MICROPROCESSORS & MICROCONTROLLERS

LECTURE NOTES

B. TECH
(III YEAR ECE & EEE)
(2020-21)

Prepared by:
Mrs. K. ANURADHA,
Assistant Professor

<u>Department of Electronics and Communication Engineering</u>

MALLA REDDY ENGINEERING COLLEGE
(Autonomous)
An UGC Autonomous Institution, Approved by AICTE and Affiliated to
JNTUH Hyderabad, Recognized under section 2(f) &12 (B) of UGC Act 1956,
Accredited by NAAC with ‘A’ Grade (II Cycle) and NBA, Maisammaguda,
Dhulapally (Post Via Kompally), Secunderabad-500 100 Website:
www.mrec.ac.in E-mail: principal@mrec.ac.in

2018-19 Onwards (MR-18)	**MALLA REDDY ENGINEERING COLLEGE (Autonomous)**	**B.Tech. V Semester**		
Code: 80414	**MICROPROCESSORS AND MICROCONTROLLERS**	**L**	**T**	**P**
Credits: 3		**3**	**-**	**-**

Pre-Requisites: Digital Electronics.

Course Objectives: This course provides the students to understand operation and programming of 8085 Microprocessor, develops real time applications using 8086 processor, understand the basic concepts of 8051 Microcontroller and interfacing with I/O devices.

MODULE I: 8085 Architecture **[8 Periods]**

Introduction to Microprocessors, Architecture of 8085, Pin Configuration and Function, internal register & flag register, Generation of Control Signals: Bus Timings: De-multiplexing of address/ data bus; Fetch Cycle, Execute Cycle, Instruction Cycle, Machine cycles, T-states, memory interfacing.

MODULE II: Instruction Set and Programming with 8085 **[10 Periods]**

Instruction for Data Transfer, Arithmetic and Logical Operations, Branching Operation, Machine Cycle Concept, Addressing Modes, Instructions Format, Stacks, Subroutine and Related Instructions, Elementary Concepts of Assemblers, Assembler Directives, Looping and Counting, Software Counters with Time Delays, Simple Programs using Instruction Set of 8085, Debugging, Programs Involving Subroutines, Programs for Code Conversion e.g. BCD to Binary, Binary to BCD, Binary to Seven-Segment LED Display. Binary to ASCII, ASCII to Binary, Program for Addition Subtraction, Programs for Multiplication and Division of Unsigned Binary Numbers.

MODULE III: 8086 Architecture **[09 Periods]**

A:8086 Architecture-Functional diagram, Register Organization, Memory Segmentation, Programming Model, Memory addresses, Physical Memory Organization, Architecture of 8086, Signal descriptions of 8086- Common Function Signals, Timing diagrams, Interrupts of 8086.

B: Interfacing I/O Devices: Interfacing of 8086 with Memory, key board and display, A/D and D/A.

MODULE IV: Introduction to Microcontroller **[10 Periods]**

A brief History of Microcontrollers, Harvard Vs Von-Neumann Architecture; RISC Vs CISC, Classification of MCS-51family based on their features (8051,8052, 8031, 8751, AT89C51), Pin configuration of 8051.

8051 Microcontroller Architecture and Instruction Set: Registers of 8051, Inbuilt RAM, Register banks, stack, on-chip and external program code memory ROM, power reset and clocking circuits, I/O port structure, addressing modes, Instruction set and programming.

MODULE V: 8051 Real Time Control [11 Periods]

Counter/Timer and Interrupts of 8051: Introduction, Registers of timer/counter, Different modes of timer/counter, Timer/counter programming, Interrupt *Vs* Polling, Types of interrupts and vector addresses, register used for interrupts initialization, programming of external interrupts, Timer interrupts.

Asynchronous Serial Communication and Programming: Introduction to serial communication, Programming the Serial Communication Interrupts, RS232 standard, RS422 Standard, RS-485 standard, Max 232/233 Driver.

Interfacing with 8051: Interfacing and programming of: ADC (0804,0808/0809,0848) & DAC(0808), dc motor, stepper motor, Relays, LED and Seven segment display, LCD, 4x4 keyboard matrix.

Text Books:

1. Ramesh Gaonkar, "Microprocessor Architecture, Programing and Application with 8085" , Penram, 5th Edition, 2002.
2. A.K.Ray, "Advanced Micro processors and Peripherals" 3rd Tata McGraw-Hill, Edition.
3. Mazidi, Mazidi&McKinlay, "The 8051 Microcontroller and Embedded Systems using Assembly and C" 2nd Edition,PHI.

Reference Books:

1. D. V Hall TMH, "Microprocessors and Interfacing" 2nd Edition, 2006
2. K. Uday Kumar, B.S. Umashankar, "The 8085 Microprocessor: Architecture, programming and Interfacing" Pearson, 2008.
3. Liu and Gibson, "Micro Computer System 8086/8088 Family Architecture, Programming and Design" PHI, 2nd Edition
4. Kenneth. J. Ayala, Cengage Learning, "The 8051 Microcontroller" 3rd Edition, 2004.

E-Resources:

1. https://www.tutorialspoint.com › Microprocessor › Microprocessor - 8085 Architecture
2. http://www.cpu-world.com/CPUs/8086/
3. https://www.journals.elsevier.com/microprocessors-and-microsystems/
4. http://rtcmagazine.com/technologies/view/Microcontrollers
5. http://nptel.ac.in/courses/106108100/
6. http://nptel.ac.in/courses/108107029/
7. nptel.ac.in/courses/106108100/

Course Outcomes:

After Completion of this course the student will able to

1. Learn basic concepts, organization of 8085 microprocessor.
2. Program the 8085 microprocessor.
3. Understand the 8086 microprocessor and it's interfacing with I/O devices.
4. Know the architecture and instruction set of 8051 Microcontroller.
5. Interface I/O devices with 8051 microcontroller.

MICROPROCESSOR AND MICROCONTROLLER

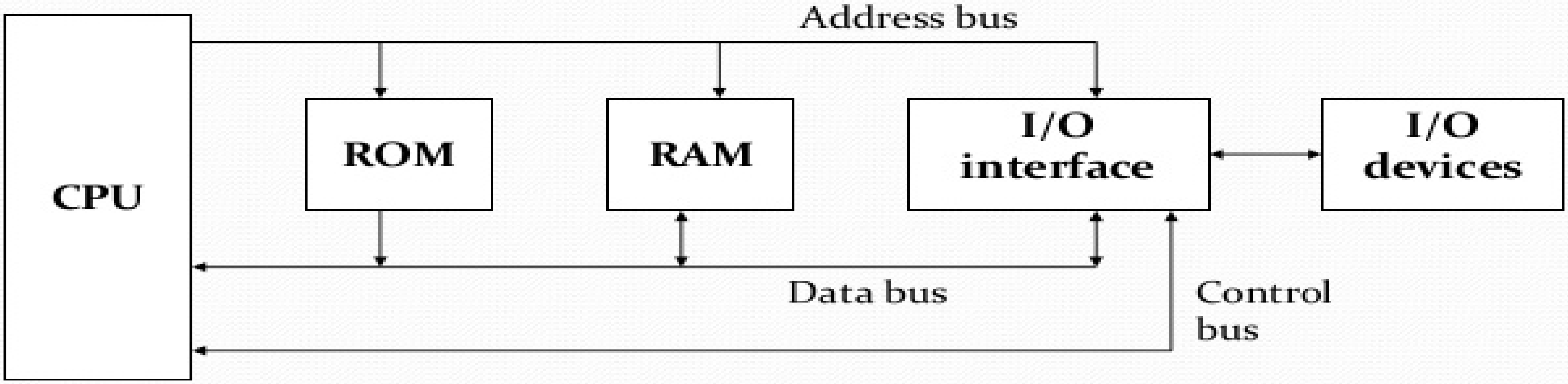
MODULE I

8O85 ARCHITECTURE

Introduction

Computer: A computer is a programmable machine that receives input, stores and manipulates data/information, and provides output in a useful format.

Basic computer system consist of a CPU, memory and I/O unit.



Block diagram of a basic computer system

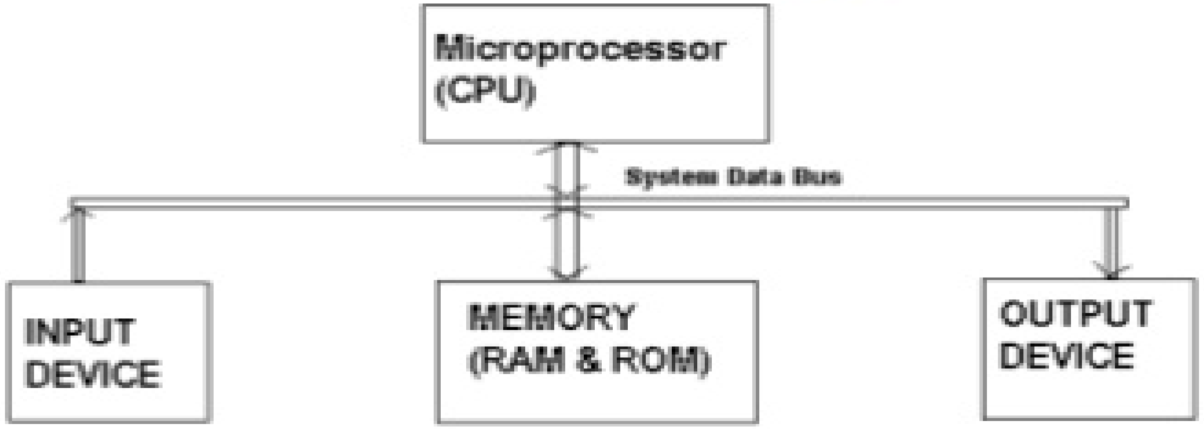
Basic Concepts of Microprocessors

- **Microcomputer:-** It is a programmable machine. The two principal characteristics of a computer are:
  - Responds to a specific set of instructions in a well-defined manner.
  - It can execute a prerecorded list of instructions (a program)
  - Its main components are
    - CPU
    - Input & Output devices
    - Memory

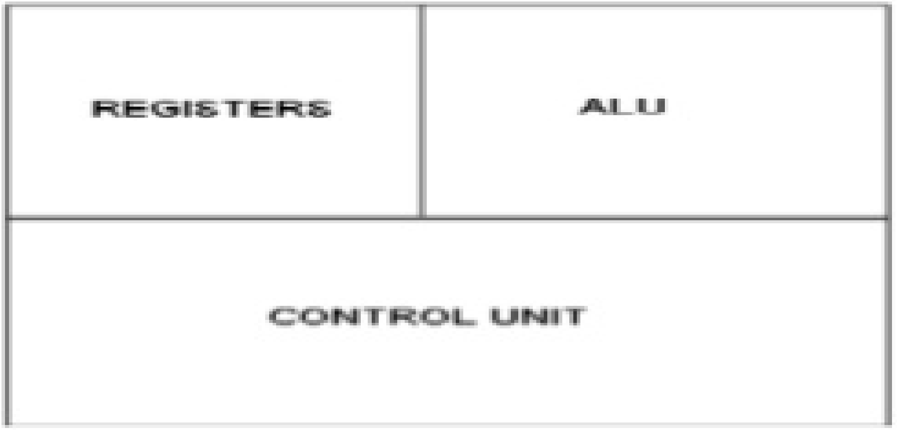
- **Microprocessor:-** It is a programmable VLSI chip which includes ALU, register circuits & control circuits. Its main units are-
  - ALU
  - Registers
  - Control Unit

- **Microcontroller:-** Silicon chip which includes microprocessor, memory & I/O in a single package

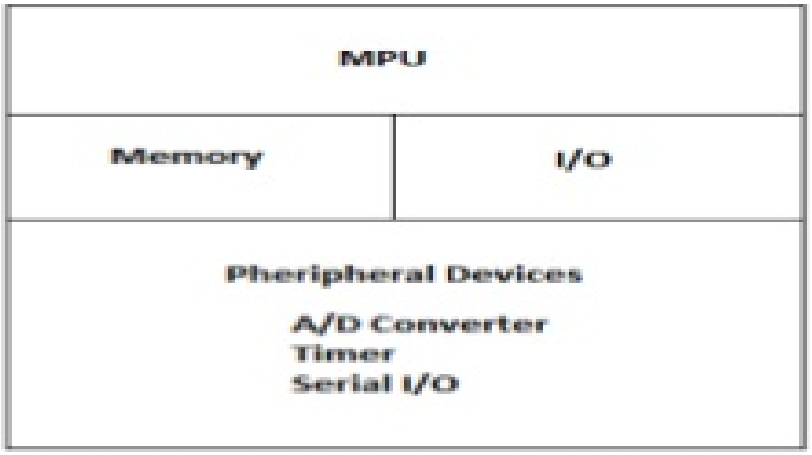
Block Diagram



Microcomputer



Microprocessor



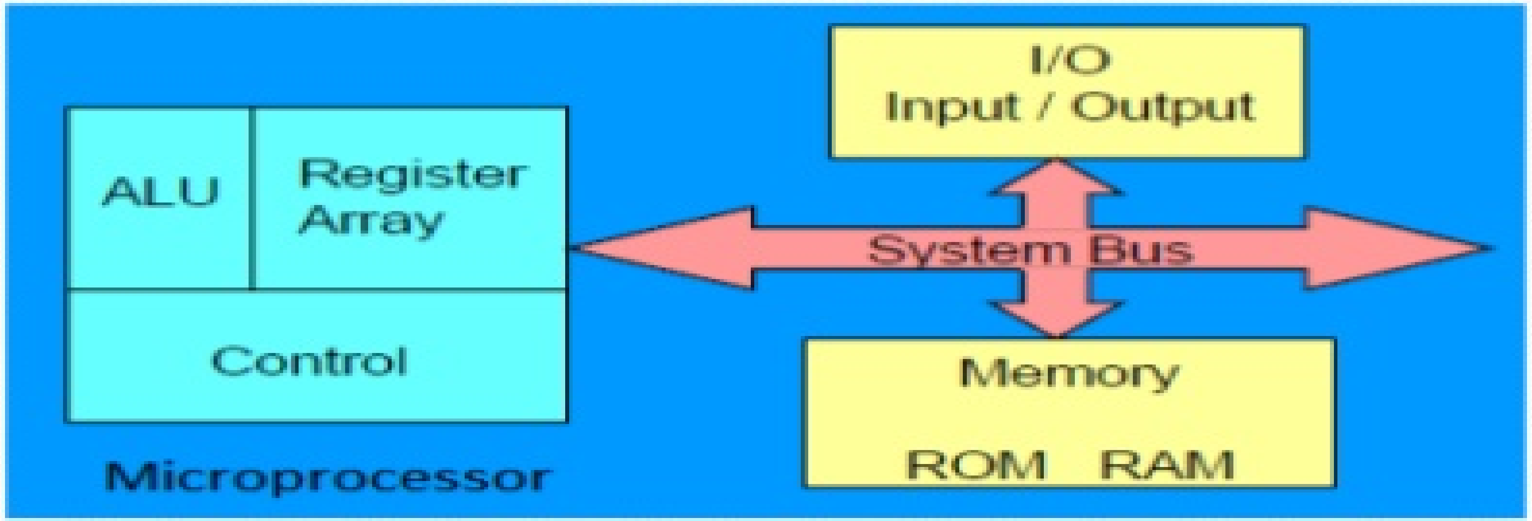
Microcontroller

Definition of the Microprocessor

✓ Microprocessor is a **Programmable, Clock driven, Register based, Electronic device** that reads instruction from a storage device, takes the data from input unit and process the data according to the instructions and provides the result to the output unit.

- **Programmable**- Perform Different set operation on the data depending on the sequence of instructions supplied by the programmer.
- **Clock Driven** – Whole task is divided into basic operations, are divided into precise system clock periods.
- **Register Based** – Storage element
- **Electronic Device** – fabricated on a chip

Microprocessor Based System with bus Architecture



- **ALU:-** Arithmetic and logical operations like add, subtraction, AND & OR.
- **Register Array: -** Store data during the execution of program.
- **Control Unit:** Provides necessary timing & control signal. It controls the flow of data between microprocessor and peripherals.

*Microprocessor is one component of microcomputer.

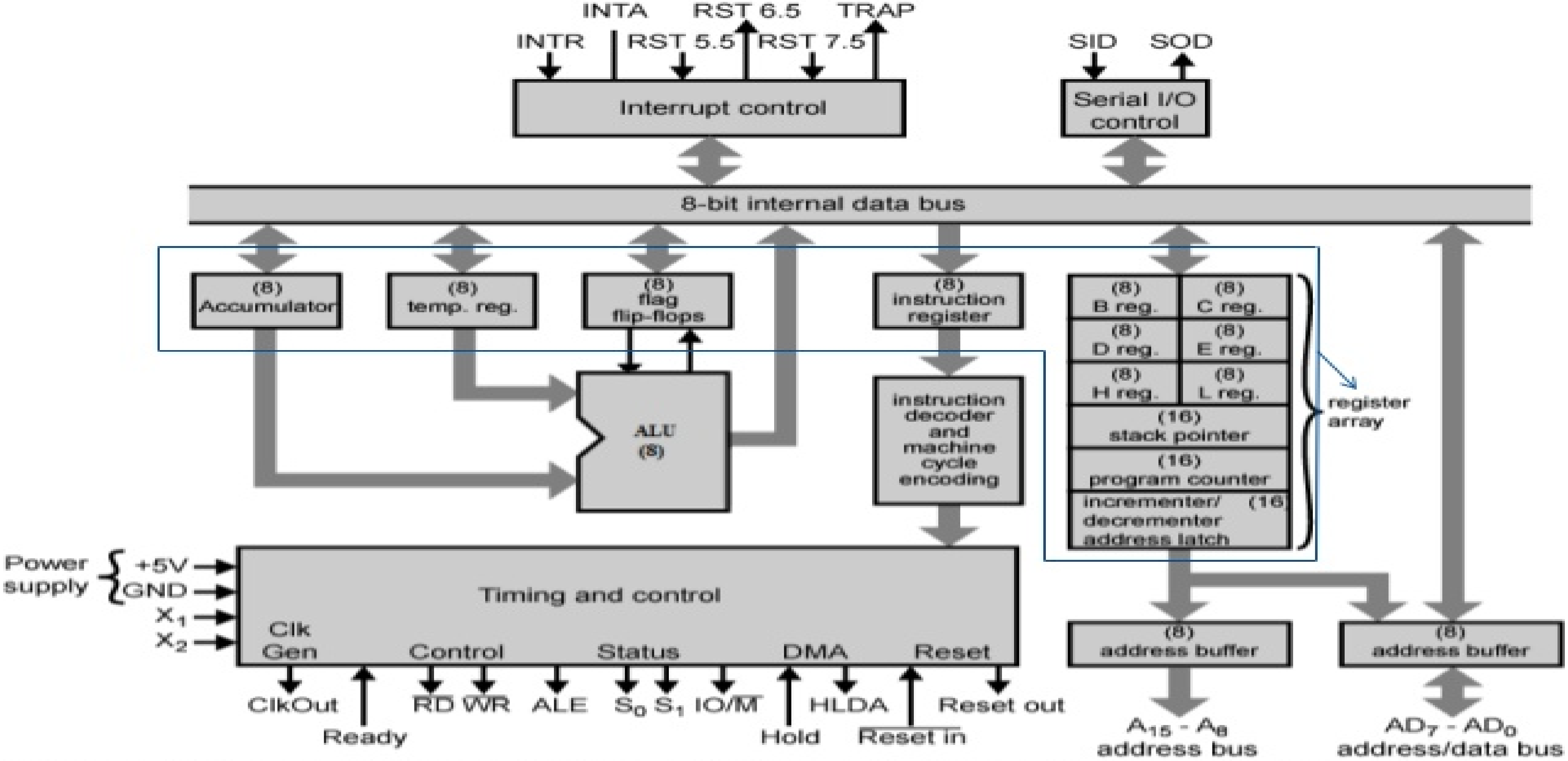
- **Memory:**
  - ➢ Stores information such as instructions and data in binary format (0 and 1).
  - ➢ Sub-system" of microprocessor-based system. sub-system includes the registers inside the microprocessor .
    - ✓ **Read Only Memory (ROM):** used to store programs that do not need alterations.
    - ✓**Random Access Memory (RAM) (R/WM)**: used to store programs that can read and altered like programs and data.
- **Input/output:** Communicates with the outside world.
- **System Bus:** Communication path between the microprocessor and peripherals.
  - ➢ group of wires to carry bits.

How does a Microprocessor works

- To execute a program, the microprocessor “reads” each instruction from memory, “interprets” it, then “executes or perform” it.
- The right name for the cycle is
  - Fetch
  - Decode
  - Execute
- This sequence is continued until all instructions are performed.

Features of Microprocessor- 8085

- 8085 is developed by INTEL
- 8 bit microprocessor: can accept 8 bit data simultaneously
- Operates on single +5V D.C. supply.
- Designed using NMOS technology
- 6200 transistor on single chip
- It provides on chip clock generator, hence it does not require external clock generator.
- Operates on 3MHz clock frequency.
- 8bit multiplexed address/data bus, which reduce the number of pins.
- 16address lines, hence it can address 2^16 = 64 K bytes of memory
- It generates 8 bit I/O addresses, hence it can access 2^8 = 256 I/O ports.
- 5 hardware interrupts i.e. TRAP, RST6.5, RST5.5, RST4.5, and INTR
- It provides DMA.



Internal Architecture (functional block diagram) of 8085

8085 Architecture…….cont…

8085 architecture consists of following blocks:

1. Register Array
2. ALU & Logical Group
3. Instruction decoder and machine cycle encoder, Timing and control circuitry
4. Interrupt control Group
5. Serial I/O control Group

8085 Architecture …… cont….

1. **Registers Array** : 14 register out of which 12 are 8 bit capacity and 2 of 16 bit. Classify into 4 types

(a) **General purpose register: (user accessible)**

- B,C,D,E,H,L are 8 bit register.(can be used singly)
- Can also be used for 16-bit register pairs- BC, DE & HL.
- Used to store the intermediate data and result
- H & L can be used as a data pointer(holds memory address)

(b) **Special Purpose Register[A, Instruction Register and Flag]**

(b.1) Accumulator (A): (user accessible)

- 8 bit register
- All the ALU operations are performed with reference to the contents of Accumulator.
- Result of an operation is stored in A.
- Store 8 bit data during I/O transfer

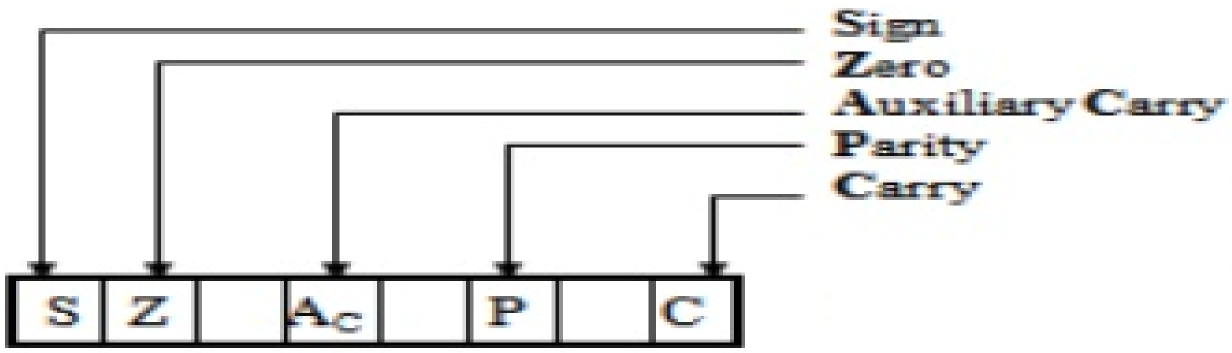
(b.2) Instruction Register: (user not accessible)

- When an instruction is fetched from memory, it is loaded in IR. Then transferred to the decoder for decoding.
- It is not programmable and can not be accessed through any instruction.

8085 Architecture …… cont….

(b.3) Flag Register(F): (user accessible)

- 8 bit Register
- Indicates the status of the ALU operation.
- ALU includes 5 flip flop, which are set or reset after an operation according to data conditions of the result in the accumulator.



(Flag Register)

Flag Register...... cont....

Flag	Significance
C or CY (Carry)	CY is set when an arithmetic operation generates a carry out, otherwise it is 0 (reset)
P (Parity)	**P= 1;** if the result of an ALU operation has an even number of 1's in A; **P= 0;** if number of 1 is odd.
AC (Auxiliary carry)	Similar to CY, AC= 1 if there is a carry from D3 to D4 Bit AC= 0 if there is a no carry from D3 to D4 Bit (not available for user)
Z(zero)	Z = 1; if result in A is 00H 0 otherwise
S(Sign)	S=1 if D7 bit of the A is 1(indicate the result is -ive) S= 0 if D7 bit of the A is 0(indicate the result is +ive)

8085 Architecture ...... cont....

(c) Temporary Register[W, Z, Temporary data register]

- Internally used by the MP(user not accessible)

(c.1) W and Z register:

- 8 bit capacity
- Used to hold temporary addresses during the execution of some instructions

(c.2) Temporary data register:

- 8 bit capacity
- Used to hold temporary data during ALU operations.

8085 Architecture …… cont….

(d) Pointer Register or special purpose [SP, PC]

(d.1) Stack Pointer(SP)

- 16 bit address which holds the address of the data present at the top of the stack memory
- It is a reserved area of the memory in the RAM to store and retrieve the temporary information.
- Also hold the content of PC when subroutines are used.
- When there is a subroutine call or on an interrupt. ie. pushing the return address on a jump, and retrieving it after the operation is complete to come back to its original location.

(d.3) Program Counter(PC)

- 16 bit address used for the execution of program
- Contain the address of the next instruction to be executed after fetching the instruction it is automatically incremented by 1.
- Not much use in programming, but as an indicator to user only.

8085 Architecture …… cont….

In addition to register MP contains some latches and buffer

- ✓ **Increment and decrement address latch**
  - ➢ 16 bit register
  - ➢ Used to increment or decrement the content of PC and SP
- ✓ **Address buffer**
  - ➢ 8 bit unidirectional buffer
  - ➢ Used to drive high order address bus(A8 to A15)
  - ➢ When it is not used under such as reset, hold and halt etc this buffer is used tristate high order address bus.
- ✓ **Data/Address buffer**
  - ➢ 8 bit bi-Directional buffer
  - ➢ Used to drive the low order address (A0 to A7) and data (D0 to D7) bus.
  - ➢ Under certain conditions such as reset, hold and halt etc this buffer is used tristate low order address bus.

8085 Architecture …… cont….

(2) ALU & Logical Group: it consists ALU, Accumulator, Temporary register and Flag Register.

(a) ALU

- Performs arithmetic and logical operations
- Stores result of arithmetic and logical operations in accumulator

(b) Accumulator

- General purpose register
- Stores one of the operand before any arithmetic and logical operations and result of operation is again stored back in Accumulator
- Store 8 bit data during I/O transfer

8085 Architecture …… cont….

(2) ALU & Logical Group……………………cont…………………

(c) Temporary Register

- 8 bit register
- During the arithmetic and logical operations one operand is available in A and other operand is always transferred to temporary register

  For Eg.: ADD B – content of B is transferred into temporary register before actual addition

(d) Flag Register

- Five flag is connected to ALU
- After the ALU operation is performed the status of result will be stored in five flags.

8085 Architecture …… cont….

(3) Instruction decoder and machine cycle encoder, Timing and control circuitry

(a) Instruction decoder and machine cycle encoder :

- ➢ Decodes the op-code stored in the Instruction Register (IR) and establishes the sequence of events to follow.
- ➢ Encodes it and transfer to the timing & control unit to perform the execution of the instruction.

(b) Timing and control circuitry

- ➢ works as the brain of the CPU
- ➢ For proper sequence and synchronization of all the operations of MP, this unit generates all the timing and control signals necessary for communication between microprocessor and peripherals.

8085 Architecture …… cont….

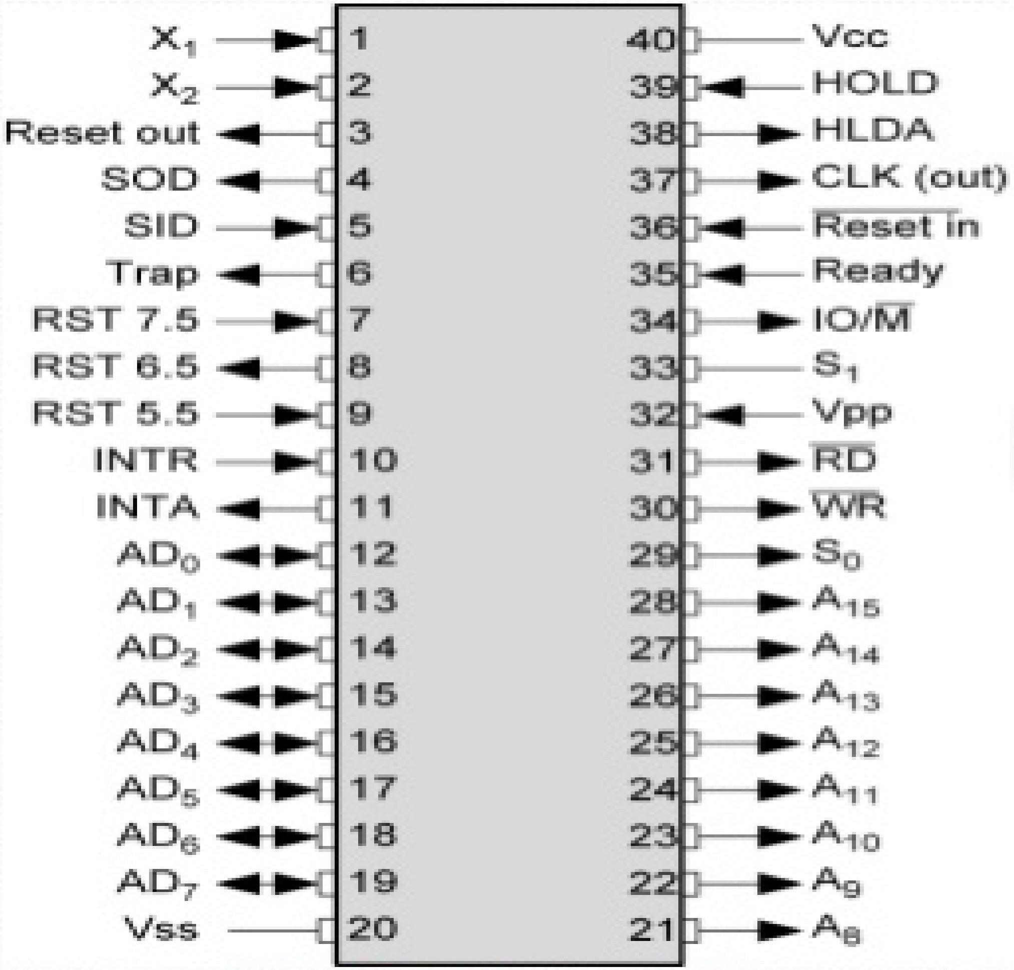
(4) Interrupt Control group

- **Interrupt:-** Occurrence of an external disturbance
- After servicing the interrupt, 8085 resumes its normal working sequence
- Transfer the control to special routines
- Five interrupts: - TRAP, RST7.5, RST6.5, RST5.5, INTR
- In response to INTR, it generates INTA signal

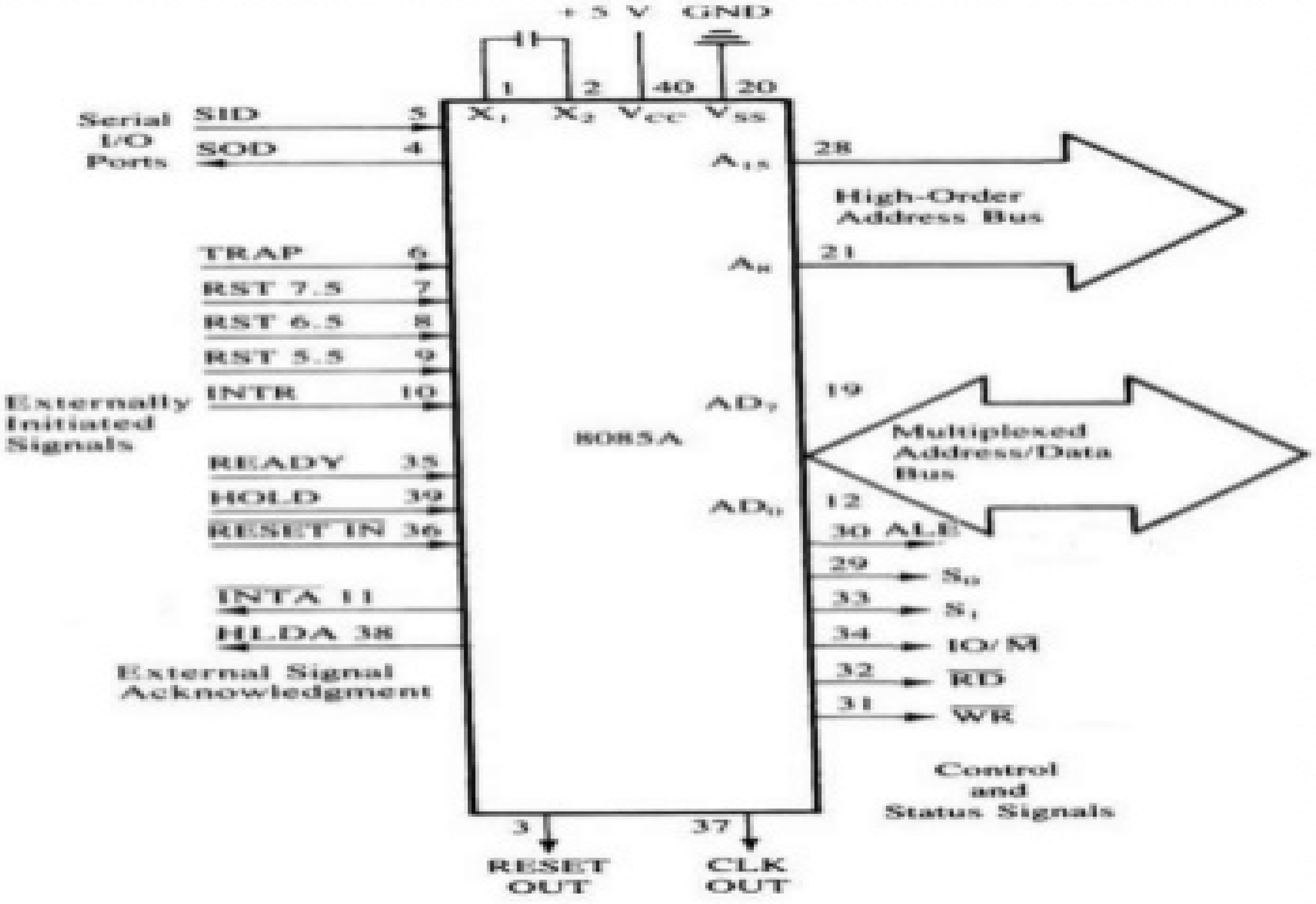
(5) Serial I/O control Group

- Data transfer red on D0- D7 lines is parallel data
- But under some condition it is used serial data transfer
- Serial data is entered through SID(serial input data) input (received)
- Serial data is outputted on SOD(serial output data) input (send)

8085 Pin Diagram



Pin Configuration



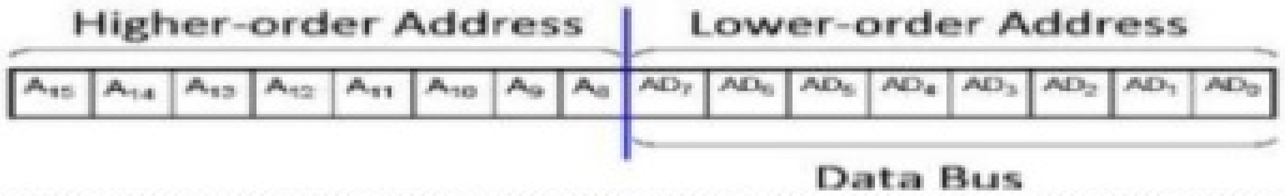
Functional Pin diagram

8085 Pin Description

- The 8085 is an 8-bit general purpose microprocessor that can address 64K Byte of memory.
- It has 40 pins and uses +5V for power. It can run at a maximum frequency of 3 MHz.
- The pins on the chip can be grouped into 6 groups:
  - Address Bus and Data Bus.
  - Status Signals.
  - Control signal
  - Interrupt signal
  - Power supply and Clock signal
  - Reset Signal
  - DMA request Signal
  - Serial I/O signal
  - Externally Initiated Signals.

The Address and Data Busses

- □ **Address Bus** (Pin 21-28)
  - ➢ 16 bit address lines A0 to A15
  - ➢ The address bus has 8 signal lines A8 – A15 which are unidirectional.
  - ➢ The other 8 address lines A0 to A7 are multiplexed (time shared) with the 8 data bits.



A_{15}	A_{14}	A_{13}	A_{12}	A_{11}	A_{10}	A_9	A_8	AD_7	AD_6	AD_5	AD_4	AD_3	AD_2	AD_1	AD_0

- □ **Data Bus** (Pin 19-12)
  - ➢ To save the number of pins lower order address pin are multiplexed with 8 bit data bus (bidirectional)
  - ➢ So, the bits AD0 – AD7 are bi-directional and serve as A0 – A7 and D0 – D7 at the same time.
  - ➢ During the execution of the instruction, these lines carry the address bits during the early part (T1 state), then during the late parts(T2 state) of the execution, they carry the 8 data bits.

Status Signals

Status Pins – ALE, S_1, S_0

1. **ALE(Address Latch Enable): (Pin 30)**
   - Used to demultiplexed the address and data bus
   - +ive going pulse generated when a new operation is started by uP.
   - ALE = 1 when the AD0 – AD7 lines have an address
   - ALE = 0 When it is low it indicates that the contents are data.
   - This signal can be used to enable a latch to save the address bits from the AD lines.

2. **S1 and S0 (Status Signal): (Pin 33 and 29)**
   - Status signals to specify the kind of operation being performed .
   - Usually un-used in small systems.

S1	S0	Operation
0	0	HALT
0	1	WRITE
1	0	READ
1	1	FETCH

Status Signals

Status Pins – ALE, S_1, S_0

1. **ALE(Address Latch Enable): (Pin 30)**
   - Used to demultiplexed the address and data bus
   - +ive going pulse generated when a new operation is started by uP.
   - ALE = 1 when the AD0 – AD7 lines have an address
   - ALE = 0 When it is low it indicates that the contents are data.
   - This signal can be used to enable a latch to save the address bits from the AD lines.

2. **S1 and S0 (Status Signal): (Pin 33 and 29)**
   - Status signals to specify the kind of operation being performed .
   - Usually un-used in small systems.

S1	S0	Operation
0	0	HALT
0	1	WRITE
1	0	READ
1	1	FETCH

Control Signals

Control Pins – RD, WR, IO/M(active low)

1. **RD: Read(Active low)** (Pin 32)
   - ➢ Read Memory or I/O device
   - ➢ Indicated that data is to be read either from memory or I/P device and data bus is ready for accepting data from the memory or I/O device.
2. **WR: Write(Active low)** (Pin 31)
   - ➢ Write Memory or I/O device
   - ➢ Indicated that data on the data bus are to be written into selected memory or I/P device.
3. **IO/$\overline{\text{M}}$: (Input Output/Memory-Active low)** (Pin 34)
   - ➢ Signal specifies that the read/write operation relates to whether memory or I/O device.
   - ➢ When (IO/M=1) the address on the address bus is for I/O device
   - ➢ When (IO/M=0) the address on the address bus is for memory

IO/M(active low)	RD	WR	Control Signal	Operation
0	0	1	MEMR	M/M Read
0	1	0	MEMW	M/M write
1	0	1	IOR	I/O Read
1	1	0	IOW	I/O Write

Control and status Signals

When S_0, S_1 is combined with IO/M(active low), we get status of machine cycle

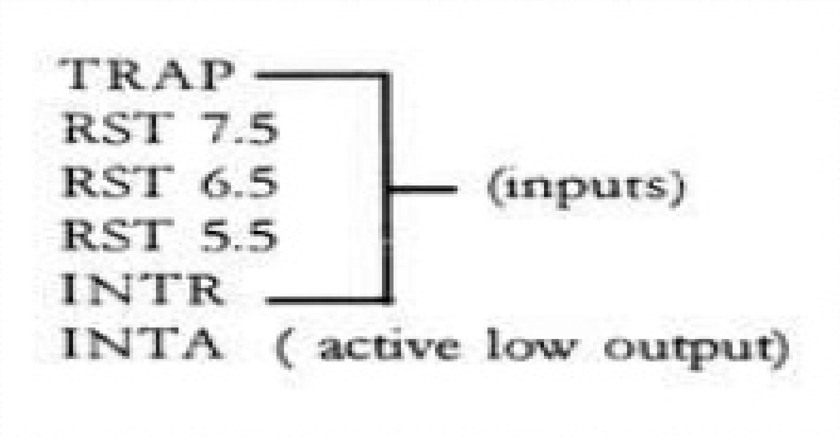
IO/M	S_1	S_0	OPERATION	Control Signal
0	1	1	Opcode fetch	$\overline{RD} = 0$
0	1	0	Memory read	$\overline{RD} = 0$
0	0	1	Memory write	$\overline{WR} = 0$
1	1	0	I/O read	$\overline{RD} = 0$
1	0	1	I/O write	$\overline{WR} = 0$
1	1	0	Interrupt Acknowledge	$\overline{INTA} = 0$
Z	0	0	Halt	$\overline{RD}$, $\overline{WR} = Z$ and $\overline{INTA} = 1$
Z	x	x	Hold	
Z	x	x	Reset	

Z= Tristate, X = don't care condition

Interrupts

➢ They are the signals initiated by an external device to request the microprocessor to do a particular task or work.

➢ There are five hardware interrupts called, **(Pin 6-11)**



TRAP
RST 7.5
RST 6.5
RST 5.5
INTR
(inputs)
INTA (active low output)

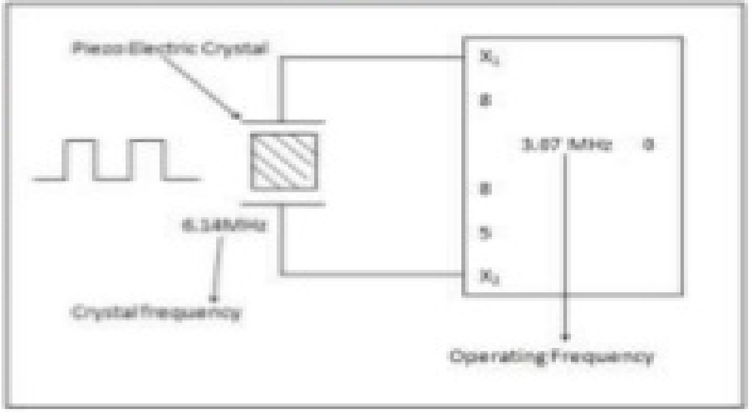
➢ On receipt of an interrupt, the microprocessor acknowledges the interrupt by the active low INTA (Interrupt Acknowledge) signal.

Power supply and Clock Signal

Vcc (Pin 40) : single +5 volt power supply

Vss (Pin 20) : Ground

There are 3 important pins in this group.



- ✓ **Xo and X1 :((Pin 1-2)**
  - ➢ Crystal or R/C network or LC network connections to set the frequency of internal clock generator.
  - ➢ The frequency is internally divided by two.
  - ➢ Since the basic operating timing frequency is 3 MHz, a 6 MHz crystal is connected to the Xo and X1 pins.

- ✓ **CLK (output): (Pin 37)**
  - ➢ Clock Output is used as the system clock for peripheral and devices interfaced with the microprocessor.

Reset Signals

- ✓ **Reset In (input, active low) (Pin 36)**
  - ➢ This signal is used to reset the microprocessor.
  - ➢ The program counter inside the microprocessor is set to zero(0000H)
  - ➢ The buses are tri-stated.

- ✓ **Reset Out (Output, Active High) (Pin 3)**
  - ➢ It indicates MP is being reset.
  - ➢ Used to reset all the connected devices when the microprocessor is reset.

DMA Request Signals

- **DMA:**
  - When 2 or more devices are connected to a common bus, to prevent the devices from interfering with each other, the tristate gates are used to disconnect all devices except the one that is communicating at a given instant .
  - The CPU controls the data transfer operation between memory and I/O device.
  - DMA operation is used for large volume data transfer between memory and an I/O device directly.
  - The CPU is disabled by tri-stating its buses and the transfer is effected directly by external control circuits.

✓ **HOLD** (Pin 38)

- This signal indicates that another device is requesting the use of address and data bus.
- So it relinquish the use of buses as soon as the current machine cycle is completed.
- MP regains the bus after the removal of a HOLD signal

✓ **HLDA** (Pin 39)

- On receipt of HOLD signal, the MP acknowledges the request by sending out HLDA signal and leaves out the control of the buses.
- After the HLDA signal the DMA controller starts the direct transfer of data.
- After the removal of HOLD request HLDA goes low.

Serial I/O Signals

These pins are used for serial data communication

- ✓ **SID (input) Serial input data (Pin 4)**
  - ➢ It is a data line for serial input
  - ➢ Used to accept serial data bit by bit from external device
  - ➢ The data on this line is loaded into accumulator bit 7 whenever a RIM instruction is executed.

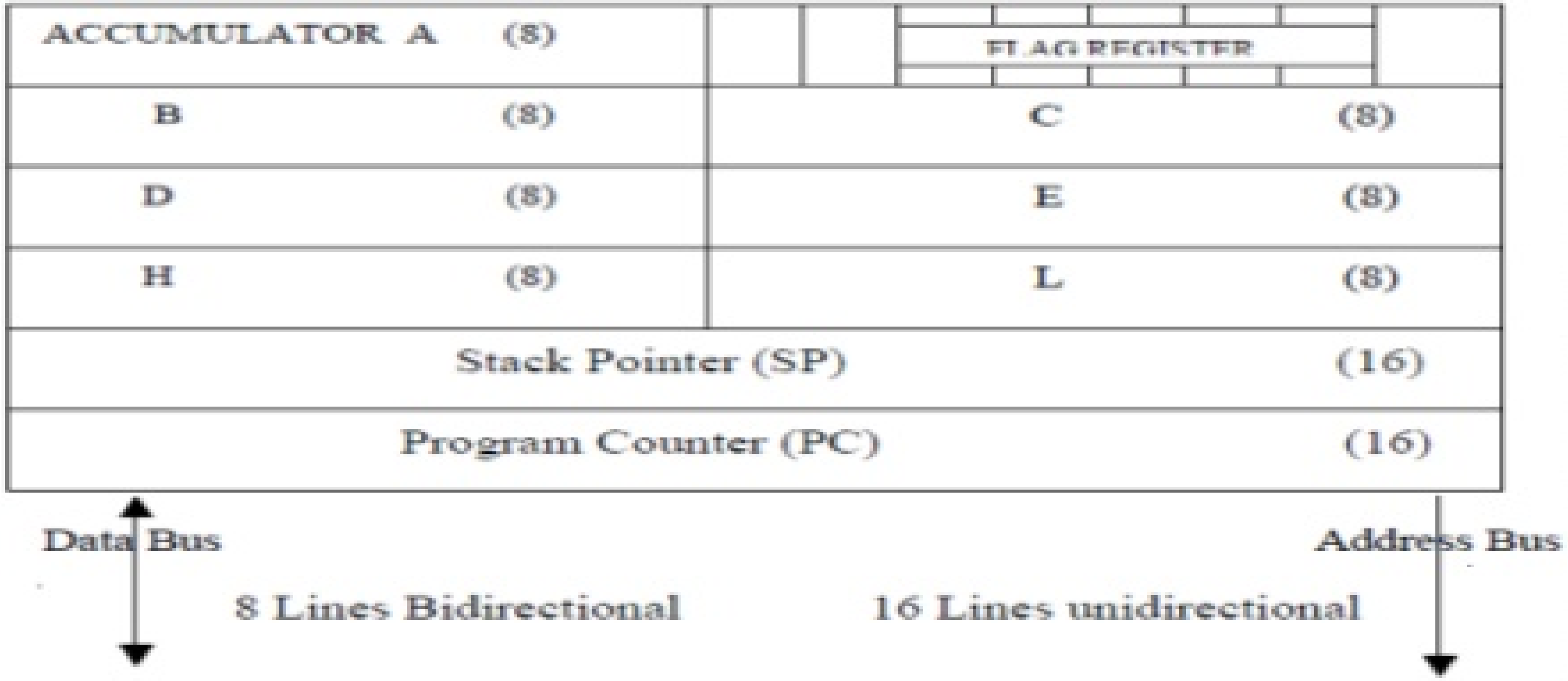
- ✓ **SOD (output) Serial output data (Pin 5)**
  - ➢ It is a data line for serial output
  - ➢ Used to transmit serial data bit by bit to the external device
  - ➢ The 7th bit of the accumulator is outputted on SOD line when SIM instruction is executed.

Externally Initiated signal

- ✓ **Ready (input) (Pin 35)**
  - ➢ Memory and I/O devices will have slower response compared to microprocessors.
  - ➢ Before completing the present job such a slow peripheral may not be able to handle further data or control signal from CPU.
  - ➢ The processor sets the READY signal after completing the present job to access the data.
  - ➢ It synchronize slower peripheral to the processor.
  - ➢ The microprocessor enters into WAIT state while the READY pin is disabled.

8085 Programming register and programming model

- The register which are programmable and available for the use are six general purpose register, A, F, PC, SP.

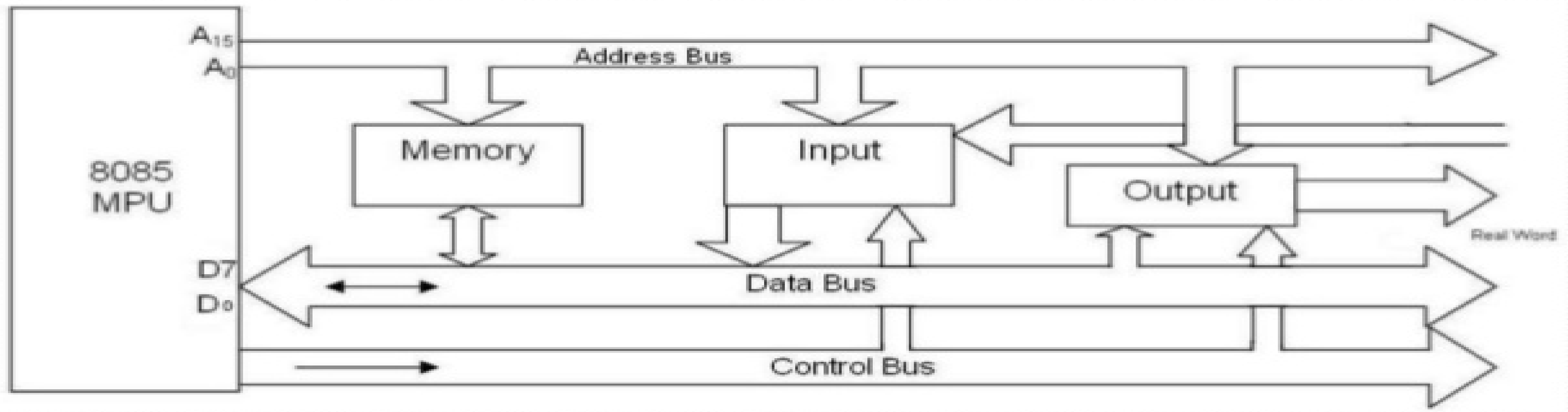


ACCUMULATOR A (8)	FLAG REGISTER
B (8)	C (8)
D (8)	E (8)
H (8)	L (8)
Stack Pointer (SP) (16)	
Program Counter (PC) (16)	

8085 programming model

Buses Structure

- Various I/O devices and memories are connected to CPU by a group of lines called as bus.

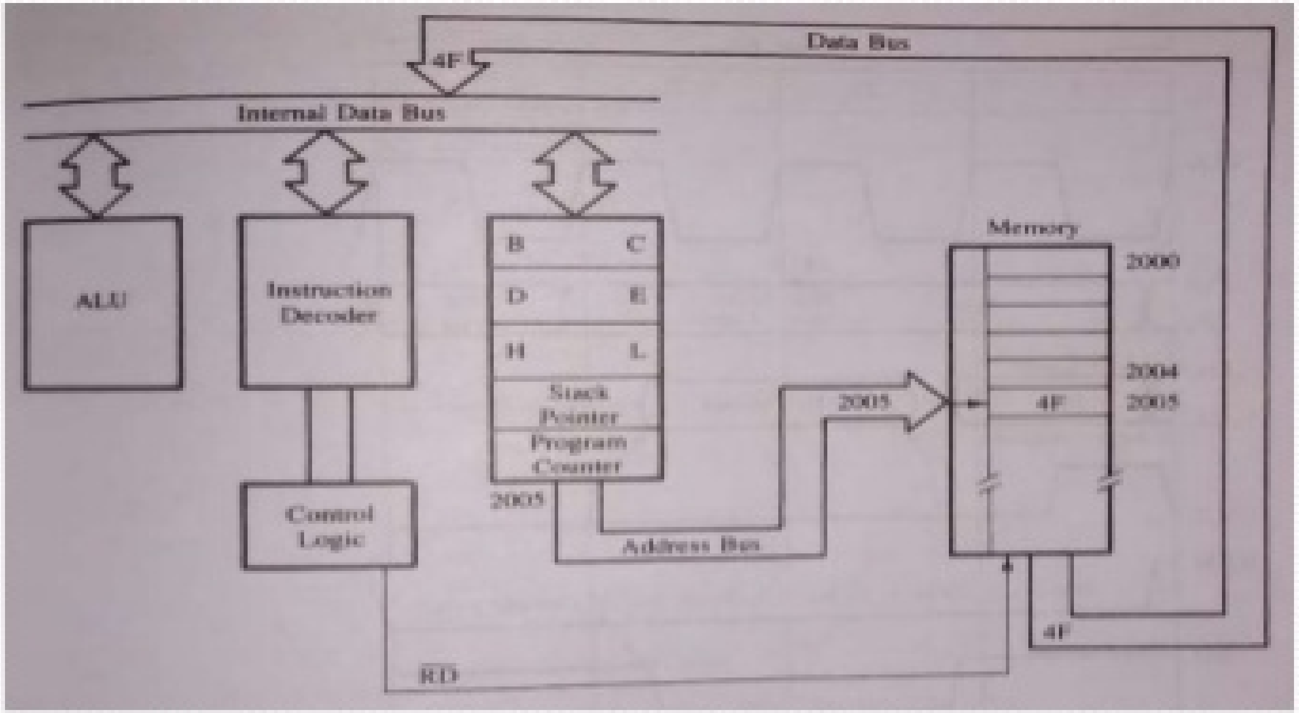


8085 Bus structure

Data flow from memory to MPU

Steps and data flow, when the instruction code 01001111 (4FH – MOV C, A) stored in the location 2005H, is being fetch.

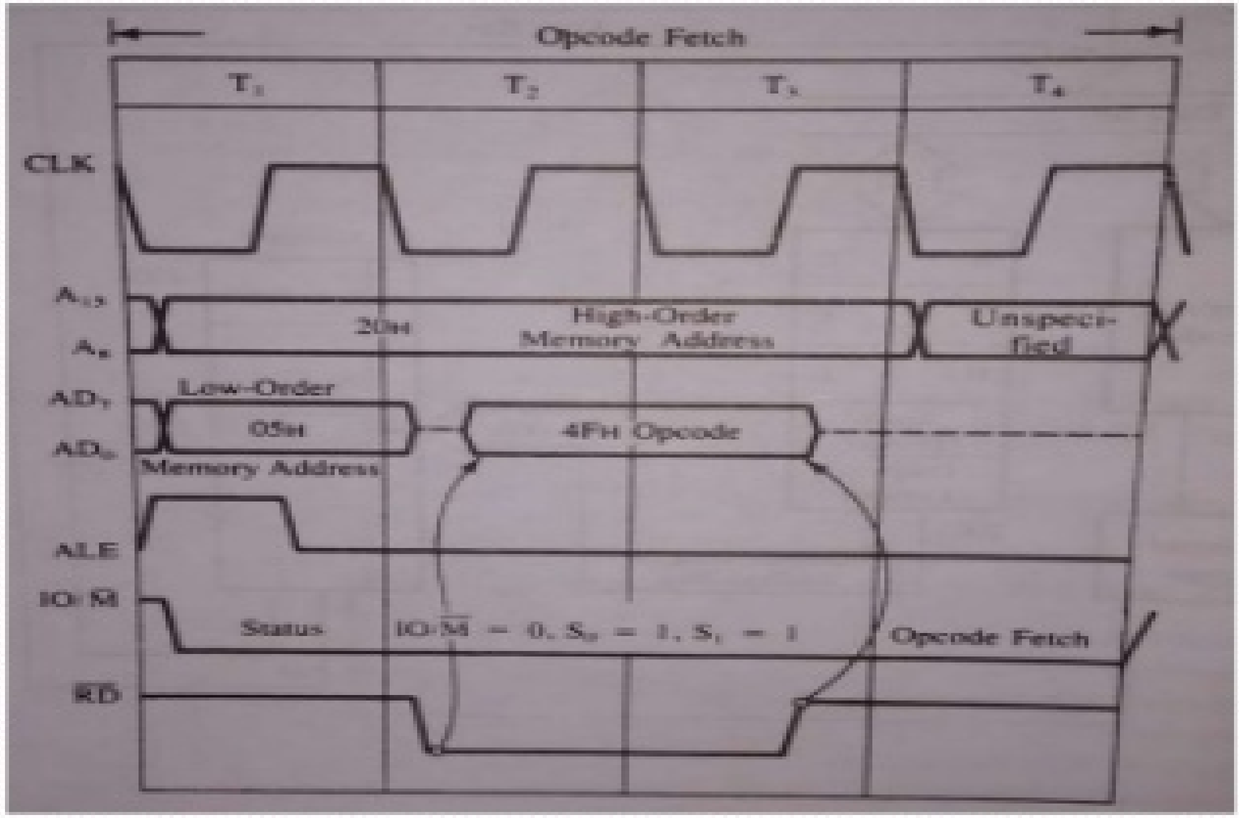
Fetch Cycle: To fetch the byte, the MPU needs to identify the memory location 2005 and enable the data flow from memory



Step 1: MPU places the 16 bit memory address from PC on the address bus
Step 2: Control unit send the signal RD to enable memory chip
Step 3: The byte from the memory location is placed on the data bus.
Step 4: The byte is placed on the instruction decoder of the MPU and task is carried out according to the instruction.

Timing: Transfer of byte from memory to MPU

- ❑How a data byte is transfer from memory to the MPU.
- ❑It shows the five different group of signals with clock



Step 1: At T1 higher order memory address 20H is placed on the A15 – A8 and the lower order memory address 05H is placed on the bus AD7-AD0, and ALE signal high. IO/M goes low(memory related signal).

Step 2: During T2 $\overline{RD}$ signal is sent out. $\overline{RD}$ is active during two clock periods.

Step 3 : During T3, Memory is enabled then instruction byte 4FH is placed on the data bus and transferred to MPU. When RD goes high it causes the bus to go into high impedance state.

Step 4: During T4, the machine code or byte is decoded by the instruction decoder and content of A is copied into register.

Generating Control Signals

Signals are used both for memory and I/O related operations. So four different control signals are generated by combining the signals $\overline{RD}$, $\overline{WR}$ and IO/$\overline{M}$.

