Microprocessor & Assembly Language Programming

Dr. Jamuna Kanta Sing

Department of Computer Science and Engineering
Jadavpur University
188, Raja S. C. Mallik Road
Kolkata 700032, India
Email: jksing@ieee.org

Course Outcomes (COs): The student will be able to …

1) Understand the basic architecture of 8-bit and 16-bit microprocessors with functional activities of different units.
2) Write assembly language programs for solving scientific problems.
3) Know the different steps for execution of an instruction and associated control signals.
4) Design system and write the associated programs for communications between microprocessor and peripherals.

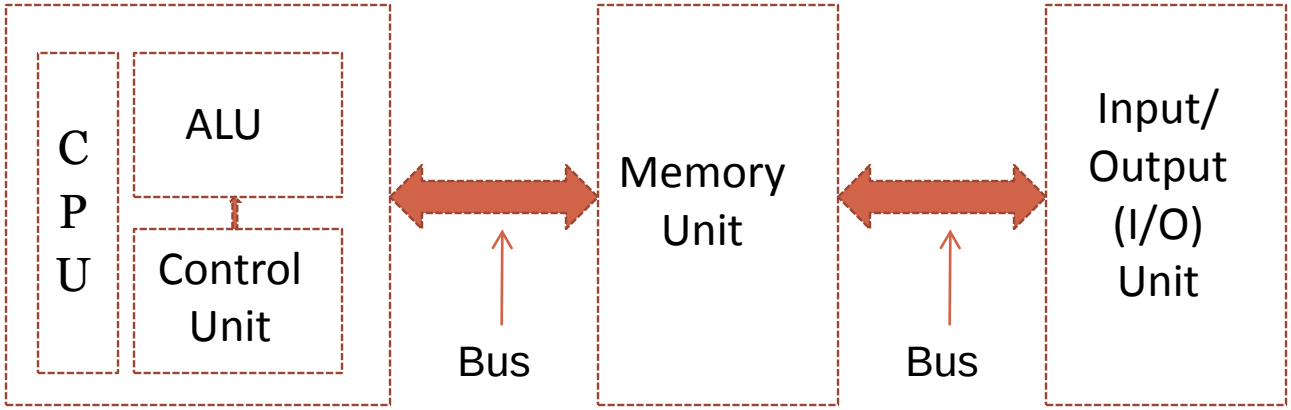
Course Contents

1) Microprocessor Structure and Operations
2) Classification of Microprocessors
3) Intel 8085 Microprocessor
4) Instruction Set of 8085 and Programming
5) Memory Organization
6) Memory Interfacing Techniques
7) Interrupts
8) Input-Output Devices
9) Interfacing Input-Output Devices
10) Serial Transmission
11) Operating Systems for Microprocessor

Text Books: (i) Lecture Notes

(ii) Ramesh Goankar: Microprocessor Architecture,

Programming and Applications with the 8085, Printice Hall.



Basic Functional Units of a Computer Systems

1. I/O Unit

   Allows the computer system to interact with the outside world by moving data into and out of the system.

   - Input Unit: Accepts coded information as data.
   - Output Unit: Sends processed result to the outside world.

2. Memory Unit: Stores programmed data.

Memory	
Main Memory	**Secondary Memory**
(i) Fast (ii) Expensive (iii) Low capacity (iv) Connects directly to the CPU (v) Program must reside during execution Example: RAM	(i) Slower (ii) Cheaper (iii) Large capacity (iv) Not connected directly to the CPU Examples: HDD, Floppy Disk, CD-ROMs, etc.

3. Central Processing Unit (CPU)

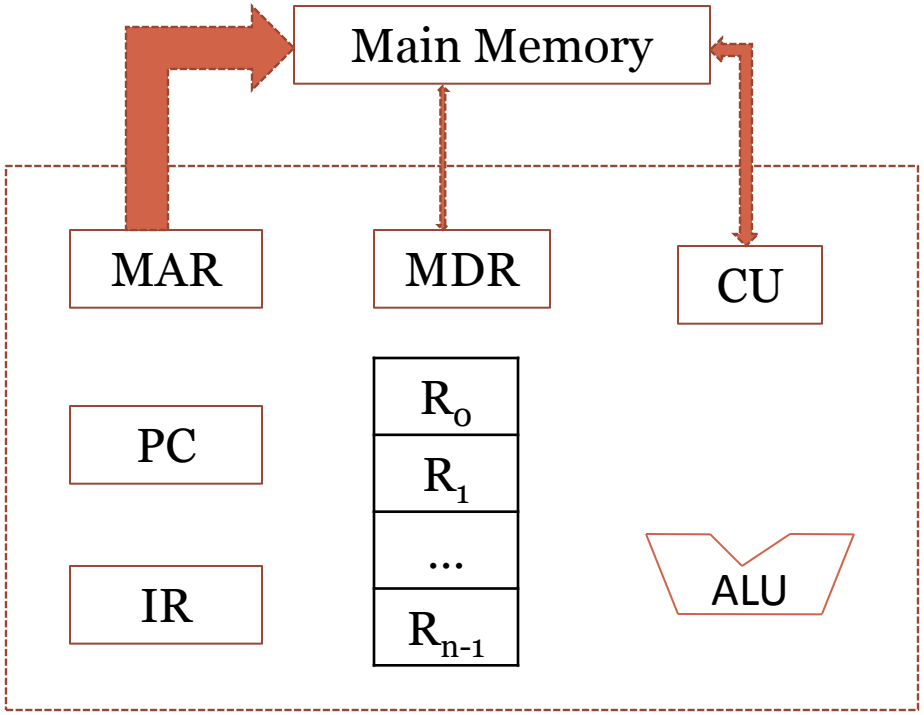
(a) Arithmetic and Logical Unit (ALU)

(i) Dedicated unit to perform arithmetic and logical operations.

(ii) Processor holds a number of high speed registers to hold temporary data.

(b) Control Unit (CU)

(i) Controls the operations all units by sending appropriate signals to them.



Interconnection between memory and CPU

Memory Address Register (MAR): Holds the address of the memory location to or from which data to be transferred.

Program counter (PC): Keeps the address of the instruction currently being executed.

Memory Data Register (MDR): Contains the data to be written into or read out of the addressed memory location.

Instruction Register (IR): Holds the instruction that is currently being executed.

How does the Microprocessor work?

1. The Instructions are stored sequentially in the memory.
2. The Microprocessor *fetches* one instruction at a time, *decodes* and *executes*.
3. The sequence of fetch, decode and execute is continued until the Microprocessor comes across an instruction to *stop*.
4. During the entire process, microprocessor uses the system bus to fetch instruction and data from the memory.
5. It uses registers within CPU to hold data temporarily and performs arithmetic and logical functions using ALU.

Classification of Microprocessors

MPU classification based on word length:

Word: It is defined as number of bits the microprocessor recognizes and process at a time. It ranges from 4-bit to 64-bit.

4-bit MPU:	Intel 4004 (1971)
8-bit MPU:	Intel 8008, 8080, 8085, Motorola 6800Zilog Z80 (1972-1976)
32-bit MPU:	Intel 8086, 8088, 80486 (1979-1989)
64-bit MPU:	Intel Pentium (1993), Pentium II (1997) – Pentium 4 (2000) Core 2 Duo

Microprocessor

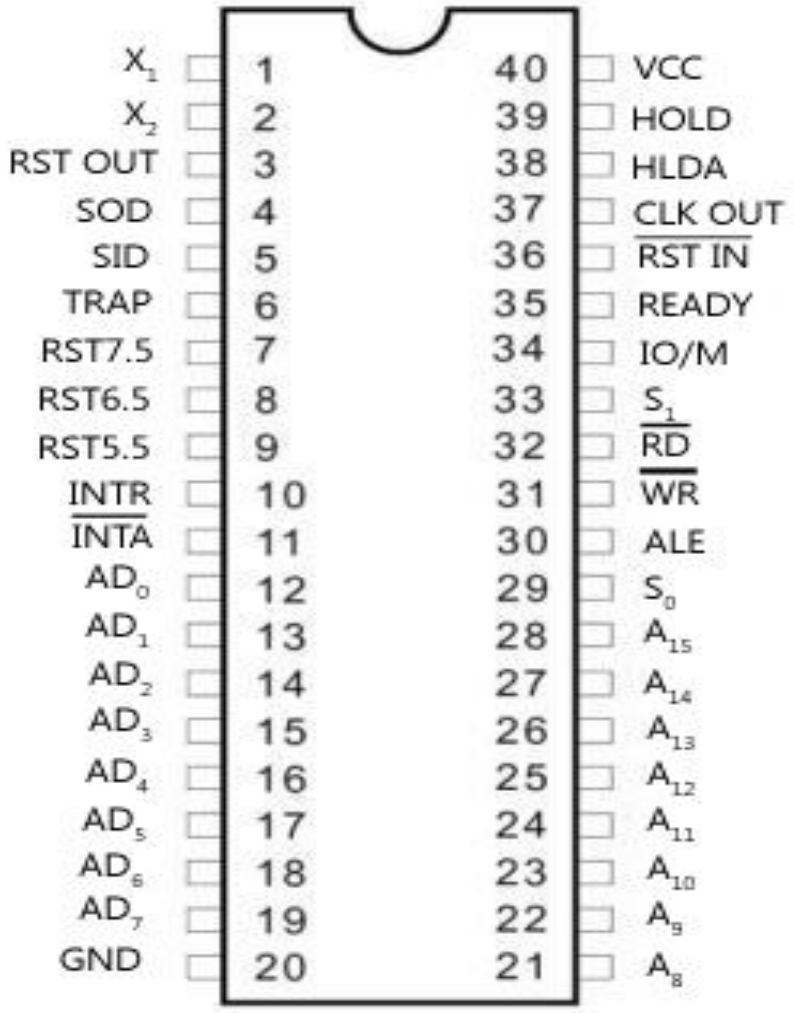
Definition:

- A programmable device to control processes or devices.
- A processing unit of a computer.
- A CPU of a microprocessor-based system.

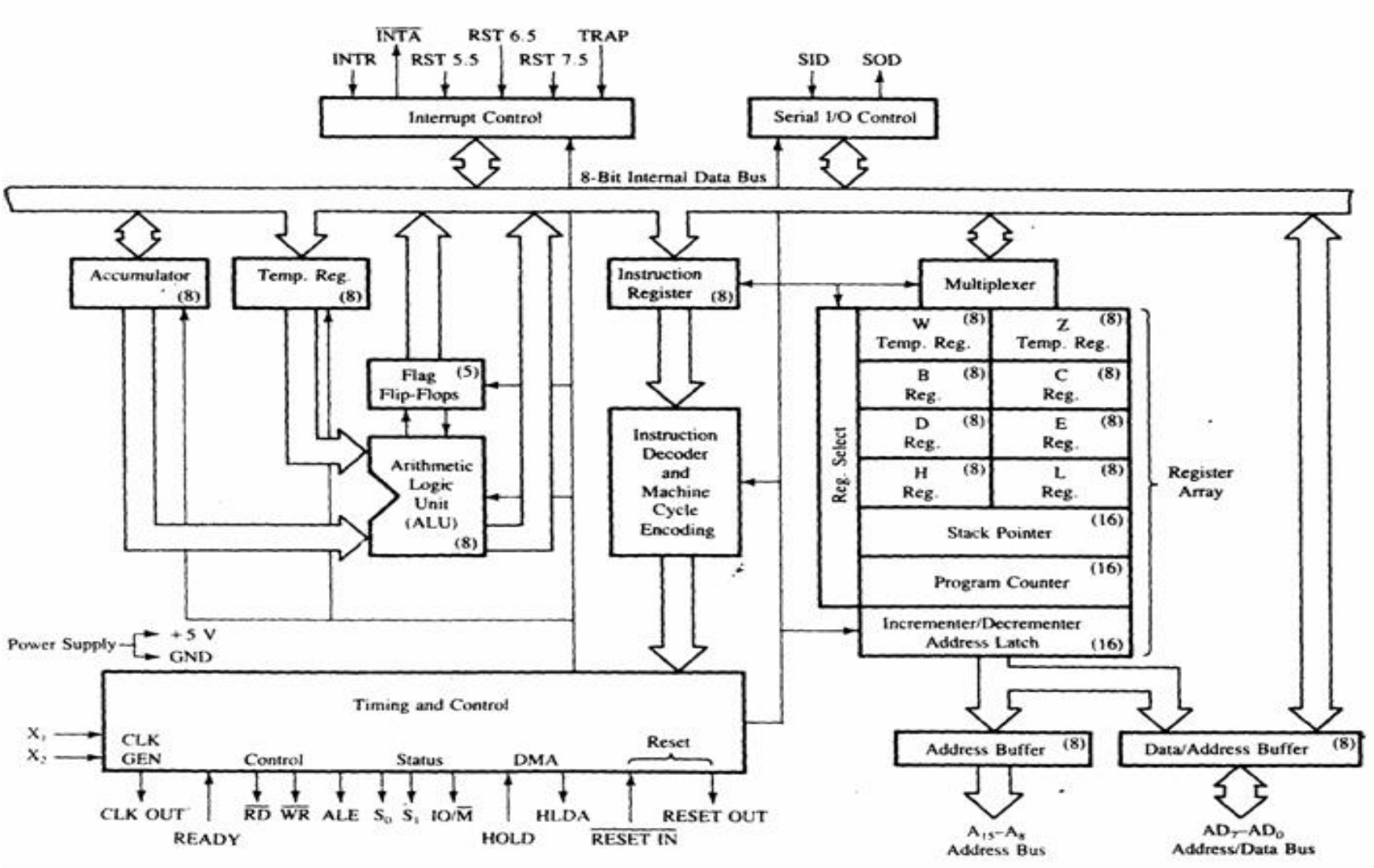
Microprocessor communicates and operates on bits – 0 and 1

4-bit string:	Nibble
8-bit string:	Byte
One or more byte:	Word

Pin Diagram of 8085 Microprocessor



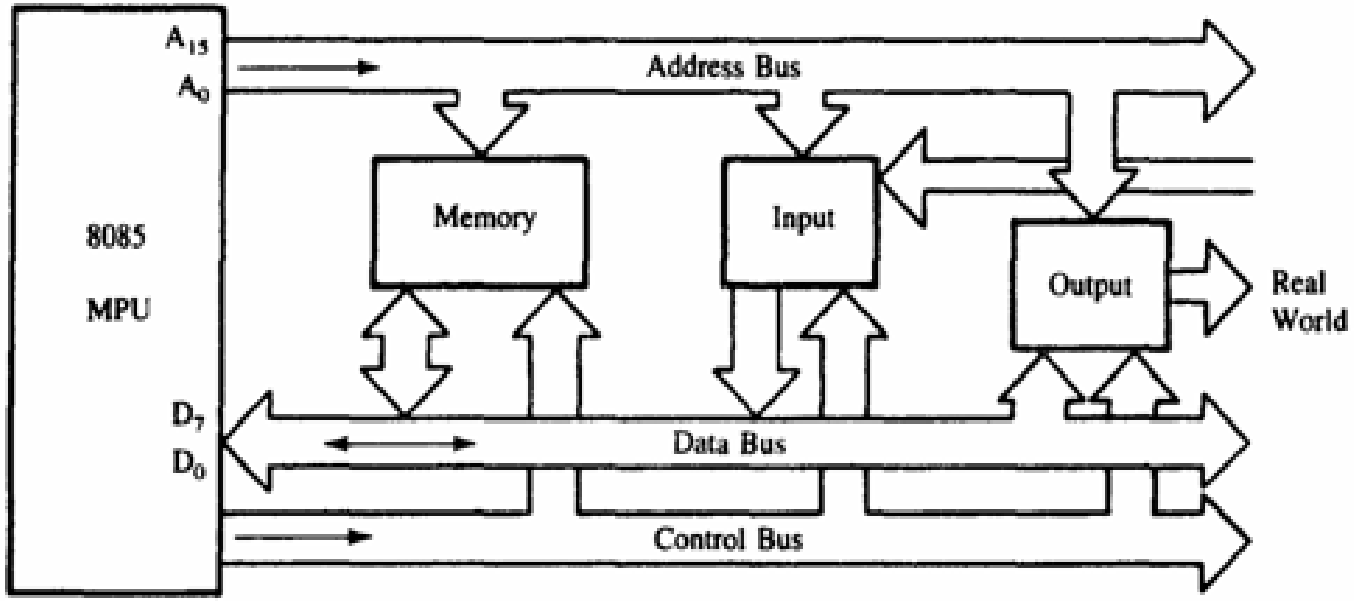
The functional block diagram or architecture of 8085 Microprocessor



8085 Bus Structure

Address Bus:

- The address bus is a group of 16 lines generally identified as A0 to A15.
- The address bus is unidirectional: bits flow in one direction-from the MPU to peripheral devices.
- The MPU uses the address bus to perform the first function: identifying a peripheral or a memory location.



Data Bus:

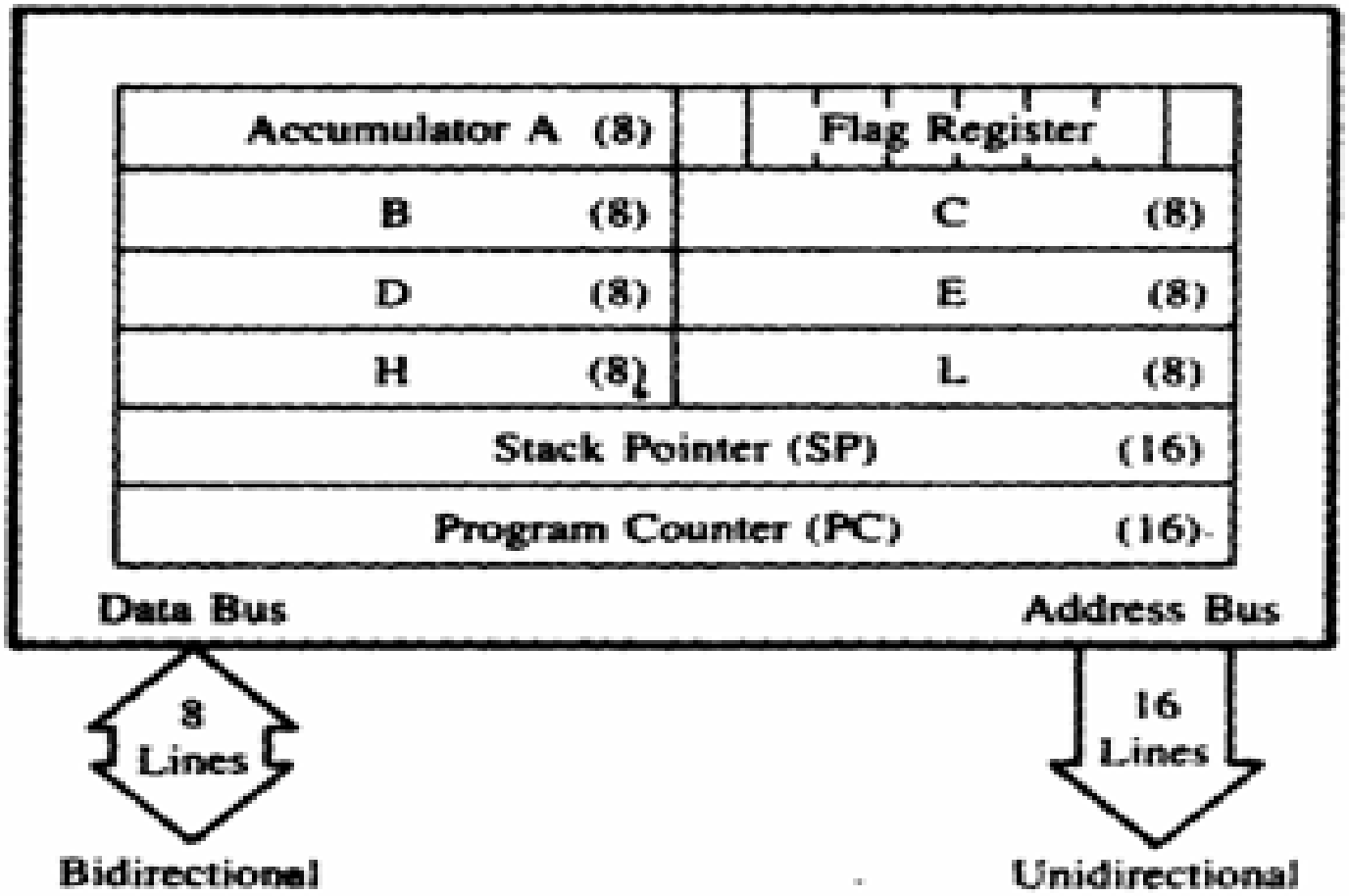
- The data bus is a group of eight lines used for data flow.
- These lines are bi-directional - data flow in both directions between the MPU and memory and peripheral devices.
- The MPU uses the data bus to perform the second function: transferring binary information.
- The eight data lines enable the MPU to manipulate 8-bit data ranging from 00 to FF (28 = 256 numbers).
- The largest number that can appear on the data bus is 11111111.

Control Bus:

- The control bus carries synchronization signals and providing timing signals.
- The MPU generates specific control signals for every operation it performs. These signals are used to identify a device type with which the MPU wants to communicate.

Registers of 8085:

- The 8085 have six general-purpose registers to store 8-bit data during program execution.
- These registers are identified as B, C, D, E, H, and L.
- They can be combined as register pairs-BC, DE, and HL-to perform some 16-bit operations.


Accumulator A (8)
Flag Register
B (8)
C (8)
D (8)
E (8)
H (8)
L (8)
Stack Pointer (SP) (16)
Program Counter (PC) (16)
Data Bus
Address Bus
8 Lines
16 Lines
Bidirectional
Unidirectional

Accumulator (A):

- The accumulator is an 8-bit register that is part of the arithmetic/logic unit (ALU).
- This register is used to store 8-bit data and to perform arithmetic and logical operations.
- The result of an operation is stored in the accumulator.

Flags:

- The ALU includes five flip-flops that are set or reset according to the result of an operation.
- The microprocessor uses the flags for testing the data conditions.
- They are Zero (Z), Carry (CY), Sign (S), Parity (P), and Auxiliary Carry (AC) flags. The most commonly used flags are Sign, Zero, and Carry.

The bit position for the flags in flag register is as follows:

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

1. **Sign Flag (S):**
   - After execution of any arithmetic and logical operation, if D7 of the result is 1, the sign flag is set. Otherwise it is reset.
   - D7 is reserved for indicating the sign; the remaining is the magnitude of number.
   - If D7 is 1, the number will be viewed as negative number. If D7 is 0, the number will be viewed as positive number.

2. **Zero Flag (z):**
   - If the result of arithmetic and logical operation is zero, then zero flag is set otherwise it is reset.

3. Auxiliary Carry Flag (AC):

- If D3 generates any carry when doing any arithmetic and logical operation, this flag is set. Otherwise it is reset.

4. Parity Flag (P):

- If the result of arithmetic and logical operation contains even number of 1's then this flag will be set and if it is odd number of 1's it will be reset.

5. Carry Flag (CY):

- If any arithmetic and logical operation result any carry then carry flag is set otherwise it is reset.

Arithmetic and Logic Unit (ALU):

- It is used to perform the arithmetic operations like addition, subtraction, multiplication, division, increment and decrement and logical operations like AND, OR and EX-OR.
- It receives the data from accumulator and registers.
- According to the result it set or reset the flags.

Program Counter (PC):

- This 16-bit register sequencing the execution of instructions.
- It is a memory pointer. Memory locations have 16-bit addresses, and that is why this is a 16-bit register.
- The function of the program counter is to point to the memory address of the next instruction to be executed.
- When an opcode is being fetched, the program counter is incremented by one to point to the next memory location.

Stack Pointer (SP):

- The stack pointer is also a 16-bit register used as a memory pointer.
- It points to a memory location in R/W memory, called the stack.
- The beginning of the stack is defined by loading a 16-bit address in the stack pointer (register).

Temporary Register: It is used to hold the data during the arithmetic and logical operations.

Instruction Register: When an instruction is fetched from the memory, it is loaded in the instruction register.

Instruction Decoder: It gets the instruction from the instruction register and decodes the instruction. It identifies the instruction to be performed.

Serial I/O Control: It has two control signals named SID and SOD for serial data transmission.

Timing and Control unit:

- It has three control signals ALE, RD (Active low) and WR (Active low) and three status signals IO/M(Active low), S0 and S1.

- ALE is used for provide control signal to synchronize the components of microprocessor and timing for instruction to perform the operation.

- RD (Active low) and WR (Active low) are used to indicate whether the operation is reading the data from memory or writing the data into memory respectively.

IO/M (Active low) is used to indicate whether the operation is belongs to the memory or peripherals.

If,

IO/M (Active Low)	S1	S2	Data Bus Status(Output)
0	0	0	Halt
0	0	1	Memory WRITE
0	1	0	Memory READ
1	0	1	IO WRITE
1	1	0	IO READ
0	1	1	Opcode fetch
1	1	1	Interrupt acknowledge

Interrupt Control Unit:

It receives hardware interrupt signals and sends an acknowledgement for receiving the interrupt signal.

