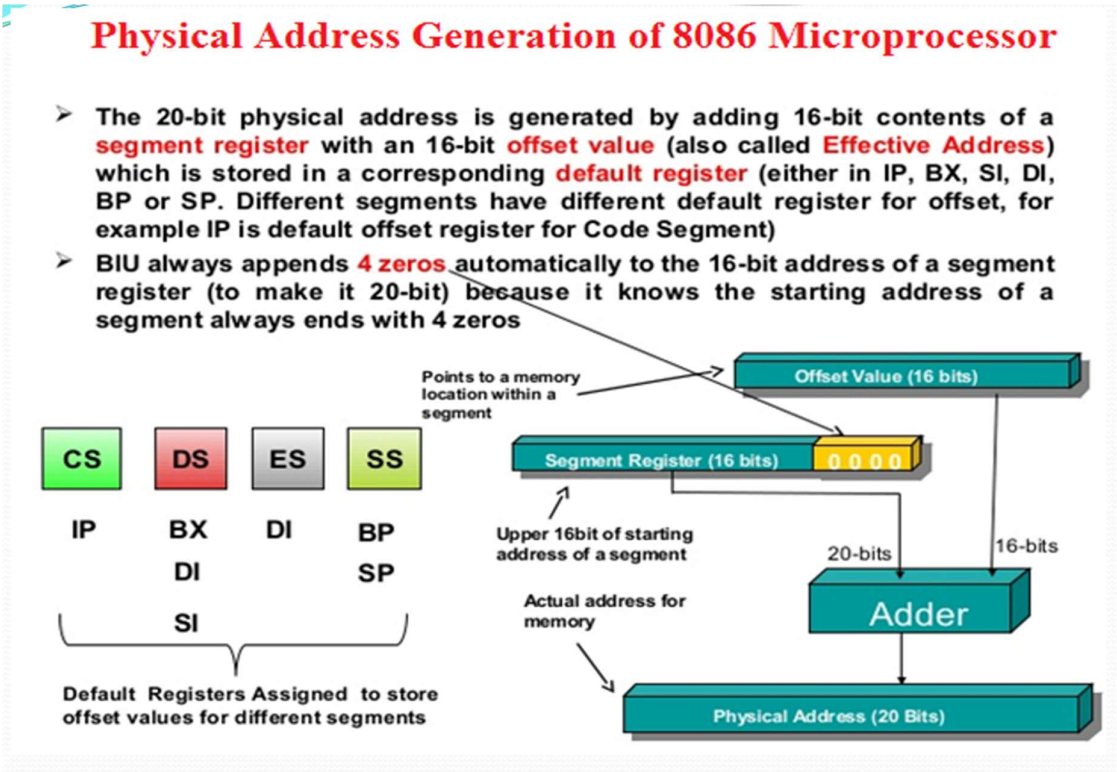
Bus Interface Unit:

Sends out addresses for memory locations
Fetches Instructions from memory
Reads/Writes data to memory
Sends out addresses for I/O ports
Reads/Writes data to Input/output ports

Execution Unit:

Tells BIU (addresses) where to fetch instructions or data.
Decodes & Executes instructions fetched by BIU unit.
Dividing the work between BIU & EU speeds up processing.
Generates control signals.

2. How physical address of 8086 microprocessor is calculated?



3. Explain the function of Queue register, AX ,BX, CX, DX register.

Register	Name of register	Function of register
AX	16-bit Accumulator	Stores the 16-bit results of arithmetic and logic operations
BX	Base register	Used to hold base value in base addressing mode to access memory data
CX	Count Register	Used to hold the count value in SHIFT, ROTATE and LOOP instructions
DX	Data Register	Used to hold data for multiplication and division operations

4. Explain the process of segmentation memory in 8086 microprocessor , what are it's advantages?

Process of Memory Segmentation

- The **8086** has a **1 MB memory space** (20-bit address bus).
- Memory is divided into logical blocks called **segments**.
- Each segment is **maximum 64 KB** in size.
- Main segments:
  - **Code Segment (CS)** – stores program instructions
  - **Data Segment (DS)** – stores data
  - **Stack Segment (SS)** – stores stack data
  - **Extra Segment (ES)** – additional data segment

  Address formation (Physical Address):

  Physical Address = Segment Address × 10H + Offset

- Segment address is stored in **segment registers** (CS, DS, SS, ES).
- Offset is stored in **IP, SP, BP, SI, DI**, etc.

Example:

CS = 2000H, IP = 0010H

Physical address = 2000H × 10H + 0010H = **20010H**

Advantages of Memory Segmentation

1. Allows access to **1 MB memory** using **16-bit registers**.
2. **Logical separation** of code, data, and stack.
3. Better **program organization and modularity**.
4. **Efficient memory utilization**.
5. Supports **relocatable programs**.

ix. ***IMPORTANT PYQs – PYQ(2024)***

x. ***Class Test questions and answers-***