$\overline{\text{MEMR}}$ = Reading from memory

$\overline{\text{MEMW}}$ = writing into memory

$\overline{\text{IOR}}$ = Reading from input port

$\overline{\text{IOW}}$ = writing to an output port

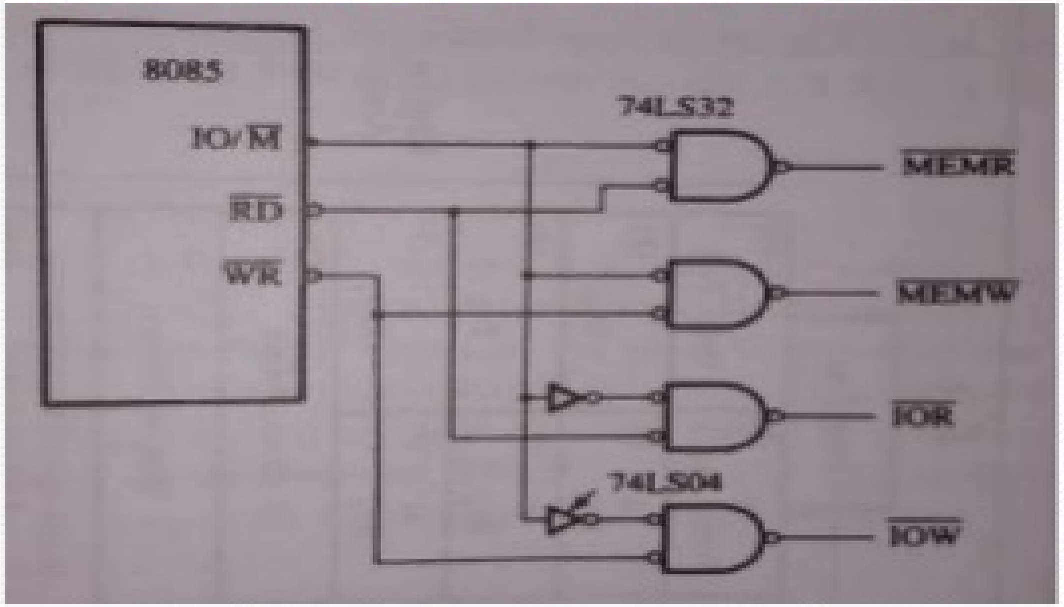


Fig: Generate Read/write control signal for memory and I/O

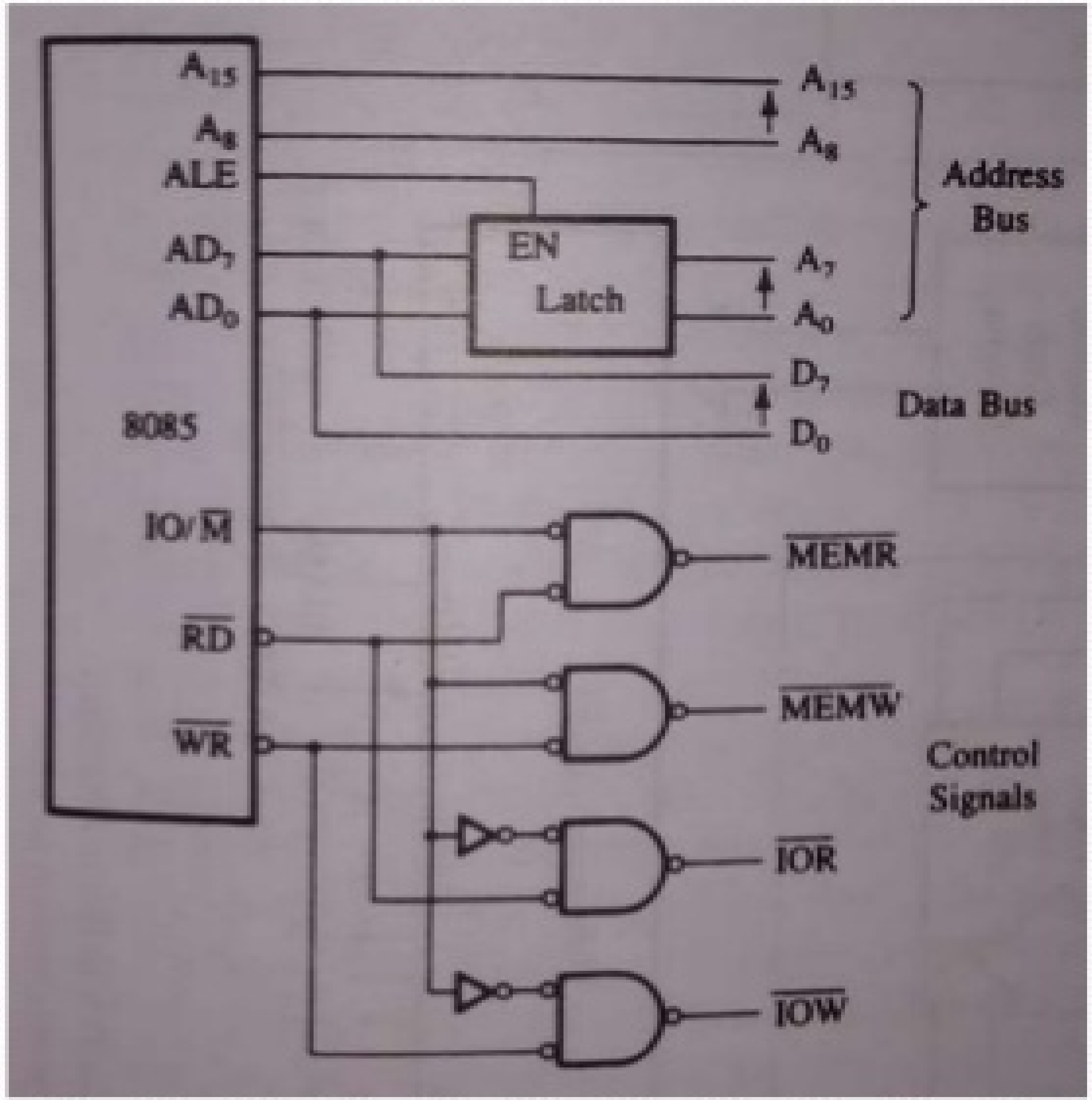


Fig: 8085 De-multiplexed address and data bus with control signal

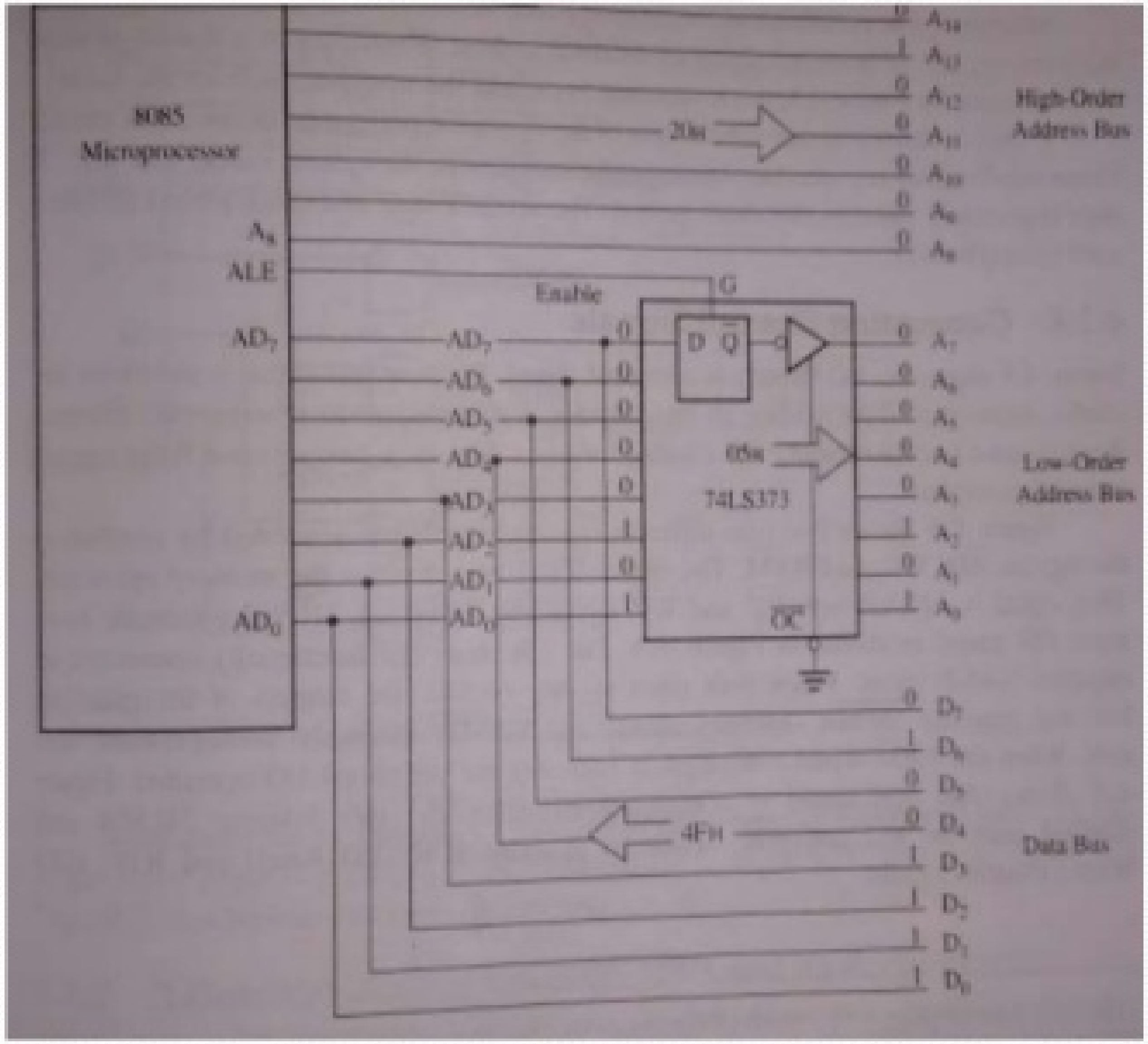
De-multiplexing AD7-AD0

- AD7– AD0 lines are serving a **dual purpose** and that they need to be demultiplexed to get all the information.
- The high order bits(20 H) of the address remain on the bus for **three clock periods**. However, the low order bits (05H) remain for only **one clock period** and they would be lost if they are not saved externally. The low order bits of the address disappear when they are needed most.
- To make sure we have the entire address for the full three clock cycles, we will use an external latch to save the value of AD7– AD0 when it is carrying the address bits. We use the **ALE signal** to enable this latch. ALE signal is connected to the enable (G) pin of the latch.

De-multiplexing AD7-AD0

- Given that ALE operates as a pulse during T1, ALE is high the latch is transparent; output changes according to input. So during T1 output of latch is 05H.

- When ALE goes low, the data byte 05H is latched until the next ALE, the output of the latch represents the low order address bus A7- A0 after latching operation.



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 .

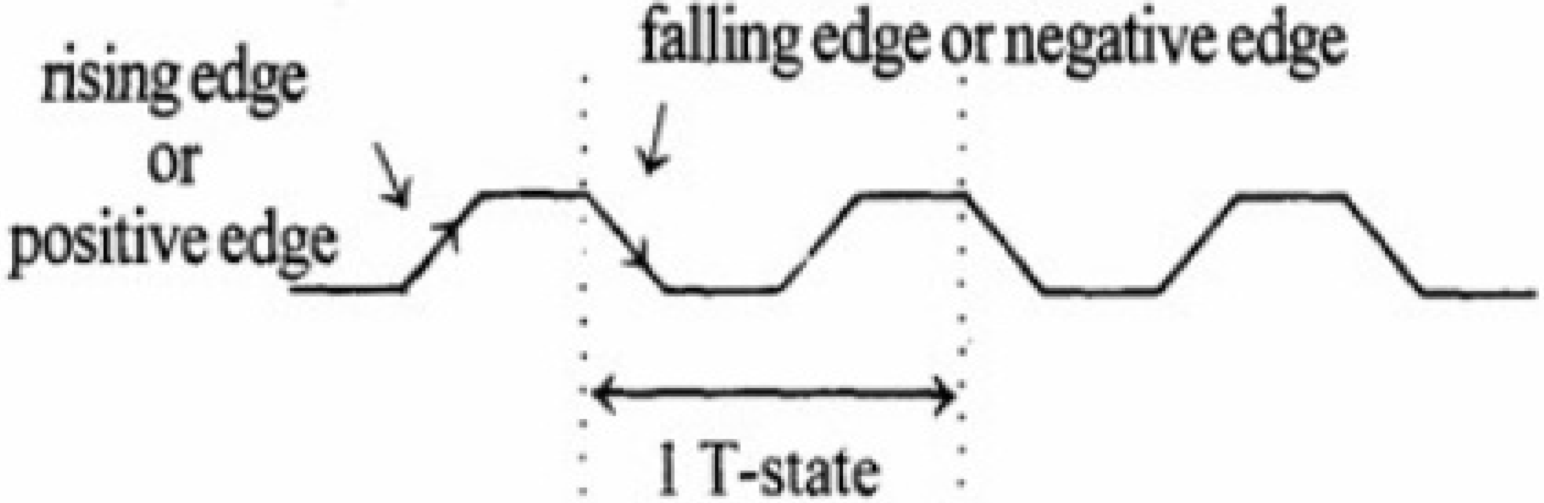
Machine Cycle:

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

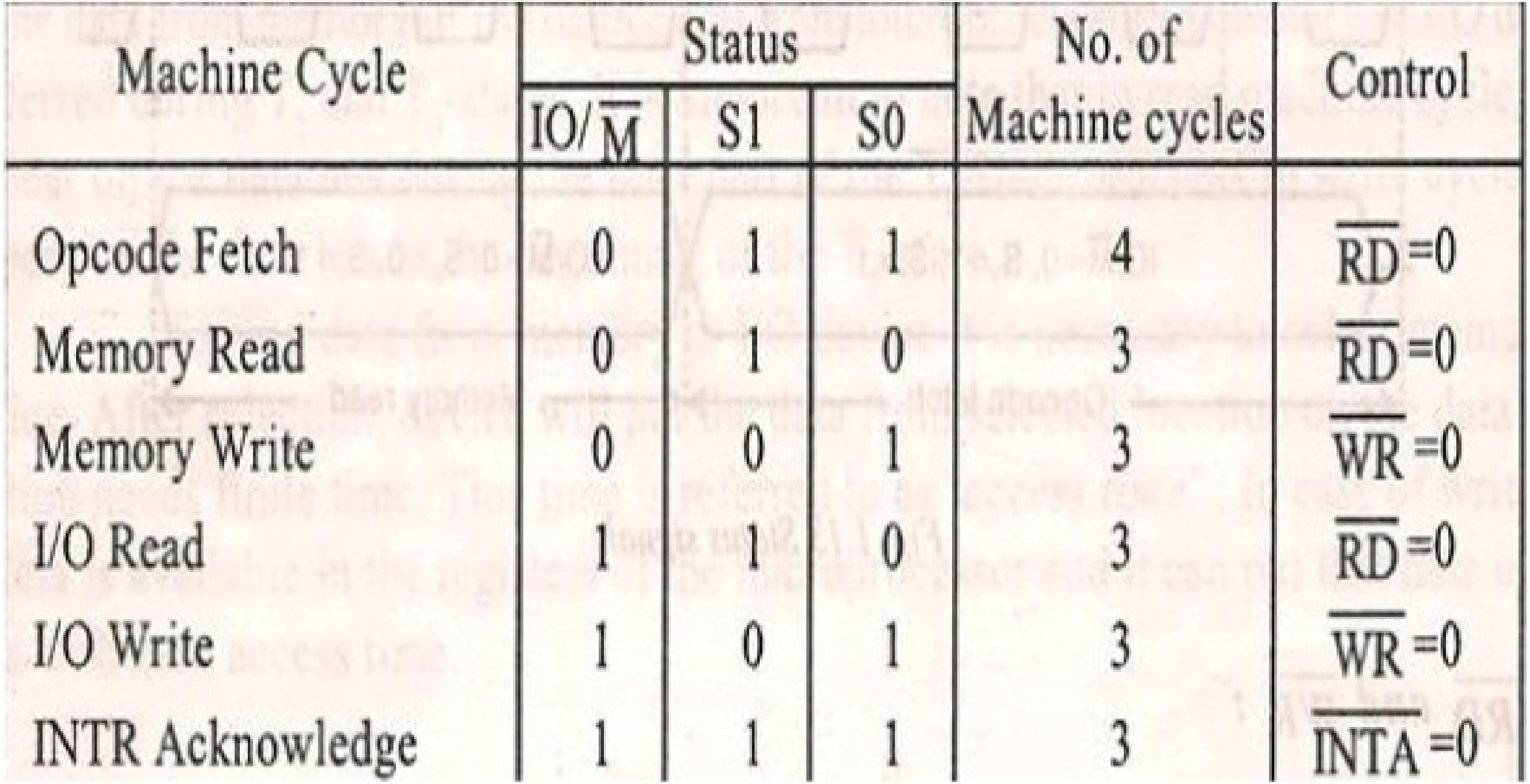


Timing diagrams

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

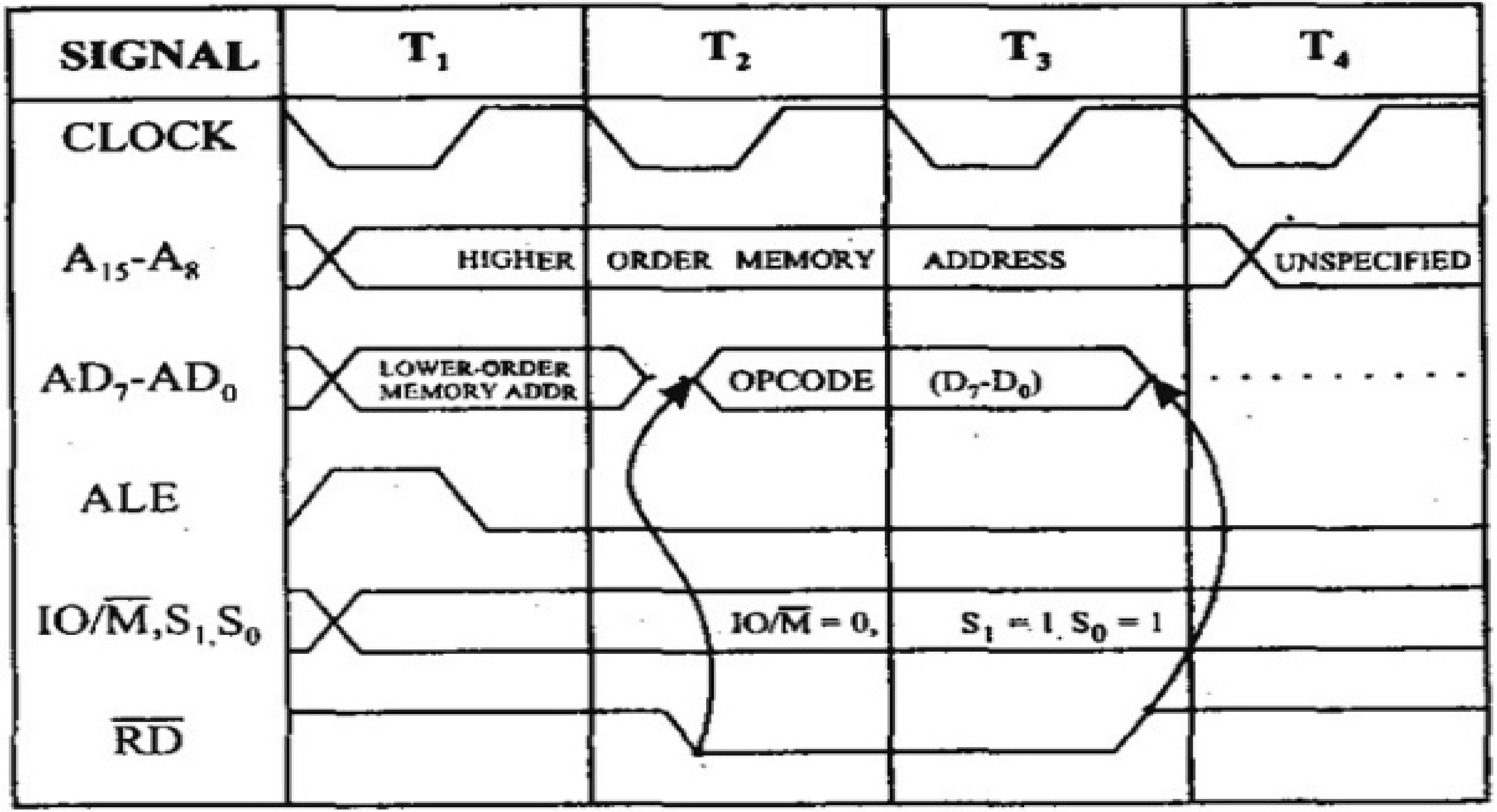
1. Op-code Fetch cycle(4T or 6T).
2. Memory read cycle (3T)
3. Memory write cycle(3T)
4. I/O read cycle(3T)
5. I/O write cycle(3T)
6. Interrupt Acknowledge cycle(6T or 12T)
7. Bus idle cycle

collected by C.Gokul AP/EEE,VCET



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

1.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.

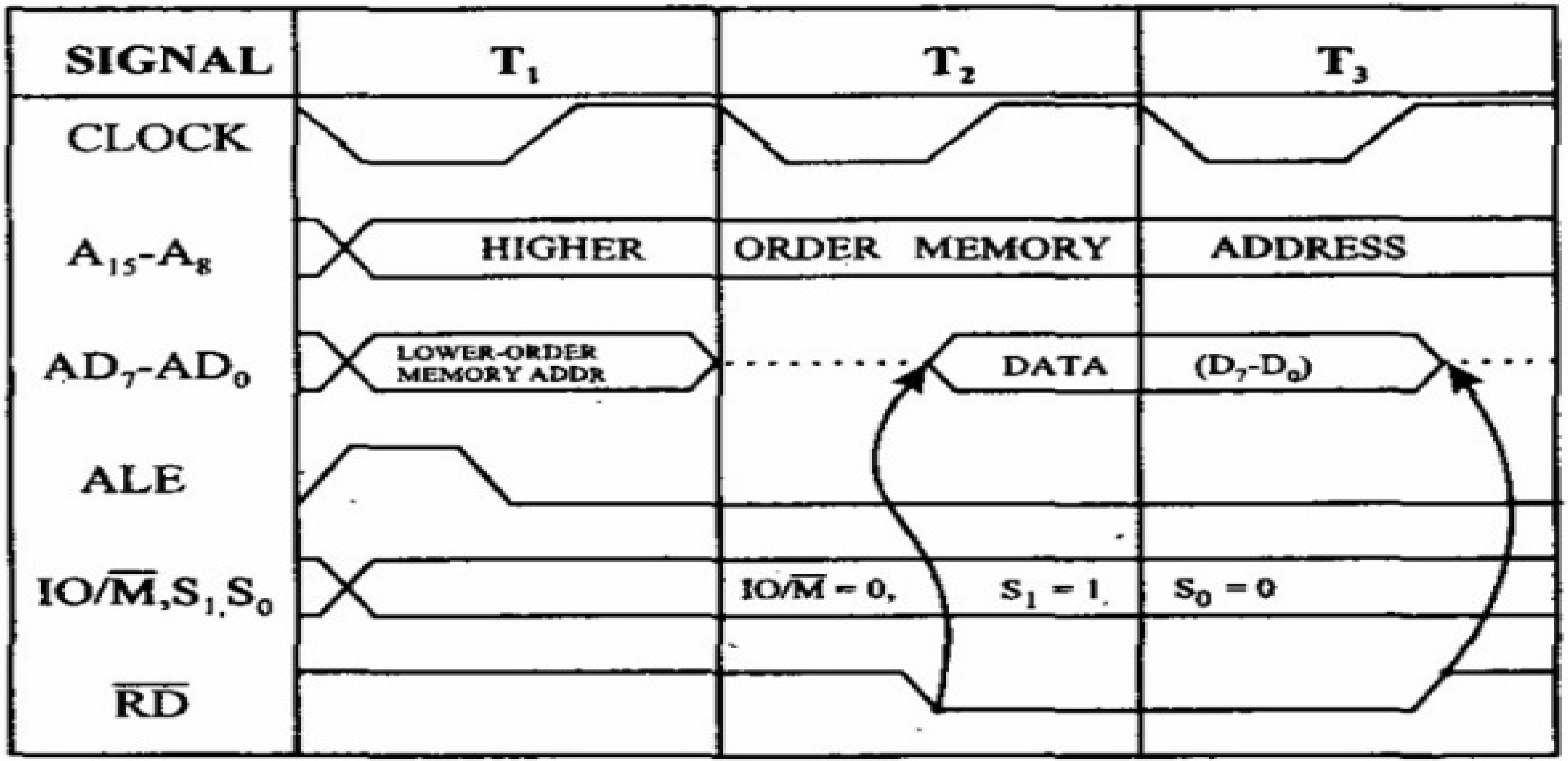
T3 State:

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.

2. 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.

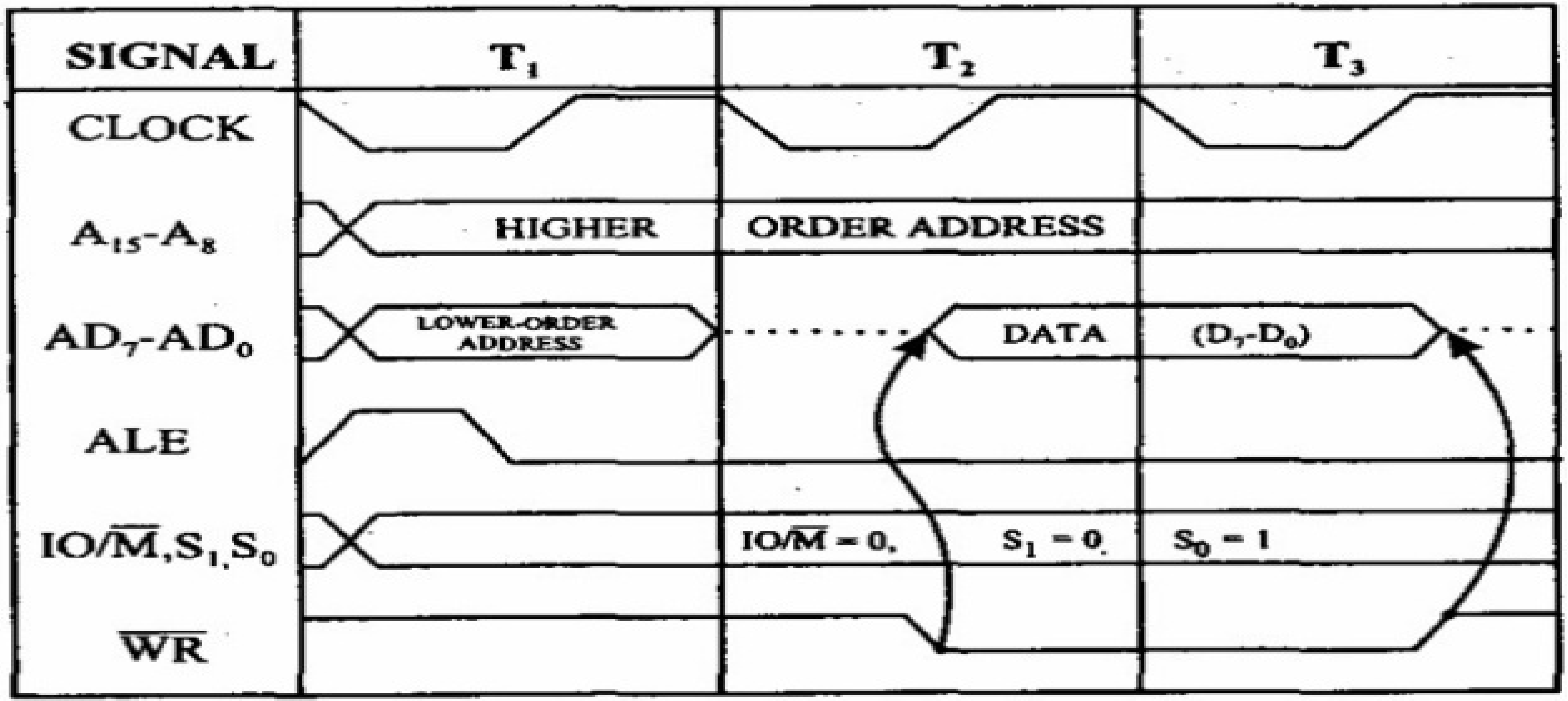
T2 state:

- 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.

3. 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.

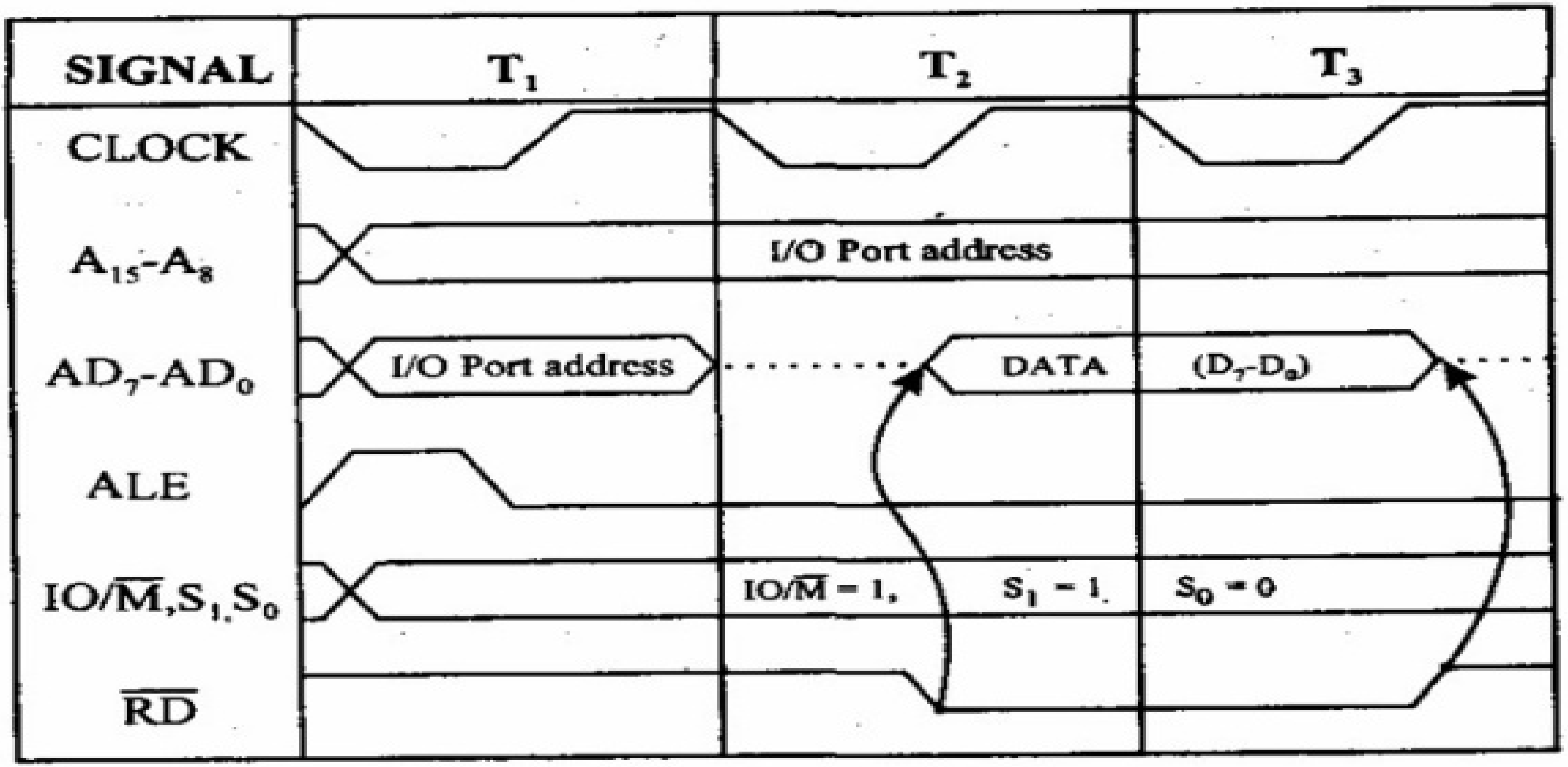
T2 state:

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

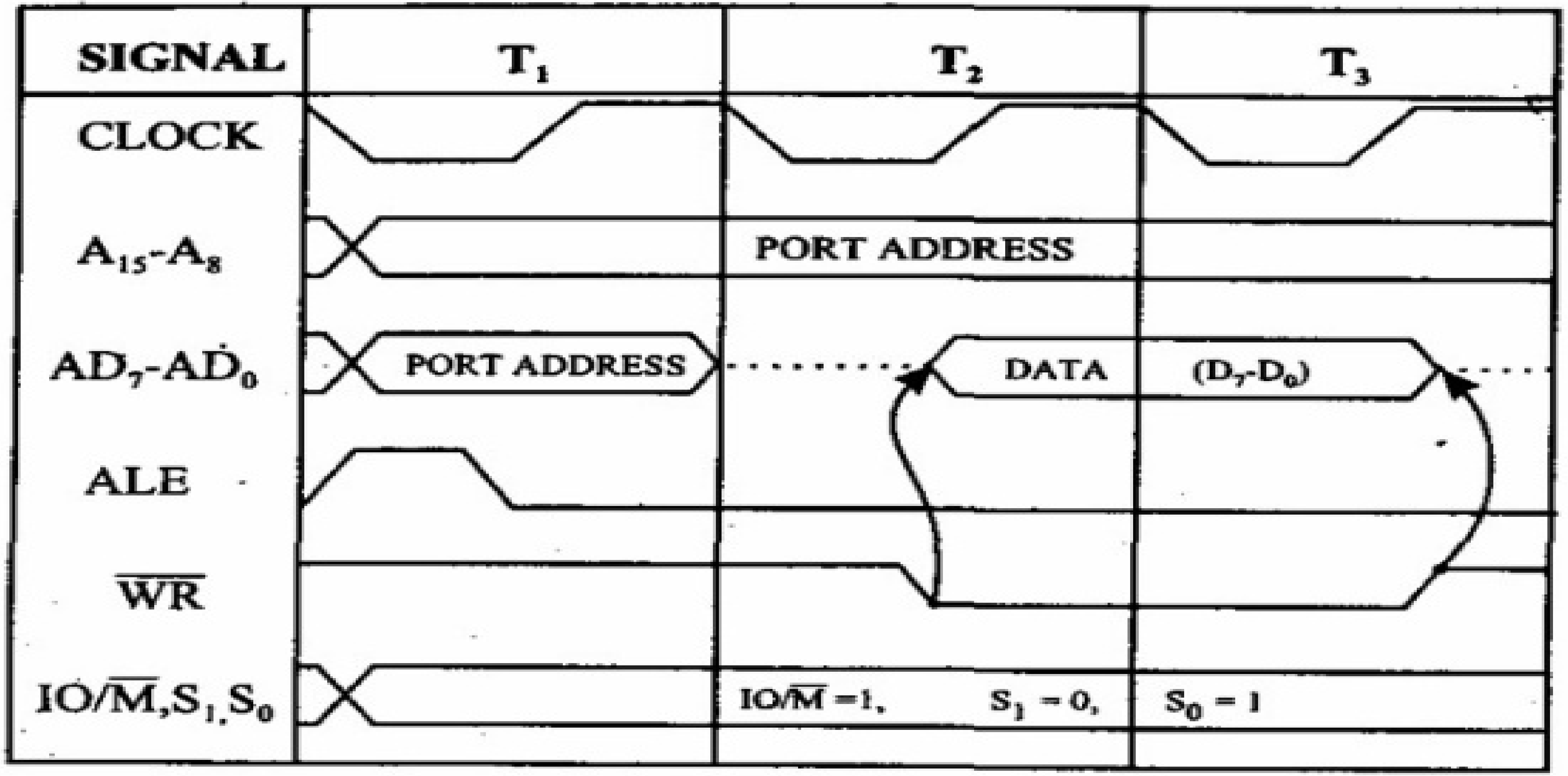
T3 State:

- 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.

4.I/O read cycle(3T)



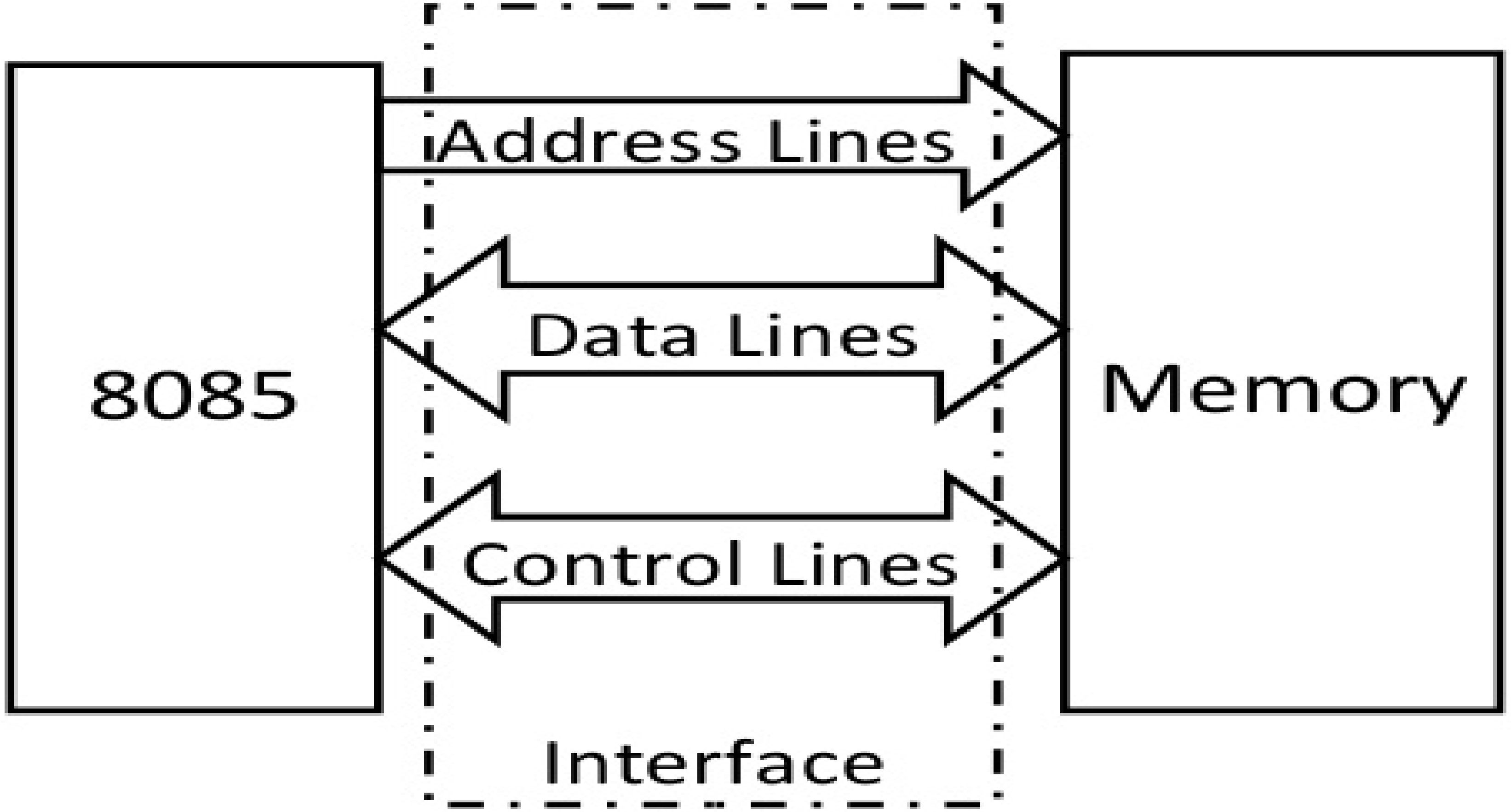
5.I/O write cycle(3T)



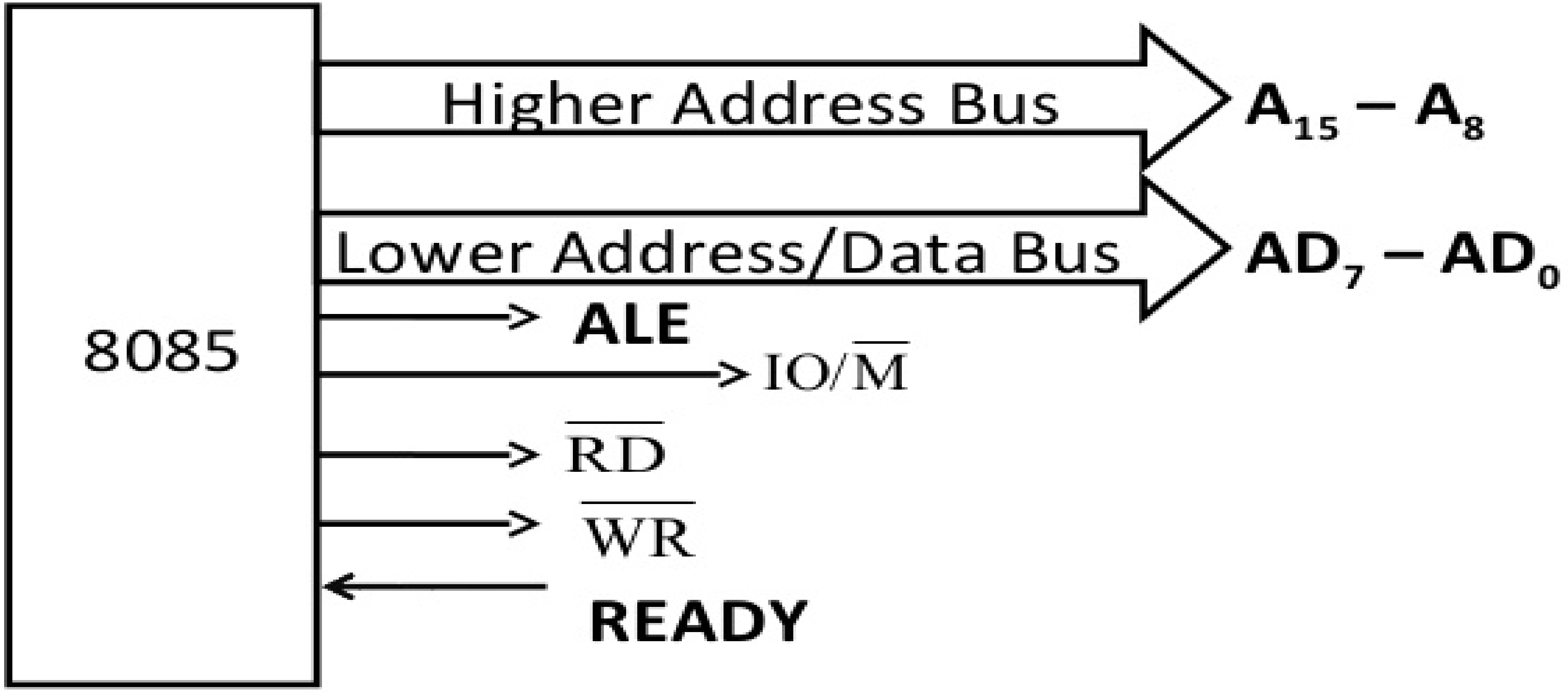
What is an Interface

- an **interface** is a concept that refers to a point of interaction between components, and is applicable at the level of both hardware and software.
- This allows a component, (such as a **graphics card** or an **Internet browser**), to function independently while using interfaces to communicate with other components via an **input/output system** and an **associated protocol**.

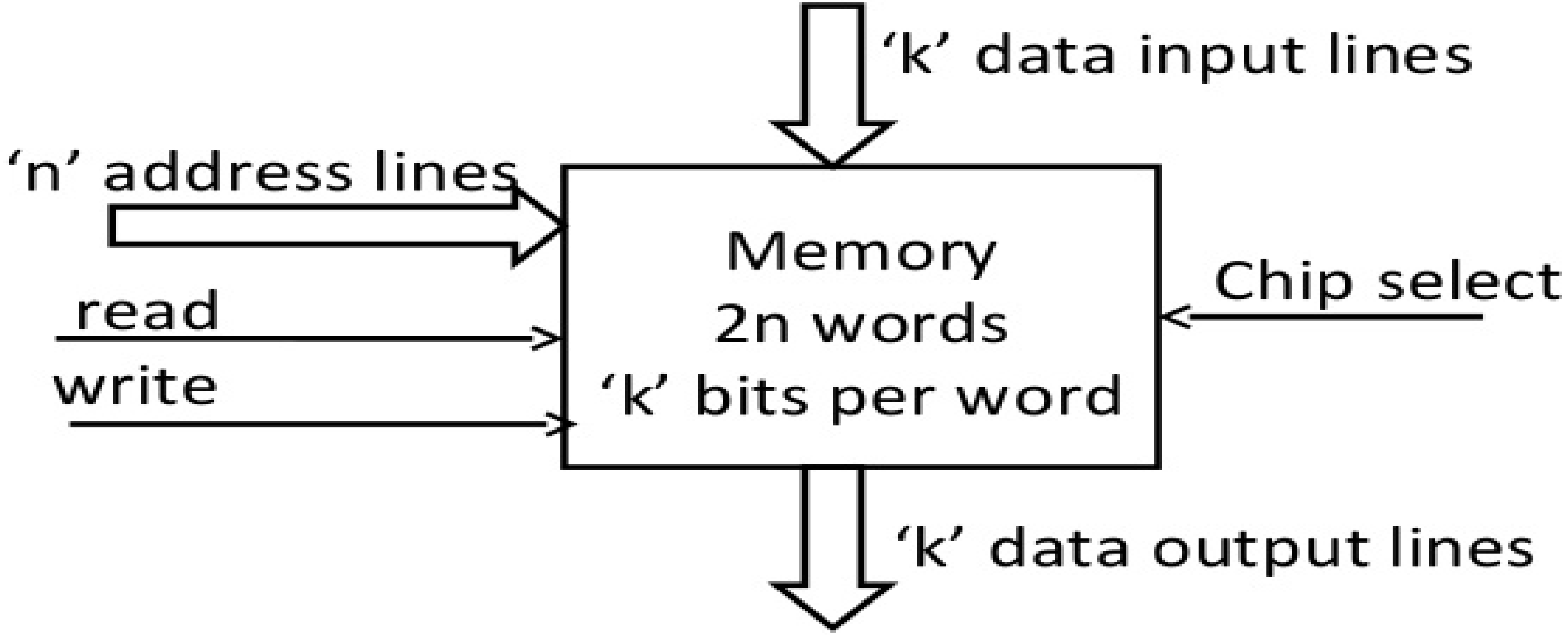
Example Block Diagram



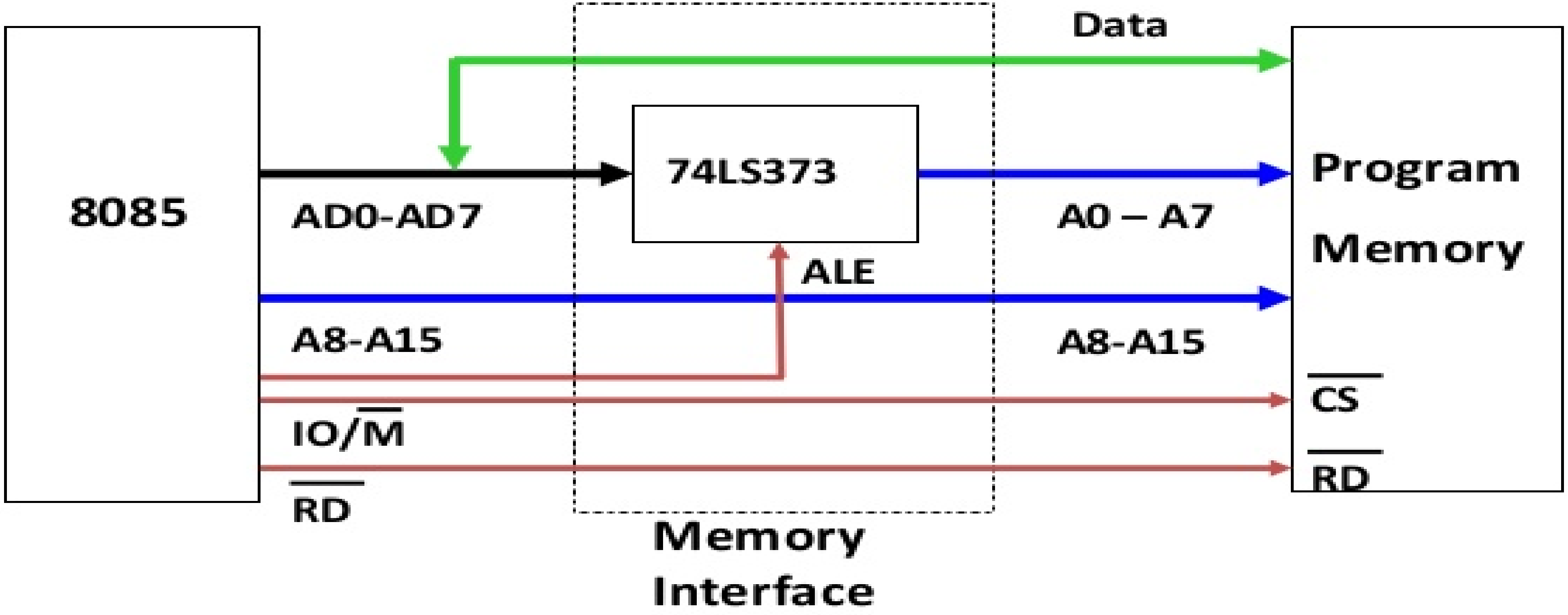
8085 Interfacing Pins



Memory Chip


‘k’ data input lines
‘n’ address lines
read
write
Memory
2n words
‘k’ bits per word
Chip select
‘k’ data output lines

8085 Interfacing with Memory chips



MODULE II

INSRUCTION SET
AND PROGRAMMING
WITH 8085

Instruction Set of 8085

An instruction is a binary pattern designed inside a microprocessor to perfo
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 TransferInstruction
- 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

0

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

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 2550H

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

/O port.

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 ofaccumulator.
- 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	Add register or memory to accumulator M

- The contents of register or memory are added to the contents of accumulator.

- If the operand is memory location, its address is specified by H-L pair. All flags are modifie

reflect the result of the addition.

- **Example:** ADD B or ADD M

Arithmetic Instructions

- The contents of register or memory and Carry Flag (CY) are added to the contents of accumulator.

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

- If the operand is memory location, its address is specified by H-L pair. All flags are modifie

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

- The contents of the register or memory location are subtracted from the contents of the accumulator.

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

- If the operand is memory location, its address is specified by H-L pair. All flags are modifie

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	R	Increment register pair by 1

- The contents of register pair are incremented by 1.The result is

- **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

- **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

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

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.

Logical Instructions

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 D0 as well as in the Carry flag.
- CY is modified according to bit D7.
- S, Z, P, AC are not affected.
- **Example:** RLC.

Logical Instructions

Opcode	Operand	Description
RRC	None	Rotate accumulator right

- Each binary bit of the accumulator is rotated right by one position.
- Bit D0 is placed in the position of D7 as well as in the Carry flag.
- CY is modified according to bit D0.
- S, Z, P, AC are not affected.
- **Example:** RRC.

Logical 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 D0.
- CY is modified according to bit D7.
- S, Z, P, AC are not affected.
- **Example:** RAL.

Logical 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 D0 is placed in the Carry flag, and the Carry flag is placed in the most significant position D7.
- CY is modified according to bit D0.
- 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

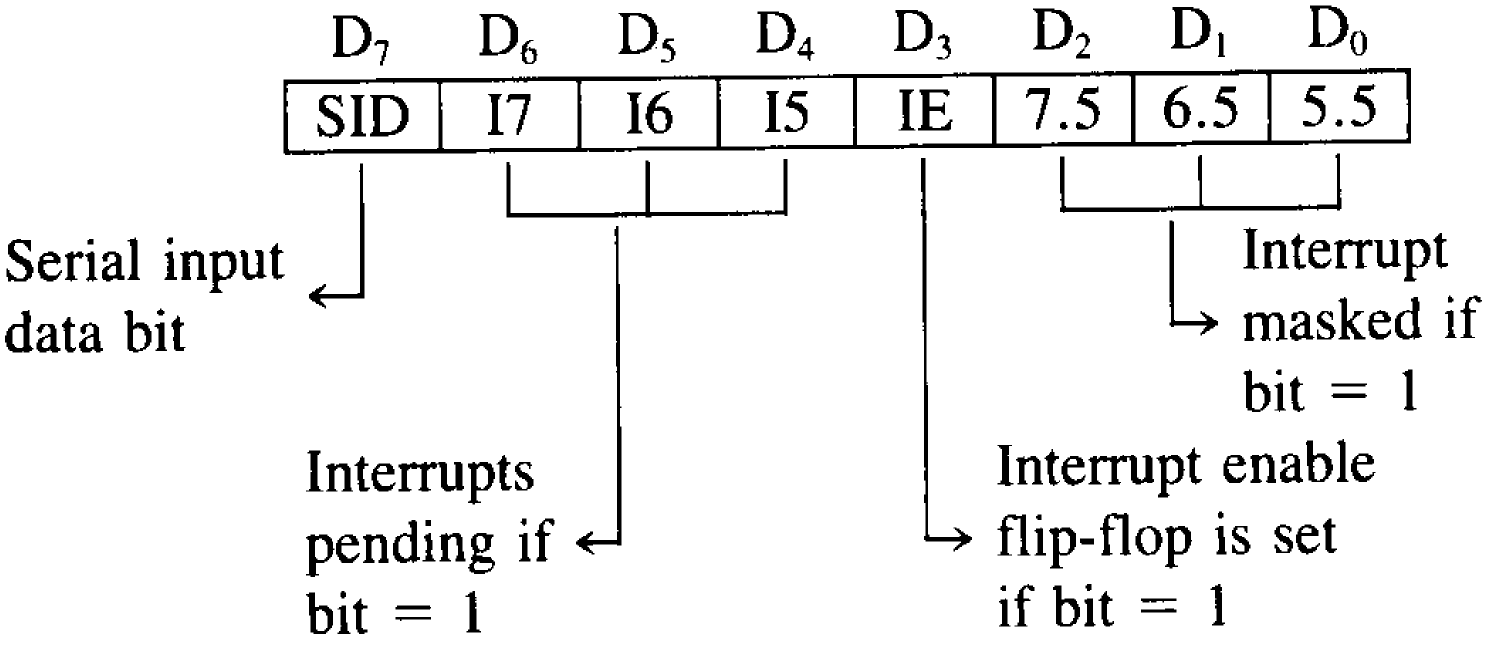
- The interrupt enable flip-flop is set and all interrupts are enabled.
- No flags areaffected.
- This instruction is necessary to re-enable the interrupts (exceptTRAP).
- **Example:** EI

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



D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
SID	I7	I6	I5	IE	7.5	6.5	5.5

Serial input data bit

Interrupts pending if bit = 1

Interrupt enable flip-flop is set if bit = 1

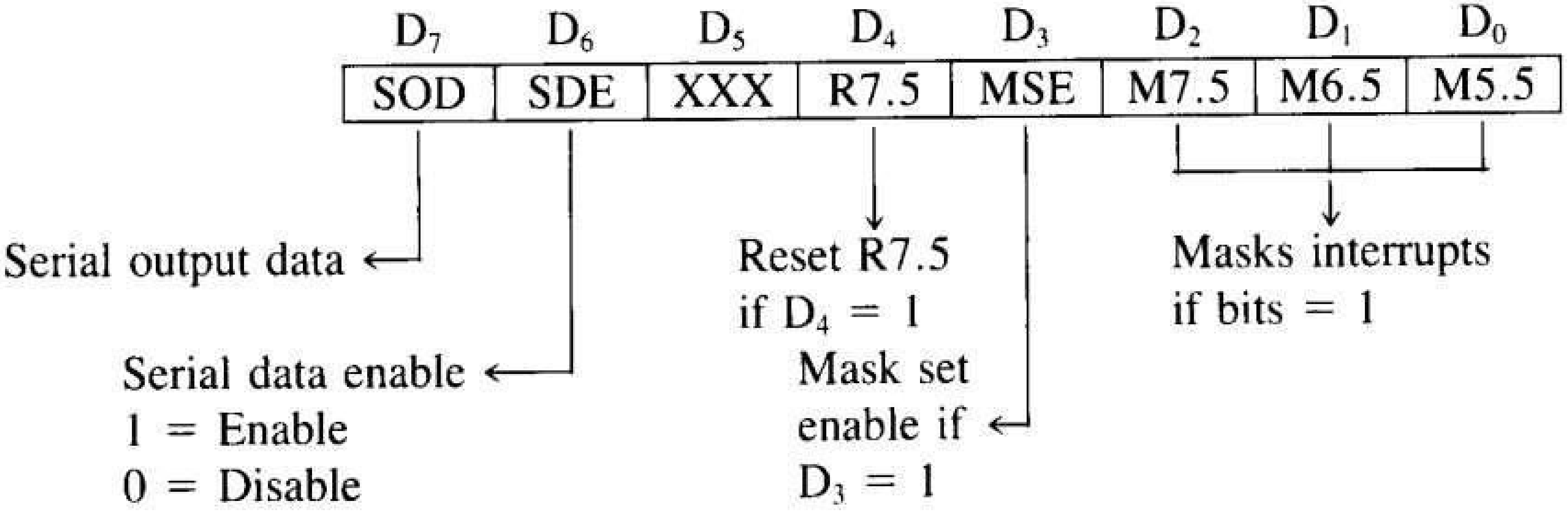
Interrupt masked if bit = 1

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



D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
SOD	SDE	XXX	R7.5	MSE	M7.5	M6.5	M5.5

Serial output data

Serial data enable
1 = Enable
0 = Disable

Reset R7.5
if $D_4 = 1$

Mask set
enable if
$D_3 = 1$

Masks interrupts
if bits = 1

Instruction format

- Instruction is a command to microprocessor to perform a given task on specified data.
- Each instruction has two parts:
  - One is the task to be performed: called **Operation code (op-code).**
  - Second is the data to be operated on called the **Operand**.
- The operand or data can be specified in various ways.
- It may include 8-bit(16-bit) data, an internal register, a memory location’s data.
- In some instructions, operand is in implicit in instruction.

Instruction Format

- Size of instruction
- Instruction : 2parts

  opcode – task to be performed

  operand – 8/16 bit data/address, internal register

  - 1 byte instruction : OPCODE

    MOV C,A
  - 2 byte instruction : OPCODE 8-bit data/address

    MVI D,#F1
  - 3 byte instruction : OPCODE low byte data/address high byte data/address

    JMP 2084

Instruction Format

An instruction is a command to the microprocessor to perform a given task on a specified data. Each instruction has two parts: one is task to be performed, called the operation code (opcode), and the second is the data to be operated on, called the operand. The operand (or data) can be specified in various ways. It may include 8-bit (or 16-bit) data, an internal register, a memory location, or 8-bit (or 16-bit) address. In some instructions, the operand is implicit.

Instruction word size

The 8085 instruction set is classified into the following three groups according to word size:

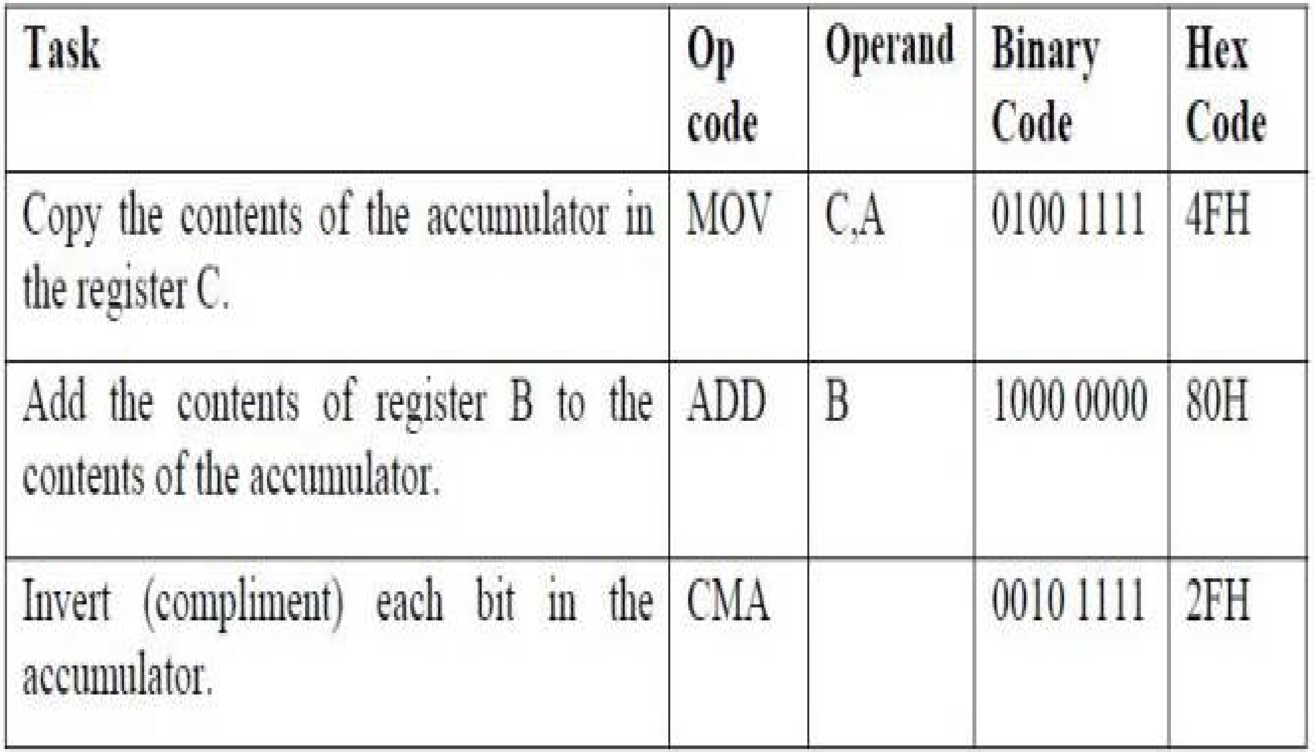
✓ One-word or 1-byte instructions
✓ Two-word or 2-byte instructions

✓ Three-word or 3-byte instructionsIn the 8085, "byte" and "word" are synonymous because it is an 8-bit microprocessor. However, instructions are commonly referred to in terms of bytes rather than words.

1 One-Byte Instructions

A 1-byte instruction includes the opcode and operand in the same byte. Operand(s) are internal register and are coded into the instruction

Table 2.1 Example for 1 byte Instruction



Task	Op code	Operand	Binary Code	Hex Code
Copy the contents of the accumulator in the register C.	MOV	C,A	0100 1111	4FH
Add the contents of register B to the contents of the accumulator.	ADD	B	1000 0000	80H
Invert (compliment) each bit in the accumulator.	CMA		0010 1111	2FH

These instructions are 1-byte instructions performing three different tasks. In the first instruction, both operand registers are specified. In the second instruction, the operand B is specified and the accumulator is assumed. Similarly, in the third instruction, the accumulator is assumed to be the implicit operand. These instructions are stored in 8- bit binary format in memory; each requires one memory location.

MOV rd, rs

rd ← rs copies contents of rs into rd.
Coded as 01 ddd sss

where ddd is a code for one of the 7 general registers which is the destination of the data, sss is the code of the source register.

Example: MOV A,B

Coded as 01111000 = 78H = 170 octal (octal was used extensively in instruction design of such processors).

ADD r

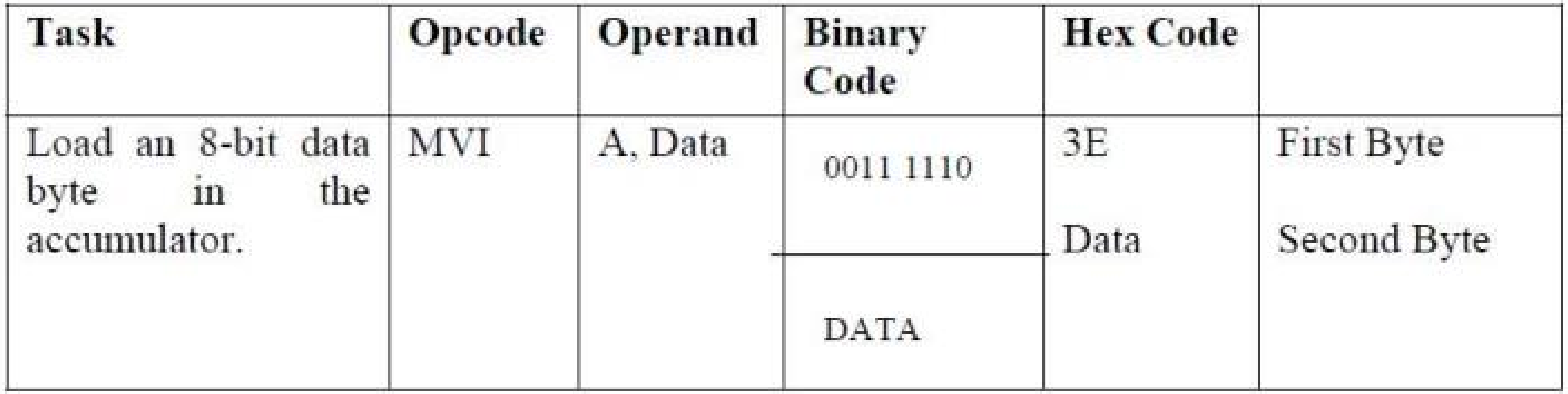
A ← A + r

2 Two-Byte Instructions

In a two-byte instruction, the first byte specifies the operation code and the second byte specifies the operand. Source operand is a data byte immediately following the opcode. For example:

Table 2.2 Example for 2 byte Instruction

Table 2.2 Example for 2 byte Instruction



Task	Opcode	Operand	Binary Code	Hex Code	
Load an 8-bit data byte in the accumulator.	MVI	A, Data	0011 1110	3E	First Byte
			DATA	Data	Second Byte

The instruction would require two memory locations to store in memory.

MVI r,data

r ← data

Example: MVI A,30H coded as 3EH 30H as two contiguous bytes.

This is an example of immediate addressing.

ADI data

A ← A + data

OUT port

0011 1110

DATA

Where port is an 8-bit device address. (Port) ← A.

Since the byte is not the data but points directly to where it is located this is called direct addressing.

3 Three-Byte Instructions

In a three-byte instruction, the first byte specifies the opcode, and the following two bytes specify the 16-bit address. Note that the second byte is the low-order address and the third byte is the high-order address.

opcode + data byte + data byte

Table 3.3 Example for 3 byte Instruction

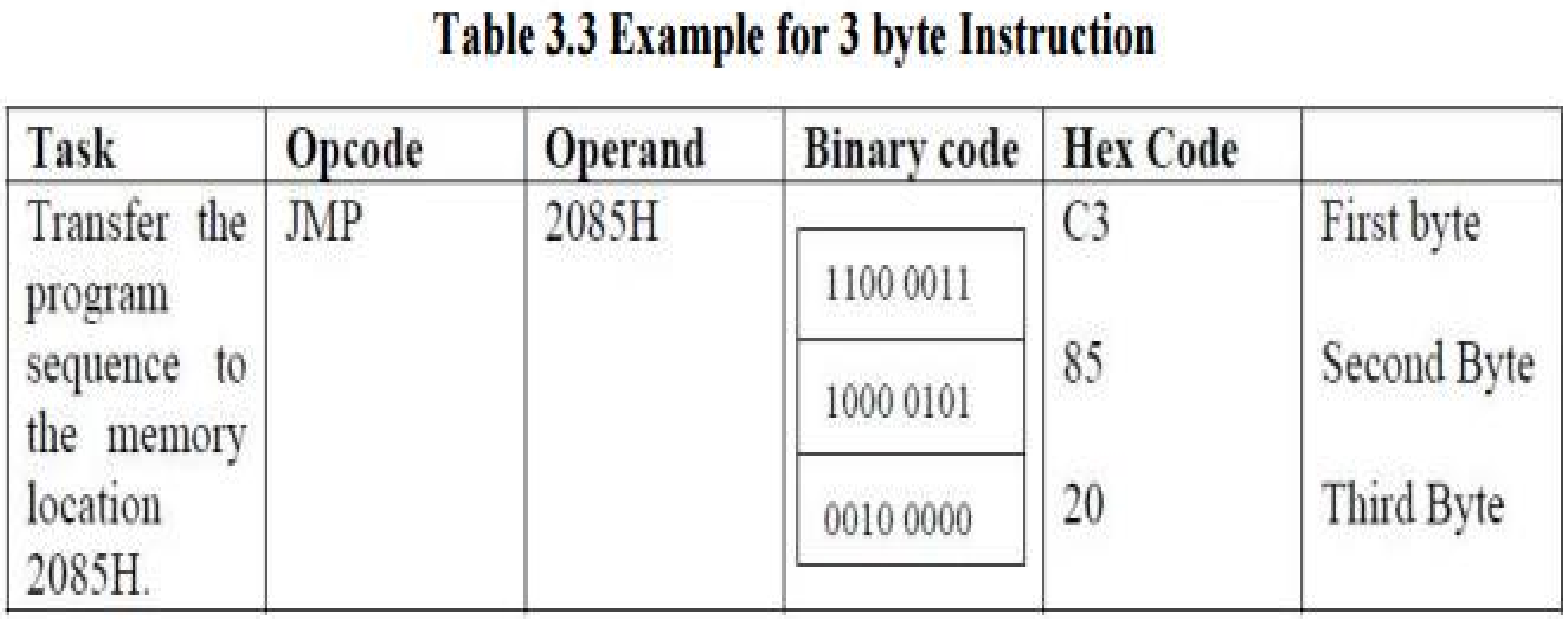


Table 3.3 Example for 3 byte Instruction

Task	Opcode	Operand	Binary code	Hex Code	
Transfer the program sequence to the memory location 2085H.	JMP	2085H	1100 0011	C3	First byte
			1000 0101	85	Second Byte
			0010 0000	20	Third Byte

This instruction would require three memory locations to store in memory.