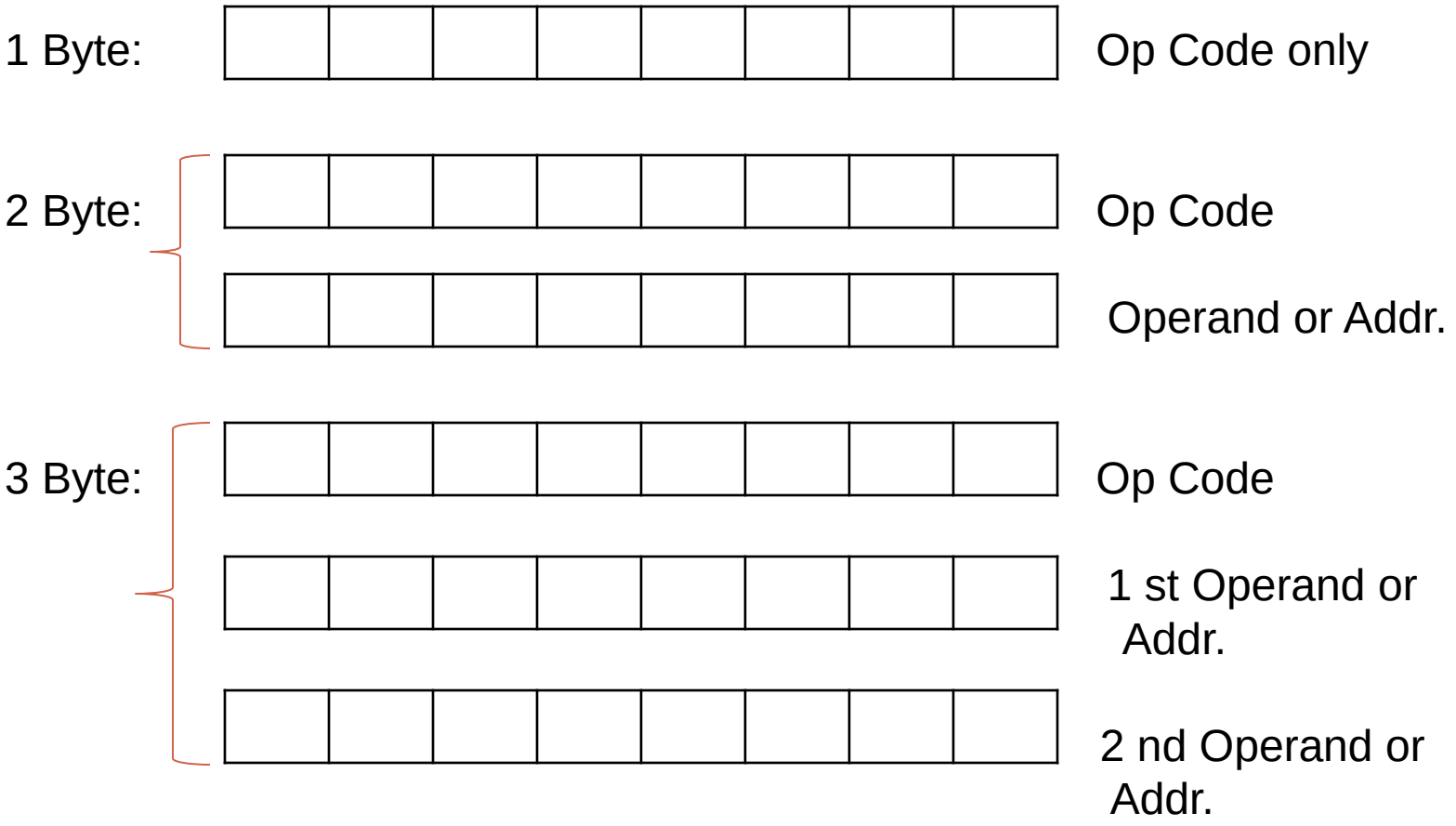
Instruction Set

Format:

Mnemonics (Op Code)	Operands

- Types of Instructions:
  - Based on:
    - I. Operation: Add, Subtract, etc.
    - II. Size: 1 Byte, 2 byte and 3 byte

Instructions based on sizes



Storing of Instruction in Main Memory

Address	Content	Instn. Type
0000_H	...	
0001_H	...	
$000i_H$	Op Code	1 byte instruction
	...	
$000j_H$	Op Code	2 byte instruction
$000j+1_H$	Operand	
	...	
$000k_H$	Op Code	3 byte instruction
$000k+1_H$	Operand $_L$	
$000k+2_H$	Operand $_H$	
	...	

- Addressing Modes: It refers to the way in which operands are specified in an instruction. In other words, it refers to the way by which processor accesses the operand.

1. **Immediate Addressing**
2. **Direct Addressing**
3. **Register Addressing**
4. **Register Indirect Addressing**
5. **Implied Addressing**

1. Immediate Addressing:

In immediate addressing mode, the data is specified in the instruction itself. The data will be a part of the program instruction.

Examples: MVI Reg, $Data_8$, Reg = {A, B, C, D, E, H, L}

MVI B, $3E_H$; B = $3E_H$;

LXI RegPair $Data_{16}$, RegPair = {B, D, H, SP}

LXI SP, 2700_H; SP = 2700_H, SPL = 00_H, SPH = 27_H

2. Direct Addressing:

In direct addressing mode, the address of the data is specified in the instruction. The data will be in memory. In this addressing mode, the program instructions and data can be stored in different memory.

Examples: LDA $Address_{16}$; STA $Address_{16}$; LDA 1050_H ; A = M[1050_H];

LHLD 3000_H ; L = M[3000_H], H = M[3001_H]

3. Register Addressing:

In register addressing mode, the instruction specifies the name of the register in which the data is available.

Examples: MOV RegD, RegS; RegD, RegS = {A, B, C, D, E, H, L}

MOV A, B ; A = B

ADD C; A = A +C

4. Register Indirect Addressing:

The instruction specifies the name of the register in which the address of the data is available. Here the data will be in memory and the address will be in the register pair.

Examples: MOV A, M ; A = M[HL]

LDAX B; A = M[BC]

5. Implied Addressing:

The instruction itself specifies the data to be operated.

Examples: CMA - Complement the content of accumulator;

RAL – Rotate Acc. left by 1 bit

The 8085 instruction set can be classified into the following five functional headings.

1. DATA TRANSFER INSTRUCTIONS:

It includes the instructions that move (copies) data between registers or between memory locations and registers. In all data transfer operations the content of source register is not altered. Hence the data transfer is copying operation.

Ex: (1) MOV A, B ; A = B

(2) MVI C, 45_H; C = 45_H

(3) LXI H, 2005_H; H = 20_H, L = 05_H

(4) MOV A, M; A = M[HL] (5) LDA 2050_H; A = M[2050_H]

(6) STA 2010_H; M[2010_H] = A

2. ARITHMETIC INSTRUCTIONS:

Includes the instructions, which performs the addition, subtraction, increment or decrement operations. The flag conditions are altered after execution of an instruction in this group.

Ex:

(1) ADD B ; A = A+B

(2) ADC B; A = A + B + CY

(3) ADD M; A = A + M[HL]

(4) ADC M; A = A + M[HL] +CY

(5) ADI 05_H; A = A + 05_H

(6) SUB B; A = A – B

(7) SUB M; A = A – M[HL]

(8) SUI 05_H; A = A – 05_H

(9) INR B; B = B + 1 (10) INX B; BC = BC + 1

(11) DCR B; B = B – 1; (12) DCX B; BC = BC - 1

3. LOGICAL INSTRUCTIONS:

The instructions which performs the logical operations like AND, OR, EXCLUSIVE- OR, complement, compare and rotate instructions are grouped under this heading. The flag conditions are altered after execution of an instruction in this group.

Ex: (1) ORA A
(2) ANI B, 01_H

4. BRANCHING INSTRUCTIONS:

The instructions that are used to transfer the program control from one memory location to another memory location are grouped under this heading.

Ex: (1) CALL 2050_H (Subroutine call)
(2) JMP 4100_H (Unconditional jump)
(3) JNZ 2040_H (Conditional jump)

5. MACHINE CONTROL INSTRUCTIONS:

It includes the instructions related to interrupts and the instruction used to stop the program execution.

Ex: (1) NOP
(2) HLT
(3) RST 1

6. STACK INSTRUCTIONS:

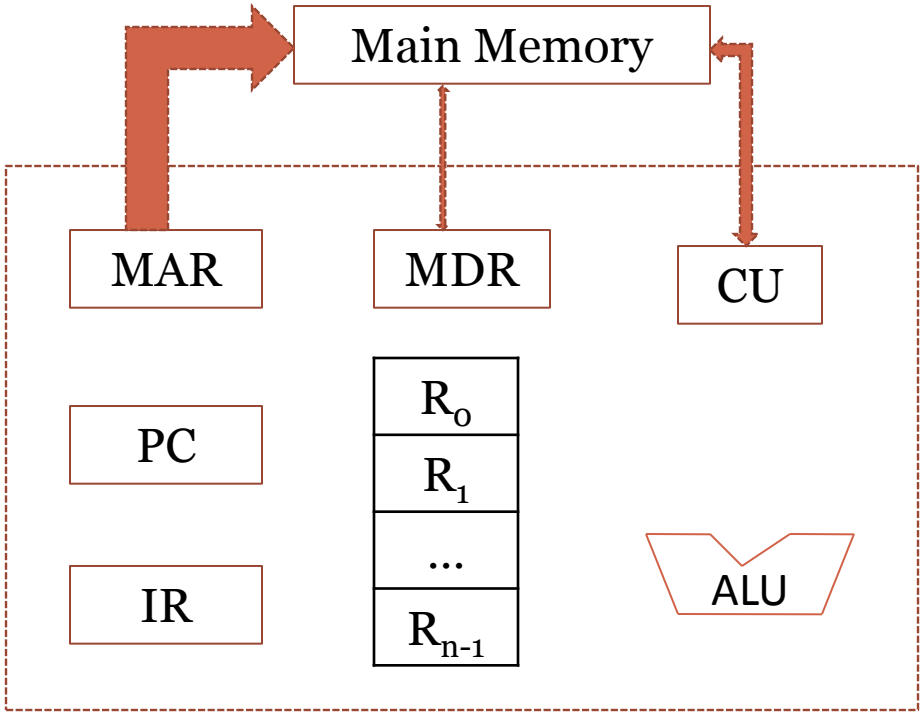
Used for stack operations.

Ex. (1) PUSH B; M[SP--] = B, M[SP--] = C
(2) POP B; C = M[++SP], B = M[++SP]

7. I/O INSTRUCTIONS:

Used for i/o operations.

Ex. (1) IN 45_H
(2) OUT 55_H


Main Memory
MAR
MDR
CU
PC
R_o
R_1
...
R_{n-1}
IR
ALU

Instruction Execution

Execution of an instruction consists of two separate cycles: (i) Fetch cycle and (ii) Execution cycle.

(i) Fetch cycle:

a) Instruction addressed by the content of program counter (PC) is loaded in instruction register (IR) from memory. MAR = [PC], MDR = M[MAR], IR = [MDR];

b) PC is incremented. PC = PC + 1;

c) Fetching of operand data initiated.

(ii) Execution cycle:

a) Instruction in IR is executed and specified operation is performed by CPU.

Programming in 8085

Example 1: Add two numbers, 05 and 03

```
MVI   A,  05H      ; A = 05
MVI   B,  03H      ; B = 03
ADD   B            ; A = A + B
STA       2050H    ; M[2050] = A
RST 5
```

m/m loc.	content
2500	3E
2501	05
2502	06
2503	03
2504	80
2505	32
2506	50
2507	20
2508	EF

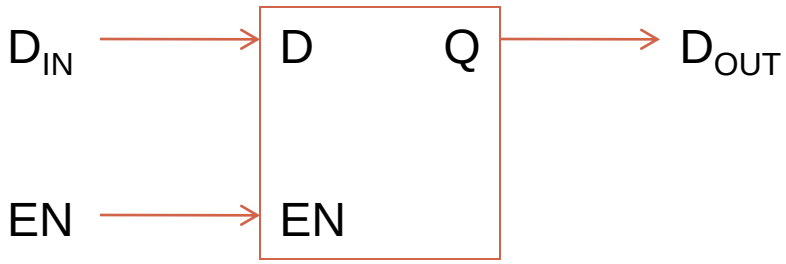
Memory locations and Addresses

Address	Data/Instruction					Word
0						0
1						1
…			…			…
i	A_{n-1}	A_{n-2}	…	A_1	A_0	i
…			…			…
2^k-1						2^k-1

Memory

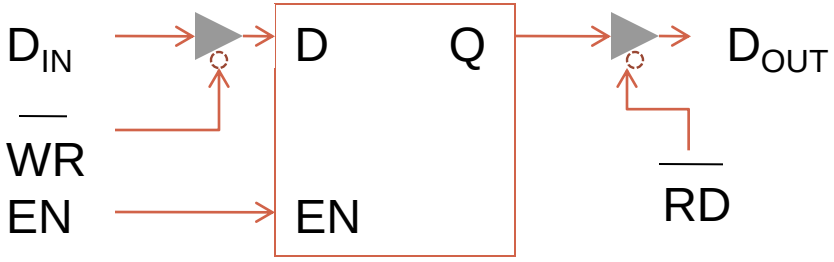
1. Read/Write (R/W) Memory
   - Made of Registers
     - Registers made of Flip-Flops (F/Fs)

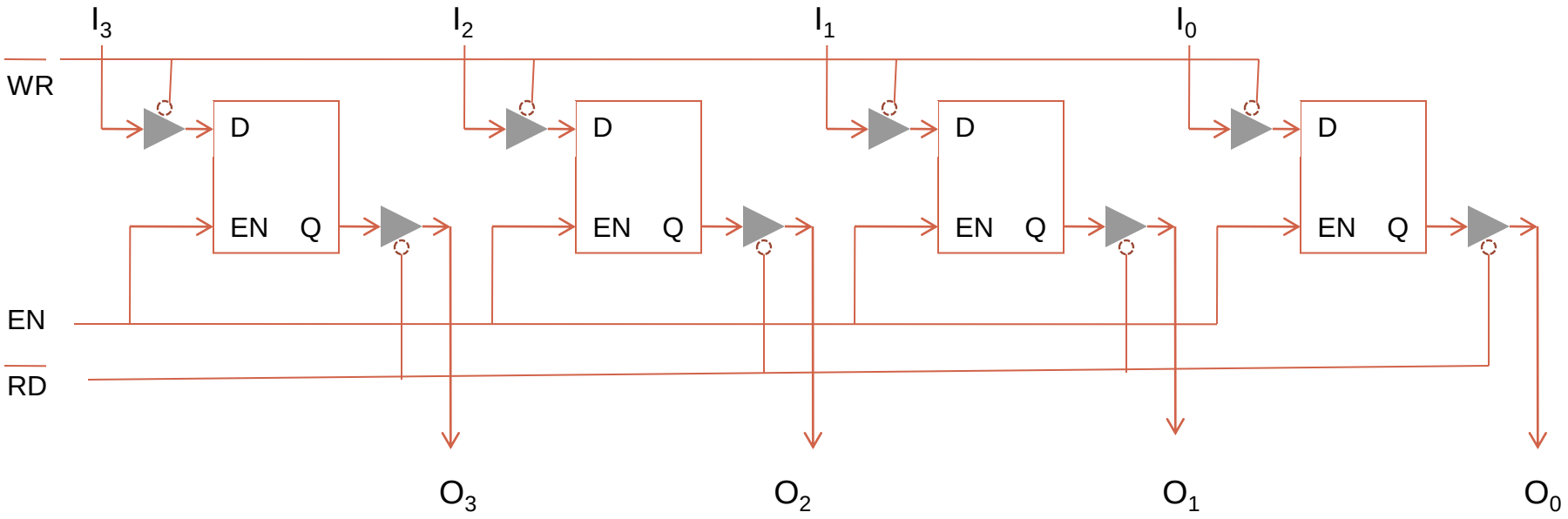
1. Read Only Memory (ROM)
   - Made of Registers
     - Registers made of diodes

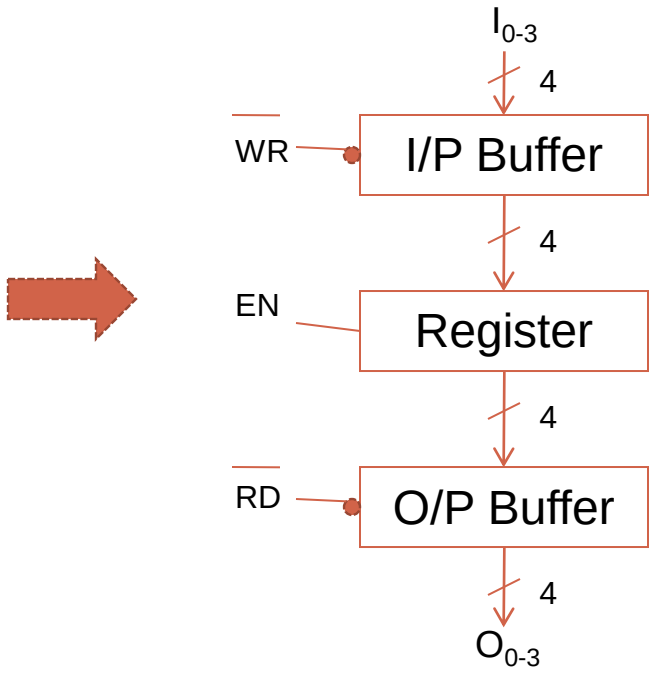


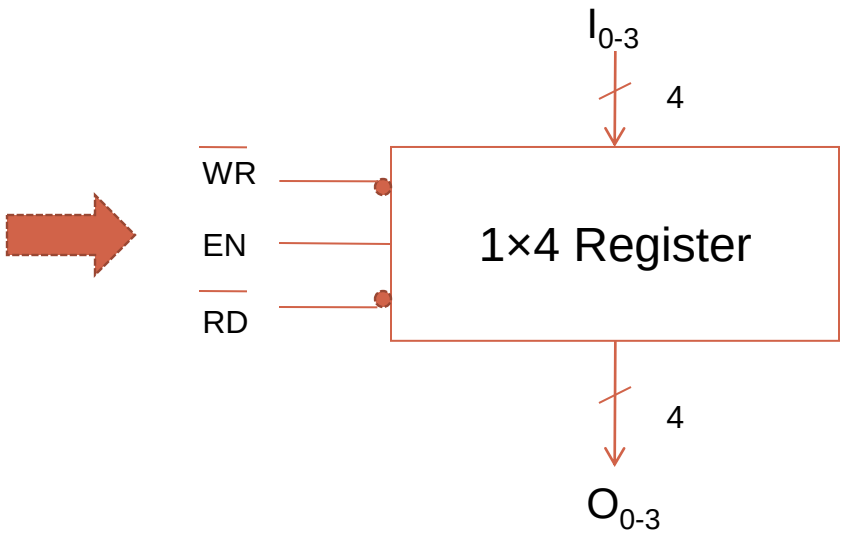
F/F or Latch

- To prevent unintentional change of input (i/p) and control of read operation, the F/F may be modified using a tri-state buffer.




I3
I2
I1
I0
WR
D
EN Q
D
EN Q
D
EN Q
D
EN Q
EN
RD
O3
O2
O1
O0


I_{0-3}
4
WR
I/P Buffer
4
EN
Register
4
RD
O/P Buffer
4
O_{0-3}


I_{0-3}
4
WR
EN
RD
1×4 Register
4
O_{0-3}

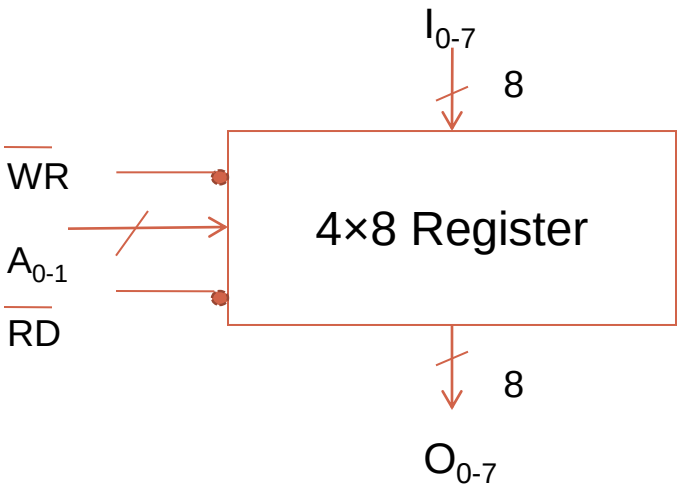
Memory Organization

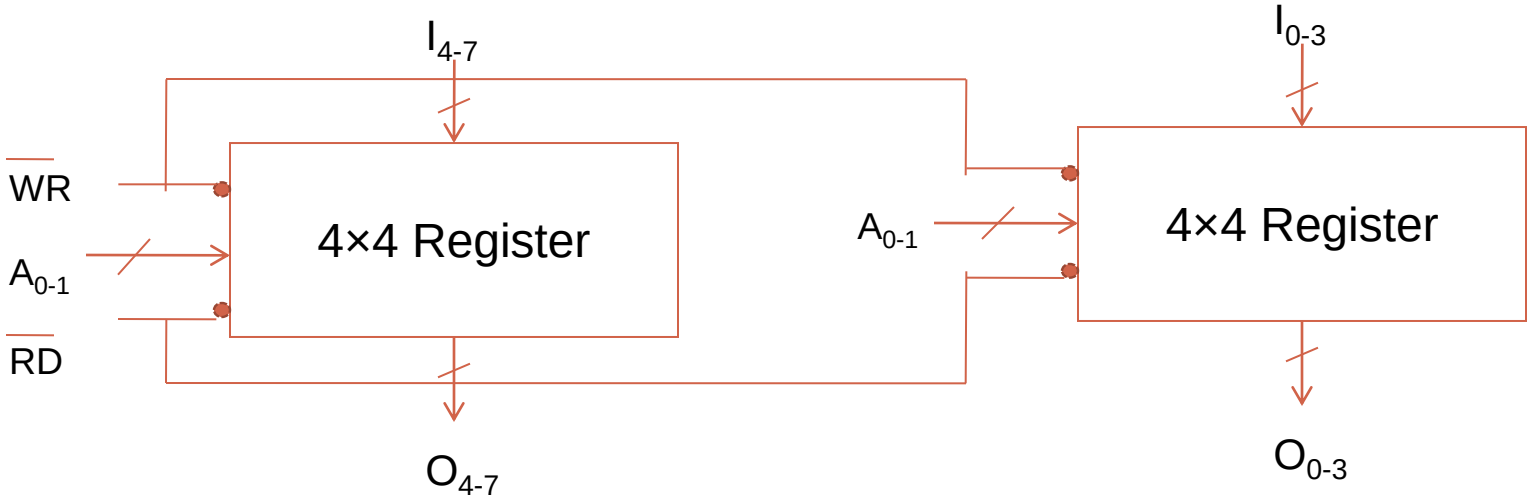
In a memory (m/m) chip all the registers are arranged in a sequence and are identified by a binary numbers called the m/m addresses.

Communication with memory

1) Select the memory chip.
2) Identify the required register.
3) Perform Read or Write operation.

Memory Organization



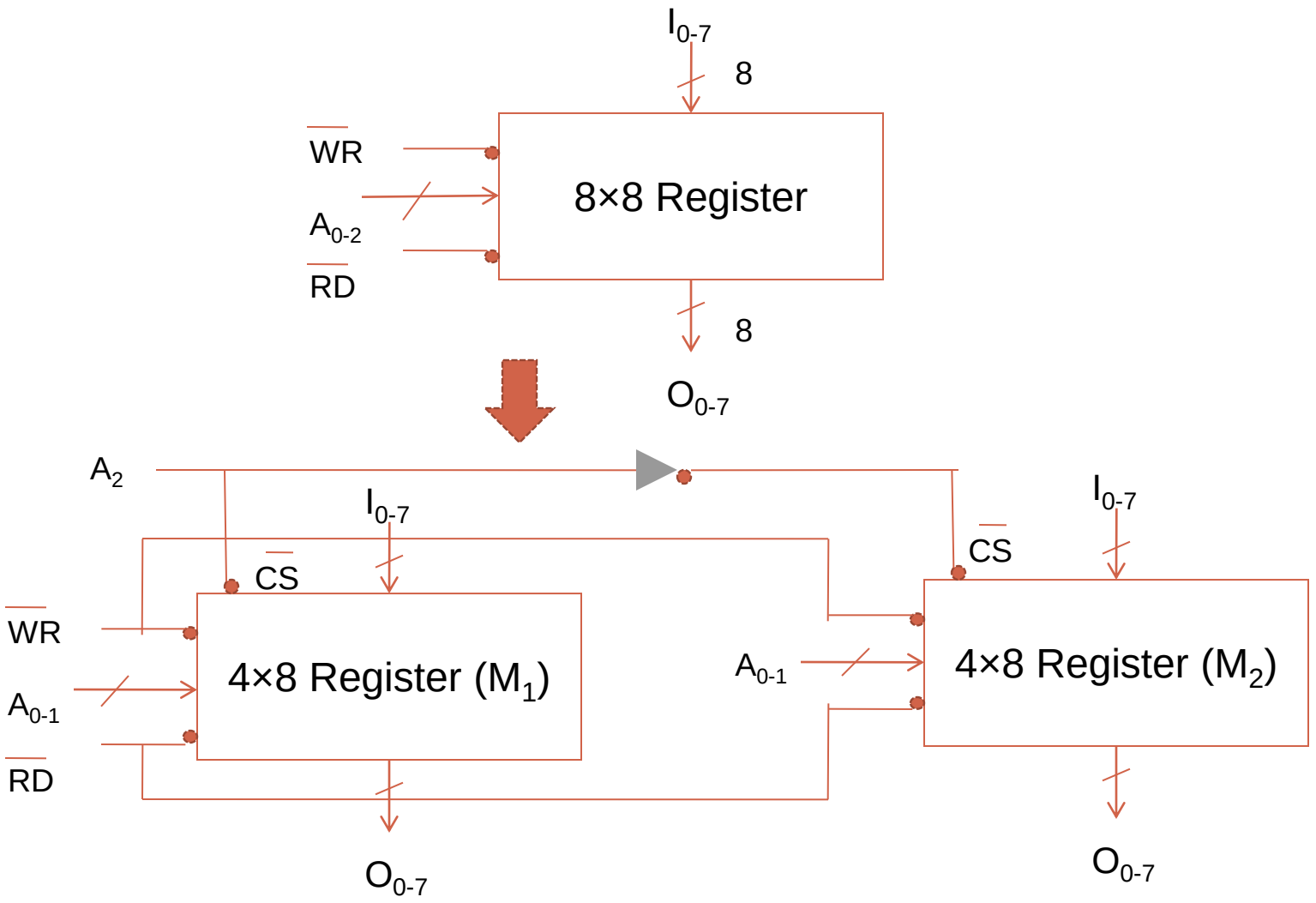


Made up with 2 chips with two 4 x 4 registers

Task:

Express an 8 x 16 Register using 8 x 4 Registers.

