Computer Science and Engineering

Subject Code : TCS 403 Course Title: Microprocessor

Course Outcomes:

After completion of the course students will be able to

1. Understanding of 8085 and 8086 microprocessors and memory segmentation
2. Analysis of Instruction set of 8085 and 8086.
3. Implementation of different programs on 8085 and 8086 based microcomputer kit.
4. Interfacing of 8255 and 8085/8086.
5. Interfacing of microprocessor with Timing Devices
6. This course will act as foundation for projects based on Embedded system and interfacing

DETAILED SYLLABUS

UNIT	CONTENTS	CONTACT HOURS
Unit – I	Introduction to Microprocessors: Evolution of Microprocessors, Classification-Brief Evolution. 8 Bit Processor: Example of an 8085 based System, Microprocessor Internal Architecture, hardware model of 8085, Pin diagram and function of each pin, memory interfacing.	9
Unit - II	Programming with 8085: Instruction set, programming model of 8085, addressing modes, assembly language programming, Timing and control, peripheral I/O, memory mapped I/O, 8085 Interrupts, Stack and subroutines.	10
Unit - III	16 Bit Processor: 16-bit Microprocessors (8086): Architecture, pin diagram, Physical address, segmentation, memory organization, Bus cycle, Addressing modes, Instruction set ,Assembly Language Programming of 8086, comparison of 8086 & 8088	8
Unit - IV	Interfacing (Data Transfer) with Microprocessor: Data Transfer Schemes: Introduction, handshaking signals, Types of transmission, 8255 (PPI), Serial Data transfer (USART 8251), memory interfacing, 8257 (DMA), programmable interrupt Controller (8259).	8
Unit - V	Interfacing of Microprocessor with Timing Devices: Programmable Interval Timer/ Counter (8253/8254): Introduction, modes, Interfacing of 8253, applications. Introduction to DAC & ADC, ADC & DAC Interfacing (0808, 0809)	9
	Total	44

Text Book: 1. Ramesh Gaonkar, “Microprocessor Architecture, Programming, and Applications with the 8085”, 5th Edition, Penram International Publication (India) Pvt. Ltd.

2. Douglas V. Hall, “Microprocessors and Interfacing”, 2nd Edition, TMH, 2006.

Reference Book: 1. Kenneth L. Short, “Microprocessors and programmed Logic”, 2nd Ed, Pearson Education Inc.
2. A.K.Ray&K.M.Bhurchandi, “Advanced Microprocessors and peripherals” , Tata McGraw Hill, 2000.2nd edition

Microprocessor

The **microprocessor** is a programmable integrated device that has computing and decision-making capability similar to that of the central processing unit(CPU) of the computer.

8085 is pronounced as "eighty-eighty-five" microprocessor.

The 8085 microprocessor is an 8-bit processor that includes on its chip most of the logic circuitry for performing computing tasks and for communicating with peripherals (Input / Output Devices).

Microprocessors are used to handle a set of tasks that control one or more external events or systems

Evolution Of Microprocessor 8085 :

Evolution of microprocessor 8085 is known as a central processing unit (CPU). It is a complete computation engine that is fabricated on a single chip. The first microprocessor was the Intel 4004, introduced in 1971. The 4004 is a 4 bit processor.

•The first microprocessor to make it into a home computer was the Intel 8080, a complete 8-bit computer on one chip, introduce in 1974

•In 1976, Intel updated the 8080 design with the 8085 by adding two instructions to enable / disable three added interrupt pins and the serial I/O pins

•. In 1978 Intel introduced the 8086, a 16-bit processor which gave rise to the X86 architecture. It did not contain floating point instructions.

•In 1980 the Intel released the 8087. It is the first math co-processor.

•The first microprocessor to make a real splash in the market was the Intel 8088, introduction in 1979 and incorporated into IBM Personal Computer.

•, The Personal Computer market moved from the 8088 to the 80286 to the 80386 to the 80486 to the Pentium to the Pentium H to the Pentium HI to the Pentium IV. All of these microprocessor are made by Intel and all of them are e improvement on the basic design of the 8088.

Evaluation of Microprocessors

Intel Microprocess				
Name	**Year**	**Transistors**	**Clock speed**	**Data width**
8080	1974	6,000	2 MHz	8 bits
8085	1976	6,500	5 MHz	8 bits
8086	1978	29,000	5 MHz	16 bits
8088	1979	29,000	5 MHz	8 bits
80286	1982	134,000	6 MHz	16 bits
80386	1985	275,000	16 MHz	32 bits
80486	1989	1,200,000	25 MHz	32 bits
Pentium	1993	3,100,000	60 MHz	32/64 bits
Pentium II	1997	7,500,000	233 MHz	64 bits
Pentium III	1999	9,500,000	450 MHz	64 bits
Pentium IV	2000	42,000,000	1.5 GHz	64 bits
Pentium IV "Prescott"	2004	125,000,000	3.6 GHz	64 bits
Intel Core 2	2006	291 million	3 GHz	64 bits
Pentium Dual Core	2007	167 million	2.93 GHz	64 bits
Intel 64 Nchalem	2009	781 million	3.33 GHz	64 bits

8085 Microprocessor

- Introduction
- Features
- Pin Configuration
- Architecture 0f 8085
- The 8085 Bus Structure
- Instruction Set
- Addressing Modes
- Timing diagrams
- References

Introduction

- Microprocessor is a Central Processing Unit (CPU) etched on a single chip. A single Integrated Circuit (IC) has all the functional components of a CPU namely Arithmetic Logic Unit (ALU), Control Unit and registers. The 8085 microprocessor is an 8-bit processor that includes on its chip most of the logic circuitry for performing computing tasks and for communicating with peripherals (Input and Output Devices).

Features

- 8 bit microprocessor(8085 microprocessor can read or write or perform arithmetic and logical operations on 8-bit data at time)
- It has 8 data lines and 16 address lines hence capacity is 2^{16} = 64 KB of memory
- Cock frequency is 3 MHz
- It requires +5V power supply.
- It is a single chip NMOS device implemented with 6200 transistors.
- It provides 74 instructions with five addressing modes.
- It provides 5 hardware interrupt and 8 software interrupts.

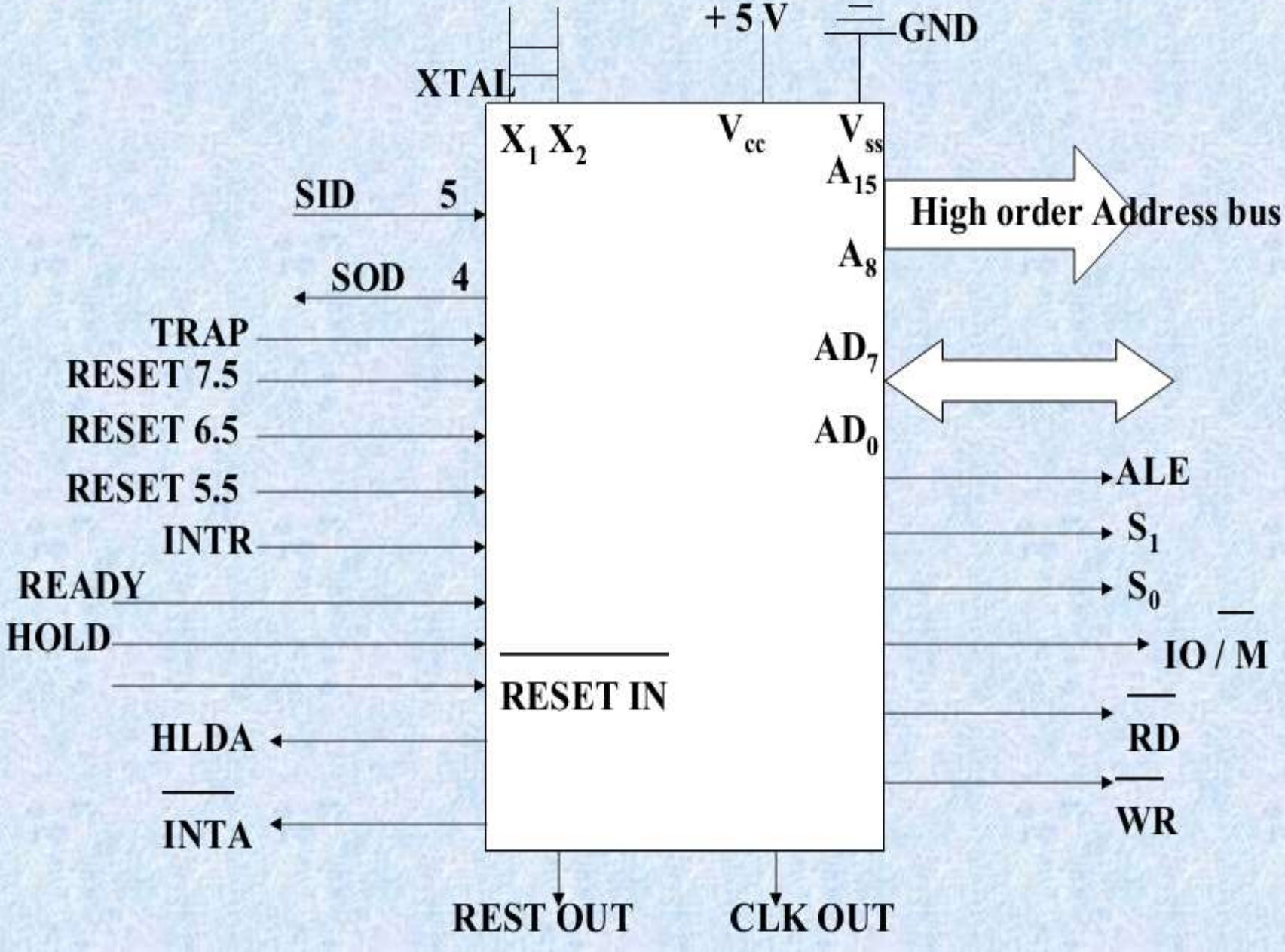
Pin Configuration

- 40 pins classified into

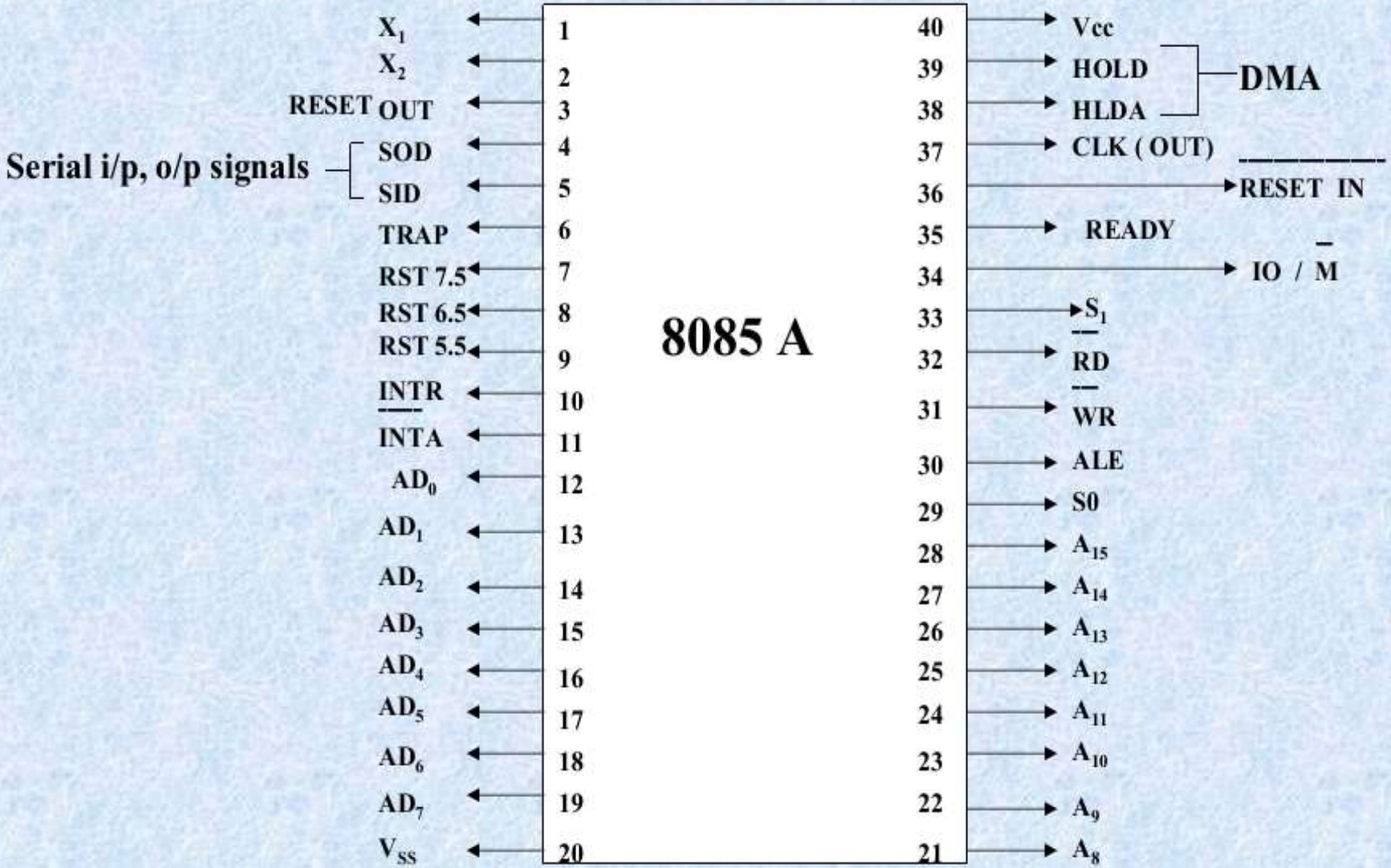
6 groups:

1. Data bus
2. Address bus
3. Control & status lines
4. Externally generated
5. Serial interface
6. Power supply & clock

Signal Groups of 8085



Pin Diagram of 8085



Pin Configuration cont…

1) Address Bus (A15-A8 and AD7-AD0):

The microprocessor 8085 has 16 bit address lines from A15-A8 and AD7-AD0. These lines are used to transfer 16 bit address of memory as well as 8-bit address of I/O ports.

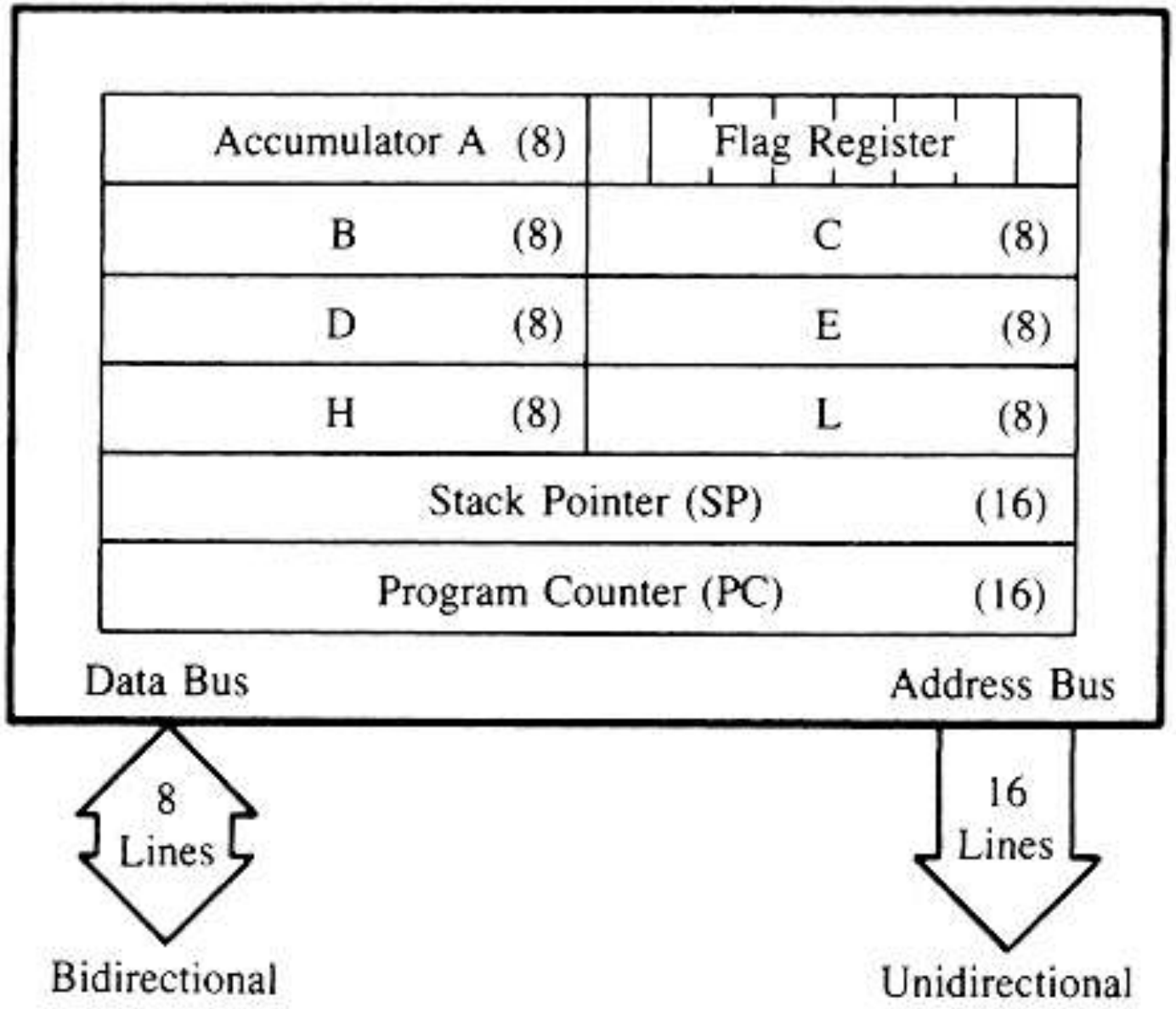
2) Data Bus:

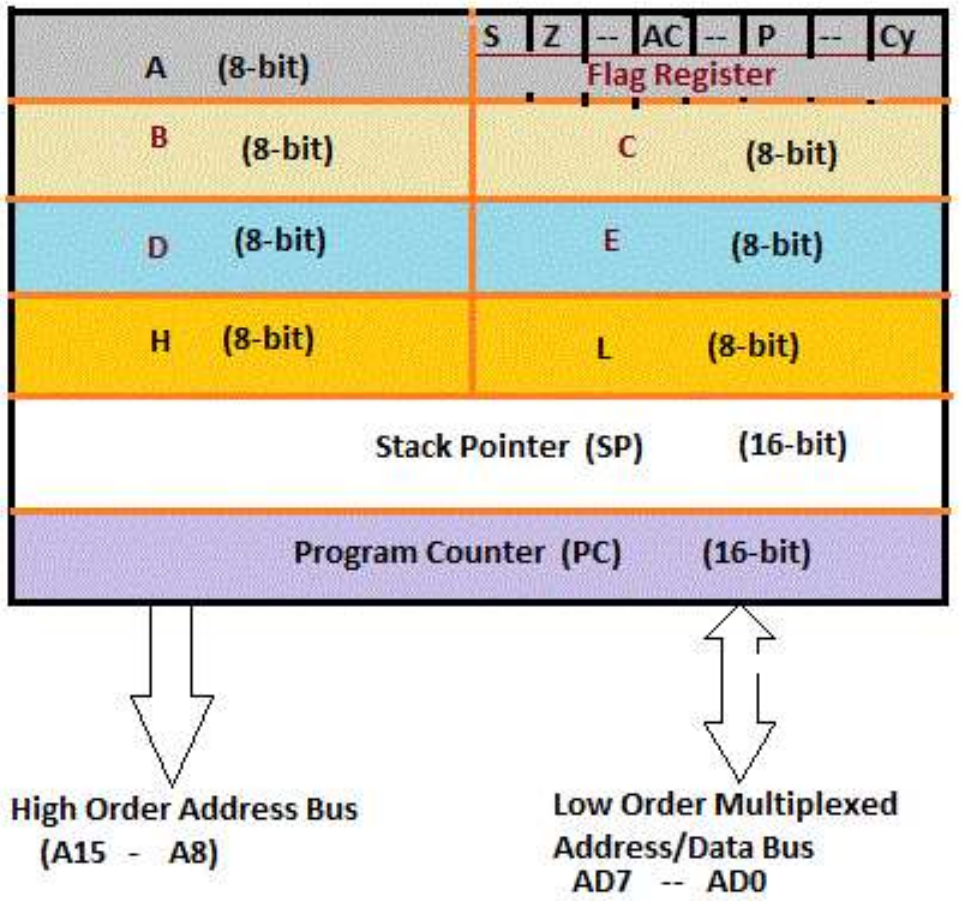
The lower 8 lines (AD7-AD0) are often called as multiplexed data lines.