Three byte instructions - opcode + data byte + data byte

LXI rp, data16

rp is one of the pairs of registers BC, DE, HL used as 16-bit registers. The two data bytes are 16-bit data in L H order of significance.

rp ← data16

LXI H,0520H coded as 21H 20H 50H in three bytes. This is also immediate addressing.

LDA addr

A ← (addr) Addr is a 16-bit address in L H order.

Example: LDA 2134H coded as 3AH 34H 21H. This is also an example of direct addressing.

ADDRESSING MODES OF 8085

Addressing Modes of 8085

- To perform any operation, we have to give the corresponding instructions to the microprocessor.
- In each instruction, programmer has to specify 3 things:
  - Operation to be performed.
  - Address of source of data.
  - Address of destination of result.

Addressing Modes of 8085

- The method by which the address of source of data or the address of destination of result is given in the instruction is called **Addressing Modes.**
- The term addressing mode refers to the way in which the operand of the instruction is specified.

Types of Addressing Modes

- Intel 8085 uses the following addressing modes:

  1. Direct Addressing Mode
  2. Register Addressing Mode
  3. Register Indirect Addressing Mode
  4. Immediate Addressing Mode
  5. Implicit Addressing Mode

Direct Addressing Mode

- In this mode, the address of the operand is given in the instruction itself.

LDA 2500 H	**Load the contents of memory location 2500 H in accumulator.**

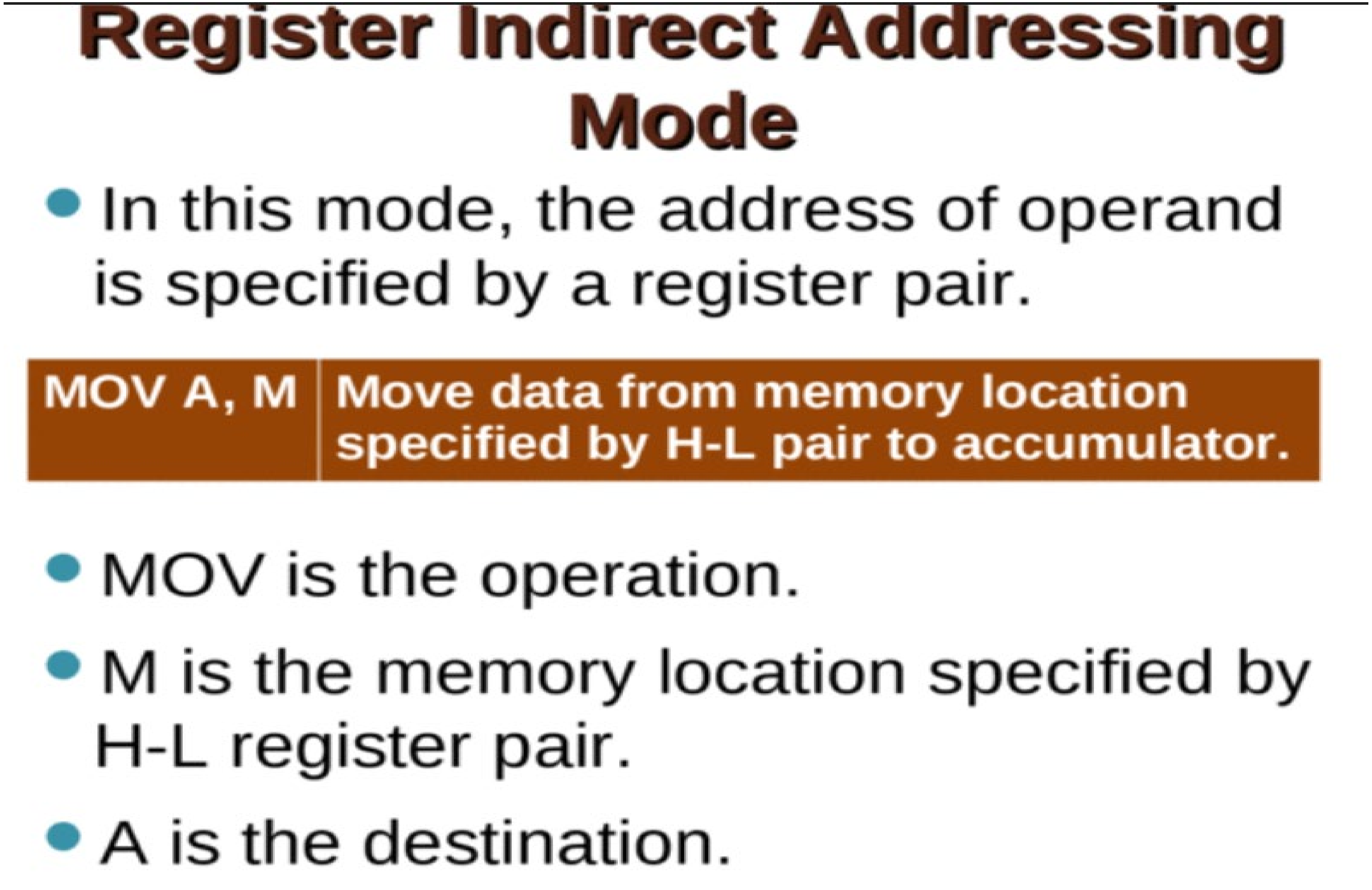
- LDA is the operation.
- 2500 H is the address of source.
- Accumulator is the destination.

Register Addressing Mode

- In this mode, the operand is in general purpose register.

MOV A, B	Move the contents of register B to A.

- MOV is the operation.
- B is the source of data.
- A is the destination.



Register Indirect Addressing Mode

- In this mode, the address of operand is specified by a register pair.

MOV A, M	Move data from memory location specified by H-L pair to accumulator.

- MOV is the operation.
- M is the memory location specified by H-L register pair.
- A is the destination.

Immediate Addressing Mode

- In this mode, the operand is specified within the instruction itself.

MVI A, 05 H	**Move 05 H in accumulator.**

- MVI is the operation.
- 05 H is the immediate data (source).
- A is the destination.

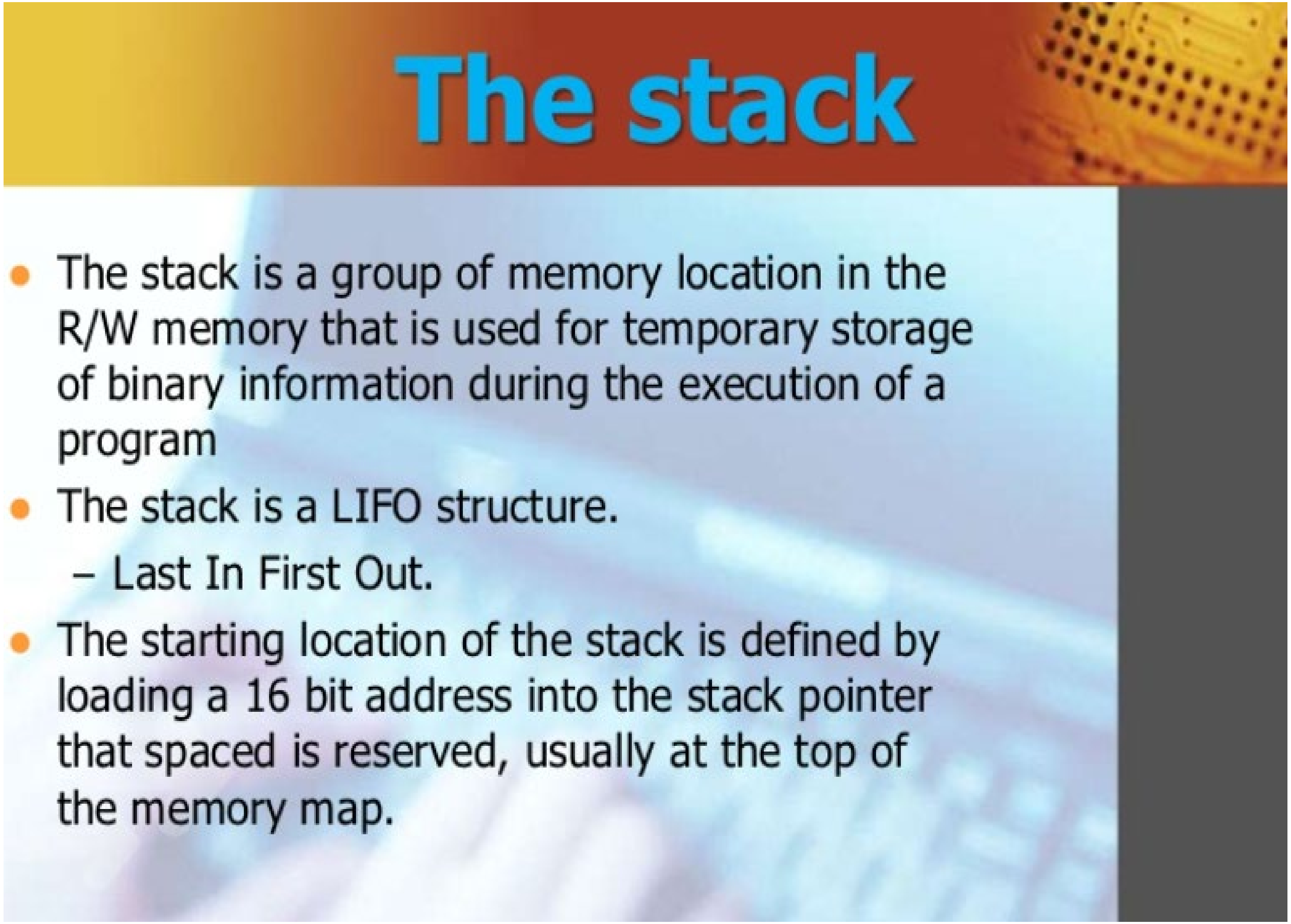
Implicit Addressing Mode

- If address of source of data as well as address of destination of result is fixed, then there is no need to give any operand along with the instruction.

CMA	Complement accumulator.

- CMA is the operation.
- A is the source.
- A is the destination.

Stack&subroutines



The stack

- The stack is a group of memory location in the R/W memory that is used for temporary storage of binary information during the execution of a program
- The stack is a LIFO structure.
  - Last In First Out.
- The starting location of the stack is defined by loading a 16 bit address into the stack pointer that spaced is reserved, usually at the top of the memory map.

The stack

- The stack normally grows backwards into memory.
- The stack can be initialized anywhere in the user memory map , but stack is initialized at the highest memory location so that there will not be any interface with the program.
- In 8085 microprocessor system the beginning of the stack is defined in the program by using the instruction
- LXI SP,16 bit.
- The LXI SP,a 16 bit state that load the 16 bit address into the stack pointer register.

Information is stored and retrieved from the stack

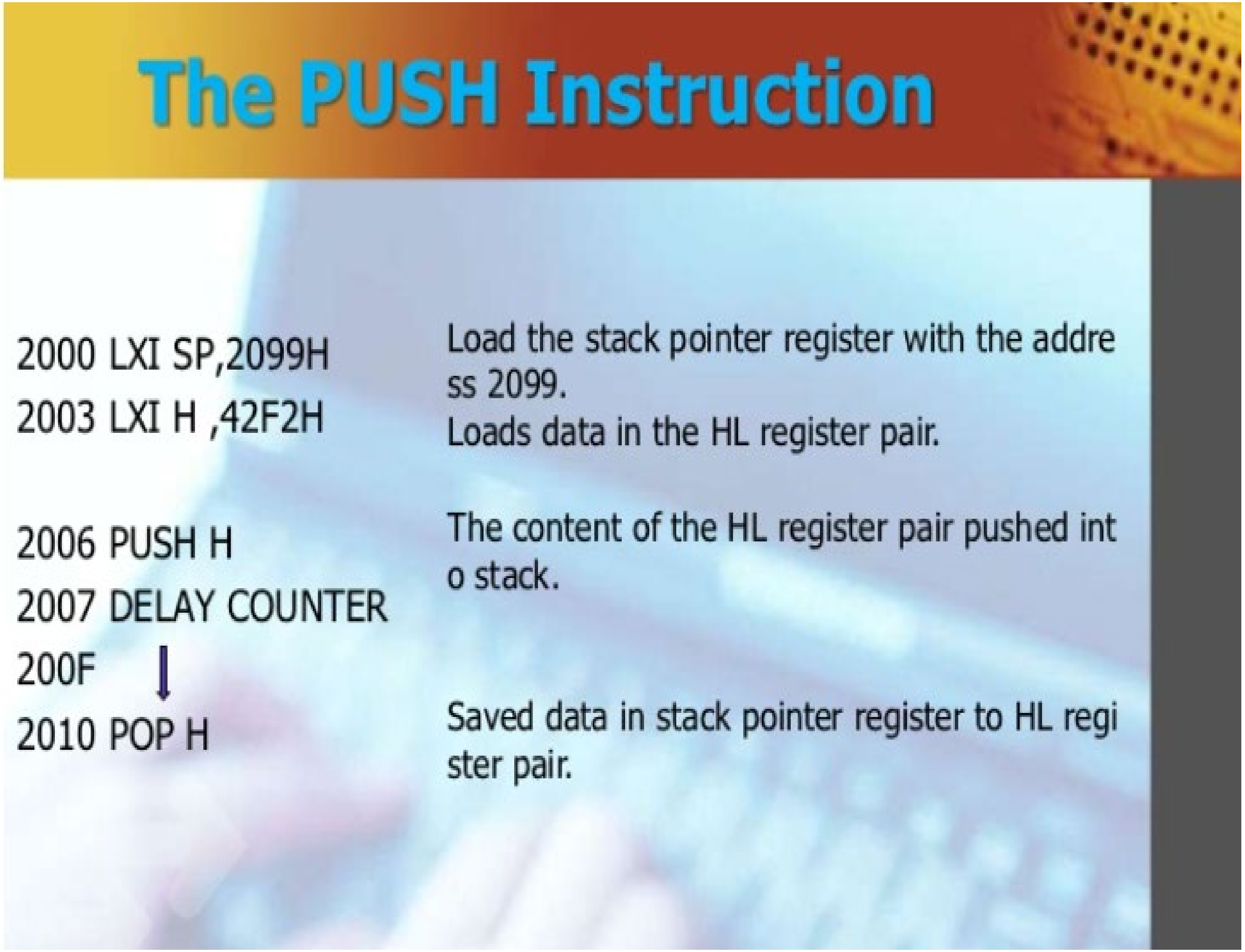
- The 8085 provide two instruction PUSH & POP for storing infor mation on the stack and retrieving it back.

- Information in the register pairs stored on the stack in reverse order by using the instruction PUSH.

- Information retrieved from the stack by using the instruction POP.

- PUSH & POP both instruction works with register pairs only.
- The storage and retrieval of the content of registers on the stack fallows the LIFO(Last-In-First-Out) sequence.

- Information in the stack location may not be destroyed until new information is stored in that memory location

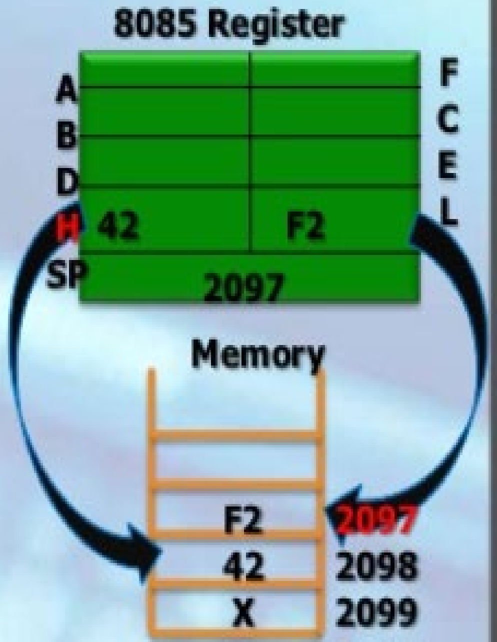


The PUSH Instruction

2000 LXI SP,2099H	Load the stack pointer register with the address 2099.
2003 LXI H ,42F2H	Loads data in the HL register pair.
2006 PUSH H	The content of the HL register pair pushed into stack.
2007 DELAY COUNTER	
200F ↓	
2010 POP H	Saved data in stack pointer register to HL register pair.

PUSH H

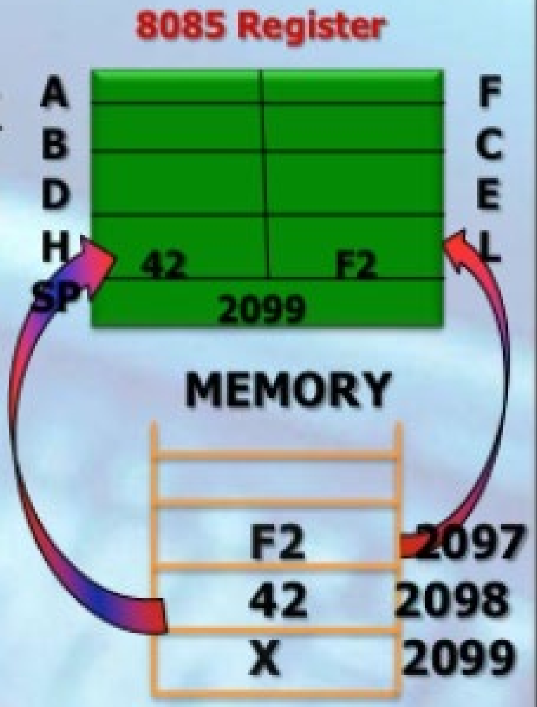
- The stack pointer is decremented by one to 2098 H , and the contents of the h register are copied to memory location 2098H.
- The stack pointer register is again decremented by one to 2097H,and the contents of the L register are copied to memory location 2097H.
- The contents of the register pair HL are not destroyed ; however HL is made available for delay counter.



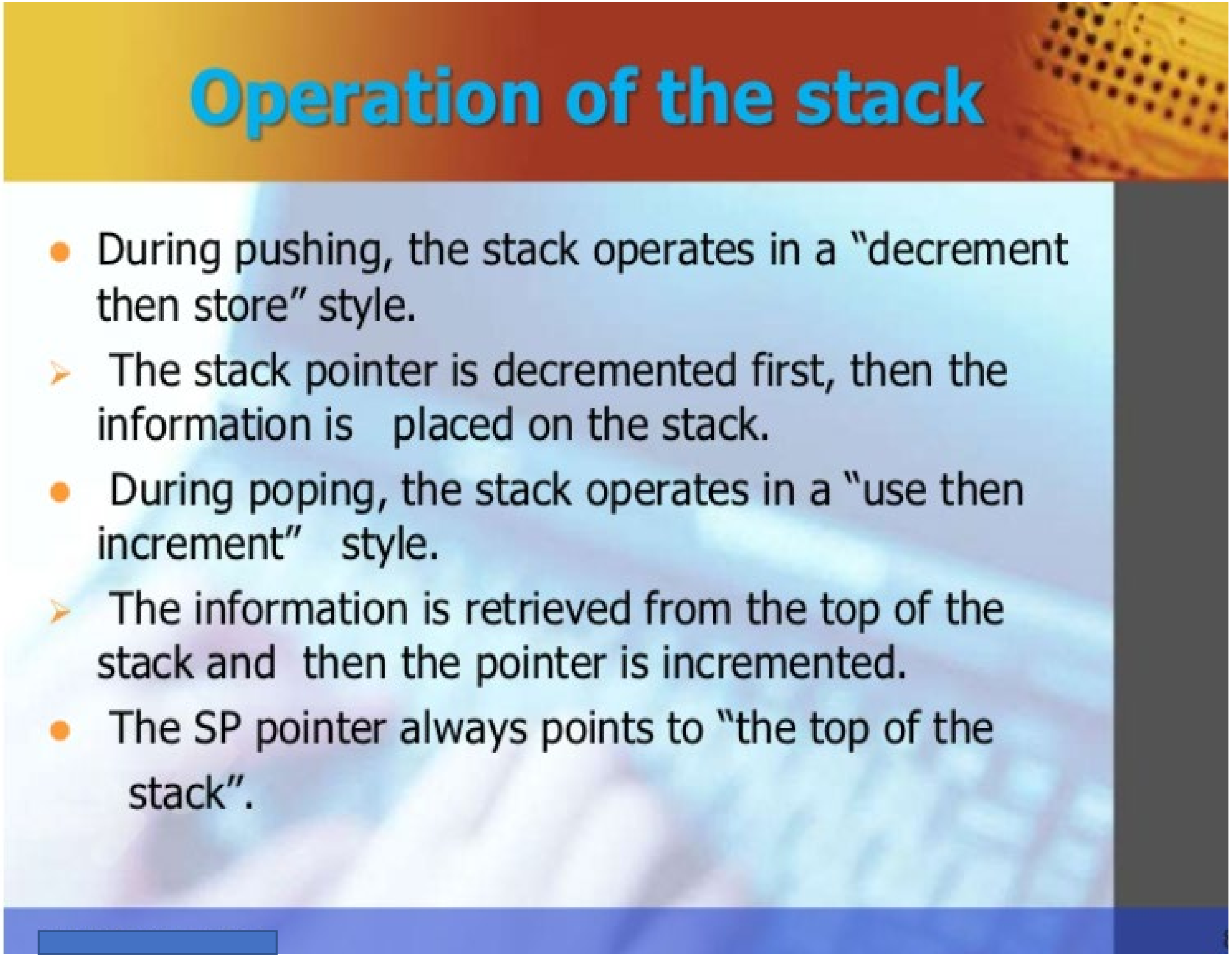
Contents on the stack &in the register after the PUSH instruction

POP H

- The contents of the top of the stack location shown by the stack pointer are copied in the L register and the stack pointer register is incremented by one to 2098 H.
- The contents of the top of the stack (now it is 2098H) are copied in the H register,and the stack pointer is incremented by one.
- The contents of memory location 2097H and 2098 are not destroyed until some other data bytes are stored in these location.



Contents on the stack and in the registers after the POP instruction

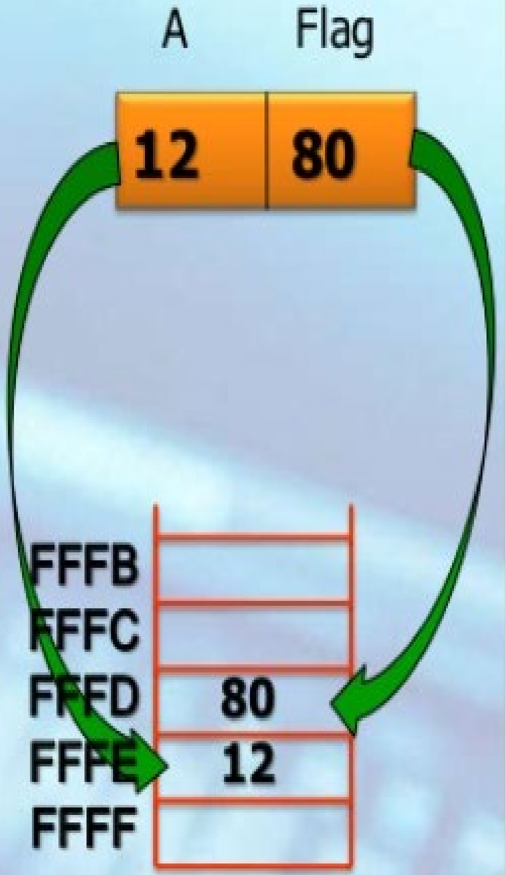


Operation of the stack

- During pushing, the stack operates in a "decrement then store" style.
- ➢ The stack pointer is decremented first, then the information is placed on the stack.
- During poping, the stack operates in a "use then increment" style.
- ➢ The information is retrieved from the top of the stack and then the pointer is incremented.
- The SP pointer always points to "the top of the stack".

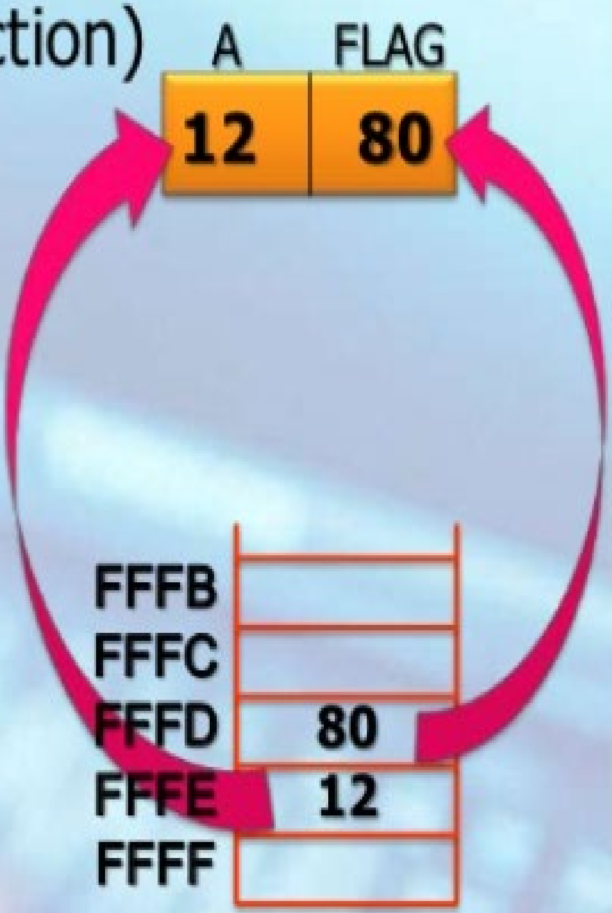
PUSH PSW Register Pair

- PUSH PSW (1 Byte Instruction)
  - Decrement SP
  - Copy the contents of register A to the memory location pointed to by SP
  - Decrement SP
  - Copy the contents of Flag register to the memory location pointed to by SP



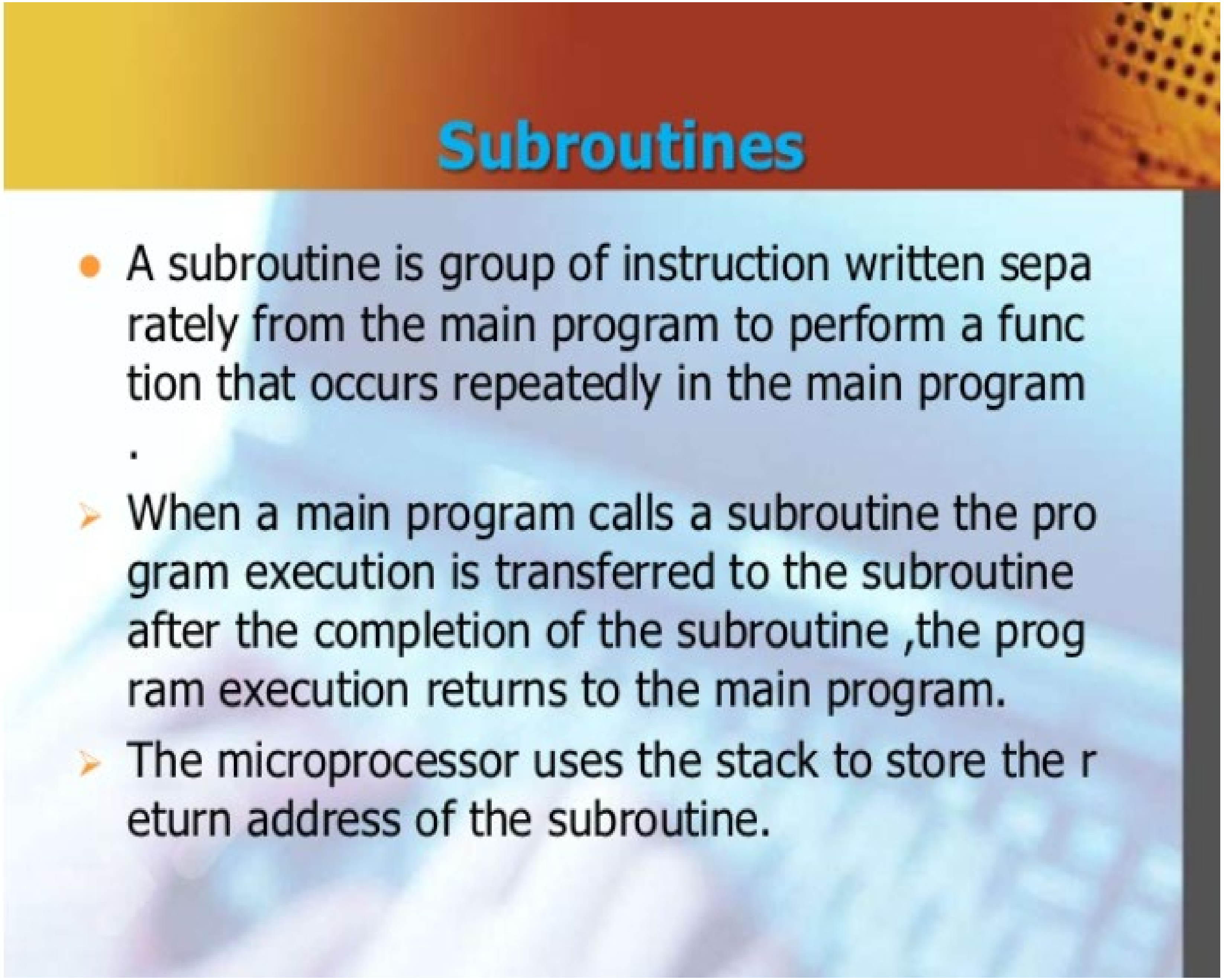
Pop PSW Register Pair

- POP PSW (1 Byte Instruction)
  - Copy the contents of the memory location pointed to by the SP to Flag register
  - Increment SP
  - Copy the contents of the memory location pointed to by the SP to register A
  - Increment SP



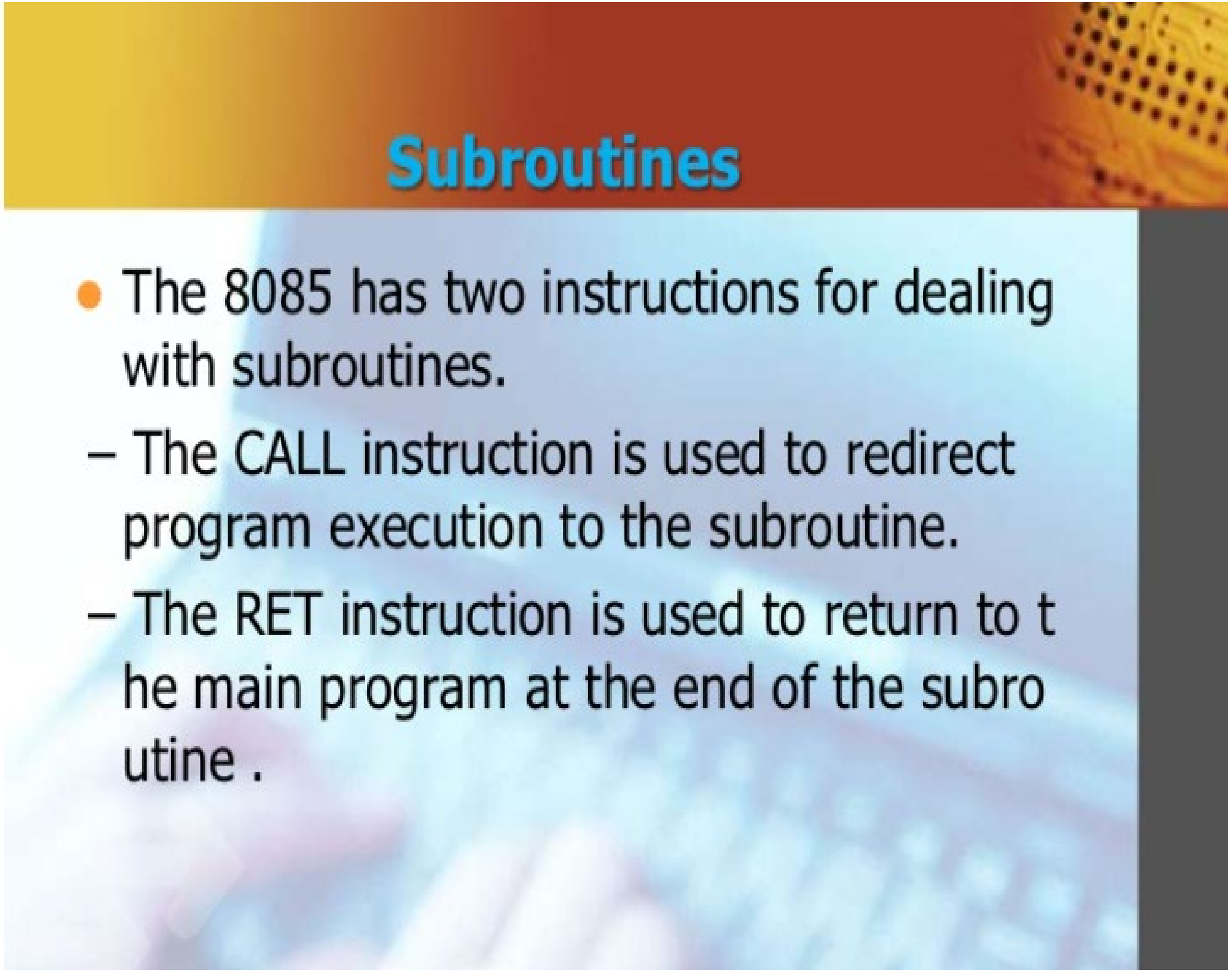
A	FLAG
12	80

FFFB	
FFFC	
FFFD	80
FFFE	12
FFFF	



Subroutines

- A subroutine is group of instruction written separately from the main program to perform a function that occurs repeatedly in the main program .
- When a main program calls a subroutine the program execution is transferred to the subroutine after the completion of the subroutine ,the program execution returns to the main program.
- The microprocessor uses the stack to store the return address of the subroutine.

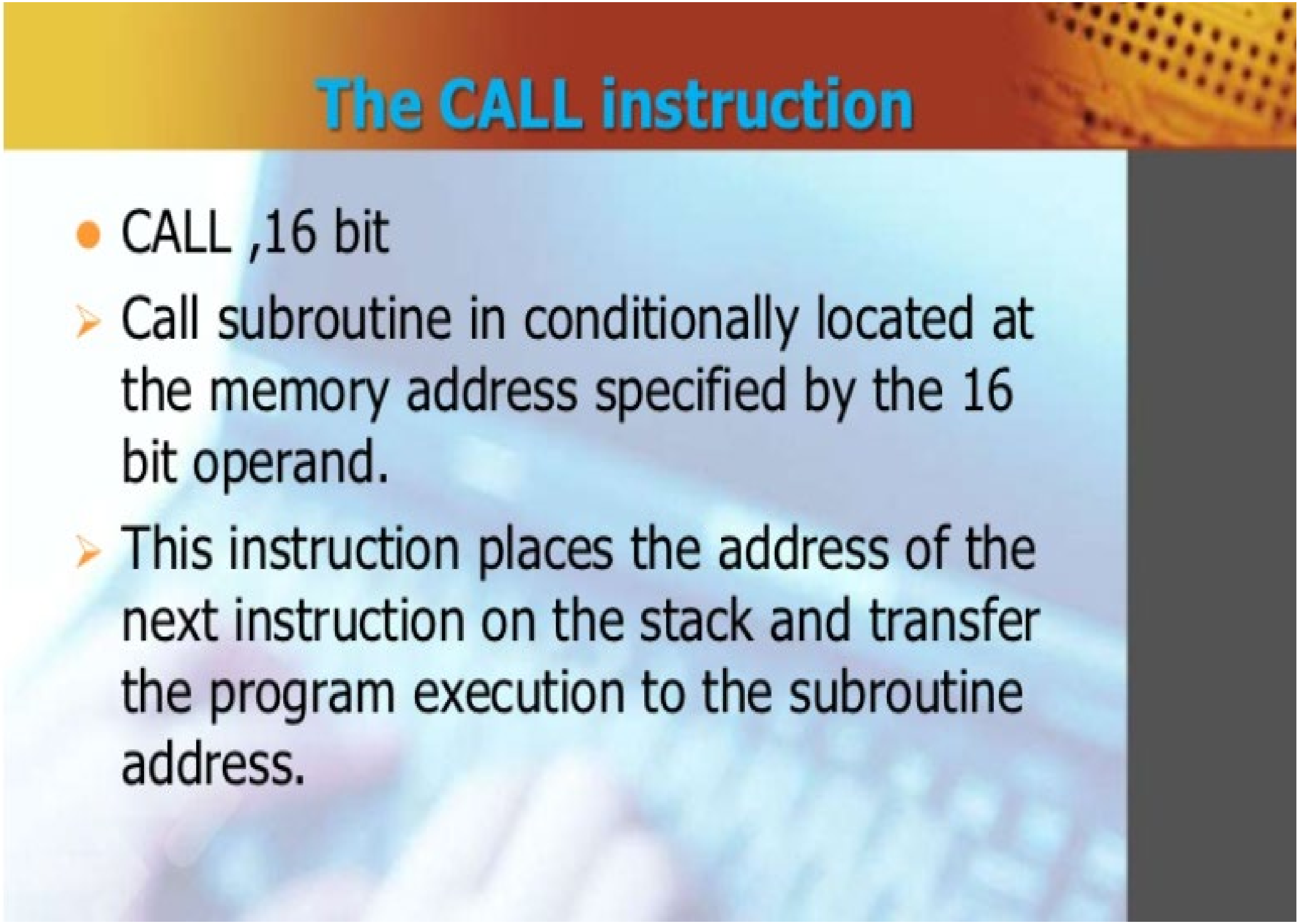


Subroutines

- The 8085 has two instructions for dealing with subroutines.

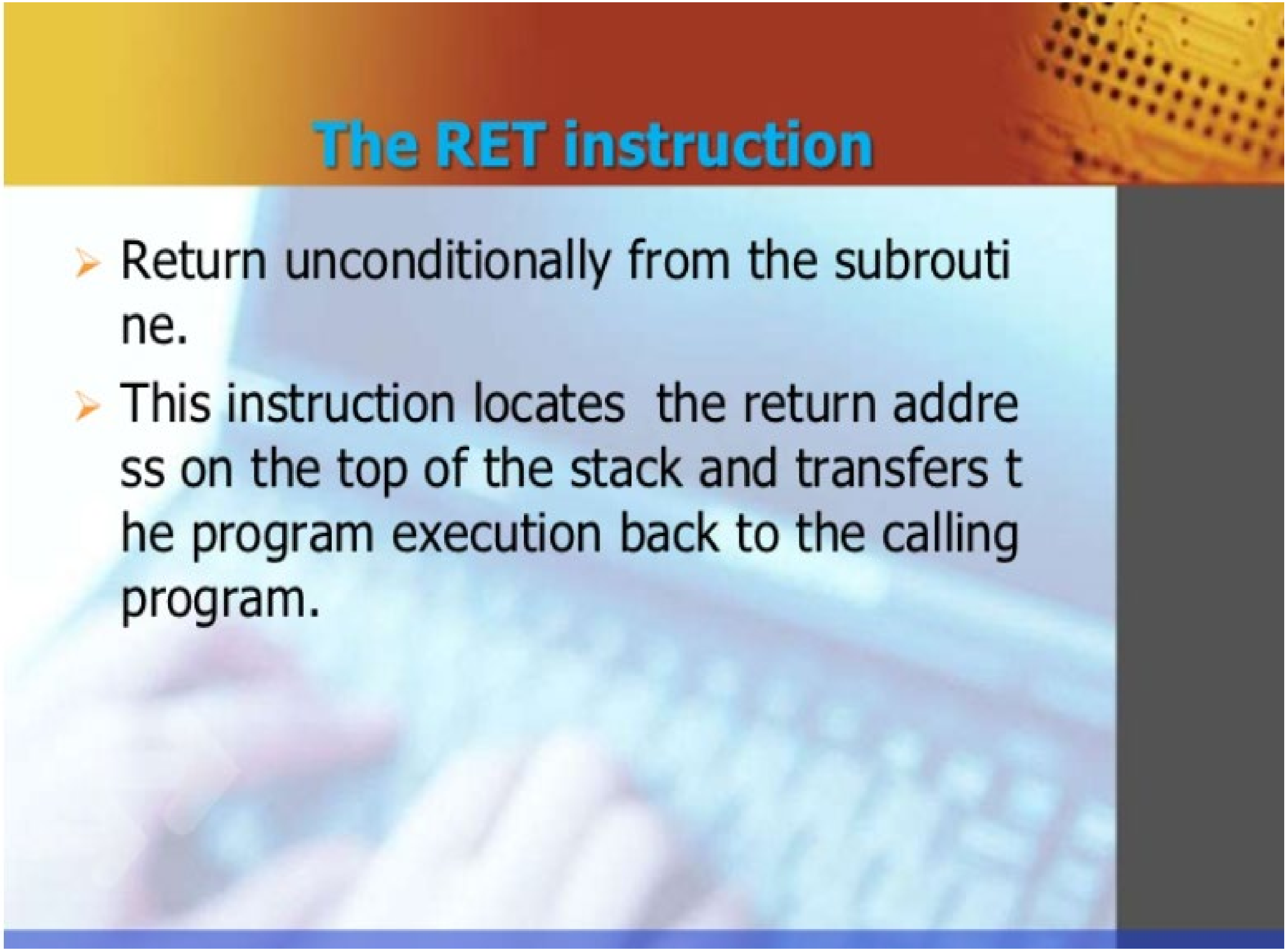
– The CALL instruction is used to redirect program execution to the subroutine.

– The RET instruction is used to return to t he main program at the end of the subro utine .



The CALL instruction

- CALL ,16 bit
- Call subroutine in conditionally located at the memory address specified by the 16 bit operand.
- This instruction places the address of the next instruction on the stack and transfer the program execution to the subroutine address.



The RET instruction

- Return unconditionally from the subroutine.
- This instruction locates the return address on the top of the stack and transfers the program execution back to the calling program.

General characteristics of CALL & RTE instruction

1. The CALL instructions are 3-byte instruction; the second byte specifies the low order byte ,and the third byte specifies the high order byte of the subroutine address.
2. The return instruction are 1-byte instructions.
3. A CALL instruction must be used in conjunction with a return instruction in the subroutine .

Necessary steps to implement a subroutine

- The stack pointer register must be initialized ,preferably at the highest memory location of the R/W memory.
- The call instruction should be used in the main program accompanied by the RET instruction in the subroutine.

Conditional CALL and RTE Instructions

- The 8085 supports conditional CALL and co nditional RTE instructions.
  - The same conditions used with conditional JUMP instructions can be used.
  - CC, call subroutine if Carry flag is set.
  - CNC, call subroutine if Carry flag is not set
  - RC, return from subroutine if Carry flag is set
  - RNC, return from subroutine if Carry flag is not set.

Assembler directives

Definition: **Assembler directives** are the instructions used by the **assembler** at the time of assembling a source program. More specifically, we can say, **assembler directives** are the commands or instructions that control the operation of the **assembler**.

Thus assembler is used to convert assembly language into machine code so that it can be understood and executed by the processor.

Therefore, to control the generation of machine codes from the assembly language, assembler directives are used.

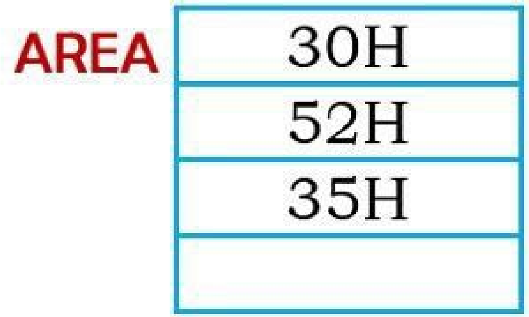
assembler directives:

- show the beginning and end of a program provided to the assembler,
- used to provide storage locations to data,
- used to give values to variables,
- define the start and end of different segments, procedures or macros etc. of a program. Assembler Directives of 8085

The assembler directives given below are used by 8085 and 8086 assemblers:

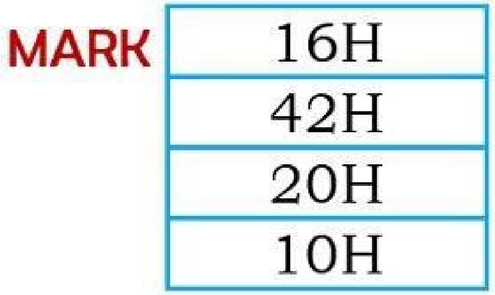
DB: *Define Byte*
This directive is used for the purpose of allocating and initializing single or multiple data bytes.

Memory name AREA has three consecutive locations where 30H, 52H and 35H are to be stored.



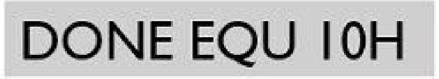
DW: *Define Word*
It is used for initialising single or multiple data words (16-bit).

These two 16-bit data 1020H and 4216H are stored at 4 consecutive locations in the memory MARK.



END: *End of program*This directive is used at the time of program termination.

EQU: *Equate*
It is used to assign any numerical value or constant to the variable.

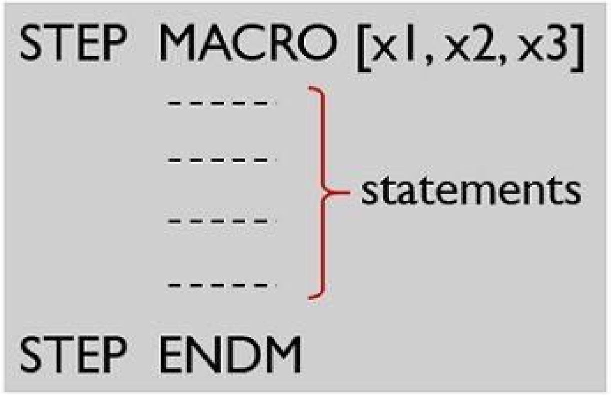


```
DONE EQU 10H
```

Variable name ‘DONE’ has value 10H

MACRO: *Represents beginning*
Shows the beginning of macro along with defining name and parameters.

ENDM: *End of macro*

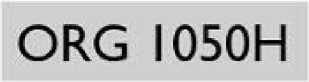


```
STEP  MACRO [x1, x2, x3]
      -----
      -----
      -----   } statements
      -----
STEP  ENDM
```

ENDM indicates the termination of macro.

where macroname (STEP) is specified by the user.

ORG: *Origin*
This directive is used at the time of assigning starting address for a module or segment.



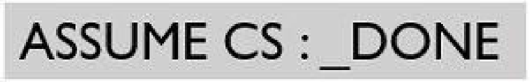
```
ORG 1050H
```

By this instruction, the assembler gets to know that the statements following this instruction, must be stored in the memory location beginning with address 1050H.

Assembler Directives of 8086

These assembler directives are specifically used by 8086:

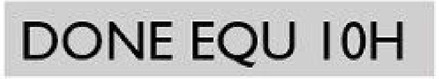
ASSUME: *Shows the segment name to the assembler*
It provides information to the assembler regarding the name of the program or data segment for that particular segment.



```
ASSUME CS : _DONE
```

This directive is used at the time of program termination.

EQU: *Equate*
It is used to assign any numerical value or constant to the variable.



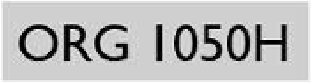
```
DONE EQU 10H
```

Variable name ‘DONE’ has value 10H

MACRO: *Represents beginning*
Shows the beginning of macro along with defining name and parameters.

ENDM: *End of macro*
ENDM indicates the termination of macro.

ORG: *Origin*
This directive is used at the time of assigning starting address for a module or segment.



```
ORG 1050H
```

By this instruction, the assembler gets to know that the statements following this instruction, must be stored in the memory location beginning with address 1050H.

Assembler Directives (cont..)

ASSUME		
DB	-	Defined Byte.
DD	-	Defined Double Word
DQ	-	Defined Quad Word
DT	-	Define Ten Bytes
DW	-	Define Word

Assembler Directives (cont..)

- **ASSUME Directive** - The ASSUME directive is used to tell the assembler that the name of the logical segment should be used for a specified segment.

  - **Example:**

  ASUME CS:CODE ;This tells the assembler that the logical segment named CODE contains the instruction statements for the program and should be treated as a code segment.

  ASUME DS:DATA ;This tells the assembler that for any instruction which refers to a data in the data segment, data will found in the logical segment DATA.

Assembler Directives (cont..)

- **DW** - The DW directive is used to define a variable of type word or to reserve storage location of type word in memory.
  - **Example:**

  MULTIPLIER DW 437Ah ; this declares a variable of type word and named it as MULTIPLIER. This variable is initialized with the value 437Ah when it is loaded into memory to run.

  EXP1 DW 1234h, 3456h, 5678h ; this declares an array of 3 words and initialized with specified values.

  STOR1 DW 100 DUP(0); Reserve an array of 100 words of memory and initialize all words with 0000.Array is named as STOR1.

Assembler Directives (cont..)

- **END** - End Program End Procedure
- **ENDP** - End Segment Equate
- **ENDS** - Align on Even Memory Address
- **EQU** -
- **EVEN** -
- **EXTRN**

Assembler Directives (cont..)

- **ENDS** - This ENDS directive is used with name of the segment to indicate the end of that logic segment.
- **Example:**

```
CODE        SEGMENT    ;Hear it Start the logic
                       ;segment  containing code
; Some instructions statements to perform the logical
;operation
CODE        ENDS       ;End of segment named as
                       ;CODE
```

Assembler Directives (cont..)

- **GROUP** - The **GROUP** directive is used to group the logical segments named after the directive into one logical group segment.
- **INCLUDE** - This **INCLUDE** directive is used to insert a block of source code from the named file into the current source module.

Assembler Directives (cont..)

- **PROC** - Procedure
- **PTR** - Pointer
- **PUBLC**
- **SEGMENT**
- **SHORT**
- **TYPE**

Assembler Directives (cont..)

- **PROC** - The **PROC** directive is used to identify the start of a procedure. The term near or far is used to specify the type of the procedure.
- Example:

SMART **PROC FAR** ; This identifies that the start of a procedure named as SMART and instructs the assembler that the procedure is far .

SMART **ENDP**

This PROC is used with ENDP to indicate the break of the procedure.

Assembler Directives (cont..)

- **PUBLIC** - The **PUBLIC** directive is used to instruct the assembler that a specified name or label will be accessed from other modules.
- Example:

PUBLIC **DIVISOR, DIVIDEND** ;these two variables are public so these are available to all modules.

If an instruction in a module refers to a variable in another assembly module, we can access that module by declaring as **EXTRN** directive.

DOS Function Calls

- **AH 00H** : Terminate a Program
- **AH 01H** : Read the Keyboard
- **AH 02H** : Write to a Standard Output Device
- **AH 08H** : Read a Standard Input without Echo
- **AH 09H** : Display a Character String
- **AH 0AH** : Buffered keyboard Input
- **INT 21H** : Call DOS Function

Assemble language programs for 8085 microprocessors

Arithmetic operations

Addition of two 8bit Numbers

```
START:NOP
LXI  H,0001H
 MOV A,M
 INX H
MOV B,M
ADD B
 INX H
MOV M,A
 HLT
```

a

Assemble language programs for 8085 microprocessors

Arithmetic operations

Substraction of two 8bit Numbers

```
START:NOP
LXI  H,0001H
 MOV A,M
 INX H
MOV B,M
SUB  B
 INX H
MOV M,A
 HLT
```

Assemble language programs for 8085 microprocessors

Arithmetic operations

Addition with carry of two 8bit Numbers

```
START:NOP
LXI  H,0001H
 MOV A,M
 INX H

MOV B,M

ADD B

ADC

 INX H

MOV M,A

 HLT
```

Assemble language programs for 8085 microprocessors

Arithmetic operations

Substractions with barrow of two 8bit Numbers

```
START:NOP
LXI  H,0001H
 MOV A,M
 INX H

MOV B,M

SUB  B

SBB

 INX H

MOV M,A

 HLT
```

Assemble language programs for 8085 microprocessors

Arithmetic operations

INCREMENT OPERATION

```
START:NOP
LXI  H,0001H
 MOV A,M
 INC A
 INX H
MOV M,A
 HLT
```

Assemble language programs for 8085 microprocessors

Arithmetic operations

DECREMENT OPERATION

```
START:NOP
LXI  H,0001H
 MOV A,M
 DEC A

 INX H

MOV M,A

 HLT
```

Assemble language programs for 8085 microprocessors

Logical operations

Anding of two 8bit Numbers

```
START:NOP
LXI  H,0001H
 MOV A,M
 INX H
MOV B,M
ANA  B
 INX H
MOV M,A
 HLT
```

Assemble language programs for 8085 microprocessors

Logical operations

ORing of two 8bit Numbers

```
START:NOP
LXI  H,0001H
 MOV A,M
 INX H

MOV B,M

ORA  B

 INX H

MOV M,A

 HLT
```

Assemble language programs for 8085 microprocessors

Logical operations

XORing of two 8bit Numbers

```
START:NOP
LXI  H,0001H
 MOV A,M
 INX H

MOV B,M

XRA  B

 INX H

MOV M,A

 HLT
```

Assemble language programs for 8085 microprocessors

Arithmetic operations

COMPLEMENT OPERATION

```
START:NOP
LXI  H,0001H
 MOV A,M
 CMP A

 INX H

MOV M,A

 HLT
```

Assemble language programs for 8085 microprocessors

Arithmetic operations

ROTATE LEFT OPERATION

```
START:NOP
LXI  H,0001H
 MOV A,M
 RLC A

 INX H

MOV M,A

 HLT
```

Assemble language programs for 8085 microprocessors

Arithmetic operations

ROTATE RIGHT OPERATION

```
START:NOP
LXI  H,0001H
 MOV A,M
 RRC A

 INX H

MOV M,A

 HLT
```

Assemble language programs for 8085 microprocessors

Arithmetic operations

ROTATE LEFT WITH CARRY OPERATION

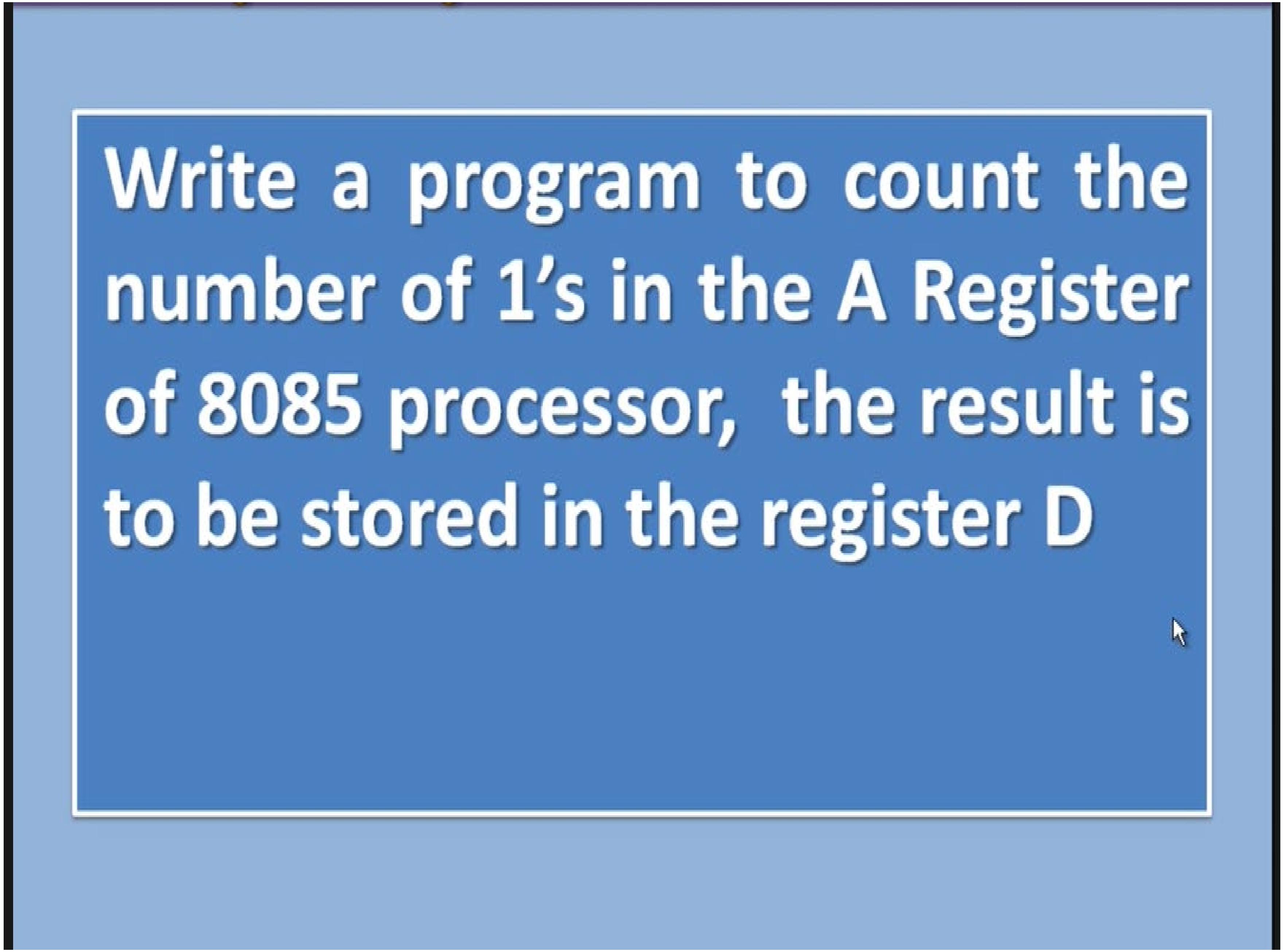
```
START:NOP
LXI  H,0001H
 MOV A,M
 RAL A

 INX H

MOV M,A

 HLT
```

Assemble language programs for 8085 microprocessors

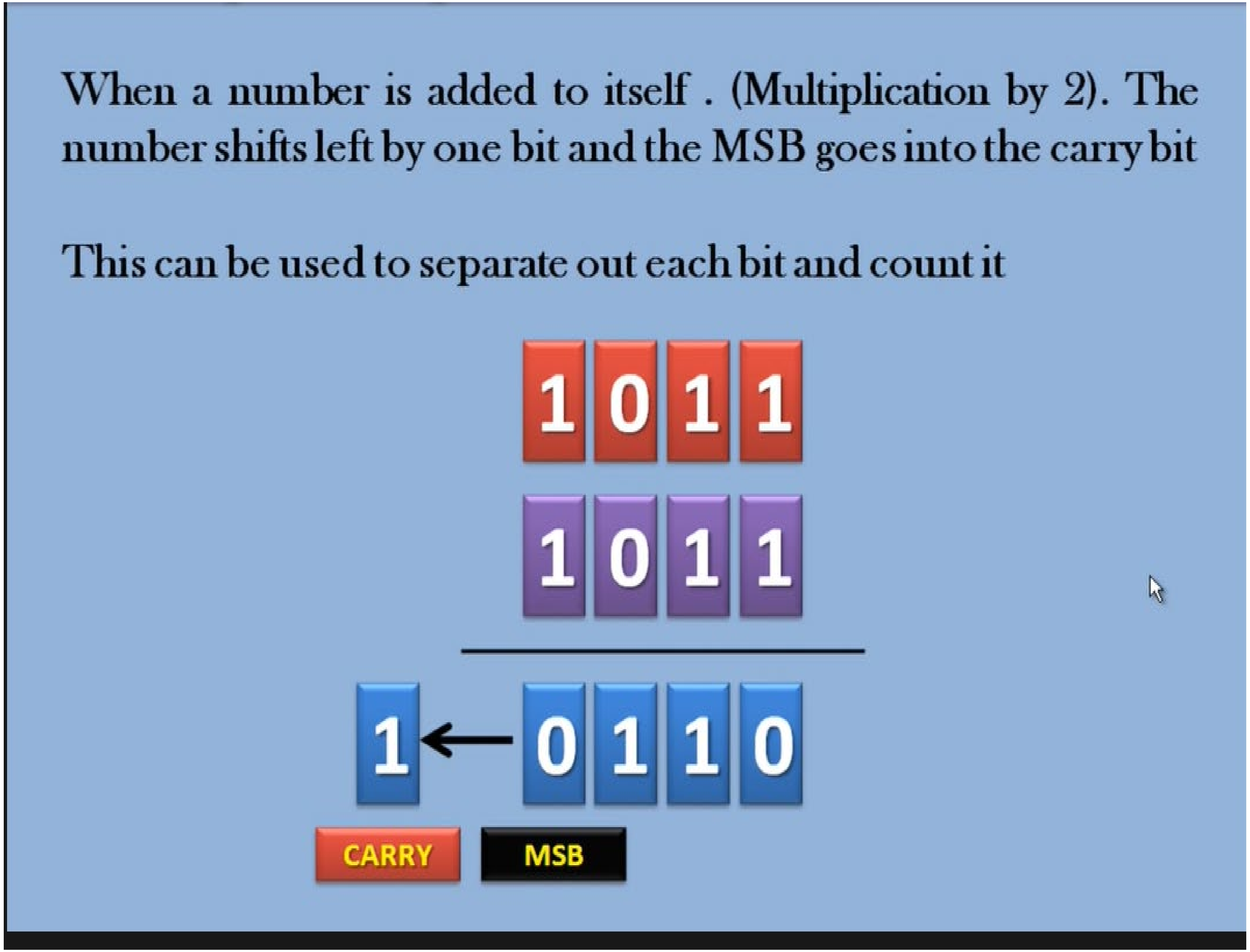
Arithmetic operations

ROTATE RIGHT WITH CARRY OPERATION

```
START:NOP
LXI  H,0001H
 MOV A,M
 RAR A

 INX H

MOV M,A

 HLT
```



Write a program to count the number of 1’s in the A Register of 8085 processor, the result is to be stored in the register D



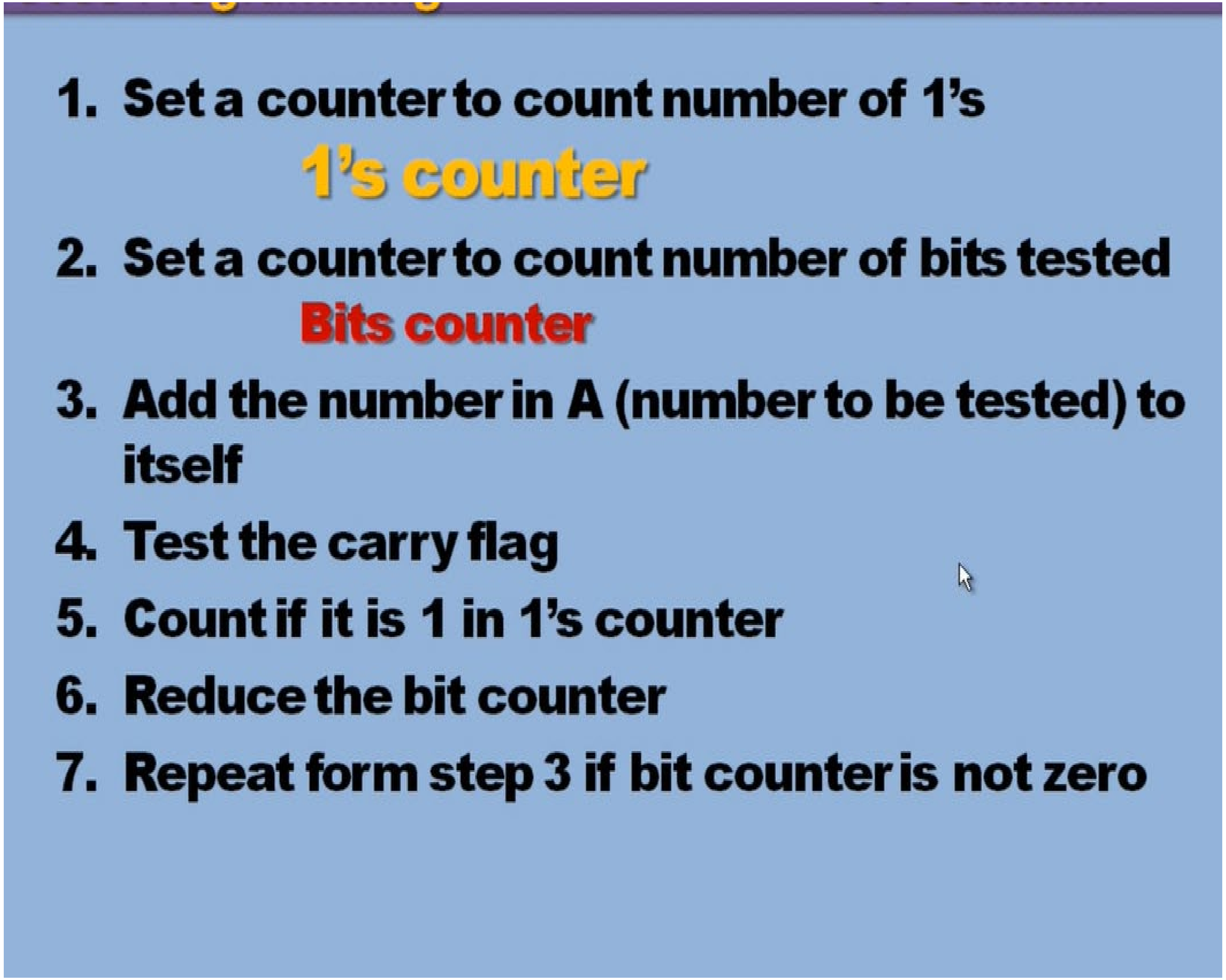
When a number is added to itself . (Multiplication by 2). The number shifts left by one bit and the MSB goes into the carry bit