Memory Organization



Made up with 2 m/m chips of 4 x 8 registers. A_2 acts as chip select.

Task:

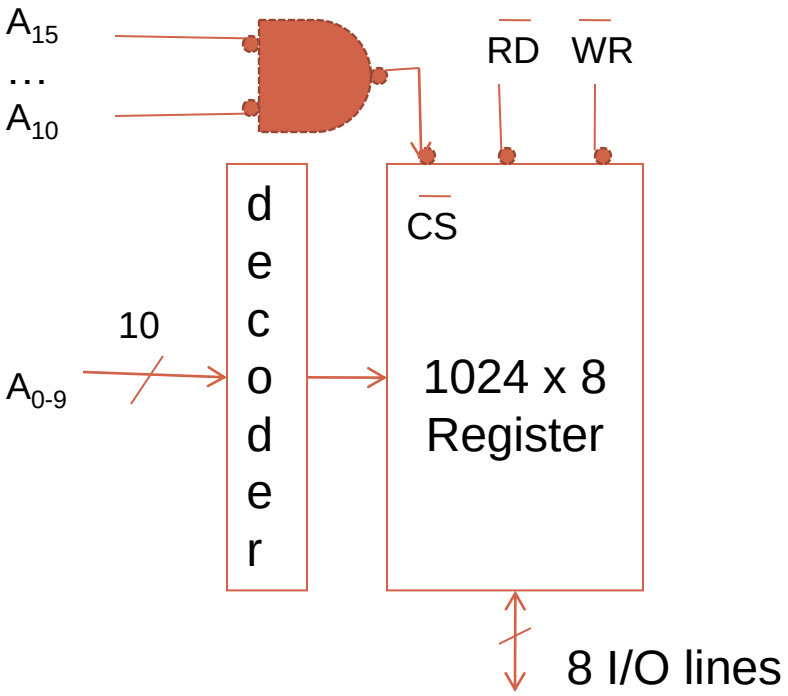
Express a 16 x 16 Register using 4 x 16 Registers.

Memory Map and Addresses

How is memory mapped and are addresses generated?

1) Select the number of address lines required to identify one register of the memory module. Assign the address lines starting from $A_{0\text{-}i}$.
2) Use the remaining address lines $A_{(i+1)\text{-}15}$ to generate a unique chip select (CS) signal.
3) Starting address is: $A_{15}\, A_{14} \ldots A_{i+1}\, 0\, 0 \ldots 0$
4) Ending address is: $A_{15}\, A_{14} \ldots A_{i+1}\, 1\, 1 \ldots 1$

Example: Memory address range of 1K (1024 x 8) memory



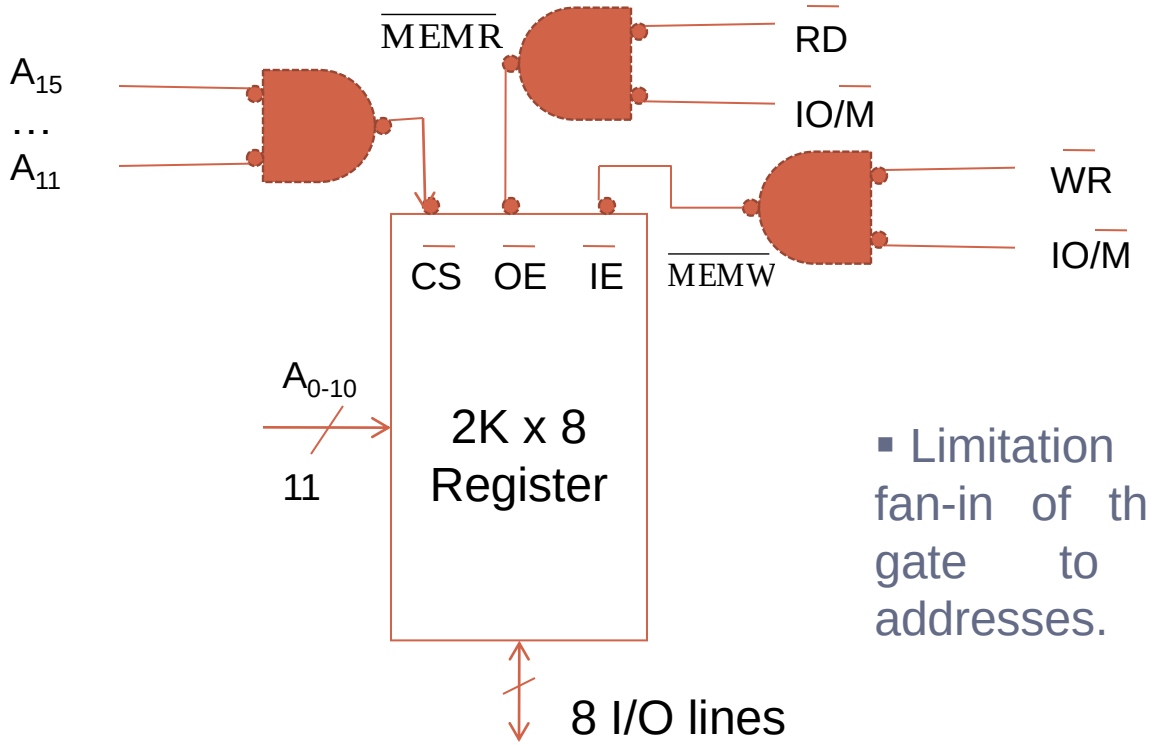
- By changing the combination of $A_{10\text{-}15}$, different memory address range can be formed.

A_{15}	A_{14}	A_{13}	A_{12}	A_{11}	A_{10}	A_9	A_8	A_7	A_6	A_5	A_4	A_3	A_2	A_1	A_0	Address range
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	= $0\,0\,0\,0_H$
0	0	0	0	0	0	1	1	1	1	1	1	1	1	1	1	= $0\,3\,F\,F_H$

Memory Interfacing

1) Connect the required address lines of the address bus to the address lines of the memory chip.
2) Decode the remaining address lines of the address bus to generate a unique chip select signal.
3) Generate control signals $\overline{\text{MEMR}}$ and/or $\overline{\text{MEMW}}$ by combining $\overline{\text{RD}}$ and $\overline{\text{WR}}$ signals with $\text{IO}/\overline{\text{M}}$ signal, respectively. Use them to enable appropriate input and/or output buffer.

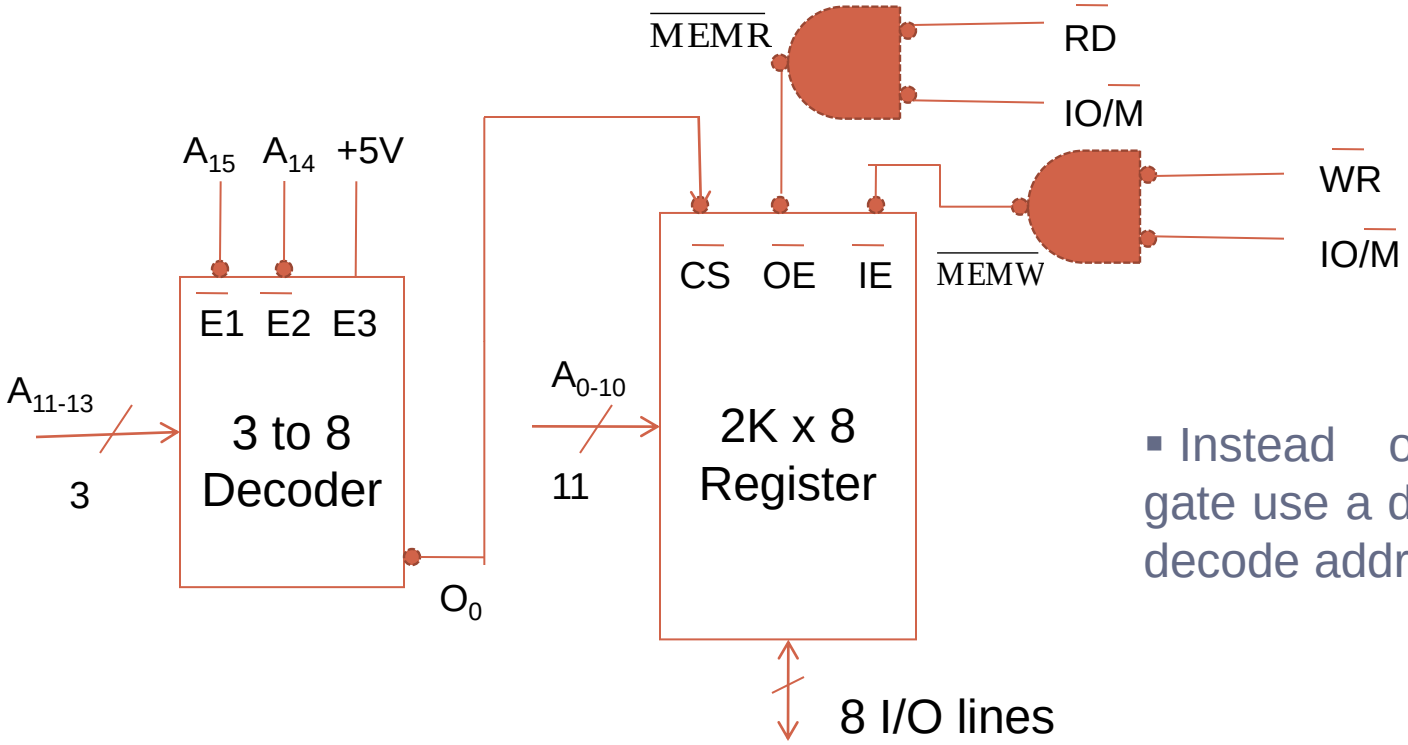
Example: Interfacing 2K Read/Write memory



- Limitation due to fan-in of the NAND gate to decode addresses.

A_{15}	A_{14}	A_{13}	A_{12}	A_{11}	A_{10}	A_9	A_8	A_7	A_6	A_5	A_4	A_3	A_2	A_1	A_0	Address range
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	= $0\,0\,0\,0_H$
0	0	0	0	0	1	1	1	1	1	1	1	1	1	1	1	= $0\,7\,F\,F_H$

Example: Interfacing 2K Read/Write memory (Contd.)



- Instead of NAND gate use a decoder to decode addresses.

A_{15}	A_{14}	A_{13}	A_{12}	A_{11}	A_{10}	A_9	A_8	A_7	A_6	A_5	A_4	A_3	A_2	A_1	A_0	Address range
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	= $0\,0\,0\,0_H$
0	0	0	0	0	1	1	1	1	1	1	1	1	1	1	1	= $0\,7\,F\,F_H$

Address Decoding

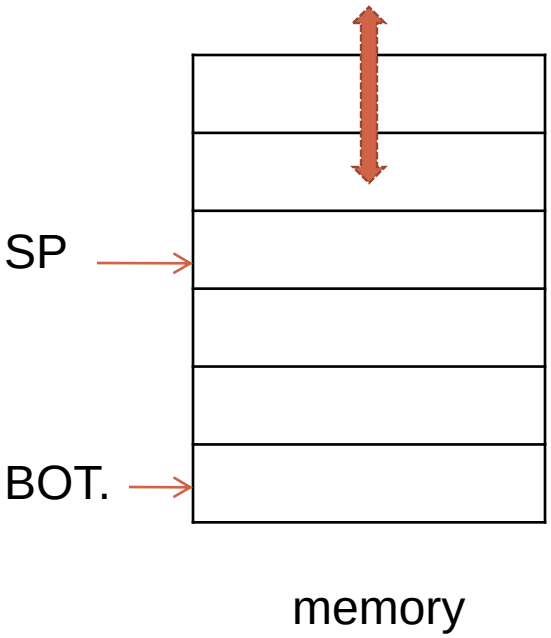
1) Complete or Absolute Decoding: All the available address lines are used for address decoding.
2) Partial Decoding: Not all the available address lines are used for address decoding. Others are kept as *don't* care condition.

Foldback or Mirror Memory: For each combination of don't care address lines, we have separate address range. Keeping one combination as primary address range, remaining address ranges are called the foldback or mirror memory of the primary address range.

Organization of data

1) Stack (LIFO - Last In First Out)

a) Addition and deletion from top of the stack by push and pop, respectively.
b) Only one pointer (SP) is required. SP points to the empty location.
c) Bottom end is fixed.



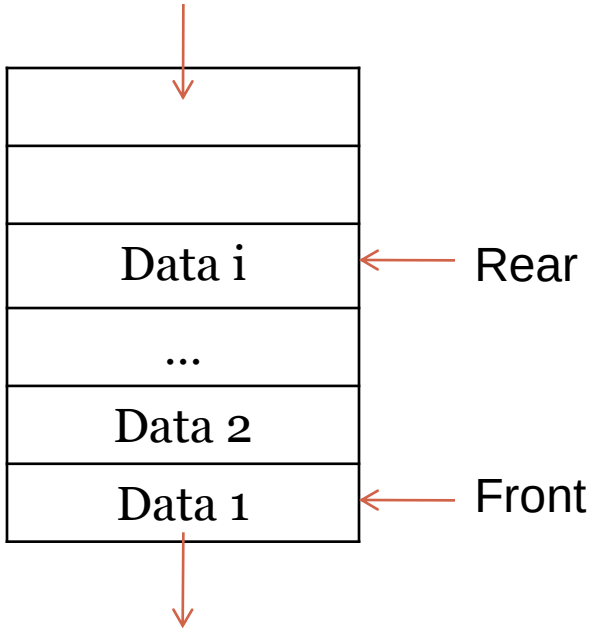
Example:

PUSH B ; M[SP--] = B, M[SP--] = C

POP B; C = M[++SP], B = M[++SP]

2) Queue (FIFO - First In First Out)

a) New data are added at the rear end and retrieved from the front end.
b) Two pointers are required to track front and rear ends.
c) Two ends are not fixed.



Subroutines: Used to perform a particular task many times on different data.

- Saves memory spaces.

Subroutine Call:

1) Store the content of PC in the link register.
2) Branch to the target address specified in the instruction.

Return from Subroutine:

1) PC is loaded with the content of link register.
2) Branch to the specified address.

Example: CALL SUB_NAME
Next Instn.

SUB_NAME: Instns
RET

Functions performed by the Microprocessor

1) Microprocessor initiated.
2) Internal operations.
3) External initiated (Peripheral operations).

1) Microprocessor initiated:

a) Memory Read.
b) Memory Write.
c) I/O Read.
d) I/O Write.

- To perform above operations microprocessor does following steps-
  1) Identify the peripheral / memory
  2) Provide synchronization signals
  3) Transfer data

MPU performs Read Operation when:

1) Fetches Op Code of an instruction from m/m.
2) Fetches the operand address portion of an instruction from m/m.
3) Fetches data from m/m or i/o device to execute an instruction.

MPU performs Write Operation when:

1) Transfers data from internal MPU register to m/m.
2) Transfers data from internal MPU register to an i/o device.

Timing Diagram is a graphical representation. It represents the execution time taken by each instruction in a graphical format. The execution time is represented in T-states.

Instruction Cycle:

The time required to execute an instruction is called instruction cycle.

Machine Cycle:

The time required to access the memory or input/output devices is called machine cycle.

T-State (Clock Cycle):

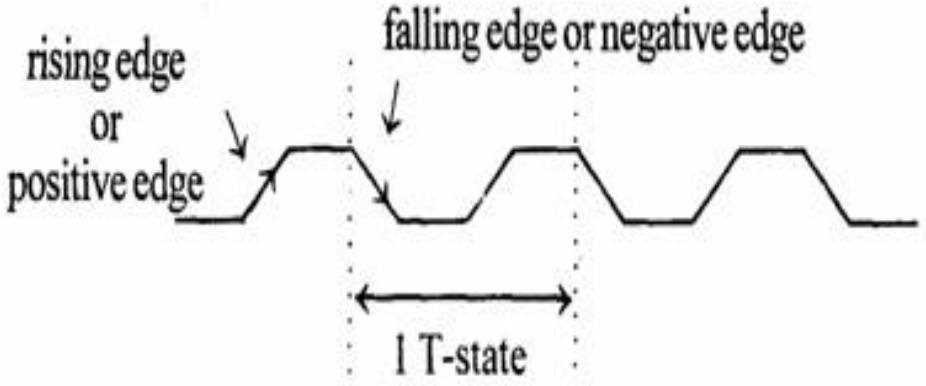
The machine cycle and instruction cycle takes multiple clock cycles (periods). A portion of an operation carried out in one system clock period is called as T-state.

MACHINE CYCLES OF 8085:

The 8085 microprocessor has 5 (five) basic machine cycles. They are:

Opcode fetch cycle (4T)
Memory read cycle (3 T)
Memory write cycle (3 T)
I/O read cycle (3 T)
I/O write cycle (3 T)

Note : *Time period, T = 1/f ; where f = Internal clock frequency*



Each instruction of the 8085 processor consists of one to five machine cycles, i.e., when the 8085 processor executes an instruction, it will execute some of the machine cycles in a specific order.

The processor takes a definite time to execute the machine cycles. The time taken by the processor to execute a machine cycle is expressed in T-states.

One T-state is equal to the time period of the internal clock signal of the processor.

The T-state starts at the falling edge of a clock.

Opcode Fetch Machine Cycle of 8085

- Each instruction of the processor has one byte opcode.
- The opcodes are stored in memory. So, the processor executes the opcode fetch machine cycle to fetch the opcode from memory.
- Hence, every instruction starts with opcode fetch machine cycle.
- The time taken by the processor to execute the opcode fetch cycle is 4T.
- In this time, the first, 3 T-states are used for fetching the opcode from memory and the remaining T-states are used for internal operations by the processor.

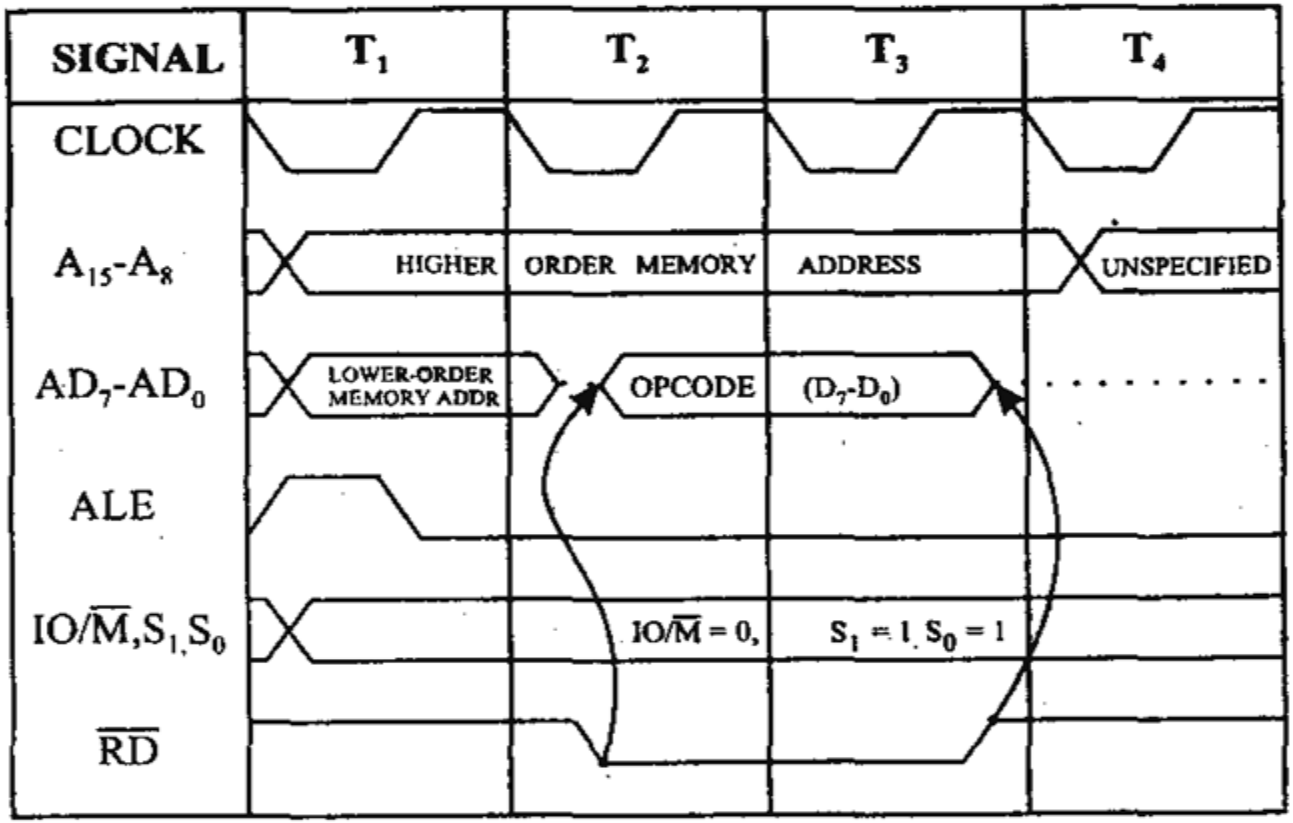


Fig. Timing diagram of Opcode fetch cycle

Memory Read Machine Cycle of 8085

- The memory read machine cycle is executed by the processor to read a data byte from memory.
- The processor takes 3T states to execute this cycle.
- The instructions which have more than one byte word size will use this machine cycle after the opcode fetch machine cycle.

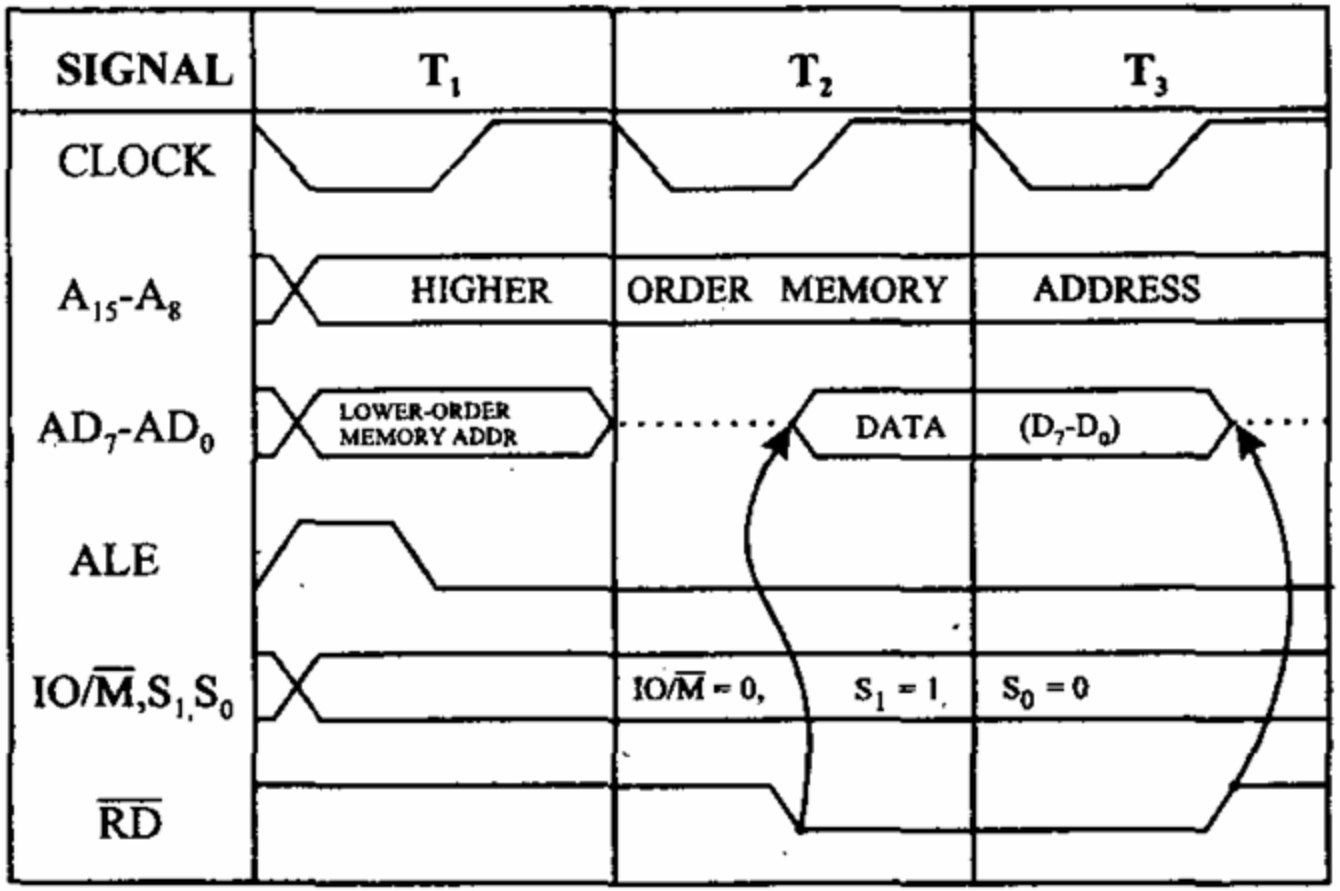


Fig. Timing diagram of memory read cycle

Memory Write Machine Cycle of 8085

- The memory write machine cycle is executed by the processor to write a data byte in a memory location.
- The processor takes, 3T states to execute this machine cycle.