Architecture 0f 8085

1. ALU
2. Timing and Control Unit
3. General Purpose Registers
4. Program Status word
5. Program Counter
6. Stack Pointer
7. Instruction Register and Decoder
8. Interrupt Control
9. Serial I/O Control
10. Address Bus
11. Data Bus

PROGRAMMING MODEL OF 8085





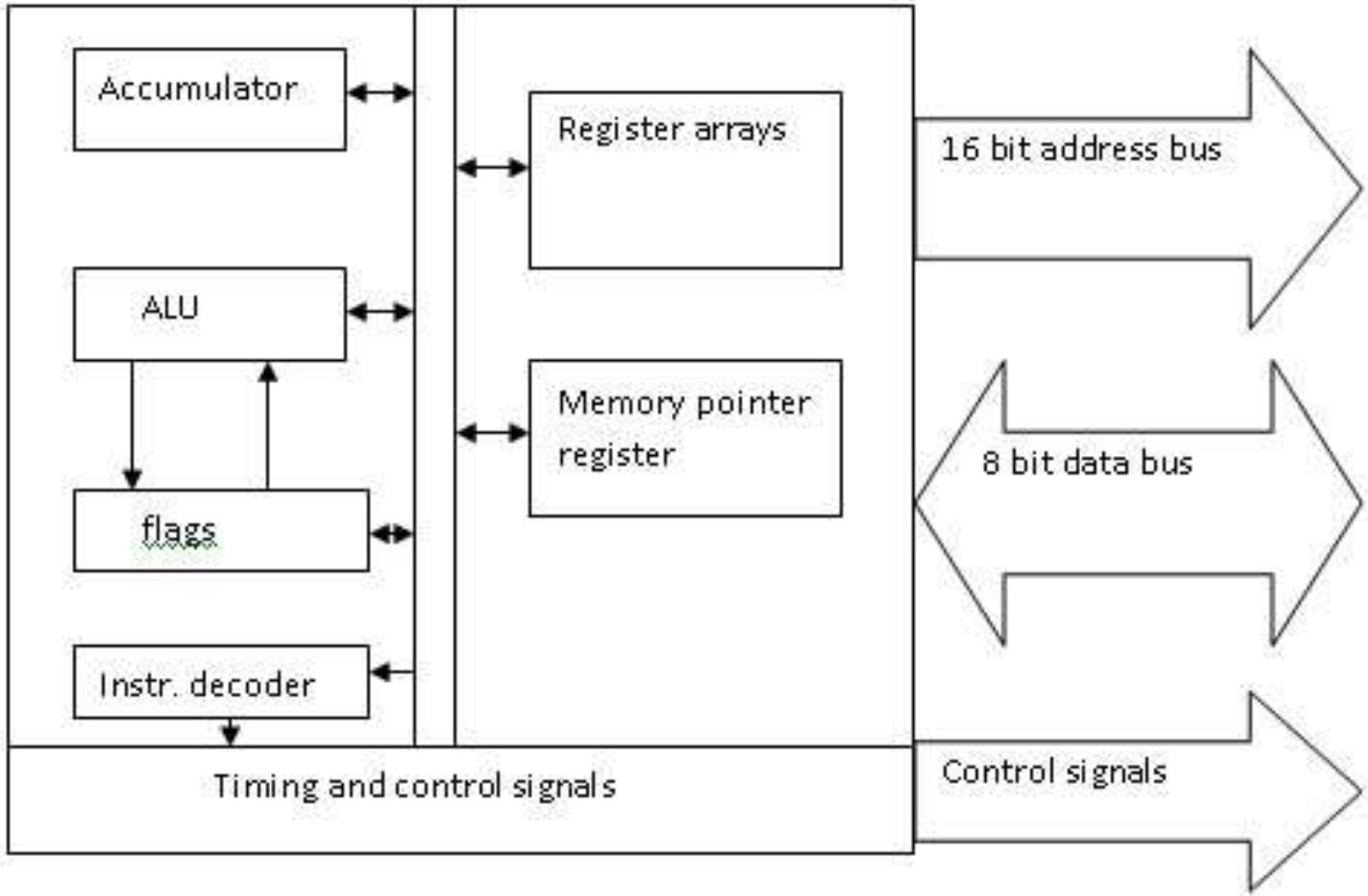
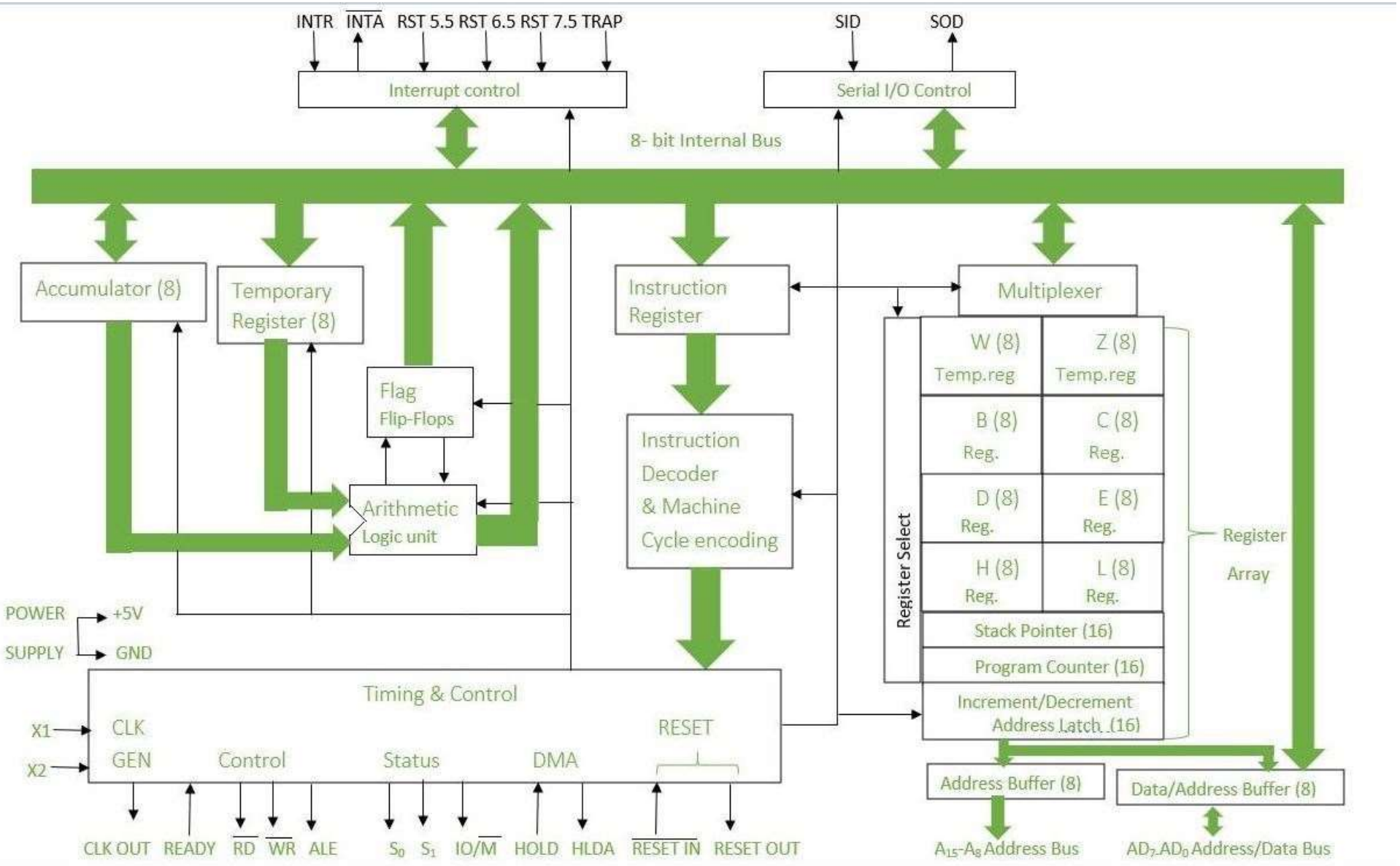


Fig. hardware model

8085 is an 8-bit, general purpose microprocessor. It consists of following functional units:-



ALU – Arithmetic & Logic Unit

ALU of 8085 performs 8-bit arithmetic & logical operations. The operations are generally performed with Accumulator as one of the operands. The result is saved in accumulator register.

Timing & Control Unit

This unit works as the brain of the CPU and generates all the timing and control signals to perform all the internal & external operations of the CPU.

Instruction Decoder & Machine Cycle Encoder Unit

This unit decodes the op-code stored in the Instruction Register (IR) and encodes it for the timing & control unit to perform the execution of the instruction.

REGISTERS

The Registers are of 8-bit & 16-bit size used for different purposes

A- Accumulator – This is an special purpose register. All the ALU operations are performed with reference to the contents of Accumulator.

B,C,D,E,H,L – General purpose registers. These registers can also used for 16-bit operations in pairs. The default pairs are BC, DE & HL.

CONTROL LINES

RD : Read: This is active low signal which indicates that the selected I/O or memory device is to be read and also is available on the data bus.

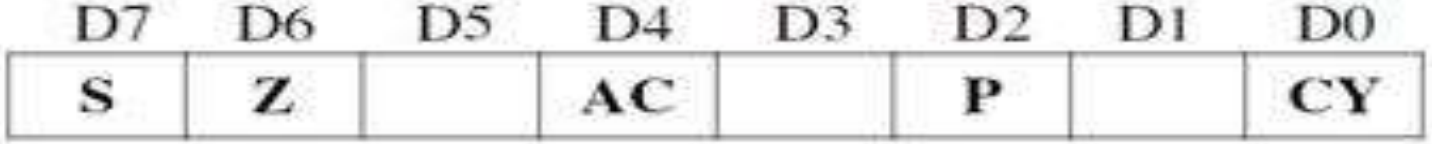
WR : Write: This is active low signal which indicates that the data on data bus are to be written into a selected memory location.

IO/ M : (Input / Output / Memory): This is used to select either Input / Output devices or memory operation. When it is high it indicates an I/O operation and when it is low, it indicates a memory operation.

STATUS LINES

- Status Pins (S1, S0): The microprocessor 8085 has two status pins as S1, S0 which is used to indicate the status of microprocessor or operation which is performed by microprocessor.

F – Flag register – This register indicates the status of the ALU operation.



D7	D6	D5	D4	D3	D2	D1	D0
S	Z		AC		P		CY

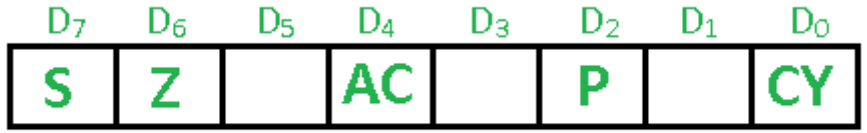
PC – Program Counter – This is a 16-bit register used to address the memory location from where an instruction is going to be executed.

SP – Stack pointer - This is a 16-bit register used to address the top of the stack memory location.

Flag register in 8085 microprocessor

The **Flag register** is a Special Purpose Register. Depending upon the value of result after any arithmetic and logical operation the flag bits become set (1) or reset (0). In 8085 microprocessor, flag register consists of 8 bits and only 5 of them are useful.

The 5 flags are:



Sign Flag (S) – After any operation if the MSB (B(7)) of the result is 1, it indicates the number is negative and the sign flag becomes set, i.e. 1. If the MSB is 0, it indicates the number is positive and the sign flag becomes reset i.e. 0.
from 00H to 7F, sign flag is 0
from 80H to FF, sign flag is 1
1- MSB is 1 (negative)
0- MSB is 0 (positive)

Example:

MVI A 30 (load 30H in register A)
MVI B 40 (load 40H in register B)
SUB B (A = A – B)
These set of instructions will set the sign flag to 1 as 30 – 40 is a negative number.

MVI A 40 (load 40H in register A)
MVI B 30 (load 30H in register B)
SUB B (A = A – B)
These set of instructions will reset the sign flag to 0 as 40 – 30 is a positive number.

Zero Flag (Z) – After any arithmetical or logical operation if the result is 0 (00)H, the zero flag becomes set i.e. 1, otherwise it becomes reset i.e. 0.
00H zero flag is 1.
from 01H to FFH zero flag is 01- zero result
0- non-zero result
Example:
MVI A 10 (load 10H in register A)
SUB A (A = A – A)
These set of instructions will set the zero flag to 1 as 10H – 10H is 00H

Auxiliary Carry Flag (AC) – This flag is used in BCD number system(0-9). If after any arithmetic or logical operation D(3) generates any carry and passes on to B(4) this flag becomes set i.e. 1, otherwise it becomes reset i.e. 0. This is the only flag register which is not accessible by the programmer1-carry out from bit 3 on addition or borrow into bit 3 on subtraction
0-otherwise
Example:
MOV A 2B (load 2BH in register A)
MOV B 39 (load 39H in register B)
ADD B (A = A + B)
These set of instructions will set the auxiliary carry flag to 1, as on adding 2B and 39, addition of lower order nibbles B and 9 will generate a carry.

Parity Flag (P) – If after any arithmetic or logical operation the result has even parity, an even number of 1 bits, the parity register becomes set i.e. 1, otherwise it becomes reset i.e. 0.1-accumulator has even number of 1 bits 0-accumulator has odd parity

Example:

MVI A 05 (load 05H in register A)

This instruction will set the parity flag to 1 as the BCD code of 05H is 00000101, which contains even number of ones i.e. 2.

Carry Flag (CY) – Carry is generated when performing n bit operations and the result is more than n bits, then this flag becomes set i.e. 1, otherwise it becomes reset i.e. 0.

During subtraction (A-B), if A>B it becomes reset and if (A<B) it becomes set.

Carry flag is also called borrow flag.1-carry out from MSB bit on addition or borrow into MSB bit on subtraction 0-no carry out or borrow into MSB bit

Example:

MVI A 30 (load 30H in register A)

MVI B 40 (load 40H in register B)

SUB B (A = A – B)

These set of instructions will set the carry flag to 1 as 30 – 40 generates a carry/borrow.

MVI A 40 (load 40H in register A)

MVI B 30 (load 30H in register B)

SUB B (A = A – B)

These set of instructions will reset the sign flag to 0 as 40 – 30 does not generate any carry/borrow.

What are buses? Why do we need buses in microprocessors?

In the world of computers and microprocessors, a bus is a connection between various components. These connections are meant for transfer of data among the components such as from CPU to memory or from CPU to a peripheral output device.

Buses can be of two types: **Address bus, Data bus and Control bus**

1.Address bus

The address bus is used to specify the address of a location in the memory or the address of an I/O device from which data transfer is to be made. It connects the CPU to the memory, input devices, and output devices. Note that in the above illustration, the address bus terminates at each of the above-mentioned components, pointing towards them. This tells us that the address bus is unidirectional. The address is specified by CPU and can only be directed from the CPU towards other devices and not the reverse.

2. Data bus

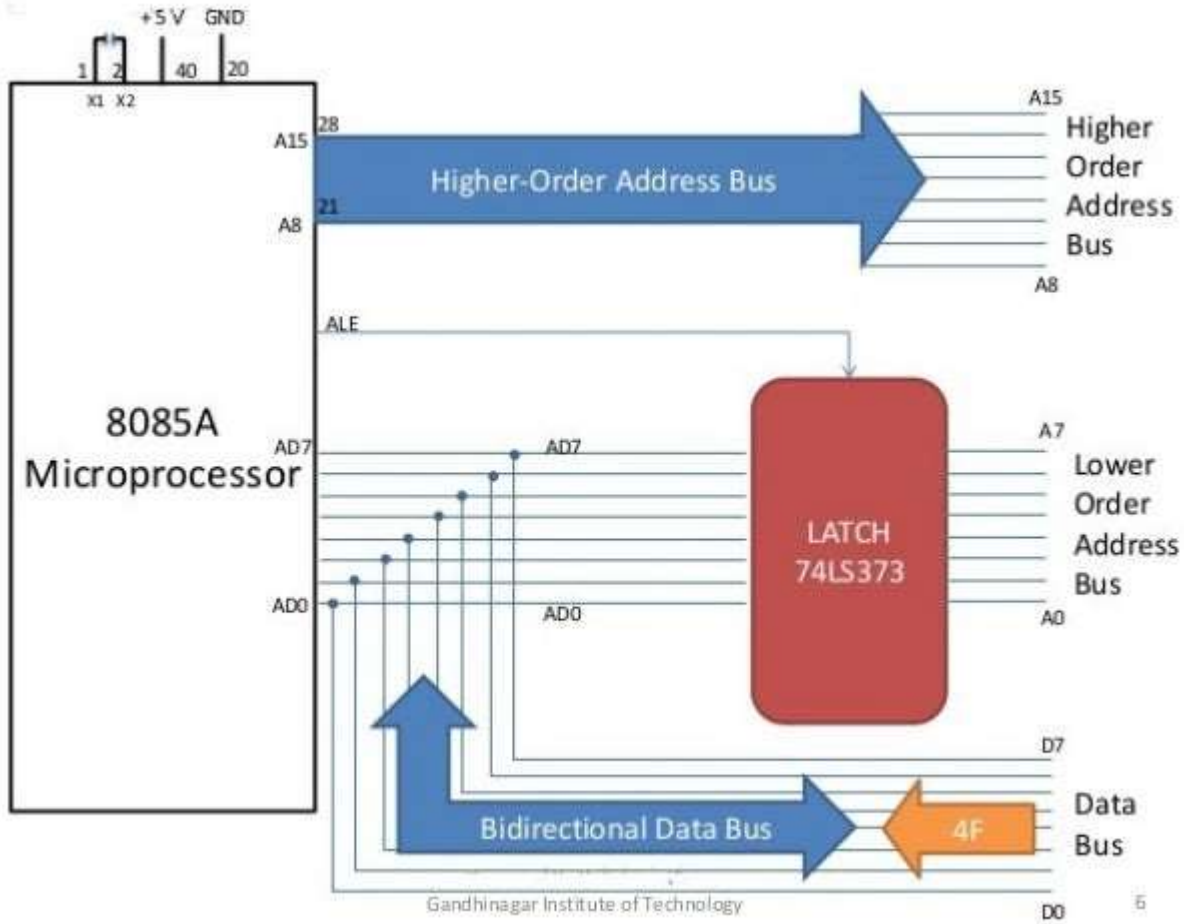
The data bus, as the name suggests, is meant for the transfer of data. 8085 is an 8-bit microprocessor. So, the data bus is 8 bits wide. We will learn in detail what that means.

SPECIAL SIGNAL

ALE (Address Latch Enable): The ALE signal is used to enable or disable the external latch IC (74373/8212).

The external latch IC is used for the de-multiplexing of AD7-AD0 lines, i.e., it is used to separate the address and data from AD7-AD0 lines.

- If ALE = 1/0 then external latch IC is enabled / disabled respectively.


+5 V
GND
1
2
40
20
X1
X2
8085A
Microprocessor
28
A15
21
A8
Higher-Order Address Bus
A15
Higher
Order
Address
Bus
A8
ALE
AD7
AD0
AD7
AD0
LATCH
74LS373
A7
Lower
Order
Address
Bus
A0
D7
Bidirectional Data Bus
4F
Data
Bus
D0

8085 Microprocessor Architecture – Demultiplexing the AD7-AD0

- Schematic diagram to latch low order address bus.

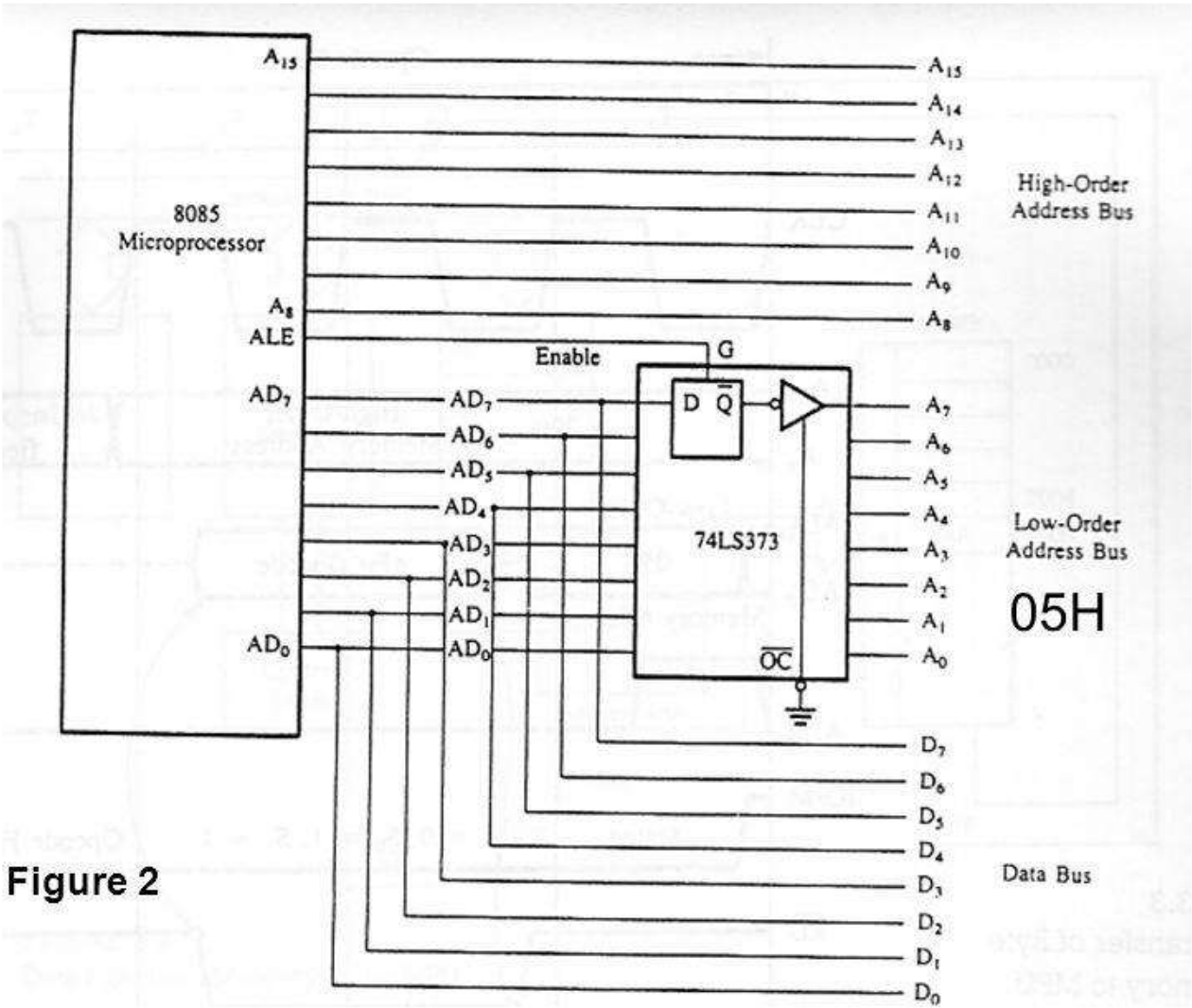


Figure 2

Control Signals of 8085:

The 8085 Microprocessor provides RD and WR signals to initiate read or write cycle. Because these Control Signals of 8085 are used both for reading/writing memory and for reading/writing an input device, it is necessary to generate separate read and write signals for memory and I/O devices.

To deal with different I/O as well as memory device individually, we have to generate four individual control signals. This control signals used to select any of the I/O or memory device, with a specific type of operation either of read or write. In our case, we are interested with two operation with memory as well as output devices. First of all, we required to fetch the instructions place inside the memory and next we write data word on output port.