This can be used to separate out each bit and count it

1 0 1 1

1 0 1 1

1 ← 0 1 1 0



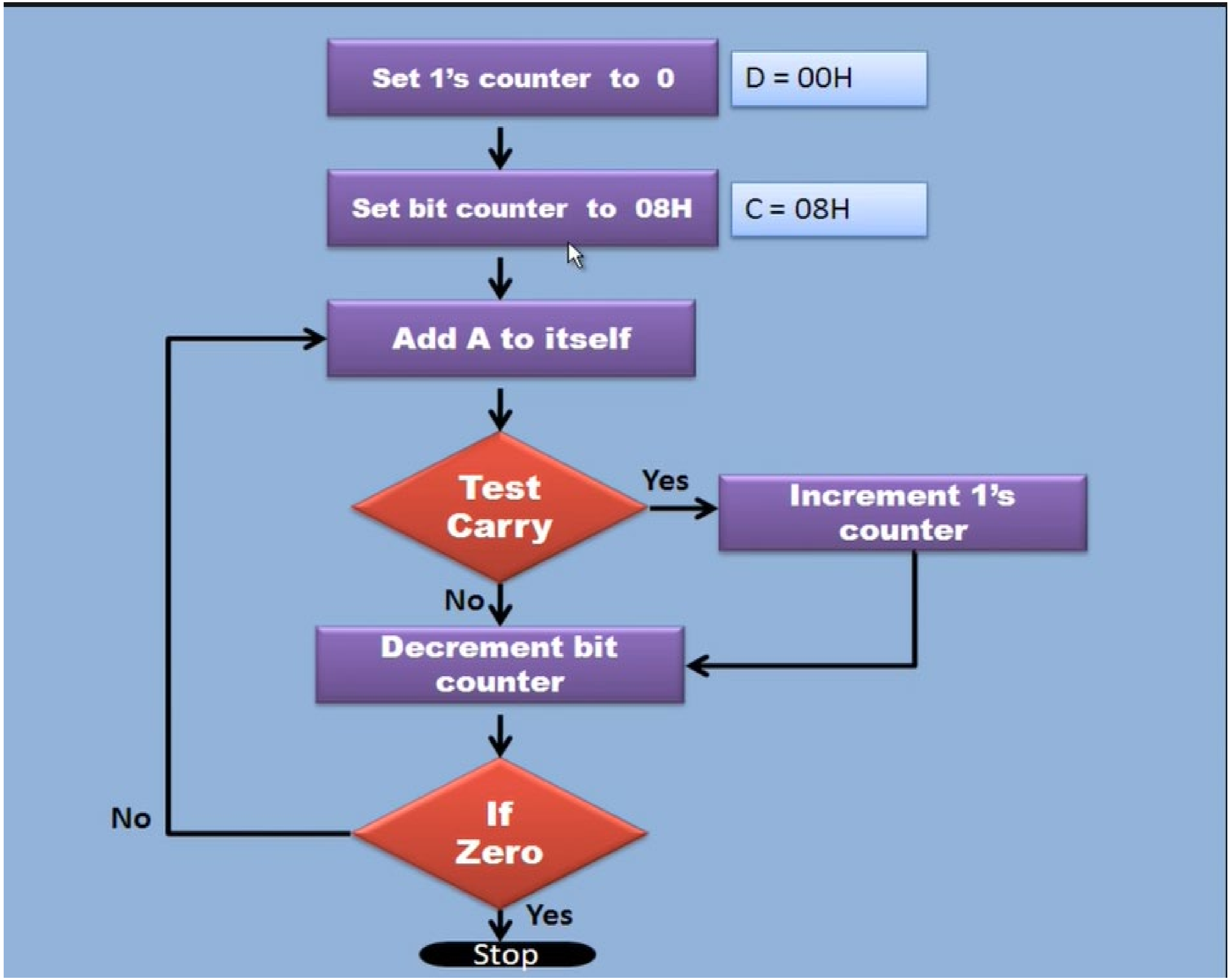
1. Set a counter to count number of 1's

   1's counter

2. Set a counter to count number of bits tested

   Bits counter

3. Add the number in A (number to be tested) to itself
4. Test the carry flag
5. Count if it is 1 in 1's counter
6. Reduce the bit counter
7. Repeat form step 3 if bit counter is not zero



Set 1's counter to 0

D = 00H

Set bit counter to 08H

C = 08H

Add A to itself

Test Carry

Yes

Increment 1's counter

No

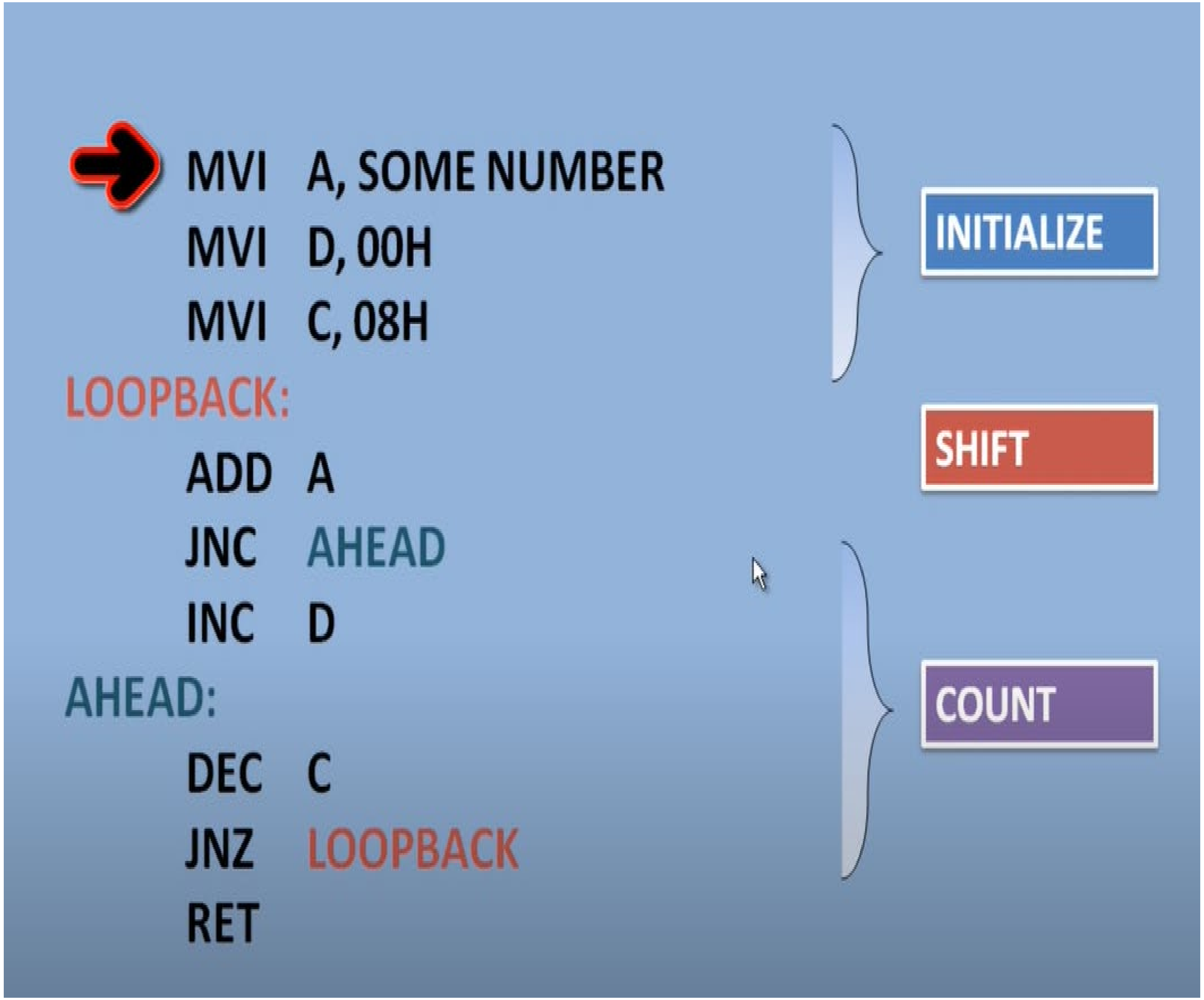
Decrement bit counter

If Zero

No

Yes

Stop

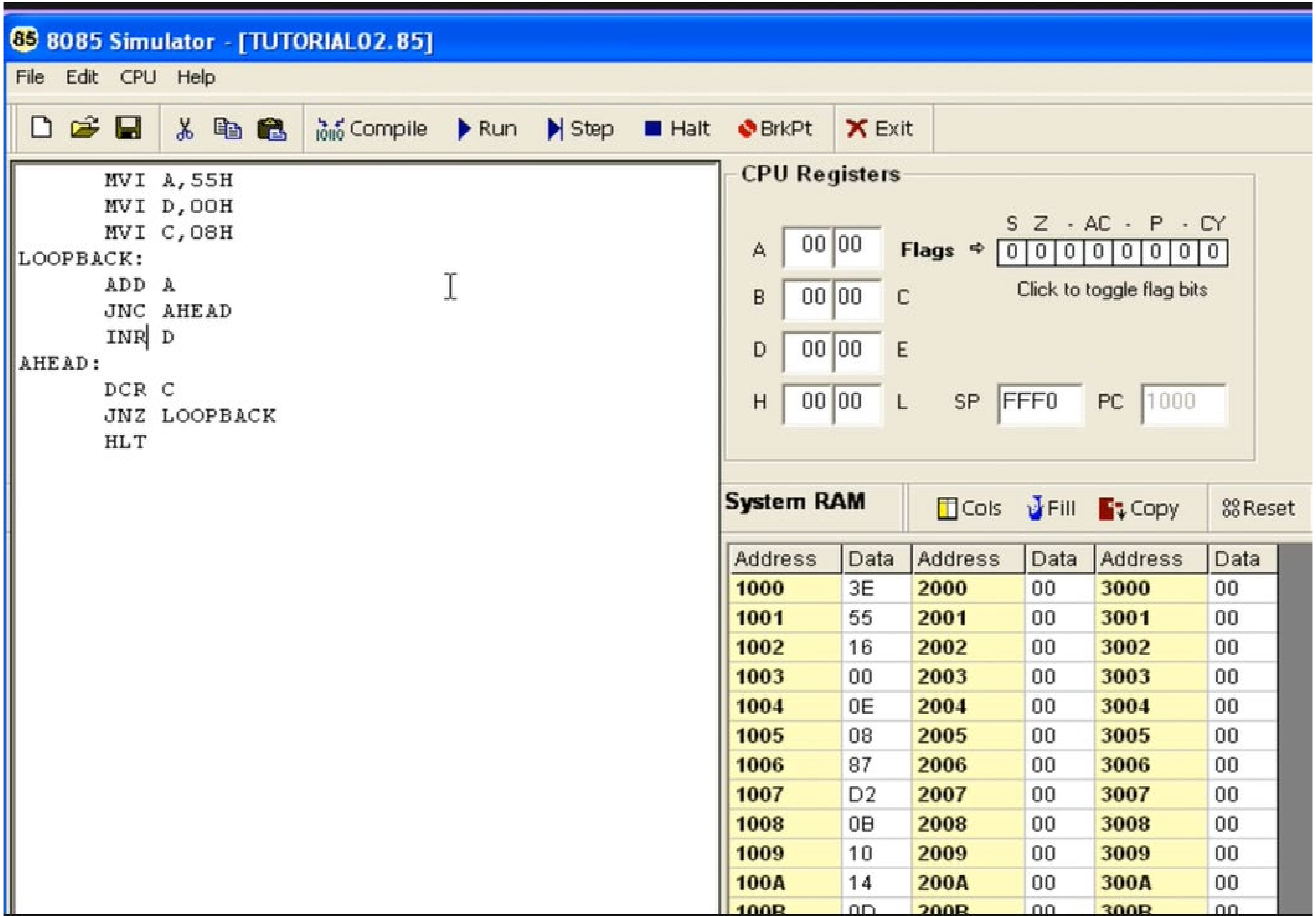


```
         MVI   A, SOME NUMBER
         MVI   D, 00H
         MVI   C, 08H
LOOPBACK:
         ADD   A
         JNC   AHEAD
         INC   D
AHEAD:
         DEC   C
         JNZ   LOOPBACK
         RET
```

INITIALIZE

SHIFT

COUNT



```
        MVI A,55H
        MVI D,00H
        MVI C,08H
LOOPBACK:
        ADD A
        JNC AHEAD
        INR D
AHEAD:
        DCR C
        JNZ LOOPBACK
        HLT
```

A 00 00

B 00 00 C

D 00 00 E

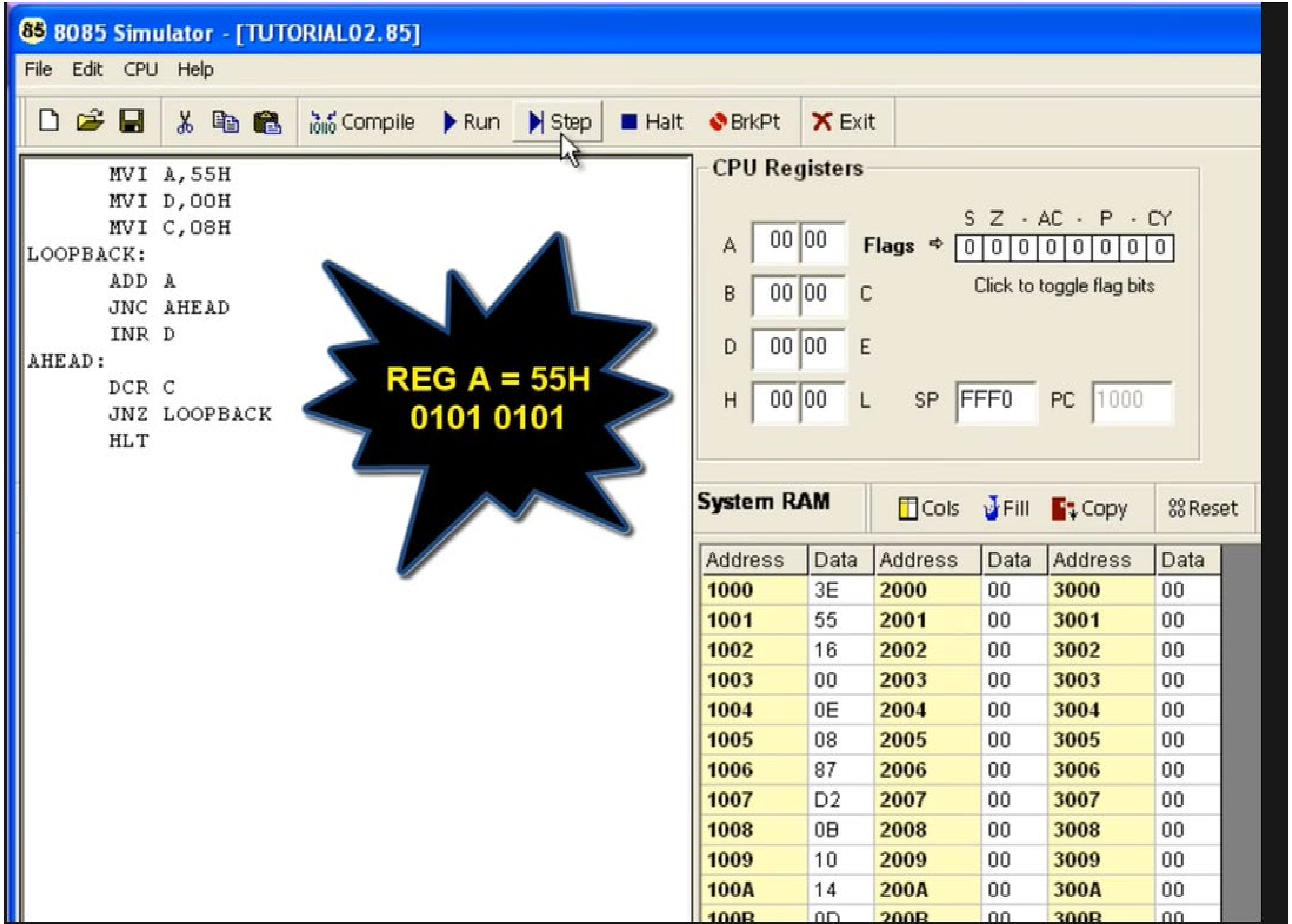
H 00 00 L

Flags S Z - AC - P - CY: 0 0 0 0 0 0 0 0

SP FFF0 PC 1000

Address	Data	Address	Data	Address	Data
1000	3E	2000	00	3000	00
1001	55	2001	00	3001	00
1002	16	2002	00	3002	00
1003	00	2003	00	3003	00
1004	0E	2004	00	3004	00
1005	08	2005	00	3005	00
1006	87	2006	00	3006	00
1007	D2	2007	00	3007	00
1008	0B	2008	00	3008	00
1009	10	2009	00	3009	00
100A	14	200A	00	300A	00
100B	0D	200B	00	300B	00

8085 PROGRAMMING COUNTING AND LOOPING



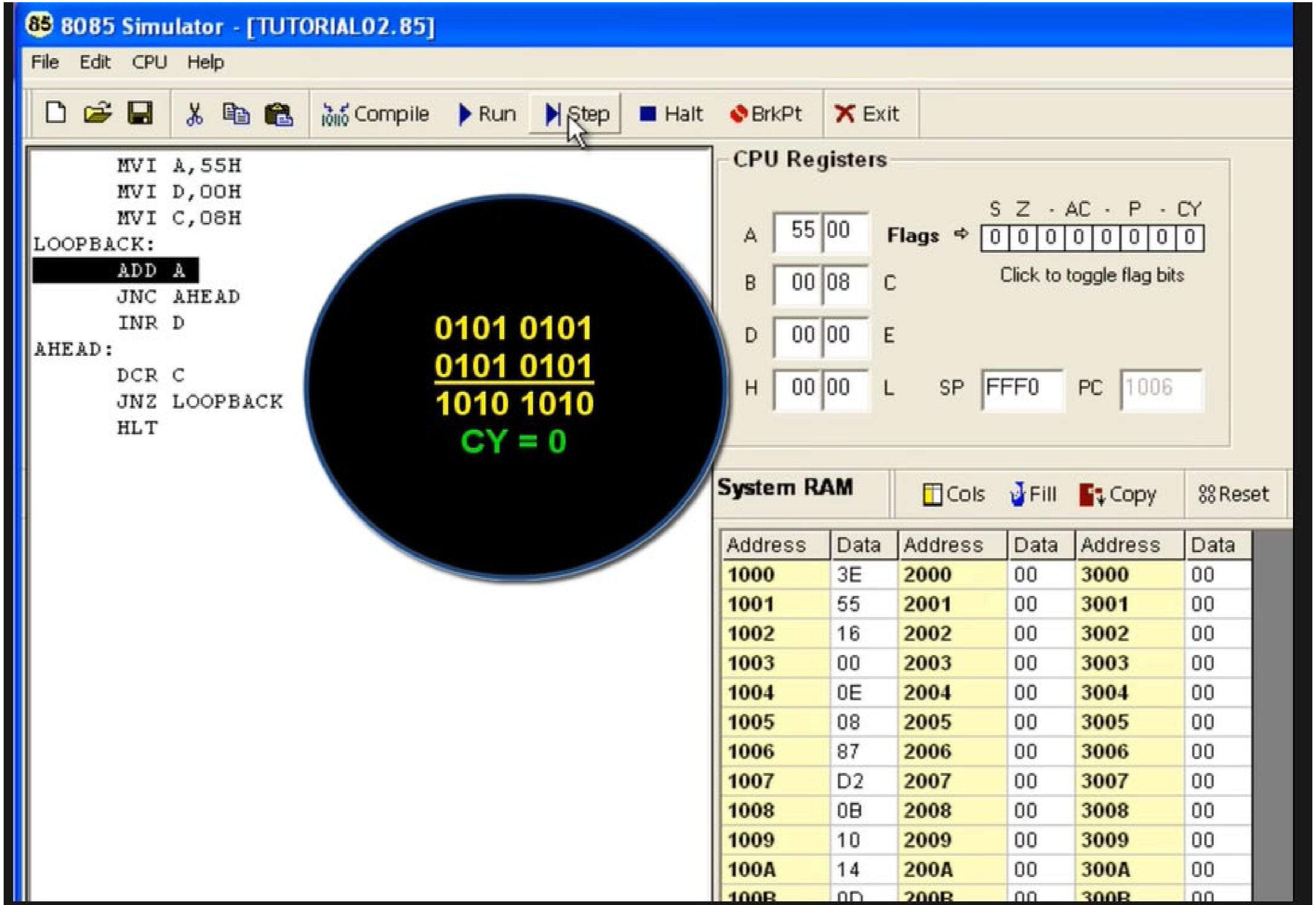
```
        MVI A,55H
        MVI D,00H
        MVI C,08H
LOOPBACK:
        ADD A
        JNC AHEAD
        INR D
AHEAD:
        DCR C
        JNZ LOOPBACK
        HLT
```

REG A = 55H
0101 0101

				S	Z	-	AC	-	P	-	CY
A	00	00	Flags	0	0	0	0	0	0	0	0
B	00	00	C								
D	00	00	E								
H	00	00	L	SP	FFF0	PC	1000				

Address	Data	Address	Data	Address	Data
1000	3E	2000	00	3000	00
1001	55	2001	00	3001	00
1002	16	2002	00	3002	00
1003	00	2003	00	3003	00
1004	0E	2004	00	3004	00
1005	08	2005	00	3005	00
1006	87	2006	00	3006	00
1007	D2	2007	00	3007	00
1008	0B	2008	00	3008	00
1009	10	2009	00	3009	00
100A	14	200A	00	300A	00
100B	0D	200B	00	300B	00

8085 PROGRAMMING COUNTING AND LOOPING



```
      MVI A,55H
      MVI D,00H
      MVI C,08H
LOOPBACK:
      ADD A
      JNC AHEAD
      INR D
AHEAD:
      DCR C
      JNZ LOOPBACK
      HLT
```

0101 0101
0101 0101
1010 1010
CY = 0

A 55 00
B 00 08 C
D 00 00 E
H 00 00 L

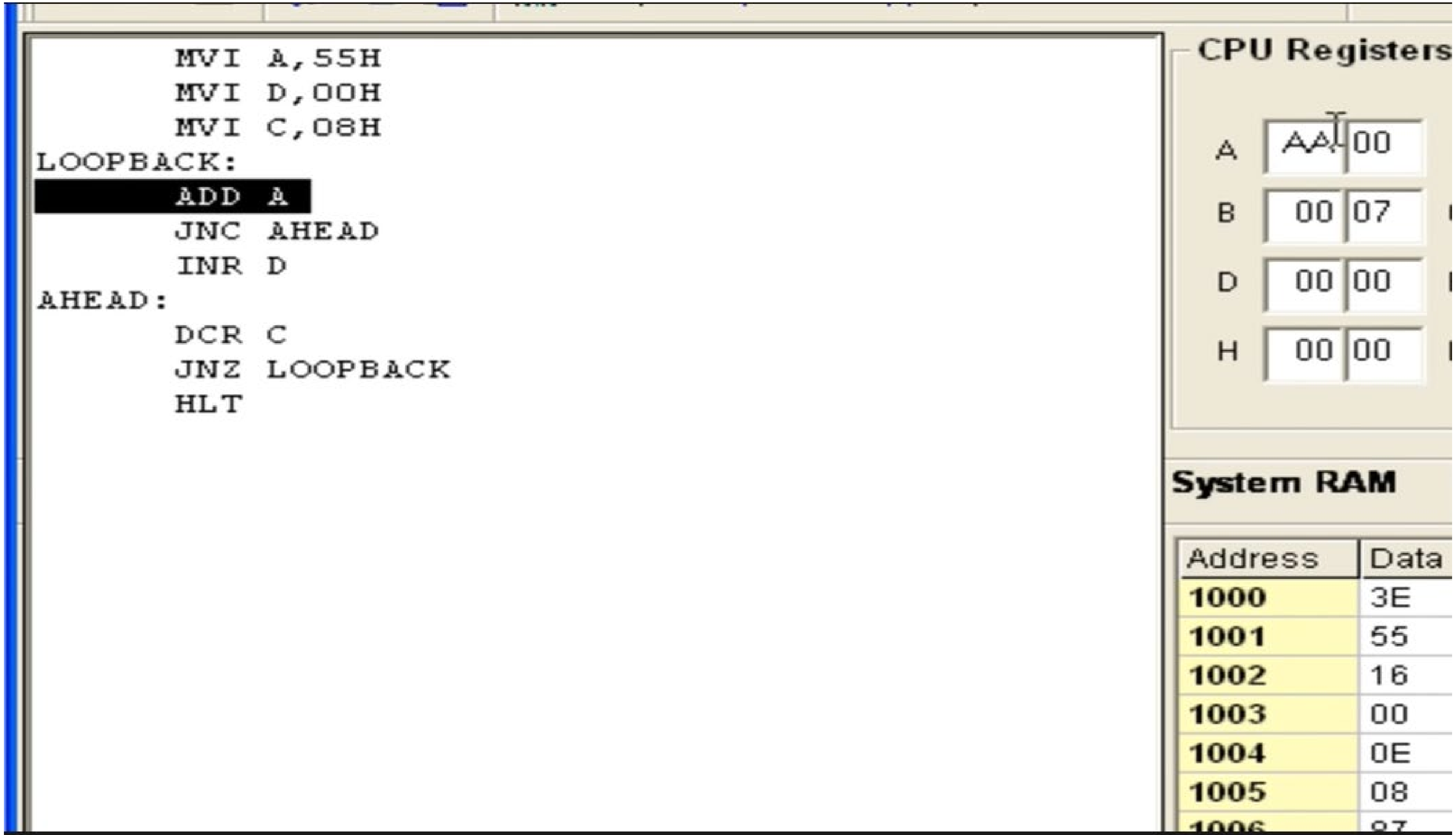
Flags

S	Z	-	AC	-	P	-	CY
0	0	0	0	0	0	0	0

SP FFF0 PC 1006

Address	Data	Address	Data	Address	Data
1000	3E	2000	00	3000	00
1001	55	2001	00	3001	00
1002	16	2002	00	3002	00
1003	00	2003	00	3003	00
1004	0E	2004	00	3004	00
1005	08	2005	00	3005	00
1006	87	2006	00	3006	00
1007	D2	2007	00	3007	00
1008	0B	2008	00	3008	00
1009	10	2009	00	3009	00
100A	14	200A	00	300A	00
100B	0D	200B	00	300B	00

8085 PROGRAMMING COUNTING AND LOOPING



```
        MVI A,55H
        MVI D,00H
        MVI C,08H
LOOPBACK:
        ADD A
        JNC AHEAD
        INR D
AHEAD:
        DCR C
        JNZ LOOPBACK
        HLT
```

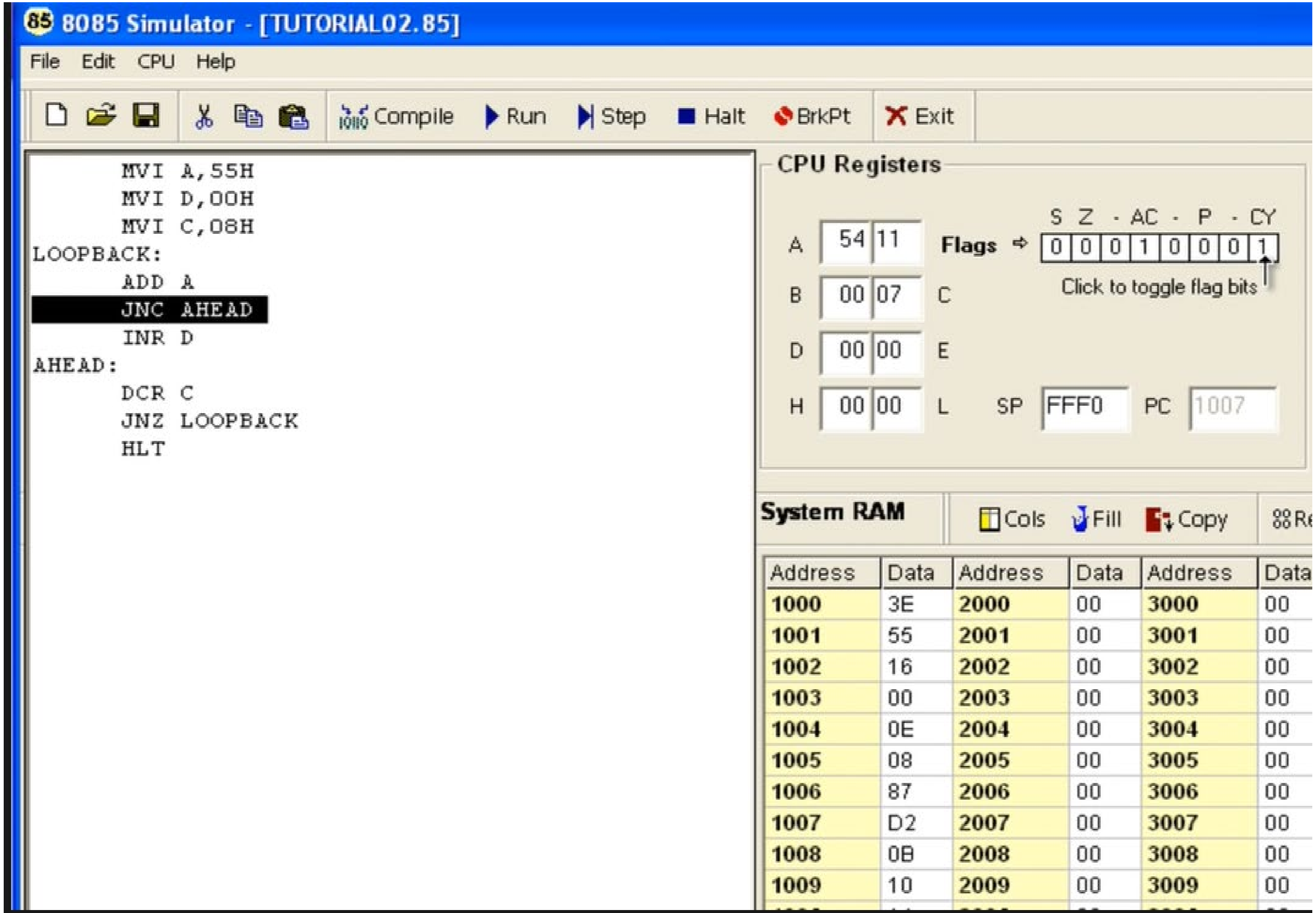
CPU Registers

```
A  AA 00
B  00 07
D  00 00
H  00 00
```

System RAM

Address	Data
1000	3E
1001	55
1002	16
1003	00
1004	0E
1005	08
1006	87

8085 PROGRAMMING COUNTING AND LOOPING



```
        MVI A,55H
        MVI D,00H
        MVI C,08H
LOOPBACK:
        ADD A
        JNC AHEAD
        INR D
AHEAD:
        DCR C
        JNZ LOOPBACK
        HLT
```

A 54 | 11 Flags

S	Z	·	AC	·	P	·	CY
0	0	0	1	0	0	0	1

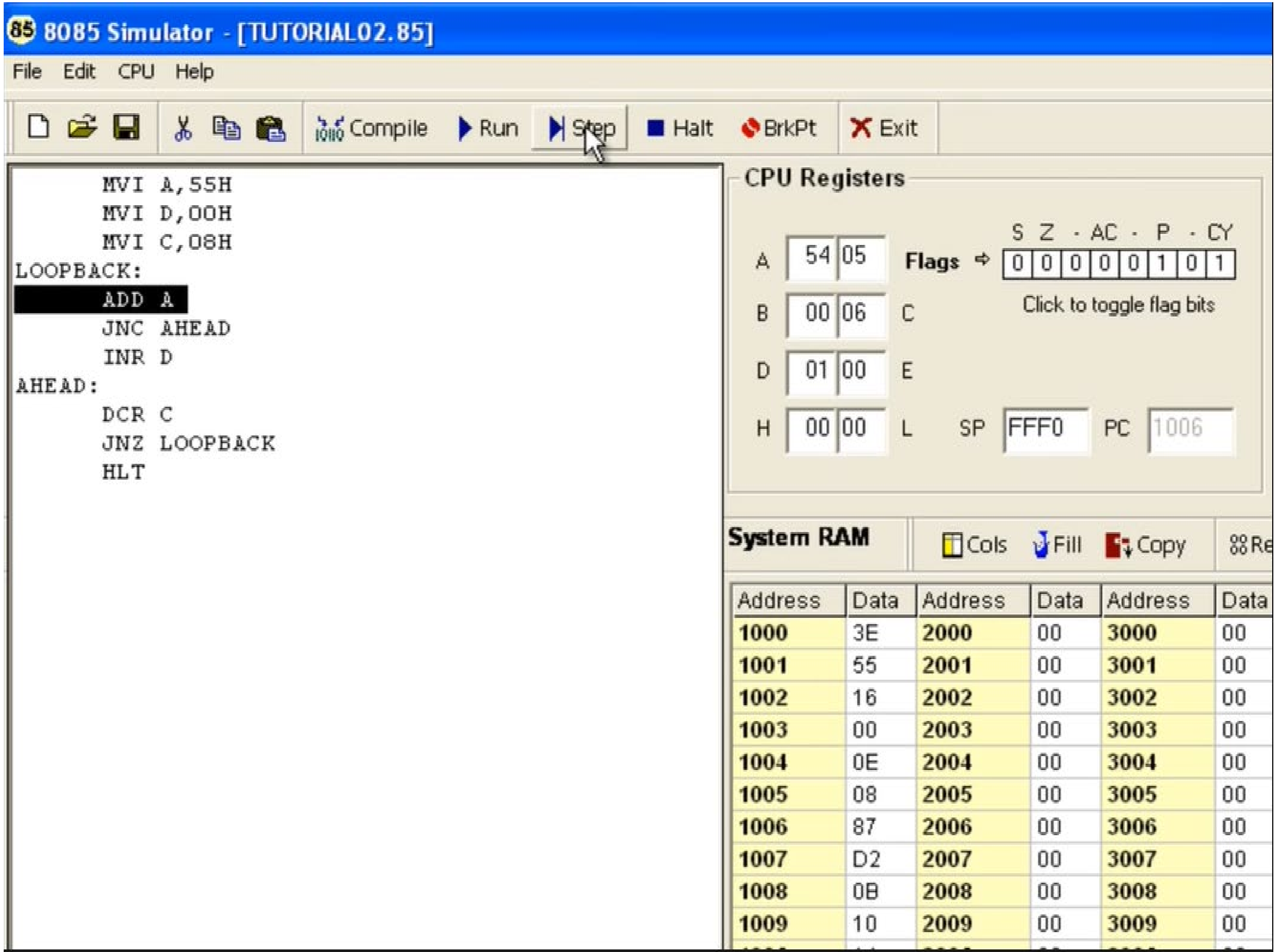
B 00 | 07 C

D 00 | 00 E

H 00 | 00 L SP FFF0 PC 1007

Address	Data	Address	Data	Address	Data
1000	3E	2000	00	3000	00
1001	55	2001	00	3001	00
1002	16	2002	00	3002	00
1003	00	2003	00	3003	00
1004	0E	2004	00	3004	00
1005	08	2005	00	3005	00
1006	87	2006	00	3006	00
1007	D2	2007	00	3007	00
1008	0B	2008	00	3008	00
1009	10	2009	00	3009	00

8085 PROGRAMMING COUNTING AND LOOPING



```
        MVI A,55H
        MVI D,00H
        MVI C,08H
LOOPBACK:
        ADD A
        JNC AHEAD
        INR D
AHEAD:
        DCR C
        JNZ LOOPBACK
        HLT
```

A 54 05 Flags

S	Z	-	AC	-	P	-	CY
0	0	0	0	0	1	0	1

Click to toggle flag bits

B 00 06 C

D 01 00 E

H 00 00 L SP FFF0 PC 1006

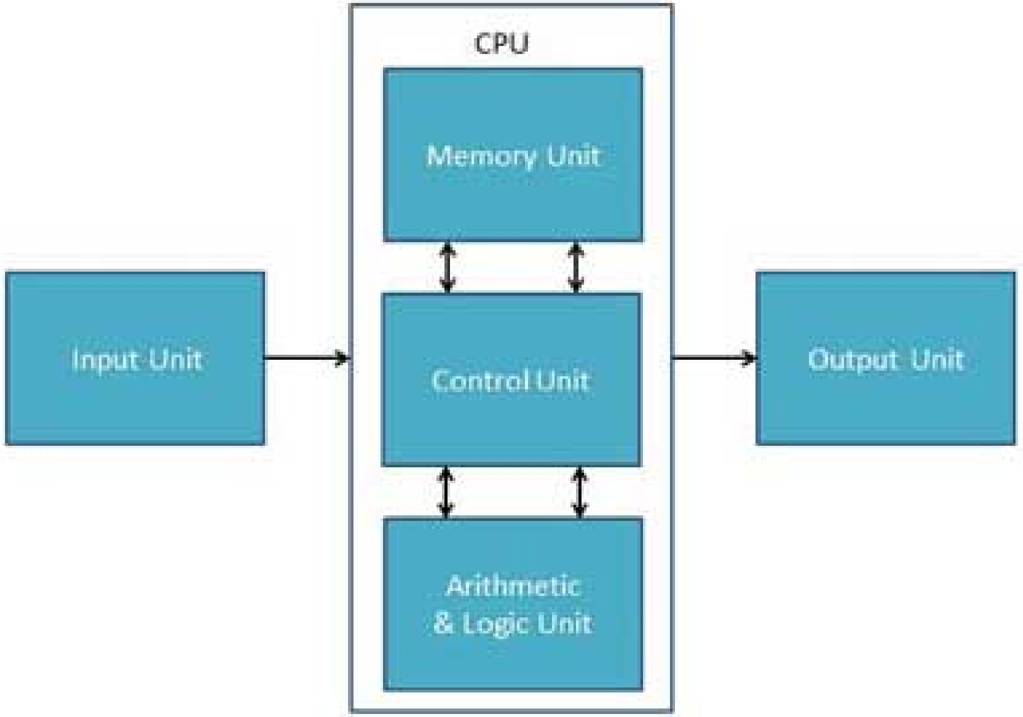
Address	Data	Address	Data	Address	Data
1000	3E	2000	00	3000	00
1001	55	2001	00	3001	00
1002	16	2002	00	3002	00
1003	00	2003	00	3003	00
1004	0E	2004	00	3004	00
1005	08	2005	00	3005	00
1006	87	2006	00	3006	00
1007	D2	2007	00	3007	00
1008	0B	2008	00	3008	00
1009	10	2009	00	3009	00

8085 PROGRAMMING COUNTING AND LOOPING

MODULE –III

8086 ARCHITECTURE

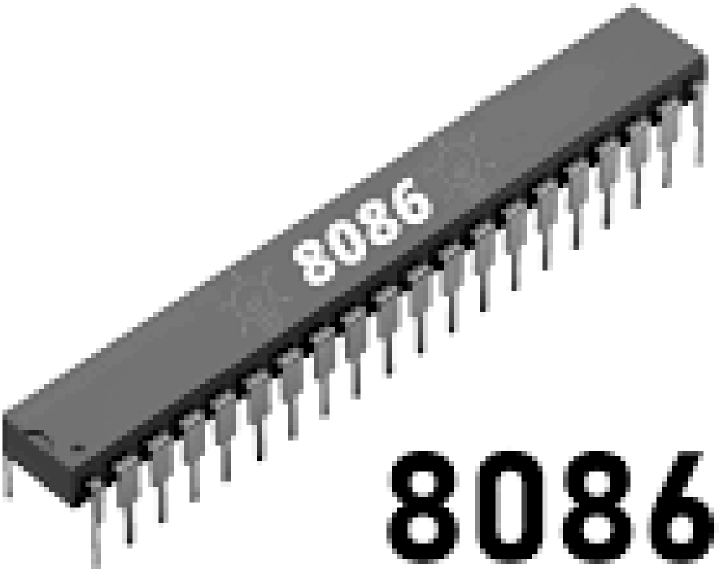
INTRODUCTION



COMPUTER

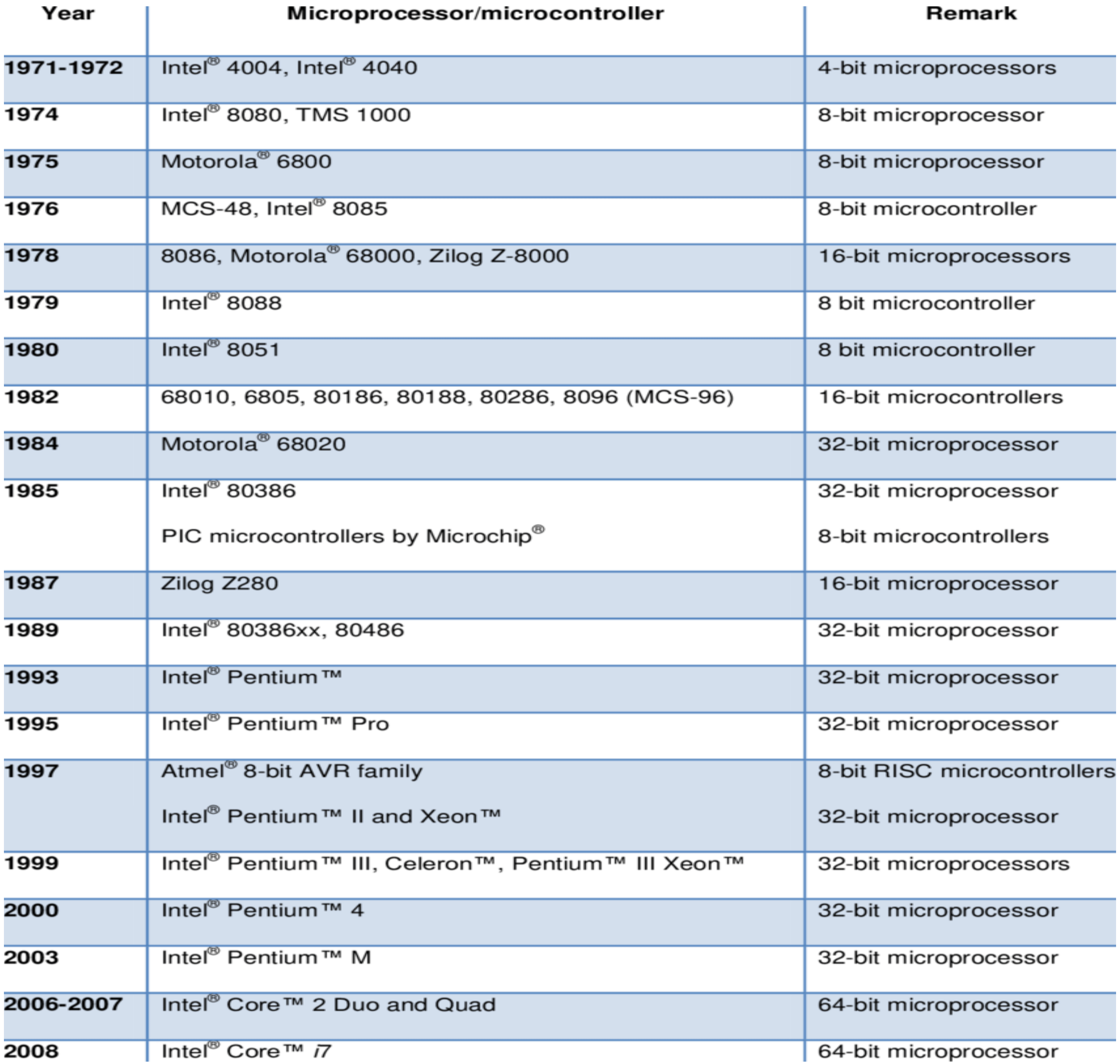
Microprocessor

- A **microprocessor** is an electronic component that is used by a computer to do its work. It is a central processing unit on a single integrated circuit chip containing millions of very small components including transistors, resistors, ge



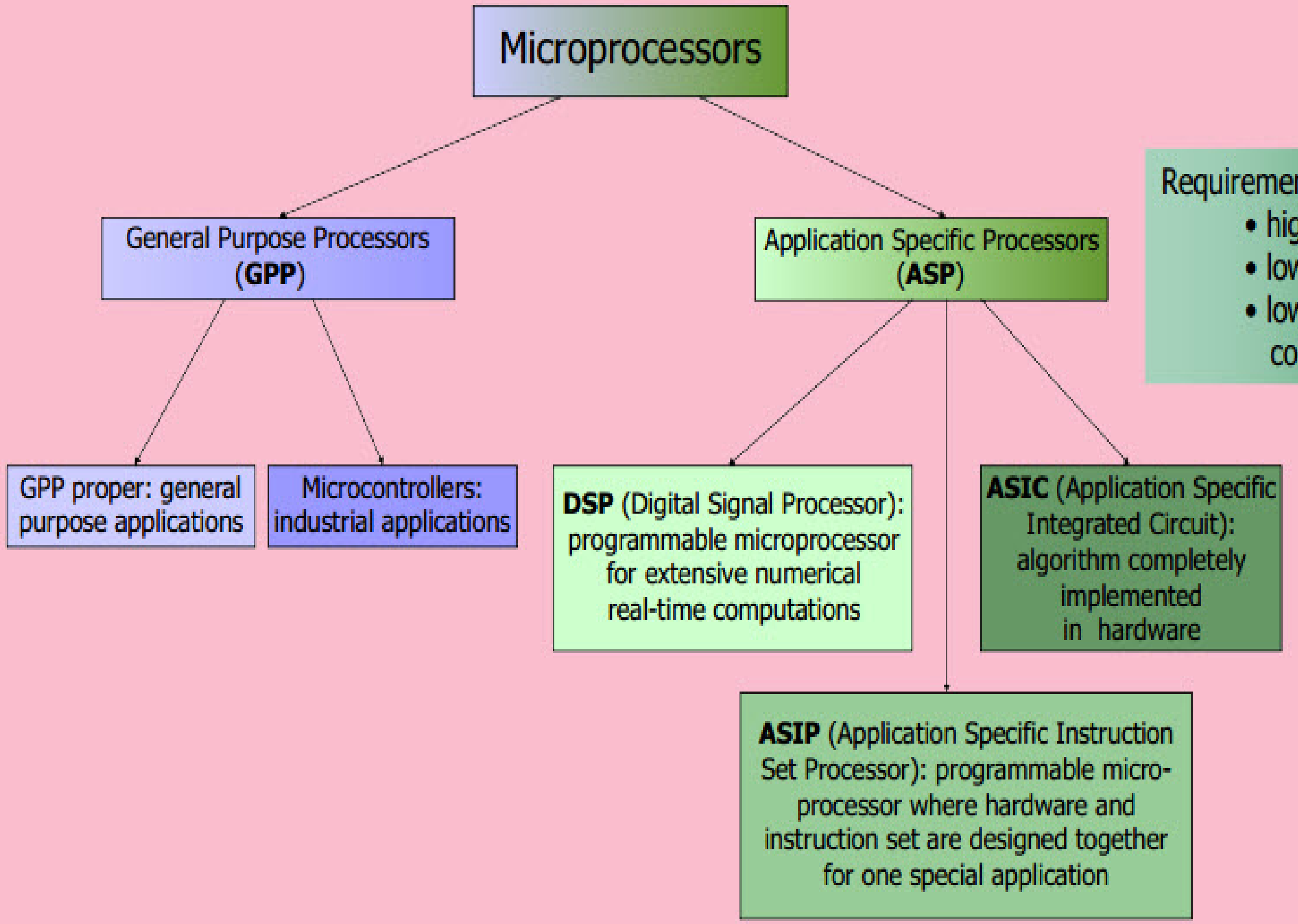
Motherboard

Types of microprocessors



Year	Microprocessor/microcontroller	Remark
1971-1972	Intel® 4004, Intel® 4040	4-bit microprocessors
1974	Intel® 8080, TMS 1000	8-bit microprocessor
1975	Motorola® 6800	8-bit microprocessor
1976	MCS-48, Intel® 8085	8-bit microcontroller
1978	8086, Motorola® 68000, Zilog Z-8000	16-bit microprocessors
1979	Intel® 8088	8 bit microcontroller
1980	Intel® 8051	8 bit microcontroller
1982	68010, 6805, 80186, 80188, 80286, 8096 (MCS-96)	16-bit microcontrollers
1984	Motorola® 68020	32-bit microprocessor
1985	Intel® 80386 PIC microcontrollers by Microchip®	32-bit microprocessor 8-bit microcontrollers
1987	Zilog Z280	16-bit microprocessor
1989	Intel® 80386xx, 80486	32-bit microprocessor
1993	Intel® Pentium™	32-bit microprocessor
1995	Intel® Pentium™ Pro	32-bit microprocessor
1997	Atmel® 8-bit AVR family Intel® Pentium™ II and Xeon™	8-bit RISC microcontrollers 32-bit microprocessor
1999	Intel® Pentium™ III, Celeron™, Pentium™ III Xeon™	32-bit microprocessors
2000	Intel® Pentium™ 4	32-bit microprocessor
2003	Intel® Pentium™ M	32-bit microprocessor
2006-2007	Intel® Core™ 2 Duo and Quad	64-bit microprocessor
2008	Intel® Core™ *i7*	64-bit microprocessor

Classification of Microprocessors



Microprocessors

General Purpose Processors (**GPP**)

Application Specific Processors (**ASP**)

GPP proper: general purpose applications

Microcontrollers: industrial applications

DSP (Digital Signal Processor): programmable microprocessor for extensive numerical real-time computations

ASIC (Application Specific Integrated Circuit): algorithm completely implemented in hardware

ASIP (Application Specific Instruction Set Processor): programmable micro-processor where hardware and instruction set are designed together for one special application

Requirements:

- high performance
- low cost
- low power consumption

Application of Microprocessors:

- Microcomputers
- Personal Computers
- Printing Machines
- Security Systems
- Play Stations
- Digital Projectors
- Washing Machines
- Microwave Ovens

Intel 8086

Architecture & Programming

Features of 8086 Microprocessor

- 1**) 8086 has 16-bit ALU; this means 16-bit numbers are directly processed by 8086.**

- **2) It has 16-bit data bus, so it can read data or write data to memory or I/O ports either 16 bits or 8 bits at a time.**

- **3) It has 20 address lines, so it can address up to 2^{20} i.e. 1048576 = 1Mbytes of memory (words i.e. 16 bit numbers are stored in consecutive memory locations). Due to the 1Mbytes memory size multiprogramming is made feasible as well as several multiprogramming features have been incorporated in 8086 design.**

Features Continued …

- **4) 8086 includes few features, which enhance multiprocessing capability (it can be used with math coprocessors like 8087, I/O processor 8089 etc.**

- **5) Operates on +5v supply and single phase (single line) clock frequency.(Clock is generated by separate peripheral chip 8284).**

- **6) 8086 comes with different versions. 8086 runs at 5 MHz, 8086-2 runs at 8 MHz, 8086-1 runs at 10 MHz.**

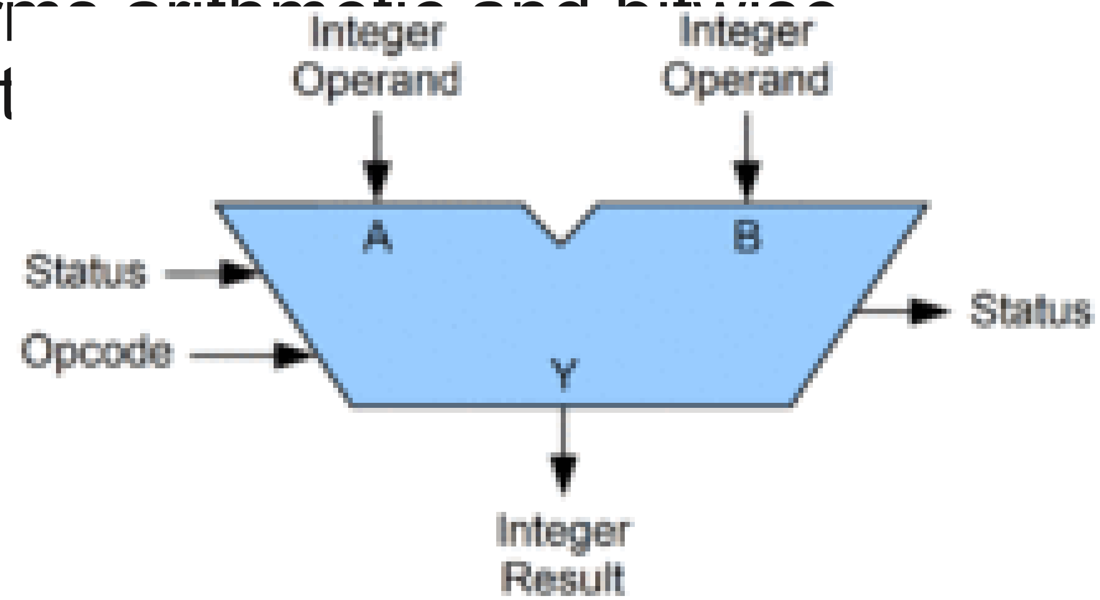
- **7) It comes in 40-pin configuration with HMOS technology having around 20,000 transistors in its circuitry.**

Features Continued …

- **8) It has multiplexed address and data bus like 8085 due to which the pin count is reduced considerably**
- **9) Higher Throughput (Speed)(This is achieved by a concept called pipelining)But the concept of 8086's principles and structures is very useful for understanding other advanced Intel microprocessors.**
- **10) it will work in the 2 modes , 1st is minimum mode and 2nd is maximum mode.**
- **Minimum mode : If only 1 processor is used in any system to perform the operation that is called as minimum mode operation.**
- **Maximum mode : If more than 1 processor is used in any system to perform the operations that is called as maximum mode operations.**

Main components present in the microprocessor

1. Arithmetic logic unit (ALU) : An **arithmetic logic unit** (**ALU**) is a combinational digital electronic circuit that performs arithmetic and bitwise operat



2. Registers : which is used to store the data and addresses of program.

It is a group of the flip flops.

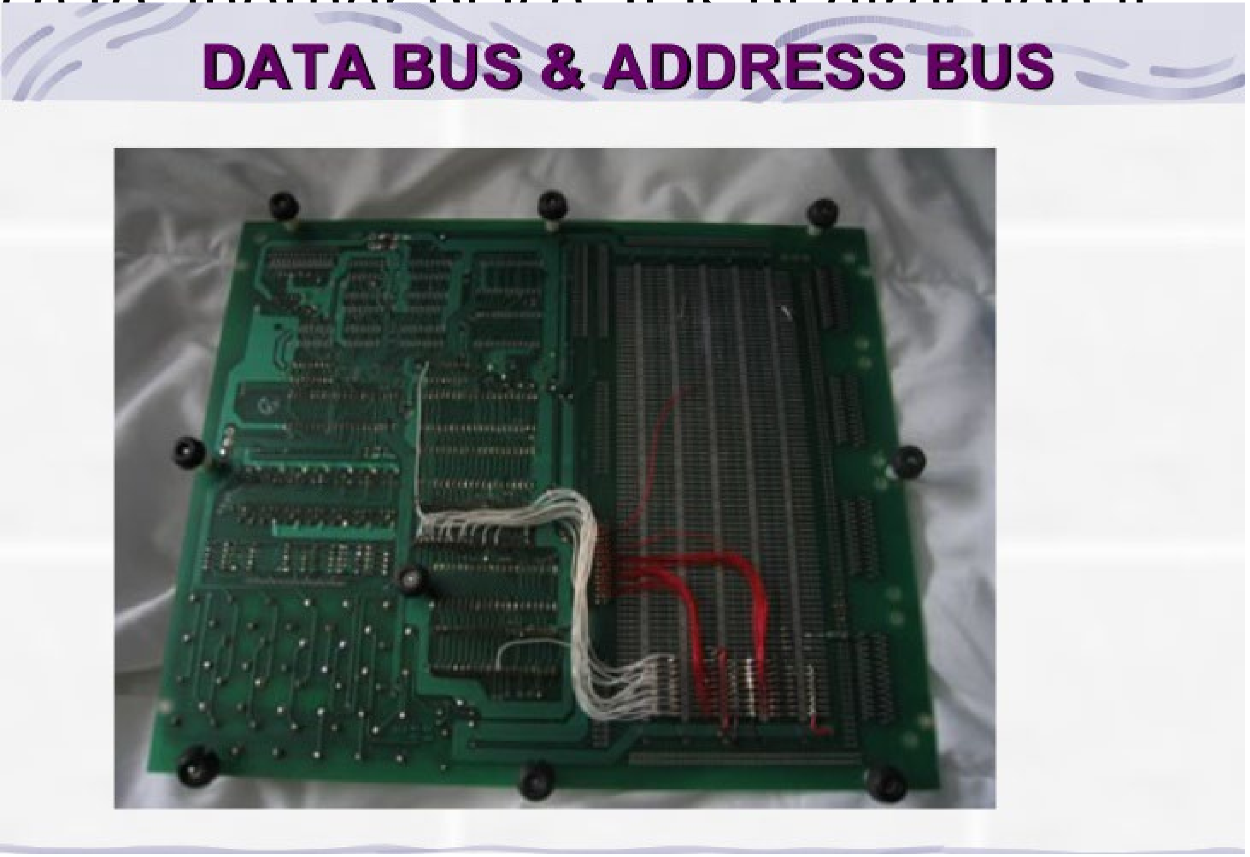
REGISTERS
analoque data storage cells

3. **Buses** : bus is the group of the parallel conducting wires which carries the data from one place to another.

These are 2 types :

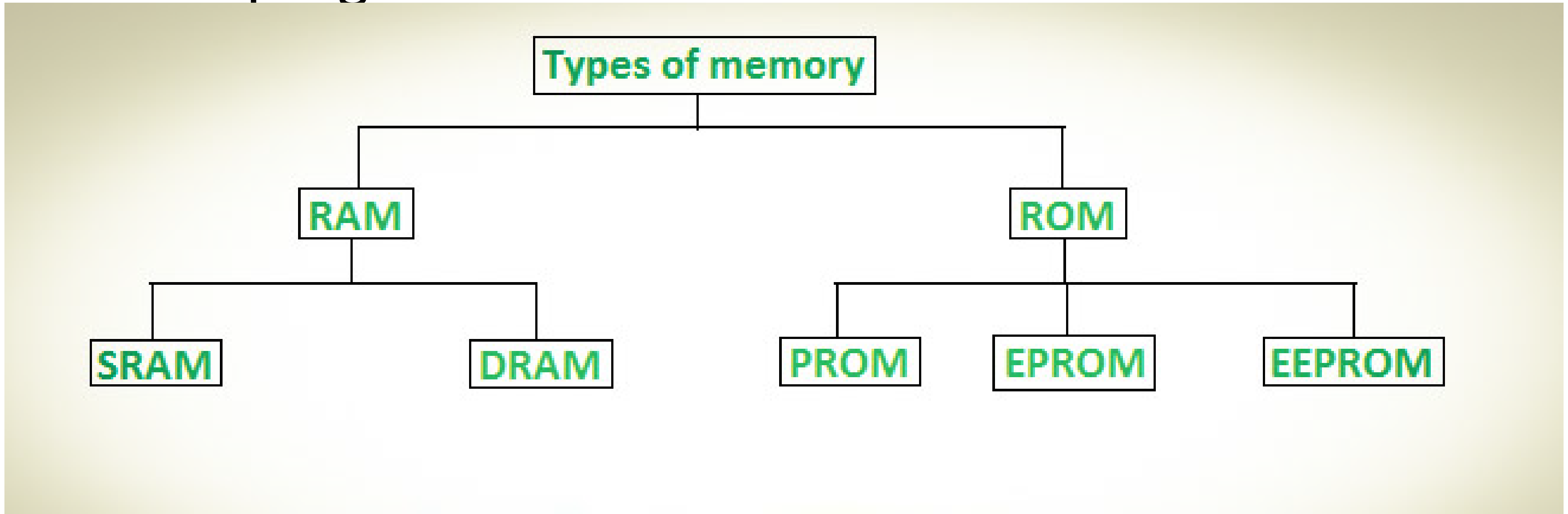
1) Data bus : which carry the data of program one place to another place. It is bi-directional

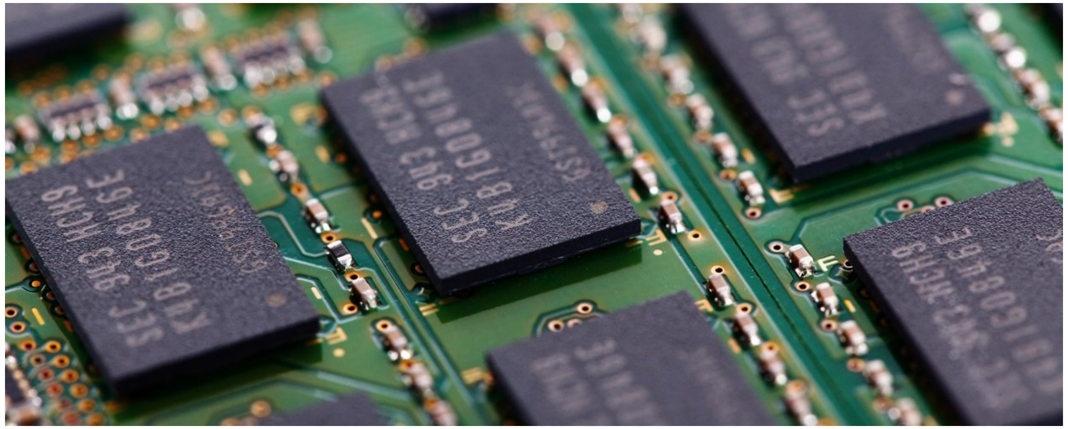
2) Ad
pro



DATA BUS & ADDRESS BUS

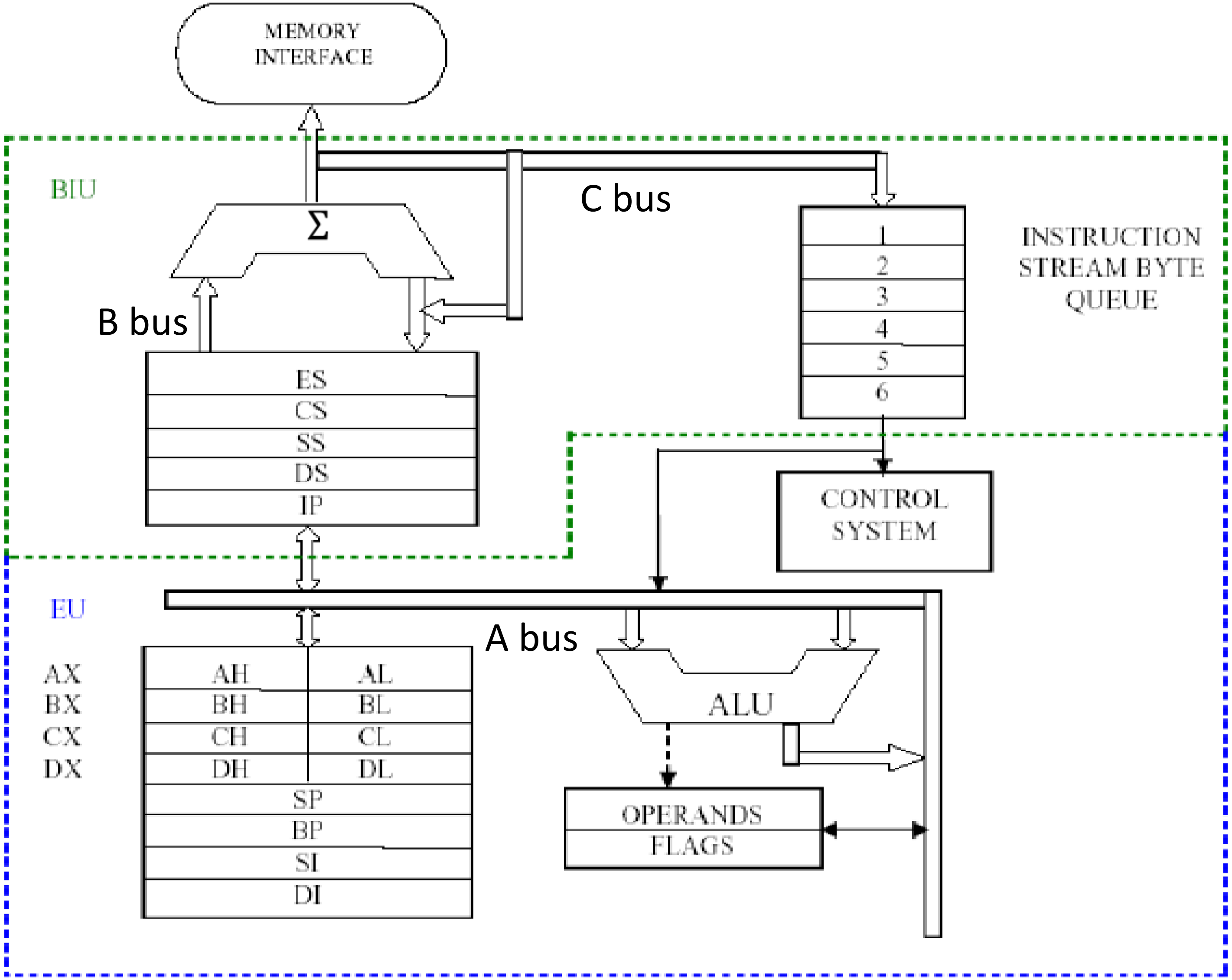
- 4. **Memory** : This is storage device which is used to store program code and data.