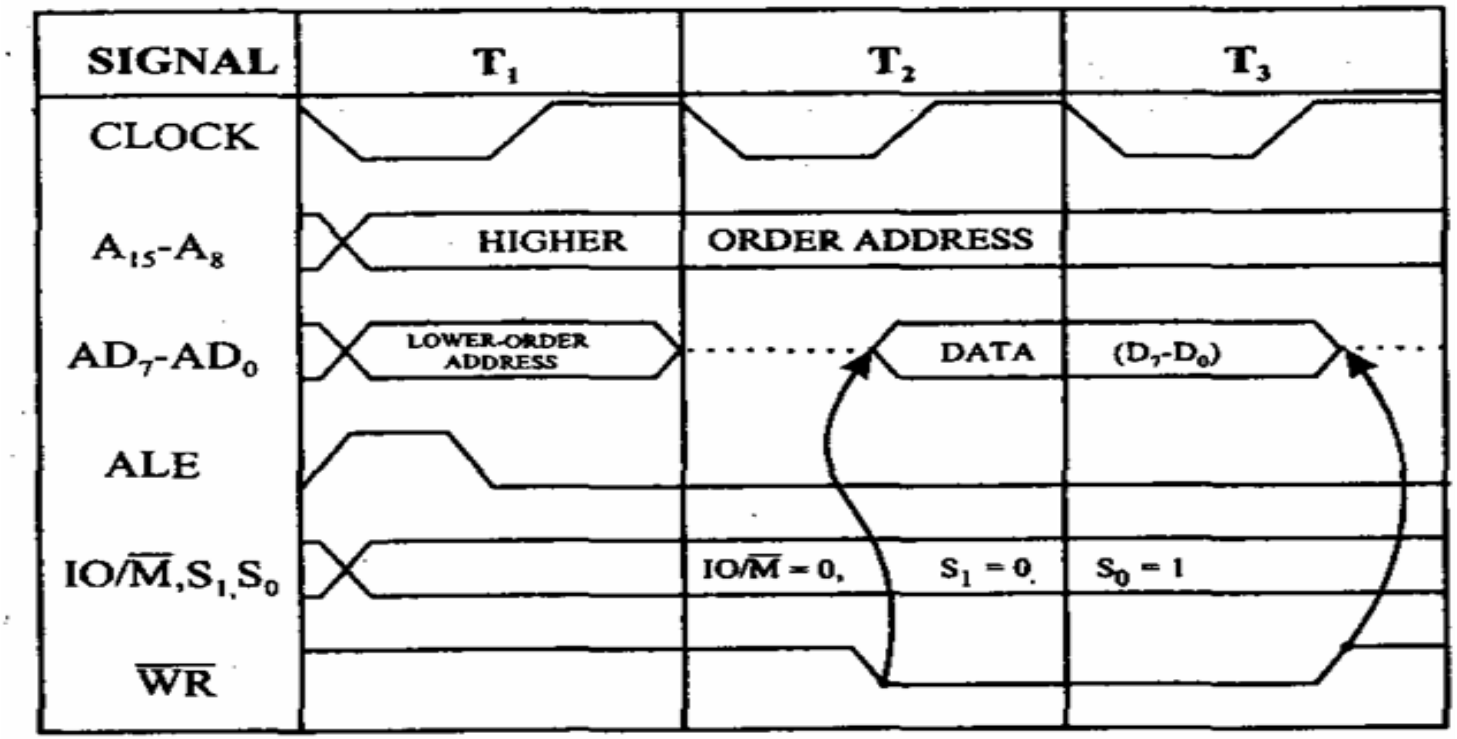


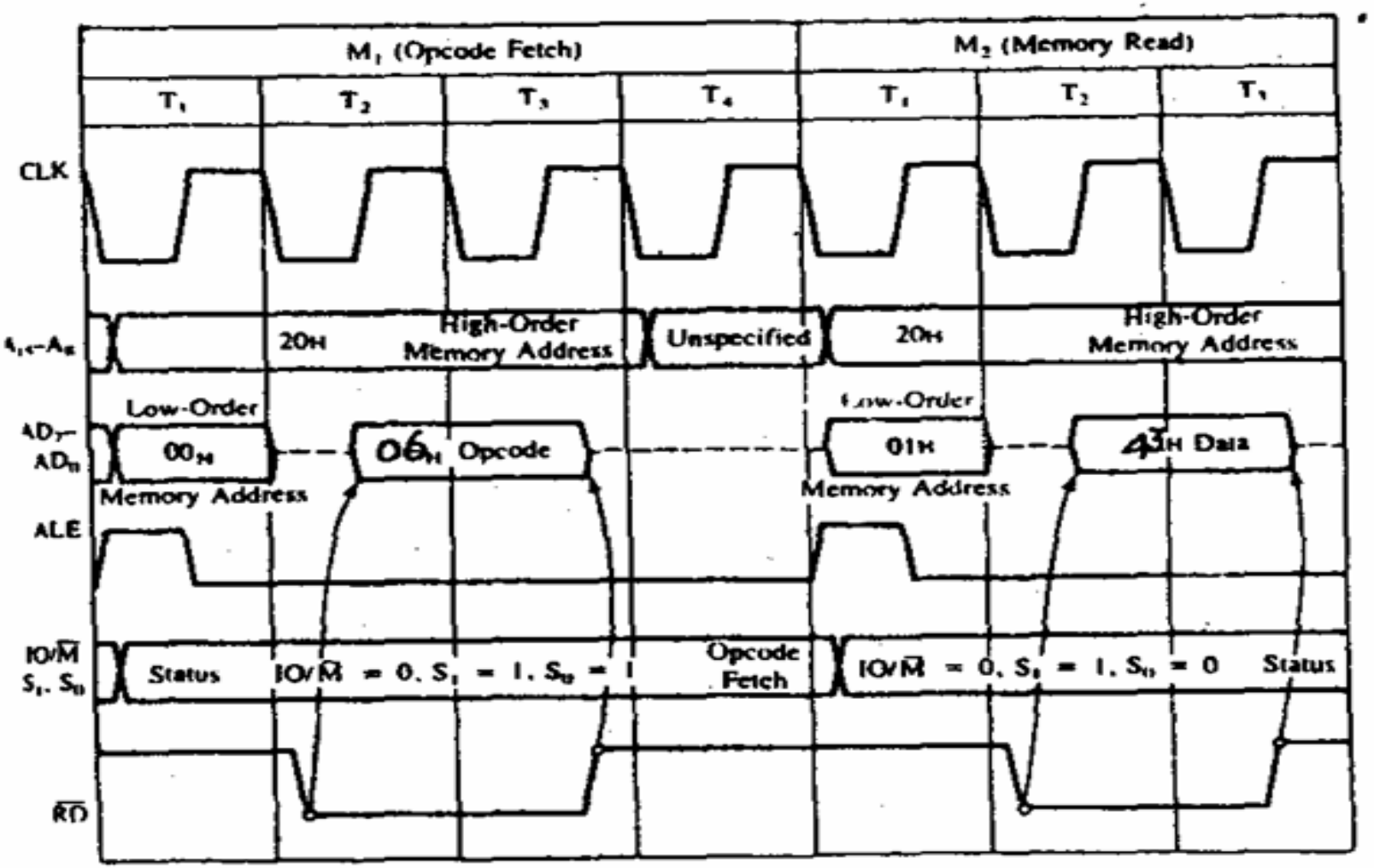
Fig. Timing diagram of memory write cycle

Timing diagram for MVI B, 43H.

Fetching the Opcode 06H from the memory 2000H. (OF machine cycle)

Read (move) the data 43H from memory 2001H. (memory read)

Address	Mnemonics	Opcode
2000	MVI B, 43_H	06_H
2001		43_H


M1 (Opcode Fetch)
M2 (Memory Read)
T1
T2
T3
T4
T1
T2
T3
CLK
A15–A8
20H
High-Order Memory Address
Unspecified
20H
High-Order Memory Address
Low-Order
AD7–AD0
00H
06H Opcode
01H
45H Data
Memory Address
Memory Address
ALE
IO/M
S1, S0
Status
IO/M = 0, S1 = 1, S0 = 1
Opcode Fetch
IO/M = 0, S1 = 1, S0 = 0
Status
RD

Timing diagram for INR M

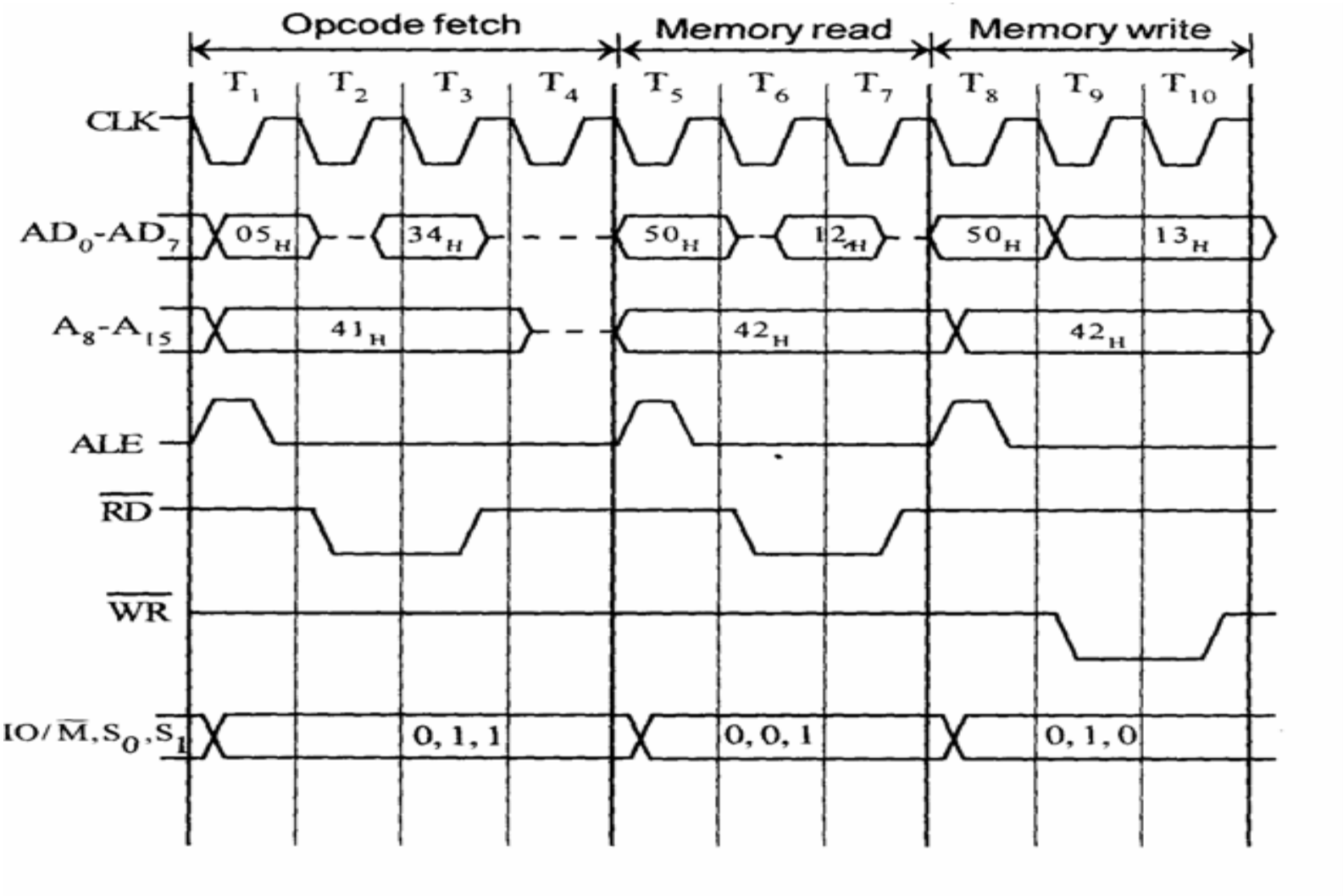
Fetching the Opcode 34H from the memory 4105H. (OF cycle)

Let the memory address (M) be 4250H. (MR cycle -To read Memory address and data)

Let the content of that memory is 12H.

Increment the memory content from 12H to 13H. (MW machine cycle)

Address	Mnemonics	Opcode
4105	INR M	34_H


Opcode fetch
Memory read
Memory write
T1
T2
T3
T4
T5
T6
T7
T8
T9
T10
CLK
AD0-AD7
05H
34H
50H
12H
50H
13H
A8-A15
41H
42H
42H
ALE
RD
WR
IO/M, S0, S1
0, 1, 1
0, 0, 1
0, 1, 0

Timing diagram for STA 526A$_H$

Address	Mnemonics	Opcode
41FF	STA 526A$_H$	32$_H$
4200		6A$_H$
4201		52$_H$

STA means Store Accumulator -The contents of the accumulator is stored in the specified address (526A).

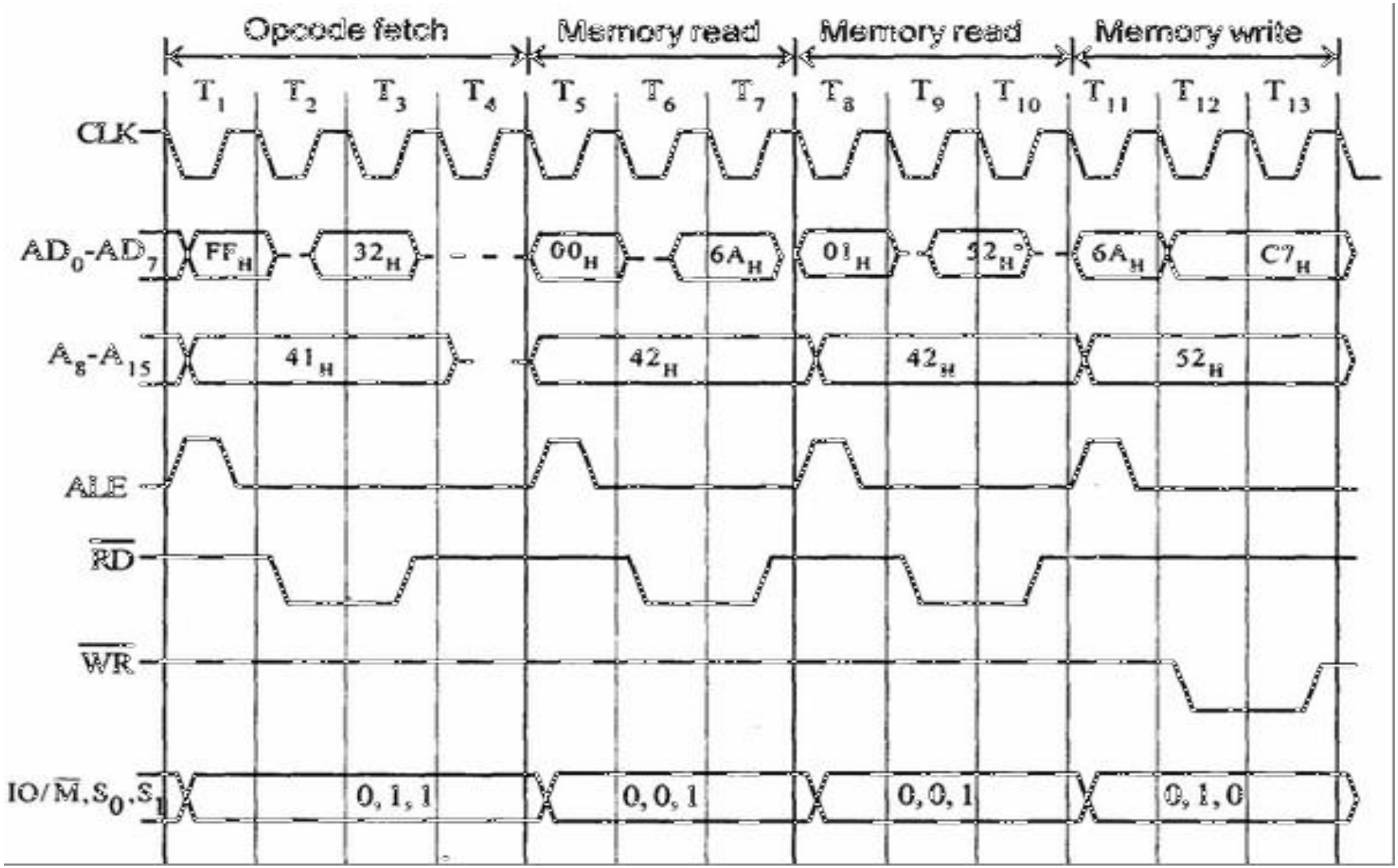
The opcode of the STA instruction 32_H is fetched from the memory $41FF_H$ - *OF machine cycle*

Then the lower order memory address is read (6A). - *Memory Read Machine Cycle*

Read the higher order memory address (52).- *Memory Read Machine Cycle*

The combination of both the addresses are considered and the content from accumulator is written in $526A_H$. - *Memory Write Machine Cycle*

Let the content of accumulator is $C7_H$. So, C7H from accumulator is now stored in $526A_H$.


Opcode fetch
Memory read
Memory read
Memory write
T1 T2 T3 T4 T5 T6 T7 T8 T9 T10 T11 T12 T13
CLK
AD0-AD7
FFH
32H
00H
6AH
01H
32H
6AH
C7H
A8-A15
41H
42H
42H
52H
ALE
RD
WR
IO/M, S0, S1
0, 1, 1
0, 0, 1
0, 0, 1
0, 1, 0

I/O Read Cycle of 8085

- The I/O Read cycle is executed by the processor to read a data byte from I/O port or from the peripheral, which is I/O-mapped in the system.
- The processor takes 3T states to execute this machine cycle.
- The IN instruction uses this machine cycle during the execution.

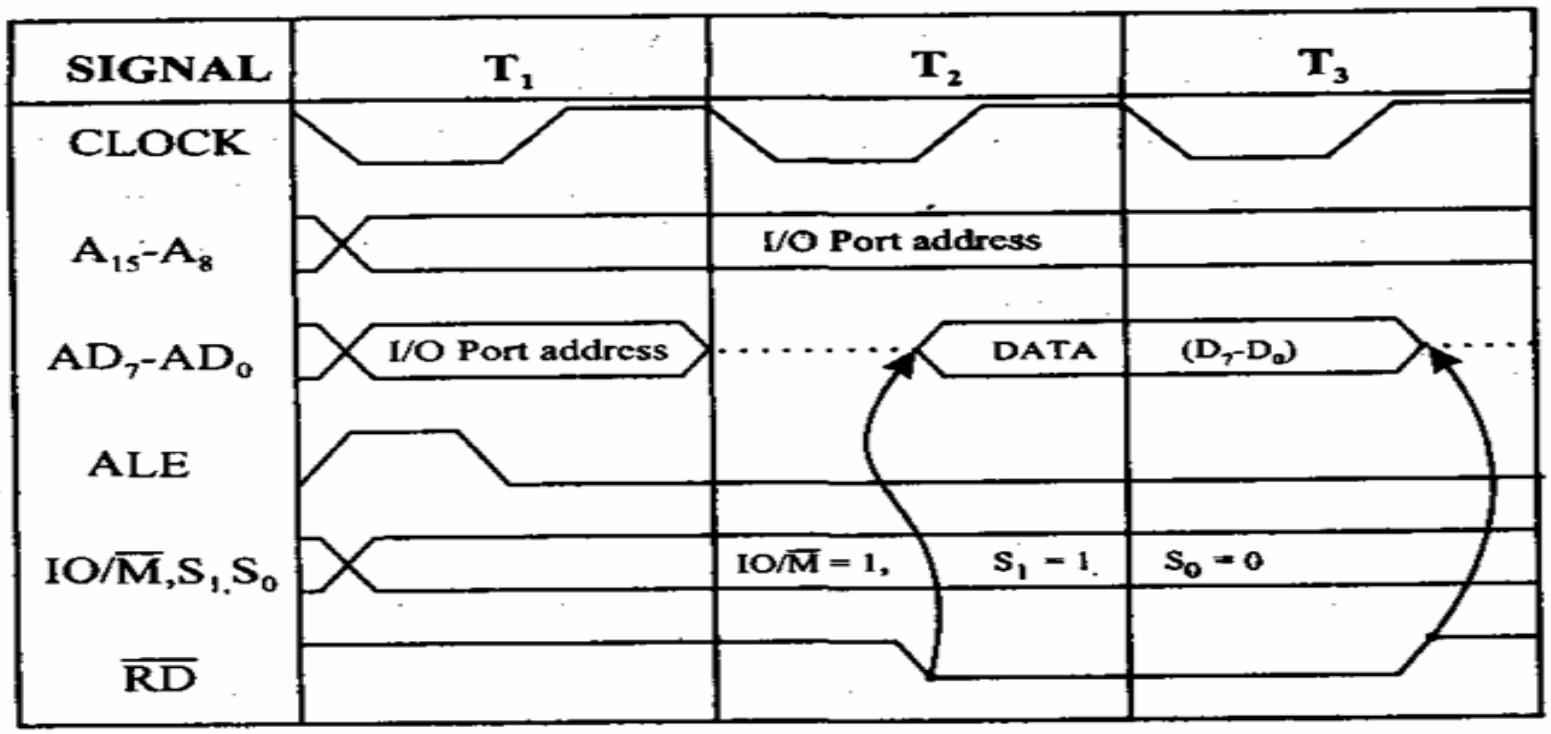


Fig. Timing diagram of I/O read cycle

I/O Write Cycle of 8085

- The I/O write machine cycle is executed by the processor to write a data byte in the I/O port or to a peripheral, which is I/O-mapped in the system.
- The processor takes, 3T states to execute this machine cycle.
- The OUT instruction uses this machine cycle during the execution.

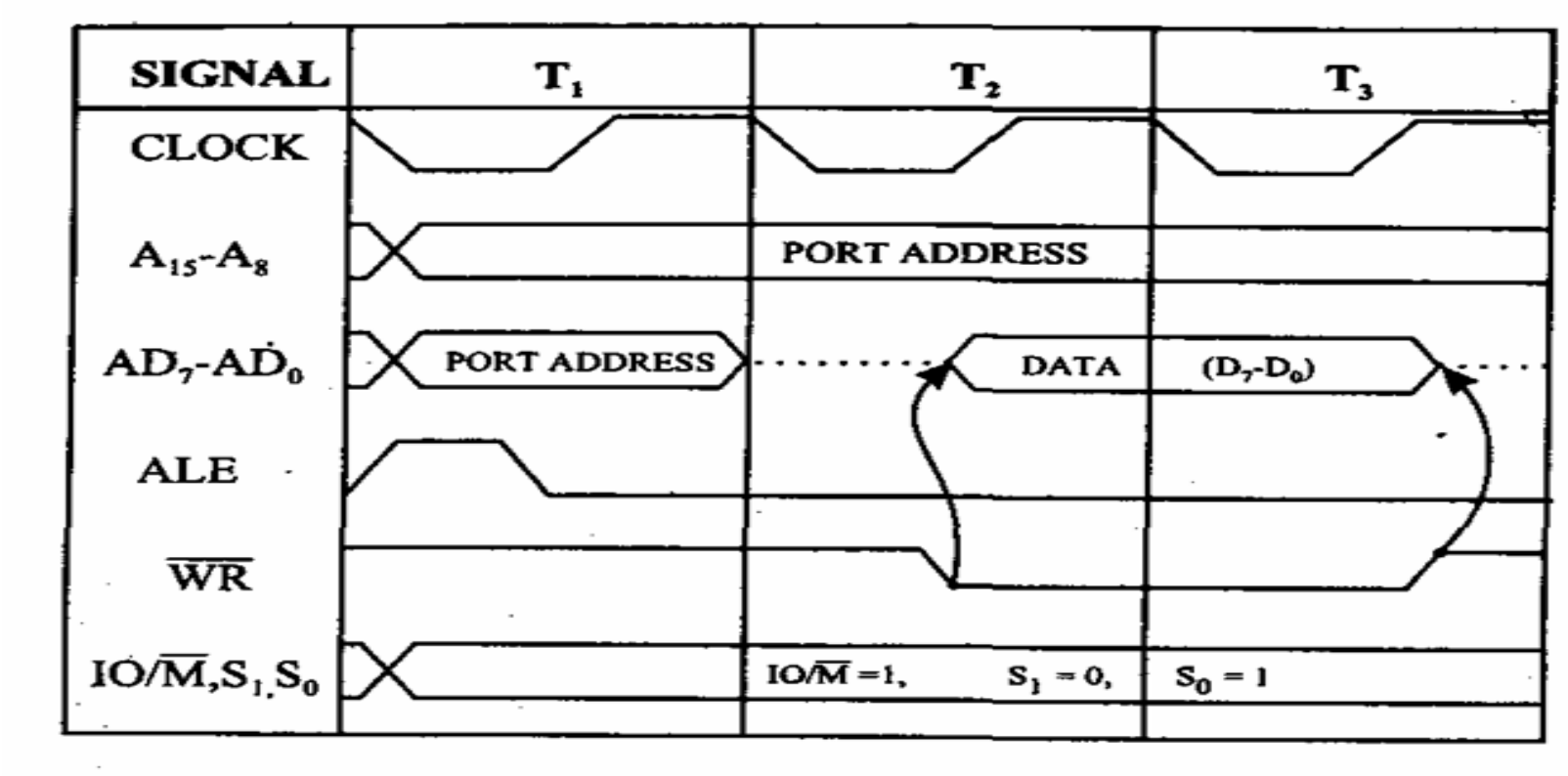


Fig. Timing diagram of I/O write cycle

Timing diagram for IN $C0_H$.