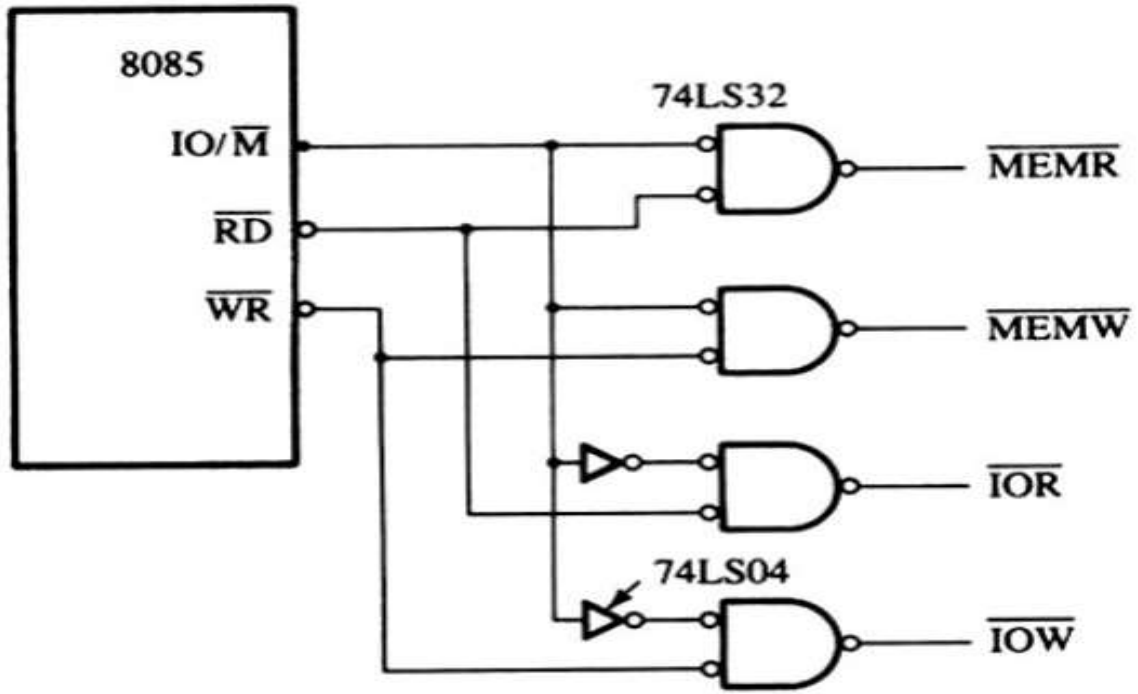
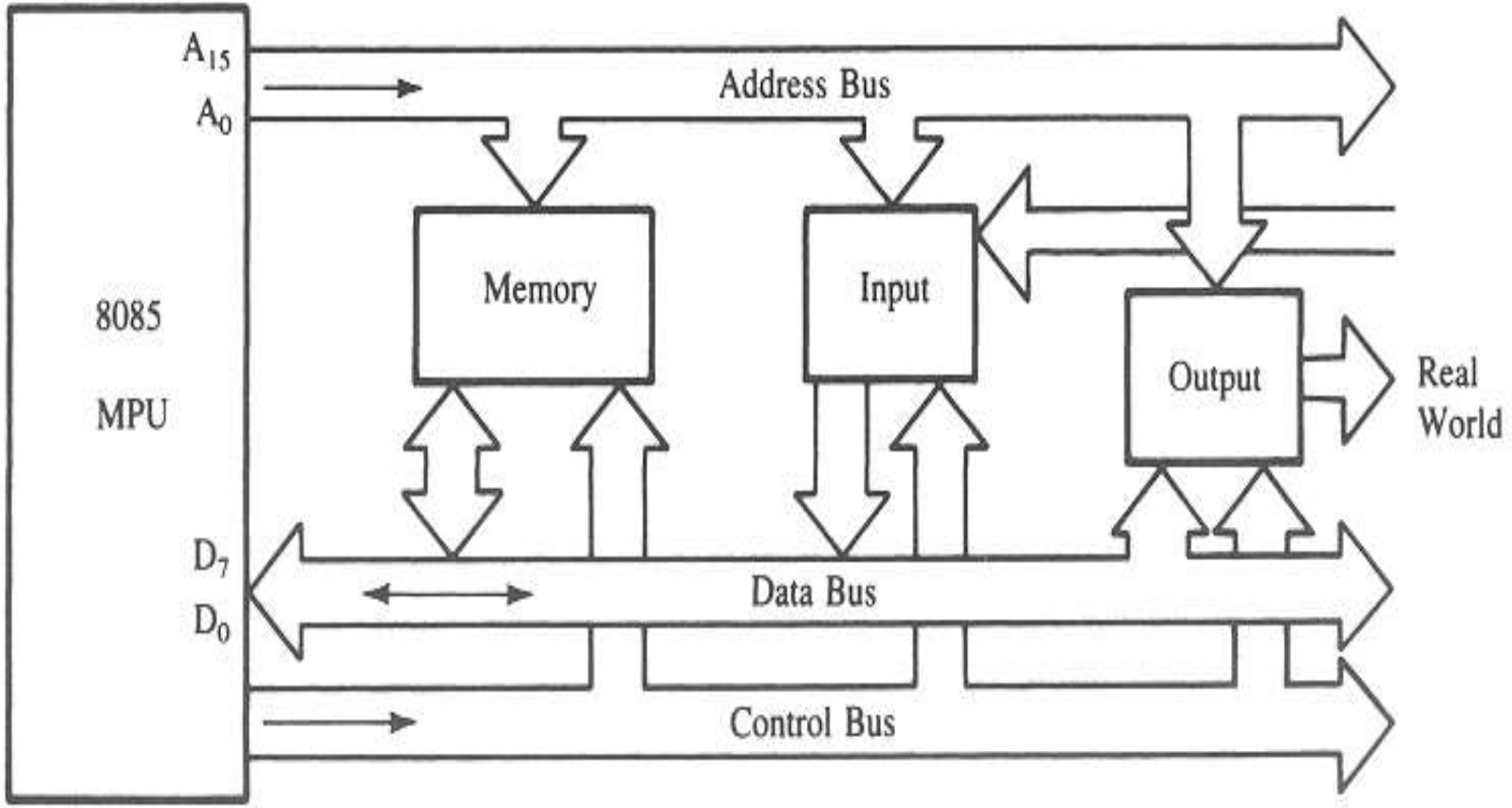


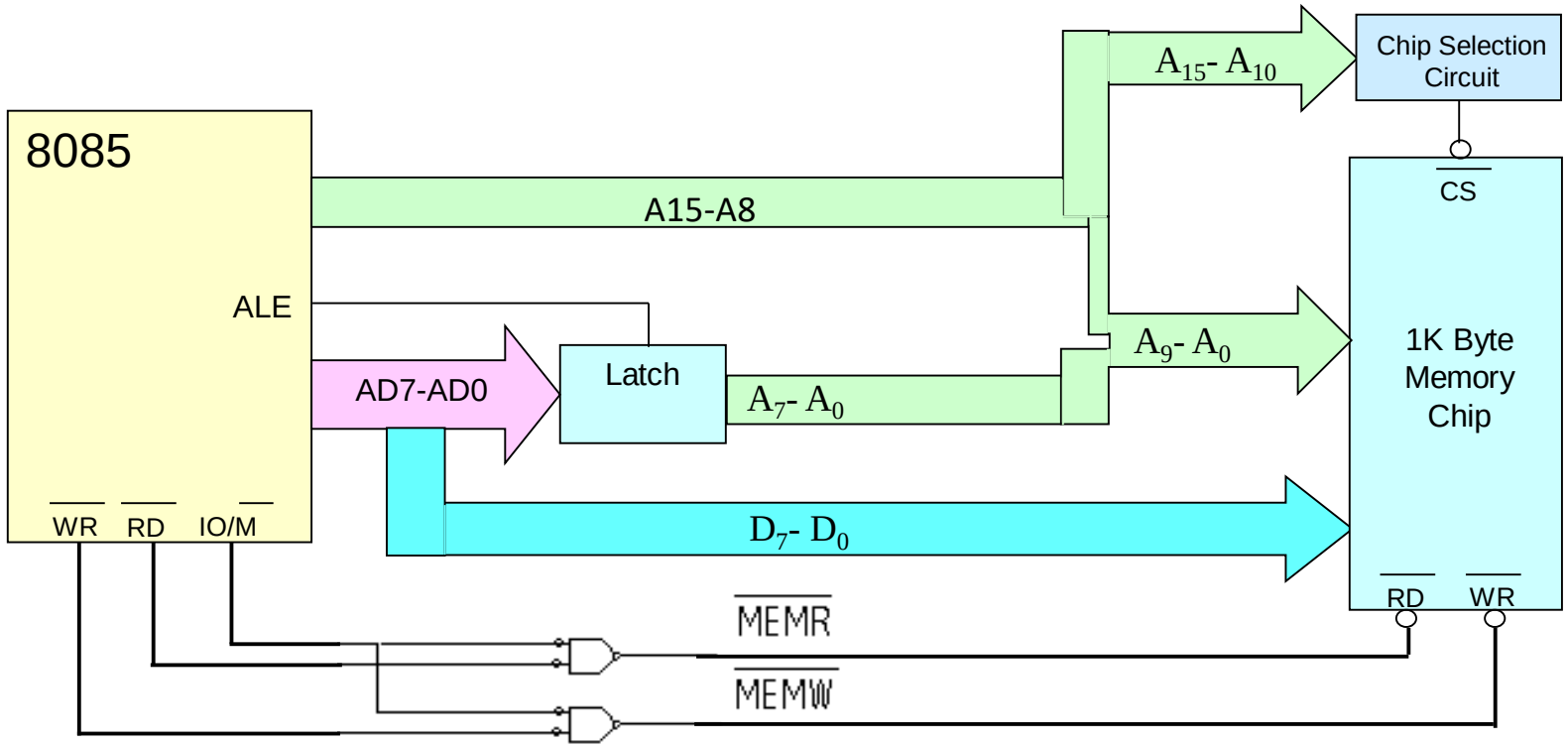
Figure: - Example of the schematic Diagram to generate control signals

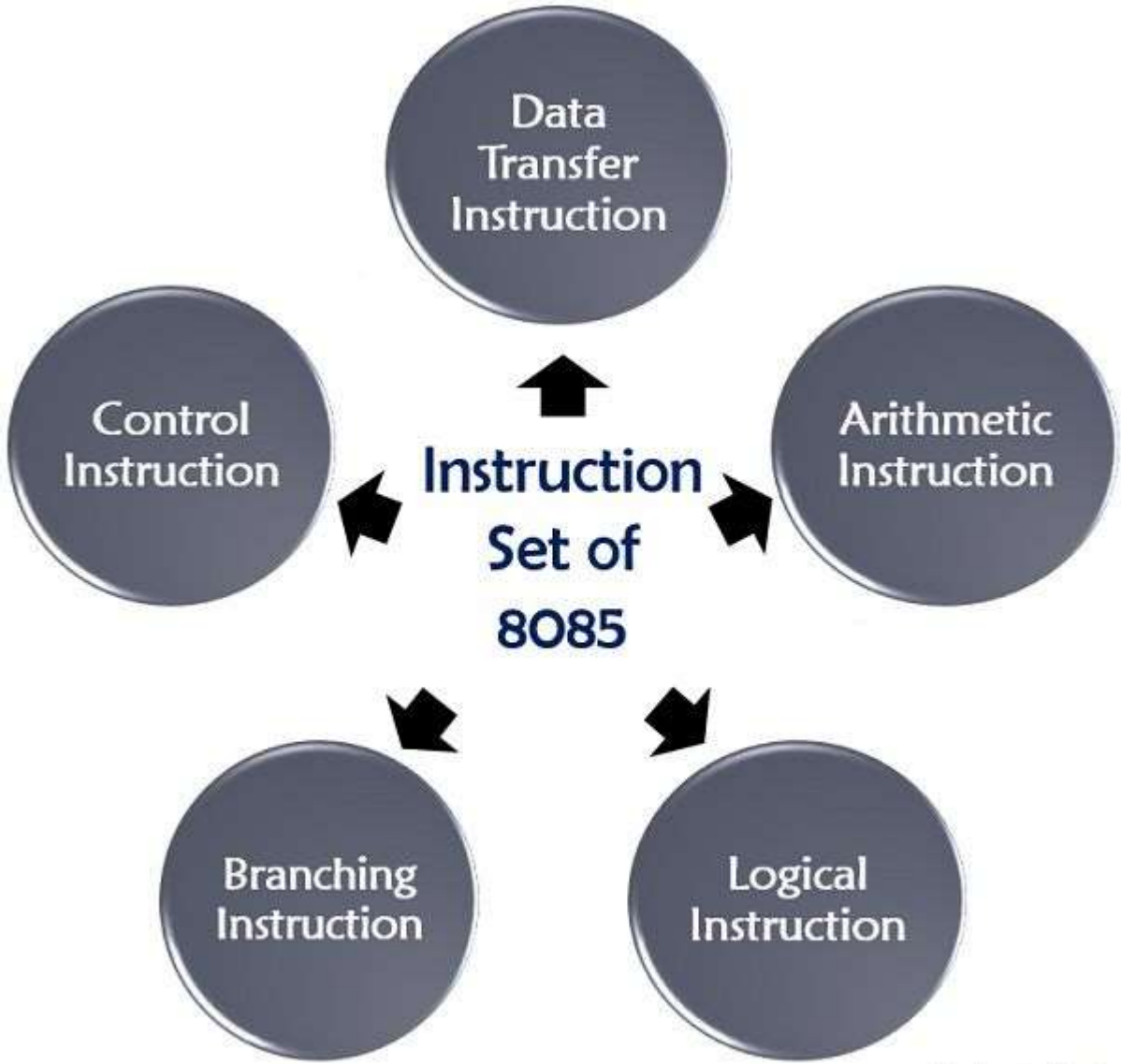
The 8085 Bus Structure

The 8-bit 8085 CPU (or MPU – Micro Processing Unit) communicates with the other units using a 16-bit address bus, an 8-bit data bus and a control bus.



Over all structure




Data Transfer Instruction
Control Instruction
Arithmetic Instruction
Instruction Set of 8085
Branching Instruction
Logical Instruction
Electronics Desk

Instruction Set

8085 Instructions

An **instruction** is a command given to the microprocessor to perform a specified operation on given data.
Some instructions of Intel 8085 microprocessor are: MOV, MVI, LDA, STA, ADD, SUB, RAL, INR, MVI, etc.

Opcode and Operands

Each instruction contains two parts: Opcode (Operation code) and Operand

- Broadly classified into two types:
  - Based on word size:
    - One Byte- Opcode only (CMA, ADD B)
    - Two Byte- Opcode ,an operand (MVI A,32H)
    - Three Byte- Opcode, operand, operand (LXI H, 4500, LDA 4200, STA 4500)
  - Based on function:
    - Data transfer group (MOV A,B; MVI A,32H;MOV C,4500)
    - Arithmetic operations (ADD B, SBI 32H,INC D, DEC B)
    - Logical operations (ANA B, ORI 05H, RLC, RAR)
    - Branching operations (JUMP, JMP, JNZ, JC, CALL, RETURN)
    - Machine control instructions (HLT, NOP,EI,DI,SIM,RIM)

Instruction Set of 8085

- An instruction is a binary pattern designed inside a microprocessor to perform a specific function.
- The entire group of instructions that a microprocessor supports is called ***Instruction Set***.
- 8085 has **246** instructions.
- Each instruction is represented by an 8-bit binary value.
- These 8-bits of binary value is called ***Op-Code*** or ***Instruction Byte***.

Classification of Instruction Set

- Data Transfer Instruction
- Arithmetic Instructions
- Logical Instructions
- Branching Instructions
- Control Instructions

Data Transfer Instructions

- These instructions move data between registers, or between memory and registers.
- These instructions copy data from source to destination.
- While copying, the contents of source are not modified.

Data Transfer Instructions

Opcode	Operand	Description
MOV	Rd, Rs M, Rs Rd, M	Copy from source to destination.

- This instruction copies the contents of the source register into the destination register.
- The contents of the source register are not altered.
- If one of the operands is a memory location, its location is specified by the contents of the HL registers.
- **Example:** MOV B, C or MOV B, M

Data Transfer Instructions

Opcode	Operand	Description
MVI	Rd, Data M, Data	Move immediate 8-bit

- The 8-bit data is stored in the destination register or memory.
- If the operand is a memory location, its location is specified by the contents of the H-L registers.
- **Example:** MVI B, 57H or MVI M, 57H

Data Transfer Instructions

Opcode	Operand	Description
LDA	16-bit address	Load Accumulator

- The contents of a memory location, specified by a 16-bit address in the operand, are copied to the accumulator.
- The contents of the source are not altered.
- **Example:** LDA 2034H

Data Transfer Instructions

Opcode	Operand	Description
LDAX	B/D Register Pair	Load accumulator indirect

- The contents of the designated register pair point to a memory location.
- This instruction copies the contents of that memory location into the accumulator.
- The contents of either the register pair or the memory location are not altered.
- **Example:** LDAX B

Data Transfer Instructions

Opcode	Operand	Description
LXI	Reg. pair, 16-bit data	Load register pair immediate

- This instruction loads 16-bit data in the register pair.
- **Example:** LXI H, 2034 H

Data Transfer Instructions

Opcode	Operand	Description
LHLD	16-bit address	Load H-L registers direct

- This instruction copies the contents of memory location pointed out by 16-bit address into register L.
- It copies the contents of next memory location into register H.
- **Example:** LHLD 2040 H

Data Transfer Instructions

Opcode	Operand	Description
STA	16-bit address	Store accumulator direct

- The contents of accumulator are copied into the memory location specified by the operand.
- **Example:** STA 2500 H

Data Transfer Instructions

Opcode	Operand	Description
STAX	Reg. pair	Store accumulator indirect

- The contents of accumulator are copied into the memory location specified by the contents of the register pair.
- **Example:** STAX B

Data Transfer Instructions

Opcode	Operand	Description
SHLD	16-bit address	Store H-L registers direct

- The contents of register L are stored into memory location specified by the 16-bit address.
- The contents of register H are stored into the next memory location.
- **Example:** SHLD 2550 H

Data Transfer Instructions

Opcode	Operand	Description
XCHG	None	Exchange H-L with D-E

- The contents of register H are exchanged with the contents of register D.
- The contents of register L are exchanged with the contents of register E.
- **Example:** XCHG

Data Transfer Instructions

Opcode	Operand	Description
SPHL	None	Copy H-L pair to the Stack Pointer (SP)

- This instruction loads the contents of H-L pair into SP.
- **Example:** SPHL

Data Transfer Instructions

Opcode	Operand	Description
XTHL	None	Exchange H–L with top of stack

- The contents of L register are exchanged with the location pointed out by the contents of the SP.
- The contents of H register are exchanged with the next location (SP + 1).
- **Example:** XTHL

Data Transfer Instructions

Opcode	Operand	Description
PCHL	None	Load program counter with H-L contents

- The contents of registers H and L are copied into the program counter (PC).
- The contents of H are placed as the high-order byte and the contents of L as the low-order byte.
- **Example:** PCHL

Data Transfer Instructions

Opcode	Operand	Description
PUSH	Reg. pair	Push register pair onto stack

- The contents of register pair are copied onto stack.
- SP is decremented and the contents of high-order registers (B, D, H, A) are copied into stack.
- SP is again decremented and the contents of low-order registers (C, E, L, Flags) are copied into stack.
- **Example:** PUSH B

Data Transfer Instructions

Opcode	Operand	Description
POP	Reg. pair	Pop stack to register pair

- The contents of top of stack are copied into register pair.
- The contents of location pointed out by SP are copied to the low-order register (C, E, L, Flags).
- SP is incremented and the contents of location are copied to the high-order register (B, D, H, A).
- **Example:** POP H

Data Transfer Instructions

Opcode	Operand	Description
OUT	8-bit port address	Copy data from accumulator to a port with 8-bit address

- The contents of accumulator are copied into the I/O port.
- **Example:** OUT 78 H

Data Transfer Instructions

Opcode	Operand	Description
IN	8-bit port address	Copy data to accumulator from a port with 8-bit address

- The contents of I/O port are copied into accumulator.
- **Example:** IN 8C H

Arithmetic Instructions

- These instructions perform the operations like:
  - Addition
  - Subtract
  - Increment
  - Decrement

Addition

- Any 8-bit number, or the contents of register, or the contents of memory location can be added to the contents of accumulator.
- The result (sum) is stored in the accumulator.
- No two other 8-bit registers can be added directly.
- **Example:** The contents of register B cannot be added directly to the contents of register C.

Subtraction

- Any 8-bit number, or the contents of register, or the contents of memory location can be subtracted from the contents of accumulator.
- The result is stored in the accumulator.
- Subtraction is performed in 2's complement form.
- If the result is negative, it is stored in 2's complement form.
- No two other 8-bit registers can be subtracted directly.

Increment / Decrement

- The 8-bit contents of a register or a memory location can be incremented or decremented by 1.
- The 16-bit contents of a register pair can be incremented or decremented by 1.
- Increment or decrement can be performed on any register or a memory location.

Arithmetic Instructions

Opcode	Operand	Description
ADD	R M	Add register or memory to accumulator

- The contents of register or memory are added to the contents of accumulator.
- The result is stored in accumulator.
- If the operand is memory location, its address is specified by H-L pair.
- All flags are modified to reflect the result of the addition.
- **Example:** ADD B or ADD M

Arithmetic Instructions

Opcode	Operand	Description
ADC	R M	Add register or memory to accumulator with carry

- The contents of register or memory and Carry Flag (CY) are added to the contents of accumulator.
- The result is stored in accumulator.
- If the operand is memory location, its address is specified by H-L pair.
- All flags are modified to reflect the result of the addition.
- **Example:** ADC B or ADC M

Arithmetic Instructions

Opcode	Operand	Description
ADI	8-bit data	Add immediate to accumulator

- The 8-bit data is added to the contents of accumulator.
- The result is stored in accumulator.
- All flags are modified to reflect the result of the addition.
- **Example:** ADI 45 H

Arithmetic Instructions

Opcode	Operand	Description
ACI	8-bit data	Add immediate to accumulator with carry

- The 8-bit data and the Carry Flag (CY) are added to the contents of accumulator.
- The result is stored in accumulator.
- All flags are modified to reflect the result of the addition.
- **Example:** ACI 45 H

Arithmetic Instructions

Opcode	Operand	Description
DAD	Reg. pair	Add register pair to H-L pair

- The 16-bit contents of the register pair are added to the contents of H-L pair.
- The result is stored in H-L pair.
- If the result is larger than 16 bits, then CY is set.
- No other flags are changed.
- **Example:** DAD B

Arithmetic Instructions

Opcode	Operand	Description
SUB	R M	Subtract register or memory from accumulator

- The contents of the register or memory location are subtracted from the contents of the accumulator.
- The result is stored in accumulator.
- If the operand is memory location, its address is specified by H-L pair.
- All flags are modified to reflect the result of subtraction.
- **Example:** SUB B or SUB M

Arithmetic Instructions

Opcode	Operand	Description
SBB	R M	Subtract register or memory from accumulator with borrow

- The contents of the register or memory location and Borrow Flag (i.e. CY) are subtracted from the contents of the accumulator.
- The result is stored in accumulator.
- If the operand is memory location, its address is specified by H-L pair.
- All flags are modified to reflect the result of subtraction.
- **Example:** SBB B or SBB M

Arithmetic Instructions

Opcode	Operand	Description
SUI	8-bit data	Subtract immediate from accumulator

- The 8-bit data is subtracted from the contents of the accumulator.
- The result is stored in accumulator.
- All flags are modified to reflect the result of subtraction.
- **Example:** SUI 45 H

Arithmetic Instructions

Opcode	Operand	Description
SBI	8-bit data	Subtract immediate from accumulator with borrow

- The 8-bit data and the Borrow Flag (i.e. CY) is subtracted from the contents of the accumulator.
- The result is stored in accumulator.
- All flags are modified to reflect the result of subtraction.
- **Example:** SBI 45 H

Arithmetic Instructions

Opcode	Operand	Description
INR	R M	Increment register or memory by 1

- The contents of register or memory location are incremented by 1.
- The result is stored in the same place.
- If the operand is a memory location, its address is specified by the contents of H-L pair.
- **Example:** INR B or INR M

Arithmetic Instructions

Opcode	Operand	Description
INX	RP	Increment register pair by 1

- The contents of register pair are incremented by 1.
- The result is stored in the same place.
- **Example:** INX H

Arithmetic Instructions

Opcode	Operand	Description
DCR	R M	Decrement register or memory by 1

- The contents of register or memory location are decremented by 1.
- The result is stored in the same place.
- If the operand is a memory location, its address is specified by the contents of H-L pair.
- **Example:** DCR B or DCR M

Arithmetic Instructions

Opcode	Operand	Description
DCX	R	Decrement register pair by 1

- The contents of register pair are decremented by 1.
- The result is stored in the same place.
- **Example:** DCX H

Logical Instructions

- These instructions perform logical operations on data stored in registers, memory and status flags.

- The logical operations are:
  - AND
  - OR
  - XOR
  - Rotate
  - Compare
  - Complement

AND, OR, XOR

- Any 8-bit data, or the contents of register, or memory location can logically have
  - AND operation
  - OR operation
  - XOR operation

  with the contents of accumulator.
- The result is stored in accumulator.

Rotate Instructions

Compare

- Any 8-bit data, or the contents of register, or memory location can be compares for:
  - Equality
  - Greater Than
  - Less Than

  with the contents of accumulator.
- The result is reflected in status flags.

Complement

- The contents of accumulator can be complemented.
- Each 0 is replaced by 1 and each 1 is replaced by 0.

Logical Instructions

Opcode	Operand	Description
CMP	R M	Compare register or memory with accumulator

- The contents of the operand (register or memory) are compared with the contents of the accumulator.
- Both contents are preserved .
- The result of the comparison is shown by setting the flags of the PSW as follows:

Logical Instructions

Opcode	Operand	Description
CMP	R M	Compare register or memory with accumulator

- if (A) < (reg/mem): carry flag is set
- if (A) = (reg/mem): zero flag is set
- if (A) > (reg/mem): carry and zero flags are reset.
- **Example:** CMP B or CMP M

Logical Instructions

Opcode	Operand	Description
CPI	8-bit data	Compare immediate with accumulator

- The 8-bit data is compared with the contents of accumulator.
- The values being compared remain unchanged.
- The result of the comparison is shown by setting the flags of the PSW as follows:

Logical Instructions

Opcode	Operand	Description
CPI	8-bit data	Compare immediate with accumulator

- if (A) < data: carry flag is set
- if (A) = data: zero flag is set
- if (A) > data: carry and zero flags are reset
- **Example:** CPI 89H

Logical Instructions

Opcode	Operand	Description
ANA	R M	Logical AND register or memory with accumulator

- The contents of the accumulator are logically ANDed with the contents of register or memory.
- The result is placed in the accumulator.
- If the operand is a memory location, its address is specified by the contents of H-L pair.
- S, Z, P are modified to reflect the result of the operation.
- CY is reset and AC is set.
- **Example:** ANA B or ANA M.