5. **Instruction queue** : it fetch the instructions from the memory and place the instructions in the serial order to perform the operation like FIFO (first In first out).

Internal Architecture of 8086



- The 8086 microprocessor architecture divided into 2 functional units:

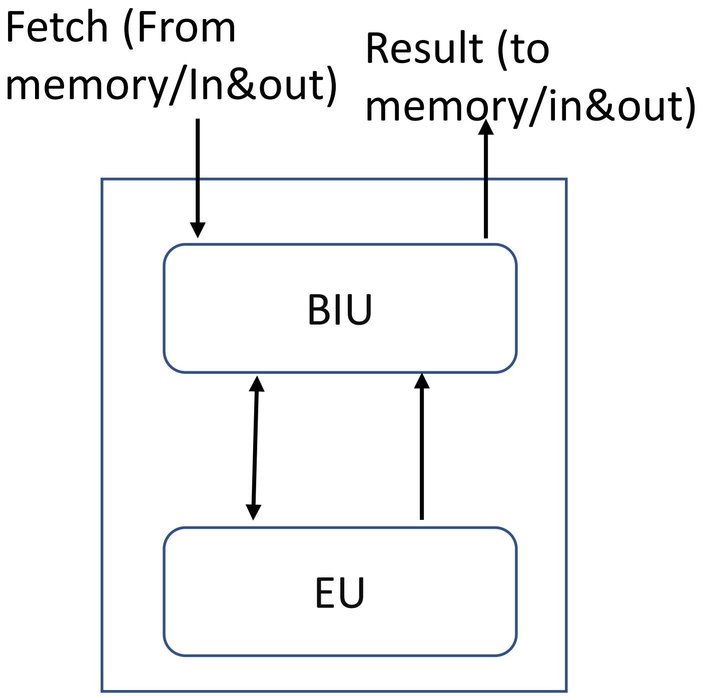
1. Bus interface unit
2. Execution unit

- **BUS INTERFACE UNIT**: it acts as bridge between external devices like memory and in /out devices to execution unit. It provides a full 16 bit bidirectional data bus and 20 bit address bus.

- The bus interface unit is responsible for performing all external bus operations.
- Instructions fetch Instruction queuing, Operand fetch and storage, Address relocation and Bus control.

- It fetch the instructions or data from memory.
- It writes the data to memory.
- It writes the data to ports.
- It reads the data from ports.
- It is also having 3 functional parts:

1. Instruction pointer (IP)
2. Segment registers
3. Instruction queue



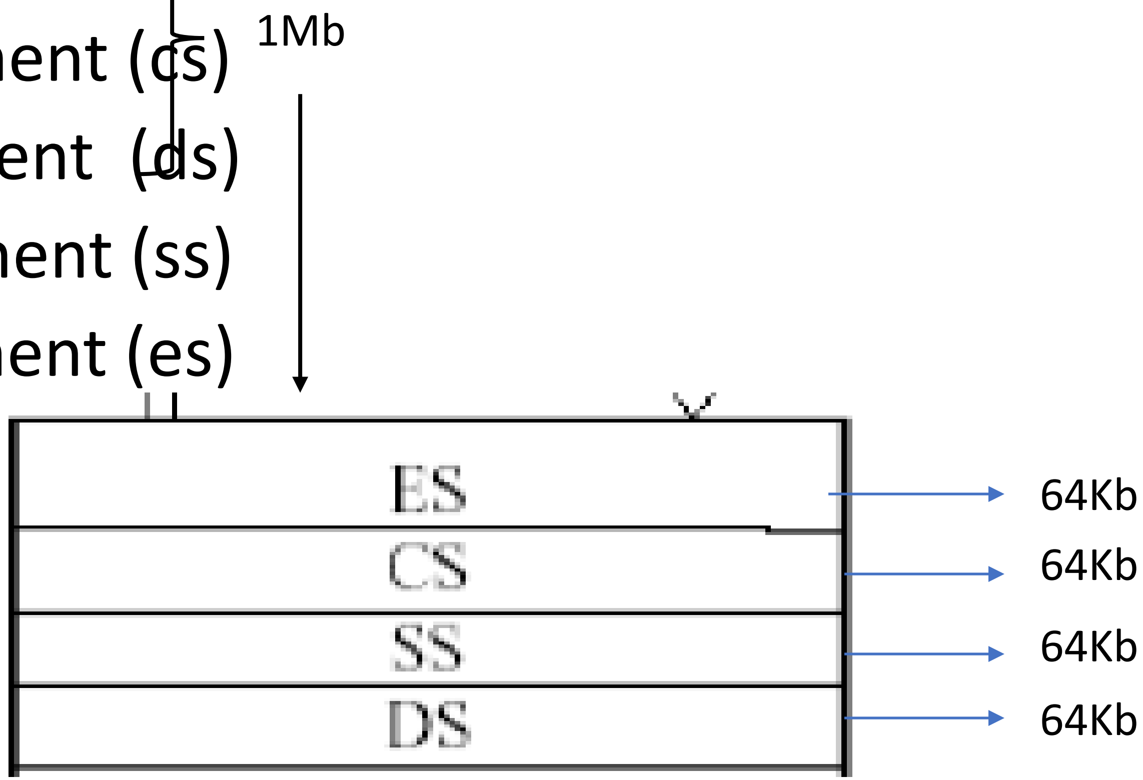
- Instruction pointer : it is a 16 bit register that keeps the address of memory location of coming instructions to be executed.
- Segment register : the memory space of mega bite of 8086 is segment into 4blocks . Each block is specified by register with maximum size of64kb.

➢Code segment (cs)

➢Data segment (ds)

➢Stack segment (ss)

➢Extra segment (es)



- ➢INSTRUCTION QUEUE: BIU performs its operation in parallel with execution unit.
- BIU fetch instruction byte while execution unit is executing operations.
- The prefetched instruction is saved in group of highspeed register and is known as instruction queue.

Execution unit

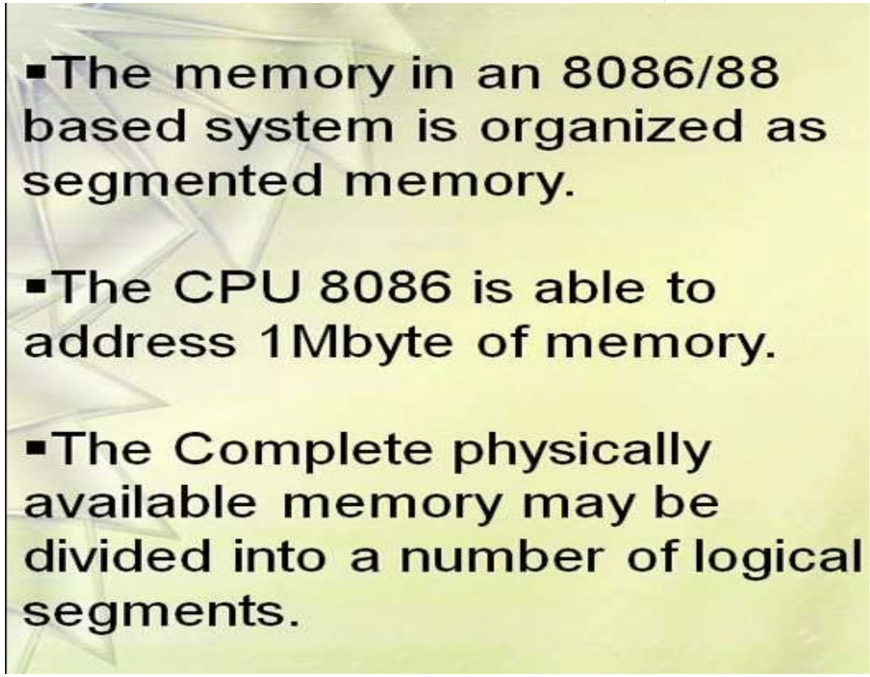
- The Execution unit is responsible for decoding and executing all instructions.
- The EU extracts instructions from the top of the queue in the BIU, decodes them, generates operands if necessary, passes them to the BIU and requests it to perform the read or write bys cycles to memory or I/O
- During the execution of the instruction, the EU tests the status and control flags and updates them based on the results of executing the instruction

Contd..

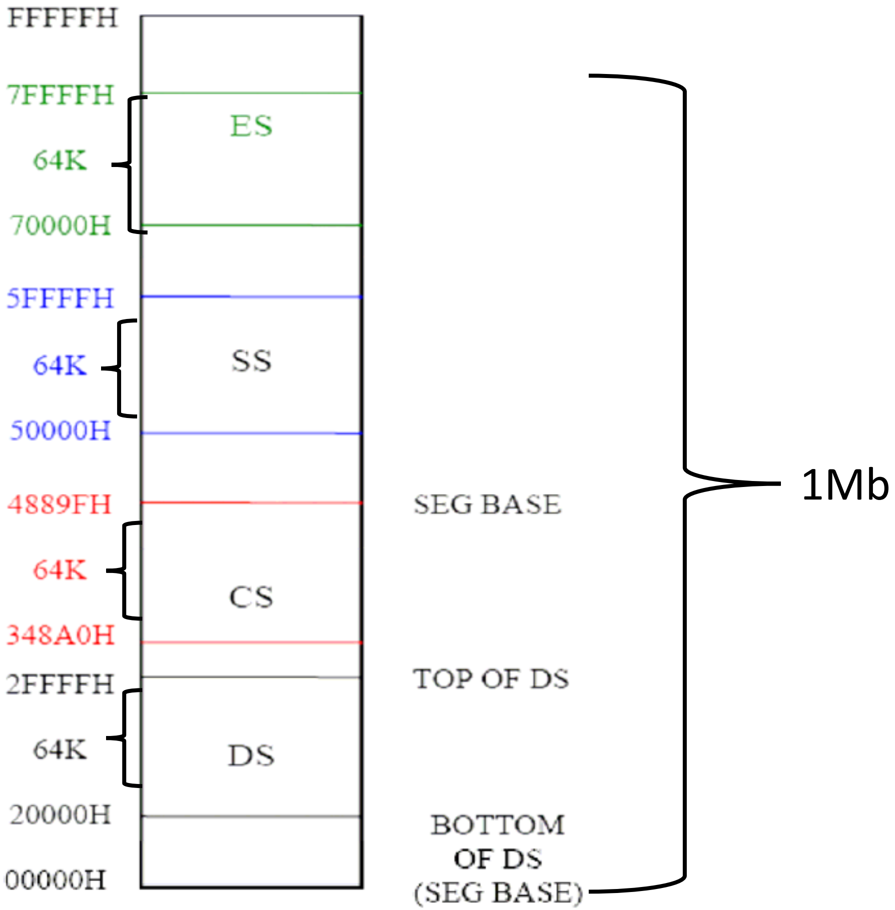
- If the queue is empty, the EU waits for the next instruction byte to be fetched and shifted to top of the queue.
- When the EU executes a branch or jump instruction, it transfers control to a location corresponding to another set of sequential instructions.
- Whenever this happens, the BIU automatically resets the queue and then begins to fetch instructions from this new location.

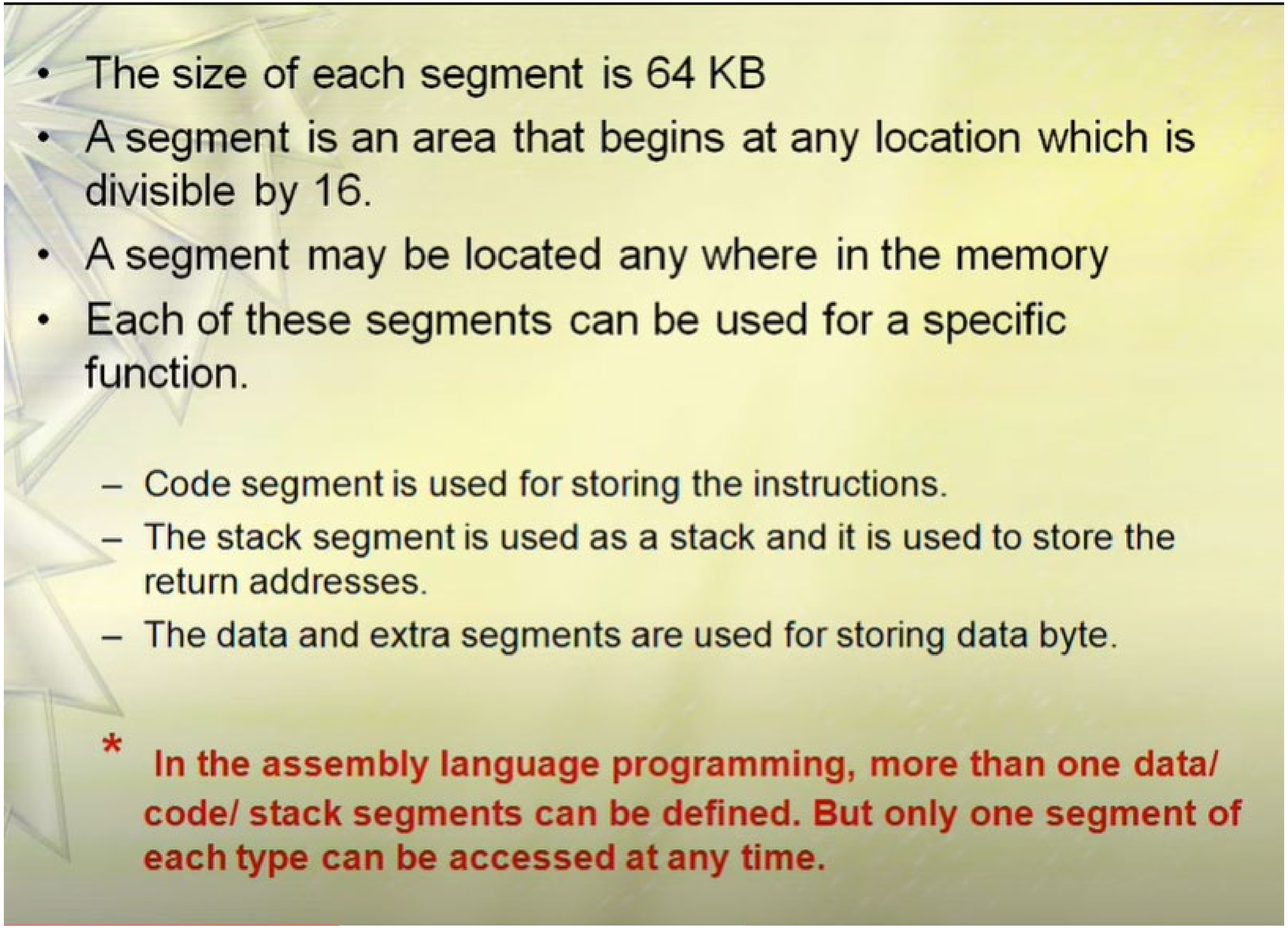
Memory Segmentation

- The division of the 1Mbyte memory of 8086 MP into 4 segments with 64Kb memory size called as memory segmentation. The 4 segments are code segment , data segment , extra segment , stack segment.



- The memory in an 8086/88 based system is organized as segmented memory.
- The CPU 8086 is able to address 1Mbyte of memory.
- The Complete physically available memory may be divided into a number of logical segments.





- The size of each segment is 64 KB
- A segment is an area that begins at any location which is divisible by 16.
- A segment may be located any where in the memory
- Each of these segments can be used for a specific function.
  - Code segment is used for storing the instructions.
  - The stack segment is used as a stack and it is used to store the return addresses.
  - The data and extra segments are used for storing data byte.

*** In the assembly language programming, more than one data/ code/ stack segments can be defined. But only one segment of each type can be accessed at any time.**

Advantages of memory segmentation

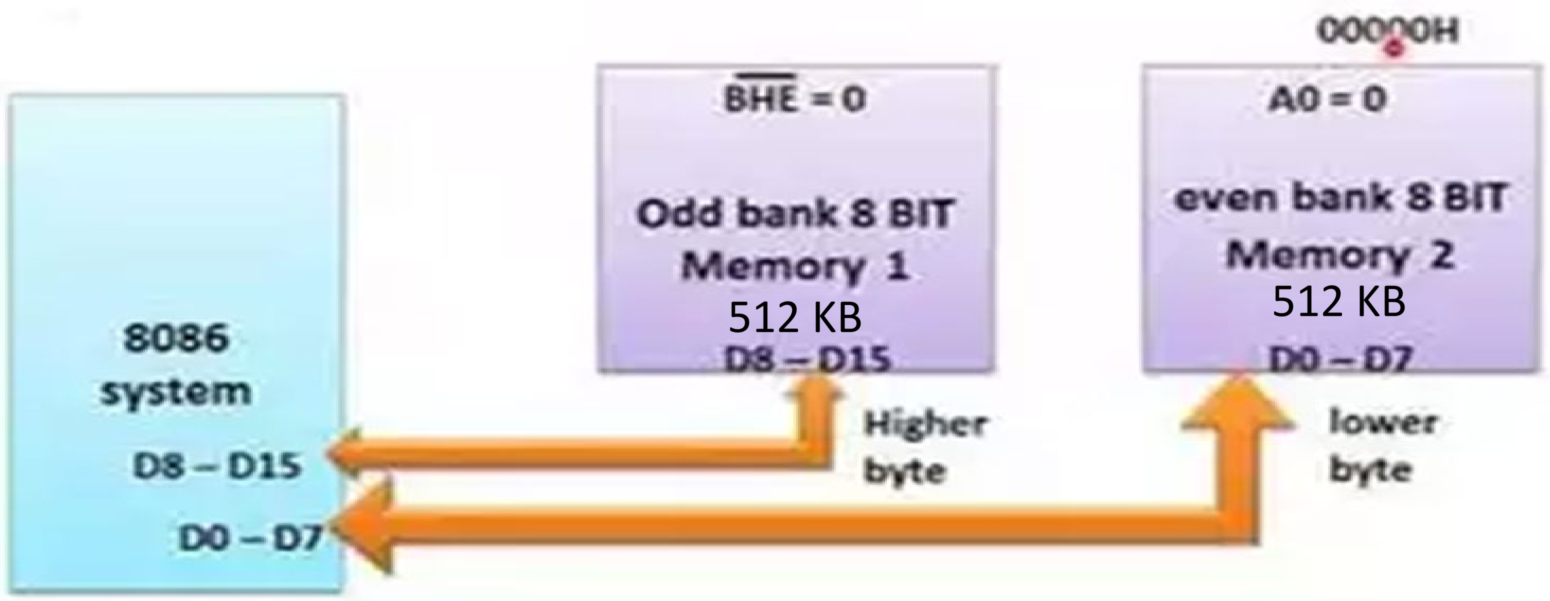
- Allow the memory capacity to be 1Mb even though the addresses associated with the individual instructions are only 16 bits wide.
- Facilitate the use of separate memory areas for the program, its data and the stack.
- Permit a program and/or its data to be put into different areas of memory each time the program is executed.
- Multitasking becomes easy.
- Allows the placing of code, data and stack portions of the same program in different parts (segments) of the m/y, for data and code protection.
- The segment registers are used to allow the instruction, data or stack portion of the program to be more than 64Kbytes long . The above can be achieved by using more than one code , data or stack segments.

PHYSICAL MEMORY ORGANISATION

Physical

8086 PHYSICAL MEMORY ORGANISATION

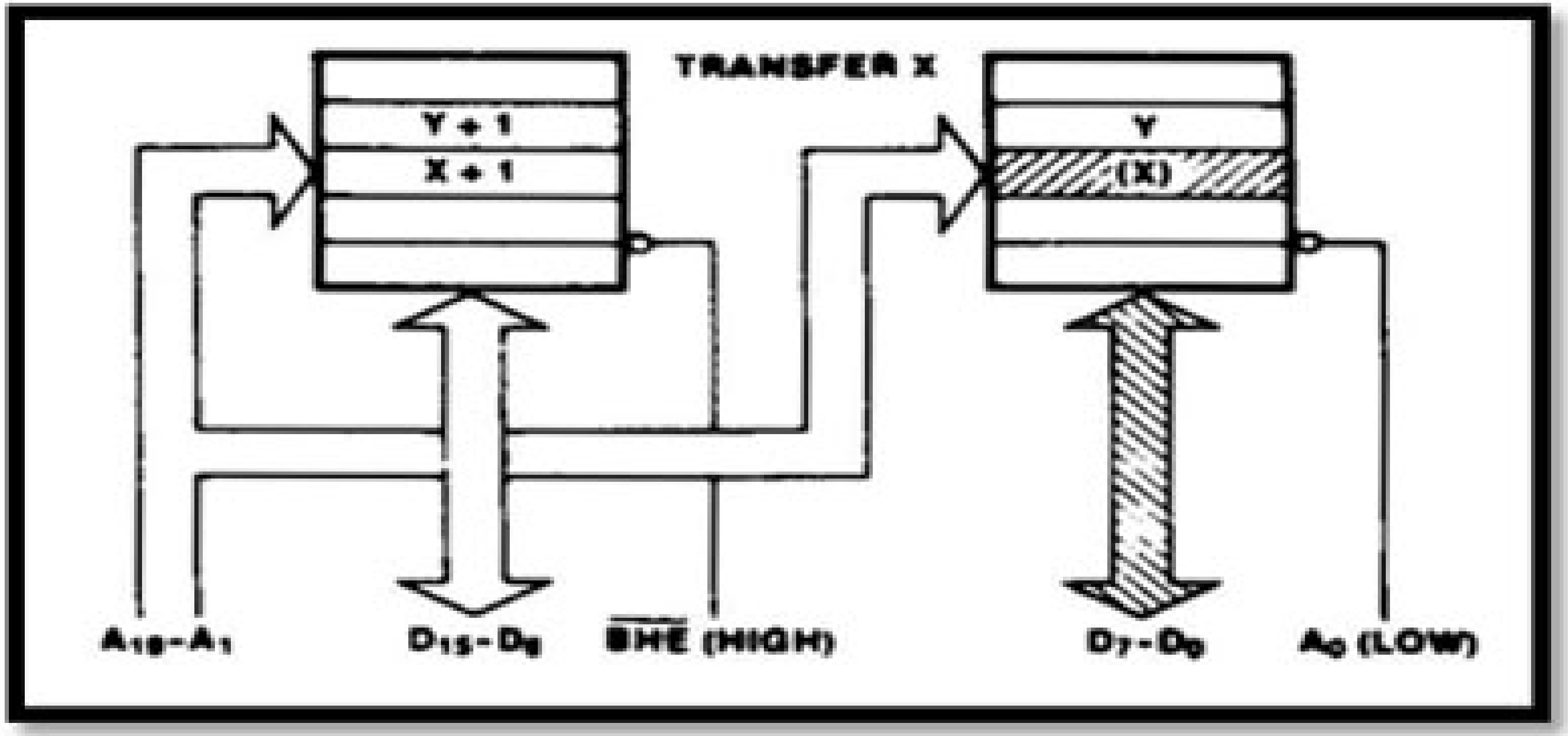
When word data come BIU required 2 machine cycles



- The 8086’s 1Mbyte memory address space is divided in to two independent 512Kbyte banks: the low (even) bank and the high (odd) bank.
- Data bytes associated with an even address (0000016, 0000216, etc.) reside in the low bank, and those with odd addresses (0000116, 0000316, etc.) reside in the high bank.
- Address bits A1 through A19 select the storage location that is to be accessed. They are applied to both banks in parallel. A0and bank high enable (BHE) are used as bank-select signals.

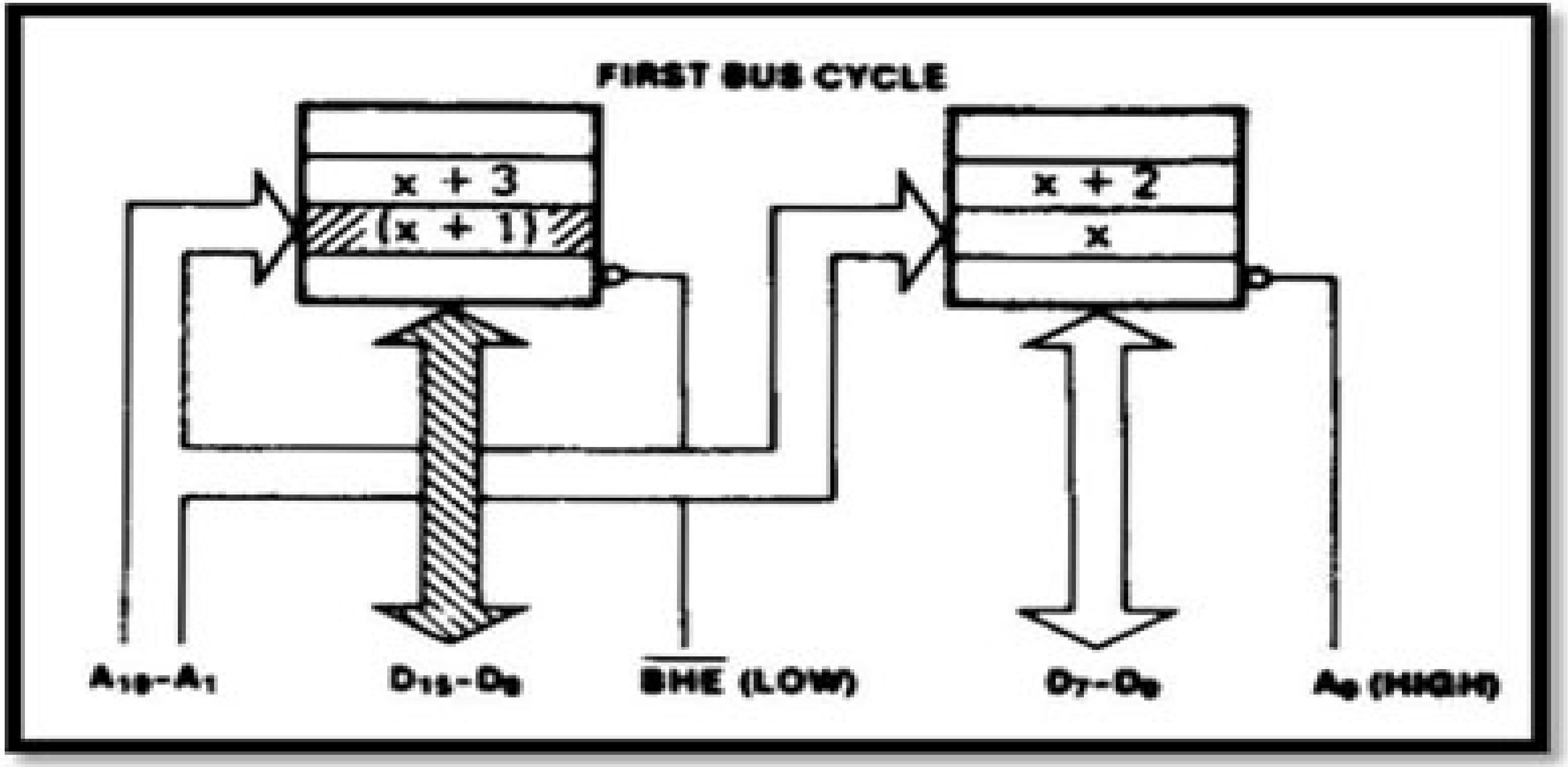
  The four different cases that happen during accessing data:

Case 1: When a byte of data at an even address (such as X) is to be accessed:



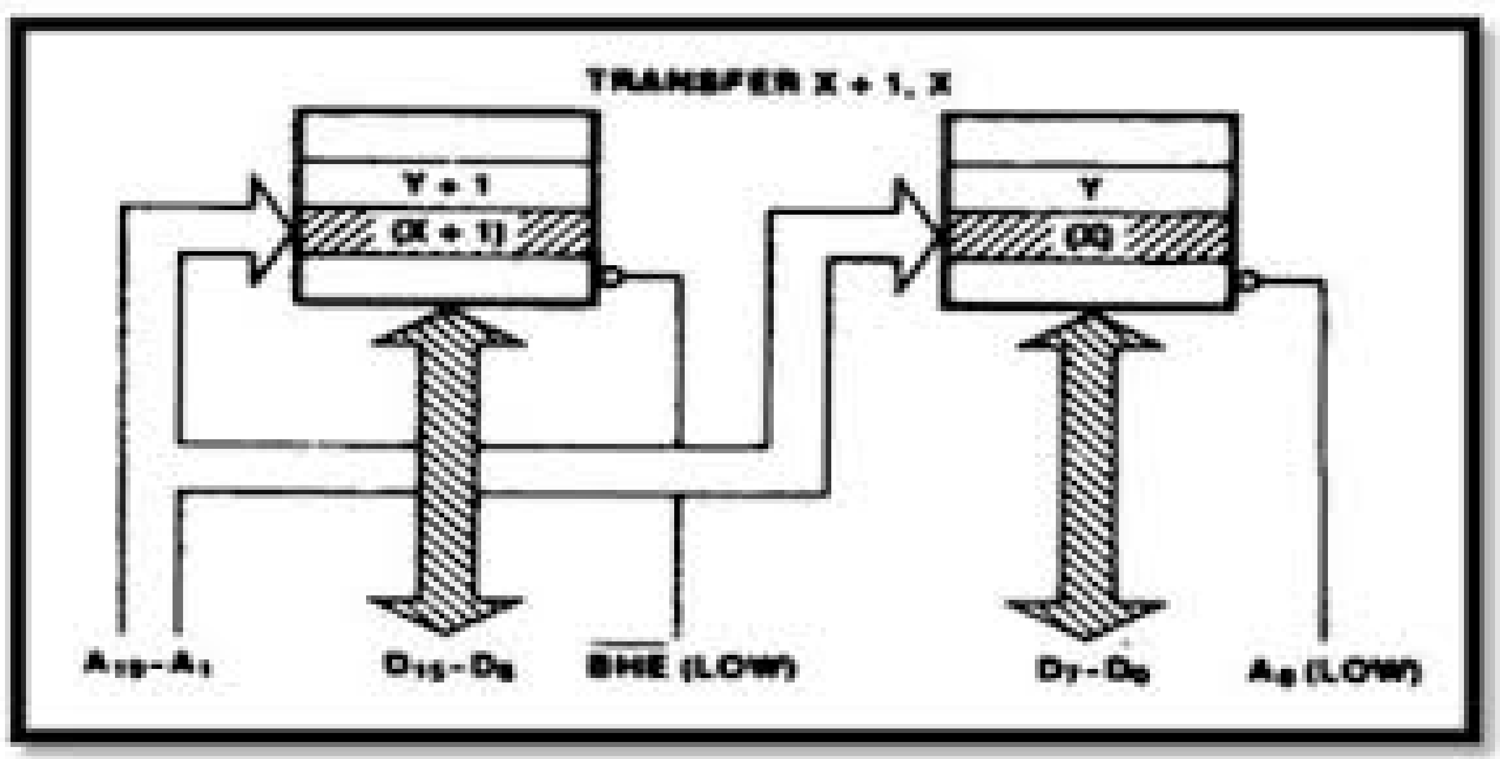
- •A0 is set to logic 0 to enable the low bank of memory.
- •BHE is set to logic 1 to disable the high bank.

Case 2: When a byte of data at an odd address (such as X+1) is to be accessed



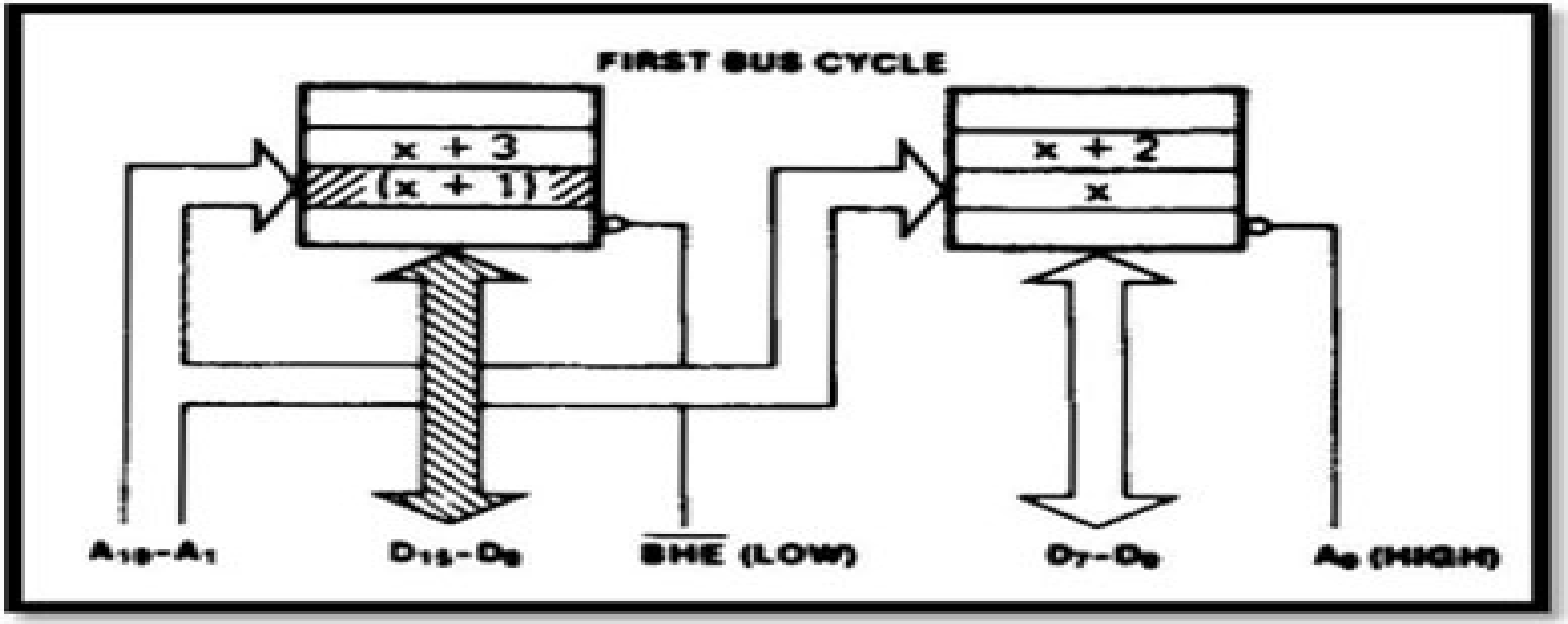
- •A0is set to logic 1 to disable the low bank of memory.
- •BHE is set to logic 0 to enable the high bank.

Case 3: When a word of data at an even address (aligned word) is to be accessed :



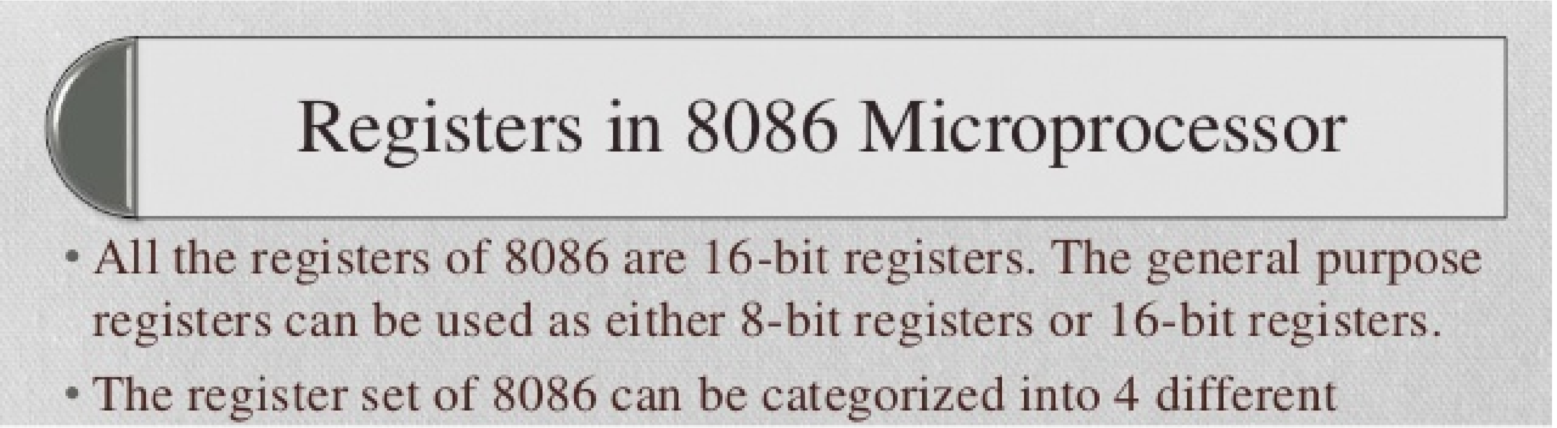
- A0 is set to logic 0 to enable the low bank of memory.
- BHE is set to logic 0 to enable the high bank.

Case 4: When a word of data at an odd address (misaligned word) is to be accessed, then the 8086 need two bus cycles to access it:
a) During the first bus cycle, the odd byte of the word (in the high bank) is addressed



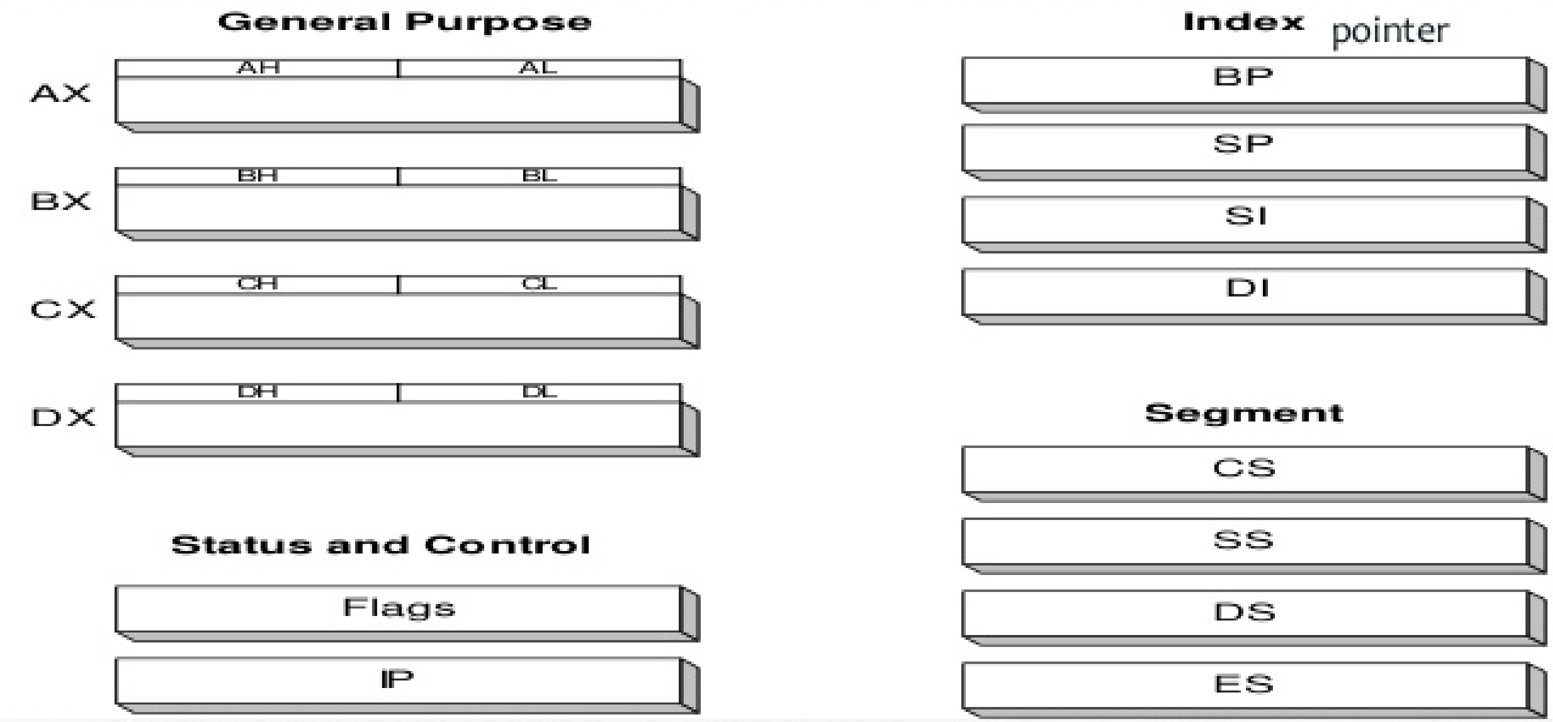
- A0 is set to logic 1 to disable the low bank of memory
- BHE is set to logic 0 to enable the high bank.

REGISTER ORGANISATION IN 8086 MP



Registers in 8086 Microprocessor

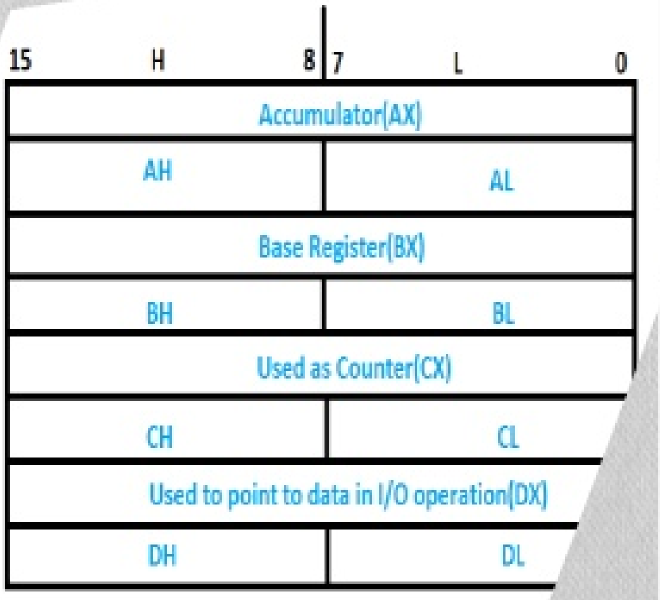
- All the registers of 8086 are 16-bit registers. The general purpose registers can be used as either 8-bit registers or 16-bit registers.
- The register set of 8086 can be categorized into 4 different



General purpose Register

The registers *AX*, *BX*,*CX* and *DX* are the general purpose 16-bit registers.

All data register can be used as either 16 bit or 8 bit. *BX* is a 16 bit register, but *BL* indicates the lower 8-bit of *BX* and *BH* indicates the higher 8-bit of *BX*.



15 H 8	7 L 0
Accumulator(AX)	
AH	AL
Base Register(BX)	
BH	BL
Used as Counter(CX)	
CH	CL
Used to point to data in I/O operation(DX)	
DH	DL

The register *CX* is used default counter in case of string and loop instructions.

AX is used as 16-bit accumulator. The lower 8-bit is designated as *AL* and higher 8-bit is designated as *AH*. *AL* can be used as an 8-bit accumulator for 8-bit operation.

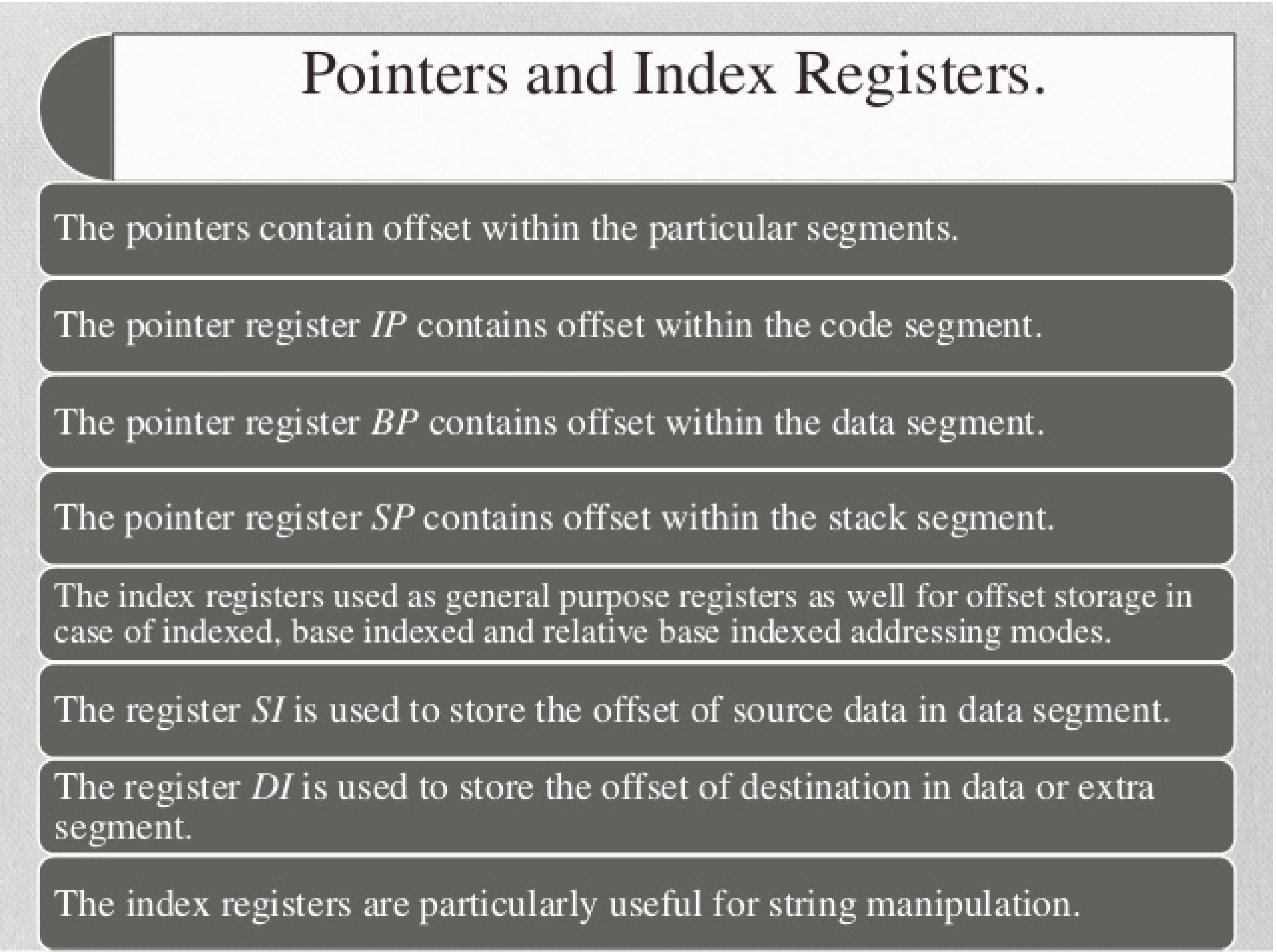
Segment Registers

Code segment register (CS): is used for addressing memory location in the code segment of the memory, where the executable program is stored.

Data segment register (DS): points to the data segment of the memory where the data is stored.

Extra Segment Register (ES) : also refers to a segment in the memory which is another data segment in the memory.

Stack Segment Register (SS): is used fro addressing stack segment of the memory. The stack segment is that segment of memory which is used to store stack data.



Pointers and Index Registers.

The pointers contain offset within the particular segments.

The pointer register *IP* contains offset within the code segment.

The pointer register *BP* contains offset within the data segment.

The pointer register *SP* contains offset within the stack segment.

The index registers used as general purpose registers as well for offset storage in case of indexed, base indexed and relative base indexed addressing modes.

The register *SI* is used to store the offset of source data in data segment.

The register *DI* is used to store the offset of destination in data or extra segment.

The index registers are particularly useful for string manipulation.

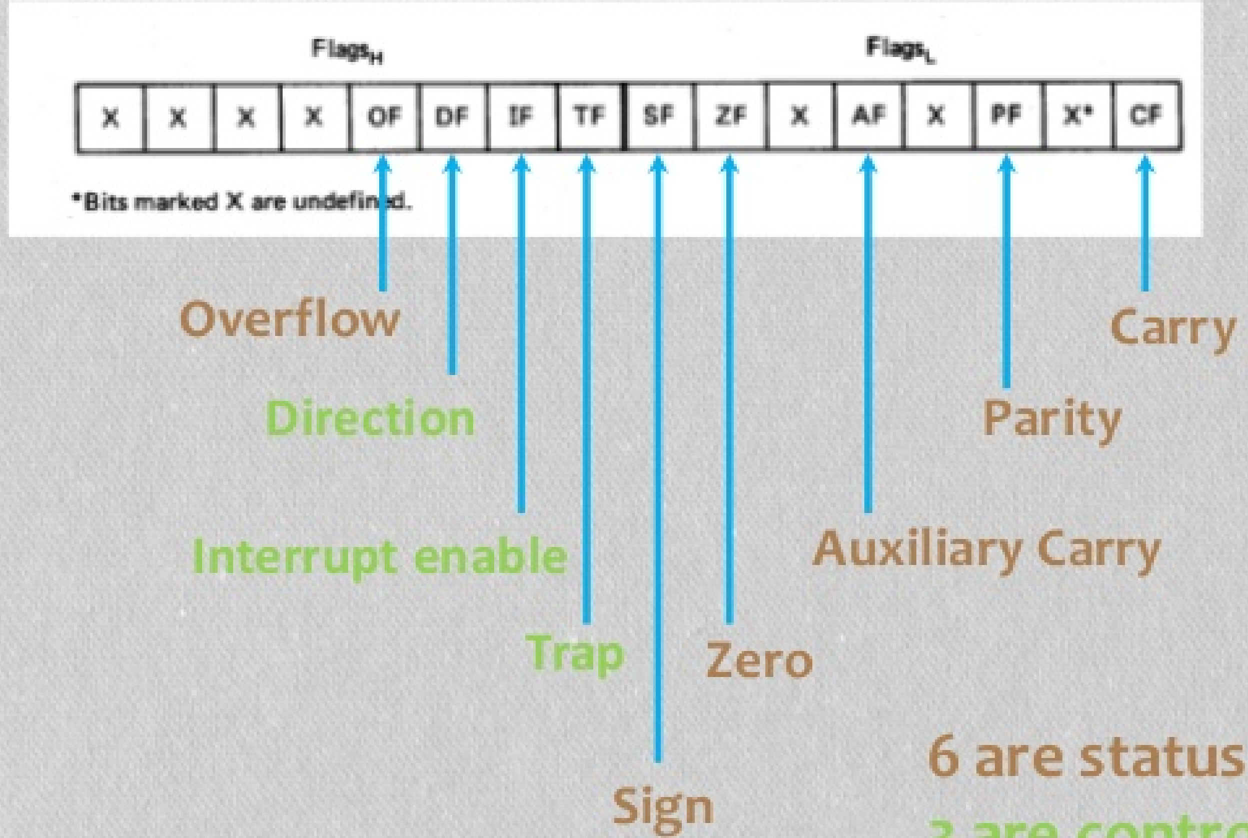
FLAG REGISTERS

The 8086 flag register contents indicate the results of computation in the *ALU*. It also contains some flag bits to control the *CPU* operations.

A 16 flag register is used in 8086. It is divided into two parts .

- Condition code or status flags- The condition code flag register is the lower byte of the 16-bit flag register. The condition code flag register is identical to 8085 flag register, with an additional overflow flag.
- Machine control flags- The control flag register is the higher byte of the flag register. It contains three flags namely direction flag(*D*), interrupt flag (*I*) and trap flag (*T*).

Flag Register



Flags$_H$								Flags$_L$							
X	X	X	X	OF	DF	IF	TF	SF	ZF	X	AF	X	PF	X*	CF

*Bits marked X are undefined.

Overflow

Direction

Interrupt enable

Trap

Sign

Zero

Auxiliary Carry

Parity

Carry

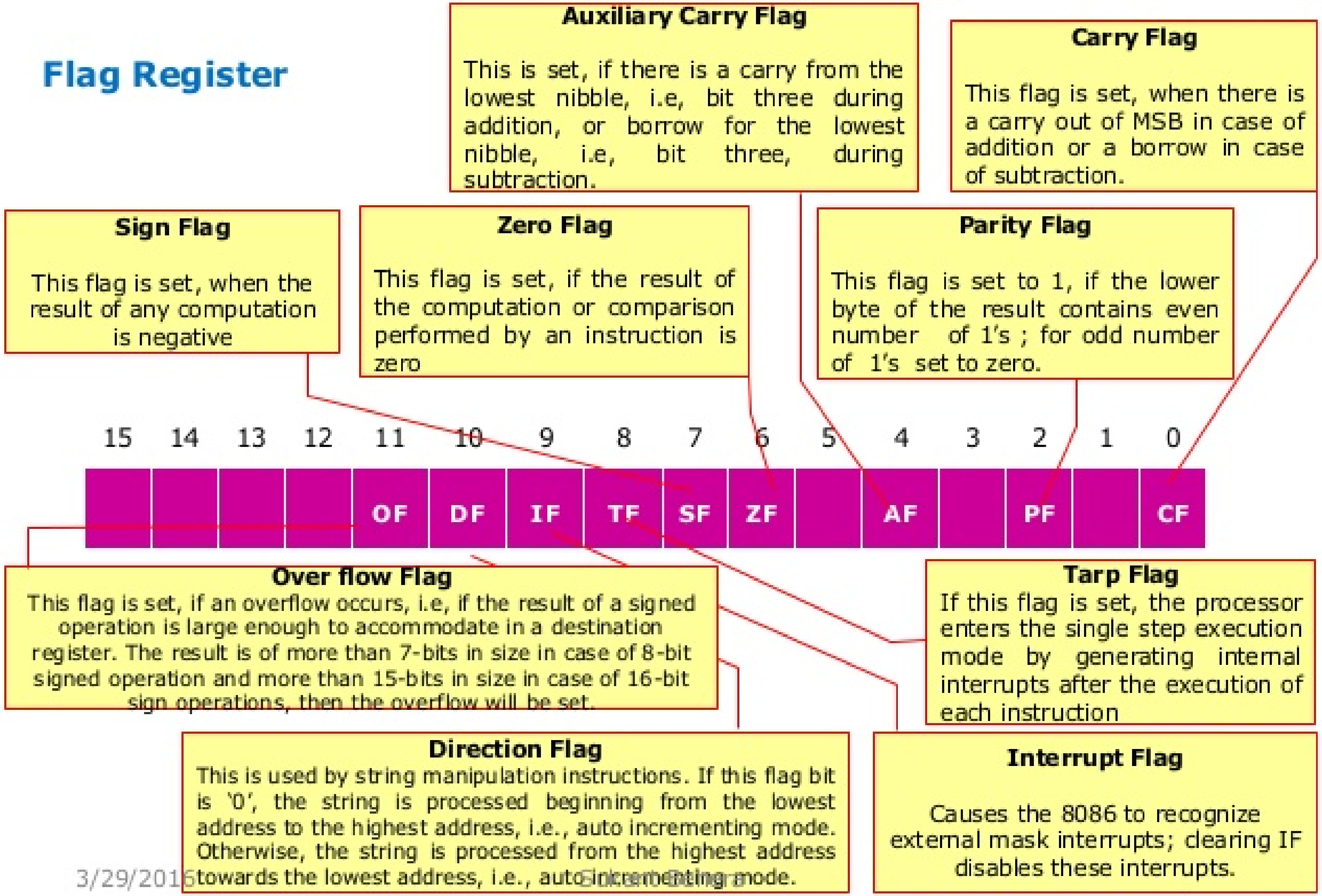
6 are status flags

3 are control flag

Architecture

Execution Unit (EU)

Flag Register



Auxiliary Carry Flag

This is set, if there is a carry from the lowest nibble, i.e, bit three during addition, or borrow for the lowest nibble, i.e, bit three, during subtraction.

Carry Flag

This flag is set, when there is a carry out of MSB in case of addition or a borrow in case of subtraction.

Sign Flag

This flag is set, when the result of any computation is negative

Zero Flag

This flag is set, if the result of the computation or comparison performed by an instruction is zero

Parity Flag

This flag is set to 1, if the lower byte of the result contains even number of 1's ; for odd number of 1's set to zero.

Over flow Flag

This flag is set, if an overflow occurs, i.e, if the result of a signed operation is large enough to accommodate in a destination register. The result is of more than 7-bits in size in case of 8-bit signed operation and more than 15-bits in size in case of 16-bit sign operations, then the overflow will be set.

Tarp Flag

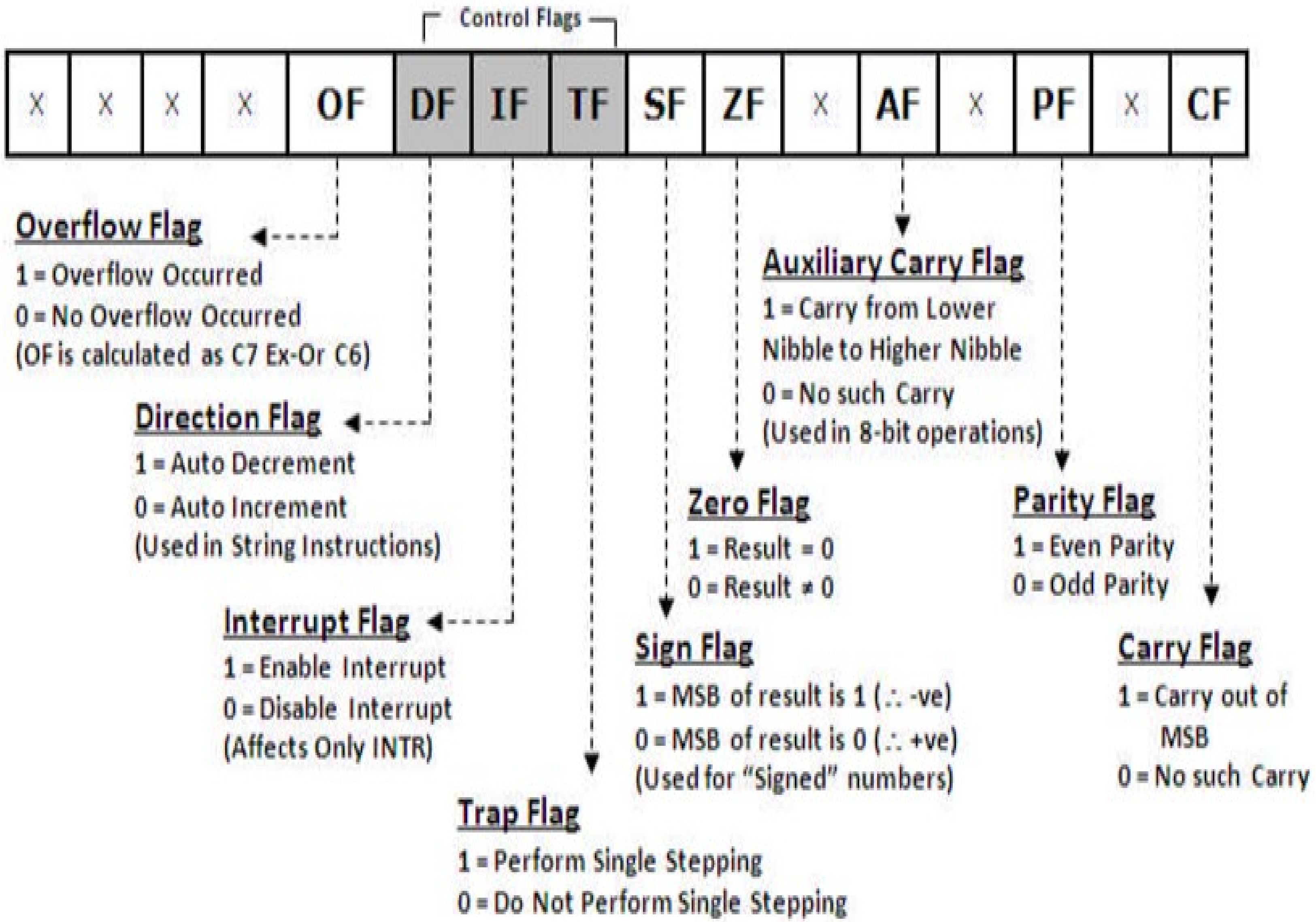
If this flag is set, the processor enters the single step execution mode by generating internal interrupts after the execution of each instruction

Direction Flag

This is used by string manipulation instructions. If this flag bit is '0', the string is processed beginning from the lowest address to the highest address, i.e., auto incrementing mode. Otherwise, the string is processed from the highest address towards the lowest address, i.e., auto incrementing mode.

Interrupt Flag

Causes the 8086 to recognize external mask interrupts; clearing IF disables these interrupts.



Control Flags

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

Overflow Flag

1 = Overflow Occurred

0 = No Overflow Occurred

(OF is calculated as C7 Ex-Or C6)

Direction Flag

1 = Auto Decrement

0 = Auto Increment

(Used in String Instructions)

Interrupt Flag

1 = Enable Interrupt

0 = Disable Interrupt

(Affects Only INTR)

Trap Flag

1 = Perform Single Stepping

0 = Do Not Perform Single Stepping

Sign Flag

1 = MSB of result is 1 (∴ -ve)

0 = MSB of result is 0 (∴ +ve)

(Used for "Signed" numbers)

Zero Flag

1 = Result = 0

0 = Result ≠ 0

Auxiliary Carry Flag

1 = Carry from Lower Nibble to Higher Nibble

0 = No such Carry

(Used in 8-bit operations)

Parity Flag

1 = Even Parity

0 = Odd Parity

Carry Flag

1 = Carry out of MSB

0 = No such Carry

Status Flag

S- Sign Flag : This flag is set, when the result of any computation is negative.

Z- Zero Flag: This flag is set, if the result of the computation or comparison performed by the previous instruction is zero.

P- Parity Flag: This flag is set to 1, if the lower byte of the result contains even number of 1's.

C- Carry Flag: This flag is set, when there is a carry out of MSB in case of addition or a borrow in case of subtraction.

Status Flag

AC-Auxiliary Carry Flag: This is set, if carry from the lowest nibble, i.e., bit three during addition, or borrow for the lowest nibble, i.e., bit three, during subtraction.

O- Over flow Flag: This flag is set, if an overflow occurs, i.e., if the result of a signed operation is large enough to accommodate in a destination register. The result is of more than 7-bits in size in case of 8-bit signed operation and more than 15-bits in size in case of 16-bit sign operations, then overflow will be set.