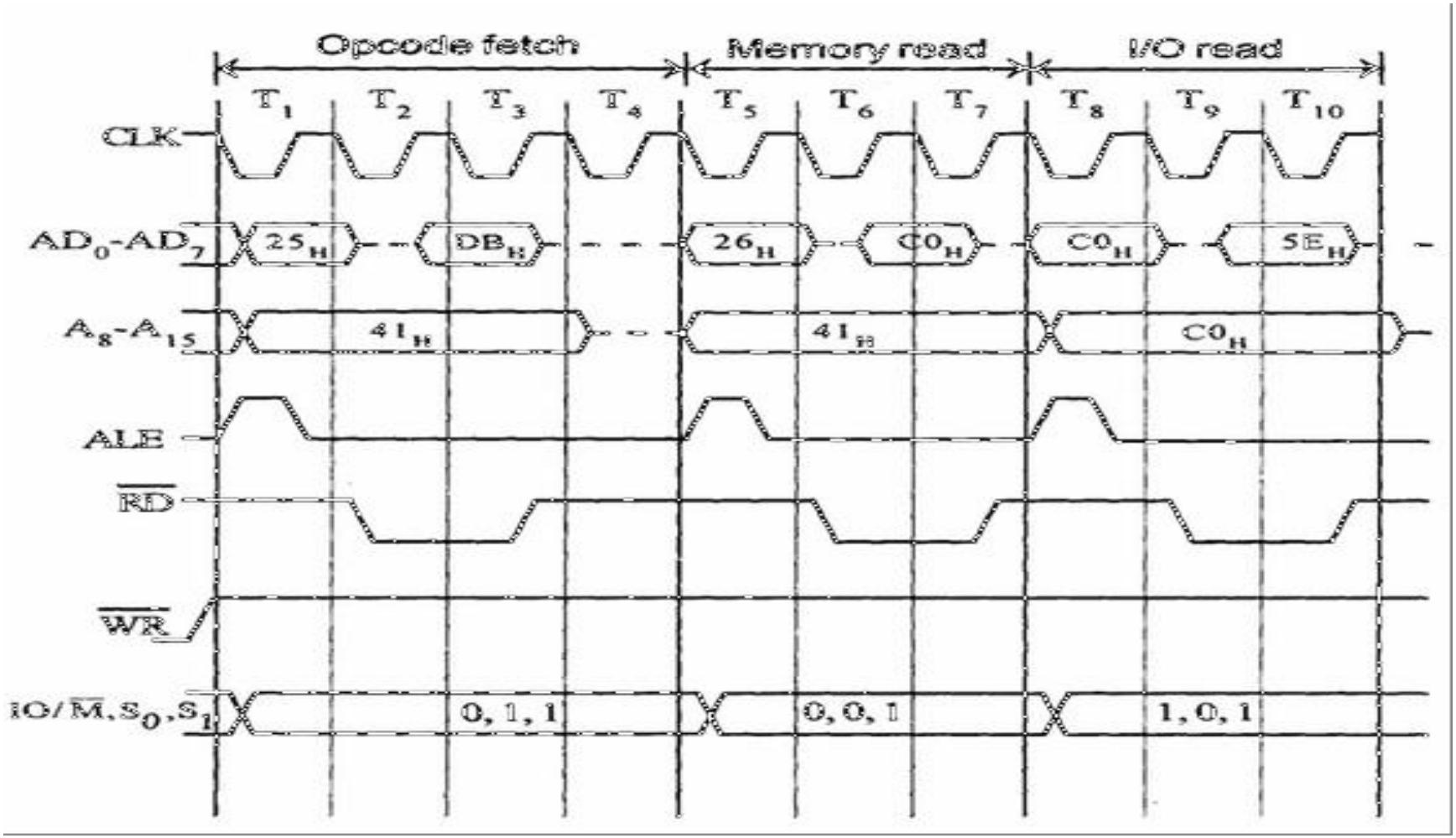
Address	Mnemonics	Opcode
4125	IN $C0_H$	DB_H
4126		$C0_H$

Fetching the Opcode DB_H from the memory 4125_H.

Read the port address $C0_H$ from 4126_H.

Read the content of port $C0_H$ and send it to the accumulator.

Let the content of port is $5E_H$.


Opcode fetch
Memory read
I/O read
T1
T2
T3
T4
T5
T6
T7
T8
T9
T10
CLK
AD0-AD7
25H
DBH
26H
C0H
C0H
5EH
A8-A15
41H
41H
C0H
ALE
RD
WR
IO/M, S0, S1
0, 1, 1
0, 0, 1
1, 0, 1

TIME DELAY

Sometimes there is a need to perform tasks after some specific delay. The amount of delay can be realized by writing a delay program.

```
        MVI     C       xx_H
Loop:   DCR     C
        JNZ     Loop
```

TimeDelay for loop (T_L) = {(No. of T states) * T_Time * xx} – 3 * T_Time

= {(4+10) * P * xx} – 3P

= (14P * xx) – 3P (1)

T_{Total} = T_L + 7P

= (14P * xx) – 3P + 7 P

= (14P * xx) + 4P (2)

Example: Let CPU CLK speed: 3 MHz

T_Time (P) = $\frac{1}{3} \times 10^{-6}$ Seconds

Interrupts

Interrupt is signals send by an external device to the processor, to request the processor to perform a particular task or work.

Mainly, in the microprocessor based system the interrupts are used for data transfer between the peripheral and the microprocessor.

The processor will check the interrupts always at the 2nd T-state of last machine cycle.

1) If there is any interrupt, it accept the interrupt and send the INTA (active low) signal to the peripheral.

2) The vectored address of particular interrupt is stored in program counter.

3) The processor executes an interrupt service routine (ISR) addressed in program counter.

4) It returned to main program by RET instruction.

Types of Interrupts:

Types of interrupts.

A.	(i)	Software	(ii)	Hardware
B.	(i)	Vectored	(ii)	Non-vectored
C.	(i)	Maskable	(ii)	Non-maskable

(i) Software interrupts:

The software interrupts are program instructions. These instructions are inserted at desired locations in a program.

The 8085 has eight software interrupts from RST 0 to RST 7. The vector address for these interrupts can be calculated as follows.

Interrupt number * 8 = vector address

Ex. For RST 5: 5 * 8 = 40 = 28_H

Vector address for interrupt RST 5 is 0028_H

The Table shows the vector addresses of all interrupts.

Interrupt	Vector address
RST 0	0000_H
RST 1	0008_H
RST 2	0010_H
RST 3	0018_H
RST 4	0020_H
RST 5	0028_H
RST 6	0030_H
RST 7	0038_H

(ii) Hardware interrupts:

An external device initiates the hardware interrupts by placing an appropriate signal at the interrupt pin of the processor.

If the interrupt is accepted then the processor executes an interrupt service routine.

The 8085 has five hardware interrupts

(1) TRAP (2) RST 7.5 (3) RST 6.5

(4) RST 5.5 (5) INTR

Interrupt	Vector address
RST 7.5	$003C_H$
RST 6.5	0034_H
RST 5.5	$002C_H$
TRAP	0024_H

TRAP:

This interrupt is a non-maskable interrupt. It is unaffected by any mask or interrupt enable.

TRAP bas the highest priority and vectored interrupt.

TRAP interrupt is edge and level triggered. This means hat the TRAP must go high and remain high until it is acknowledged.

In sudden power failure, it executes a ISR and send the data from main memory to backup memory.

The signal, which overrides the TRAP, is HOLD signal. (i.e., If the processor receives HOLD and TRAP at the same time then HOLD is recognized first and then TRAP is recognized).

There are two ways to clear TRAP interrupt.

1. By resetting microprocessor (External signal)
2. By giving a high TRAP ACKNOWLEDGE (Internal signal)

RST 7.5:

The RST 7.5 interrupt is a maskable interrupt.

It has the second highest priority.

It is edge sensitive. i.e. Input goes to high and no need to maintain high state until it recognized.

Maskable interrupt. It is disabled by,

1. DI instruction
2. System or processor reset.
3. After reorganization of interrupt.

RST 6.5 and 5.5:

The RST 6.5 and RST 5.5 both are level triggered. i.e. Input goes to high and stay high until it recognized.

Maskable interrupt. It is disabled by,

1. DI, SIM instruction
2. System or processor reset.
3. After reorganization of interrupt.

Enabled by EI instruction.

The RST 6.5 has the third priority whereas RST 5.5 has the fourth priority.

INTR:

INTR is a maskable interrupt. It is disabled by,

1. DI, SIM instruction
2. System or processor reset.
3. After reorganization of interrupt.

Enabled by EI instruction.

Non-vectored interrupt. After receiving INTA (active low) signal, it has to supply the address of ISR.

It has lowest priority.

It is a level sensitive interrupts. ie. Input goes to high and it is necessary to maintain high state until it recognized.

The following sequence of events occurs when INTR signal goes high.

1. The 8085 checks the status of INTR signal during execution of each instruction.

2. If INTR signal is high, then 8085 complete its current instruction and sends active low interrupt acknowledge INTA (active low) signal, if the interrupt is enabled.

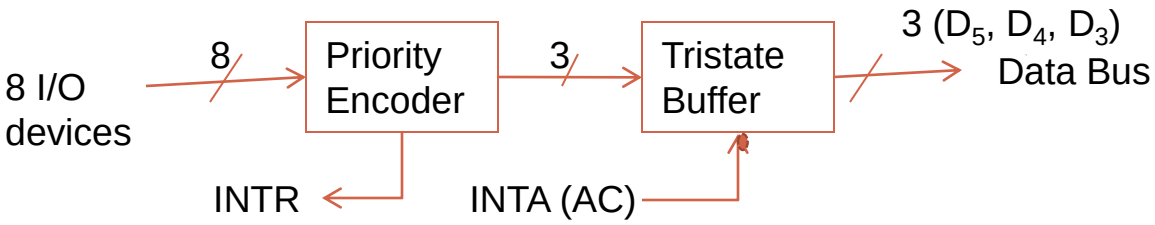
3. In response to the acknowledge signal, external logic places an RST instruction on the data bus. In the case of multi-byte instruction, additional interrupt acknowledge machine cycles are generated by the 8085 to transfer the additional bytes into the microprocessor.

4. On receiving the RST instruction, the 8085 save the address of next instruction on stack and execute the received instruction. The program is transferred to vector location of RST instruction and execute the ISR.

5. If ISR is written in some other location---
   Vectored location should have a JMP instruction to the ISR.

6. At the end of ISR, it should have EI instruction to enable the interrupt again and the RET instruction to resume program execution from the point before the interruption.

Multiple Interrupts & Priorities



	D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0	OpCode
RST 0	1	1	0	0	0	1	1	1	= C7
RST 1	1	1	0	0	1	1	1	1	= CF
RST 2	1	1	0	1	0	1	1	1	= D7
RST 3	1	1	0	1	1	1	1	1	= DF
RST 4	1	1	1	0	0	1	1	1	= E7
RST 5	1	1	1	0	1	1	1	1	= EF
RST 6	1	1	1	1	0	1	1	1	= F7
RST 7	1	1	1	1	1	1	1	1	= FF

SIM and RIM for interrupts:

The 8085 provide additional masking facility for RST 7.5, RST 6.5 and RST 5.5 using SIM instruction.

The status of these interrupts can be read by executing RIM instruction.

The masking or unmasking of RST 7.5, RST 6.5 and RST 5.5 interrupts can be performed by moving an 8-bit data to accumulator and then executing SIM instruction.

The format of the 8-bit data is shown below.

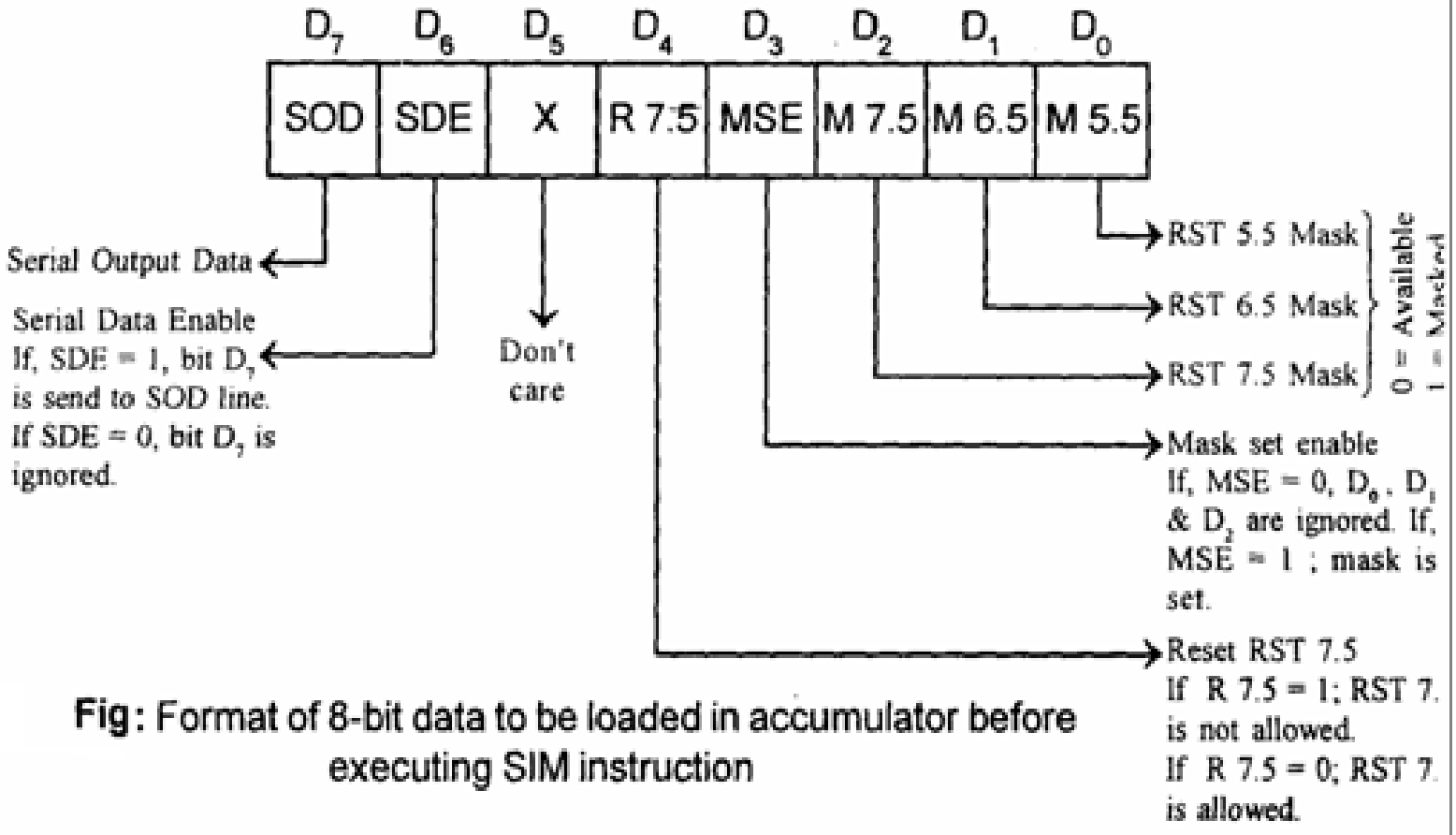
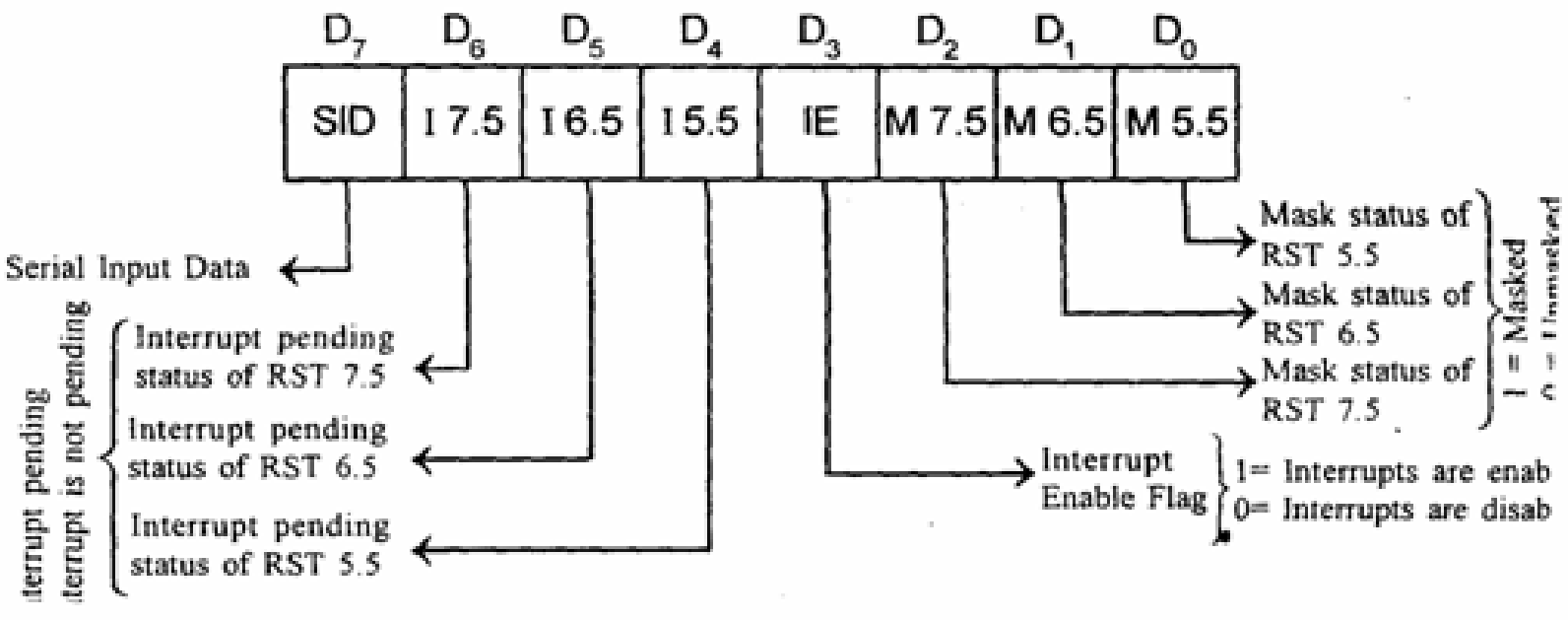


Fig: Format of 8-bit data to be loaded in accumulator before executing SIM instruction

The status of pending interrupts can be read from accumulator after executing RIM instruction.

When RIM instruction is executed an 8-bit data is loaded in accumulator, which can be interpreted as shown in fig.



Interrupt type	Trigger	Priority	Maskable	Vector address
TRAP	Edge and Level	1st	No	0024H
RST 7.5	Edge	2nd	Yes	003CH
RST 6.5	Level	3rd	Yes	0034H
RST 5.5	Level	4th	Yes	002CH
INTR	Level	5th	Yes	-

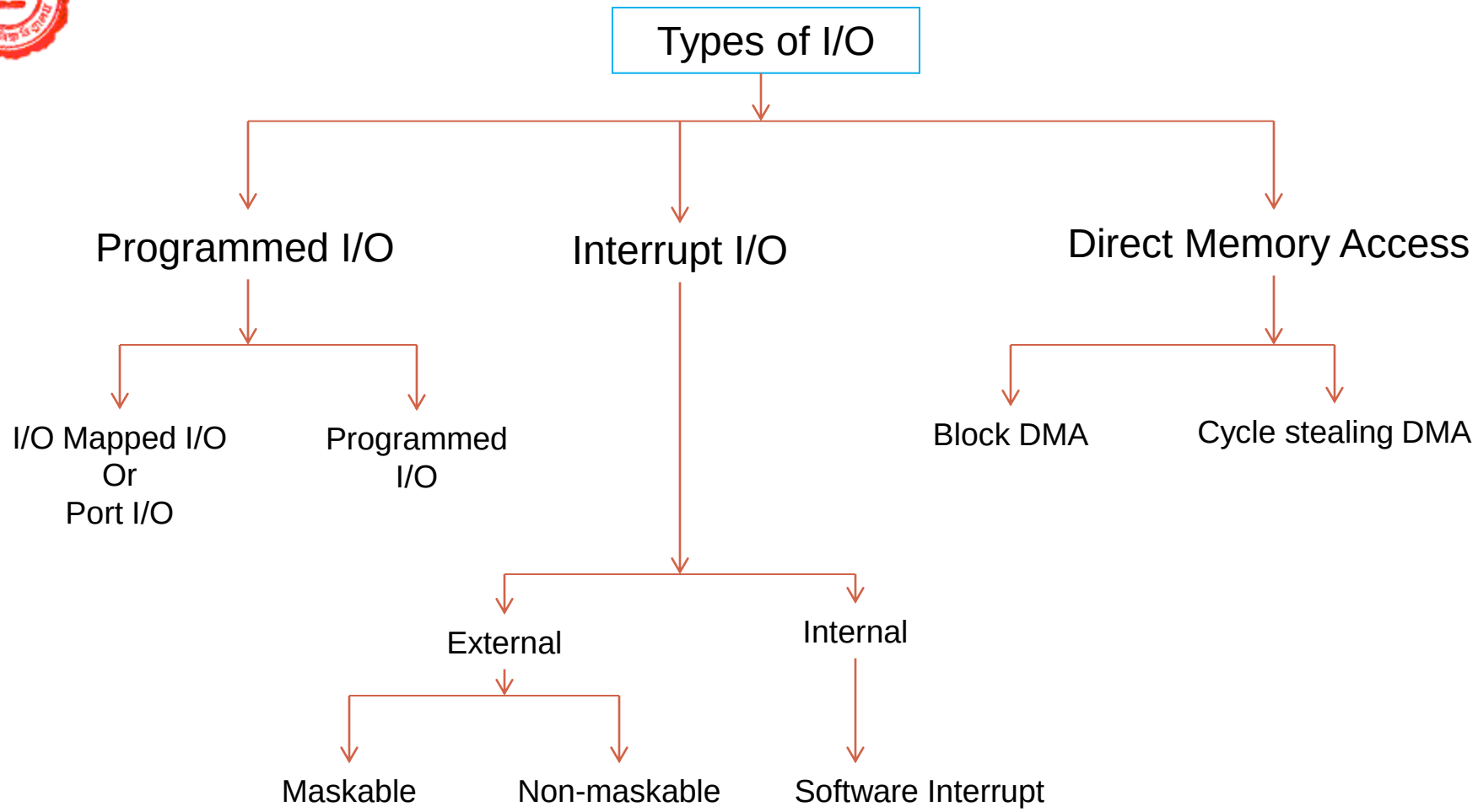
I/O STRUCTURE OF A TYPICAL MICROCOMPUTER

There are three major types of data transfer between the microcomputer and art I/O device. They are,

Programmed I/O : In programmed I/O the data transfer is accomplished through an I/O port and controlled by software.

Interrupt driven I/O : In interrupt driven I/O, the I/O device will interrupt the processor, and initiate data transfer.

Direct memory access (DMA) : In DMA, the data transfer between memory and I/O can be performed by bypassing the microprocessor.


Types of I/O
Programmed I/O
Interrupt I/O
Direct Memory Access
I/O Mapped I/O
Or
Port I/O
Programmed
I/O
Block DMA
Cycle stealing DMA
External
Internal
Maskable
Non-maskable
Software Interrupt

Programmable Interfacing Device (PID)

I/O Devices are not directly connected to the MPU, but via PID

PID should have:

1) I/O registers
2) Tri-state buffers
3) Capability of bi-directional data transfer
4) Handshake and interrupt signals
5) Control logic
6) Chip select logic
7) Interrupt control logic

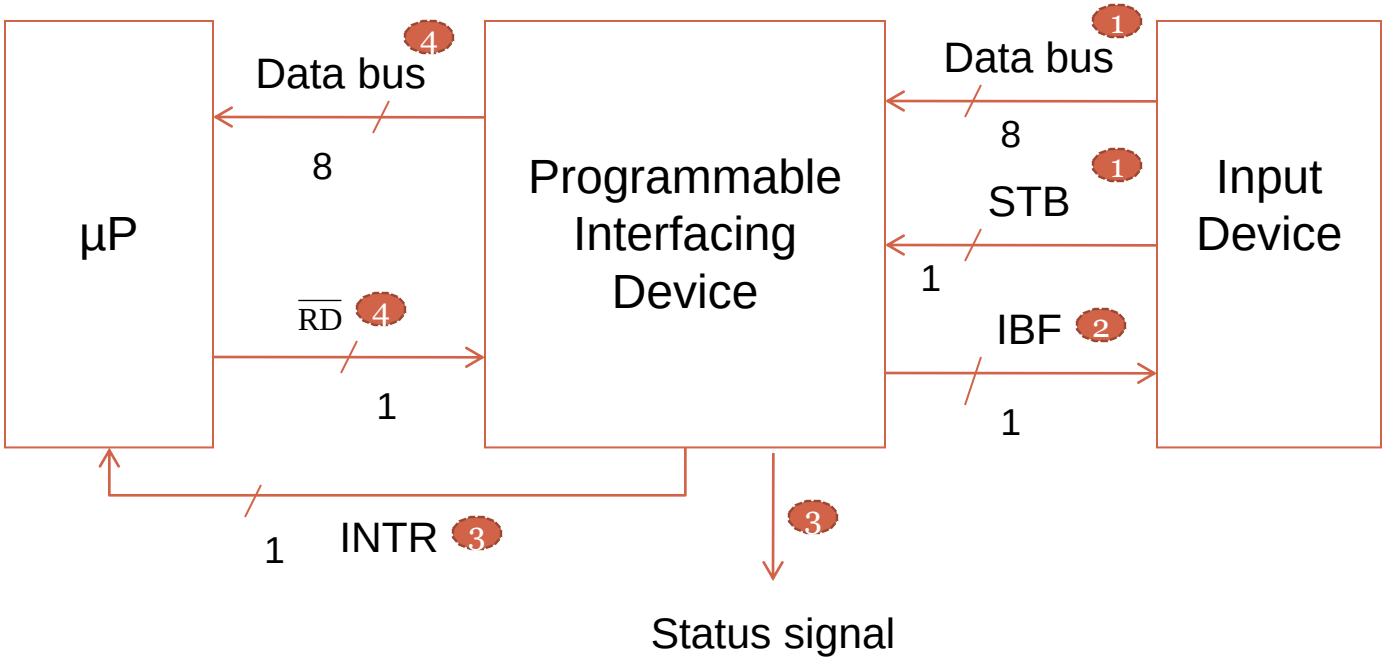
1. Data transfer from input device to processor: (General sequences)

1) The input device will load the data to the port.
2) When the port receives a data, it sends message to the processor to read the data.
3) The processor will read the data from the port.
4) After a data have been read by the processor the input device will load the next data into the port.