Logical Instructions

Opcode	Operand	Description
ANI	8-bit data	Logical AND immediate with accumulator

- The contents of the accumulator are logically ANDed with the 8-bit data.
- The result is placed in the accumulator.
- S, Z, P are modified to reflect the result.
- CY is reset, AC is set.
- **Example:** ANI 86H.

Logical Instructions

Opcode	Operand	Description
ORA	R M	Logical OR register or memory with accumulator

- The contents of the accumulator are logically ORed with the contents of the register or memory.
- The result is placed in the accumulator.
- If the operand is a memory location, its address is specified by the contents of H-L pair.
- S, Z, P are modified to reflect the result.
- CY and AC are reset.
- **Example:** ORA B or ORA M.

Logical Instructions

Opcode	Operand	Description
ORI	8-bit data	Logical OR immediate with accumulator

- The contents of the accumulator are logically ORed with the 8-bit data.
- The result is placed in the accumulator.
- S, Z, P are modified to reflect the result.
- CY and AC are reset.
- **Example:** ORI 86H.

Logical Instructions

Opcode	Operand	Description
XRA	R M	Logical XOR register or memory with accumulator

- The contents of the accumulator are XORed with the contents of the register or memory.
- The result is placed in the accumulator.
- If the operand is a memory location, its address is specified by the contents of H-L pair.
- S, Z, P are modified to reflect the result of the operation.
- CY and AC are reset.
- **Example:** XRA B or XRA M.

Logical Instructions

Opcode	Operand	Description
XRI	8-bit data	XOR immediate with accumulator

- The contents of the accumulator are XORed with the 8-bit data.
- The result is placed in the accumulator.
- S, Z, P are modified to reflect the result.
- CY and AC are reset.
- **Example:** XRI 86H.

Rotate Instructions:

Each bit in the accumulator can be shifted either left or right to the next position.

Opcode	Operand	Description
RLC	None	Rotate accumulator left

- Each binary bit of the accumulator is rotated left by one position.
- Bit D7 is placed in the position of Do as well as in the Carry flag.
- CY is modified according to bit D7.
- S, Z, P, AC are not affected.
- **Example:** RLC.

Rotate Instructions:

Opcode	Operand	Description
RRC	None	Rotate accumulator right

- Each binary bit of the accumulator is rotated right by one position.
- Bit Do is placed in the position of D7 as well as in the Carry flag.
- CY is modified according to bit Do.
- S, Z, P, AC are not affected.
- **Example:** RRC.

Rotate Instructions:

Opcode	Operand	Description
RAL	None	Rotate accumulator left through carry

- Each binary bit of the accumulator is rotated left by one position through the Carry flag.
- Bit D7 is placed in the Carry flag, and the Carry flag is placed in the least significant position Do.
- CY is modified according to bit D7.
- S, Z, P, AC are not affected.
- **Example:** RAL.

Rotate Instructions:

Opcode	Operand	Description
RAR	None	Rotate accumulator right through carry

- Each binary bit of the accumulator is rotated right by one position through the Carry flag.
- Bit Do is placed in the Carry flag, and the Carry flag is placed in the most significant position D7.
- CY is modified according to bit Do.
- S, Z, P, AC are not affected.
- **Example:** RAR.

Logical Instructions

Opcode	Operand	Description
CMA	None	Complement accumulator

- The contents of the accumulator are complemented.
- No flags are affected.
- **Example:** CMA.

Logical Instructions

Opcode	Operand	Description
CMC	None	Complement carry

- The Carry flag is complemented.
- No other flags are affected.
- **Example:** CMC.

Logical Instructions

Opcode	Operand	Description
STC	None	Set carry

- The Carry flag is set to 1.
- No other flags are affected.
- **Example:** STC.

Branching Instructions

- The branching instruction alter the normal sequential flow.

- These instructions alter either unconditionally or conditionally.

Branching Instructions

Opcode	Operand	Description
JMP	16-bit address	Jump unconditionally

- The program sequence is transferred to the memory location specified by the 16-bit address given in the operand.
- **Example:** JMP 2034 H.

Branching Instructions

Opcode	Operand	Description
Jx	16-bit address	Jump conditionally

- The program sequence is transferred to the memory location specified by the 16-bit address given in the operand based on the specified flag of the PSW.
- **Example:** JZ 2034 H.

Jump Conditionally

Opcode	Description	Status Flags
JC	Jump if Carry	CY = 1
JNC	Jump if No Carry	CY = 0
JP	Jump if Positive	S = 0
JM	Jump if Minus	S = 1
JZ	Jump if Zero	Z = 1
JNZ	Jump if No Zero	Z = 0
JPE	Jump if Parity Even	P = 1
JPO	Jump if Parity Odd	P = 0

Branching Instructions

Opcode	Operand	Description
CALL	16-bit address	Call unconditionally

- The program sequence is transferred to the memory location specified by the 16-bit address given in the operand.
- Before the transfer, the address of the next instruction after CALL (the contents of the program counter) is pushed onto the stack.
- **Example:** CALL 2034 H.

Branching Instructions

Opcode	Operand	Description
Cx	16-bit address	Call conditionally

- The program sequence is transferred to the memory location specified by the 16-bit address given in the operand based on the specified flag of the PSW.
- Before the transfer, the address of the next instruction after the call (the contents of the program counter) is pushed onto the stack.
- **Example:** CZ 2034 H.

Call Conditionally

Opcode	Description	Status Flags
CC	Call if Carry	CY = 1
CNC	Call if No Carry	CY = 0
CP	Call if Positive	S = 0
CM	Call if Minus	S = 1
CZ	Call if Zero	Z = 1
CNZ	Call if No Zero	Z = 0
CPE	Call if Parity Even	P = 1
CPO	Call if Parity Odd	P = 0

Branching Instructions

Opcode	Operand	Description
RET	None	Return unconditionally

- The program sequence is transferred from the subroutine to the calling program.
- The two bytes from the top of the stack are copied into the program counter, and program execution begins at the new address.
- **Example:** RET.

Branching Instructions

Opcode	Operand	Description
Rx	None	Call conditionally

- The program sequence is transferred from the subroutine to the calling program based on the specified flag of the PSW.
- The two bytes from the top of the stack are copied into the program counter, and program execution begins at the new address.
- **Example:** RZ.

Return Conditionally

Opcode	Description	Status Flags
RC	Return if Carry	CY = 1
RNC	Return if No Carry	CY = 0
RP	Return if Positive	S = 0
RM	Return if Minus	S = 1
RZ	Return if Zero	Z = 1
RNZ	Return if No Zero	Z = 0
RPE	Return if Parity Even	P = 1
RPO	Return if Parity Odd	P = 0

Branching Instructions

Opcode	Operand	Description
RST	0 – 7	Restart (Software Interrupts)

- The RST instruction jumps the control to one of eight memory locations depending upon the number.
- These are used as software instructions in a program to transfer program execution to one of the eight locations.
- **Example:** RST 3.

Restart Address Table

Instructions	Restart 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

Control Instructions

- The control instructions control the operation of microprocessor.

Control Instructions

Opcode	Operand	Description
NOP	None	No operation

- No operation is performed.
- The instruction is fetched and decoded but no operation is executed.
- **Example:** NOP

Control Instructions

Opcode	Operand	Description
HLT	None	Halt

- The CPU finishes executing the current instruction and halts any further execution.
- An interrupt or reset is necessary to exit from the halt state.
- **Example:** HLT

Control Instructions

Opcode	Operand	Description
DI	None	Disable interrupt

- The interrupt enable flip-flop is reset and all the interrupts except the TRAP are disabled.
- No flags are affected.
- **Example:** DI

Control Instructions

Opcode	Operand	Description
EI	None	Enable interrupt

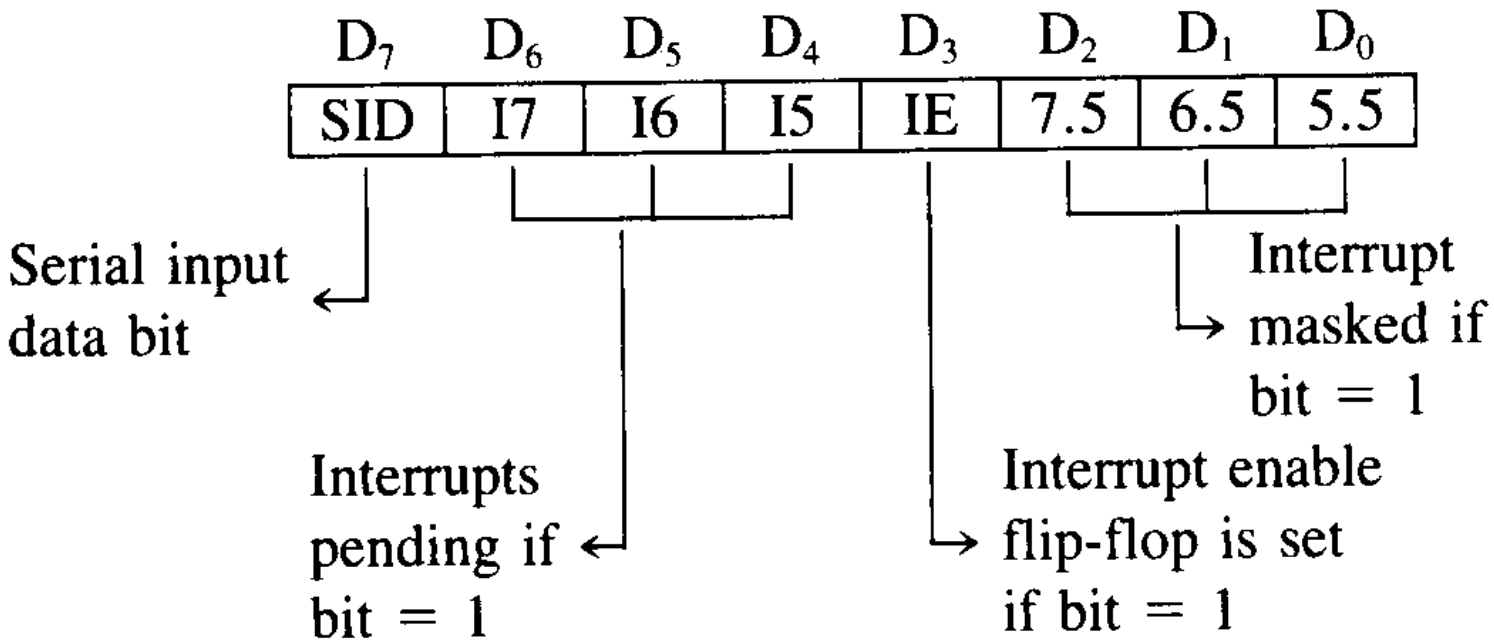
- The interrupt enable flip-flop is set and all interrupts are enabled.
- No flags are affected.
- This instruction is necessary to re-enable the interrupts (except TRAP).
- **Example:** EI

Control Instructions

Opcode	Operand	Description
RIM	None	Read Interrupt Mask

- This is a multipurpose instruction used to read the status of interrupts 7.5, 6.5, 5.5 and read serial data input bit.
- The instruction loads eight bits in the accumulator with the following interpretations.
- **Example:** RIM

RIM Instruction

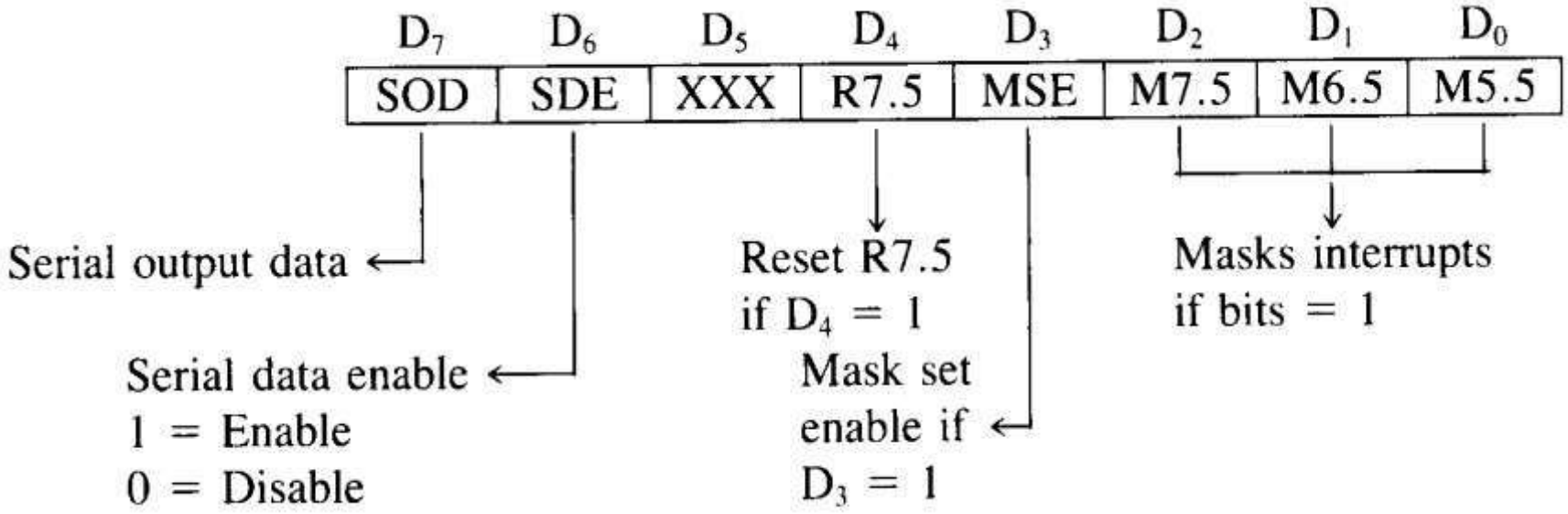


Control Instructions

Opcode	Operand	Description
SIM	None	Set Interrupt Mask

- This is a multipurpose instruction and used to implement the 8085 interrupts 7.5, 6.5, 5.5, and serial data output.
- The instruction interprets the accumulator contents as follows.
- **Example:** SIM

SIM Instruction



Programming in 8085

```
MVI A, 49H      : "Store 49H in the accumulator"
STA 2501H       : "Copy accumulator contents at address 2501H"
HLT             : "Stop"
```

Store 8-bit data in memory using indirect addressing

```
LXI H       : "Load H-L pair with 2501H"
MVI M       : "Store 49H in memory location pointed by H-L register pair (2501H)"
HLT         : "Stop"
```

Add two 8-bit numbers

Example

(2501 H) = 99H (2502 H) = 39H Result (2503 H) = 99H + 39H = D2H Since, 1 0 0 1 1 0 0 1 (99H) + 0 0 1 1 1 0 0 1 (39H) 1 1 0 1 0 0 1 0 (D2H)

Program

```
LXI H, 2501H  : "Get address of first number in H-L pair. Now H-L points to 2501H"
MOV A, M      : "Get first operand in accumulator"
INX H         : "Increment content of H-L pair. Now, H-L points 2502H"
ADD M         : "Add first and second operand"
INX H         : "H-L points 4002H"
MOV M, A      : "Store result at 2503H"
HLT           : "Stop"
```

Subtract two 8-bit numbers

Example

(2501 H) = 49H (2502 H) = 32H Result (2503 H) = 49H - 32H = 17H

Program

1.LXI H, 2501H : "Get address of first number in H-L pair. Now H-L points to 2501H"
2.MOV A, M : "Get first operand in accumulator"
3.INX H : "Increment content of H-L pair. Now, H-L points 2502H"
4.SUB M : "Subtract first to second operand"
5.INX H : "H-L points 4002H"
6.MOV M, A : "Store result at 2503H"
7.HLT : "Stop"

Add two 16-bits numbers

Add the 16-bit number in memory locations 2501H and 2502H to the 16-bit number in memory locations 2503H and 2504H. The most significant eight bits of the two numbers to be added are in memory locations 2502H and 4004H. Store the result in memory locations 2505H and 2506H with the most significant byte in memory location 2506H.

Example

(2501H) = 15H (2502H) = 1CH (2503H) = B7H (2504H) = 5AH Result = 1C15 + 5AB7H = 76CCH (2505H) = CCH (2506H) = 76H

Add two 16-bits number with ADD and ADC instruction

1.LHLD 2501H : "Get 1st 16-bit number in H-L pair"
2.XCHG : "Save 1st 16-bit number in DE"
3.LHLD 2503H : "Get 2nd 16-bit number in H-L pair"
4.MOV A, E : "Get lower byte of the 1st number"
5.ADD L : "Add lower byte of the 2nd number"
6.MOV L, A : "Store result in L-register"
7.MOV A, D : "Get higher byte of the 1st number"
8.ADC H : "Add higher byte of the 2nd number with CARRY"
9.MOV H, A : "Store result in H-register"
10.SHLD 4004H : "Store 16-bit result in memory locations 2505H and 2506H"
11.HLT : "Stop"

Add two 16-bits numbers with DAD instruction

1.LHLD 2501H : "Get 1st 16-bit number"
2.XCHG : "Save 1st 16-bit number in DE"
3.LHLD 2503H : "Get 2nd 16-bit number in H-L"
4.DAD D : "Add DE and HL"
5.SHLD 2505H : "Store 16-bit result in memory locations 2505H and 2506H".
6.HLT : "Stop"

Add two 16-bits numbers with DAD instruction

```
LHLD 2501H  : "Get 1st 16-bit number"
XCHG        : "Save 1st 16-bit number in DE"
LHLD 2503H  : "Get 2nd 16-bit number in H-L"
DAD D       : "Add DE and HL"
SHLD 2505H  : "Store 16-bit result in memory locations 2505H and 2506H".
HLT         : "Stop"
```

Subtract two 16-bit numbers

Example

(2500H) = 19H (2501H) = 6AH (2504H) = 15H (2503H) = 5CH Result = 6A19H ? 5C15H = OE04H (2504H) = 04H (2505H) = OEH

Program

```
LHLD 2500H      : "Get first 16-bit number in HL"
XCHG            : "Save first 16-bit number in DE"
LHLD 2502H      : "Get second 16-bit number in HL"
MOV A, E        : "Get lower byte of the first number"
SUB L           : "Subtract lower byte of the second number"
MOV L, A        : "Store the result in L register"
MOV A, D        : "Get higher byte of the first number"
SBB H           : "Subtract higher byte of second number with borrow"
MOV H, A        : "Store l6-bit result in memory locations 2504H and 2505H"
SHLD 2504H      : "Store l6-bit result in memory locations 2504H and 2505H"
HLT             : "Terminate program execution"
```

Add contents of two memory locations

Example

2500H) = 7FH (2501H) = 89H Result = 7FH + 89H = lO8H (2502H) = 08H (2503H) = 01H

Program

```
LXI H, 2500H    : "HL Points 2500H"
MOV A, M        : "Get first operand"
INX H           : "HL Points 2501H"
ADD M           : "Add second operand"
INX H           : "HL Points 2502H"
MOV M, A        : "Store the lower byte of result at 2502H"
MVI A, 00       : "Initialize higher byte result with 00H"
ADC A           : "Add carry in the high byte result"
INX H           : "HL Points 2503H"
MOV M, A        : "Store the higher byte of result at 2503H"
HLT             : "Terminate program execution"
```

Finding 1's complement of a number

To obtain one's complement of a number its 0 bits are replaced by 1 and 1 by 0.

Example 1

(2501H) = 96 H = 1001 0110 (9) (6) One's complement = 0110 1001 = 69 H Result = (2502H) = 69H

Example 2

(2501H) = E4H Result = (2502H) = 1BH

Program

The number is placed in the memory location 2501 H.

The result is stored in the memory location 2502 H.

```
LDA 2501H   : "Get the number IN accumulator"
CMA         : "take its complement"
STA 2502H   : "Store result in 2502H"
HLT         : "Stop"
```

Finding 2's complement of a number

2's complement of a number is obtained by adding 1 to the 1's complement of the number.

Example

To find the two's complement of 96. 96 = 1001 0110 1's complement = 0110 1001 = 69 + 0000 0001 2's complement = 0110 1010 = 6 A

Program

The number is placed in the memory location 2501 H.

The result is to be stored in the memory location 2502 H.

```
LDA 2501 H  : "Get data in accumulator"
CMA         : "Take its 1's complement"
ADI, 01 H   : "Add one in the number"
STA 2502 H  : "Store the result in 2502 H"
HLT         : "Stop"
```

Count number of 1's in a number

Example

2501 H = 04 2502 H = 34 H 2503 H = A9H 2504 H = 78H 2505 H = 56H Result = 2503 H = A9H

Program

Count number of 1's of the content of the register D and store the count in the register B.

```
MVI B, 00H
MVI C, 08H
MOV A, D
BACK: RAR
JNC SKIP
INR B
SKIP: DCR C
JNZ BACK
HLT
```

Program: The first number 98H is placed in the memory location 2501 H.,The second number 87H is placed in the memory location 2502H. The result is stored in the memory location 2503 H.

LXI H, 2501H : "Address of first number in H-L pair“
MOV A, M : "1stt number in accumulator"
INX H : "Address of 2nd number in H-L pair"
CMP M : "compare 2nd number with 1st number"
JNC AHEAD : "No, larger is in accumulator.
MOV A, M : "Yes, get 2nd number in the accumulator"
AHEAD: STA 2503 H : "Store larger number in 2503H"
HLT : "Stop"

Find smaller of two numbers

Example

2501H = 84 H 2502H = 99 H Result = 84 H(2503H)

Program

The first number 84H is placed in the memory location 2501 H., The second number 99H is placed in the memory location 2502H.
The result is stored in the memory location 2503 H.
LXI H, 2501H : "Address of first number in H-L pair"
MOV A, M : "1stt number in accumulator"
INX H : "Address of 2nd number in H-L pair"
CMP M : "compare 2nd number with 1st number"
JC AHEAD : "Yes, smaller number is in accumulator.
MOV A, M : "No, get 2nd number in the accumulator"
AHEAD::STA 2503 H : "Store smaller number in 2503H“
HLT : "Stop"

Calculate the sum of series of even numbers

Example

2500 H = 4H, 2501 H = 20H, 2502 H = 15H, 2503 H = 13H, 2504 H = 22H
Result = 2505 H = 20+22= 42H

Program

The numbers are placed in the memory locations 2501 to 2504H.

The sum is to be stored in the memory location 2450H.

As there are 4 numbers in the series, count = 04

The initial value of the sum is made 00. The even number of the series are taken one by one and added to the sum.

```
        LDA 2500H
        MOV C, A        : "Initialize counter"
        MVI B, 00H      : "sum = 0"
        LXI H, 2501H    : "Initialize pointer"
BACK:   MOV A, M        : "Get the number"
        ANI 01H         : "Mask Bit l to Bit7"
        JNZ SKIP        : "Don't add if number is ODD"
        MOV A, B        : "Get the sum"
        ADD M           : "SUM = SUM + data"
        MOV B, A        : "Store result in B register"
SKIP:   INX H           : "increment pointer"
        DCR C           : "Decrement counter"
        JNZ BACK        : "if counter 0 repeat"
        STA 2505H       : "store sum"
        HLT             : "Stop"
```

Calculate the sum of series of even numbers

Example

2500 H = 4H 2501 H = 20H 2502 H = 15H 2503 H = 13H 2504 H = 22H Result = 2505 H = 20+22= 42H

Program

The numbers are placed in the memory locations 2501 to 2504H.
The sum is to be stored in the memory location 2450H.
As there are 4 numbers in the series, count = 04
The initial value of the sum is made 00. The even number of the series are taken one by one and added to the sum.

```
        LDA 2500H
        MOV C, A        : "Initialize counter"
        MVI B, 00H      : "sum = 0"
        LXI H, 2501H    : "Initialize pointer"
BACK: MOV A, M    : "Get the number"
        ANI 01H         : "Mask Bit l to Bit7"
        JNZ SKIP        : "Don't add if number is ODD"
        MOV A, B        : "Get the sum"
        ADD M           : "SUM = SUM + data"
        MOV B, A        : "Store result in B register"
SKIP: INX H       : "increment pointer"
        DCR C           : "Decrement counter"
        JNZ BACK        : "if counter 0 repeat"
        STA 2505H       : "store sum"
        HLT             : "Stop"
```

Calculate the sum of series of odd numbers

Example

2500 H = 4H 2501 H = 9AH 2502 H = 52H 2503 H = 89H 2504 H = 3FH Result = 2505 H = 89H + 3FH= C8H

Program

The numbers are placed in the memory locations 2501 to 2504H.
The sum is to be stored in the memory location 2450H.
As there are 4 numbers in the series, count = 04
The initial value of the sum is made 00. The odd number of the series are taken one by one and added to the sum.

```
LDA 2500H
MOV C, A        : "Initialize counter"
LXI H, 2501H    : "Initialize pointer"
MVI E, 00       : "Sum low = 0"
MOV D, E        : "Sum high = 0"
BACK: MOV A, M    : "Get the number"
ANI 01H         : "Mask Bit 1 to Bit-7"
JZ SKIP         : "Don't add if number is even"
MOV A, E        : "Get the lower byte of sum"
ADD M           : "Sum = sum + data"
MOV E, A        : "Store result in E register"
JNC SKIP
INR D           : "Add carry to MSB of SUM"
SKIP: INX H     : "Increment pointer"
```

Find the square of given number

Program

Find the square of 07 (decimal) using lookup table techniques.