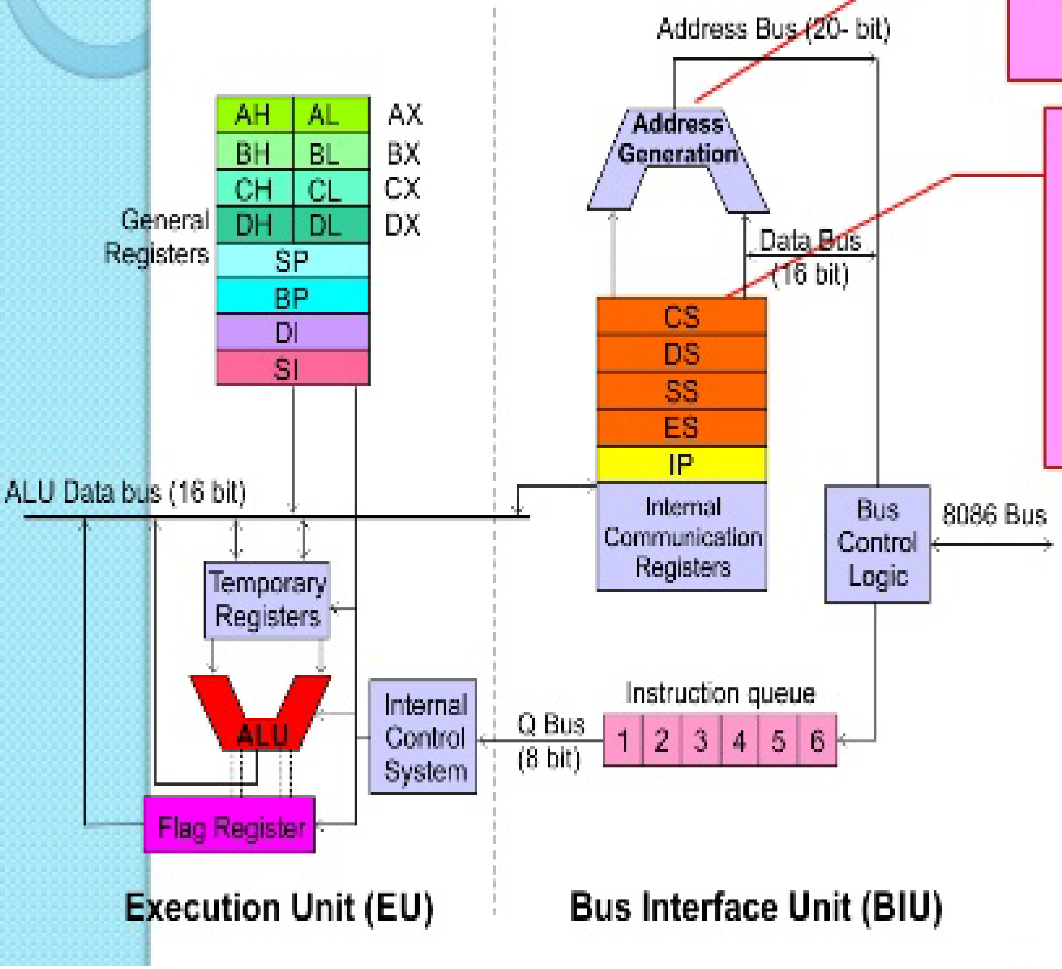
Control flag

T- Tarp Flag: If this flag is set, the processor enters the single step execution mode.

I- Interrupt Flag: If this flag is set, the maskable interrupt are recognized by the CPU, otherwise they are ignored.

D- Direction Flag: This is used by string manipulation instructions. If this flag bit is ‘0’, the string is processed beginning from the lowest address to the highest address, i.e., auto incrementing mode. Otherwise, the string is processed from the highest address towards the lowest address, i.e., auto incrementing mode.

BUS INTERFACE UNIT



Dedicated Adder to generate 20 bit address

Four 16-bit segment registers

Code Segment (CS)
Data Segment (DS)
Stack Segment (SS)
Extra Segment (ES)

BUS INTERFACE UNIT

ADDRESS GENERATION

- Eg: physical address=155A5h (20 bit)
- Segment address=1005h (16 bit)

  0001 0000 0000 0101

- Offset=5555h (16 bit)

0001 0000 0000 0101 0000 ← Segment addr left shifted with appended zeros

+ 0101 0101 0101 0101 ← Offset

0001 0101 0101 1010 0101

1 5 5 A 5 ← Physical address

Generation of 20 bit physical address

The 20-bit Physical address is often represented as:

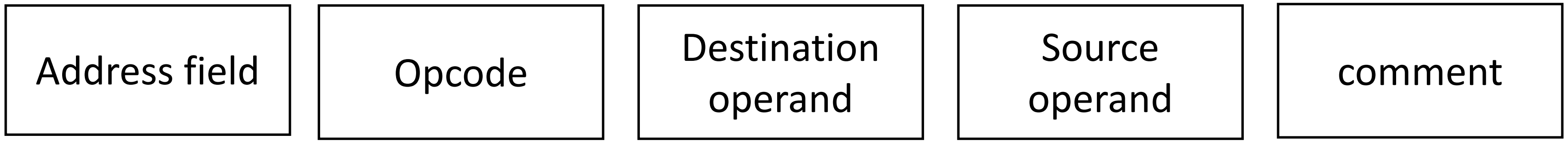
Segment Base address+ Offset OR CS + IP

```
 CS      3 4 8 0 0 →Implied Zero (from shft Left)
+IP      1 2 3 4
 ----------------------
         3 5 A3 4 H
```

- So , physical address = base address of segment registers + offset value of pointer registers/index registers.
- Base address of segment registers obtained by appending the 0 at the LSB position of segment base values.

- Always physical addresses are generated by adding like these:
- Code segment base address + offset value of instruction pointer
- Data segment base address + off set value of base pointer
- Extra segment base address + off set value of index registers/base pointer
- Stack segment base address + off set value of stack pointer
- If segment value is given in hexadecimal then physical address can be calculated as
- Physical address=segment value×10+offset value
- For example CS=2345 ,offset of IP is1020 THEN
- Pysical Address =2345×10+1020
- =23450+1020
- =24470

PROGRAMING MODEL



Address field

Opcode

Destination operand

Source operand

comment

- Address field is address of the instruction where the instruction is stored in the memory.
- Opcode means operation code.
- Destination operand is the place where data to be transferred.it may register / memory.
- Source operand is the place from which data is transmitted to destination apprehend . It may be memory / register.
- Comment is description of programme statement .

Addressing modes of 8086

Every instruction of programme has to operate on data. The different way in which source operand is denoted in an instruction are know as addressing modes

The addressing mode describes the types of operands and the way they are accessed for

executing an instruction. According to the flow of instruction execution, the instructions may be categorized as

1. Sequential control flow instructions (Data Category)

2. Control transfer instructions(Branch Category)

Sequential control flow instructions are the instructions which after execution, transfer

control to the next instruction appearing immediately after it (in the sequence) in the program.

For example

the arithmetic, logic, data transfer and processor control instructions are Sequential

control flow instructions.

The control transfer instructions on the other hand transfer control to some predefined

address or the address somehow specified in the instruction, after their execution.

For example

INT, CALL, RET & JUMP instructions fall under this category.

The addressing modes for Sequential flow instructions are explained as follows. .

1. Immediate addressing mode

2. Direct addressing mode

3. Register addressing mode

4. Register indirect addressing mode

5. Indexed addressing mode

6. Register relative addressing mode

7. Based indexed addressing mode

8. . Relative based indexed

1. Immediate addressing mode:

In this type of addressing, immediate data is a part of instruction, and appears

in the form of successive byte or bytes.

Example: MOV AX, 0005H.

In the above example, 0005H is the immediate data. The immediate data may be 8- bit or 16-bit in size.

2. Direct addressing mode:

In the direct addressing mode, a 16-bit memory address (offset) directly specified in the

instruction as a part of it.

Example: MOV AX, [5000H].

3. Register addressing mode:

In the register addressing mode, the data is stored in a register and it is referred using the

particular register. All the registers, except IP, may be used in this mode.

Example: MOV BX, AX

4. Register indirect addressing mode:

Sometimes, the address of the memory location which contains data or operands is determined in an indirect way, using the offset registers. The mode of addressing is known as register indirect mode.

In this addressing mode, the offset address of data is in either BX or SI or DI Register. The default segment is either DS or ES.

Example: MOV AX, [BX].

5. Indexed addressing mode:

In this addressing mode, offset of the operand is stored one of the index registers. DS & ES are the default segments for index registers SI & DI respectively.

Example: MOV AX, [SI]

Here, data is available at an offset address stored in SI in DS.

6. Register relative addressing mode:

In this addressing mode, the data is available at an effective address formed by adding an 8-bit or 16-bit displacement with the content of any one of the register BX, BP, SI & DI in the default(either in DS & ES) segment.

Example: MOV AX, 50H [BX]

7. Based indexed addressing mode:

The effective address of data is formed in this addressing mode, by adding content of a base register (any one of BX or BP) to the content of an index register (any one of SI or DI).

Thedefault segment register may be ES or DS.

Example: MOV AX, [BX][SI]

8. Relative based indexed:

The effective address is formed by adding an 8 or 16-bit displacement with the sum of contents of any of the base registers (BX or BP) and any one of the index registers, in a default segment.

Example: MOV AX, 50H [BX] [SI]

Addressing Modes for control transfer instructions

For the control transfer instructions, the addressing modes depend upon whether the destination location is within the same segment or in a different one.

It also depends upon the method of passing the destination address to the processor.

Basically, there are two addressing modes for the control transfer instructions,

They are **Inter segment and intra segment addressing modes.**

If the location to which the control is to be transferred lies in a different segment other than the current one, the mode is called intersegment mode.

If the destination location lies in the same segment, the mode is called intra segment mode.

- **Addressing Modes for control transfer instructions:**
- 1. Intersegment
- ▪ Intersegment direct
- ▪ Intersegment indirect
- 2. Intrasegment
- ▪ Intrasegment direct
- ▪ Intrasegment indirect
- **1. Intersegment direct:**
- In this mode, the address to which the control is to be transferred is in a different segment.
- This addressing mode provides a means of branching from one code segment to another code
- segment. Here, the CS and IP of the destination address are specified directly in the instruction.
- **Example: JMP 5000H, 2000H; jump to effective address 2000H in segment 5000H.**
- **2. Intersegment indirect:**
- In this mode, the address to which the control is to be transferred lies in a different
- segment and it is passed to the instruction indirectly, i.e. contents of a memory block
- containing four bytes, i.e. IP(LSB), IP(MSB), CS(LSB) and CS(MSB) sequentially. The
- starting address of the memory block may be referred using any of the addressing modes,
- except immediate mode.
- **Example: JMP [2000H].**
- Jump to an address in the other segment specified at effective address 2000H in DS.

3. Intrasegment direct mode:

In this mode, the address to which the control is to be transferred lies in the same segment in which the control transfers instruction lies and appears directly in the instruction as an immediate displacement value. In this addressing mode, the displacement is computed relative to the content of the instruction pointer.

The effective address to which the control will be transferred is given by the sum of 8 or 16 bit displacement and current content of IP. In case of jump instruction, if the signed displacement (d) is of 8-bits (i.e. -128<d<+127), it as short jump and if it is of 16 bits (i.e. -32768<d<+32767), it is termed as long jump.

Example: JMP SHORT LABEL.

4. Intrasegment indirect mode:

In this mode, the displacement to which the control is to be transferred is in the same segment in which the control transfer instruction lies, but it is passed to the instruction directly. Here, the branch address is found as the content of a register or a memory location.

This addressing mode may be used in unconditional branch instructions.

Example: JMP [BX]; Jump to effective address stored in BX.

INSTRUCTION SET OF 8086

- Classified into 7 categories:
- 1] Data Transfer
- 2] Arithmetic
- 3] Logical
- 4] Control
- 5]Processor Control Instructions
- 6] String Manipulation
- 7] Interrupt Control

Data Transfer Instructions

- Note : Data Transfer Instructions do not affect any flags
- 1] **MOV dest, src**
- Note that source and destination cannot be memory location. Also source and destination must be same type.
- 2] **PUSH Src:** ***Copies word on stack.***
- 3] **POP dest:** ***Copies word from stack into dest. Reg.***
- 4] **IN acc, port :** ***Copies 8 or 16 bit data from port to accumulator.***
- a) Fixed Port
- b) Variable Port
- 5] **OUT port, acc**

Data Transfer Instructions Cont…

- 6] **LES Reg, Mem:** ***Load register and extra segment*** *register with words from memory.*
- 7] **LDS Reg,Mem:** ***Load register and data segment*** *register with words from memory.*
- 8] **LEA Reg,Src:** ***load Effective address.*** (Offset is loaded in specified register)
- 9] **LAHF:** ***Copy lower byte of flag register into AH*** *register.*
- 10] **SAHF:** ***Copy AH register to lower byte of flag***

Data Transfer Instructions Cont …

- 11] **XCHG dest, src:** *Exchange contents of source and destination.*
- 12] **XLAT:** ***Translate a byte in AL.***

  This instruction replaces the byte in AL with byte pointed by BX.To point desired byte in look up table instruction adds contains of BX with AL (BX+ AL). Goes to this location and loads into AL.

Arithmetic Instructions

- 1]**ADD dest,src**
- 2] **ADC dest,src:** ***Add with carry***
- 3] **AAA :** ***ASCII adjust after addition.***

  We can add two ASCII numbers directly and use AAA after addition so as to get result directly in BCD. (Works with AL only)
- 4] **DAA :** ***Decimal adjust accumulator.***

  (Works with AL only)

Arithmetic Instructions Cont…

- 5] **SUB dest, src**

- 6] **SBB dest, src:** ***Subtract with borrow.***

- 7] **AAS:** ***ASCII adjust for subtraction***
  (same as AAA and works with AL only)

- 8] **DAS :** ***Decimal adjust after Subtraction.***
  (works with AL only)

- 9] **MUL src**

- 10] **IMUL src:** ***Multiplication of signed byte.***

Arithmetic Instructions Cont…

- 11] **AAM:** ***BCD adjust after multiply.***

  (works with AL only)
- 12]**DIV src**

  If any one attempts to divide by 0 , then ?
- 13] **IDIV:** ***Division of signed numbers***
- 14]**AAD:** ***BCD to Binary convert before*** *Division.*
- 15] **DEC dest**

Arithmetic Instructions Cont…

- 16] **INC dest**
- 17] **CWD:** ***Convert signed word to signed*** *double word.*
- 18] **CBW :** ***Convert signed byte to signed*** *word.*

  (CBW and CWD works only with AL, AX and DX)
- 19] **NEG dest:** ***Forms 2's complement.***

Logical Instructions

- 1] **AND dest, src**
- 2] **NOT dest:** ***Invert each bit in destination***
- 3] **OR dest, src**
- 4] **XOR dest, src**
- 5] **RCL dest, count :** ***Rotate left through Carry***

  Rotate as many times as directly specified in the instruction. For more no.of rotations, count can be specified in CL register.

- 6] **RCR dest, count :** ***Rotate right through carry***
- 7] **ROL dest, count :** ***Rotate left (into carry as well as into LSB)***
- 8] **ROR dest, Count :** ***Rotate left (into carry as well as into MSB)***

Logical Instructions Cont…

- 9] **SAL/ SHL dest, count :** ***Shift left and append 0s on right.***
- 10] **SAR dest, count :** ***Shift right*** retain a copy of the S-bit and shift all bits to right.
- 11]**SHR dest, count :** ***Shift right append 0s on left***
- 12] **TEST dest, src:** ***AND logically, updates flags but source and dest are unchanged.***

Logical Instructions Cont…

- 13] **CMP dest, src**
- CF, ZF and SF are used

Ex. CMP CX,BX

	CF	ZF	SF
• CX = BX	0	1	0
• CX> BX	0	0	0
• CX<BX	1	0	1

CONTROL TRANSFER INSTRUCTIONS

- 1]**CALL :** ***Call a procedure***

  Two types of calls:

  i) Near Call (Intrasegment)

  ii) Far Call (Intersegment)

- 2] **RET :** ***Return execution from procedure***
- 3] **JMP :** ***Unconditional Jump to specified destination. Two types near and Far***

CONTROL TRANSFER INSTRUCTIONS Cont…

- 4] **JA / JNBE:** ***Jump if above / Jump if not below***

  The terms above and below are used when we refer to the magnitude of Unsigned number .

  Used normally after CMP.
- 5] **JAE / JNB / JNC**
- **6] JB / JC / JNAE**
- **7] JBE / JNA**
- **8] JE/ JZ**

CONTROL TRANSFER INSTRUCTIONS Cont...

- 9] **JCXZ:** ***Jump if CX is Zero.***
- 10] **JG / JNLE:** ***Jump if Greater /Jump if NOT less than or equal.***

  The term greater than or less than is used in connection with two signed numbers.
- 11] **JGE / JNL:**
- 12] **JL / JNGE :**
- 13] **JLE / JNG :**
- 14]**JNE / JNZ :**

CONTROL TRANSFER INSTRUCTIONS Cont...

- 15] **JNO :** ***Jump if no overflow***
- 16] **JNS :** ***Jump if no sign***
- 17] **JS**
- 18] **JO**
- 19] **JNP / JPO**
- 20] **JP / JPE**

  In all above conditional instructions the destination of jump is in the range of -128 to + 127 bytes from the address after jump.

CONTROL TRANSFER INSTRUCTIONS Cont…

- 21] **LOOP:** ***Loop to the specified label if CX is not equal to Zero.***

  The count is loaded in CX reg. Every time LOOP is executed, CX is automatically decremented - used in delay programs

- 22] **LOOPE/ LOOPZ:** ***Loop while CX is not equal to zero and ZF = 1.***

- 23] **LOOPNE / LOOPNZ:** ***Loop while CX not equal to zero and ZF = 0.***

  In all above LOOP instructions the destination of jump is in the range of -128 to + 127 bytes from the address after LOOP.

PROCESSOR CONTROL

- 1] **CLC:** ***Clear Carry flag.***
- 2] **STC :*Set carry Flag***
- 3] **CMC :*Complement Carry Flag***
- 4] **CLD:** ***Clear Direction Flag.***
- 5] **STD:** ***Set Direction Flag***
- 6] **CLI :*Clear Interrupt Flag.***
- 7] **STI :** ***Set Interrupt Flag.***
- 8] **HLT:** ***Halt Processing.***

PROCESSOR CONTROL Cont...

- 9] **NOP :** ***No Operation***
- 10] **ESC:** ***Escape***

  Executed by Co-processors and actions are performed according to 6 bit coding in the instruction.
- 11] **LOCK :** ***Assert bus lock Signal***

  This is a prefix instruction.
- 12] **WAIT :*****Wait for test or Interrupt Signal.***

  Assert wait states.

STRING CONTROL

- 1] **MOVS/ MOVSB/ MOVSW**

  Dest string name,src string name

  This instn moves data byte or word from location in DS to location in ES.

- 2] **REP / REPE / REPZ / REPNE / REPNZ**

  Repeat string instructions until specified conditions exist.

  This is prefix a instruction.

STRING CONTROL Contd…

- 3] **CMPS / CMPSB / CMPSW**

  Compare string bytes or string words.

- 4] **SCAS / SCASB / SCASW**

  Scan a string byte or string word.

  Compares byte in AL or word in AX. String address is to be loaded in DI.

- 5] **STOS / STOSB / STOSW**

  Store byte or word in a string.

  Copies a byte or word in AL or AX to memory location pointed by DI.

- 6] **LODS / LODSB /LODSW**

  Load a byte or word in AL or AX

- Copies byte or word from memory location pointed by SI into AL or AX register.

Interrupt Control

- 1]**INT type**

- 2] **INTO** ***Interrupt on overflow***

- 3] **IRET** ***Interrupt return***

ASSEMBLER DIRECTIVES

- 1] **ASSUME**

  Used to tell assembler the name of logical segment. Ex. ASSUME CS: Code here
- 2] **END**
- 3] **DB**
- 4] **DW**
- 5] **DD** ***Define Double Word***
- 6] **DQ** ***Define Quad Word***
- 7] **DT** ***Define Ten Bytes***

ASSEMBLER DIRECTIVES Cont…

- 8] **PROC** ***Procedure***

  PROC DELAY NEAR
- 9] **ENDP**
- 10] **ENDS**
- 11] **EQU**
- 12] **EVEN:** ***Align on even memory address.***
- 13] **ORG**
- 14] **OFFSET**

  Ex: MOV BX, Offset of Data Here
- 15] **PTR** ***Pointer***

ASSEMBLER DIRECTIVES Cont...

- 16] **LABEL**

  Ex: AGAIN LABEL FAR

- 17] **EXTRN**

  Tells the assembler that the names or labels following this directive is in some other assembly module.

- 18] **PUBLIC**

  Links modules together

ASSEMBLER DIRECTIVES Cont...

- 19] **INCLUDE**
  Include source code from file.

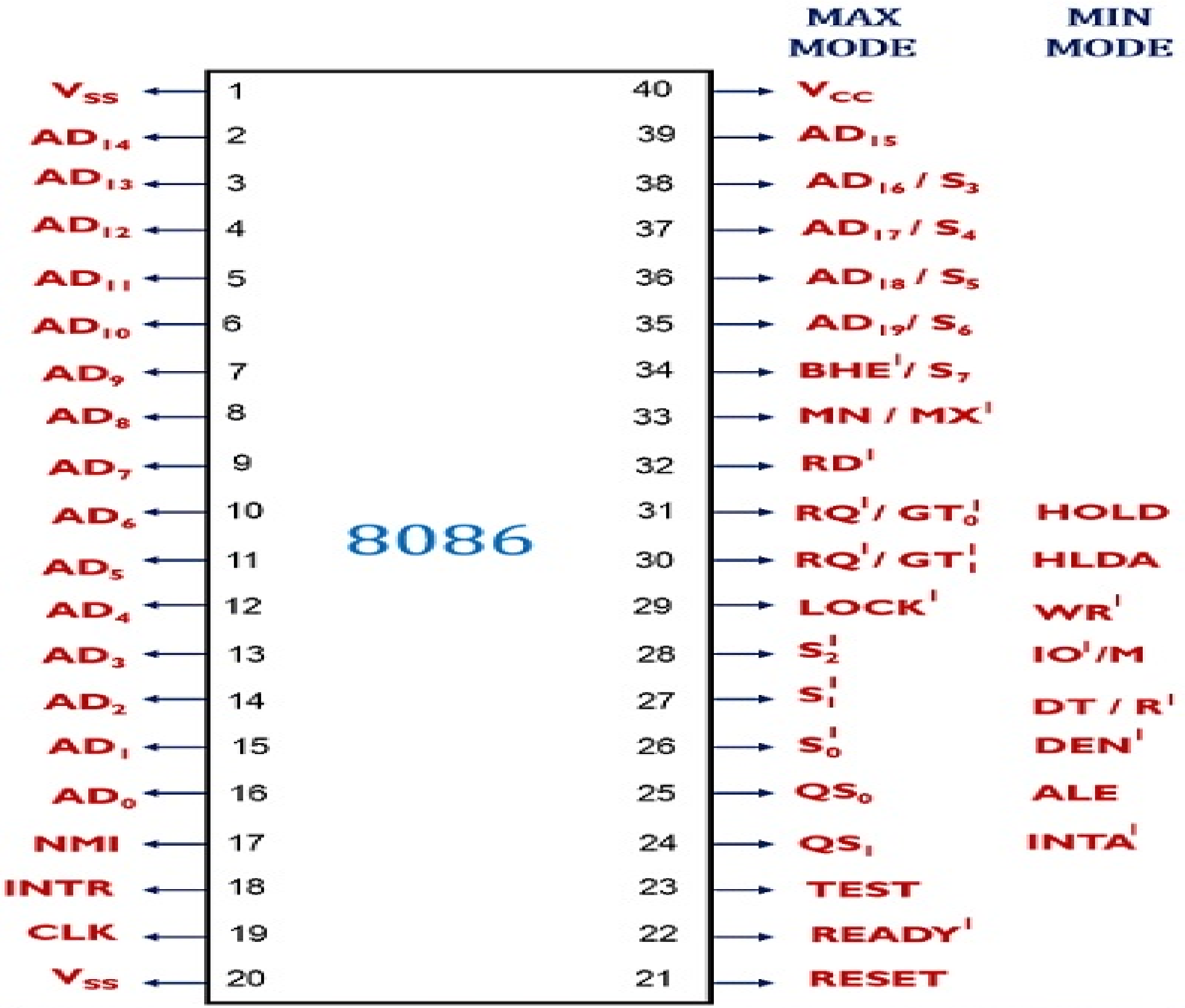
- 20] **NAME**
  To give specific name to module.

- 21] **GROUP**
  Grouping of logical segments.

- 22] **SEGMENT**

- 23] **SHORT**
  Operator that tells assembler about short displacement.

- 24] **TYPE**
  Type of variable whether byte or word.

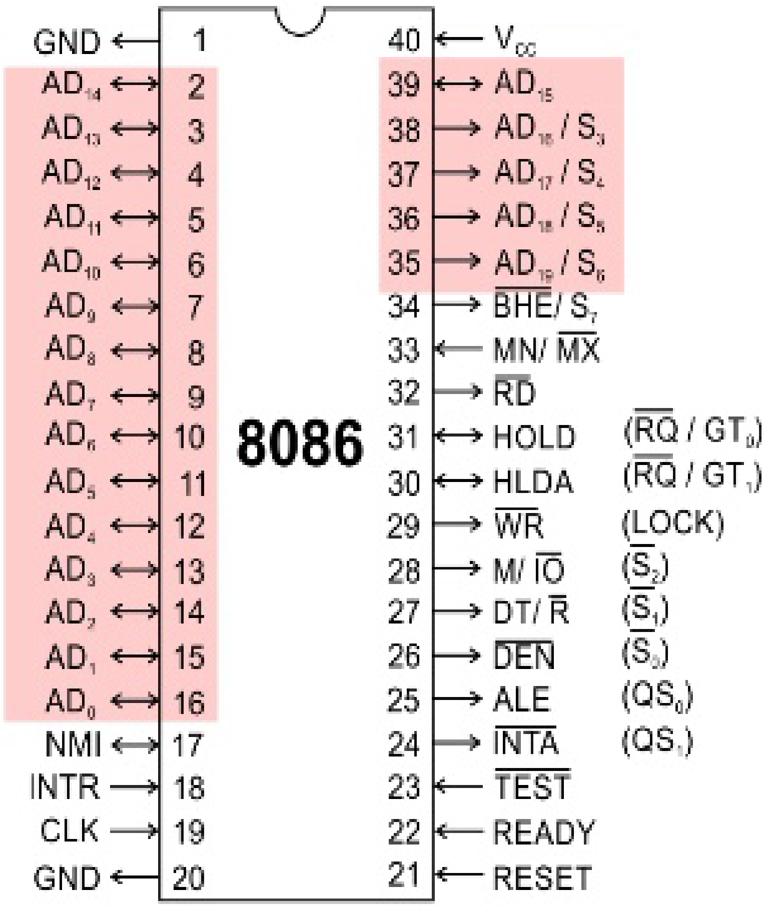


Pin diagram of 8086 Microprocessor

Electronics Desk

Pins and Signals

Common signals



AD_0-AD_{15} (Bidirectional)

Address/Data bus

Low order address bus; these are multiplexed with data.

When AD lines are used to transmit memory address the symbol A is used instead of AD, for example A_0-A_{15}.

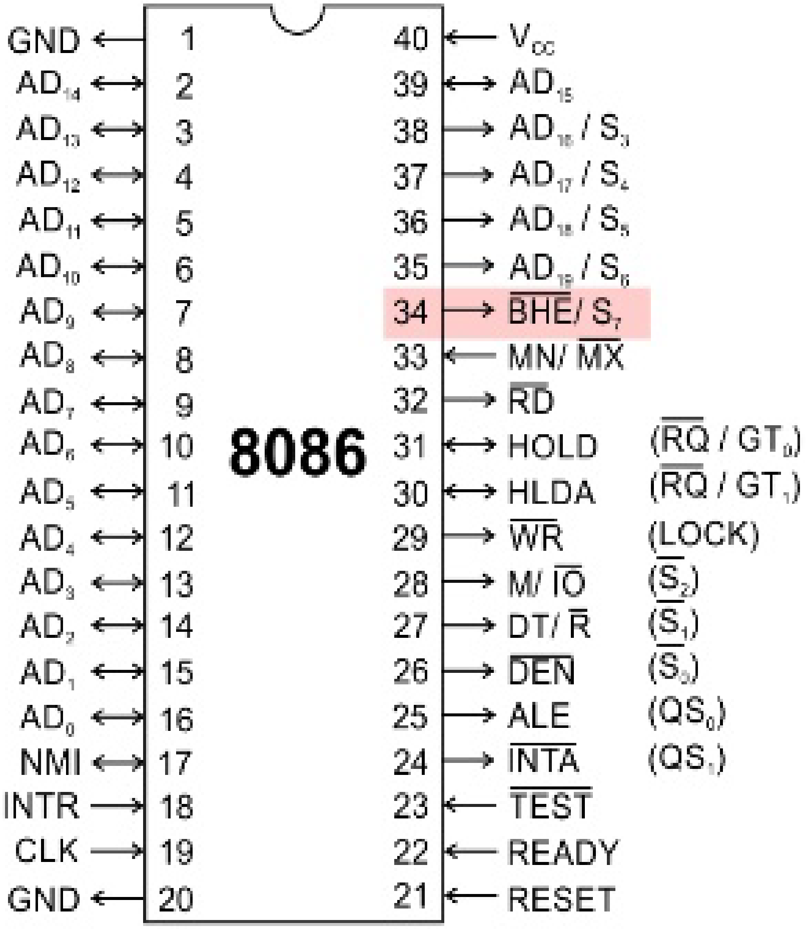
When data are transmitted over AD lines the symbol D is used in place of AD, for example D_0-D_7, D_8-D_{15} or D_0-D_{15}.

A_{16}/S_3, A_{17}/S_4, A_{18}/S_5, A_{19}/S_6

High order address bus. These are multiplexed with status signals

Pins and Signals

Common signals



BHE (Active Low)/S_7 (Output)

Bus High Enable/Status

It is used to enable data onto the most significant half of data bus, D_8-D_{15}. 8-bit device connected to upper half of the data bus use BHE (Active Low) signal. It is multiplexed with status signal S_7.

MN/ MX

MINIMUM / MAXIMUM

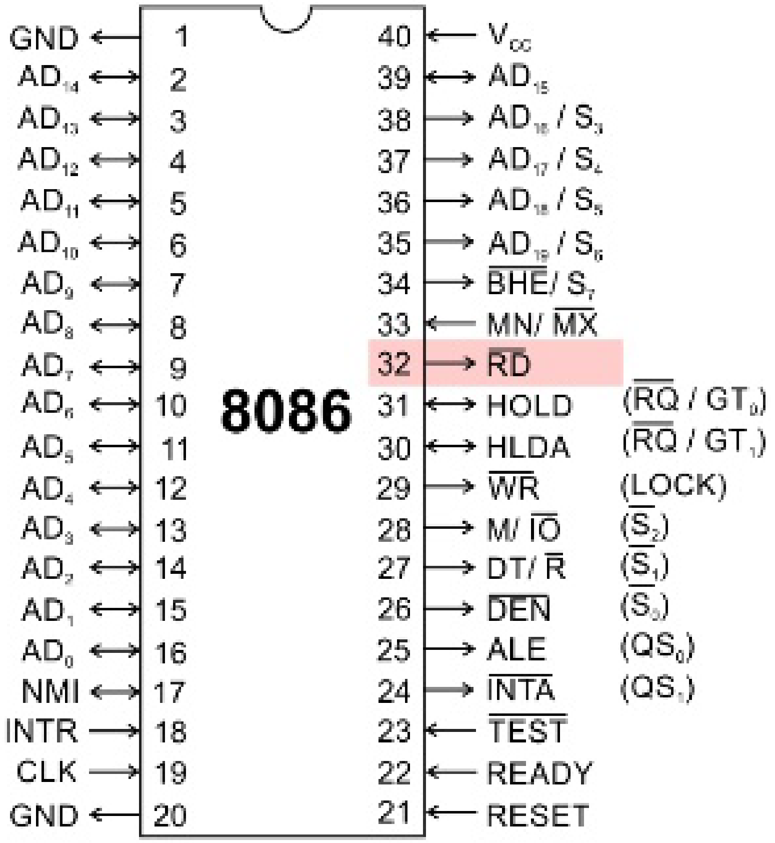
This pin signal indicates what mode the processor is to operate in.

RD (Read) (Active Low)

The signal is used for read operation.
It is an output signal.
It is active when low.

Pins and Signals

Common signals



TEST

$\overline{\text{TEST}}$ input is tested by the 'WAIT' instruction.

8086 will enter a wait state after execution of the WAIT instruction and will resume execution only when the $\overline{\text{TEST}}$ is made low by an active hardware.

This is used to synchronize an external activity to the processor internal operation.

READY

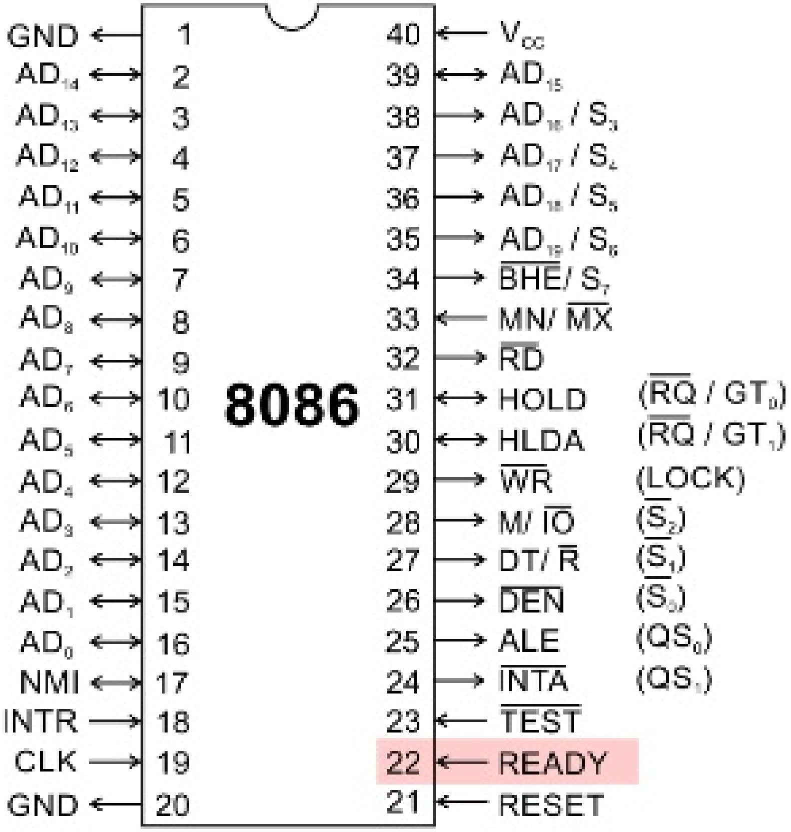
This is the acknowledgement from the slow device or memory that they have completed the data transfer.

The signal made available by the devices is synchronized by the 8284A clock generator to provide ready input to the 8086.

The signal is active high.

Pins and Signals

Common signals



RESET (Input)

Causes the processor to immediately terminate its present activity.

The signal must be active HIGH for at least four clock cycles.

CLK

The clock input provides the basic timing for processor operation and bus control activity. Its an asymmetric square wave with 33% duty cycle.

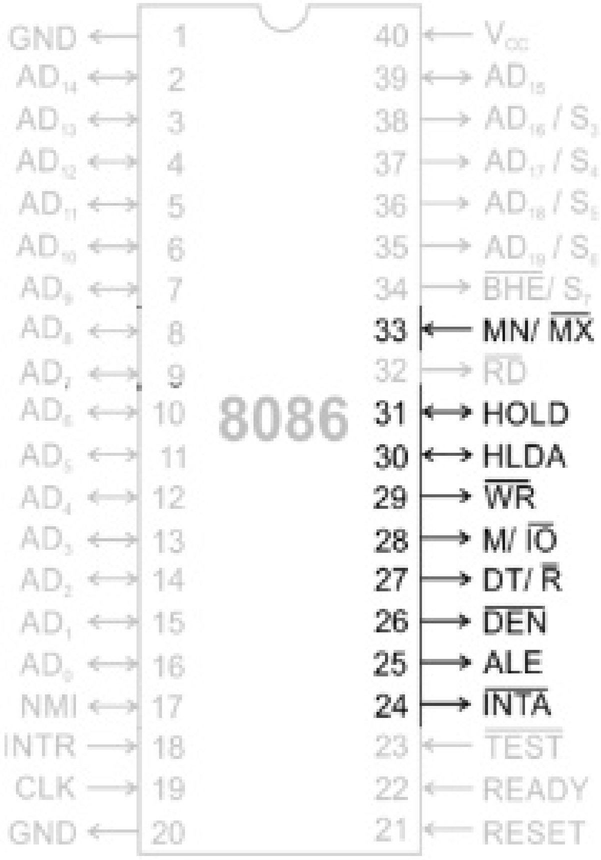
INTR Interrupt Request

This is a triggered input. This is sampled during the last clock cycles of each instruction to determine the availability of the request. If any interrupt request is pending, the processor enters the interrupt acknowledge cycle.

This signal is active high and internally synchronized.

Pins and Signals

Min/ Max Pins



The 8086 microprocessor can work in two modes of operations : Minimum mode and Maximum mode.

In the minimum mode of operation the microprocessor do not associate with any co-processors and can not be used for multiprocessor systems.

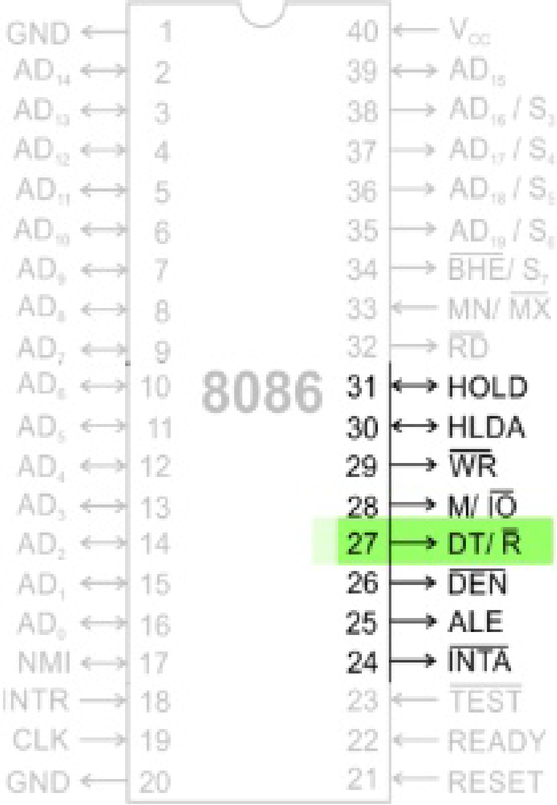
In the maximum mode the 8086 can work in multi-processor or co-processor configuration.

Minimum or maximum mode operations are decided by the pin MN/ MX(Active low).

When this pin is high 8086 operates in minimum mode otherwise it operates in Maximum mode.

Pins and Signals

Minimum mode signals



Pins 24 -31

For minimum mode operation, the MN/ $\overline{\text{MX}}$ is tied to VCC (logic high)

8086 itself generates all the bus control signals

DT/$\overline{\text{R}}$ (**Data Transmit/ Receive**) Output signal from the processor to control the direction of data flow through the data transceivers

$\overline{\text{DEN}}$ (**Data Enable**) Output signal from the processor used as out put enable for the transceivers

ALE (**Address Latch Enable**) Used to demultiplex the address and data lines using external latches

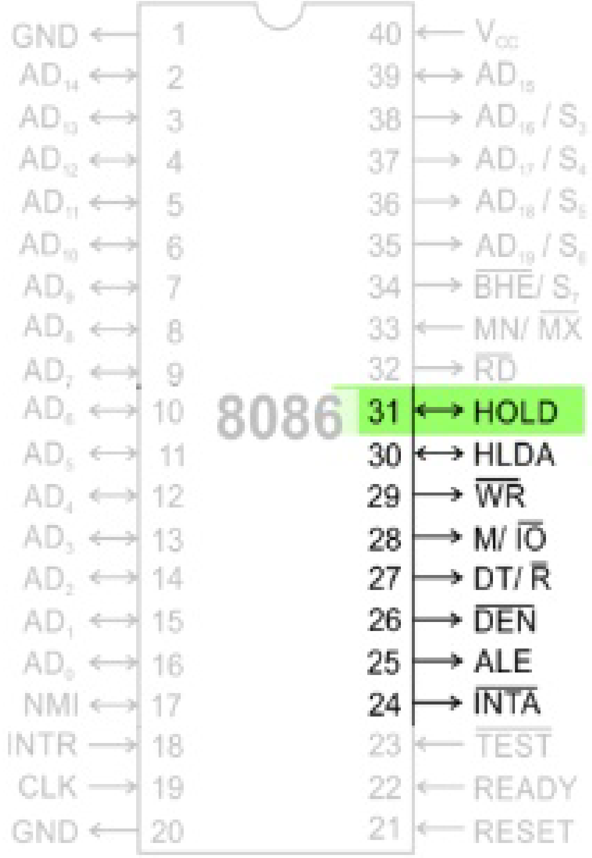
M/$\overline{\text{IO}}$ Used to differentiate memory access and I/O access. For memory reference instructions, it is high. For IN and OUT instructions, it is low.

$\overline{\text{WR}}$ Write control signal; asserted low Whenever processor writes data to memory or I/O port

$\overline{\text{INTA}}$ (**Interrupt Acknowledge**) When the interrupt request is accepted by the processor, the output is low on this line.

Pins and Signals

Minimum mode signals



Pins 24 -31

For minimum mode operation, the MN / $\overline{MX}$ is tied to VCC (logic high)

8086 itself generates all the bus control signals

HOLD **Input signal to the processor form the bus masters as a request to grant the control of the bus.**

Usually used by the DMA controller to get the control of the bus.

HLDA **(Hold Acknowledge) Acknowledge signal by the processor to the bus master requesting the control of the bus through HOLD.**

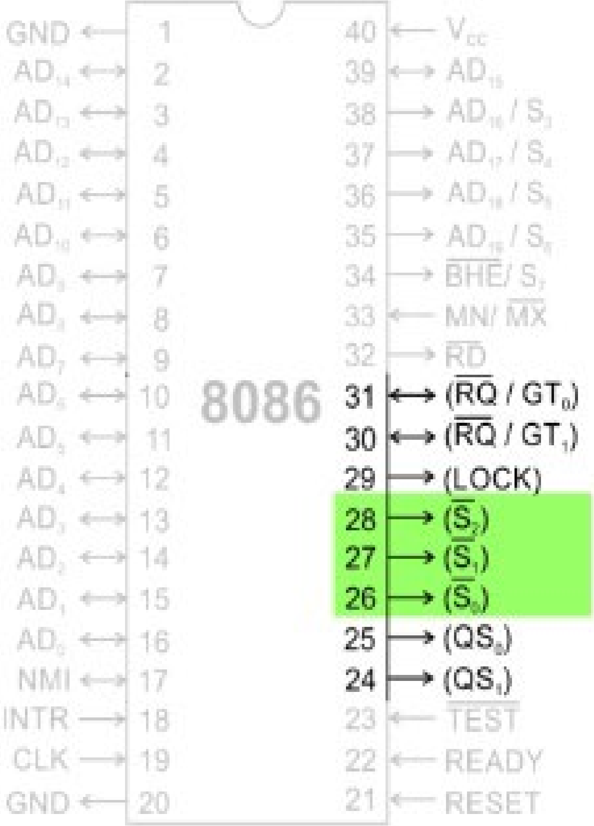
The acknowledge is asserted high, when the processor accepts HOLD.

Pins and Signals

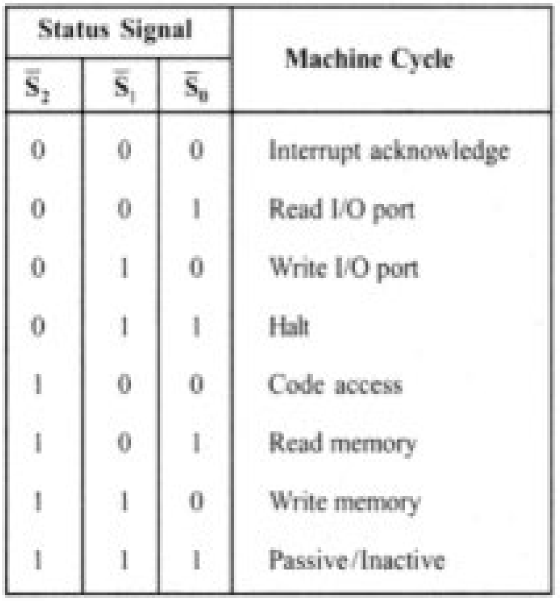
Maximum mode signals

During maximum mode operation, the MN/ $\overline{MX}$ is grounded (logic low)

Pins 24 -31 are reassigned



$\overline{S_0}$, $\overline{S_1}$, $\overline{S_2}$ **Status signals; used by the 8086 bus controller to generate bus timing and control signals. These are decoded as shown.**



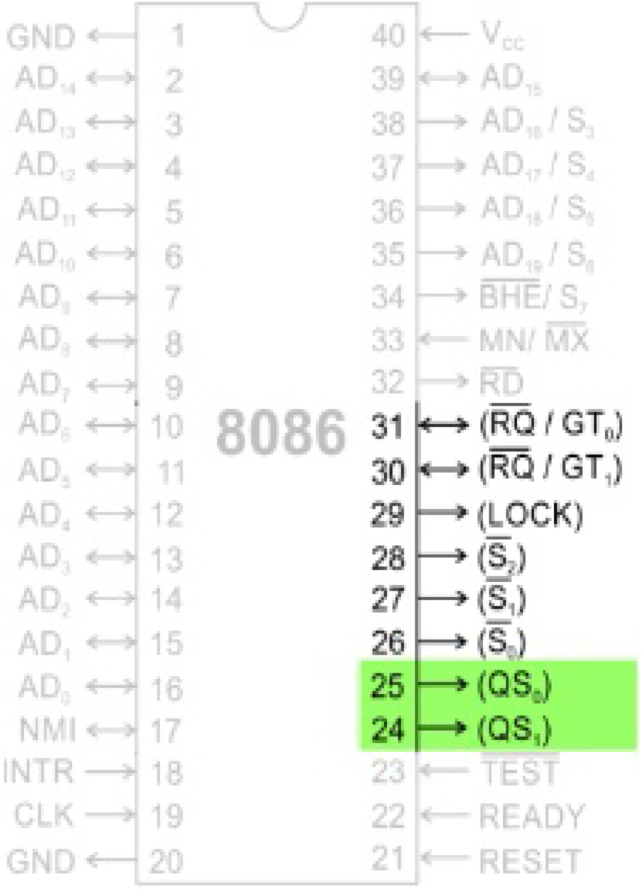
Status Signal			Machine Cycle
$\overline{S}_2$	$\overline{S}_1$	$\overline{S}_0$	
0	0	0	Interrupt acknowledge
0	0	1	Read I/O port
0	1	0	Write I/O port
0	1	1	Halt
1	0	0	Code access
1	0	1	Read memory
1	1	0	Write memory
1	1	1	Passive/Inactive

Pins and Signals

Maximum mode signals

During maximum mode operation, the MN/ $\overline{\text{MX}}$ is grounded (logic low)

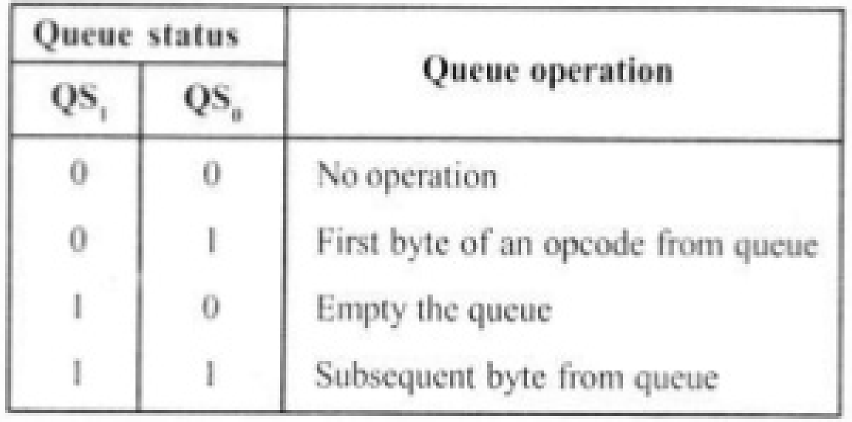
Pins 24 -31 are reassigned



$\overline{QS_0}$, $\overline{QS_1}$ **(Queue Status) The processor provides the status of queue in these lines.**

The queue status can be used by external device to track the internal status of the queue in 8086.

The output on QS_0 and QS_1 can be interpreted as shown in the table.



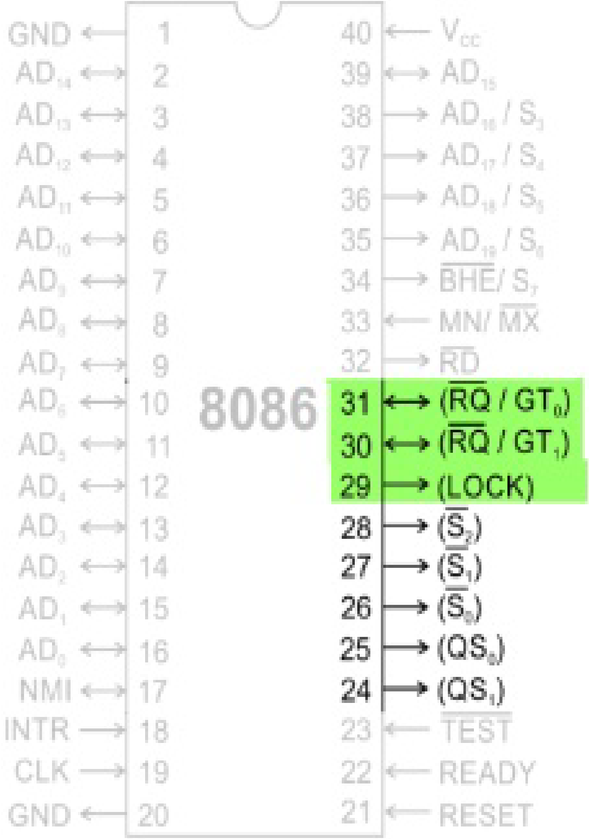
Queue status		Queue operation
QS_1	QS_0	
0	0	No operation
0	1	First byte of an opcode from queue
1	0	Empty the queue
1	1	Subsequent byte from queue

Pins and Signals

Maximum mode signals

During maximum mode operation, the MN/ $\overline{MX}$ is grounded (logic low)

Pins 24 -31 are reassigned



$\overline{RQ}/\overline{GT_0}$, $\overline{RQ}/\overline{GT_1}$

(Bus Request/ Bus Grant) These requests are used by other local bus masters to force the processor to release the local bus at the end of the processor's current bus cycle.

These pins are bidirectional.

The request on $\overline{GT_0}$ will have higher priority than $\overline{GT_1}$

$\overline{LOCK}$

An output signal activated by the LOCK prefix instruction.

Remains active until the completion of the instruction prefixed by LOCK.

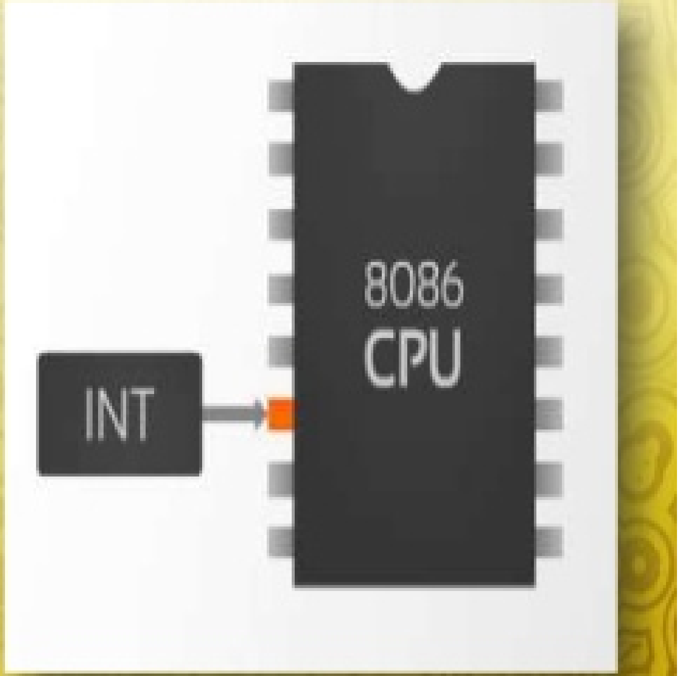
The 8086 output low on the $\overline{LOCK}$ pin while executing an instruction prefixed by LOCK to prevent other bus masters from gaining control of the system bus.

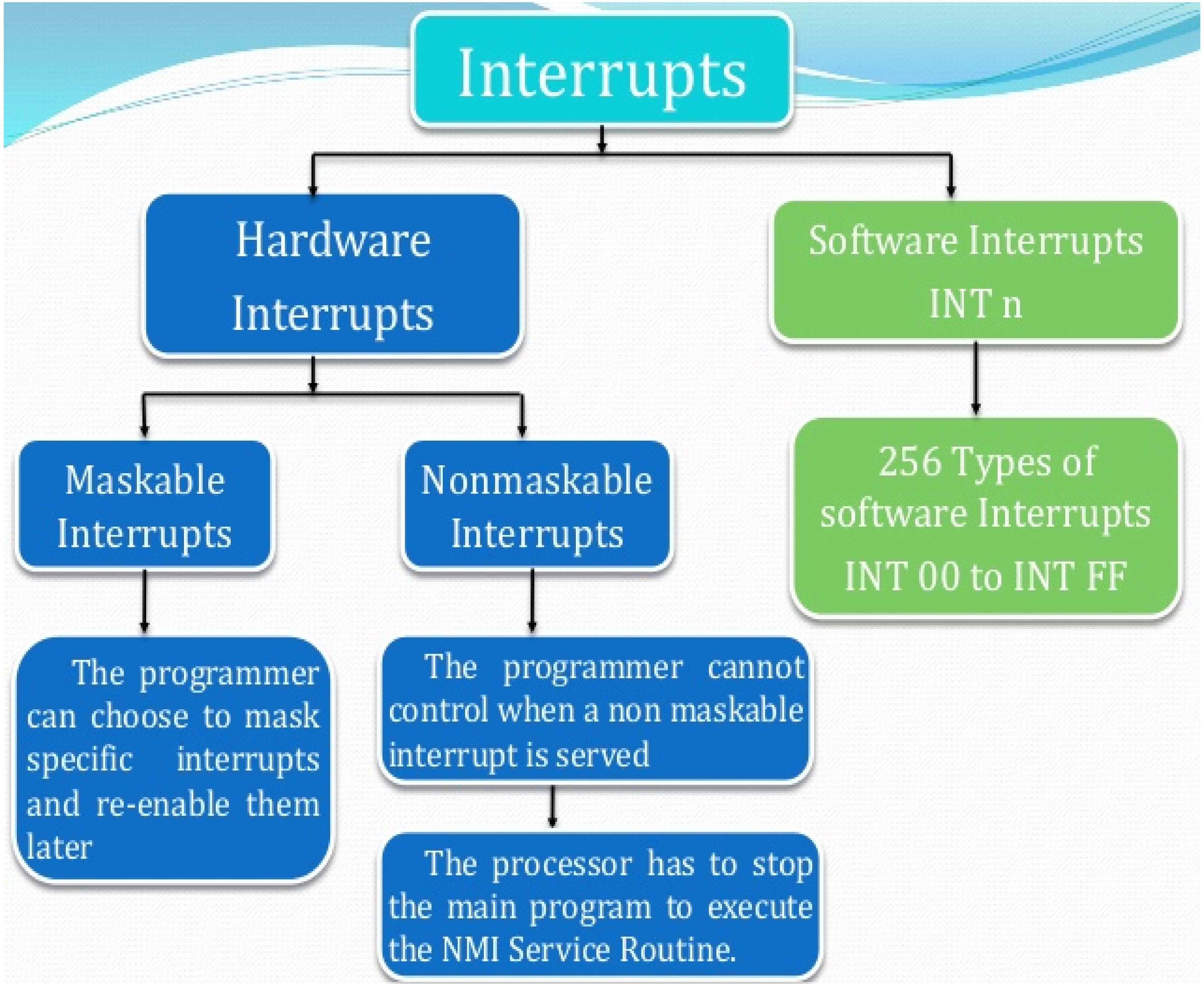
Macro

A **Macro** is a set of instructions grouped under a single unit. It is another method for implementing modular programming in the **8086** microprocessors (The first one was using Procedures). ... The advantage of using **Macro** is that it avoids the overhead time involved in calling and returning (as in the procedures).

Interrupts

- An interrupt is used to cause a temporary halt in the execution of program.
- The meaning of 'interrupts' is to break the sequence of operation.
- While the Microprocessor is executing a program, an 'interrupt' breaks the normal sequence of execution of instructions, diverts its execution to some other program called Interrupt Service Routine (ISR).
- After executing ISR, IRET returns the control back again to the main program. Interrupt processing is an alternative to polling.





Interrupts

Hardware Interrupts

Software Interrupts

INT n

Maskable Interrupts

Nonmaskable Interrupts

256 Types of software Interrupts

INT 00 to INT FF

The programmer can choose to mask specific interrupts and re-enable them later

The programmer cannot control when a non maskable interrupt is served

The processor has to stop the main program to execute the NMI Service Routine.

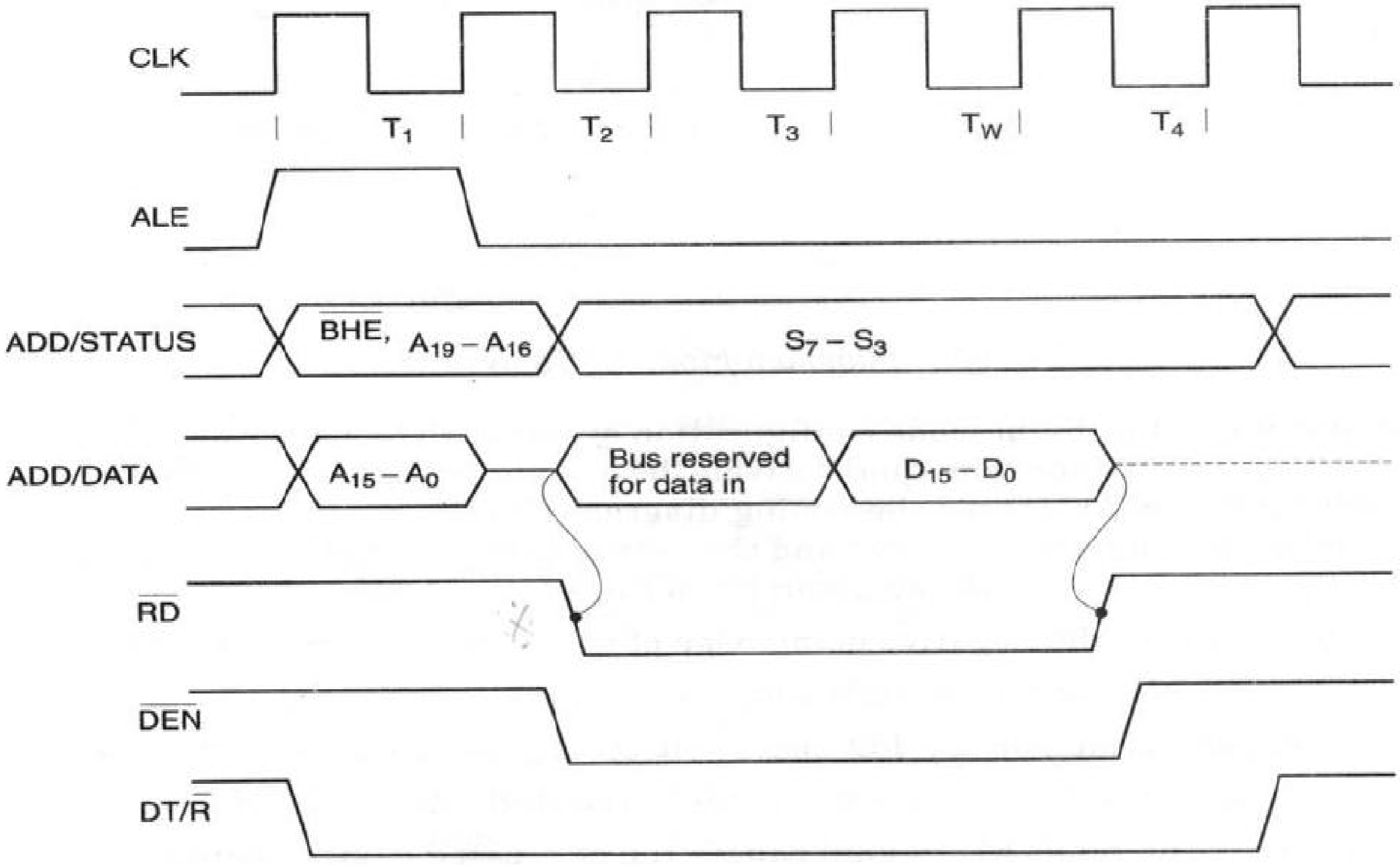
TIMING DIAGRAMS FOR 8086

- The graphical representation of operation of microprocessor with clock cycles is called as timing diagram

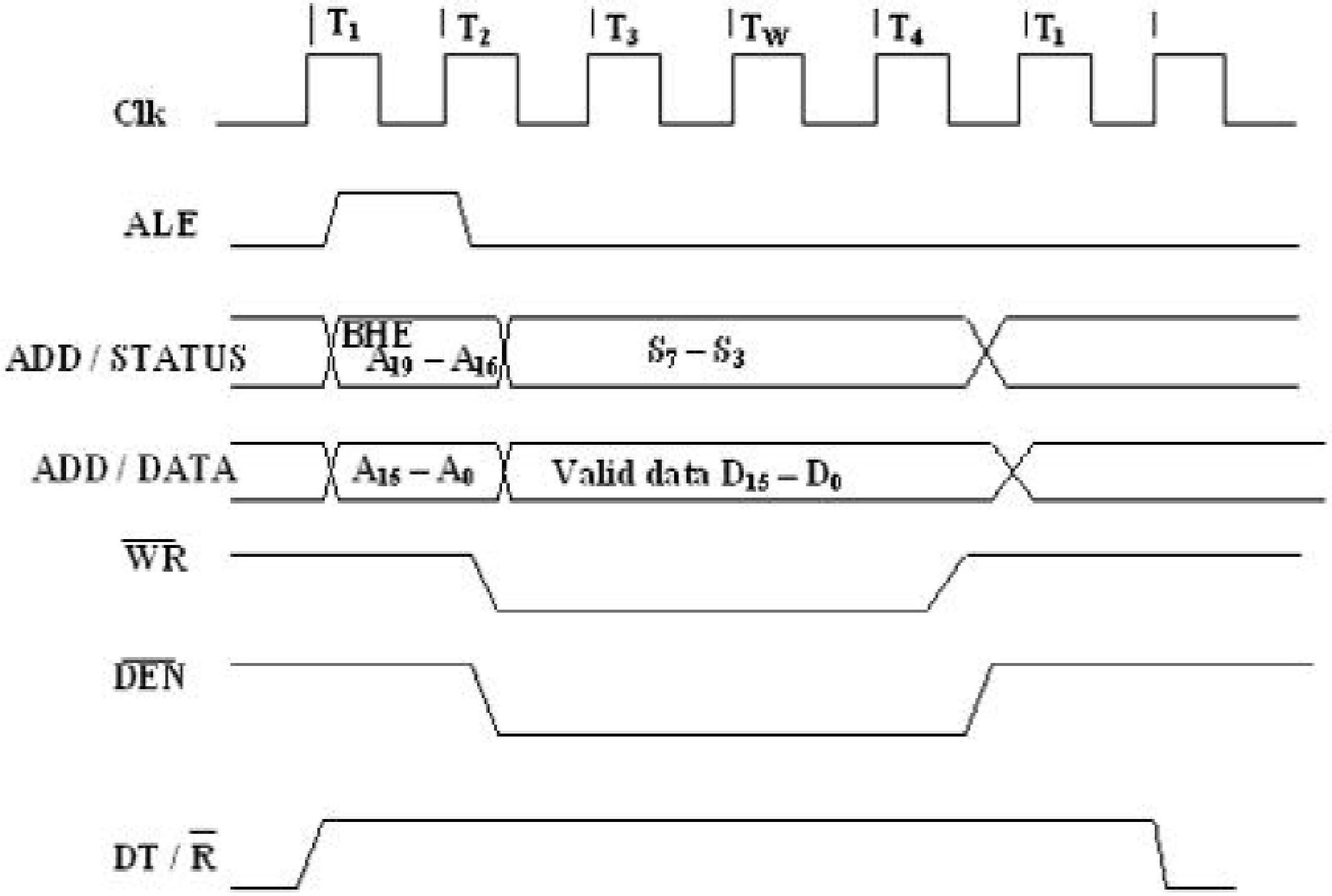
- **TIMING DIAGRAMS FOR 8086 IN MINIMUM MODE BUS CYCLE AND TIME STATES**
- A bus cycle or machine cycle defines the sequence of events when the MPU communicates with an external device, which starts with an address being output on the system bus followed by a read or write data transfer. •
- Types of bus cycles: Memory Read Bus Cycle, Memory Write Bus Cycle Input/output Read Bus Cycle, Input/output Write Bus Cycle.
- One cycle of clock is called a state or t-state. The bus cycle of the 8086 microprocessor consists of at least four clock periods.
- These four time states are called T1, T2, T3 and T4.
- This group of states is called a MACHINE CYCLE.
- The total time required to fetch and execute an instruction is called an instruction cycle. An instruction cycle consists of one or more machine cycle.