2. Data transfer from processor to output device: (General sequences)

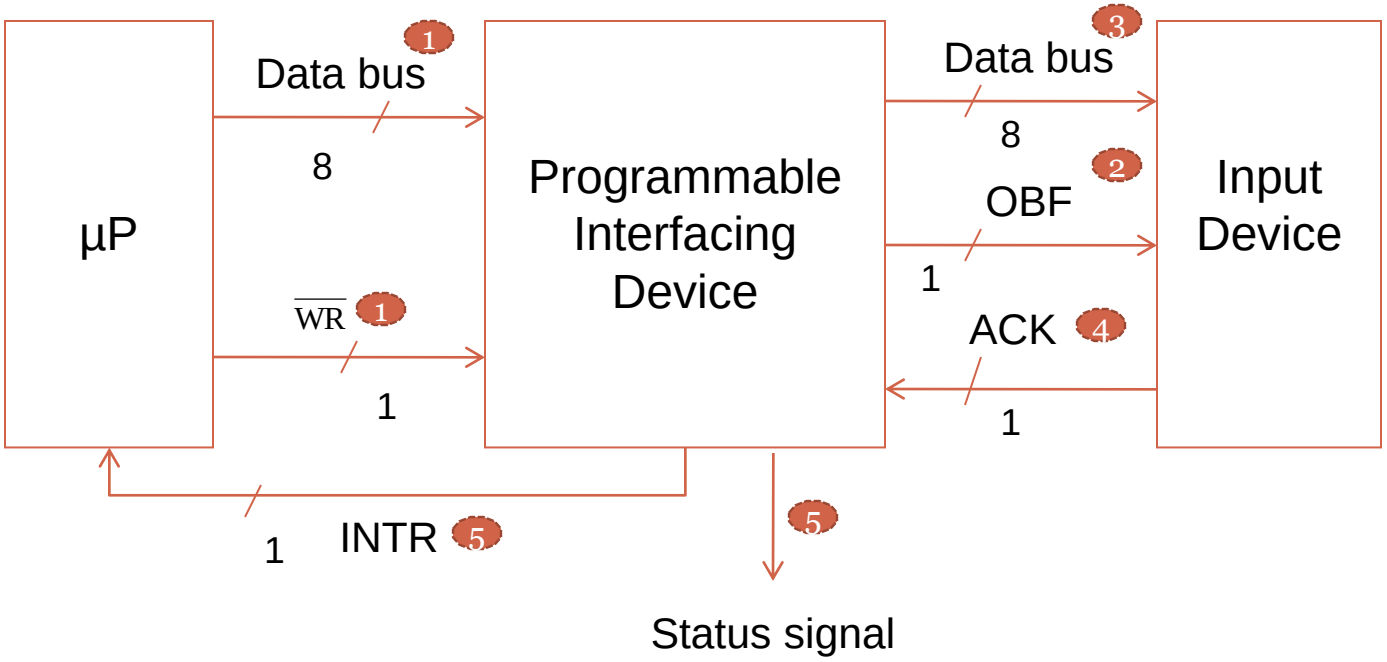
1) The processor will load the data to the port.
2) The port will send a message to the output device to read the data.
3) The output device will read the data from the port.
4) After the data have been read by the output device the processor can load the next data to the port.

Data transfer from I/O device to Microprocessor



STB: Strobe Signal
IBF: Input Buffer Full
INTR: Interrupt Request

Data transfer from Microprocessor to I/O device



OBF: Output Buffer Full
ACK: Acknowledgement
INTR: Interrupt Request

There are two types for interfacing I/O devices:

1) Memory mapped I/O device.
2) Standard I/O mapped I/O device or isolated I/O mapping.

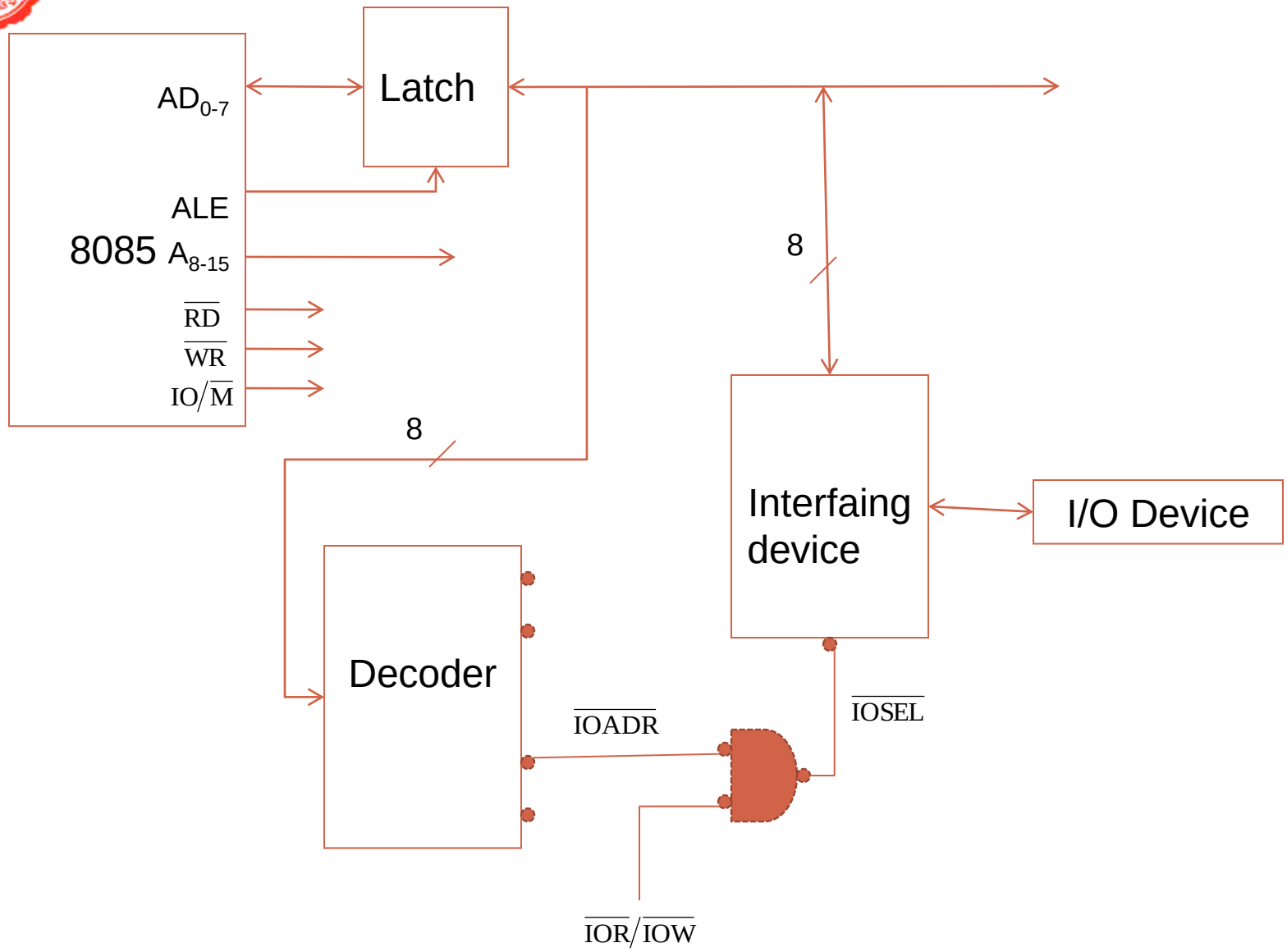
Memory Mapping of I/O device	I/O Mapping of I/O device
1. 16-bit addresses are provided for I/O devices.	1. 8-bit addresses are provided for I/O devices.
2. The devices are accessed by memory read or memory write cycles.	2. The devices are accessed by I/O read or I/O write cycle. During these cycles the 8-bit address is available on both low order address lines and high order address lines.
3. The I/O ports or peripherals can be treated like memory locations and so all instructions related to memory can be used for data transfer between I/O device and the processor.	3. Only IN and OUT instructions can be used for data transfer between I/O device and the processor.
4. In memory mapped ports the data can be moved from any register to ports and vice-versa.	4. In I/O mapped ports the data transfer can take place only between the accumulator and ports.
5. When memory mapping is used for I/O devices, the full memory address space cannot be used for addressing memory. Hence memory mapping is useful only for small systems, where the memory requirement is less.	5. When I/O mapping is used for I/O devices then the full memory address space can be used for addressing memory. Hence it is suitable for systems which requires large memory capacity.
6. In memory mapped I/O devices, a large number of I/O ports can be interfaced.	6. In I/O mapping only 256 ports ($2^8 = 256$) can be interfaced.
7. For accessing the memory mapped devices, the processor executes memory read or write cycle. During this cycle IO/$\overline{M}$ is asserted **low** (IO/$\overline{M} = 0$).	7. For accessing the I/O mapped devices, the processor executes I/O read or write cycle. During this cycle IO/$\overline{M}$ is asserted **high** (IO/$\overline{M} = 1$).

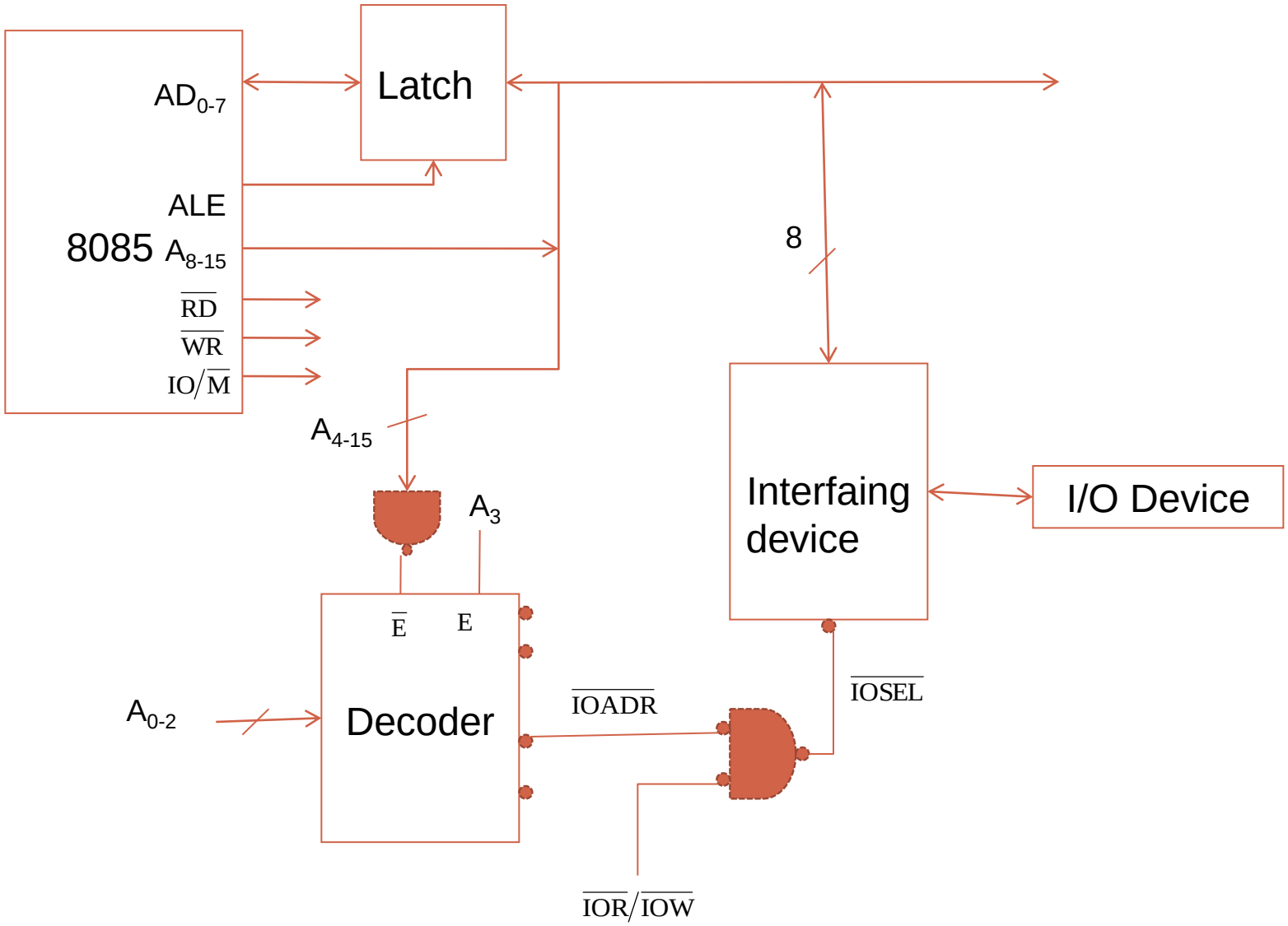
Device Selection

1) Decode the address bus to generate a unique pulse corresponding to the device address on the bus, I/O address pulse.

2) Combine (AND) the I/O address pulse with the control signal to generate a device select (I/O select) pulse that is generated only when both the signals are asserted.

3) Use the I/O select pulse to activate the interfacing device (I/O port).


Latch
AD_{0-7}
ALE
8085
A_{8-15}
$\overline{RD}$
$\overline{WR}$
$IO/\overline{M}$
8
8
Interfaing device
I/O Device
Decoder
$\overline{IOADR}$
$\overline{IOSEL}$
$\overline{IOR}/\overline{IOW}$



I/O Port Address: $FFF9_H$

Peripheral Interfacing is considered to be a main part of Microprocessor, as it is the only way to interact with the external world. The interfacing happens with the ports of the Microprocessor.

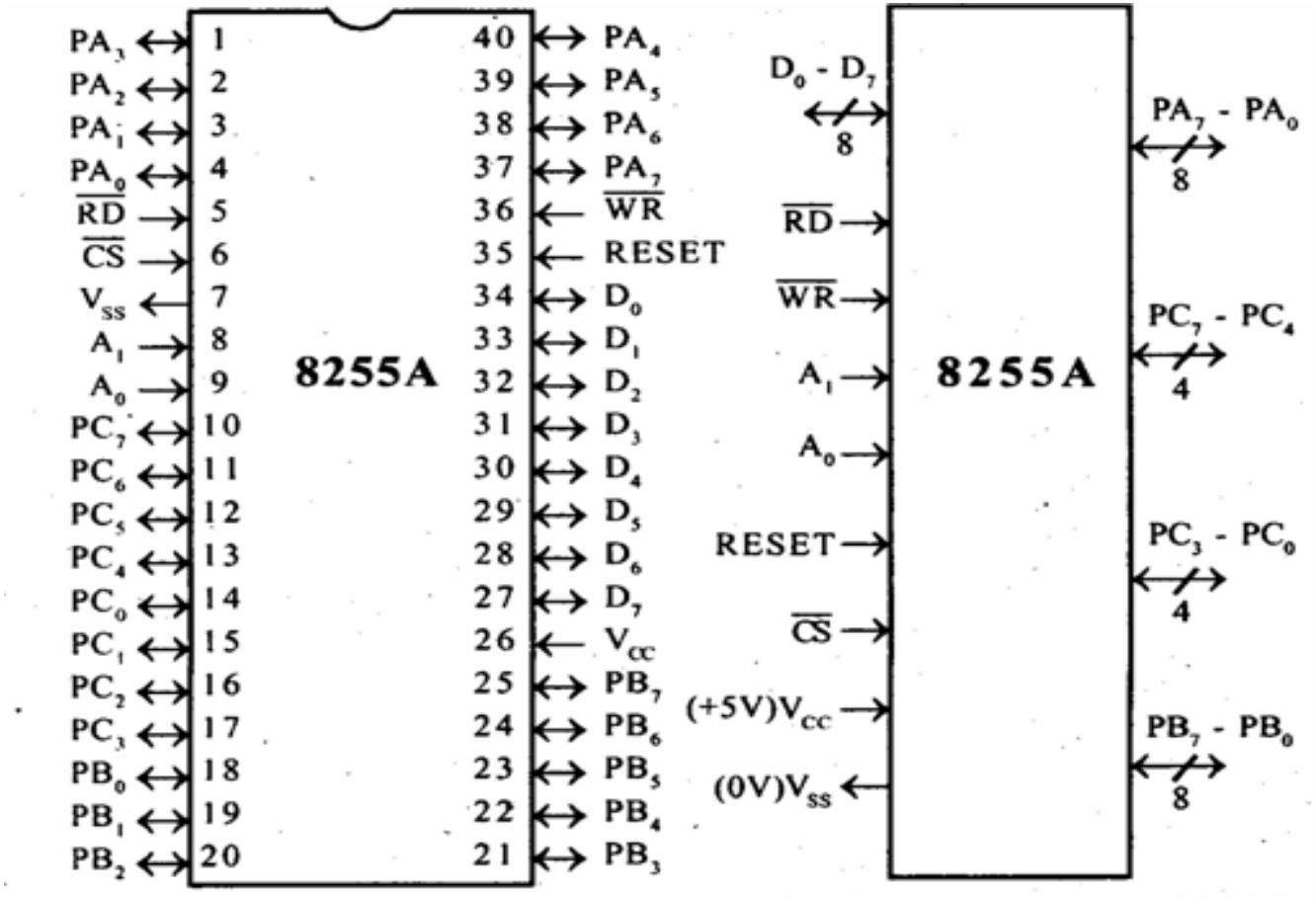
The main IC's which can be interfaced with 8085 are:

1. 8255 PPI (programmable Peripheral Interface)
2. 8259 PIC (Programmable Interrupt Controller)
3. 8251 USART (Universal Synchronous Asynchronous Receiver & Transmitter)
4. 8279 Key board display controller
5. 8253 Timer/ Counter
6. A/D and D/A converter interfacing.

PROGRAMMABLE PERIPHERAL INTERFACE - INTEL 8255

Pins, Signals and internal block diagram of 8255:

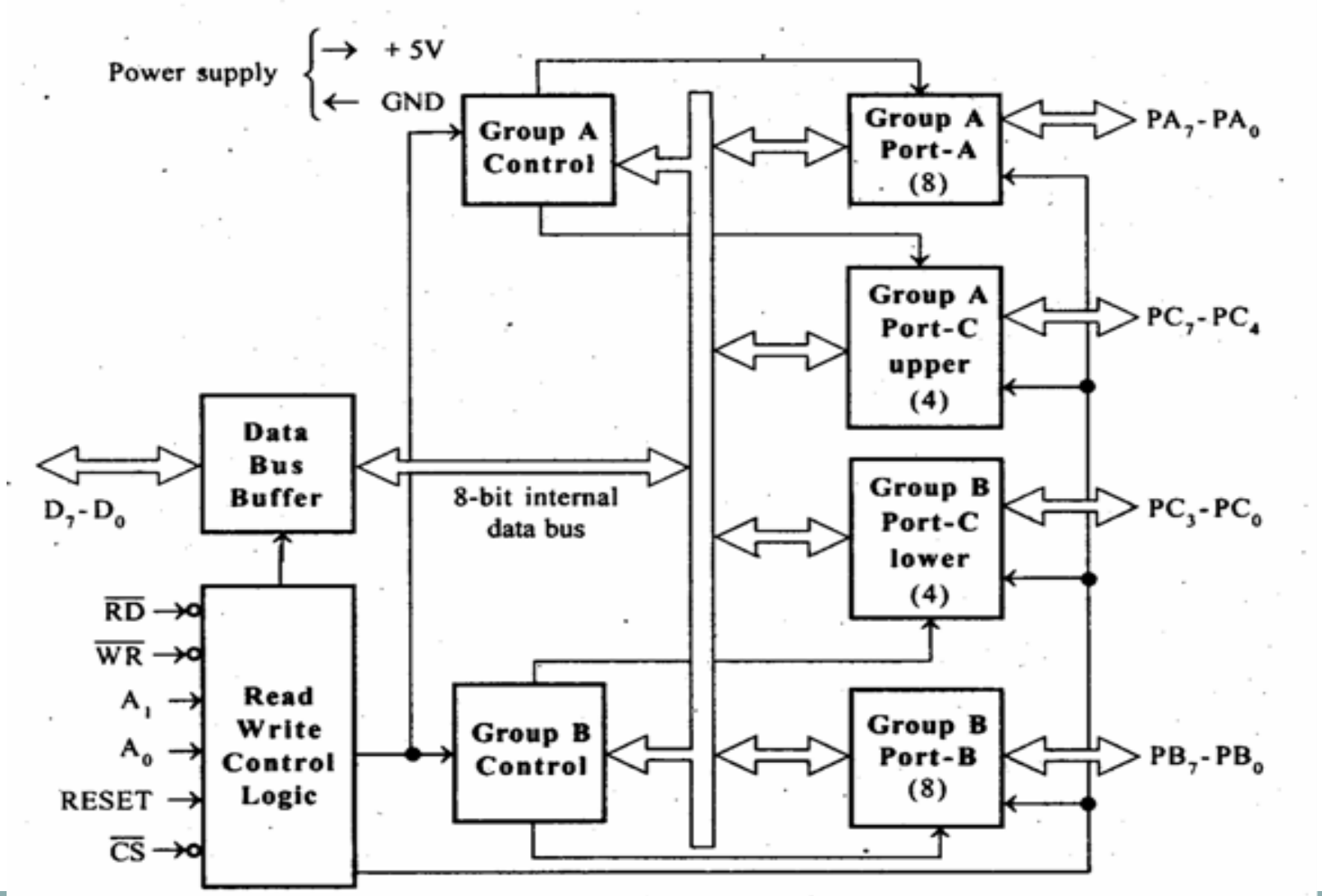
It has 40 pins and requires a single +5V supply.



PIN DESCRIPTION - INTEL 8255

Pin	Description
D_0 - D_7	Data lines
RESET	Reset input
$\overline{CS}$	Chip select
$\overline{RD}$	Read control
$\overline{WR}$	Write control
A_0, A_1	Internal address
PA_7 - PA_0	Port-A pins
PB_7 - PB_0	Port-B pins
PC_7 - PC_0	Port-C pins
V_{CC}	+5V
V_{SS}	0V (GND)

The internal block diagram of 8255



The INTEL 8255 is a device used to parallel data transfer between processor and slow peripheral devices like ADC, DAC, keyboard, 7-segment display, LCD, etc. and can be used as I/O mode or Bit Set/Reset (BSR) mode.

The 8255 has three ports: Port-A, Port-B and Port-C.

Port-A can be programmed to work in any one of the three operating modes mode-0, mode-1 and mode-2 as input or output port.

Port-B can be programmed to work either in mode-0 or mode-1 as input or output port.

Port-C (8-pins) has different assignments depending on the mode of port-A and port-B.

Mode 0: Simple input or output

- Outputs are latched.
- Inputs are not latched.
- Ports do not have handshake or interrupt capability.

Mode 1: Input or Output with handshake

- Port A & B works as I/O ports.
- Each port uses 3 lines from port C as handshake signals. Remaining 2 lines can be used as simple I/O.
- Input and Output data are latched.
- Interrupt logic is supported.

Mode 2: Bidirectional Data Transfer

- Port A can be used as bidirectional port.
- Port B as either Mode 0 or Mode 1.
- Port A uses 5 lines of Port C for handshake.
- Remaining 3 lines can be used simple I/O or handshake for Port B.

If port-A and B are programmed in mode-0, then the port-C can perform any one of the following functions.

1. As 8-bit parallel port in mode-0 for input or output.
2. As two numbers of 4-bit parallel ports in mode-0 for input or output.
3. The individual pins of port-C can be set or reset for various control applications.

If port-A is programmed in mode- 1/mode-2 and port-B is programmed in mode-1 then some of the pins of port-C are used for handshake signals and the remaining pins can be used as input/ output lines or individually set/reset for control applications.

The read/write control logic requires six control signals.

1. RD (low): This control signal enables the read operation. When this signal is low, the microprocessor reads data from a selected I/O port of the 8255A.

2. WR (low): This control signal enables the write operation. When this signal goes low, the microprocessor writes into a selected I/O port or the control register.

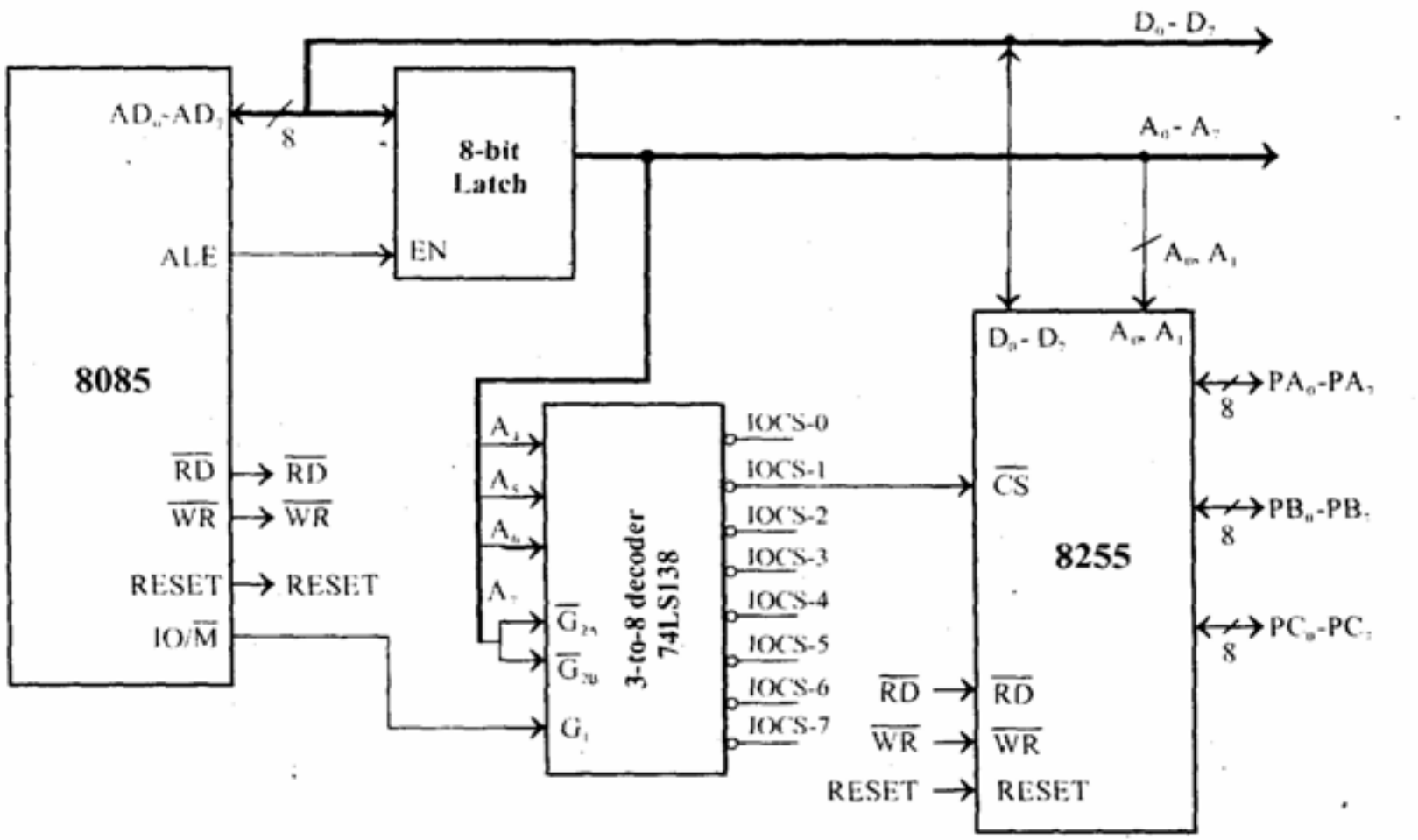
3. RESET: This is an active high signal. It clears the control register and set all ports in the input mode.