The number 07 D is in the memory.

The result is to be stored in the memory location 2501H.

The table for square is stored from 2600 to 2609 H.

```
LDA 2500H   : "Get data in accumulator"
MOV L, A    : "Get data in register L"
MVI H, 26H  : "Get 26 in register H"
MOV A, M    : "Square of data in accumulator"
STA 2501 H  : "Store Square in 2501 H"
HLT : "Stop"
```

separate even numbers from given numbers

Program

Let's see the program to separate even numbers from t
given list of 50 numbers and store them in other locati
starting from 2600H.

Starting location of 50 number list is 2500H.

```
LXI H, 2500H       : "Initialize memory pointer 1"
LXI D, 2600H       : "Initialize memory pointer2"
MVI C, 32H         : "Initialize counter"
BACK:MOV A, M      : "Get the number"
ANI 01H            : "Check for even number"
JNZ SKIP           : "If ODD, don't store"
MOV A, M           : "Get the number"
STAX D             : "Store the number in result list"
INX D              : "Increment pointer 2"
SKIP: INX H        : "Increment pointer 1"
DCR C              : "Decrement counter"
JNZ BACK           : "If not zero, repeat"
HLT                : "Stop"
```

Memory Interfacing in 8085:

Memory is an integral part of a microprocessor system, and in this section, we will discuss how to interface a memory device with the microprocessor. The Memory Interfacing in 8085 is used to access memory quite frequently to read instruction codes and data stored in memory. This read/write operations are monitored by control signals.

Memory Structure and its Requirements:

As mentioned earlier, read/write memories consist of an array of registers, in which each register has unique address. The size of the memory is N x M as shown in Fig. 4.13 (a) where N is the number of registers and M is the word length, in number of bits.

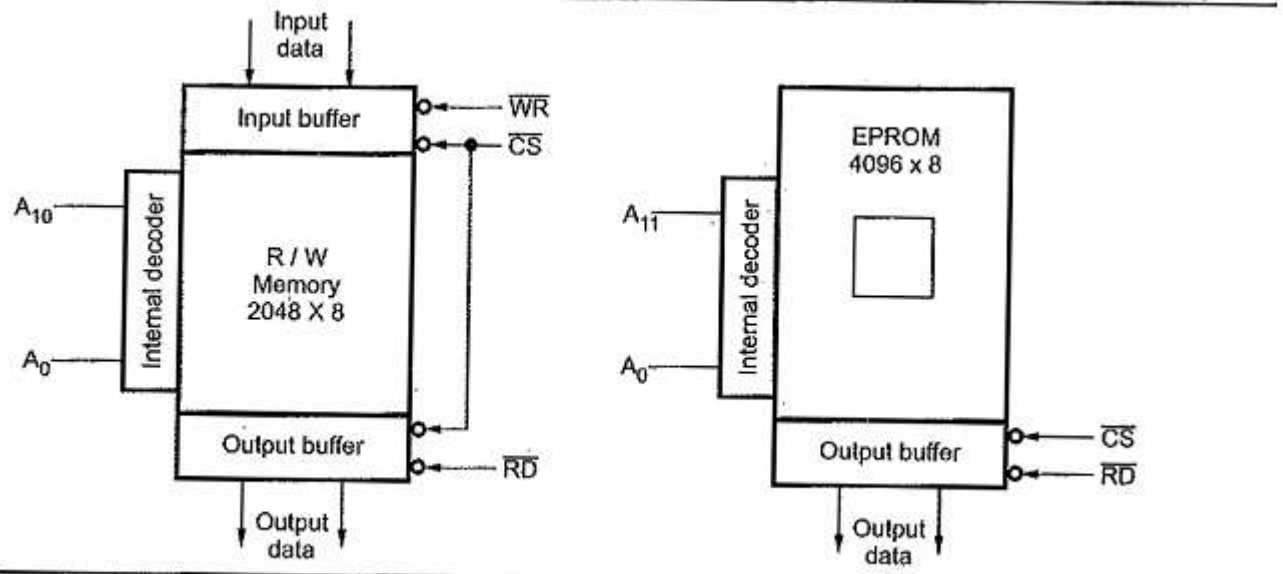


Fig. 4.13 (a) Logic diagram for RAM Fig. 4.13 (b) Logic diagram for EPROM

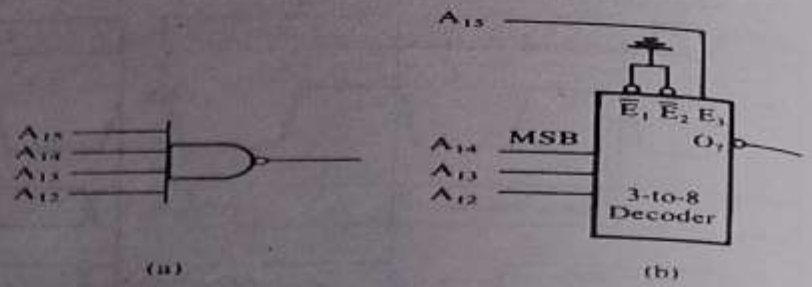


FIGURE 4.14
Address Decoding Using NAND Gate (a) and 3-to-8 Decoder (b)

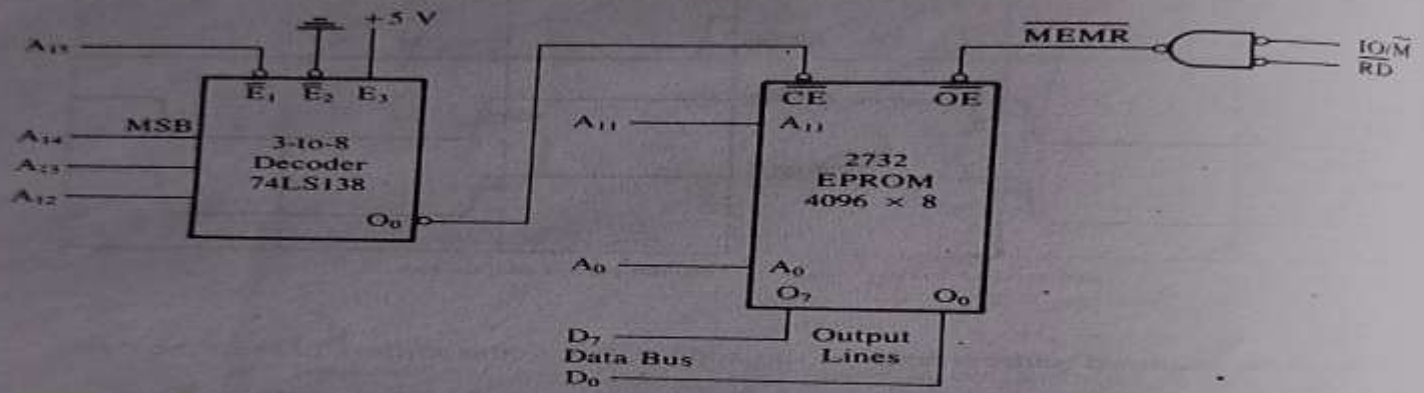


FIGURE 4.15
Interfacing the 2732 EPROM

lines are active. In this circuit, the Enable lines $\overline{E}_1$ and $\overline{E}_2$ are enabled by grounding, and A_{15} must be at logic 1 to enable E_3. We will use this address decoding scheme to interface a 4K EPROM and a 2K R/W memory as illustrated in the next two examples.

4.3.4 Interfacing Circuit

Figure 4.15 shows an interfacing circuit using a 3-to-8 decoder to interface the 2732 EPROM memory chip. It is assumed here that the chip has already been programmed, and we will analyze the interfacing circuit in terms of the same three steps outlined previously:

Step 1: The 8085 address lines A_{11}–A_0 are connected to pins A_{11}–A_0 of the memory chip to address 4096 registers.

Step 2: The decoder is used to decode four address lines A_{15}–A_{12}. The output O_0 of the decoder is connected to Chip Enable (CE). The CE is asserted only when the address on A_{15}–A_{12} is 0000; A_{15} (low) enables the decoder and the input 000 asserts the output O_0.

Step 3: For this EPROM, we need one control signal: Memory Read (MEMR), active low. The MEMR is connected to OE to enable the output buffer; OE is the same as RD in Figure 4.11.

4.35 Address Decoding and Memory Addresses

We can obtain the address range of this memory chip by analyzing the possible logic levels on the 16 address lines. The logic levels on the address lines A_{15}–A_{12} must be 0000 to assert the Chip Enable, and the address lines A_{11}–A_0 can assume any combinations from all 0s to all 1s. Therefore, the memory address of this chip ranges from 0000H to 0FFFH, as shown below.

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	
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	= 0000H
0	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	= 0FFFH
Chip Enable				Register Select												

We can verify the memory address range in terms of our analogy of page and line numbers, as discussed in Chapter 3, Section 3.22. The chip's 4096 bytes of memory can be viewed as 16 pages with 256 lines each. The high-order Hex digits range from 00 to 0F, indicating 16 pages—0000H to 00FFH and 0100H to 01FFH, for example.

Now, to examine how an address is decoded and how the microprocessor reads from this memory, let us assume that the 8085 places the address 0FFFH on the address bus. The address 0000 (0H) goes to the decoder, and the output line O_0 of the decoder selects the chip. The remaining address FFFH goes on the address lines of the chip, and the internal decoder of the chip decodes the address and selects the register FFFH. Thus, the address 0FFFH selects the register as shown in Figure 4.16. When the 8085 asserts the RD signal, the output buffer is enabled and the contents of the register 0FFFH are placed on the data bus for the processor to read.

Analyze the interfacing circuit in Figure 4.17 and find its memory address range.

Figure 4.17 shows the interfacing of the 6116 memory chip with 2048 (2K) registers. The memory chip requires 11 address lines (A_{10}–A_0) to decode 2048 registers. The remaining address lines A_{15}–A_{11} are connected to the decoder. However, in this circuit, the decoder is enabled by the IO/M signal in addition to the address lines A_{15} and A_{14}, and the RD and

4.5.3 Circuit Analysis

Figure 4.20 shows the schematic of the interfacing circuit based on the problem analysis.

1. When the logic levels of A_{15}–A_{12} are all 0s, and the processor asserts IO/$\overline{M}$ to read from memory, the output O_0 goes active and selects the chip. The address range of the EPROM is as follows:

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	
0 0 0 0	0 0 0 0 0 0 0 0 0 0 0 0	= 0000H
$MSEL_0$	1 1 1 1 1 1 1 1 1 1 1 1	= 0FFFH

2. The 6116 R/W memory is selected by the output signal of the decoder O_2. In this circuit the address line A_{11} is at don't care logic. Assuming A_{11} is at logic 0, the address range is as follows:

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	
0 0 1 0	0 0 0 0 0 0 0 0 0 0 0 0	= 2000H
$MSEL_2$	0 1 1 1 1 1 1 1 1 1 1 1	= 27FFH

Assuming A_{12} is at logic 1, the foldback (or mirror) address range is as follows:

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

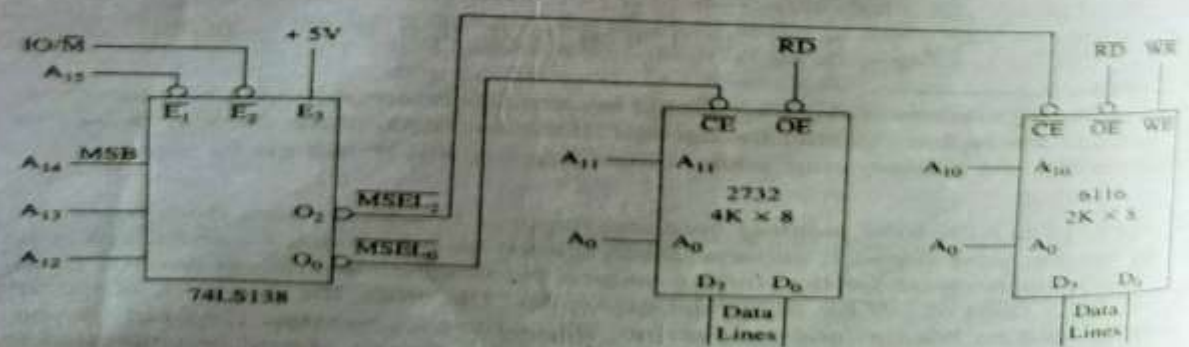
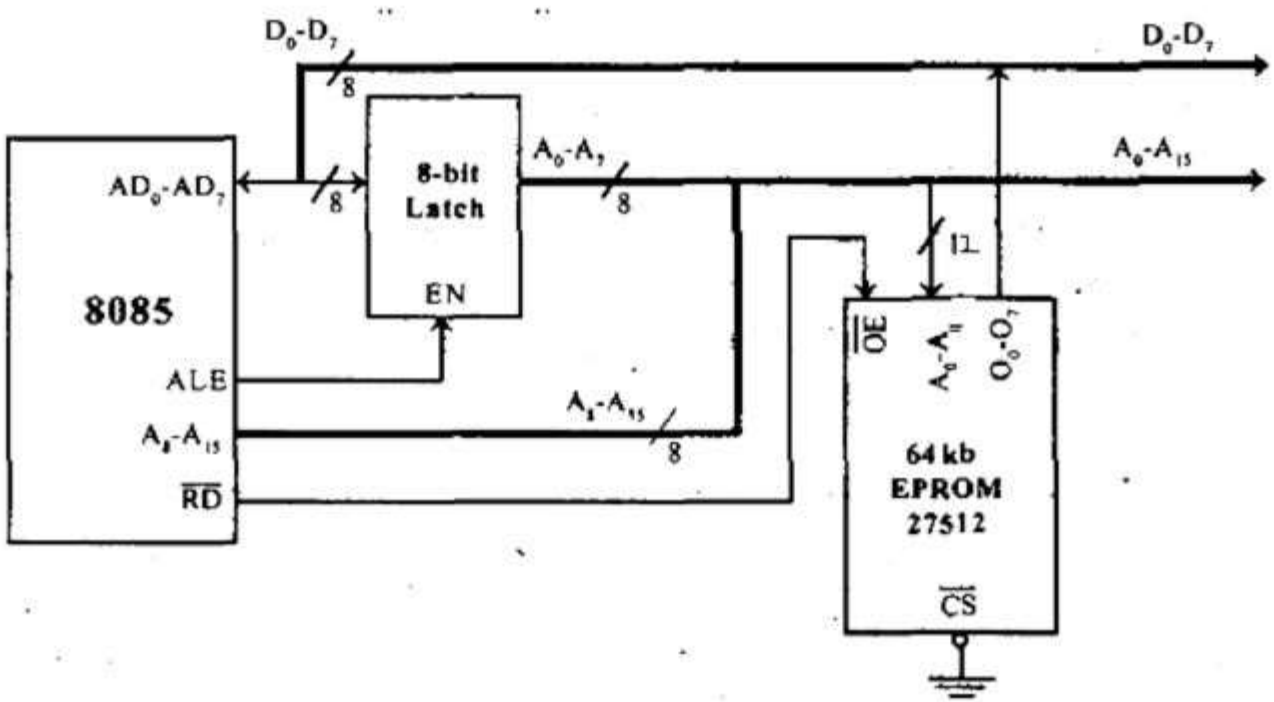
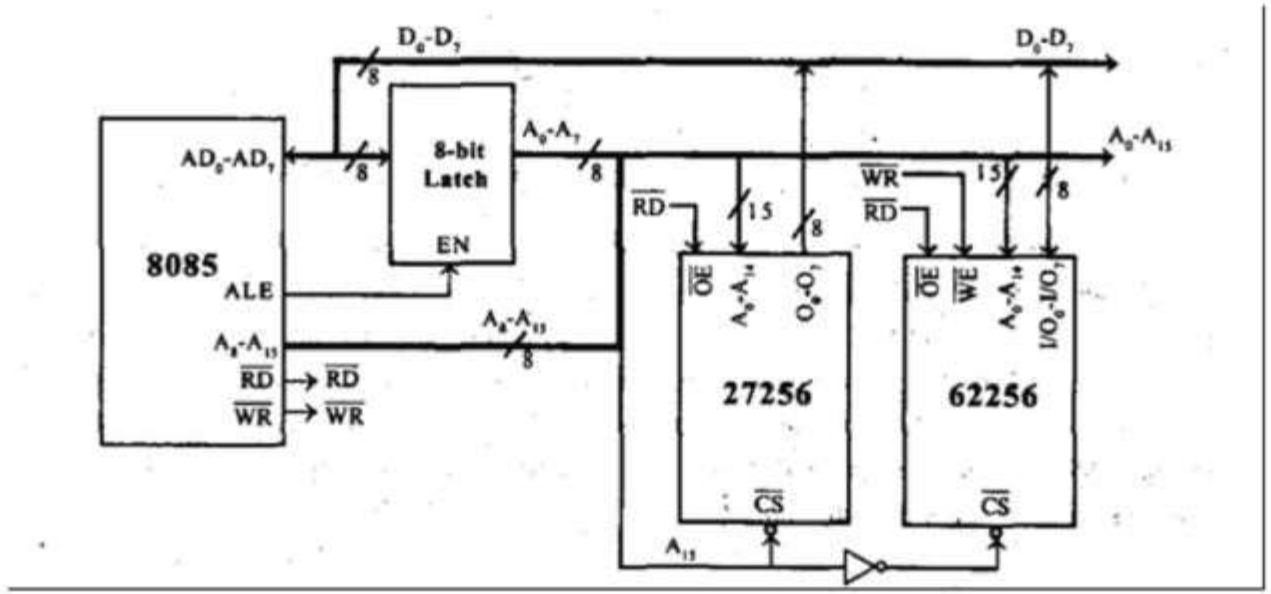


FIGURE 4.20
Schematic of Memory Design for MCTS Project

Interfacing 64Kb EPROM with 8085



Interfacing 32Kb EPROM and 32Kb RAM with 8085



Interfacing 16Kb EPROM and 16Kb RAM with 8085

- EXAMPLE-3

- Consider a system in which 32kb memory space is implemented using four numbers of 8kb memory. Interface the EPROM and RAM with 8085 processor.

- The total memory capacity is 32Kb. So, let two number of 8kb n memory be EPROM and the remaining two numbers be RAM.
- Each 8kb memory requires 13 address lines and so the address lines A0- A12 of the processor are connected to 13 address pins of all the memory.
- The address lines and A13 - A14 can be decoded using a 2-to-4 decoder to generate four chip select signals.
- These four chip select signals can be used to select one of the four memory IC at any one time.
- The address line A15 is used as enable for decoder.
- The simplified schematic memory organization is shown.

Input Output Interfacing 8085 Microprocessor:

Any application of a microprocessor based system requires the transfer of data between external circuitry to the microprocessor and microprocessor to the external circuitry. User can give information to the microprocessor using keyboard and user can see the result or output information from the microprocessor with the help of display device. The transfer of data between keyboard and microprocessor, and microprocessor and display device is called Input Output Interfacing 8085 Microprocessor or I/O data transfer. This data transfer is done with the help of I/O ports.

Input Port :

It is used to read data from the input device such as keyboard. The simplest form of input port is a buffer. The input device is connected to the microprocessor through buffer as shown in the Fig. 4.28. This buffer is a tri-state buffer and its output is available only when enable signal is active.

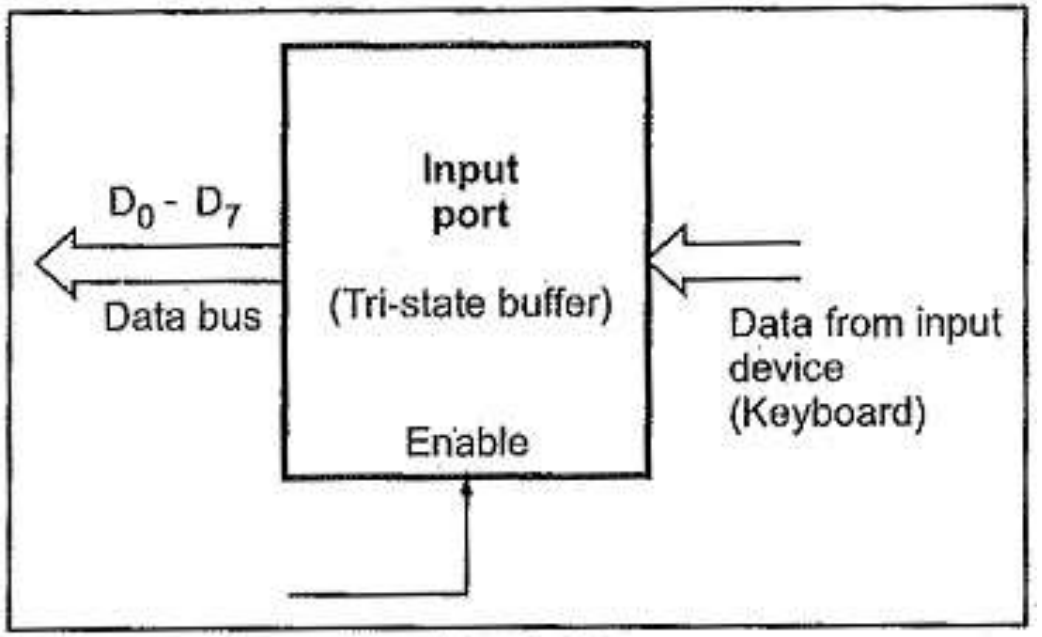


Fig. 4.28

Output Port :

It is used to send data to the output device such as display from the microprocessor. The simplest form of output port is a latch. The output device is connected to the microprocessor through latch as shown in the Fig. 4.29.