- The following figure **shows a memory read cycle of the 8086:**
- • During **period T1,**
- o The 8086 outputs the **20-bit address of the memory location to be accessed on its multiplexed**
- **address/data bus. BHE is also output along with the address during T1.**
- o At the same time a pulse is also produced at **ALE. The trailing edge or the high level of this pulse is used**
- to **latch the address in external circuitry.**
- o Signal **M/IO is set to logic 1 and signal DT/R is set to the 0 logic level and both are maintained**
- throughout all four periods of the bus cycle.
- • Beginning with **period T2,**
- o Status bits **S3 through S6 are output on the upper four address bus lines. This status information is**
- maintained through periods **T3 and T4.**
- o On the other hand, address/data bus lines **AD0 through AD7 are put in the high-Z state during T2.**
- o Late in period **T2, RD is switched to logic 0. This indicates to the memory subsystem that a read cycle is**
- in progress. **DEN is switched to logic 0 to enable external circuitry to allow the data to move from**
- memory onto the microprocessor's data bus.
- • During **period T3,**
- o The memory must provide **valid data during T3 and maintain it until after the processor terminates the**
- read operation. The data read by the 8086 microprocessor can be carried over all **16 data bus lines.**
- • During **T4,**
- o The 8086 switches **RD to the inactive 1 logic level to terminate the read operation. DEN returns to its**
- inactive logic level late during **T4 to disable the external circuitry.**

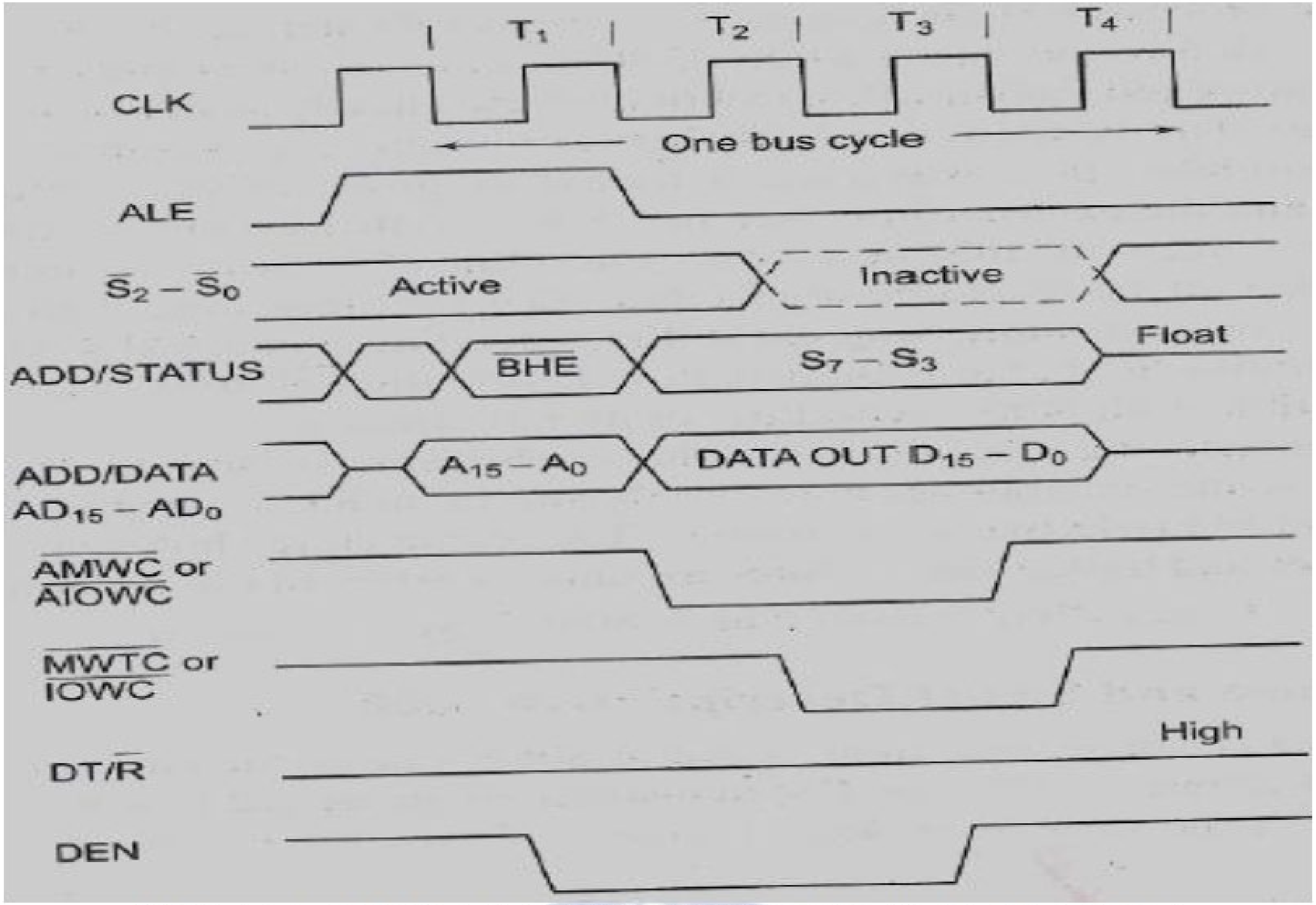
MEMORY READ CYCLE FOR 8086 IN MINIMUM MODE



- The following figure shows a **memory write cycle of the 8086:**
- • During **period T1,**
- o The **address along with BHE is output and latched with the ALE pulse.**
- o **M/IO is set to logic 1 to indicate a memory cycle.**
- o However, this time **DT/R is switched to logic 1. This signals external circuits that the 8086 is going to**
- **transmit data over the bus.**
- • Beginning with **period T2,**
- o **WR is switched to logic 0 telling the memory subsystem that a write operation is to follow.**
- o The 8086 puts the **data on the bus late in T2 and maintains the data valid through T4. Data will be carried**
- over all **16 data bus lines.**
- o **DEN enables the external circuitry to provide a path for data from the processor to the memory**

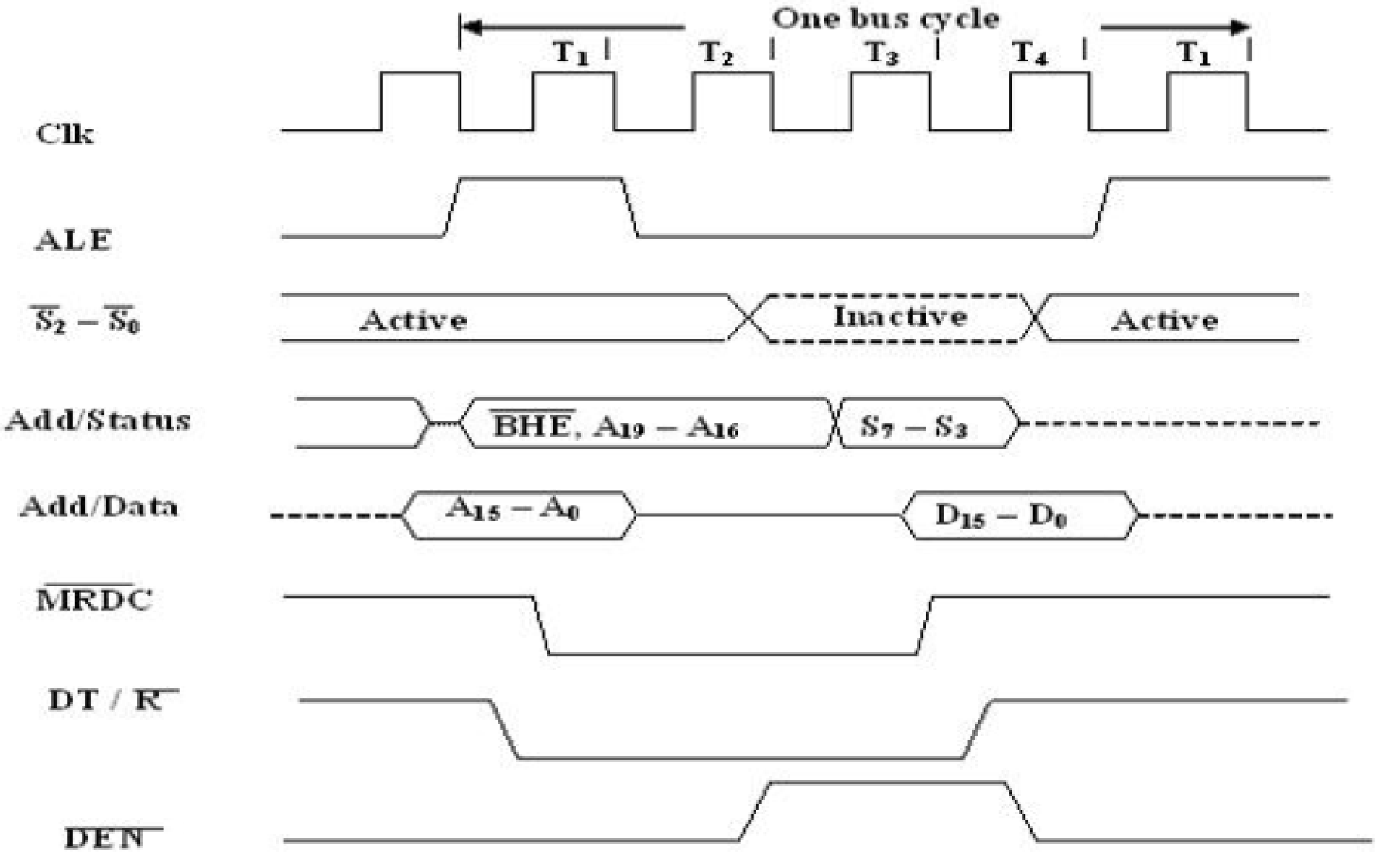


Write Cycle Timing Diagram for Minimum Mode



WRITE CYCLE TIMING DIAGRAM FOR 8086

MAXIMUM MODE TIMING DIGRAMS



Memory Read Timing in Maximum Mode

I/O INTERFACE

- 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 based system using keyboard and user can see the result or output information from the microprocessor based system with the help of display device. The transfer of data between keyboard and microprocessor, and microprocessor and display device is called input/output data transfer or I/O data transfer. This data transfer is done with the help of I/O port
- The generation of path between two devices to flow data,or Interconnection of two devices is called interfacing.

Input port:

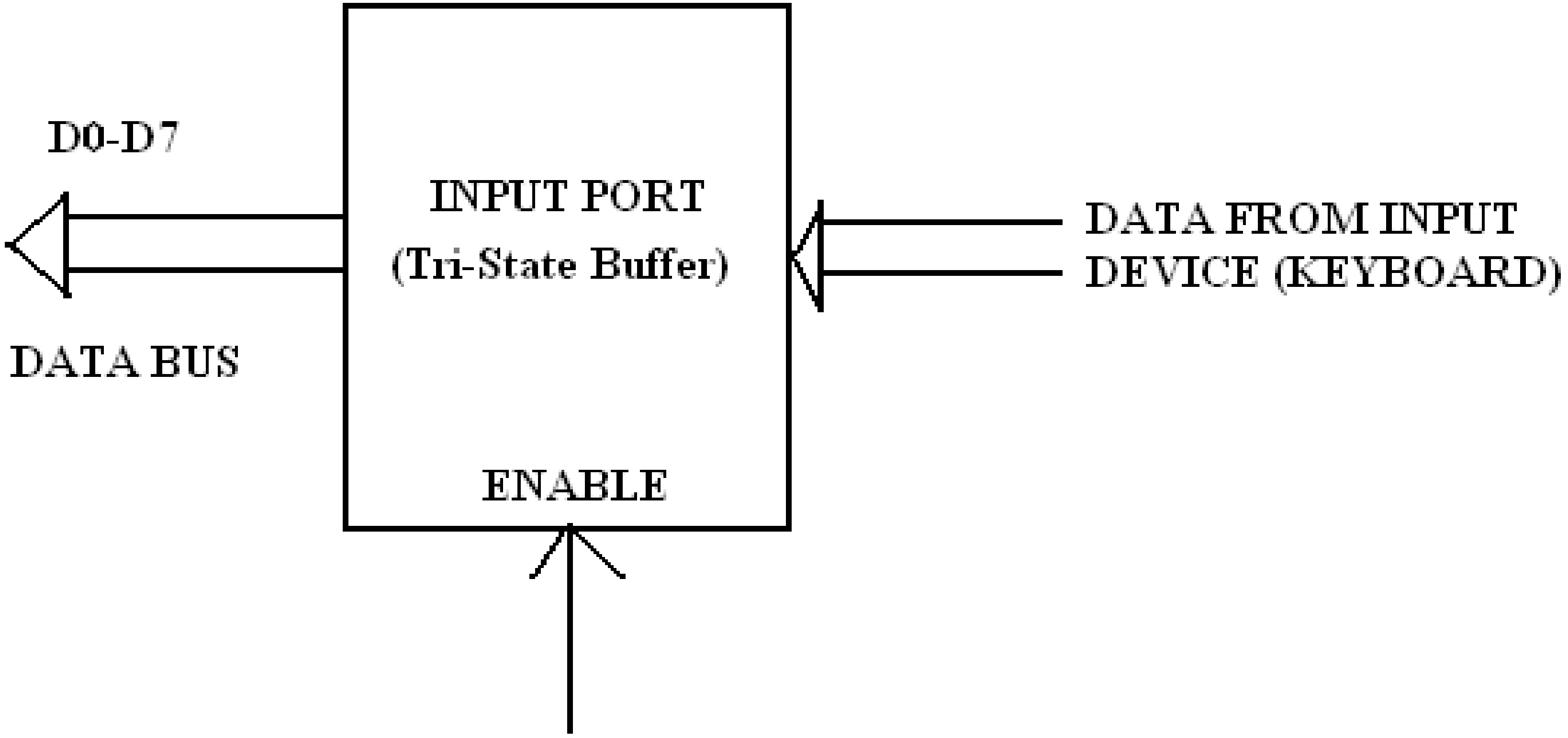


FIG.1 INPUT PORT

- 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.1. This buffer is a tri-state buffer and its output is available only when enable signal is active. When microprocessor wants to read data from the input device (keyboard), the control signals from the microprocessor activates the buffer by asserting enable input of the buffer. Once the buffer is enabled, data from the input device is available on the data bus. Microprocessor reads this data by initiating read command

Output port:

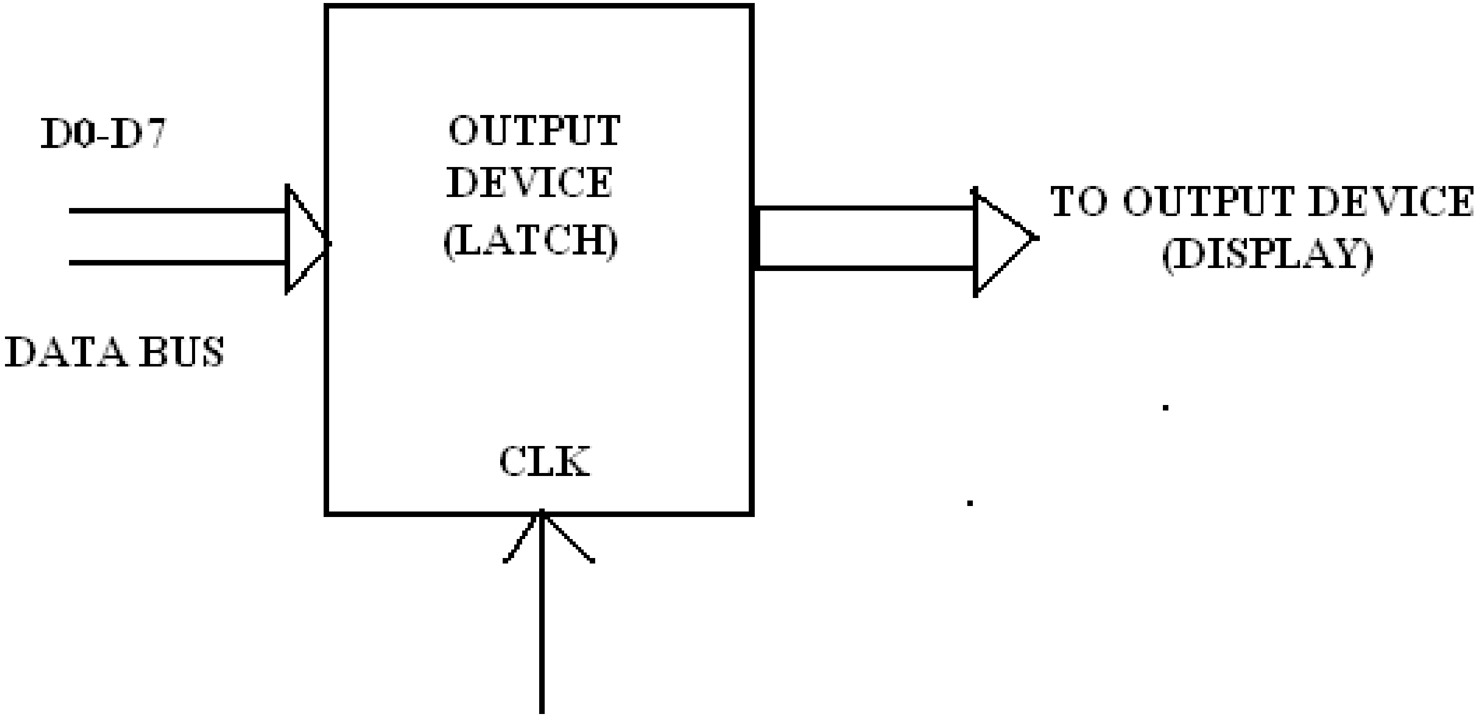


FIG.2 OUTPUT PORT

- 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.2. When microprocessor wants to send data to the output device is 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.

Interfacing Analog to Digital Data Converters

- In most of the cases, the PIO 8255 is used for interfacing the analog to digital converters with microprocessor.
- □We have already studied 8255 interfacing with 8086 as an I/O port, in previous section. This section we will only emphasize the interfacing techniques of analog to digital converters with 8255.
- □The analog to digital converters is treated as an input device by the microprocessor that sends an initializing signal to the ADC to start the analogy to digital data conversation process. The start of conversation signal is a pulse of a specific duration.
- □The process of analog to digital conversion is a slow
- □Process and the microprocessor have to wait for the digital data till the conversion is over. After the conversion is over, the ADC sends end of conversion EOC signal to inform the microprocessor that the conversion is over and the result is ready at the output buffer of the ADC. The set asks of issuing an SOC pulse to ADC, reading EOC signal from the ADC and reading the digital output of the ADC are carried out by the CPU using 8255 I/O ports.

- The time taken by the ADC from the active edge of SOC pulse till the active edge of EOC signal is called as the conversion delay of the ADC.
- □It may range anywhere from a few microseconds in caseof fast ADC to even a few hundred milliseconds in case of slow ADCs.
- □The available ADC in the market use different conversion techniques for conversion of analog signal to digitals. Successive approximation techniques and dual slope integration techniques are the most popular techniques used in the integrated ADC chip.
- □General algorithm for ADC interfacing contains the following steps:
- □Ensure the stability of analog input, applied to the ADC.
- □Issue start of conversion pulse to ADC
- □Read end of conversion signal to mark the end of conversion processes.
- □Read digital data output of the ADC as equivalent digital output.
- □Analog input voltage must be constant at the input of the ADC right from the start of conversion till the end of the conversion to get correct results. This may be ensured by as ample and hold circuit which samples the analog signal and holds it constant for specific time duration. The microprocessor may issue a hold signal to the sample and hold circuit.
- □If the applied input changes before the complete conversion process is over, the digital equivalent of the analog input calculated by the ADC may not be correct.

ADC 0808/0809:

- The analog to digital converter chips 0808 and 0809 are 8-bit CMOS, successive approximation converters. This technique is one of the fast techniques for analog to digital conversion. The conversion delay is 100µs at a clock frequency of 640 KHz, which is quite low as compared to other converters. These converters do not need any external zero or full scale adjustments as they are already taken care of by internal circuits.
- □These converters internally have a 3:8 analog multiplexer so that at a time eight different analog conversion by using address lines - ADD A, ADD B, ADD C, as shown. Using these address inputs, multichannel data

Fig (1) and Fig (2) show the block diagrams and pin diagrams for ADC 0808/0809.

Fig (1) and Fig (2) show the block diagrams and pin diagrams for ADC 0808/0809

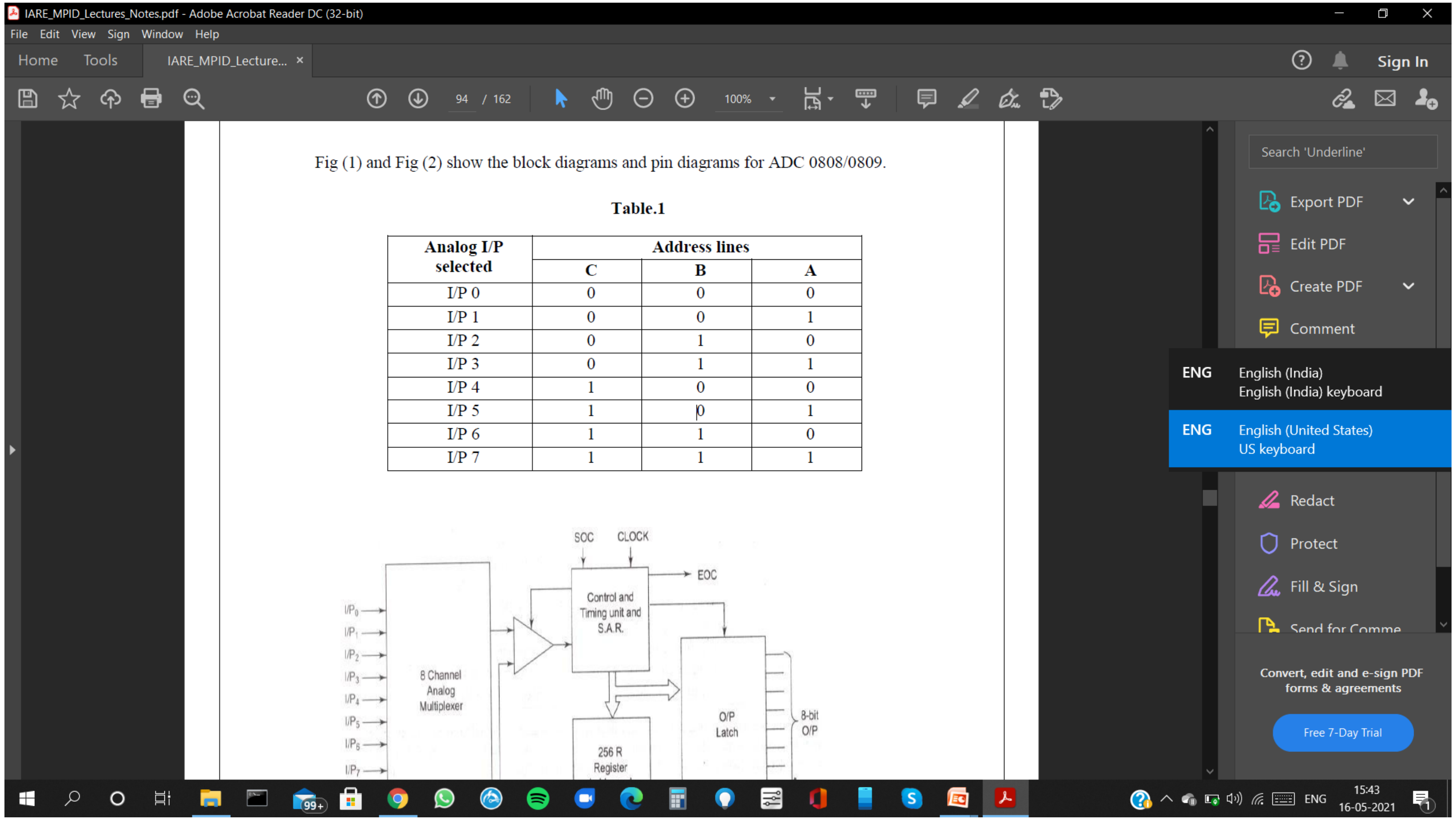

IARE_MPID_Lectures_Notes.pdf - Adobe Acrobat Reader DC (32-bit)
File Edit View Sign Window Help
Home Tools IARE_MPID_Lecture... ×
Sign In
94 / 162
100%
Search 'Underline'
Export PDF
Edit PDF
Create PDF
Comment
ENG English (India) English (India) keyboard
ENG English (United States) US keyboard
Redact
Protect
Fill & Sign
Send for Comme
Convert, edit and e-sign PDF forms & agreements
Free 7-Day Trial
ENG
15:43
16-05-2021
SOC
CLOCK
EOC
Control and Timing unit and S.A.R.
I/P0
I/P1
I/P2
I/P3
I/P4
I/P5
I/P6
I/P7
8 Channel Analog Multiplexer
O/P Latch
8-bit O/P
256 R Register

Fig (1) and Fig (2) show the block diagrams and pin diagrams for ADC 0808/0809.

Table.1

Analog I/P selected	Address lines		
	C	B	A
I/P 0	0	0	0
I/P 1	0	0	1
I/P 2	0	1	0
I/P 3	0	1	1
I/P 4	1	0	0
I/P 5	1	0	1
I/P 6	1	1	0
I/P 7	1	1	1

Fig.1 Block Diagram of ADC 0808/0809

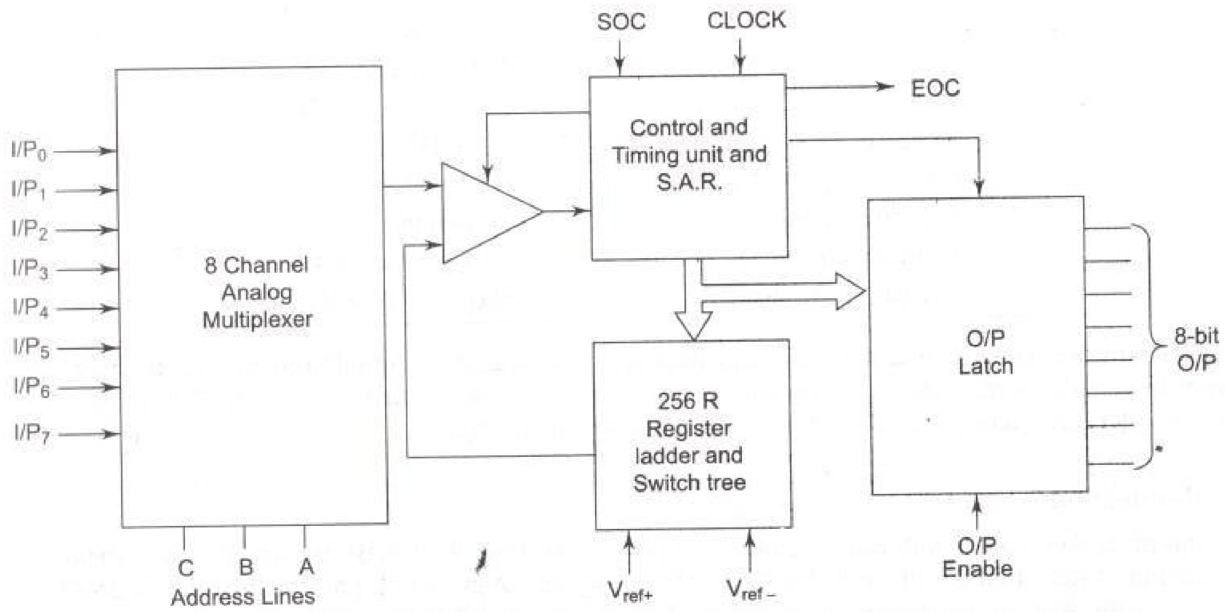
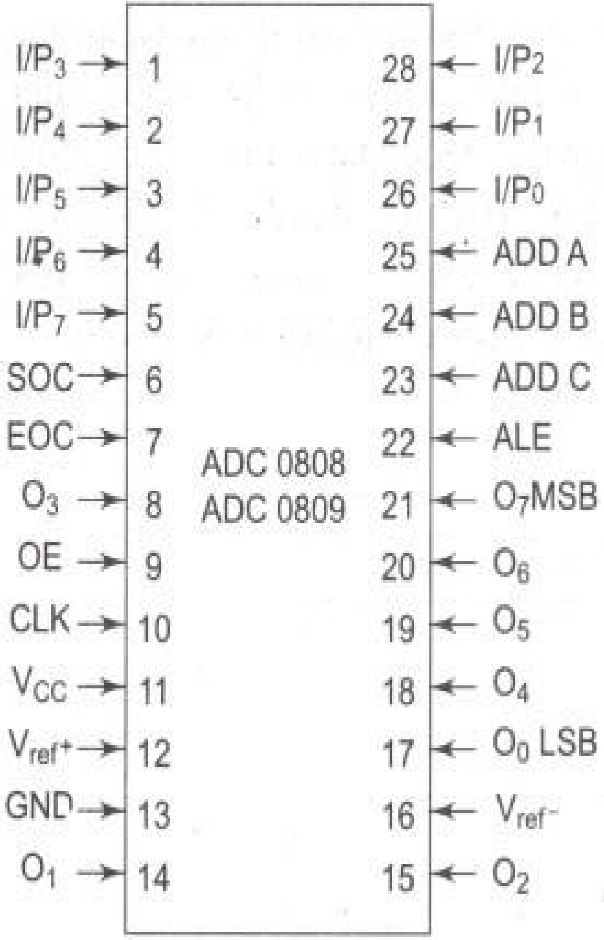

SOC
CLOCK
EOC
Control and Timing unit and S.A.R.
I/P0
I/P1
I/P2
I/P3
I/P4
I/P5
I/P6
I/P7
8 Channel Analog Multiplexer
O/P Latch
8-bit O/P
256 R Register ladder and Switch tree
C
B
A
Address Lines
Vref+
Vref –
O/P Enable

Fig.2 Pin Diagram of ADC 0808/0809



I/P_0 - I/P_7	Analog inputs
ADD A, B, C	Address lines for selecting analog inputs
O_7 - O_0	Digital 8-bit output with O_7 MSB and O_0 LSB
SOC	Start of conversion signal pin
EOC	End of conversion signal pin
OE	Output latch enable pin, if high enable output
CLK	Clock input for ADC
V_{CC}, GND	Supply pins +5V and GND
V_{ref+} and V_{ref-}	Reference voltage positive (+5 Volts maximum) and Reference voltage negative (0V minimum)

Interfacing ADC0808 with 8086 Interfacing Digital

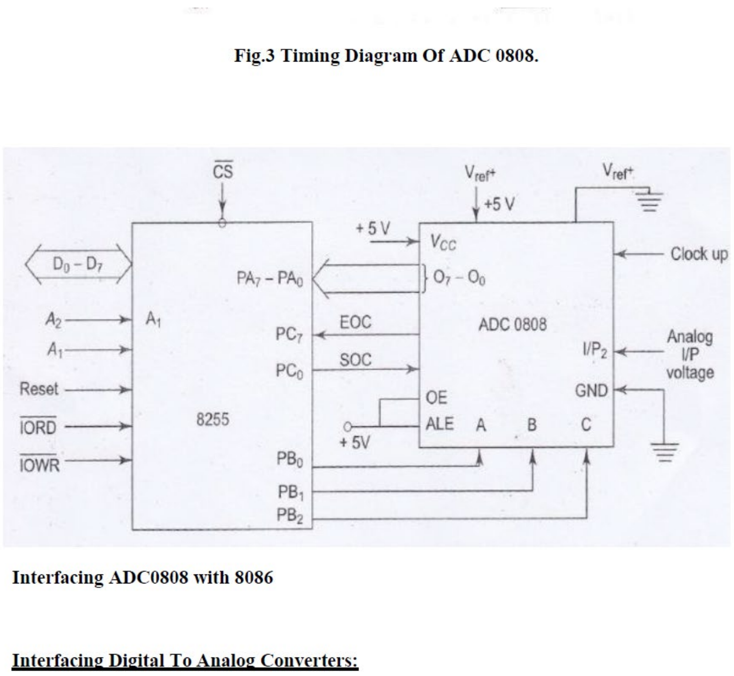


Fig.3 Timing Diagram Of ADC 0808.

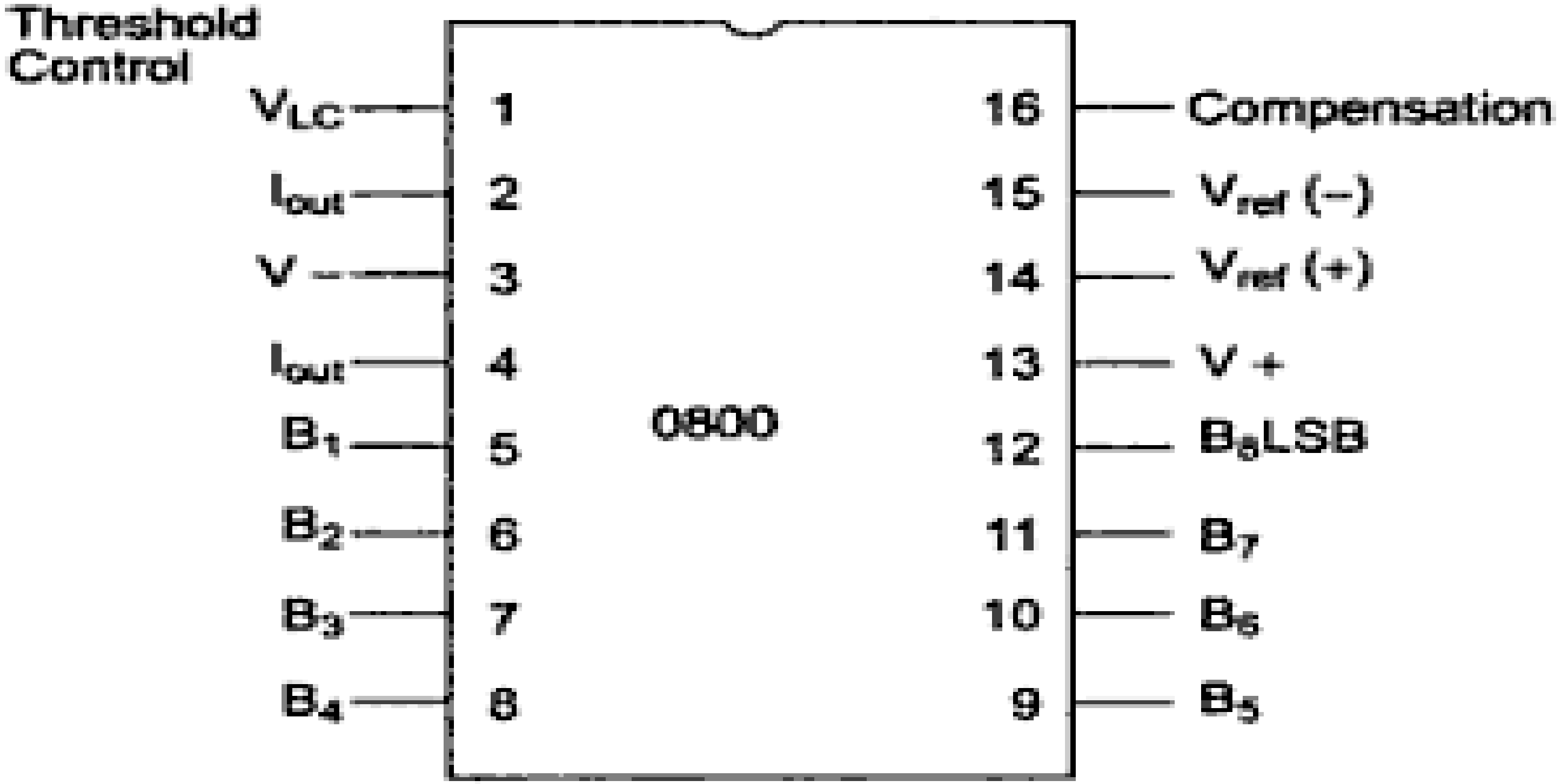
Interfacing ADC0808 with 8086

Interfacing Digital To Analog Converters:

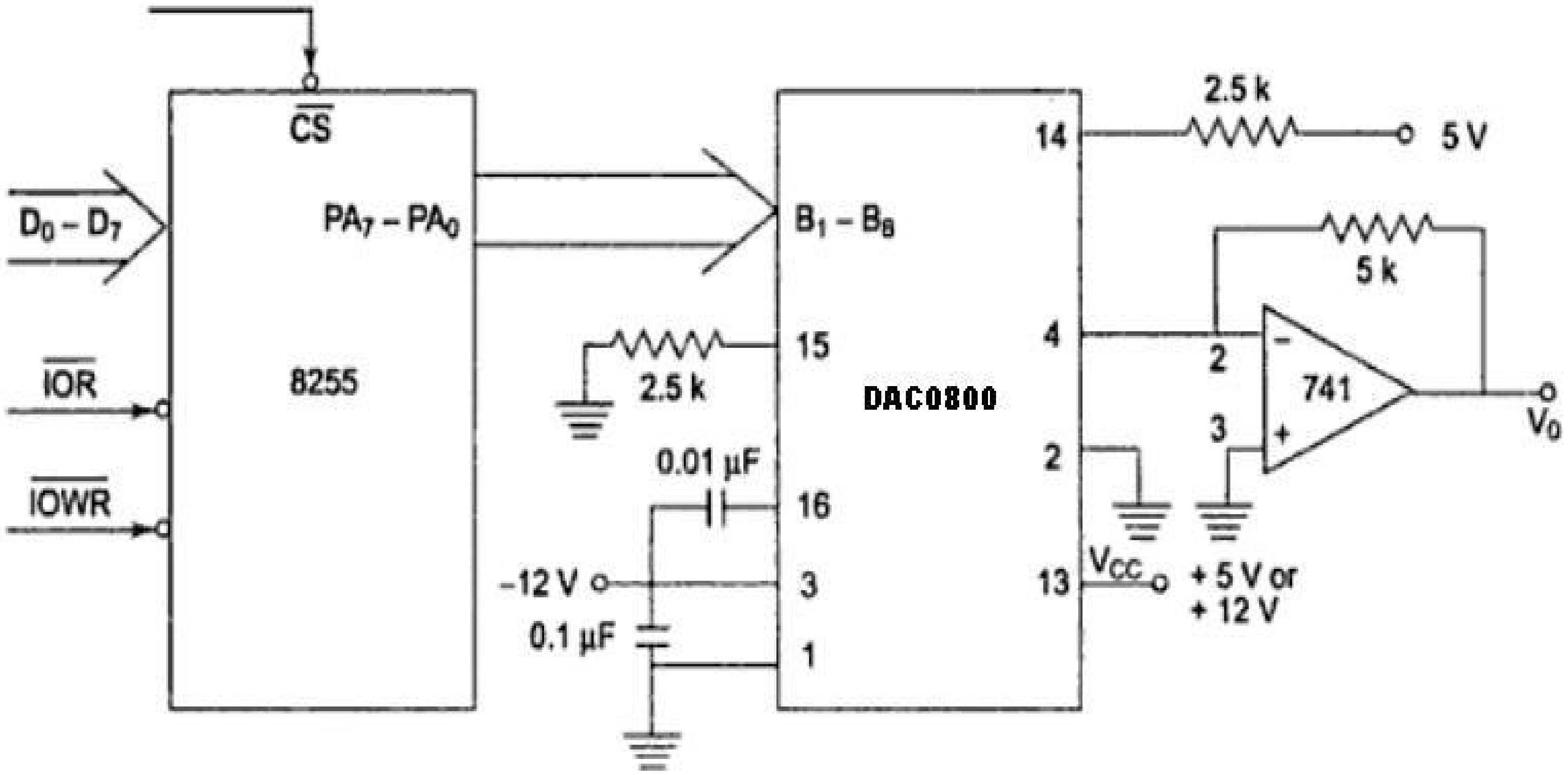
Interfacing Digital To Analog Converters:
The

- The digital to analog converters convert binary numbers into their analog equivalent voltages. The DAC find applications in areas like digitally controlled gains, motor speed controls, programmable gain amplifiers, etc.
- **DAC0800 8-bit Digital to Analog Converter**
- □The DAC 0800 is a monolithic 8-bit DAC manufactured by National Semiconductor.
- □It has settling time around 100ms and can operate on a range of power supply voltages i.e. from 4.5V to +18V.
- Usually the supply V+ is 5V or +12V.
- □The V-pin can be kept at a minimum of -12V

Pin Diagram of DAC 0800
Interfacing DAC0800



Interfacing DAC0800 with 8086 Ad 7523 8-Bit Multiplying DAC:



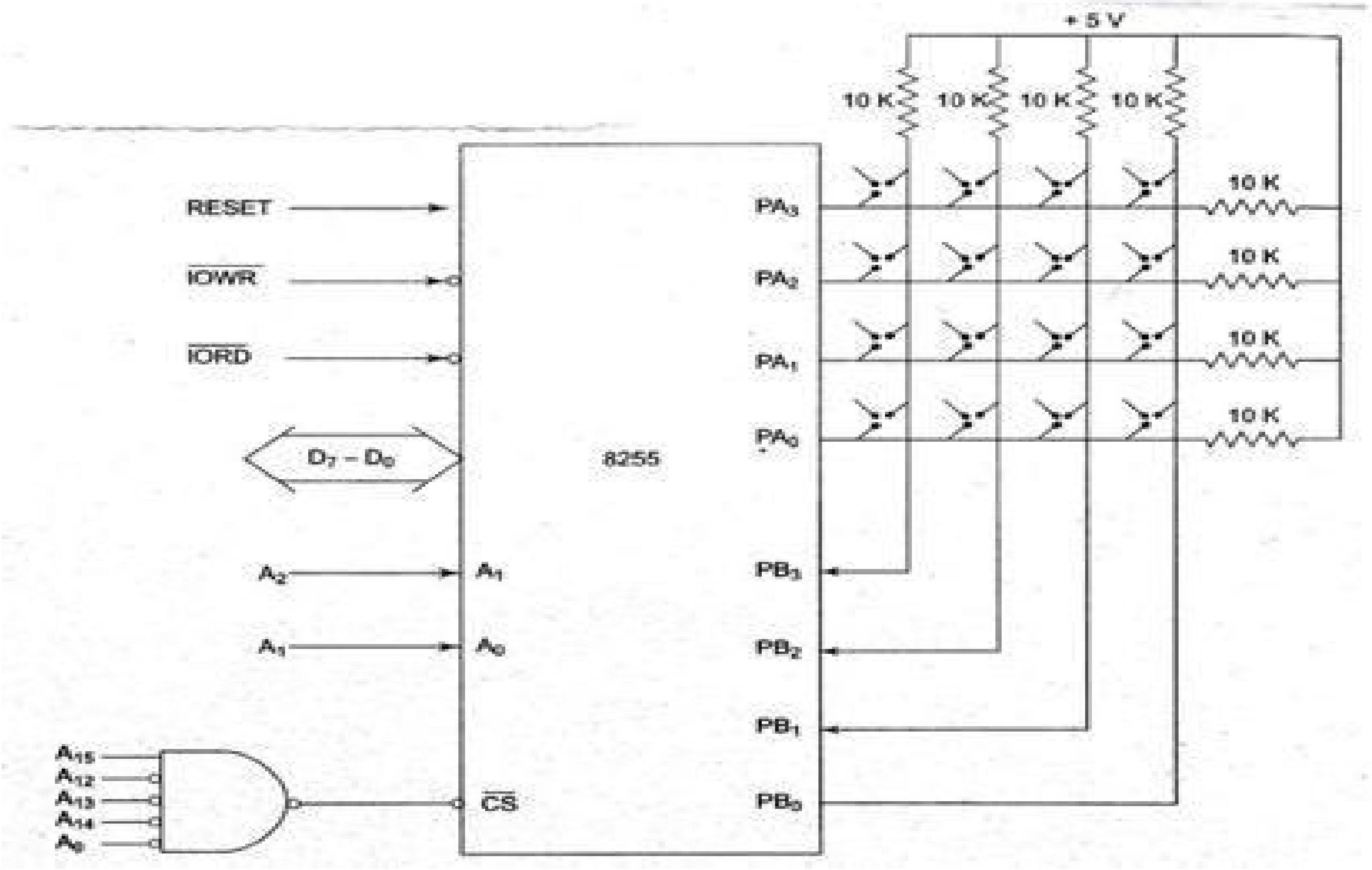
Keyboard Interfacing □

- In most keyboards, the key switches are connected in a matrix of Rows and Columns.
- □Getting meaningful data from a keyboard requires three major tasks:
- 1. etect a keypress
- 2. Debounce the keypress.
- 3. Encode the keypress (produce a standard code for the pressed key).
- □Logic „0‟ is read by the microprocessor when the key is pressed.
- **Key Debounce:**
- Whenever a mechanical push-bottom is pressed or released once,the mechanical components of the key do not change the positionsmoothly; rather it generates a transient response.
- These may be interpreted as the multiple pressures and responded accordingly.

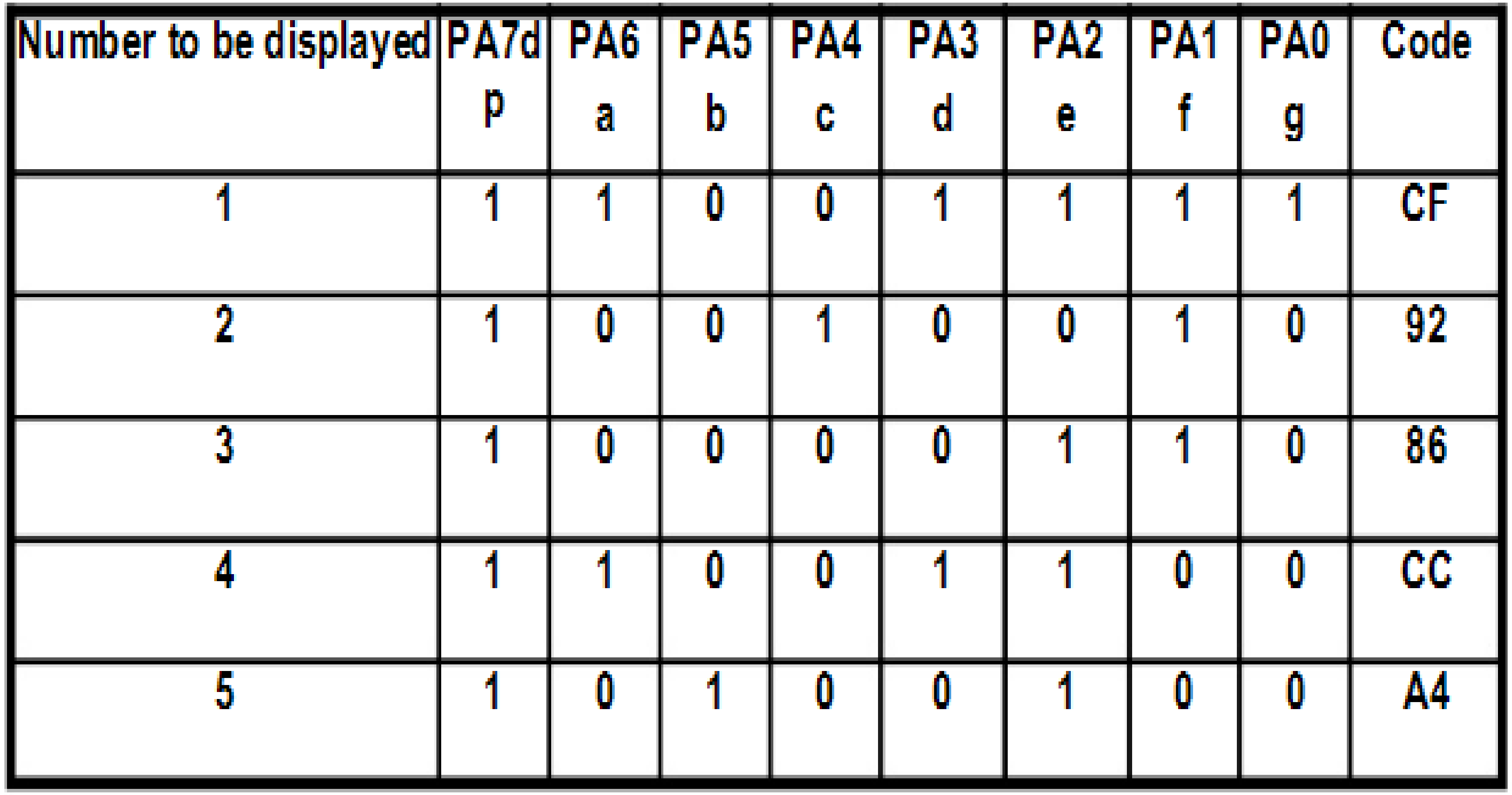
- The rows of the matrix are connected to four output Port lines, &columns are connected to four input Port lines.
- When no keys are pressed, the column lines are held high by the pull-up resistors connected to +5v.
- Pressing a key connects a row & a column.
- To detect if any key is pressed is to output 0‟s to all rows & then check columns to see it a pressed key has connected a low (zero) to a column.
- Once the columns are found to be all high, the program enters another loop, which waits until a low appears on one of the columns i.e indicating a key press.
- A simple 20/10 m sec delay is executed to debounce task.
- After the debounce time, another check is made to see if the key is still pressed. If the columns are now all high, then no key is pressed & the initial detection was caused by a noise pulse.
- To avoid this problem, two schemes are suggested:
- 1. Use of Bistable multivibrator at the output of the key to debounce it.
- 2. The microprocessor has to wait for the transient period (at least for 10 ms), so that the transient response settles down and reaches a steady state.
- If any of the columns are low now, then the assumption is made that it was a valid key press.

- The final task is to determine the row & column of the pressed key &convert this information to Hex-code for the pressed key.
- The 4-bit code from I/P port & the 4-bit code from O/P port (row &column) are converted to Hex-code.

INTERFACING 4×4 KEYBOARD

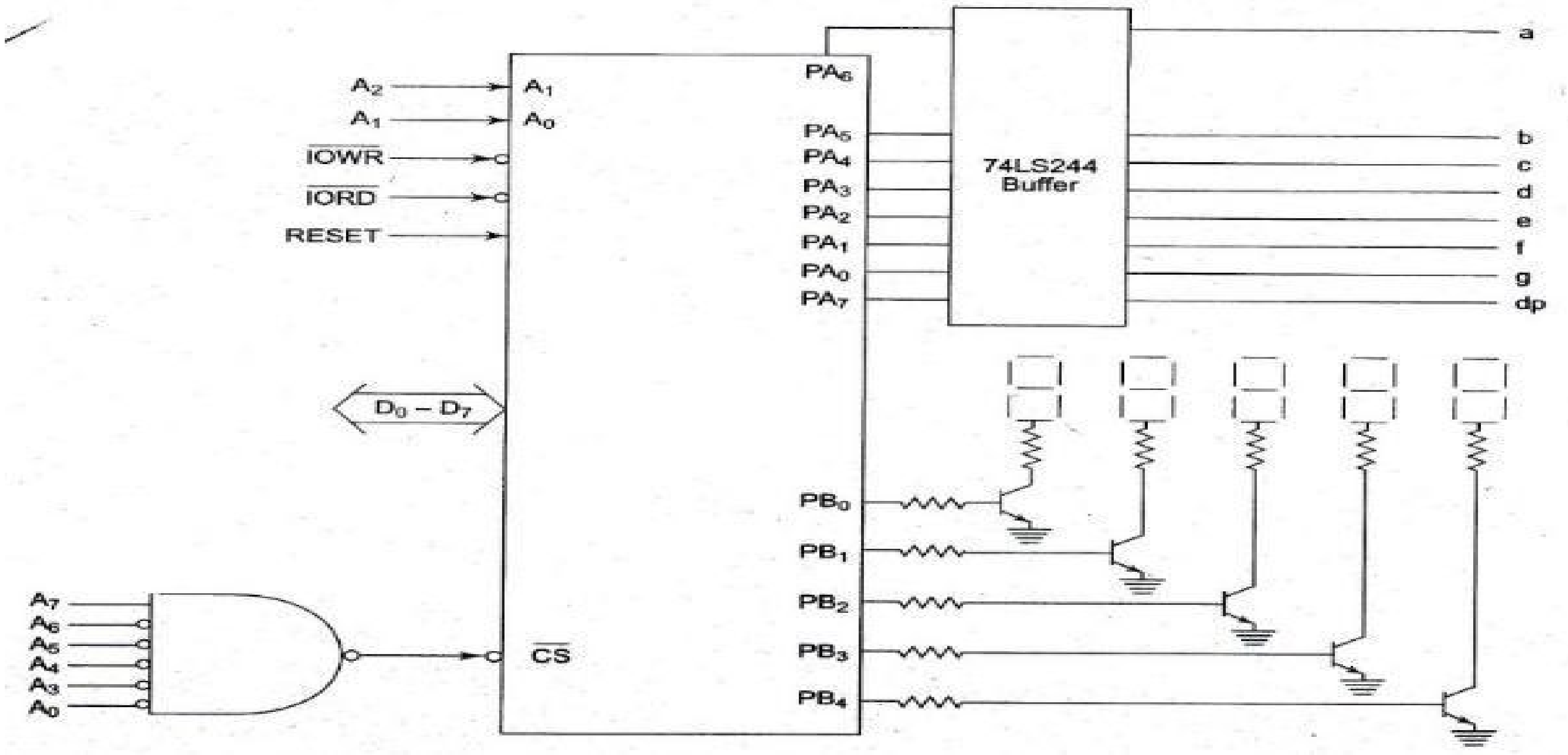


DISPLAY INTERFACE

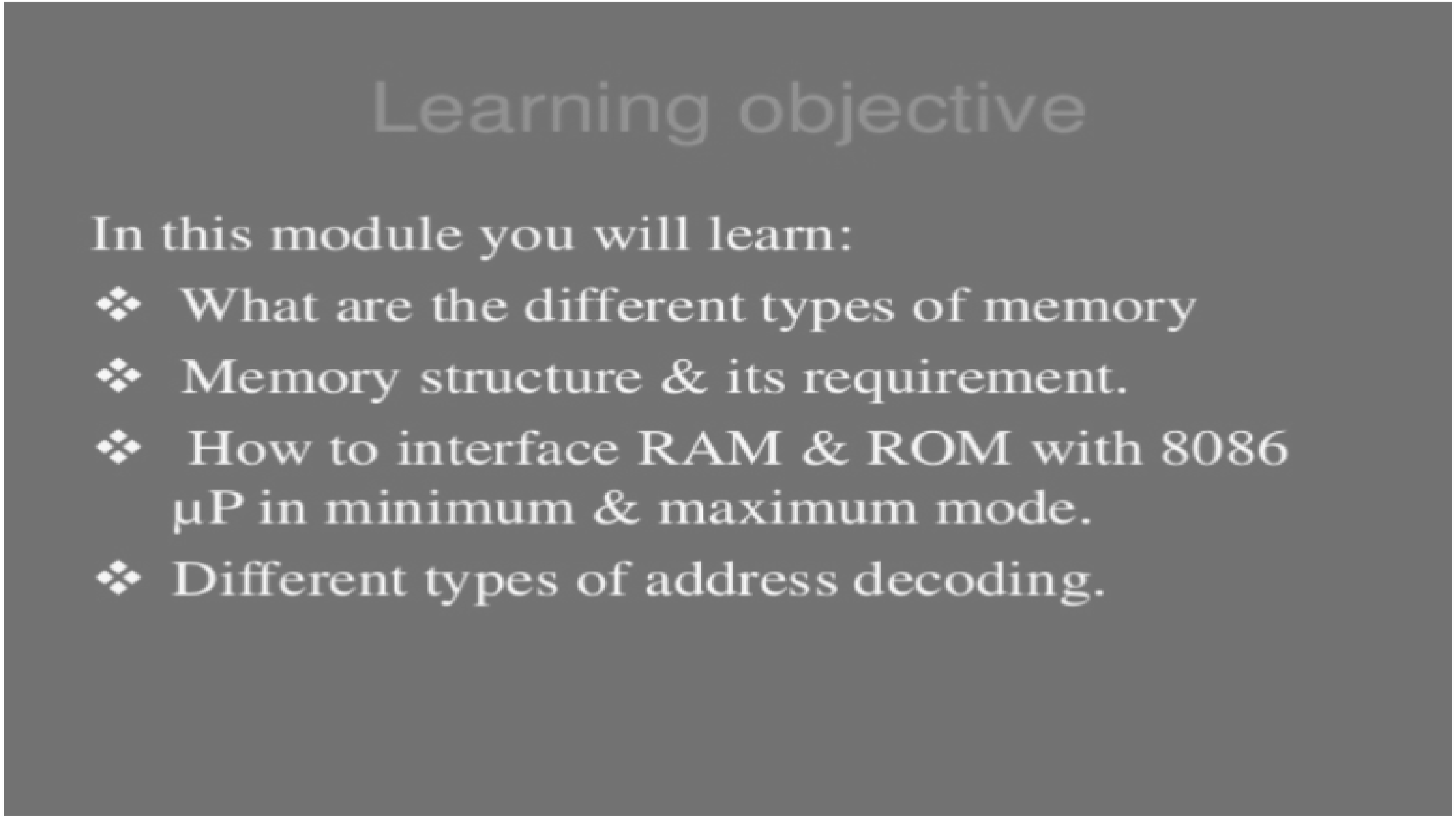


Number to be displayed	PA7d	PA6	PA5	PA4	PA3	PA2	PA1	PA0	Code
	p	a	b	c	d	e	f	g	
1	1	1	0	0	1	1	1	1	CF
2	1	0	0	1	0	0	1	0	92
3	1	0	0	0	0	1	1	0	86
4	1	1	0	0	1	1	0	0	CC
5	1	0	1	0	0	1	0	0	A4

Interfacing multiplexed 7-segment display



Interfacing of memory with 8086 microprocessor



Learning objective

In this module you will learn:

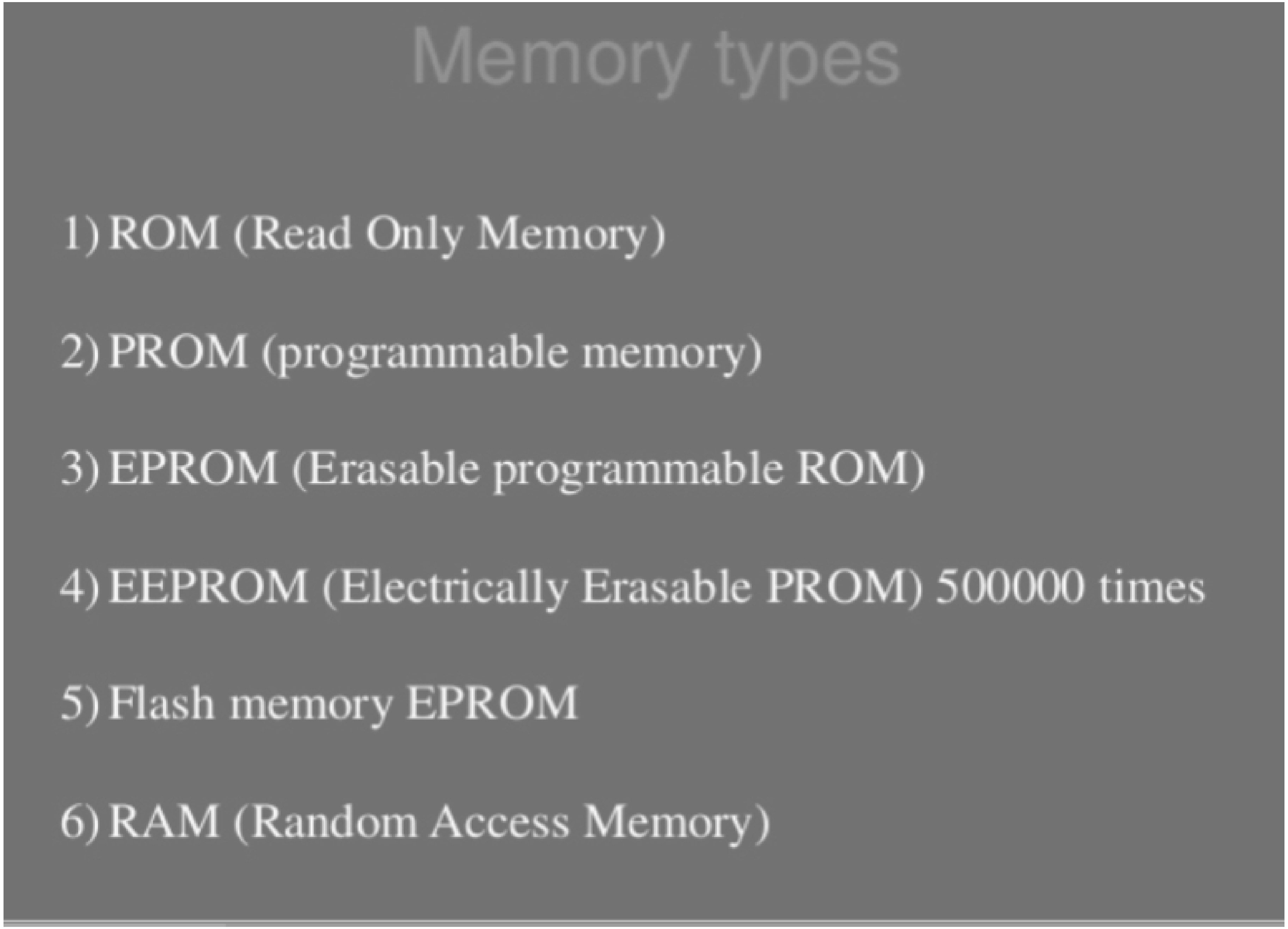
- ❖ What are the different types of memory
- ❖ Memory structure & its requirement.
- ❖ How to interface RAM & ROM with 8086 µP in minimum & maximum mode.
- ❖ Different types of address decoding.

introduction

- Memory is simply a device that can be used to store the information .
- The semiconductor memories are extensively used because of their small size, low cost, high speed, high reliability & ease of expansion of the memory size.
- It consist of mainly flip-flop & some additional circuitry such as buffers, one flip flop can hold one bit of data.

Memory fundamentals

- Memory capacity
  - ➢ The no. of bits that a semiconductor memory chip can store is called its chip capacity.
- Memory Organization:
  - ➢ Each memory chip contains 2^N locations, where N is the no. of address pins on the chip.
  - ➢ Each location contains M bits, where M is the no. of data pins on the chip.
  - ➢ The entire chip will contain 2^N x M bits.
  - ➢ E.g. for 4K x 4, 2^{12}=4096 locations, each location holding 4 bits, so N=12 & M=4.