4. CS (low), A0 and A1: These are device select signals. They are:

Internal Devices	A_1	A_0
Port A	0	0
Port B	0	1
Port C	1	0
Control Register	1	1

Interfacing of 8255 with 8085 processor

A simple schematic for interfacing the 8255 with 8085 processor:



The 8255 can be either memory mapped or I/O mapped in the system. In the schematic shown in above is I/O mapped in the system.

Using a 3-to-8 decoder generates the chip select signals for I/O mapped devices.

The address lines A4, A5 and A6 are decoded to generate eight chip select signals (IOCS-0 to IOCS-7) and in this, the chip select IOCS- 1 is used to select 8255.

The address line A7 and the control signal IO/M (low) are used as enable for the decoder.

The address line A0 of 8085 is connected to A0 of 8255 and A1 of 8085 is connected to A1 of 8255 to provide the internal addresses.

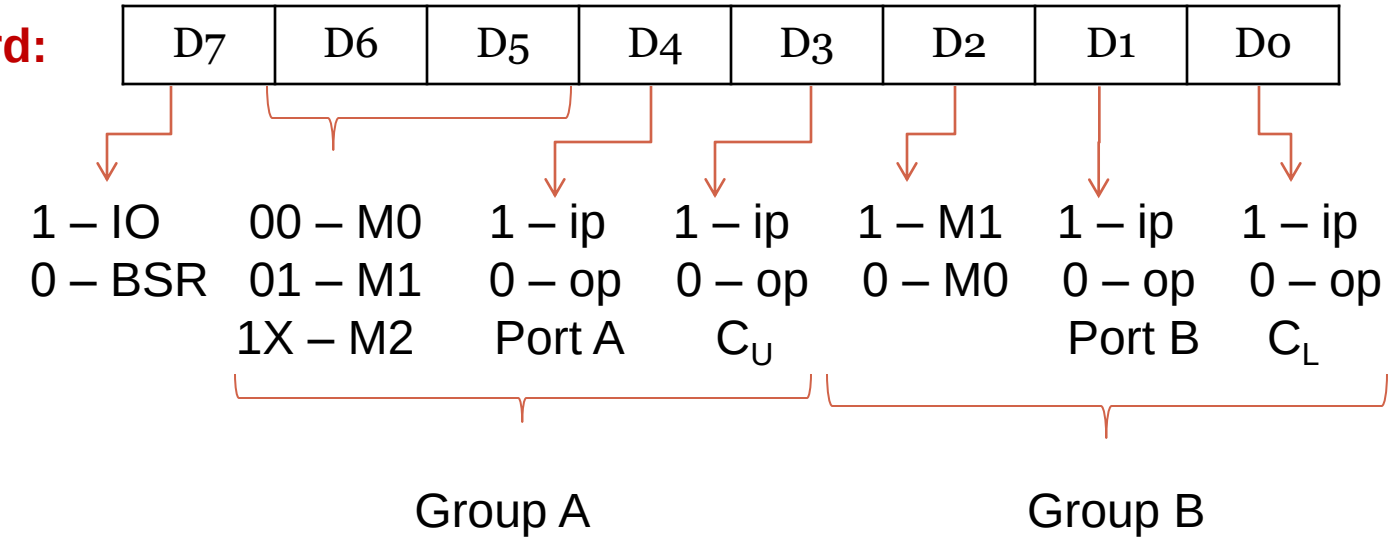
The data lines D0-D7 are connected to D0-D7 of the processor to achieve parallel data transfer.

The I/O addresses allotted to the internal devices of 8255 are listed in table.

Internal Device	Binary Address								Hexa Address
	Decoder input and enable				Input to address pins of 8255				
	A_7	A_6	A_5	A_4	A_3	A_2	A_1	A_0	
Port-A	0	0	0	1	x	x	0	0	10
Port-B	0	0	0	1	x	x	0	1	11
Port-C	0	0	0	1	x	x	1	0	12
Control Register	0	0	0	1	x	x	1	1	13

Note : *Don't care "x" is considered as zero.*

Control Word:



Example programming

Example: (i) Read from Port B and display at Port A in Mode 0
(ii) Read from Port C_L and display at Port C_U

Control word: 1 0 0 0 0 0 1 1 = 83H

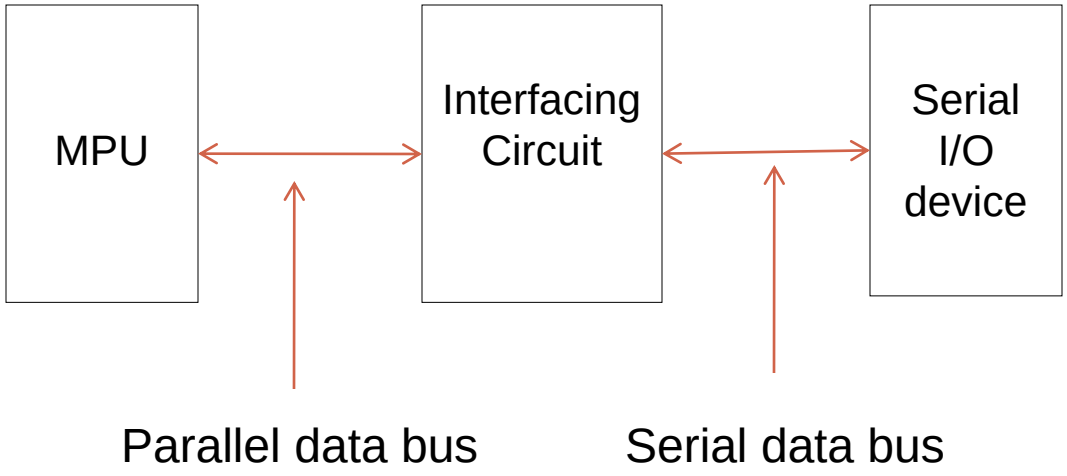
```
(i)     MVI     A, 83
        OUT     13H
        IN      11H
        OUT     10H

(ii)    IN      12H
        ANI     0FH
        RLC
        RLC
        RLC
        RLC
        OUT     12H
        RST     1
```

Serial I/O data Transfer

Basic concepts:

- Transfers in the form of ASCII codes.
- Only one transmission line, so transfer one bit at a time.



Transmission format:

a) Synchronous:

1) Data transferred with synchronous bits.
2) Have common clock.
3) High speed.

b) Asynchronous:

1) Data transferred with hand shaking signals.
2) Do not have common clock.
3) Low speed.

Communication lines may be:

1) Simplex: Only one direction.
2) Duplex:
   a) Half duplex: One way at any point of time.
   b) Full duplex: Both ways at any point of time.

Baud: bits/sec.
Bit Time: 1/Baud

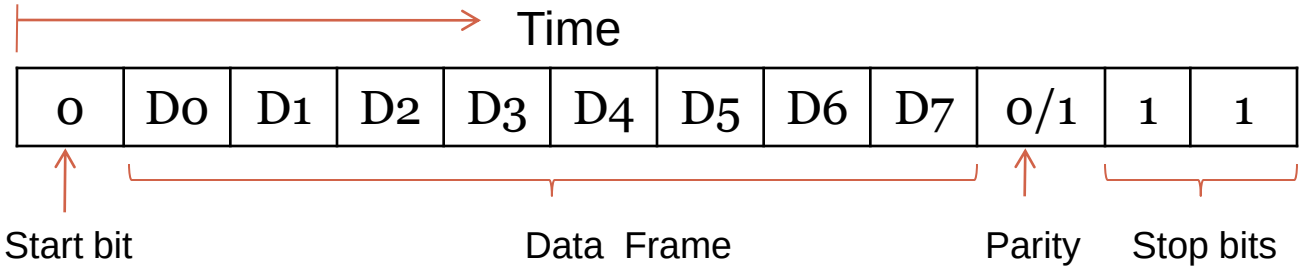
Error checking:

1) Parity Check: To correct one bit position
   a) Odd parity
   b) Even parity
2) Checksum: More than one bit positions
3) Cyclic redundancy Check (CRC)

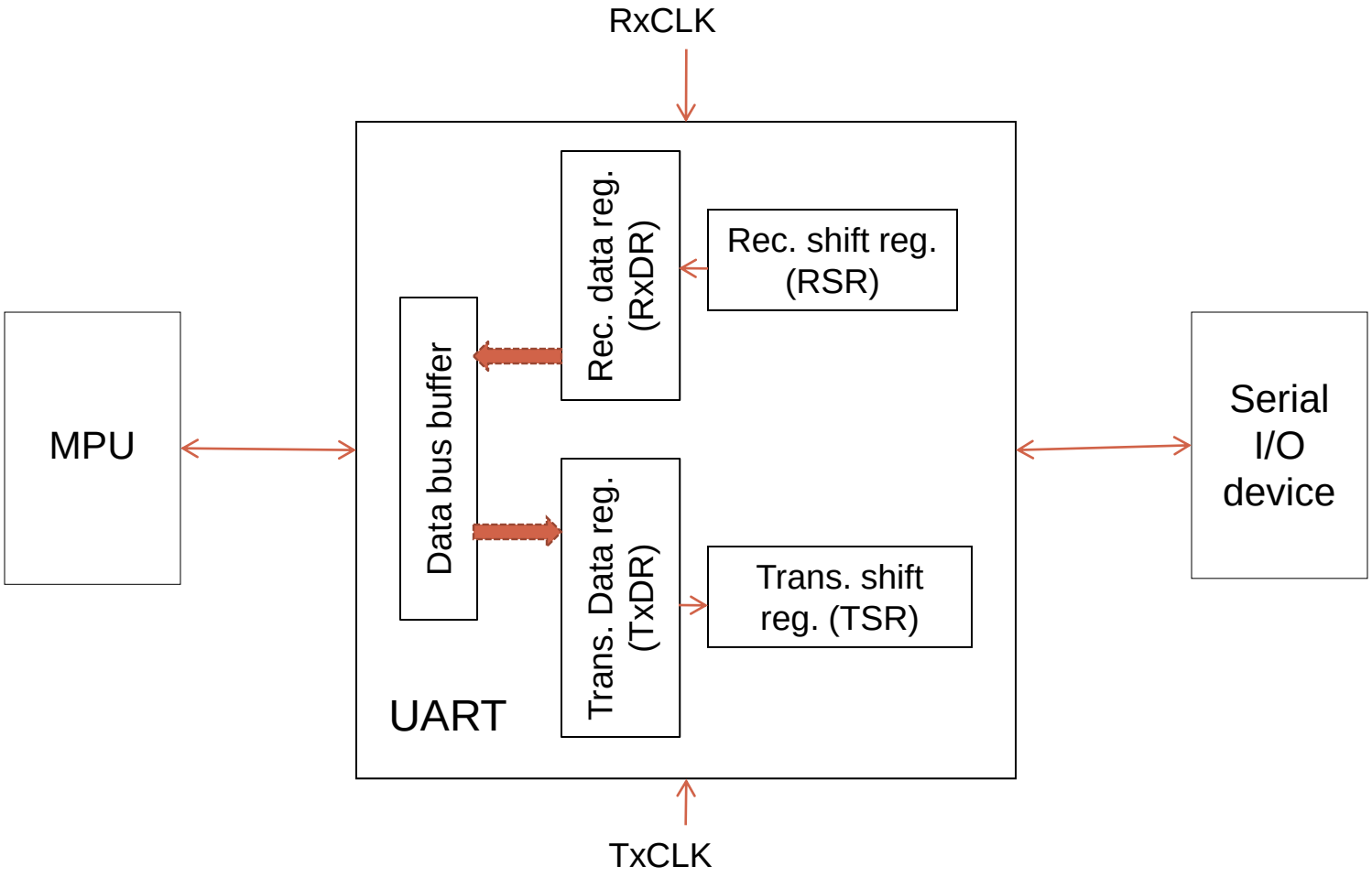
Asynchronous Data Transfer

Format used to transmit a single data word:

1) A *START* bit, which is always a logic 0.
2) 5-8 data bits, representing actual information.
3) An optional *parity* bit for error detection. Parity: *Odd* or *Even*.
4) 2 *STOP* bits, which are always logic 1.



Parallel/Serial Interface – The UART



Synchronizing the Receiver Section to the Serial Data

1) Initially, logic in receiver line is 1.
2) Receiver synchronizes itself on the negative-going transition by the START bit.
3) After this, the receiver waits for half-bit time and then samples the serial input bit.
4) Remaining bits are samples at bit-time intervals and store them in RSR.
5) Checks for errors.

MPU Transmitting

To transfer data to serial I/O device, MPU does following steps:

1) Before any communication, the MPU *initializes* the UART by writing a proper *control word* to UART's control register. This serve to set up the UART for proper *baud rate*, no. of *data bits*, *interrupts*, *parity* and no. of *STOP* bits.
2) The MPU reads the status register of UART and checks if Transmitter Data Register Empty (TDRE) is 1.
3) If TDRE = 1, the MPU writes a data word into the TxRD register to be eventually transmitted serially to the serial I/O device.
4) Repeats steps 2-3 until the MPU has written all its data into the output device.

MPU Receiving

To receive a data word from UART the MPU does the following steps:

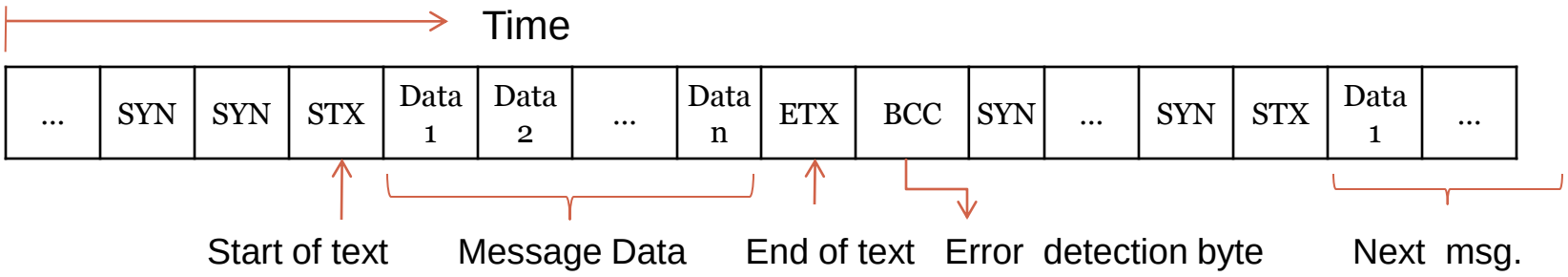
The MPU should initialize the UART with interrupt capability

1) Whenever the UART has a data ready (RDRF = 1), its lowers the IRQ (L) signal to interrupt the MPU.
2) In response to interrupt, the MPU reads the status register of UART to check errors.
3) If any error occurs, the MPU branches to error subroutine. Otherwise, it reads the data word from the RxDR register into the accumulator. The MPU returns to the program it was executing prior to interrupt.
4) For next data, steps 1-3 are repeated.

Synchronous Serial Data Transmission

1) In synchronous serial data transmission, individual data words are transmitted continuously one after another without inserting any START or STOP bits.
2) Transmitter sends repeatedly a special character *Sync Character (SYN) (at least 2 SYN)* before any data block to synchronize the receiver.
3) The receiver uses SYN character to synchronize its internal clock with that of the transmitter.
4) When transmitter wants to send data block, it stops sending SYN characters and sends data block one after another.
5) In response, after receiving last SYN character, the receiver will treat every subsequent 8-bits as a data and place them in a shift register and convert them into parallel data that can be read by MPU.

Message Format



Parallel I/O Interface Chips

Capabilities:

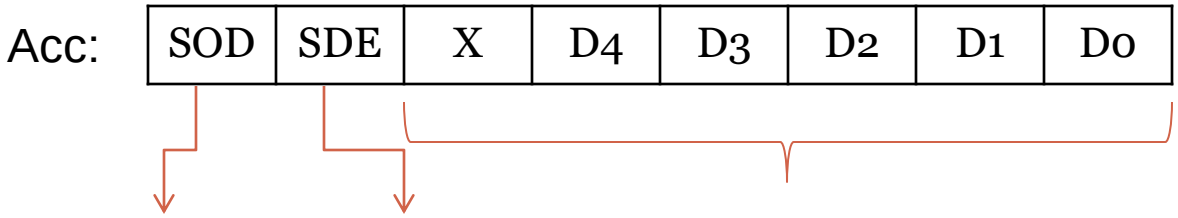
1) Programmability:
2) Handshaking:
3) Interrupt Control:
4) Timers:

Peripheral Interface Adapter (PIA)

1) Two independent 8-bit bidirectional ports to connect peripheral devices.
2) Data direction registers to specify Input/Output port.
3) Control registers.
4) Output Registers to hold data.
5) Some other registers

8085 – Serial I/O Lines

1) SOD (Serial Output Data): Used by SIM Instruction
2) SID (serial Input Data): Used by RIM Instruction

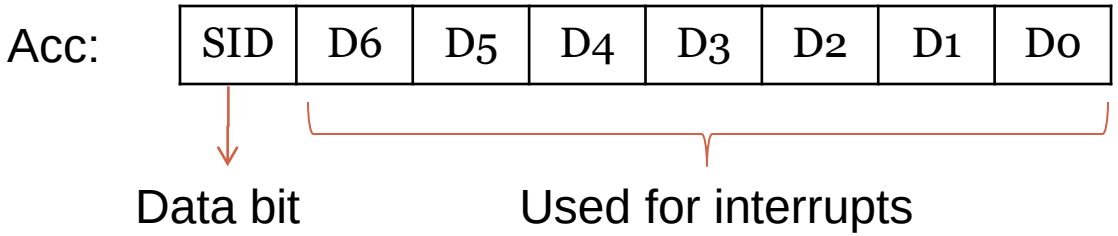


Data bit = 1 (Enable SOD line) Used for interrupts
= 0 (Disable SOD line)

Example:

```
MVI     A       80H
RAR
SIM
```

SID line & RIM instruction:



Data transfer using SOD & SID lines

Example: Subroutine to transfer an ASCII character stored in Register B using SOD line.

```
L1:     MVI     C       0B_H    ; C = 11 bits
        XRA     A               ; clear CY & ACC = 0
        MVI     A       80_H    ; D7 = 1
L2:     RAR                     ; D7 = CY, D6 = 1, CY = D0
        SIM                     ; output D7 (start bit)
        CALL    BIT_TIME        ; wait for bit time
        STC                     ; CY = 1
        MOV     A       B       ; A = B
        RAR                     ; D7 = 1, CY = D0
        MOV     B       A       ; save next bit
        DCR     C               ;
        JNZ     L2
        RET
```

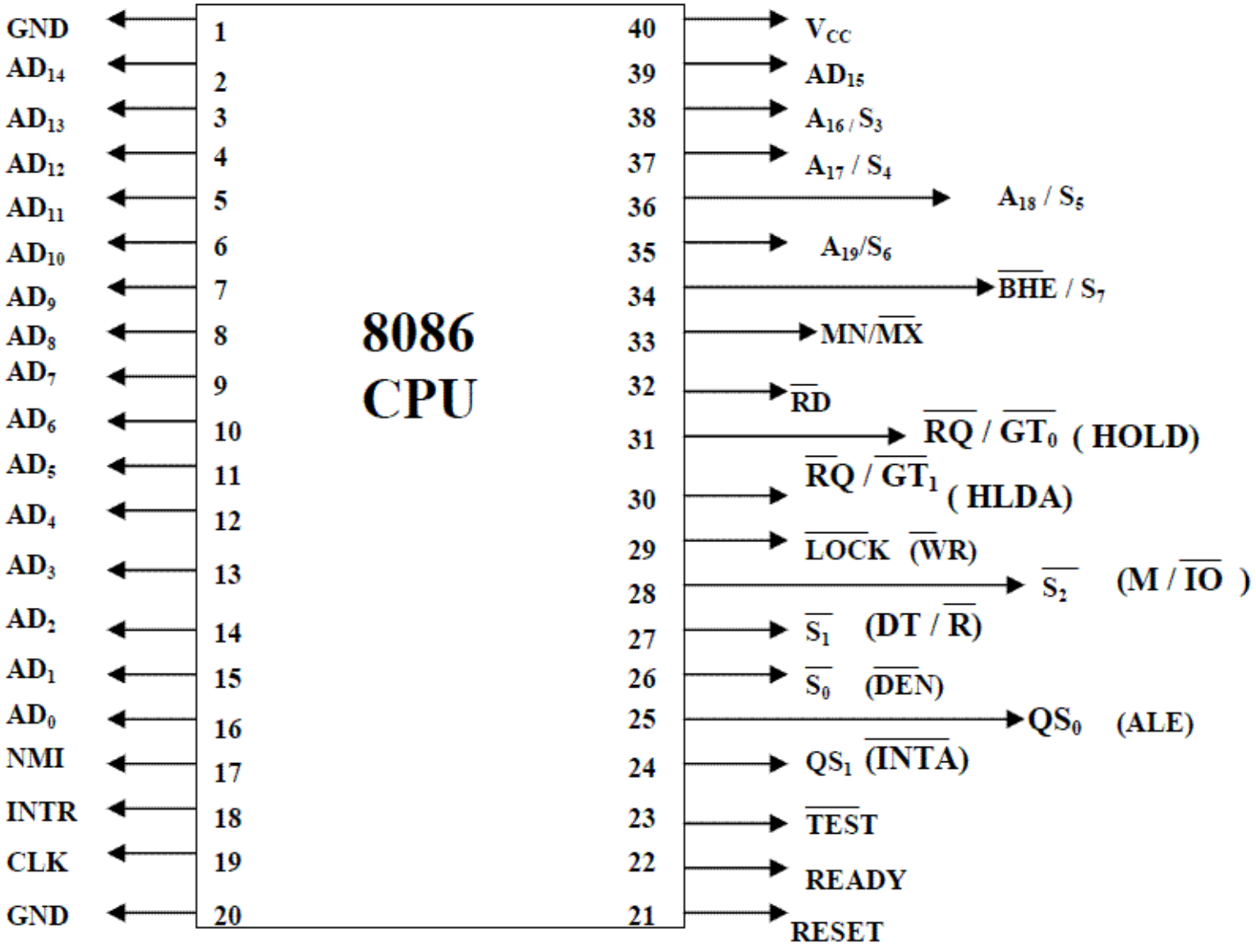
Read using SID line

```
L1:     RIM
        RAL
        JC      L1
        CALL    H_BIT
        MVI     B       00H
        MVI     C       08H
L2:     CALL    BIT_TIME
        RIM
        RAL
        MOV     A       B
        RAR
        MOV     B       A
        DCR     C
        JNZ     L2
        RET
```

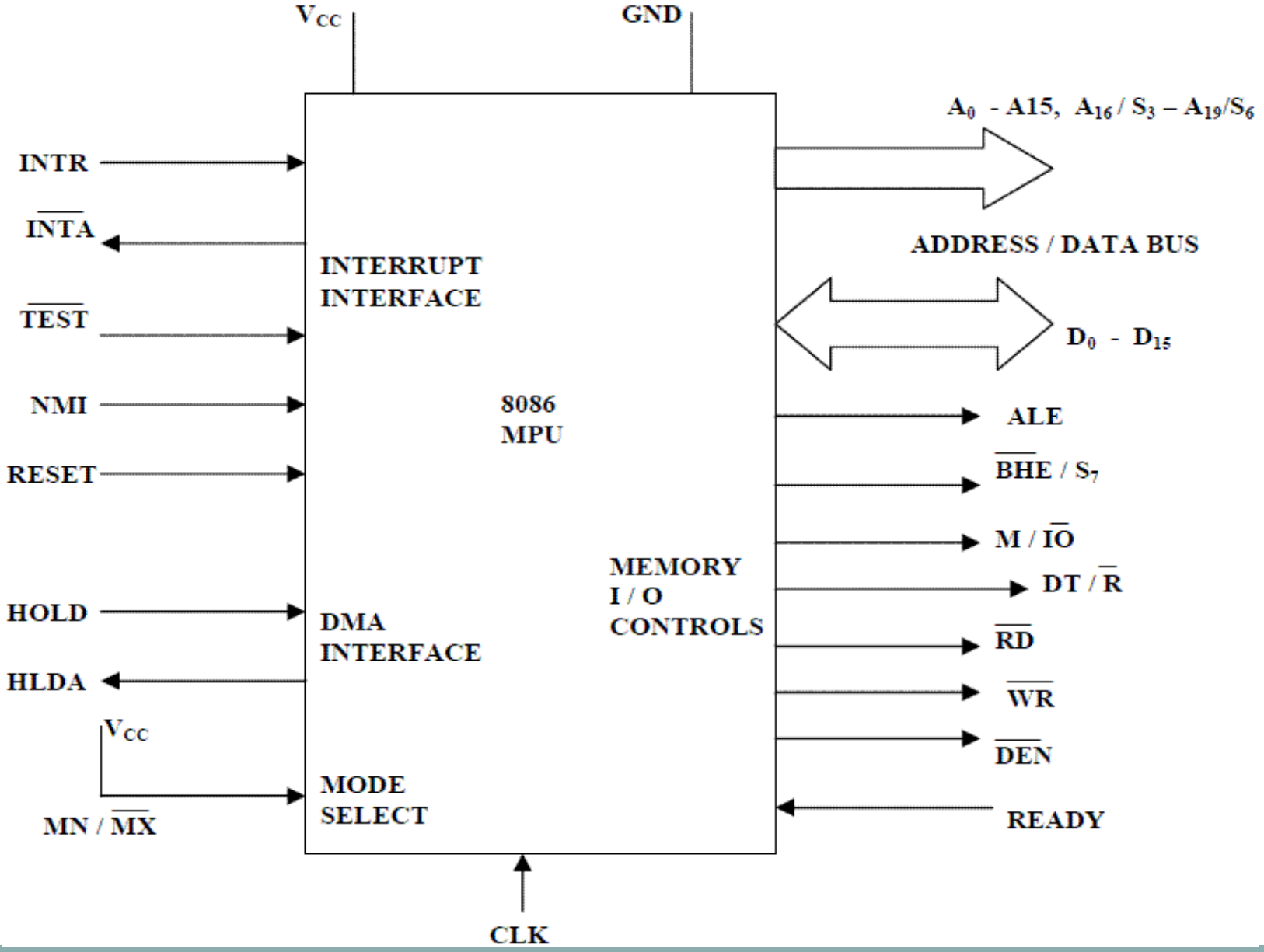
Overview or Features of 8086

1) It is a 16-bit Microprocessor (μp). It's ALU, internal registers works with 16-bit binary word.
2) 8086 has a 20 bit address bus can access up to 2^{20}= 1 MB memory locations.
3) 8086 has a 16-bit data bus. It can read or write data to a memory/port either 16bits or 8 bit at a time.
4) It can support up to 64K I/O ports.
5) It provides 14, 16 -bit registers.
6) Frequency range of 8086 is 6-10 MHz
7) It has multiplexed address and data bus AD0- AD15 and A16 – A19.
8) It requires single phase clock with 33% duty cycle to provide internal timing.
9) It can prefetch upto 6 instruction bytes from memory and queues them in order to speed up instruction execution.
10) It requires +5V power supply.