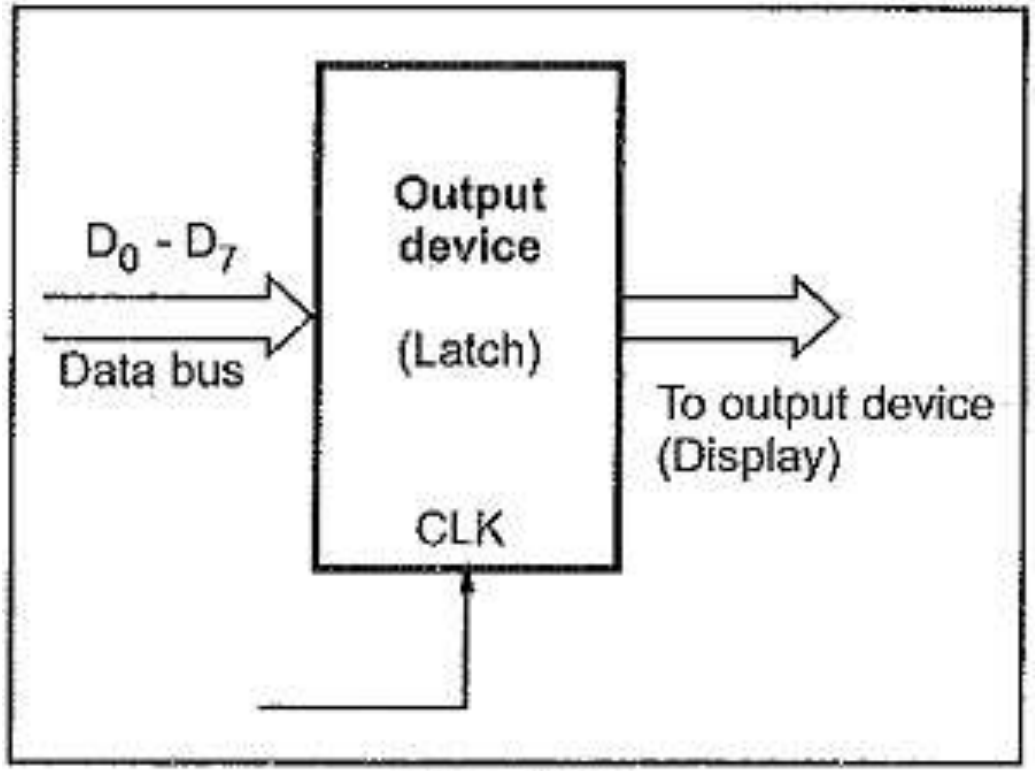


Fig. 4.29

When microprocessor wants to send data to the output device, it puts the data on the data bus and activates the clock signal of the latch, latching the data from the data bus at the output of latch. It is then available at the output of latch for the output device.

I/O Device Selection:

As mentioned earlier, the 8085 gives 8 bit I/O address. This means it can select one of the 256 I/O ports. To select an appropriate I/O device, it is necessary to do following things.

1.Decode the address to generate unique signal corresponding to the device address on the bus.

2.When device address signal and control signal (IOR or IOW) both are low, generate device select signal.

3.Use device select signal to activate the Input Output Interfacing Techniques.

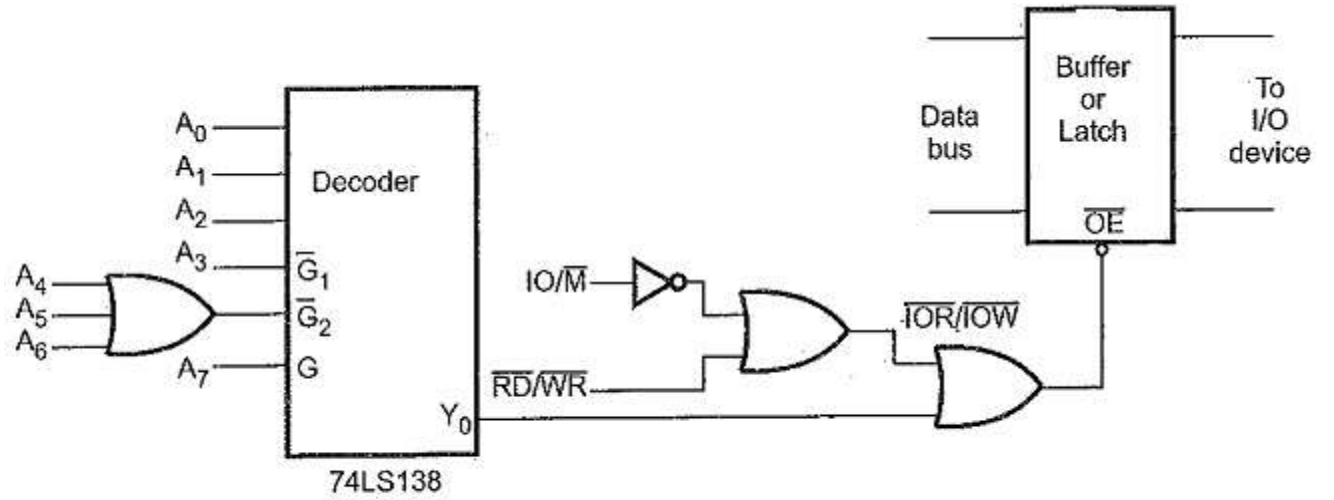


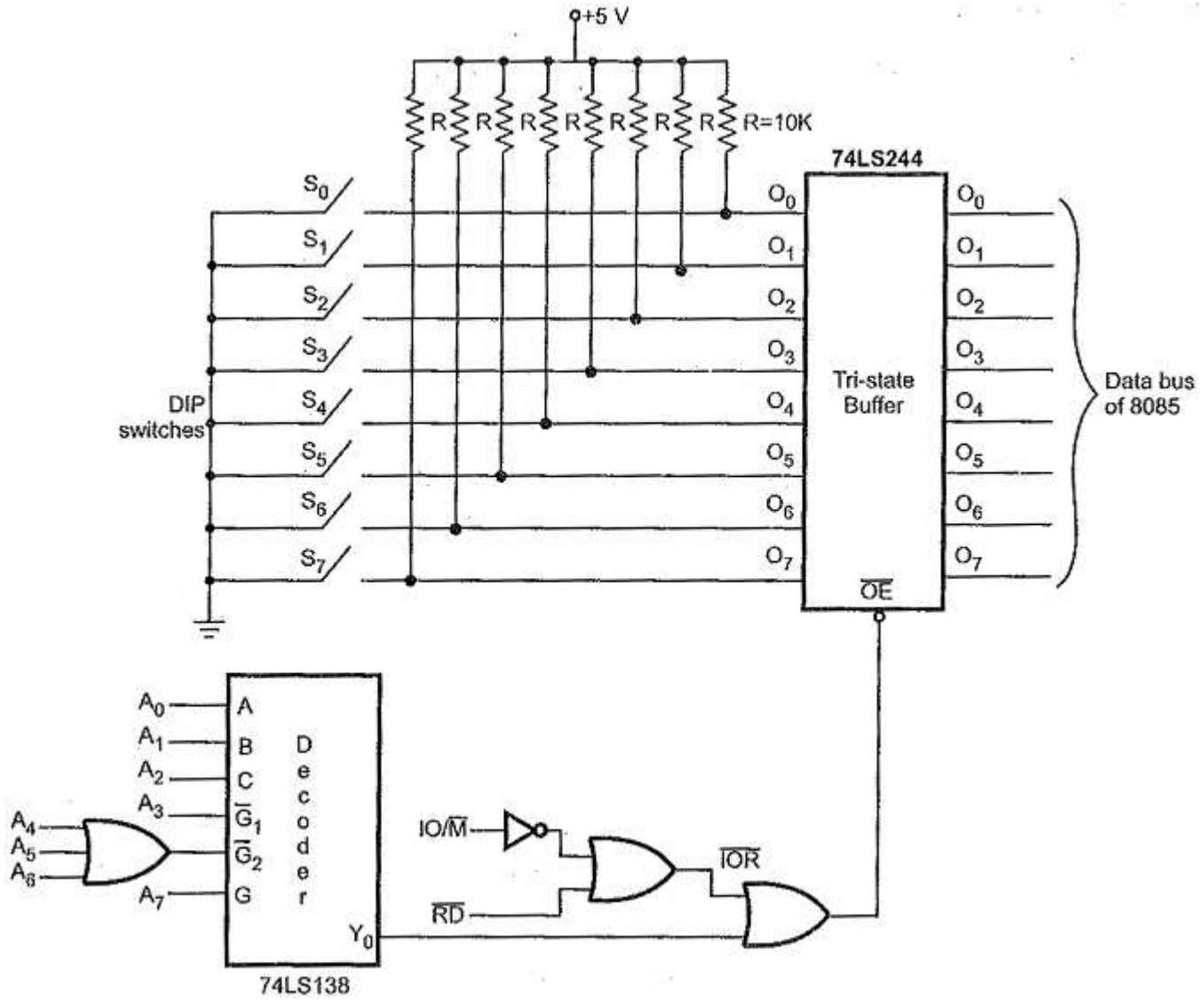
Fig 4.32 Absolute Decoding Circuit for the I/O Device

Interfacing Input Device :

The microprocessor 8085 accepts 8 bit data from the input device such as keyboard, sensors, transducers etc.

Fig. 4.33 (see Fig. on next page) shows the circuit diagram to Input Output Interfacing Techniques (buffer) which is used to read the status of 8 switches.

The address for this input device is 80H as device select signal goes low when address is 80H.



When the switch is in the released position, the status of line is high otherwise status is low.
With this information microprocessor can check a particular key is pressed or not.

Interfacing Output Device :

The microprocessor 8085 sends 8 bit data to the output device such as 7 segment displays, LEDs, printer etc. Fig 4.34 shows the circuit diagram to interface output port (latch) which is used to send the signal for glowing the LEDs. LED will glow when output pin status is low. The IC 74LS138 and 3 input OR gate is used to generate device select signal. The latch enable signal is active high. So NOR gate is used to generate latch enable signal, which goes high when Y_1 and IOW both are low.

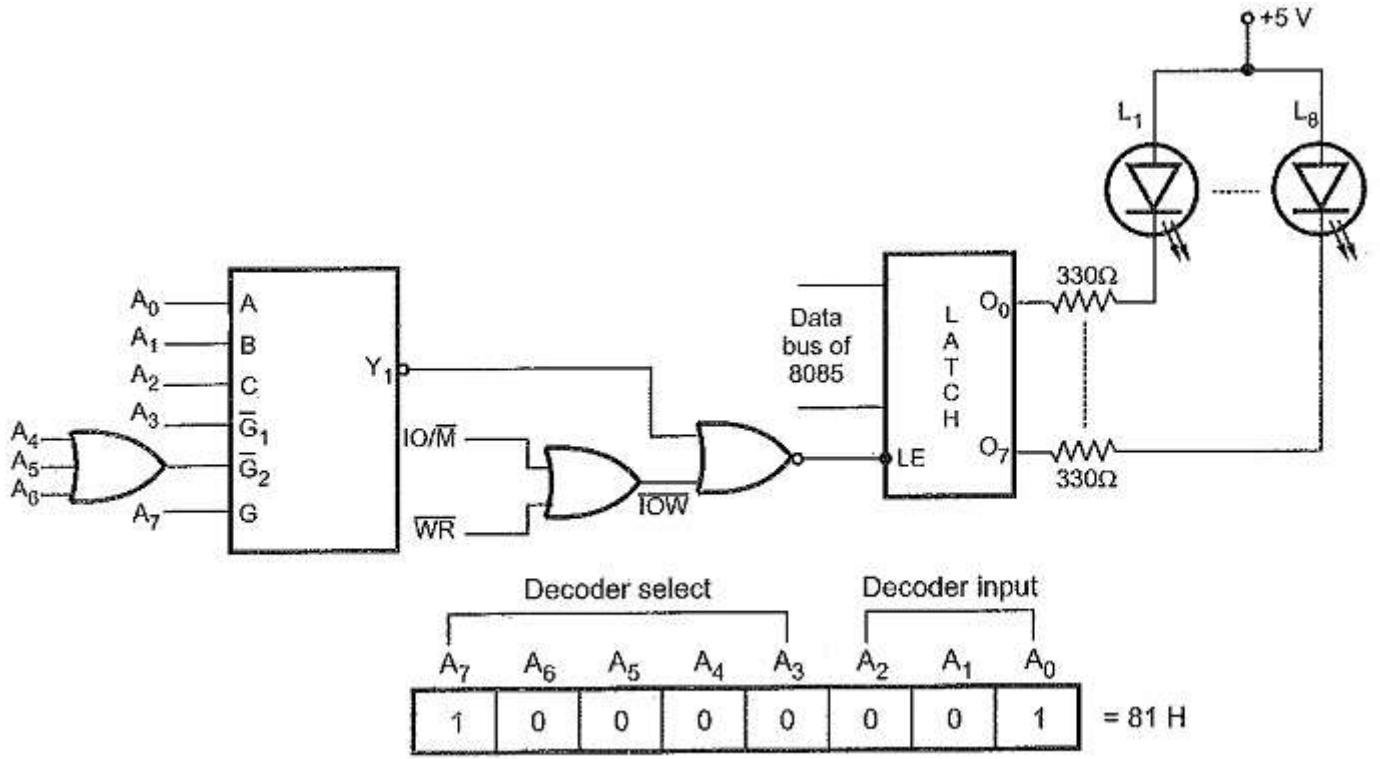


Fig. 4.34 Circuit Diagram to Interface Output Port

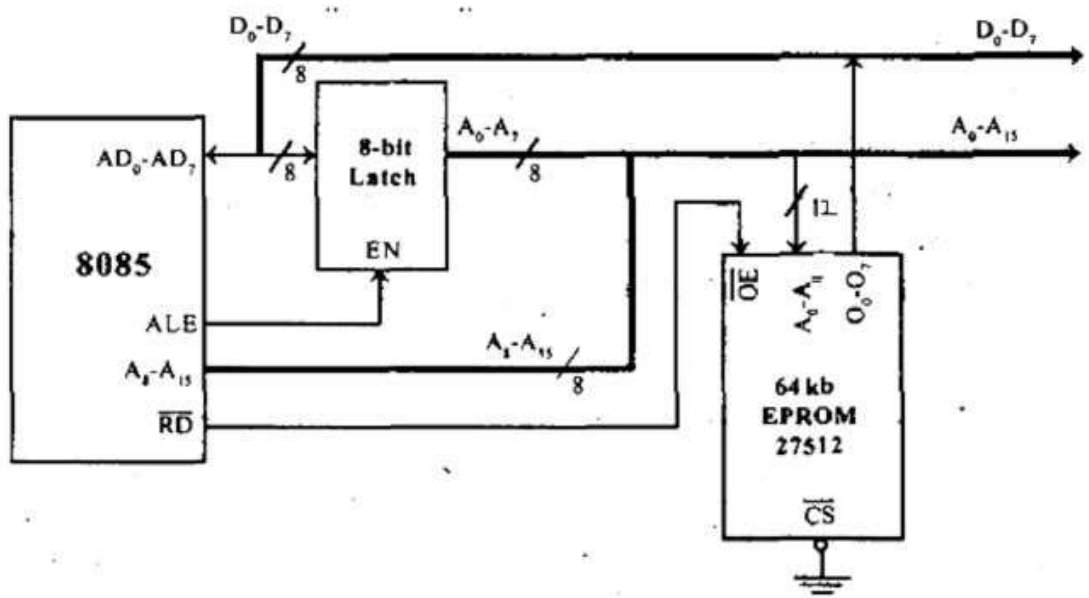
Interfacing 64Kb EPROM with 8085

- EXAMPLE-1

- Consider a system in which the full memory space 64kb is utilized for EPROM memory. Interface the EPROM with 8085 processor.

- The memory capacity is 64 Kbytes. i.e
- 2^n = 64 x 1024 bytes where n = address lines.
- So, n = 16.
- In this system the entire 16 address lines of the processor are connected to address input pins of memory IC in order to address the internal locations of memory.
- The chip select (CS) pin of EPROM is permanently tied to logic low (i.e., tied to ground).
- Since the processor is connected to EPROM, the active low RD pin is connected to active low output enable pin of EPROM.
- The range of address for EPROM is 0000H to FFFFH.

Interfacing 64Kb EPROM with 8085



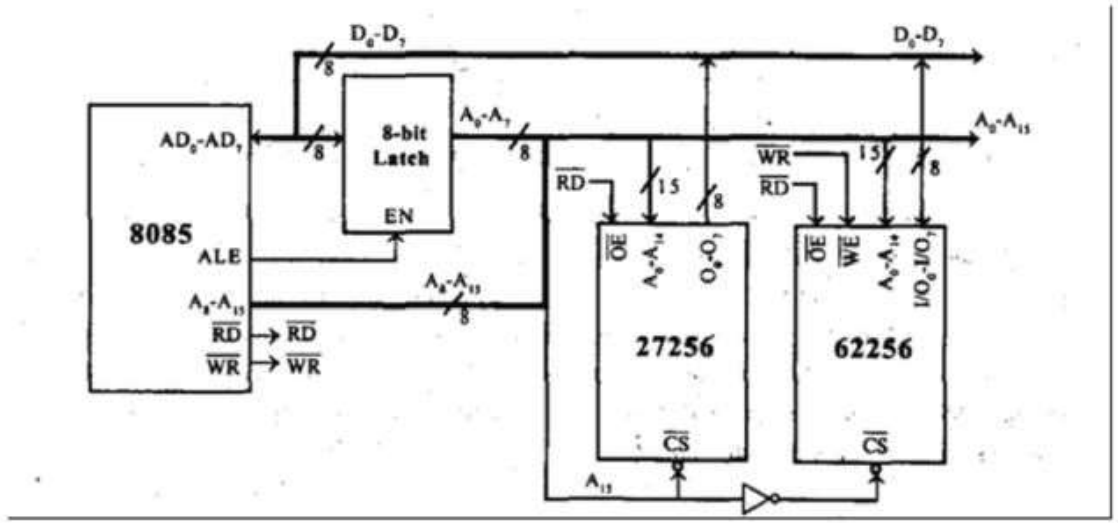
Interfacing 32Kb EPROM and 32Kb RAM with 8085

- EXAMPLE-2

- Consider a system in which the available 64kb memory space is equally divided between EPROM and RAM. Interface the EPROM and RAM with 8085 processor.

- Implement 32kb memory capacity of EPROM using single IC 27256.
- 32kb RAM capacity is implemented using single IC 62256.
- The 32kb memory requires 15 address lines and so the address lines A0 - A14 of the processor are connected to 15 address pins of both EPROM and RAM.
- The unused address line A15 is used as to chip select. If A15 is 1, it select RAM and If A15 is 0, it select EPROM.
- Inverter is used for selecting the memory.
- The memory used is both Ram and EPROM, so the low RD and WR pins of processor are connected to low WE and OE pins of memory respectively.
- The address range of EPROM will be 0000H to 7FFFH and that of RAM will be 8000H to FFFFH.

Interfacing 32Kb EPROM and 32Kb RAM with 8085



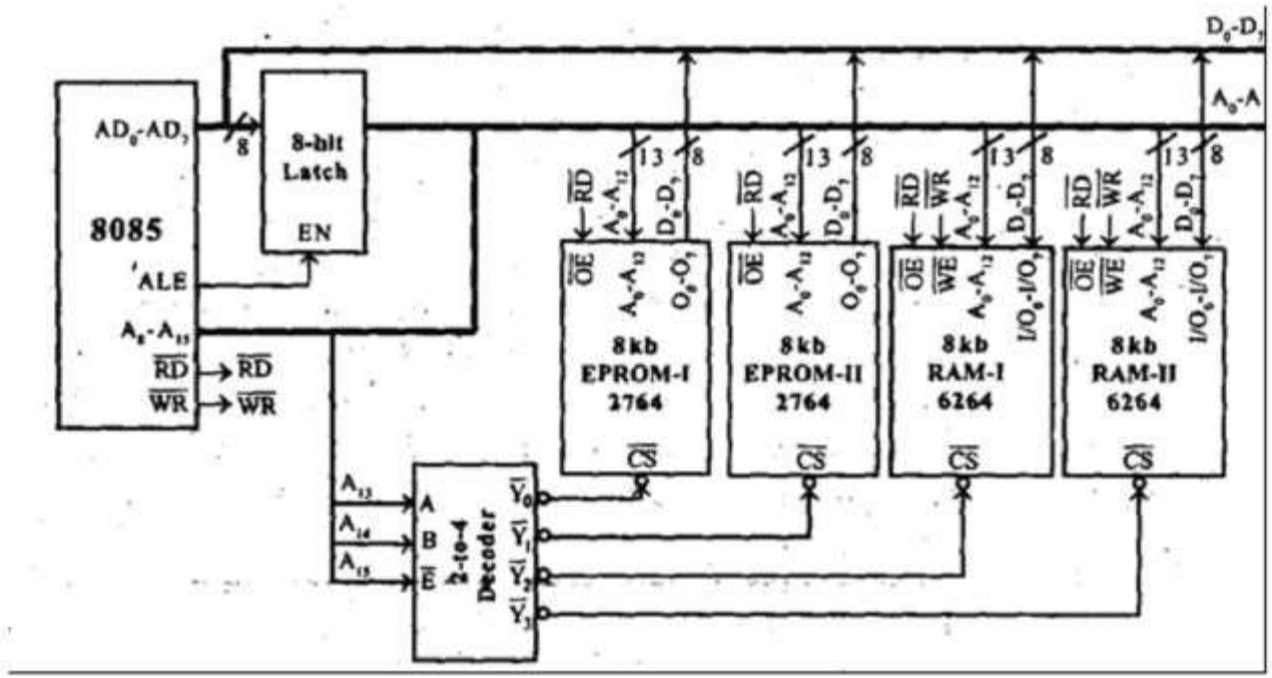
Interfacing 16Kb EPROM and 16Kb RAM with 8085

- EXAMPLE-3

- Consider a system in which 32kb memory space is implemented using four numbers of 8kb memory. Interface the EPROM and RAM with 8085 processor.

- The total memory capacity is 32Kb. So, let two number of 8kb n memory be EPROM and the remaining two numbers be RAM.
- Each 8kb memory requires 13 address lines and so the address lines A0- A12 of the processor are connected to 13 address pins of all the memory.
- The address lines and A13 - A14 can be decoded using a 2-to-4 decoder to generate four chip select signals.
- These four chip select signals can be used to select one of the four memory IC at any one time.
- The address line A15 is used as enable for decoder.
- The simplified schematic memory organization is shown.

Interfacing 16Kb EPROM and 16Kb RAM with 8085



Interfacing 16Kb EPROM and 16Kb RAM with 8085

Device	Binary address																Hexa address
	Decoder enable/input			Input to address pins of memory IC													
	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	
8kb EPROM - I	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0000
	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0001
	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0002
	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1FFF
8kb EPROM - II	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	2000
	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	1	2001
	0	0	1	0	0	0	0	0	0	0	0	0	0	0	1	0	2002
	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	3FPF
8kb RAM - I	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	4000
	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	1	4001
	0	1	0	0	0	0	0	0	0	0	0	0	0	0	1	0	4002
	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
	0	1	0	1	1	1	1	1	1	1	1	1	1	1	1	1	5FFF
8kb RAM -II	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	6000
	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	1	6001
	0	1	1	0	0	0	0	0	0	0	0	0	0	0	1	0	6002
	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	7FFF

Interfacing 4 no. 8Kb EPROM and 4 no. 8Kb RAM with 8085

- EXAMPLE-4

- Consider a system in which the 64kb memory space is implemented using eight numbers of 8kb memory. Interface the EPROM and RAM with 8085 processor.

- The total memory capacity is 64Kb. So, let 4 numbers of 8Kb EPROM and 4 numbers of 8Kb RAM.
- Each 8kb memory requires 13 address lines. So the address line A0 - A12 of the processor are connected to 13address pins of all the memory lCs.
- The address lines A13, A14 and A]5 are decoded using a 3-to-8 decoder to generate eight chip select signals. These eight chip select signals can be used to select one of the eight memories at any one time.
- The memory interfacing is shown in following figure.