Pin Diagram of 8086



Signal Groups of 8086



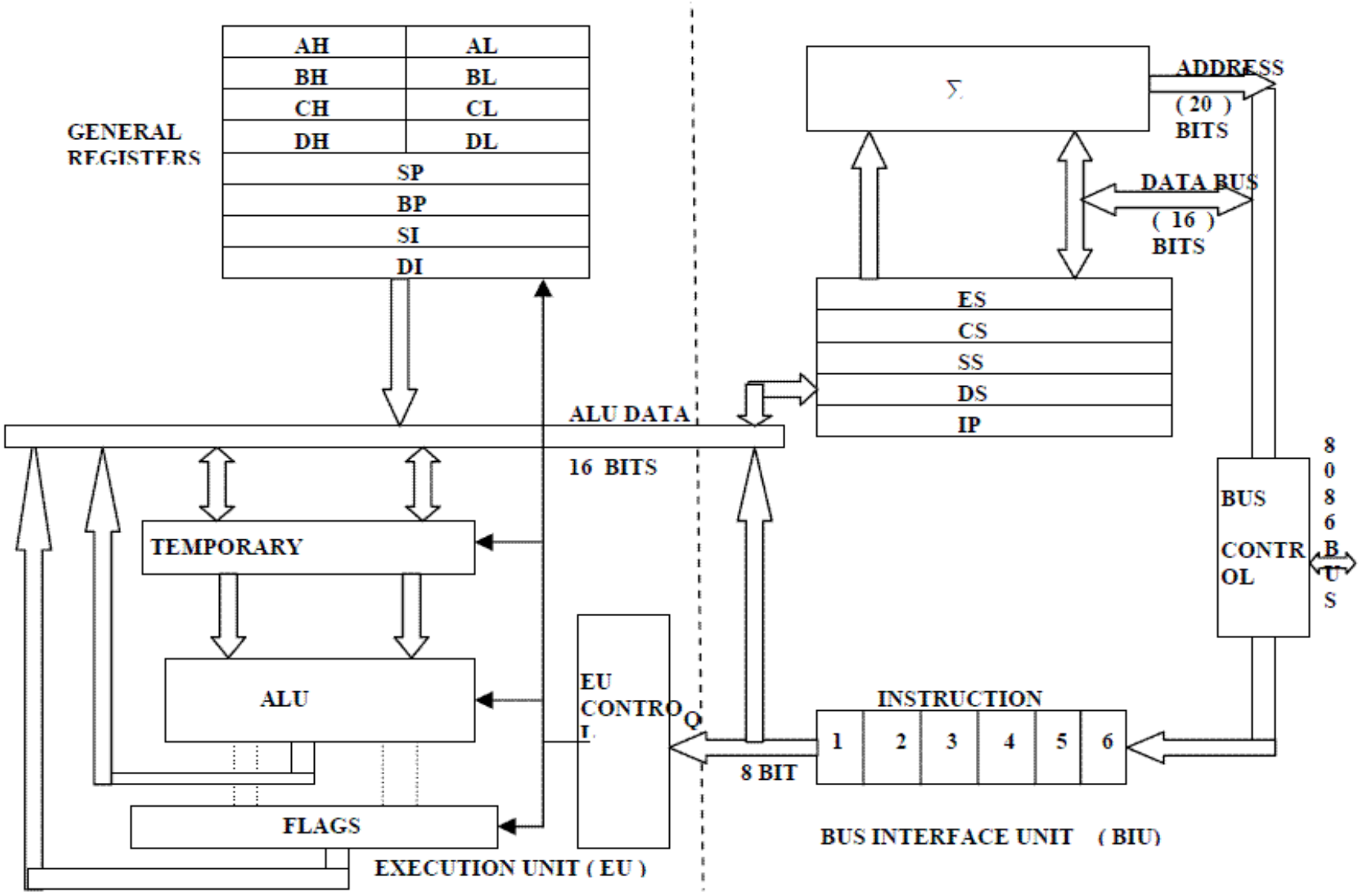
Architechture of 8086 or Functional Block diagram of 8086

8086 has two blocks (i) Bus Interfacing Unit (BIU) and (ii) Execution Unit (EU).

1) The BIU performs all bus operations such as instruction fetching, reading and writing operands for memory and calculating the addresses of the memory operands.
2) The instruction bytes are transferred to the instruction queue.
3) EU executes instructions from the instruction system byte queue.

1) Both units operate asynchronously to give the 8086 an overlapping instruction fetch and execution mechanism which is called as Pipelining. This results in efficient use of the system bus and system performance.
2) BIU contains Instruction queue, Segment registers, Instruction pointer, Address adder.
3) EU contains Control circuitry, Instruction decoder, ALU, Pointer and Index register, Flag register.

Block Diagram of 8086



BUS INTERFACE UNIT

- It provides a full 16 bit bidirectional data bus and 20 bit address bus.
- The bus interface unit is responsible for performing all external bus operations.
- Specifically it has the following functions:
- Instruction fetch, Instruction queuing, Operand fetch and storage, Address relocation and Bus control.

- The BIU uses a mechanism known as an instruction stream queue to implement a *pipeline architecture*.
- This queue permits prefetch of up to six bytes of instruction code. Whenever the queue of the BIU is not full, it has room for at least two more bytes and at the same time the EU is not requesting it to read or write operands from memory, the BIU is free to look ahead in the program by prefetching the next sequential instruction.
- These prefetching instructions are held in its FIFO queue. With its 16 bit data bus, the BIU fetches two instruction bytes in a single memory cycle.
- After a byte is loaded at the input end of the queue, it automatically shifts up through the FIFO to the empty location nearest the output.

- The EU accesses the queue from the output end. It reads one instruction byte after the other from the output of the queue. If the queue is full and the EU is not requesting access to operand in memory.
- These intervals of no bus activity, which may occur between bus cycles are known as Idle state.
- If the BIU is already in the process of fetching an instruction when the EU request it to read or write operands from memory or I/O, the BIU first completes the instruction fetch bus cycle before initiating the operand read / write cycle.

- The BIU also contains a dedicated adder which is used to generate the 20-bit physical address that is output on the address bus. This address is formed by adding an appended 16 bit segment address and a 16 bit offset address.
- For example: The physical address of the next instruction to be fetched is formed by combining the current contents of the code segment CS register and the current contents of the instruction pointer IP register.
- The BIU is also responsible for generating bus control signals such as those for memory read or write and I/O read or write.

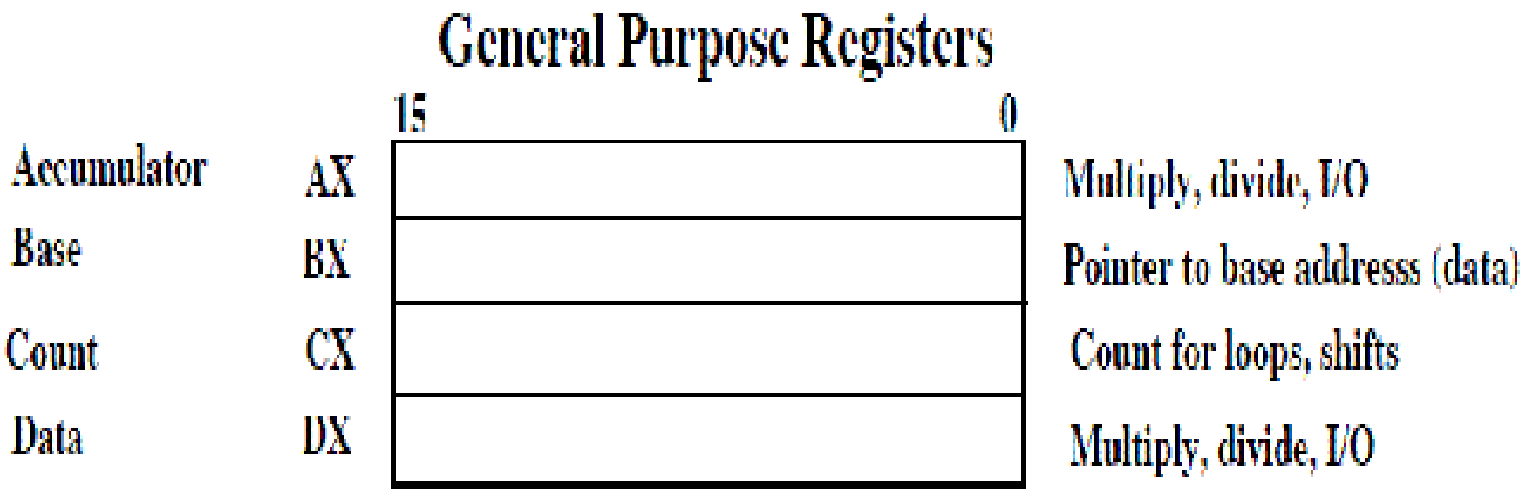
EXECUTION UNIT

- The Execution unit is responsible for decoding and executing all instructions.
- The EU extracts instructions from the top of the queue in the BIU, decodes them, generates operands if necessary, passes them to the BIU and requests it to perform the read or write bys cycles to memory or I/O and perform the operation specified by the instruction on the operands.
- During the execution of the instruction, the EU tests the status and control flags and updates them based on the results of executing the instruction.
- If the queue is empty, the EU waits for the next instruction byte to be fetched and shifted to top of the queue.
- When the EU executes a branch or jump instruction, it transfers control to a location corresponding to another set of sequential instructions.
- Whenever this happens, the BIU automatically resets the queue and then begins to fetch instructions from this new location to refill the queue.

The 8086 microprocessor has a total of fourteen registers that are accessible to the programmer. It is divided into four groups. They are:

1) Four General purpose registers
2) Four Index/Pointer registers
3) Four Segment registers
4) Two Other registers

General purpose registers :



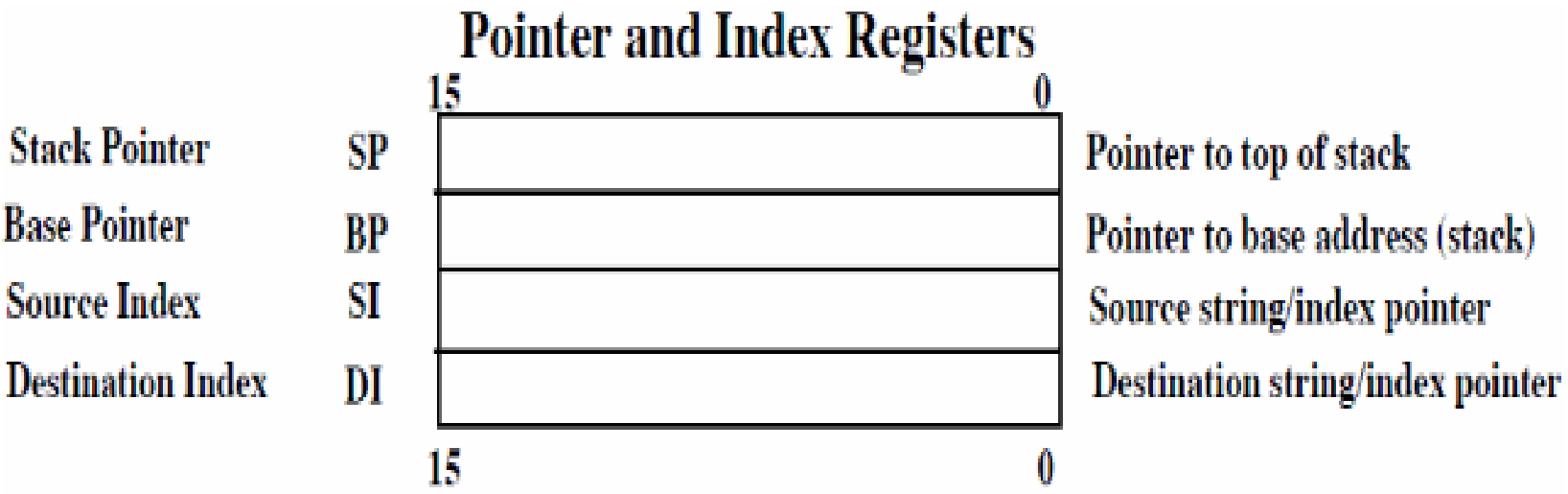
Accumulator register consists of two 8-bit registers AL and AH, which can be combined together and used as a 16-bit register AX. AL in this case contains the lower-order byte of the word, and AH contains the high-order byte. Accumulator can be used for I/O operations and string manipulation.

Base register consists of two 8-bit registers BL and BH, which can be combined together and used as a 16-bit register BX. BL in this case contains the low-order byte of the word, and BH contains the high-order byte. BX register usually contains a data pointer used for based, based indexed or register indirect addressing.

Count register consists of two 8-bit registers CL and CH, which can be combined together and used as a 16-bit register CX. When combined, CL register contains the lower-order byte of the word, and CH contains the high-order byte. Count register can be used in Loop, shift/rotate instructions and as a counter in string manipulation

Data register consists of two 8-bit registers DL and DH, which can be combined together and used as a 16-bit register DX. When combined, DL register contains the low order byte of the word, and DH contains the high-order byte. Data register can be used as a port number in I/O operations. In integer 32-bit multiply and divide instruction the DX register contains high-order word of the initial or resulting number.

These registers can also be called as Special Purpose registers.



Stack Pointer (SP) is a 16-bit register pointing to program stack, ie it is used to hold the address of the top of stack. The stack is maintained as a LIFO with its bottom at the start of the stack segment (specified by the SS segment register).Unlike the SP register, the BP can be used to specify the offset of other program segments.

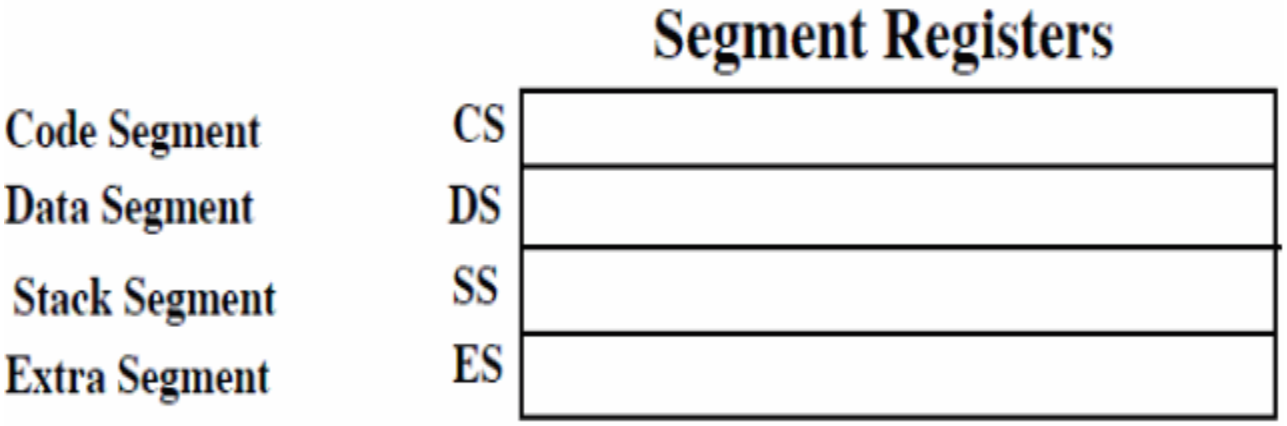
Base Pointer (BP) is a 16-bit register pointing to data in stack segment. It is usually used by subroutines to locate variables that were passed on the stack by a calling program. BP register is usually used for based, based indexed or register indirect addressing.

Source Index (SI) is a 16-bit register. SI is used for indexed, based indexed and register indirect addressing, as well as a source data address in string manipulation instructions. Used in conjunction with the DS register to point to data locations in the data segment.

Destination Index (DI) is a 16-bit register. Used in conjunction with the ES register in string operations. DI is used for indexed, based indexed and register indirect addressing, as well as a destination data address in string manipulation instructions. In short, Destination Index and SI Source Index registers are used to hold address.

Segment Registers

Most of the registers contain data/instruction offsets within 64 KB memory segment. There are four different 64 KB segments for instructions, stack, data and extra data. To specify where in 1 MB of processor memory these 4 segments are located the processor uses four segment registers.



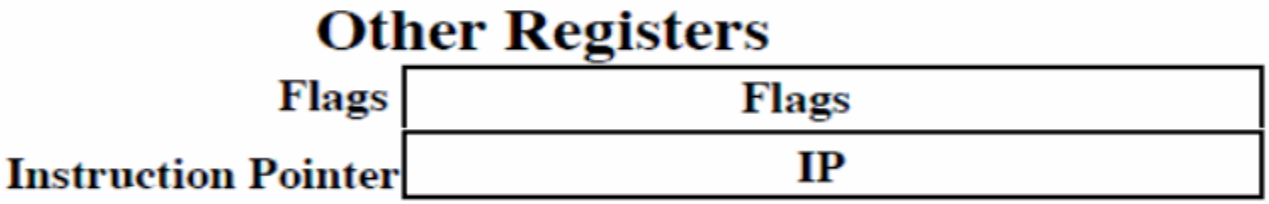
Code segment (CS) is a 16-bit register containing address of 64 KB segment with processor instructions. The processor uses CS segment for all accesses to instructions referenced by instruction pointer (IP) register. CS register cannot be changed directly. The CS register is automatically updated during far jump, far call and far return instructions.

Stack segment (SS) is a 16-bit register containing address of 64KB segment with program stack. By default, the processor assumes that all data referenced by the stack pointer (SP) and base pointer (BP) registers is located in the stack segment. SS register can be changed directly using POP instruction.

Data segment (DS) is a 16-bit register containing address of 64KB segment with program data. By default, the processor assumes that all data referenced by general registers (AX, BX, CX, DX) and index register (SI, DI) is located in the data segment. DS register can be changed directly using POP and LDS instructions.

Extra segment (ES) used to hold the starting address of Extra segment. Extra segment is provided for programs that need to access a second data segment. Segment registers cannot be used in arithmetic operations.

Other registers of 8086

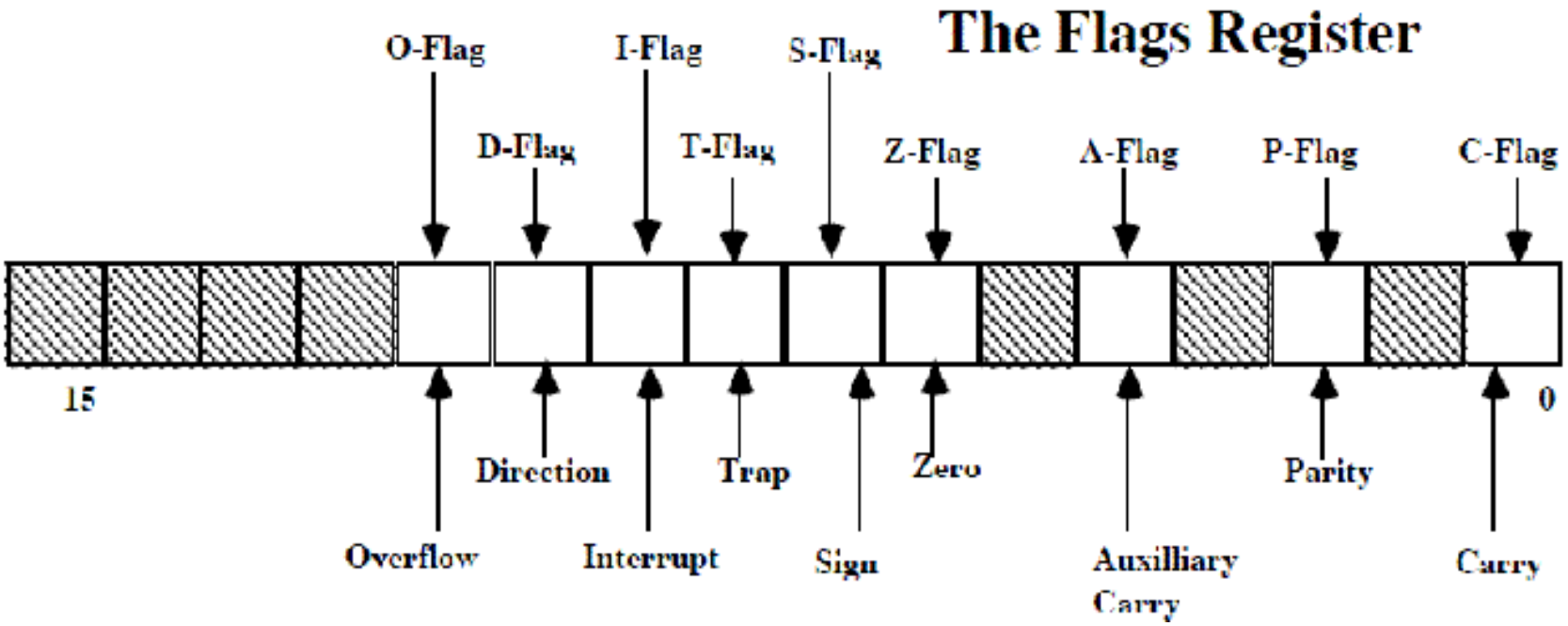


Instruction Pointer (IP) is a 16-bit register. This is a crucially important register which is used to control which instruction the CPU executes. The ip, or program counter, is used to store the memory location of the next instruction to be executed. The CPU checks the program counter to ascertain which instruction to carry out next. It then updates the program counter to point to the next instruction. Thus the program counter will always point to the next instruction to be executed.

Flag Register contains a group of status bits called flags that indicate the status of the CPU or the result of arithmetic operations. There are two types of flags:

1. The **status flags** which reflect the result of executing an instruction. The programmer cannot set/reset these flags directly.
2. The **control flags** enable or disable certain CPU operations. The programmer can set/reset these bits to control the CPU's operation.

Nine individual bits of the status register are used as control flags (3 of them) and status flags (6 of them).The remaining 7 are not used. A flag can only take on the values 0 and 1. We say a flag is set if it has the value 1.The status flags are used to record specific characteristics of arithmetic and of logical instructions.



Control Flags: There are three control flags

1. **The Direction Flag (D):** Affects the direction of moving data blocks by such instructions as MOVS, CMPS and SCAS. The flag values are 0 = up and 1 = down and can be set/reset by the STD (set D) and CLD (clear D) instructions.

2. **The Interrupt Flag (I):** Dictates whether or not system interrupts can occur. Interrupts are actions initiated by hardware block such as input devices that will interrupt the normal execution of programs. The flag values are 0 = disable interrupts or 1 = enable interrupts and can be manipulated by the CLI (clear I) and STI (set I) instructions.

3. **The Trap Flag (T):** Determines whether or not the CPU is halted after the execution of each instruction. When this flag is set (i.e. = 1), the programmer can single step through his program to debug any errors. When this flag = 0 this feature is off. This flag can be set by the INT 3 instruction.

Status Flags: There are six status flags

1. **The Carry Flag (C):** This flag is set when the result of an unsigned arithmetic operation is too large to fit in the destination register. This happens when there is an end carry in an addition operation or there an end borrows in a subtraction operation. A value of 1 = carry and 0 = no carry.

2. **The Overflow Flag (O):** This flag is set when the result of a signed arithmetic operation is too large to fit in the destination register (i.e. when an overflow occurs). Overflow can occur when adding two numbers with the same sign (i.e. both positive or both negative). A value of 1 = overflow and 0 = no overflow.

3. **The Sign Flag (S):** This flag is set when the result of an arithmetic or logic operation is negative. This flag is a copy of the MSB of the result (i.e. the sign bit). A value of 1 means negative and 0 = positive.

4. **The Zero Flag (Z):** This flag is set when the result of an arithmetic or logic operation is equal to zero. A value of 1 means the result is zero and a value of 0 means the result is not zero.

5. **The Auxiliary Carry Flag (A):** This flag is set when an operation causes a carry from bit 3 to bit 4 (or a borrow from bit 4 to bit 3) of an operand. A value of 1 = carry and 0 = no carry.

6. **The Parity Flag (P):** This flags reflects the number of 1s in the result of an operation. If the number of 1s is even its value = 1 and if the number of 1s is odd then its value = 0.

- CS + IP = Physical / Affective address
- SP + SS
- DS + SI
- ES + DI

Eg:

CS = 348A, IP = 4214

CS = 348A0
IP = 4214
38AB4 Physical address