Interfacing 4 no. 8Kb EPROM and 4 no. 8Kb RAM with 8085

Memory IC chip	Binary address																Hexa address
	Decoder input			Input to memory address pins													
	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	
EPROM I	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0000
	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0001
	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0002
	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1FFF
EPROM II	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	2000
	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	1	2001
	0	0	1	0	0	0	0	0	0	0	0	0	0	0	1	0	2002
	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	3FFF
EPROM III	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	4000
	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	1	4001
	0	1	0	0	0	0	0	0	0	0	0	0	0	0	1	0	4002
	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
	0	1	0	1	1	1	1	1	1	1	1	1	1	1	1	1	5FFF
RAM I	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	6000
	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	1	6001
	0	1	1	0	0	0	0	0	0	0	0	0	0	0	1	0	6002
	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	7FFF
RAM II	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	8000
	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	8001
	1	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	8002
	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
	1	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	9FFF
RAM III	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	A000
	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	1	A001
	1	0	1	0	0	0	0	0	0	0	0	0	0	0	1	0	A002
	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
	1	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	BFFF
RAM IV	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	C000
	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	1	C001
	1	1	0	0	0	0	0	0	0	0	0	0	0	0	1	0	C002
	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
	1	1	0	1	1	1	1	1	1	1	1	1	1	1	1	1	DFFF
RAM V	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	E000
	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	1	E001
	1	1	1	0	0	0	0	0	0	0	0	0	0	0	1	0	E002
	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	FFFF

Comparison Between Memory Mapped I/O and I/O Mapped I/O:

Memory mapped I/O	I/O mapped I/O
1. In this device address is 16 bit. Thus A_0 to A_{15} lines are used to generate device address.	1. In this I/O device address is 8 bit. Thus A_0 to A_7 or A_8 to A_{15} lines are used to generate device address.
2. $\overline{MEMR}$ and $\overline{MEMW}$ control signals are used to control read and write I/O operations.	2. $\overline{IOR}$ and $\overline{IOW}$ control signals are used to control read and write I/O operations.
3. Instructions available are LDA addr, STA addr, LDAX rp, STAX rp, MOV M,R, MOV R,M ADD M, CMP M etc.	3. Instructions available are IN and OUT.
4. Data transfer is between any register and I/O device.	4. Data transfer is between accumulator and I/O device.
5. Maximum number of I/O devices are 65536 (theoretically).	5. Maximum number of I/O devices are 256.
6. Execution speed using LDA addr, STA addr is 13 T-state and 7 T-states for MOV M, r and MOV r, M instructions.	6. Execution speed is 10 T-states.
7. Decoding 16 bit address may require more hardware.	7. Decoding 8 bit address will require less hardware.

Timing Diagram

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 .

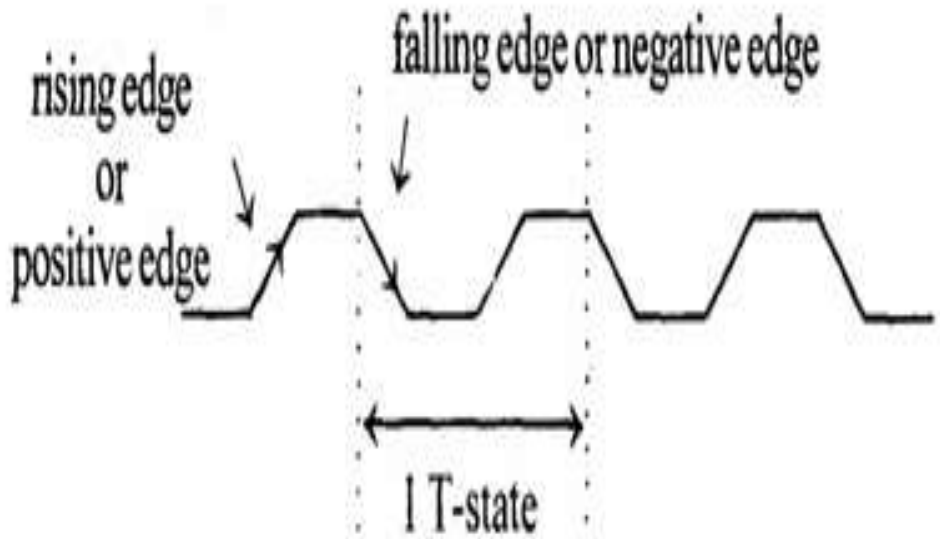
Machine Cycle:

The time required to access the memory or input/output devices .

T-State:

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

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



Timing diagrams

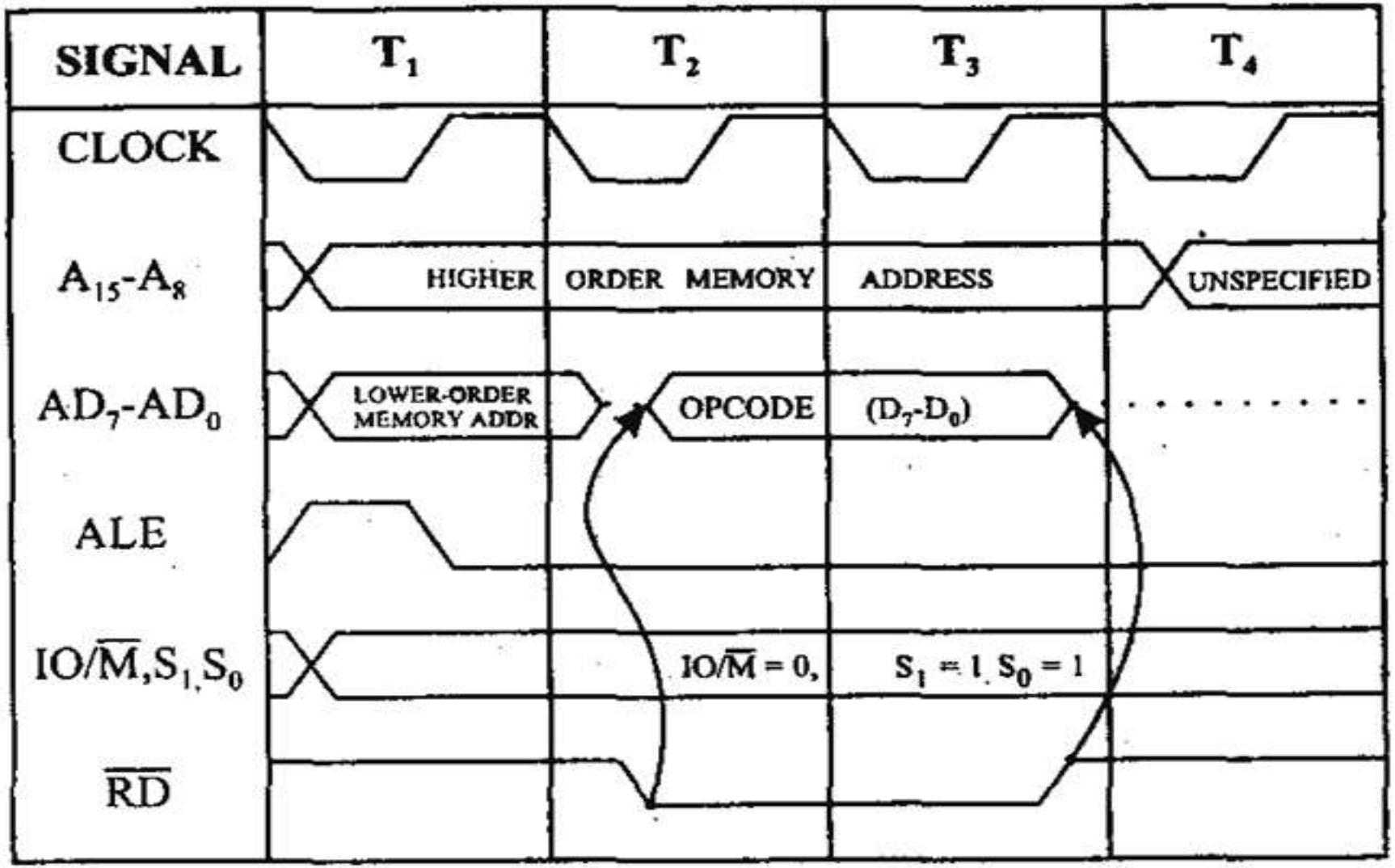
- The 8085 microprocessor has 7 basic machine cycle. They are

1. Op-code Fetch cycle(4T or 6T).
2. Memory read cycle (3T)
3. Memory write cycle(3T)
4. I/O read cycle(3T)
5. I/O write cycle(3T)
6. Interrupt Acknowledge cycle(6T or 12T)
7. Bus idle cycle

Machine Cycle	Status			No. of Machine cycles	Control
	IO/$\overline{M}$	S1	S0		
Opcode Fetch	0	1	1	4	$\overline{RD}$ =0
Memory Read	0	1	0	3	$\overline{RD}$ =0
Memory Write	0	0	1	3	$\overline{WR}$ =0
I/O Read	1	1	0	3	$\overline{RD}$ =0
I/O Write	1	0	1	3	$\overline{WR}$ =0
INTR Acknowledge	1	1	1	3	$\overline{INTA}$ =0

- Following Buses and Control Signals must be shown in a Timing Diagram:
- Higher Order Address Bus.
- Lower Address/Data bus
- ALE
- RD
- WR
- IO/M

Opcode fetch cycle(4T or 6T)



Opcode Fetch

- The Opcode fetch cycle, fetches the instructions from memory and delivers it to the instruction register of the microprocessor
- Opcode fetch machine cycle consists of **4 T-states**.

T1 State:

During the T1 state, the contents of the program counter are placed on the 16 bit address bus. The higher order 8 bits are transferred to address bus (A8-A15) and lower order 8 bits are transferred to multiplexed A/D (AD0-AD7) bus.

ALE (address latch enable) signal goes **high**. As soon as ALE goes high, the memory latches the AD0-AD7 bus. At the middle of the T state the **ALE goes low**

T2 State:

During the beginning of this state, the **RD' signal goes low** to enable memory. It is during this state, the selected memory location is placed on D0-D7 of the Address/Data multiplexed bus.

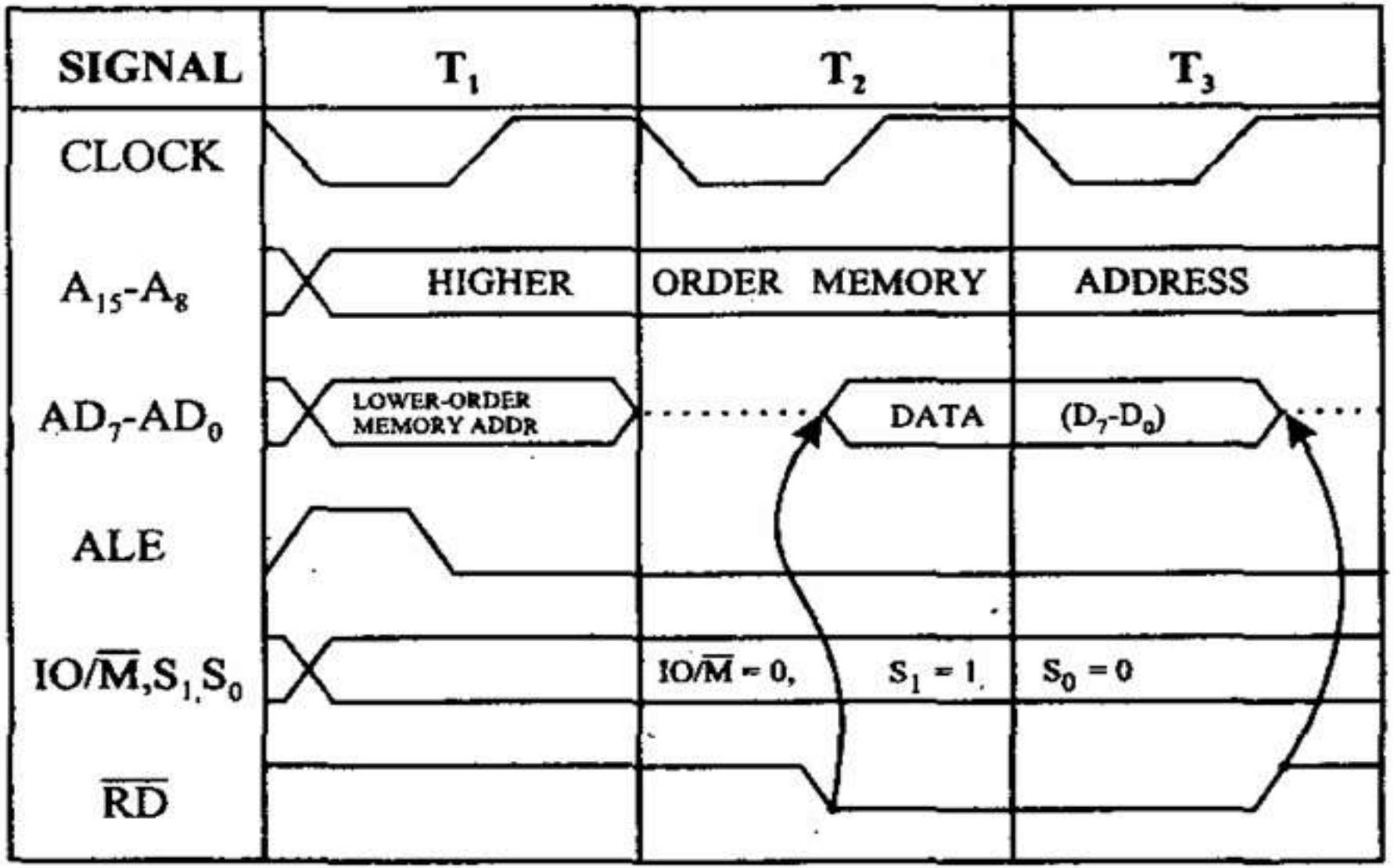
T3 State:

In the previous state the Opcode is placed in D0-D7 of the A/D bus. In this state of the cycle, the Opcode of the A/D bus is transferred to the instruction register of the microprocessor. Now the **RD' goes high** after this action and thus disables the memory from A/D bus.

T4 State:

In this state the Opcode which was fetched from the memory is decoded.

Memory read cycle (3T)



- These machine cycles have 3 T-states.

T1 state:

- The higher order address bus **(A8-A15)** and lower order address and data multiplexed (AD0-AD7) bus. **ALE goes high** so that the memory latches the (AD0-AD7) so that complete 16-bit address are available.

  The mp identifies the memory read machine cycle from the status signals **IO/M'=0, S1=1, S0=0**. This condition indicates the memory read cycle.

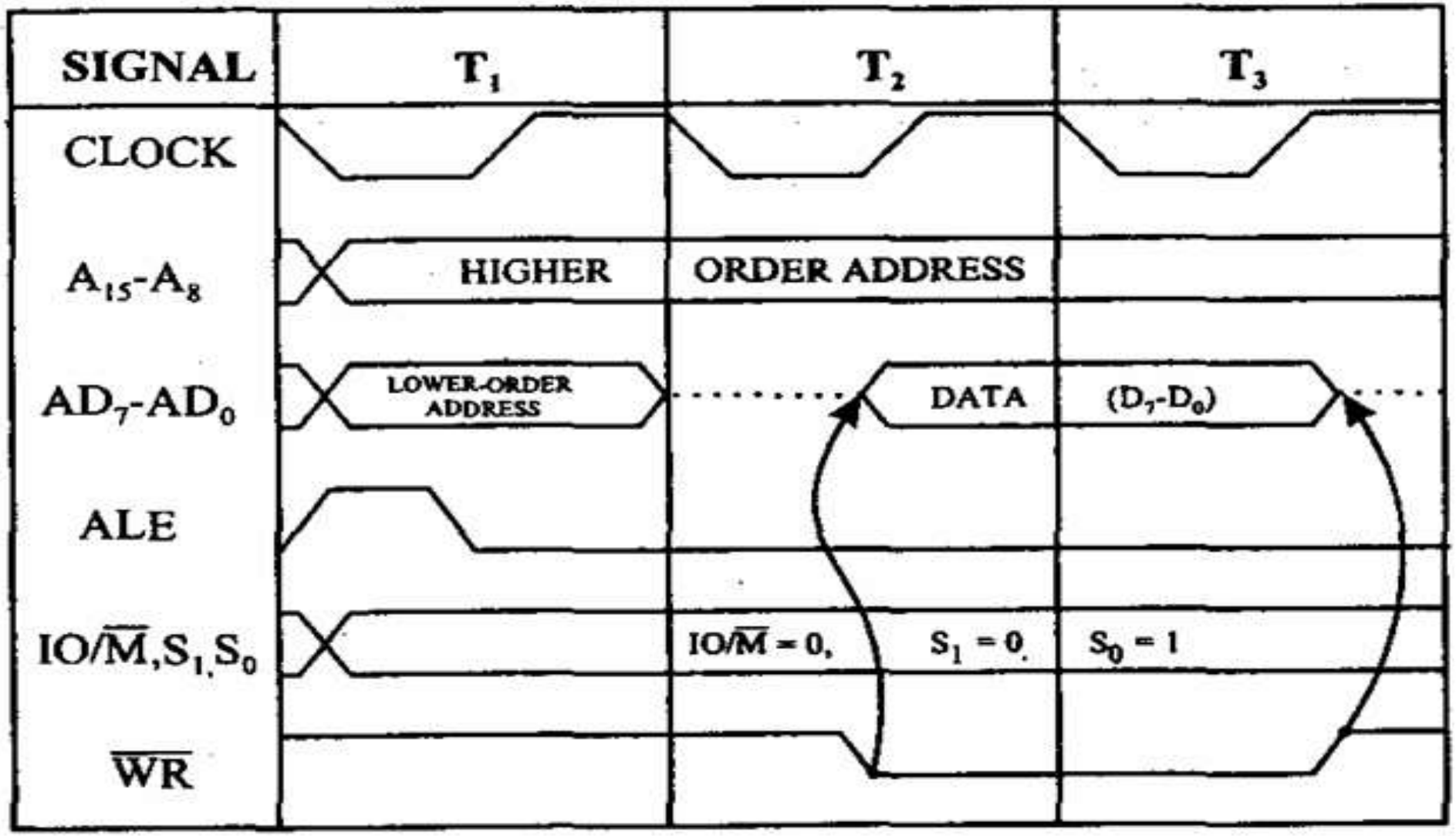
T2 state:

- Selected memory location is placed on the (D0-D7) of the A/D multiplexed bus. RD' goes **LOW**

T3 State:

- The data which was loaded on the previous state is transferred to the microprocessor. In the middle of the T3 state RD' goes high and disables the memory read operation. The data which was obtained from the memory is then decoded.

Memory write cycle (3T)



- These machine cycles have 3 T-states.

T1 state:

- The higher order address bus **(A8-A15)** and lower order address and data multiplexed (AD0-AD7) bus. **ALE goes high** so that the memory latches the (AD0-AD7) so that complete 16-bit address are available.

  The mp identifies the memory read machine cycle from the status signals **IO/M'=0, S1=0, S0=1**. This condition indicates the memory read cycle.

T2 state:

- Selected memory location is placed on the (D0-D7) of the A/D multiplexed bus. WR' goes **LOW**

T3 State:

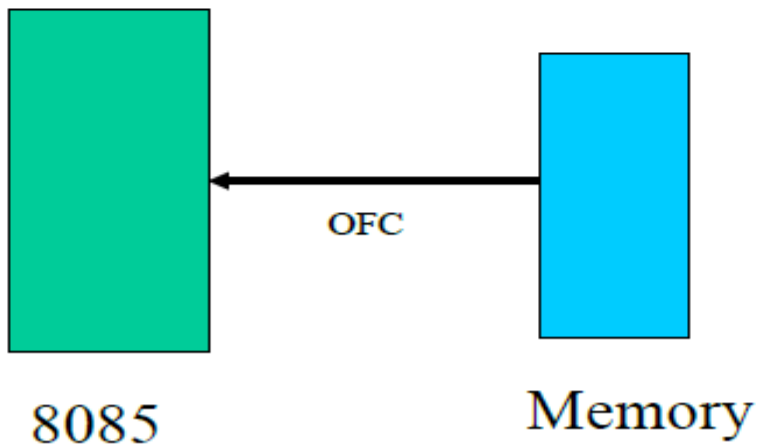
- In the middle of the T3 state WR' goes high and disables the memory write operation. The data which was obtained from the memory is then decoded.

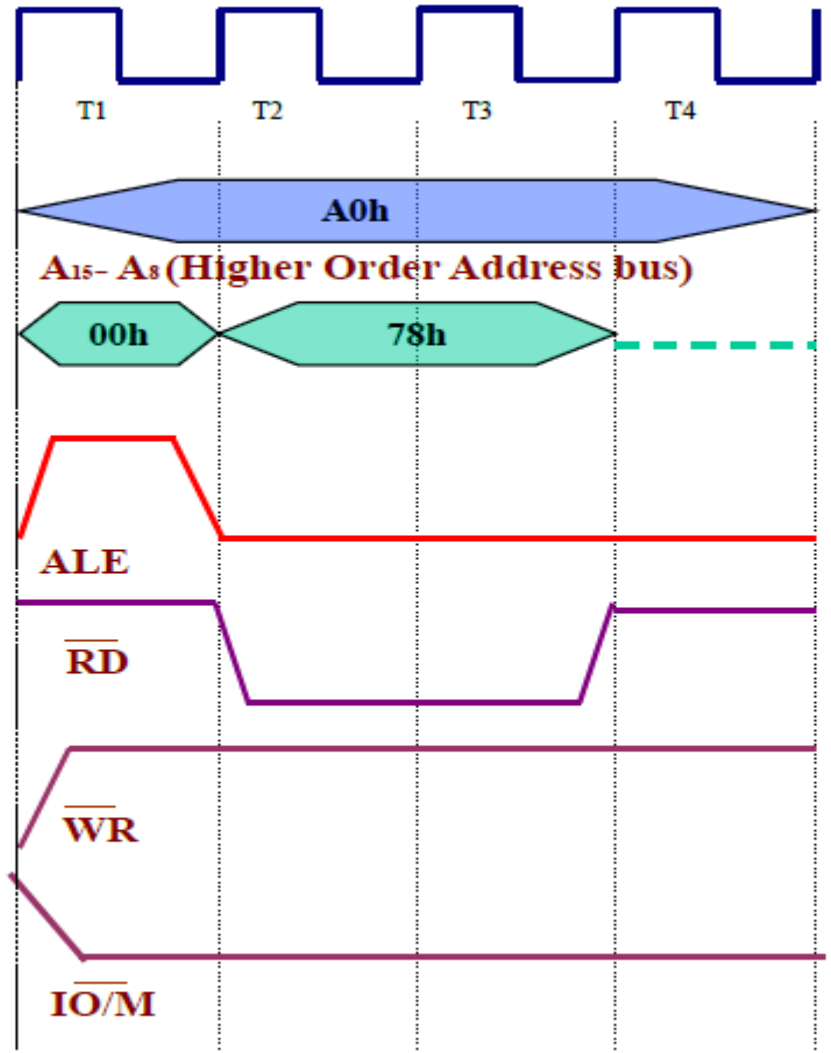
Instruction:

A000h MOV A,B

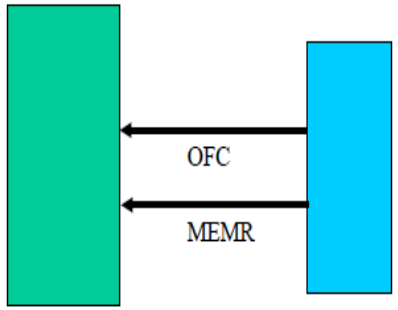
Corresponding Coding:

A000h 78





Op-code fetch Cycle



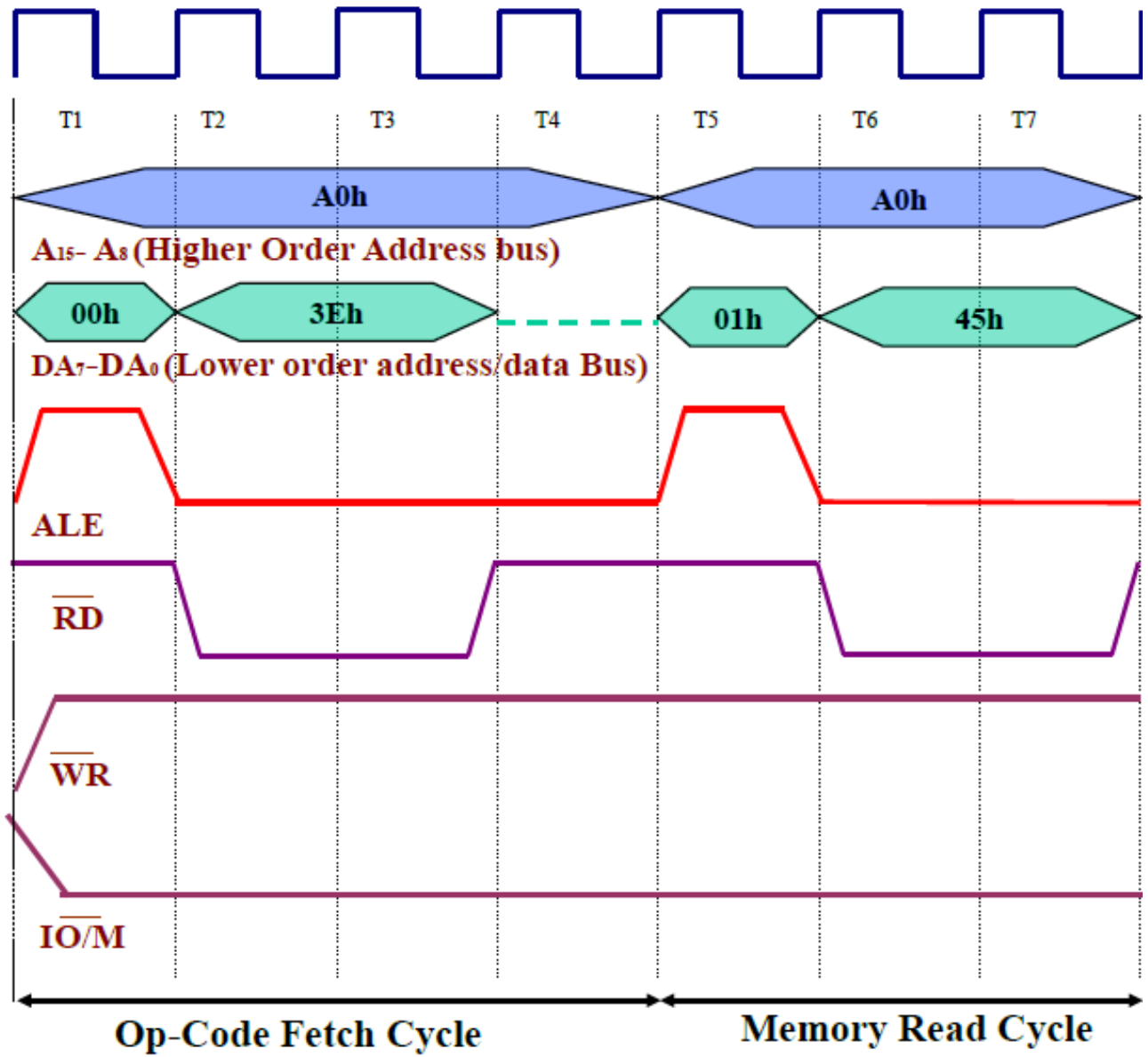
Instruction:

A000h MVI A,45h

Corresponding Coding:

A000h 3E

A001h 45



Instruction:

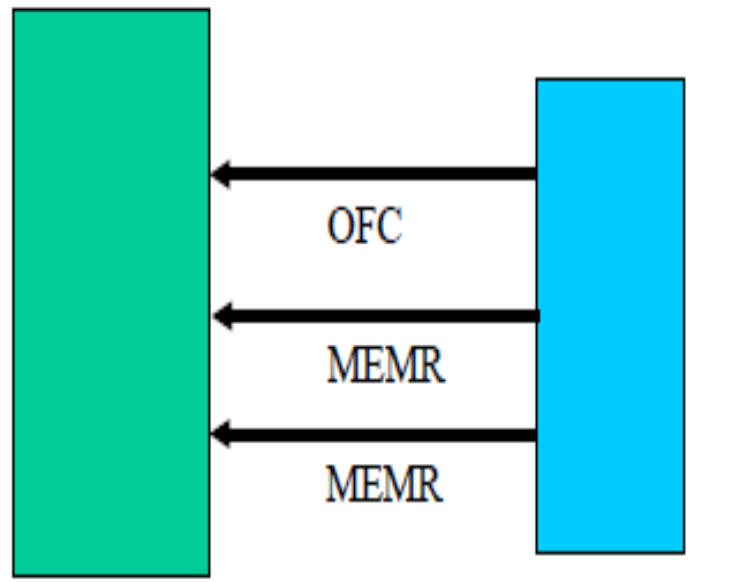
A000h LXI A,FO45h

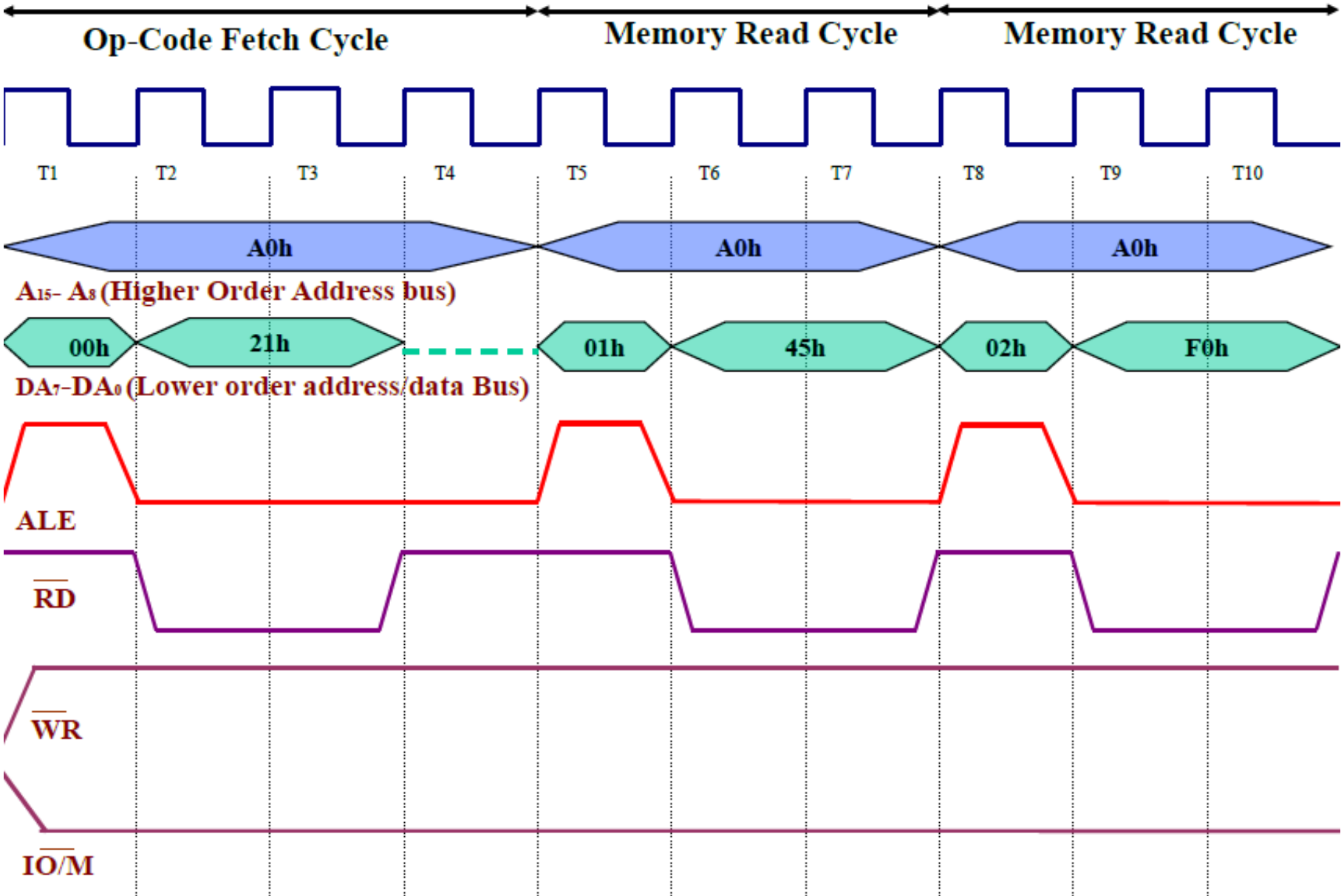
Corresponding Coding:

A000h 21

A001h 45

A002h F0




Op-Code Fetch Cycle
Memory Read Cycle
Memory Read Cycle
T1
T2
T3
T4
T5
T6
T7
T8
T9
T10
A0h
A0h
A0h
A15- A8 (Higher Order Address bus)
00h
21h
01h
45h
02h
F0h
DA7-DA0 (Lower order address/data Bus)
ALE
RD
WR
IO/M

Addressing Modes

- Immediate (MOV A,B ;ADD B; SUB E;ANA C)
- Register (MVI A,05H;LXI B, 20AEH; ADI 05H;ORI 07H)
- Direct (LDA 4500H;STA 7500H;IN 09H;OUT 70H)
- Indirect (MOV A, M;MOV M,A;ADD M;ORA M)
- Implied(implicit) (HLT; NOP;RST;RET)

Addressing modes of 8085

The way operand is specified in instruction is called addressing mode.
8085 has following addressing modes:

1.Immediate addressing :- In this mode, 8 or 16 bit data is specified in the instruction itself as its one of the operands.
Normally, instructions have "I" in their mnemonics.
Eg: MVI A,33H.

2.Register addressing :- In this mode, operands are microprocessors registers only i.e. operations is performed within various registers.
Eg: MOV C,B.

3.Direct addressing :- In this mode, one of the operands is data stored in memory. The memory address of operand (data) is directly given in instruction itself. Eg: STA 3050H.

4.Indirect addressing :- In this mode, one of the operands is data stored in memory. The memory address of operand (data) is specified by register pair.
Eg: LDAX D.

5.**Implied/ Inherent/ Implicit addressing** :- This mode does not require any operand, the data is specified by opcode itself. Eg: CMA

8085 microprocessor instruction set has 74 operation codes that result in 246 instructions. 8085 instructions can be classified into following five functional groups:

Data Transfer/ Copy Operations - These types of instructions transfer data from source to destination, without modifying the contents of the source. Data Transfer can take place between registers, between a memory location and a register, between an I/O device and accumulator.

MOV : This instruction copies content of source register (Rs) into destination register (Rd). The contents of source register are not altered. If one of operand is memory location, its location is specified by contents of HL pair.
Eg: MOV A,B.
MVI : The 8 bit data is stored in destination register or memory.
Eg: MVI B,57H.
LDA : It is called load accumulator. The content of memory location specified by 16 bit address in operand are copied to accumulator. The contents of source are not altered. Eg: LDA 2014.
LXI : It is called load register pair immediate. It loads 16 bit data in register pair designated in operand. Eg: LXI H,2034.
LDAX : It is load accumulator indirect. The contents of register pair points to a memory location. This instruction copies contents of memory location to accumulator. The contents of either register pair or memory are not altered. Eg: LDAX B.

Data Transfer/ Copy Operations CONTD...,

LHLD : It is load H&L registers directly. The instruction copies the content of memory location pointed by 16-bit address into register L and copies content of next memory location into register H. The contents of source memory are not altered. Eg: LHLD 2040.

STA : It is store Accumulator Direct. The contents of accumulator are copied into memory location specified by operand. Eg: STA 4350H.

STAX : It is store Accumulator Indirect. The contents of accumulator are copied into memory location specified by contents of operand. Eg: STAX B.

SHLD : Store HL pair directly. The contents of register L are stored into memory location specified by 16 bit address and contents of register H are stored in next memory location. Eg: SHLD 2470.

XCHG : Exchange H&L with D&E. The contents of H are exchanged with contents of register D and contents of register L are exchanged with contents of register E. Eg: XCHG.

Arithmetic Operations - These instructions are responsible for performing arithmetic operations like addition, subtraction, increment and decrement. Addition includes addition between any 8 bit numbers / the contents of register / the contents of memory location and contents of accumulator and the sum will be stored in accumulator. Same process is done in case of subtraction. Increment/ Decrement means 8 bit number/ contents of a register or a memory location can be incremented/ decremented by one and same is the case with 16 bits contents of register pair.

ADD : The contents of operand (register or memory) are added to contents of accumulator. All flags are modified to reflect the result of addition.
Eg: ADD B.

ADC : The contents of operand & carry flag are added to contents of accumulator and result is stored in accumulator. Eg: ADC B.

ADI : It is Add immediate to Accumulator. The 8 bit data (operand) is added to contents of accumulator and the result is stored in accumulator.
Eg: ADI 45.

ACI : The 8 bit data and carry flag are added to contents of accumulator and result is stored in accumulator itself. Eg: ACI 45.

DAD : The 16 bit contents of specified register pair are added to contents of HL pair and the sum is stored in HL register pair. If result is greater than 16 bit, carry flag is set and no other flags are modified. Eg: DAD B.

SUB : The contents of operand (register or memory) are subtracted from accumulator. If operand is memory, its location is specified by contents of HL register pair. All flags are modified. (Common to all) Eg: SUB M.
SBB : The contents of operand and borrow flag are subtracted from contents of accumulator. Eg: SBB C.
SUI : 8 bit data is subtracted from contents of accumulator. Eg: SUI 45.
SBI : The 8 bit data and borrow flag are subtracted from contents of accumulator and result is stored in accumulator. Eg: SBI 45.
INR : The contents of designated register or memory are incremented by 1. Result is stored in same place. Eg: INR B.
INX : The contents of designated register is incremented by 1. Result is stored in same place. Eg: INX B.
DCR : The contents of designated register or memory are decremented by 1. Result is stored in same place. Eg: DCR C.
DCX : The contents of designated register is decremented by 1. Result is stored in same place. Eg: DCX B.
DAA : It is Decimal Adjust Accumulator. The contents of accumulator are changed from a binary value to two 4 bit BCD digit. It uses auxiliary flag to perform BCD conversion. All flags are modified. If value of lower order 4 bits in accumulator is greater than 9 or if Auxiliary Carry is set, the instruction adds 6 to lower order 4 bit.

Logical Operations - These type of instructions are used for performing logical operations like AND, OR, EX-OR, rotate, compare and complement with the contents of accumulator. Results are stored in accumulator.

- **CMP** : Compare contents of register or memory with accumulator by performing subtraction. Eg: CMP B.
- **CPI** : Compare immediate data with accumulator. Eg: CPI 45.
- **ANA** : Logical AND register or memory with accumulator. Eg: ANA B.
- **ANI** : Logically AND the contents of accumulator & 8 bit data and result is stored in accumulator. Eg: ANI 24.
- **XRA** : Exclusive OR the contents of memory or register with accumulator. All flags are modified and CY, AC are reset. Eg: XRA B.
- **XRI** : 8 bit data/ operand is exclusicve ORed with accumulator and stored in accumulator. Eg: XRI 45.
- **ORA** : Logically ORed contents of accumulator with register or memory. All flags are modified. Eg: ORA M.
- **ORI** : Logically ORed 8 bit data with contents of accumulator. Eg: ORI 45.

- **RLC** : Each bit of accumulator is rotated left by one position. CY is modified according to bit D7 and remaining not affected.
- **RRC** : Each bit of accumulator is rotated right by one position. CY is modified according to bit D0 and remaining not affected.
- **RAL** : Each bit of accumulator is rotated left through carry flag. Bit D7 is placed in carry flag and carry flag is placed at D0. Remaining flag are not modified.
- **RAR** : Each bit of accumulator is rotated right through carry flag. Bit D0 is placed in carry flag and carry flag is placed at D7. Remaining flag are not modified.
- **CMA** : The contents of accumulator are complemented. This means 0 is changed to 1 or vice-versa.
- **CMC** : The contents of carry flag are complemented. This means reset(0)-> set(1).
- **STC** : The carry flag is set to 1 & remaining are unaffected.

Branching Control Operations - These types of instructions can change the sequence of program either conditionally or unconditionally. Conditional instructions can make the program to jump to that particular memory location if condition is satisfied while unconditional instructions will always make the program to jump as soon as the instruction arrives.

- **JMP** : It is Jump Unconditionally. The program sequence is transferred to memory location specified by 16 bit address. Eg: JMP 2034.
- **Jump Conditionally** : These are JC (jump on carry), JNC (jump on no carry, CY=0), JP (jump on positive, S=0), JM (jump on negative, S=1), JZ (jump on zero, Z=1), JNZ (jump on no zero, Z=0), JPE (jump on parity even, P=1), JPO (jump on parity odd, P=0). Eg: JC2043
- **CALL** : It in Unconditional subroutine call. The program sequence is transferred to memory location specified by 16 bit address. Before the transfer, the address of next instruction after CALL is stored/pushed into stack. Eg: CALL 2034.
- **Call Conditionally** : These are CC, CNC, CP, CM, CZ, CNZ, CPE, CPO. Eg: CC 2034.
- **RET** : Return from subroutine unconditionally
- **Return Conditionally** : These are RC, RNC, RP, RM, RZ, RNZ, RPE, RPO.

Stack, I/O & Machine Control Operations - These instructions communicate with Input/ Output Ports and control machine functions such as Halt, Interrupt, or do nothing. In this port address of Input & Output devices are exchanged with accumulator.

- **IN** : Input data to accumulator from port with 8 bit address. The contents of input port designated by 8 bit address (operand) are read and loaded into accumulator. Eg: IN 82.
- **OUT** : Output data from accumulator to port with 8 bit address. Contents of accumulator are copied into I/O port specified by operand. Eg: OUT 87.
- **SPHL** : It copies HL registers to stack pointers. The instruction loads of contents of H & L register into stack pointer register. H to higher order address and L to lower order address.
- **XTHL** : The contents of L register are exchanged with stack location pointed out by contents of stack pointer register. The contents of H register are exchanged with next stack location. The contents of source register are not altered.
- **PUSH** : The contents of register pair are copied onto stack. The stack pointer is decremented & contents of higher order register are copied onto that location. The stack pointer is decremented by 1 and contents of lower order register are copied to that location. Eg: PUSH B.
- **POP** : The contents of memory location pointed out by stack pointer register are copied to lower order register. The stack pointer is incremented by 1 & contents of that memory location are copied to higher order register. The stack pointer is again incremented by 1. Eg: POP H.

What is a Subroutine is assembly language?

A subroutine is a small program written separately from the main program to perform a particular task that you may repeatedly require in the main program. Essentially, the concept of a subroutine is that it is used to avoid the repetition of smaller programs.

Subroutines are written separately and are stored in a memory location that is different from the main program. You can call a subroutine multiple times from the main program using a simple CALL instruction.

What are the conditional CALL statements in assembly language?

You can use conditional CALL statements, too, according to your needs. These statements enter a subroutine only when a certain condition is met.
The subroutine can be exited from using a return (RET) instruction.

CC	Call at address if cy (carry flag) = 1
CNC	Call at address if cy (carry flag) = 0
CZ	Call at address if ZF (zero flag) = 1
CNZ	Call at address if ZF (zero flag) = 0
CPE	Call at address if PF (parity flag) = 1
CPO	Call at address if PF (parity flag) = 0
CN	Call at address if SF (signed flag) = 1
CP	Call at address if SF (signed flag) = 0

RC	Return from subroutine if cy (carry flag) = 1
RNC	Return from subroutine if cy (carry flag) = 0
RZ	Return from subroutine if ZF (zero flag) = 1
RNZ	Return from subroutine if ZF (zero flag) = 0
RPE	Return from subroutine if PF (parity flag) = 1
RPO	Return from subroutine if PF (parity flag) = 0
RN	Return from subroutine if SF (signed flag) = 1
RP	Return from subroutine if SF (signed flag) = 0

Interrupts

- Interrupt is a process where an external device can get the attention of the microprocessor.
  - The process starts from the I/O device
  - The process is asynchronous.
- Classification of Interrupts
  - Interrupts can be classified into two types:
    - Maskable Interrupts (Can be delayed or Rejected)
    - Non-Maskable Interrupts (Can not be delayed or Rejected)
- Interrupts can also be classified into:
    - Vectored (the address of the service routine is hard-wired)
    - Non-vectored (the address of the service routine needs to be supplied externally by the device)

Interrupt classification

- **Hardware Interrupt**→ An interrupt caused by an "External signal "
- **Software Interrupt** →An interrupt caused by "Special Instruction"
- **Maskable Interrupts**→ Can be delayed or Rejected
- **Non-Maskable Interrupts**→ Can not be delayed or Rejected (Service must)
- **Vectored** →Where the subroutine starts is referred to as **Vector Location**
- **Non-vectored** → The address of the service routine needs to be supplied externally by the device

Interrupt Name	Maskable	Vectored
INTR	Yes	No
RST 5.5	Yes	Yes
RST 6.5	Yes	Yes
RST 7.5	Yes	Yes
TRAP	No	Yes

Interrupt Vectors & the Vector Table

- An interrupt vector is a pointer to where the ISR is stored in memory.
- All interrupts (vectored or otherwise) are mapped onto a memory area called the Interrupt Vector Table (IVT).
  - The IVT is usually located in (0000H - 00FFH).

Vector Address = Interrupt number * 8

Interrupt Name	Calculation	Vector Address
INTR	--	--
TRAP (RST 4.5)	4.5x8=36	0024H
RST 5.5	5.5x8=44	002CH
RST 6.5	6.5x8=52	0034H
RST 7.5	7.5x8=60	003CH

8085 Interrupts Summary

Interrupt Name	Triggering Method	Priority	Maskable	Masking Method	Vector Address
TRAP RST 4.5	Edge & Level Sensitive	1st Highest	No	None	0024H
RST 7.5	Edge Sensitive	2nd	Yes	DI / EI SIM	003CH
RST 6.5	Level Sensitive	3rd	Yes	DI / EI SIM	0034H
RST 5.5	Level Sensitive	4th	Yes	DI / EI SIM	002CH
INTR	Level Sensitive	5th Lowest	Yes	Pin ($\overline{\text{INTR}}$ & $\overline{\text{INTA}}$)	--

Software Interrupt

- The 8085 recognizes 8 RESTART instructions:

  RST n (RST0 - RST7)

  - Each of these would send the execution to a redetermined hard-wired memory location:

Restart Instruction	Vector Address
RST 0	CALL 0000H
RST 1	CALL 0008H
RST 2	CALL 0010H
RST 3	CALL 0018H
RST 4	CALL 0020H
RST 5	CALL 0028H
RST 6	CALL 0030H
RST 7	CALL 0038H

TRAP

- TRAP is the only non-maskable interrupt.
  - It does not need to be enabled because it cannot be disabled.
- It has the highest priority amongst interrupts.
- It is edge and level sensitive.
  - It needs to be high and stay high to be recognized.
  - Once it is recognized, it won't be recognized again until it goes low, then high again.

- TRAP is usually used for power failure and emergency shutoff.

The 8085 Non-Vectored Interrupt Process

1. The interrupt process should be enabled using the EI instruction.
2. The 8085 checks for an interrupt during the execution of every instruction.
3. If INTR is high, MP completes current instruction, disables the interrupt and sends INTA (Interrupt acknowledge) signal to the device that interrupted
4. INTA allows the I/O device to send a RST instruction through data bus.
5. Upon receiving the INTA signal, MP saves the memory location of the next instruction on the stack and the program is transferred to 'call' location (ISR Call) specified by the RST instruction

The 8085 Non-Vectored Interrupt Process

6. Microprocessor Performs the ISR.
7. ISR must include the ‘EI’ instruction to enable the further interrupt within the program.
8. RET instruction at the end of the ISR allows the MP to retrieve the return address from the stack and the program is transferred back to where the program was interrupted.

References

- **www.google.com**
- **www.wikipedia.com**
- **www.studymafia.org**
- **www.pptplanet.com**
- Gursharan Singh Tatla professorgstatla@gmail.com