Memory types

1) ROM (Read Only Memory)

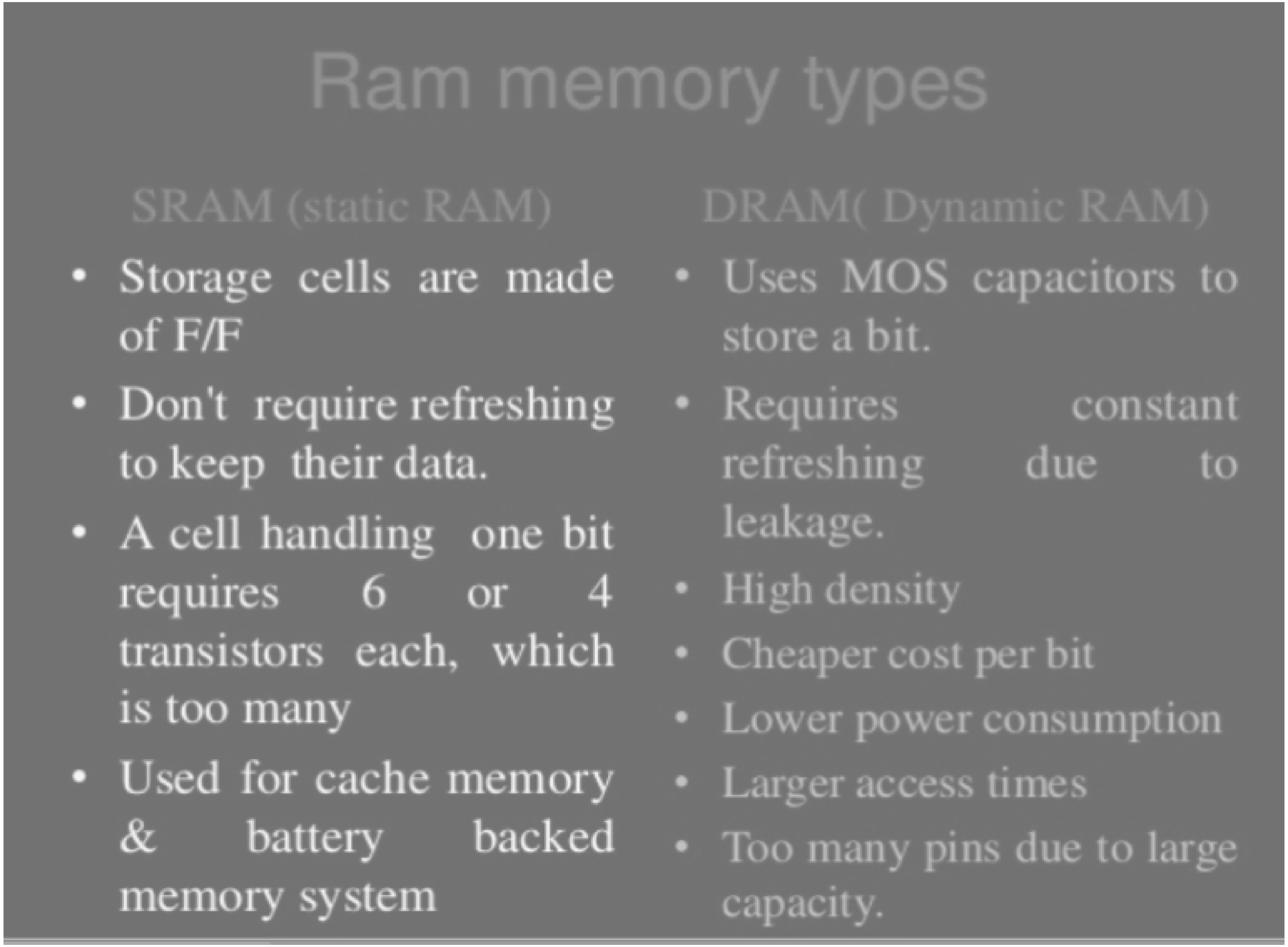
2) PROM (programmable memory)

3) EPROM (Erasable programmable ROM)

4) EEPROM (Electrically Erasable PROM) 500000 times

5) Flash memory EPROM

6) RAM (Random Access Memory)



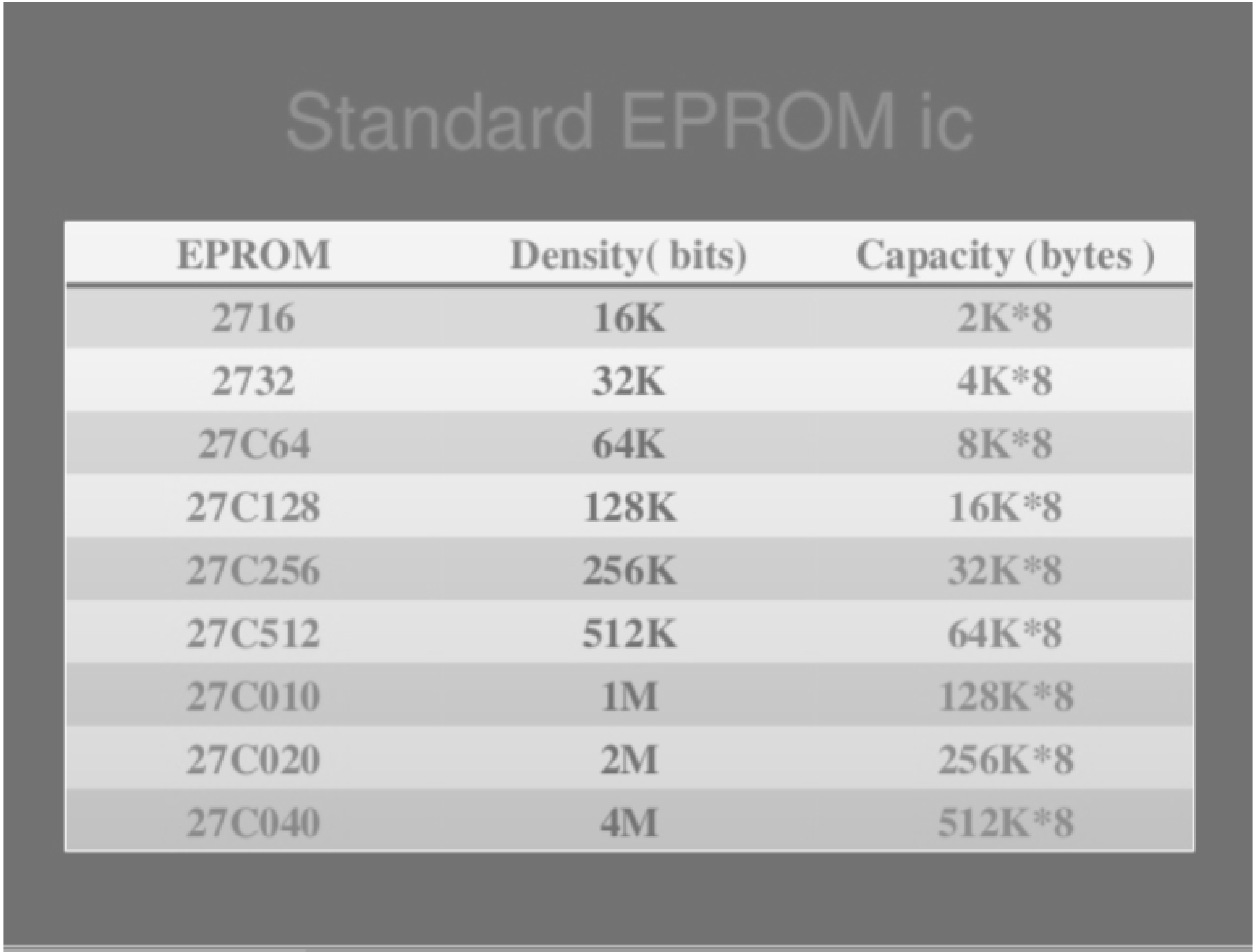
Ram memory types

SRAM (static RAM)

- Storage cells are made of F/F
- Don't require refreshing to keep their data.
- A cell handling one bit requires 6 or 4 transistors each, which is too many
- Used for cache memory & battery backed memory system

DRAM(Dynamic RAM)

- Uses MOS capacitors to store a bit.
- Requires constant refreshing due to leakage.
- High density
- Cheaper cost per bit
- Lower power consumption
- Larger access times
- Too many pins due to large capacity.



Standard EPROM ic

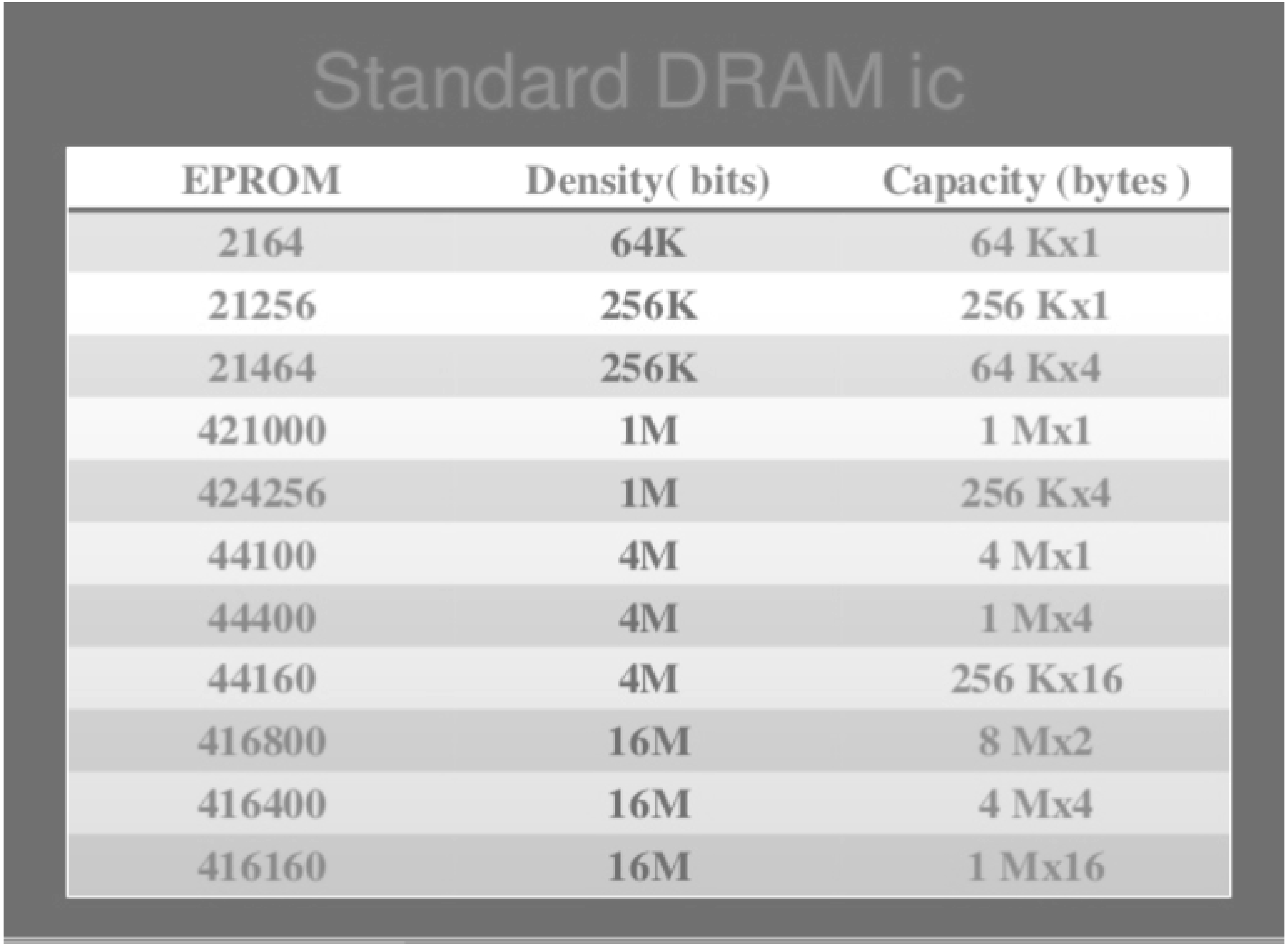
EPROM	Density(bits)	Capacity (bytes)
2716	16K	2K*8
2732	32K	4K*8
27C64	64K	8K*8
27C128	128K	16K*8
27C256	256K	32K*8
27C512	512K	64K*8
27C010	1M	128K*8
27C020	2M	256K*8
27C040	4M	512K*8

Standard SRAM ic

SRAM	Density(bits)	Organization
4361	64K	64K*1
4363	64K	16K*1
4364	64K	8K*8
43254	256K	64K*4
43256A		32K*8
431000A		128K*8

Absolute or full decoding

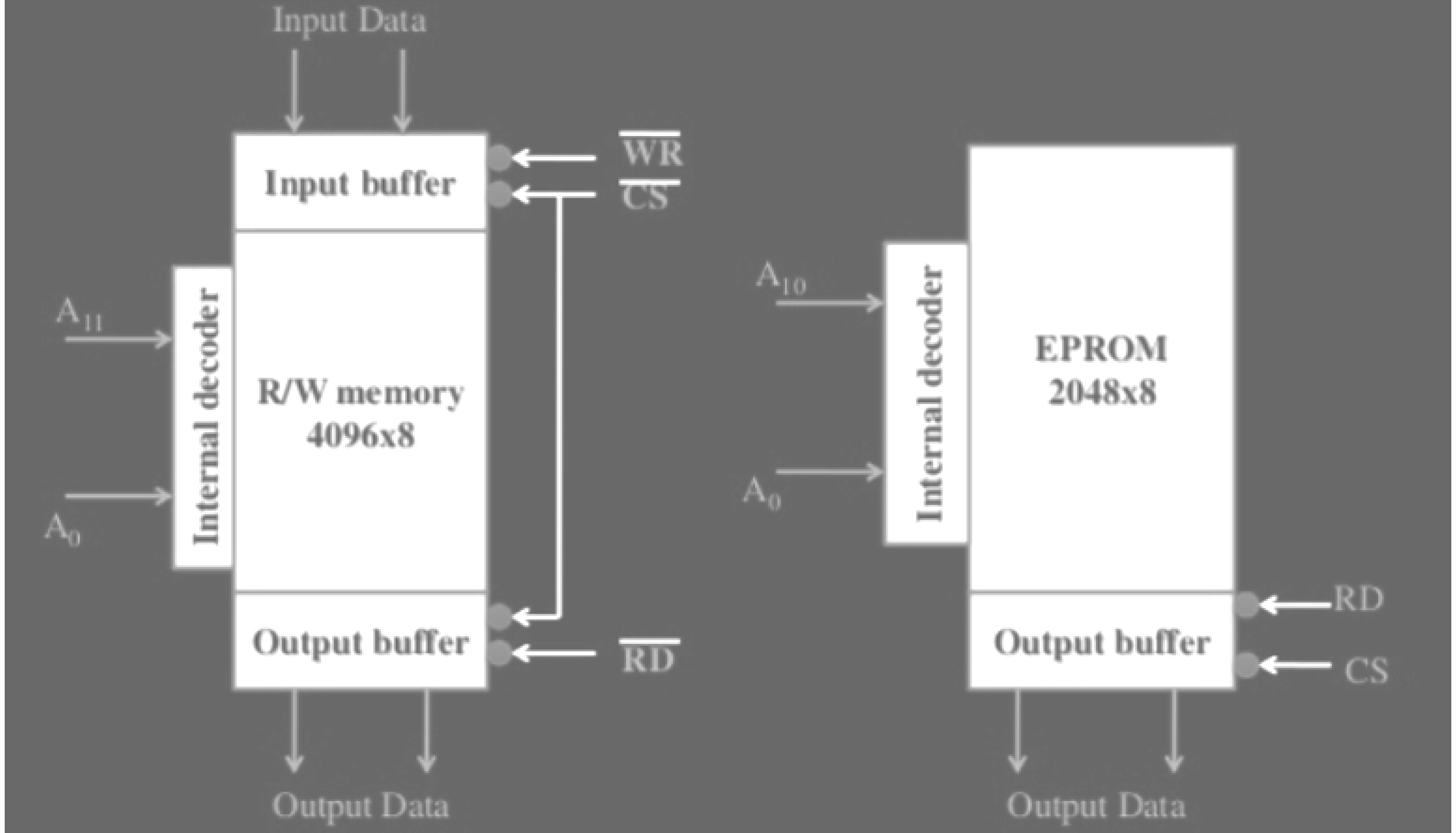
- All the higher address lines are decoded to select the memory chip.
- The memory chip is selected only for the specified logic levels on these higher order address lines.
- So each location have fixed address.
- This technique is expensive
- It needs more hardware than partial decoding.



Standard DRAM ic

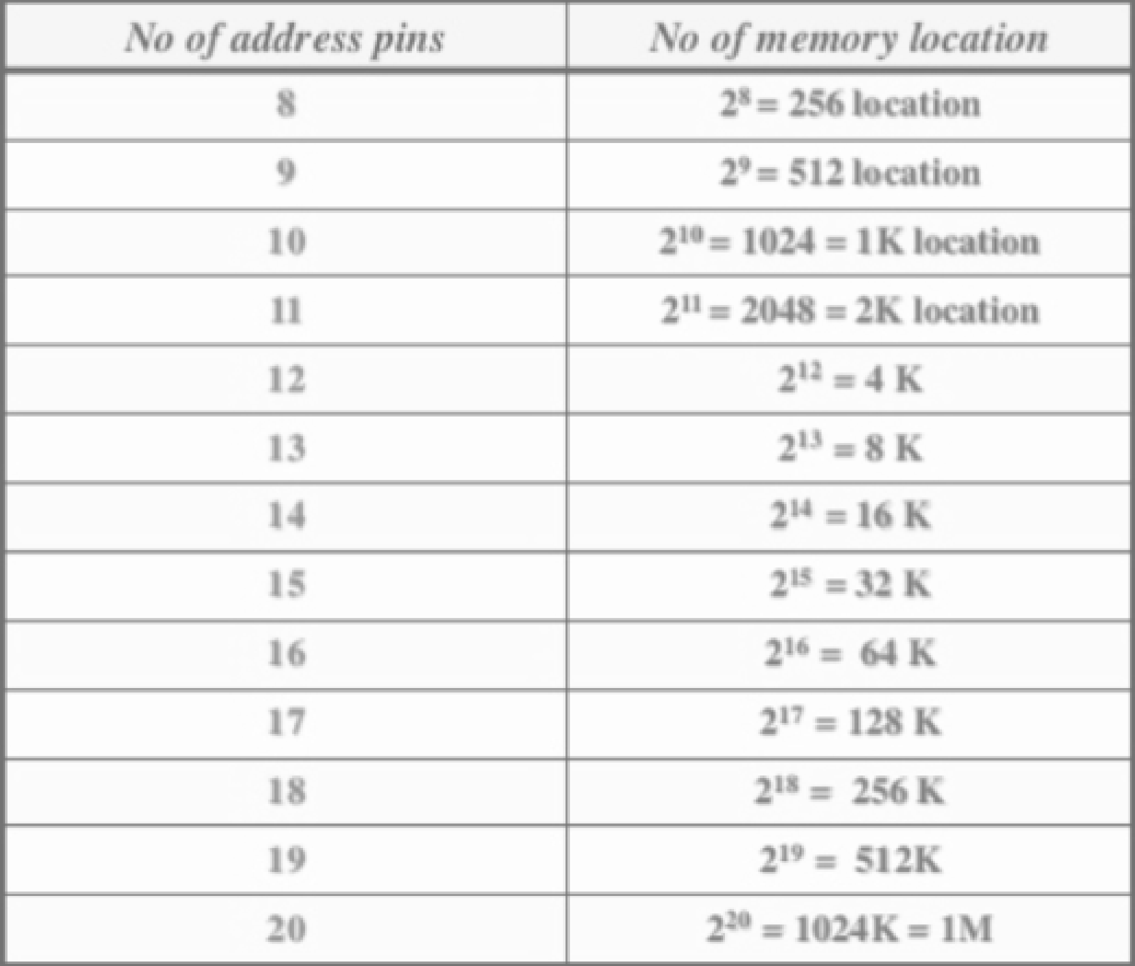
EPROM	Density(bits)	Capacity (bytes)
2164	64K	64 Kx1
21256	256K	256 Kx1
21464	256K	64 Kx4
421000	1M	1 Mx1
424256	1M	256 Kx4
44100	4M	4 Mx1
44400	4M	1 Mx4
44160	4M	256 Kx16
416800	16M	8 Mx2
416400	16M	4 Mx4
416160	16M	1 Mx16

MEMORY STRUCTURE & ITS REQUIREMENT

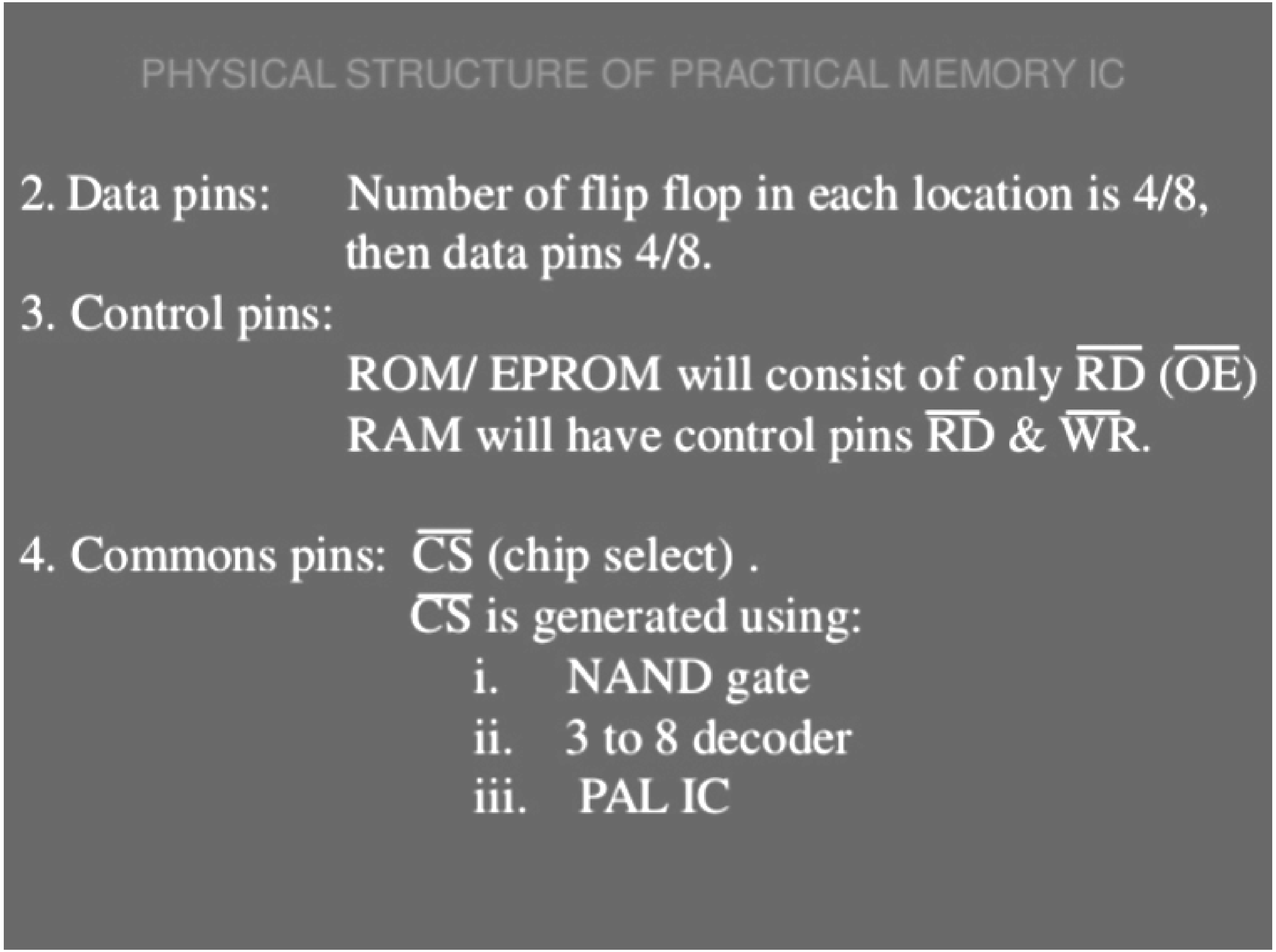


PHYSICAL STRUCTURE OF PRACTICAL MEMORY IC

1. Address Pins:



No of address pins	*No of memory location*
8	2^8= 256 location
9	2^9= 512 location
10	2^{10}= 1024 = 1K location
11	2^{11} = 2048 = 2K location
12	2^{12} = 4 K
13	2^{13} = 8 K
14	2^{14} = 16 K
15	2^{15} = 32 K
16	2^{16} = 64 K
17	2^{17} = 128 K
18	2^{18} = 256 K
19	2^{19} = 512K
20	2^{20} = 1024K = 1M



PHYSICAL STRUCTURE OF PRACTICAL MEMORY IC

2. Data pins: Number of flip flop in each location is 4/8, then data pins 4/8.

3. Control pins:

   ROM/ EPROM will consist of only $\overline{RD}$ ($\overline{OE}$)

   RAM will have control pins $\overline{RD}$ & $\overline{WR}$.

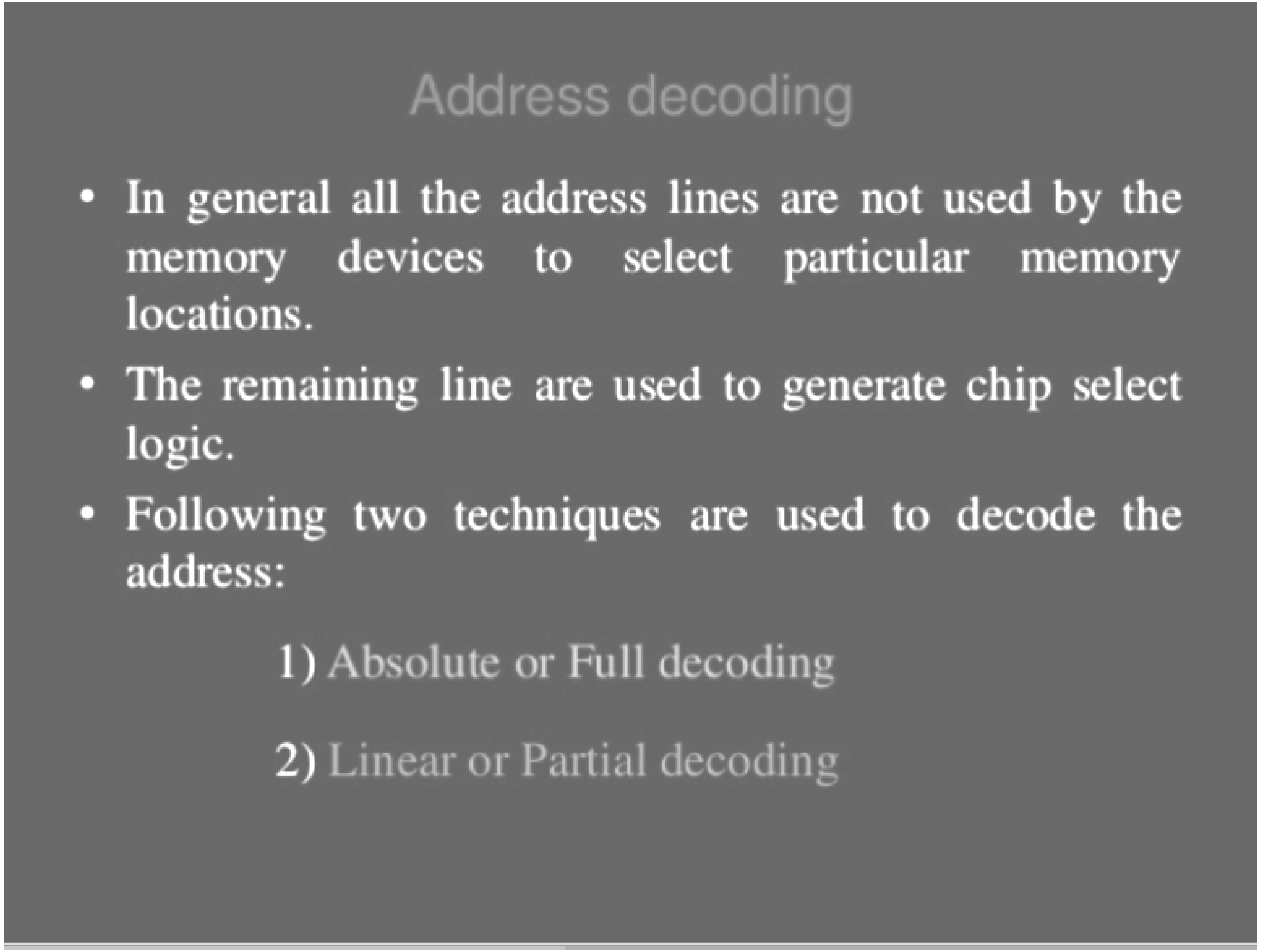
4. Commons pins: $\overline{CS}$ (chip select) .

   $\overline{CS}$ is generated using:

   i. NAND gate

   ii. 3 to 8 decoder

   iii. PAL IC

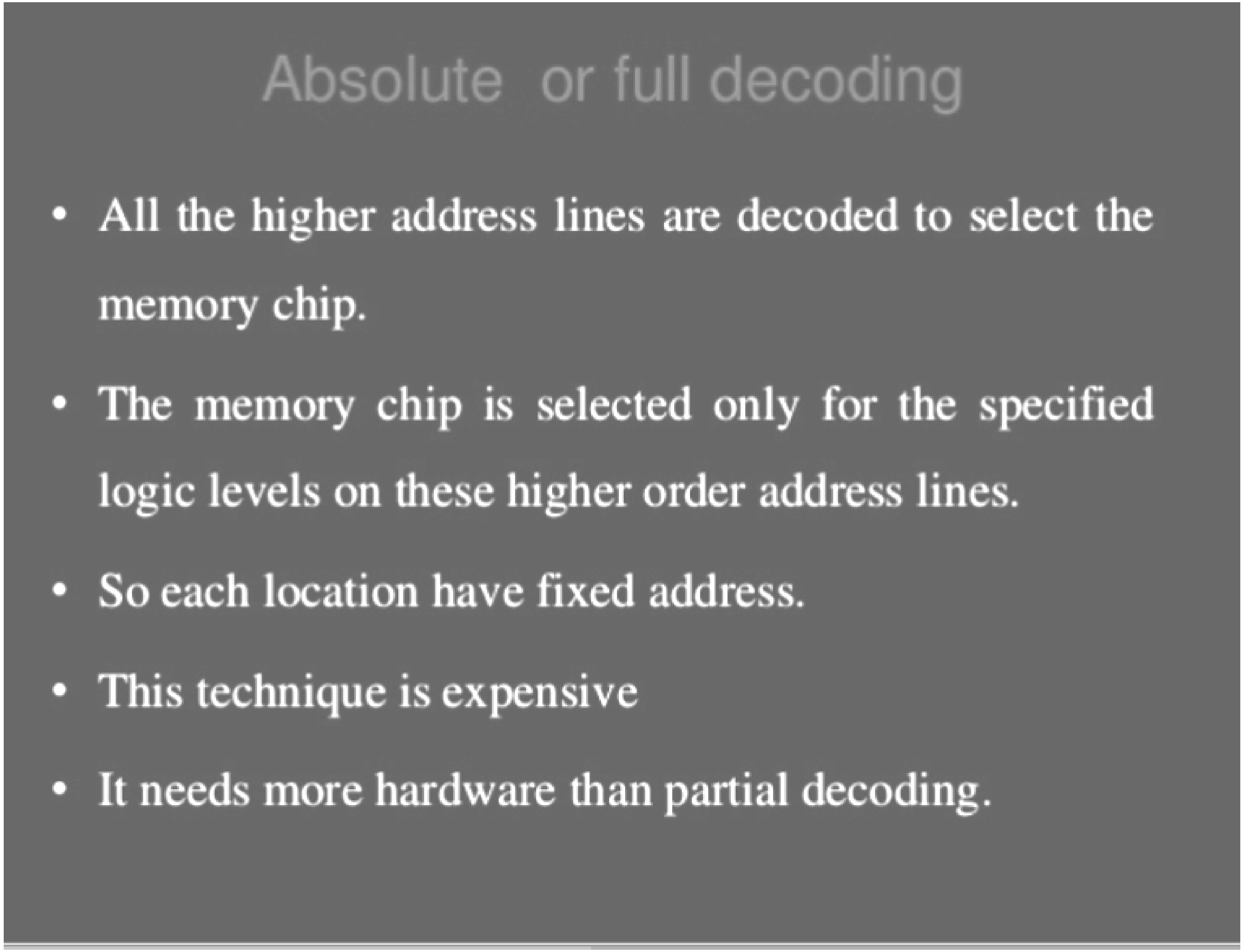


Address decoding

- In general all the address lines are not used by the memory devices to select particular memory locations.
- The remaining line are used to generate chip select logic.
- Following two techniques are used to decode the address:

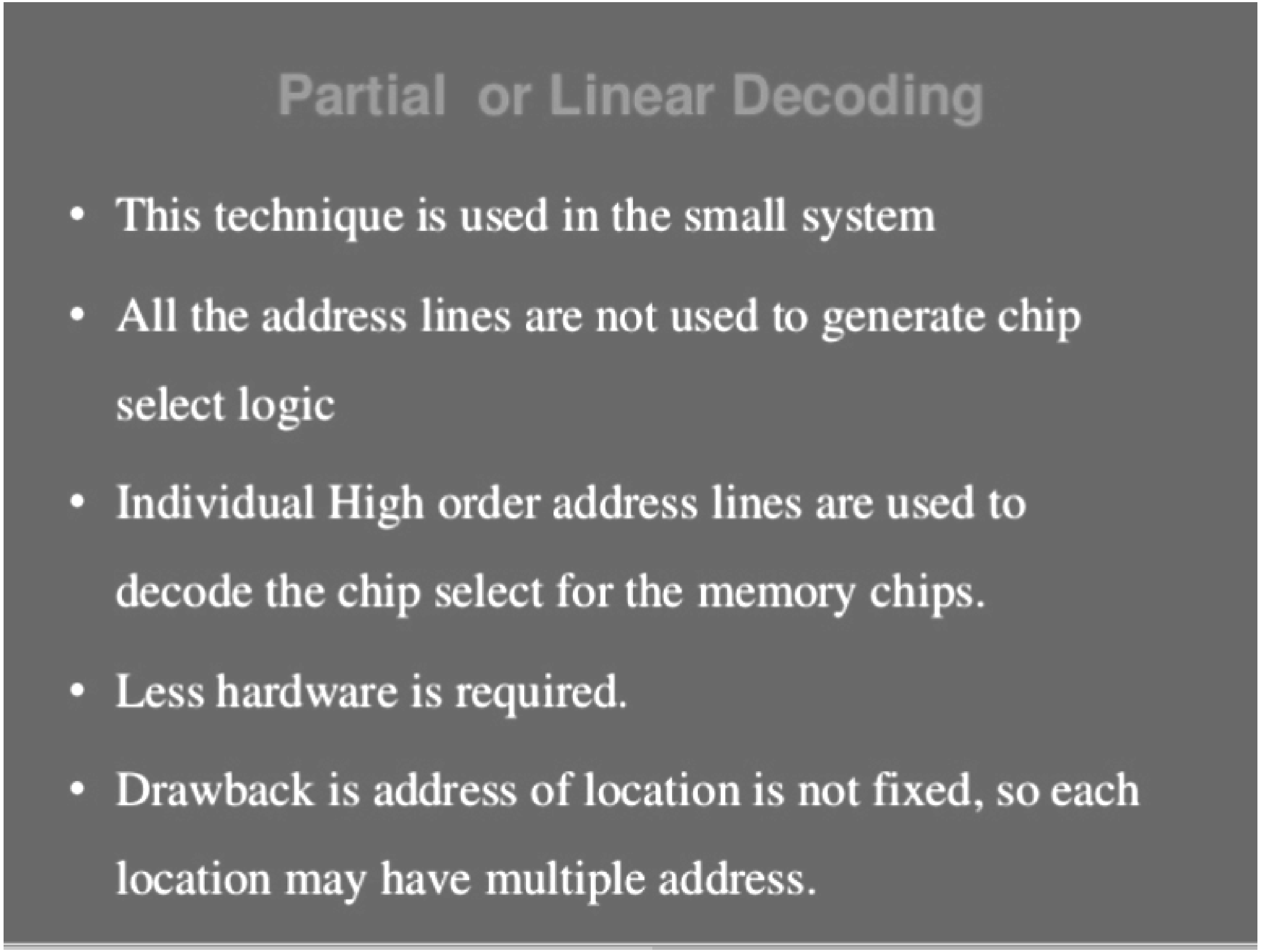
1) Absolute or Full decoding

2) Linear or Partial decoding



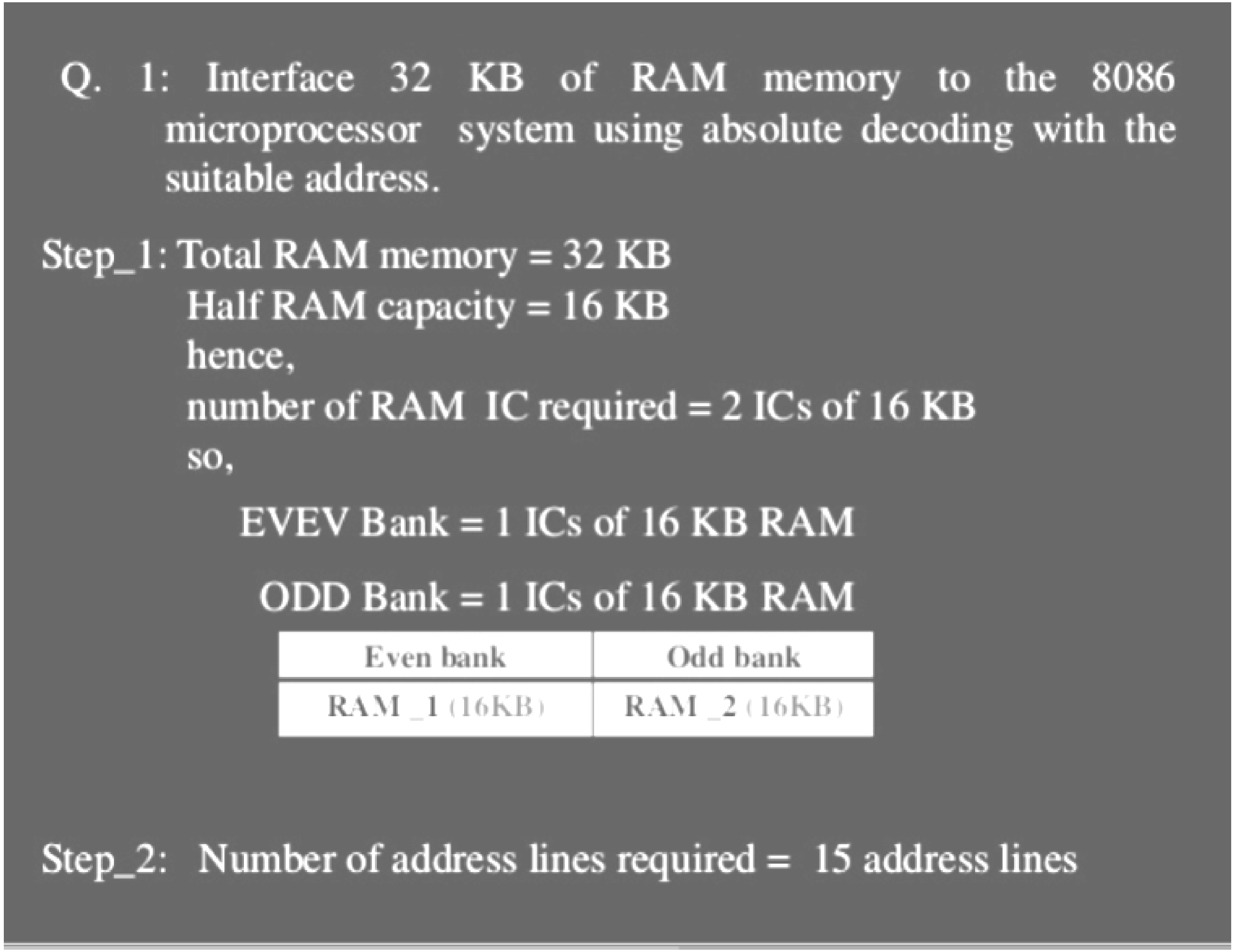
Absolute or full decoding

- All the higher address lines are decoded to select the memory chip.
- The memory chip is selected only for the specified logic levels on these higher order address lines.
- So each location have fixed address.
- This technique is expensive
- It needs more hardware than partial decoding.



Partial or Linear Decoding

- This technique is used in the small system
- All the address lines are not used to generate chip select logic
- Individual High order address lines are used to decode the chip select for the memory chips.
- Less hardware is required.
- Drawback is address of location is not fixed, so each location may have multiple address.



Q. 1: Interface 32 KB of RAM memory to the 8086 microprocessor system using absolute decoding with the suitable address.

Step_1: Total RAM memory = 32 KB
Half RAM capacity = 16 KB
hence,
number of RAM IC required = 2 ICs of 16 KB
so,

EVEV Bank = 1 ICs of 16 KB RAM

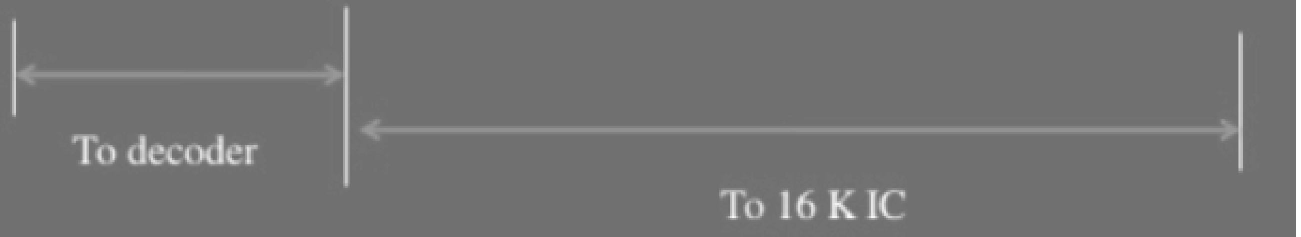
ODD Bank = 1 ICs of 16 KB RAM

Even bank	Odd bank
RAM _1 (16KB)	RAM _2 (16KB)

Step_2: Number of address lines required = 15 address lines

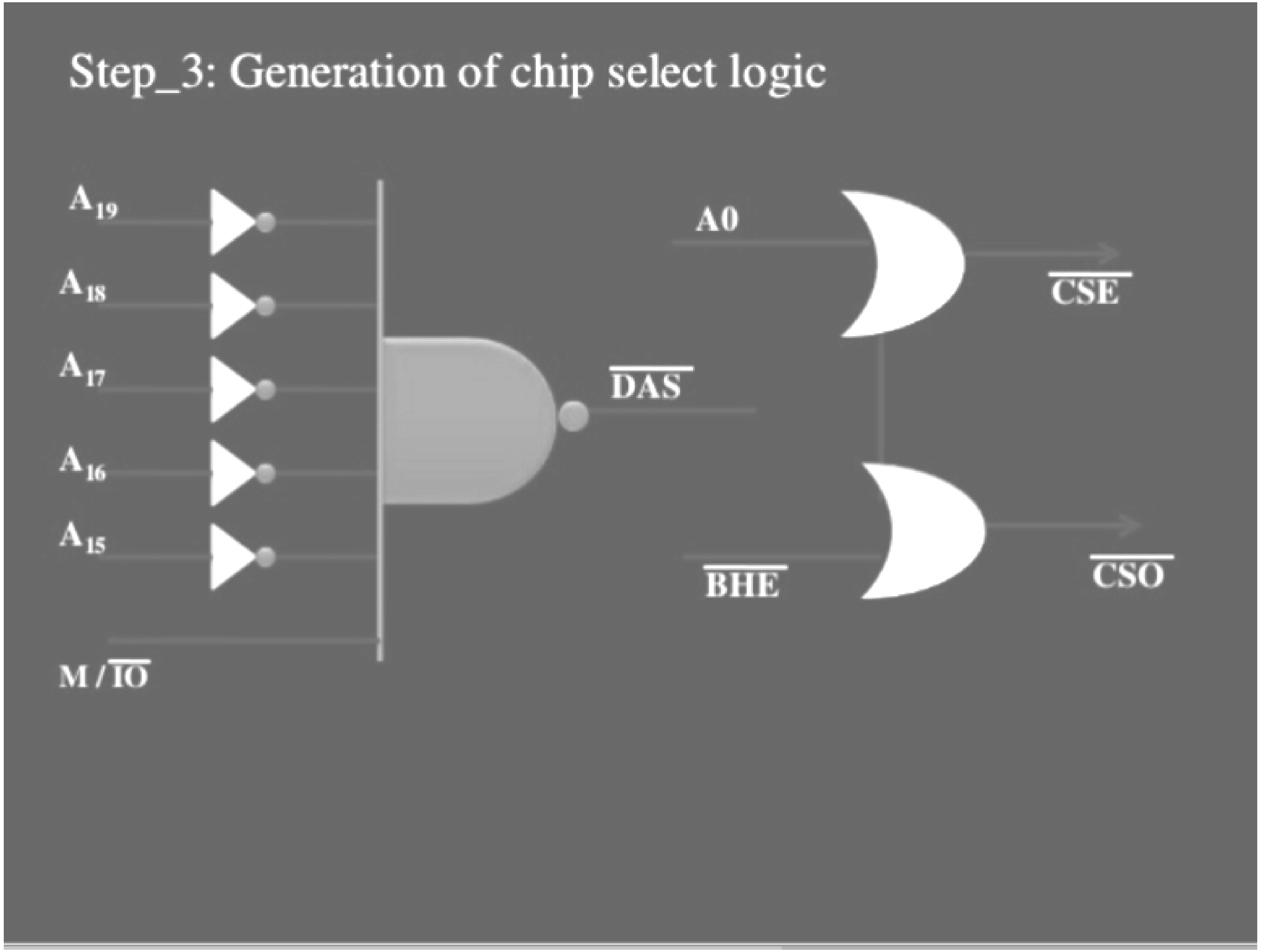
Step_3: Address decoding table

MEMORY IC	HEX ADDRESS	BINARY ADDRESS																			
		A_{19}	A_{18}	A_{17}	A_{16}	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
16 K x 8 RAM-(1)	OOOOO	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	07FFE	0	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0
16 K x 8 RAM-(3)	8000	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	0FFFE	0	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0

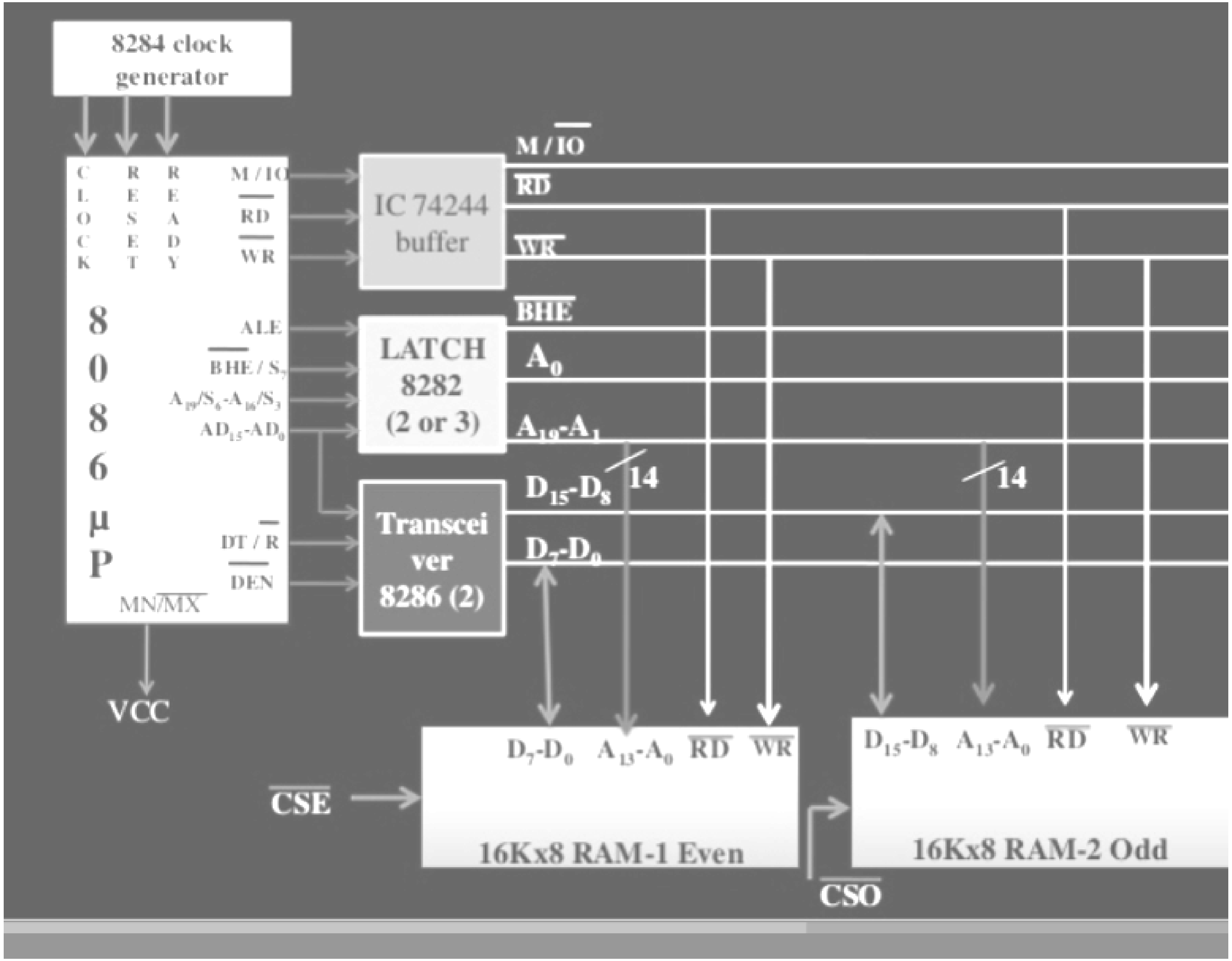


To decoder

To 16 K IC



Step_3: Generation of chip select logic


8284 clock generator
CLOCK
RESET
READY
$M/\overline{IO}$
$\overline{RD}$
$\overline{WR}$
8086 μP
ALE
$\overline{BHE}/S_7$
A_{19}/S_6-A_{16}/S_3
AD_{15}-AD_0
$DT/\overline{R}$
$\overline{DEN}$
$MN/\overline{MX}$
VCC
IC 74244 buffer
LATCH 8282 (2 or 3)
Transceiver 8286 (2)
$M/\overline{IO}$
$\overline{RD}$
$\overline{WR}$
$\overline{BHE}$
A_0
A_{19}-A_1
14
D_{15}-D_8
14
D_7-D_0
D_7-D_0
A_{13}-A_0
$\overline{RD}$
$\overline{WR}$
$\overline{CSE}$
16Kx8 RAM-1 Even
D_{15}-D_8
A_{13}-A_0
$\overline{RD}$
$\overline{WR}$
16Kx8 RAM-2 Odd
$\overline{CSO}$

Q. 2: Interface 32 K word of memory to the 8086 microprocessor system . Available memory chips are 16 K x 8 RAM. Use suitable decoder for generating chip select logic.

Step_1: Total memory = 32 K word = 32*2 K = 64 K

IC available = 16 K

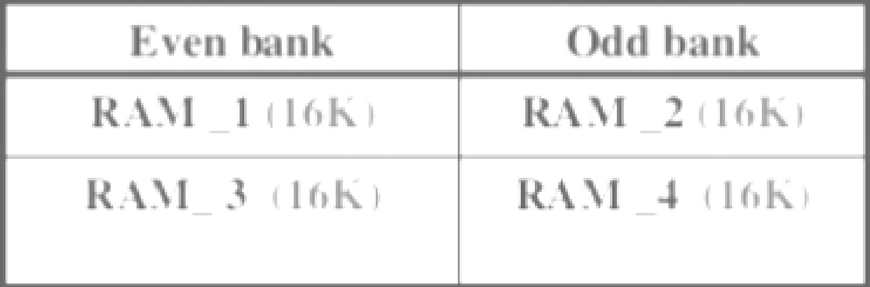
hence,

number of RAM IC required = 64 K x 8/ 16 Kx8 = 4 ICs

so,

EVEV Bank = 2 ICs of 16 Kx8 RAM

ODD Bank = 2 ICs of 16 Kx8 RAM



Even bank	Odd bank
RAM _1 (16K)	RAM _2 (16K)
RAM_ 3 (16K)	RAM _4 (16K)

Step_2: Number of address lines required = 15 address lines

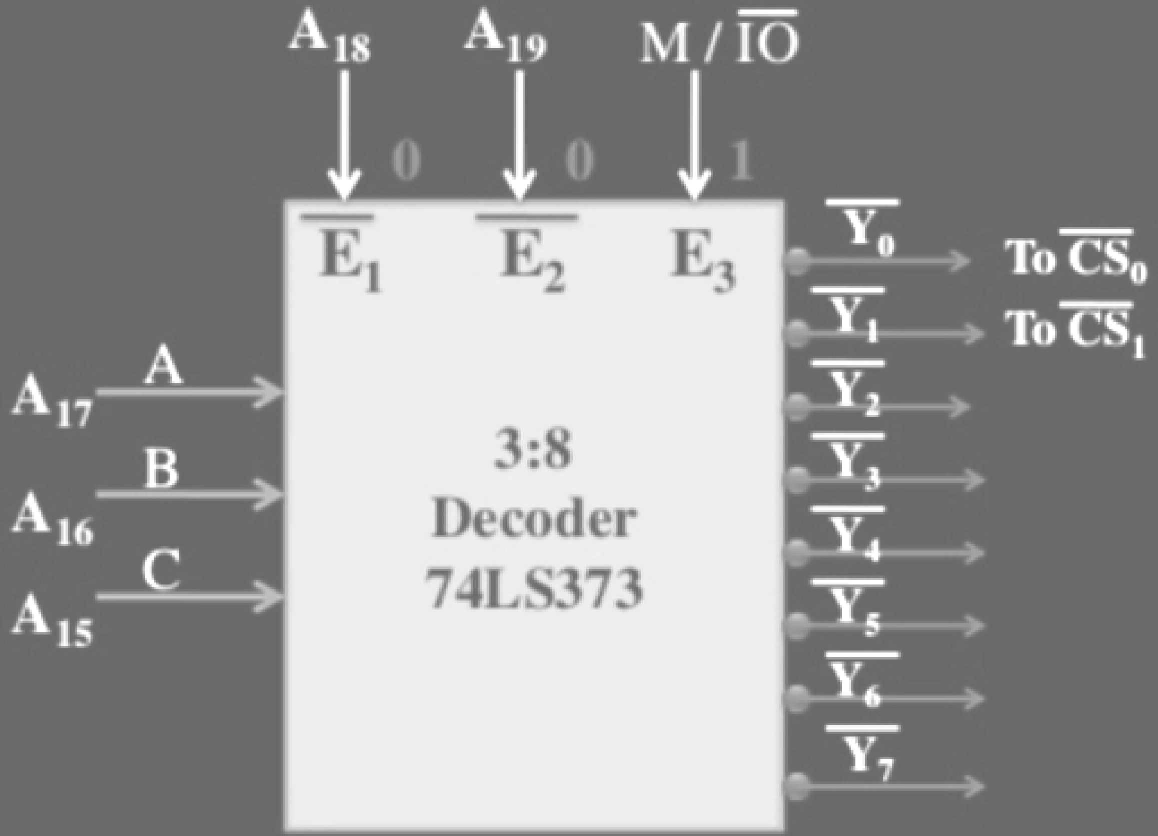
Step_3: Address decoding table

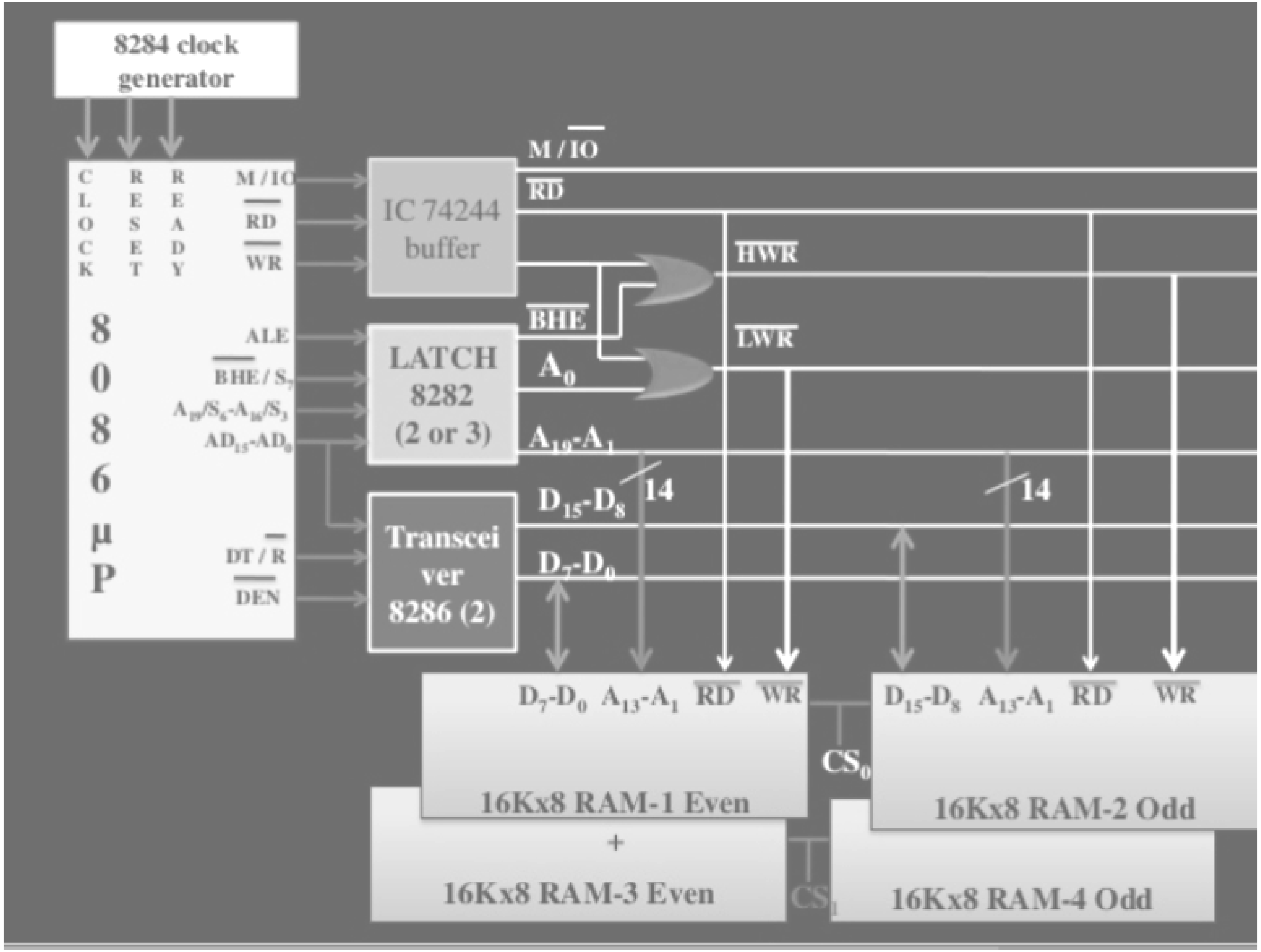
MEMORY IC	HEX ADDRESS	BINARY ADDRESS																			
		A_{19}	A_{18}	A_{17}	A_{16}	A_{15}	A_{14}	A_{13}	A_{12}	A_{11}	A_{10}	A_9	A_8	A_7	A_6	A_5	A_4	A_3	A_2	A_1	A_0
16 K x 8 RAM-(1)	00000	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	07FFE	0	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0
16 K x 8 RAM-(3)	8000	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	0FFFE	0	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0

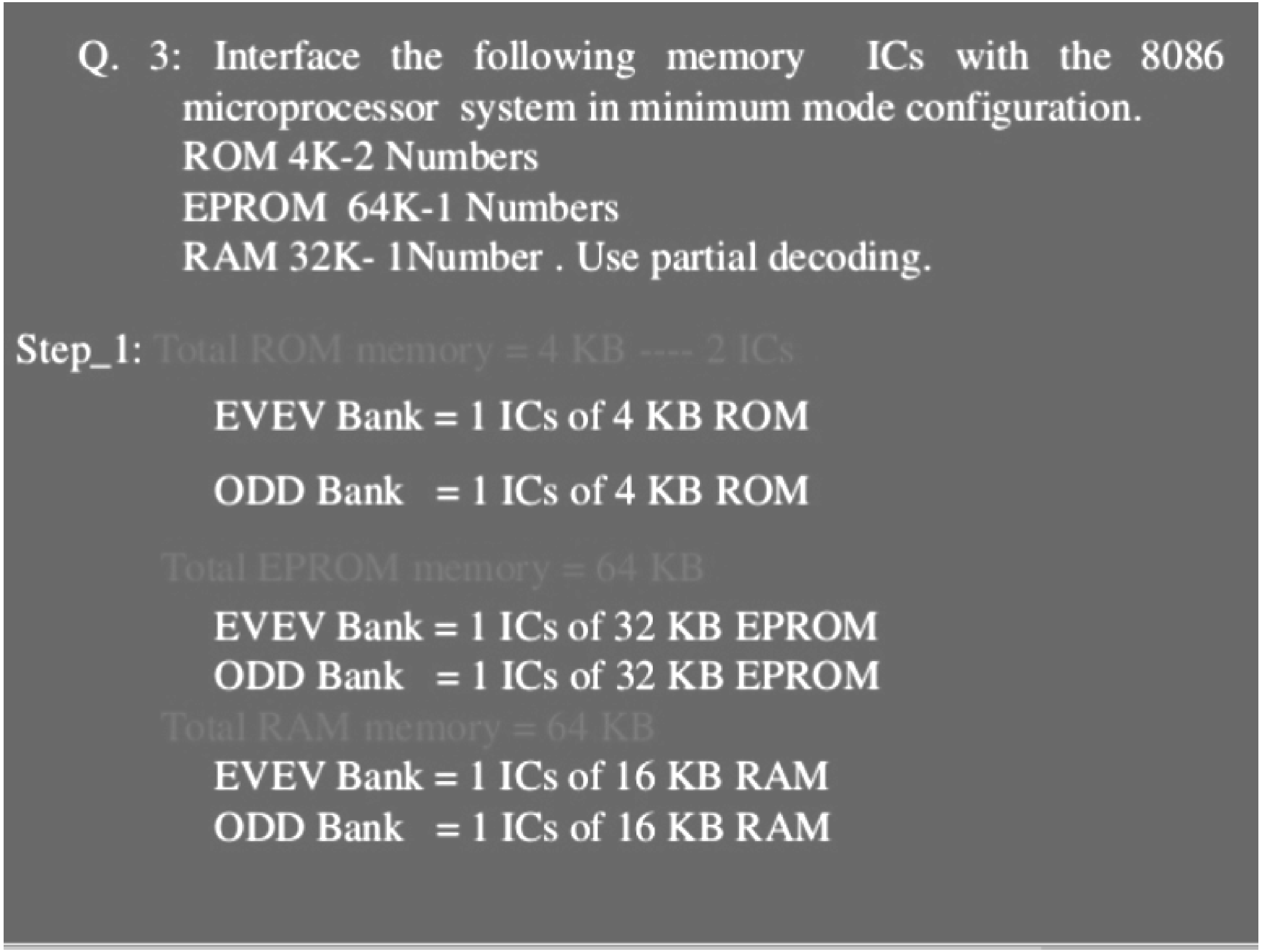
To decoder

To 16 K IC

Step_3: Generation of chip select logic




8284 clock generator
CLOCK
RESET
READY
8086 µP
M / IO
$\overline{RD}$
$\overline{WR}$
ALE
$\overline{BHE}$ / S_7
A_{19}/S_6-A_{16}/S_3
AD_{15}-AD_0
DT / $\overline{R}$
$\overline{DEN}$
IC 74244 buffer
LATCH 8282 (2 or 3)
Transceiver 8286 (2)
M / $\overline{IO}$
$\overline{RD}$
$\overline{HWR}$
$\overline{BHE}$
$\overline{LWR}$
A_0
A_{19}-A_1
14
D_{15}-D_8
14
D_7-D_0
D_7-D_0
A_{13}-A_1
$\overline{RD}$
$\overline{WR}$
CS_0
D_{15}-D_8
A_{13}-A_1
$\overline{RD}$
$\overline{WR}$
16Kx8 RAM-1 Even
+
16Kx8 RAM-3 Even
CS_1
16Kx8 RAM-2 Odd
16Kx8 RAM-4 Odd



Q. 3: Interface the following memory ICs with the 8086 microprocessor system in minimum mode configuration.
ROM 4K-2 Numbers
EPROM 64K-1 Numbers
RAM 32K- 1Number . Use partial decoding.

Step_1: Total ROM memory = 4 KB ---- 2 ICs

EVEV Bank = 1 ICs of 4 KB ROM

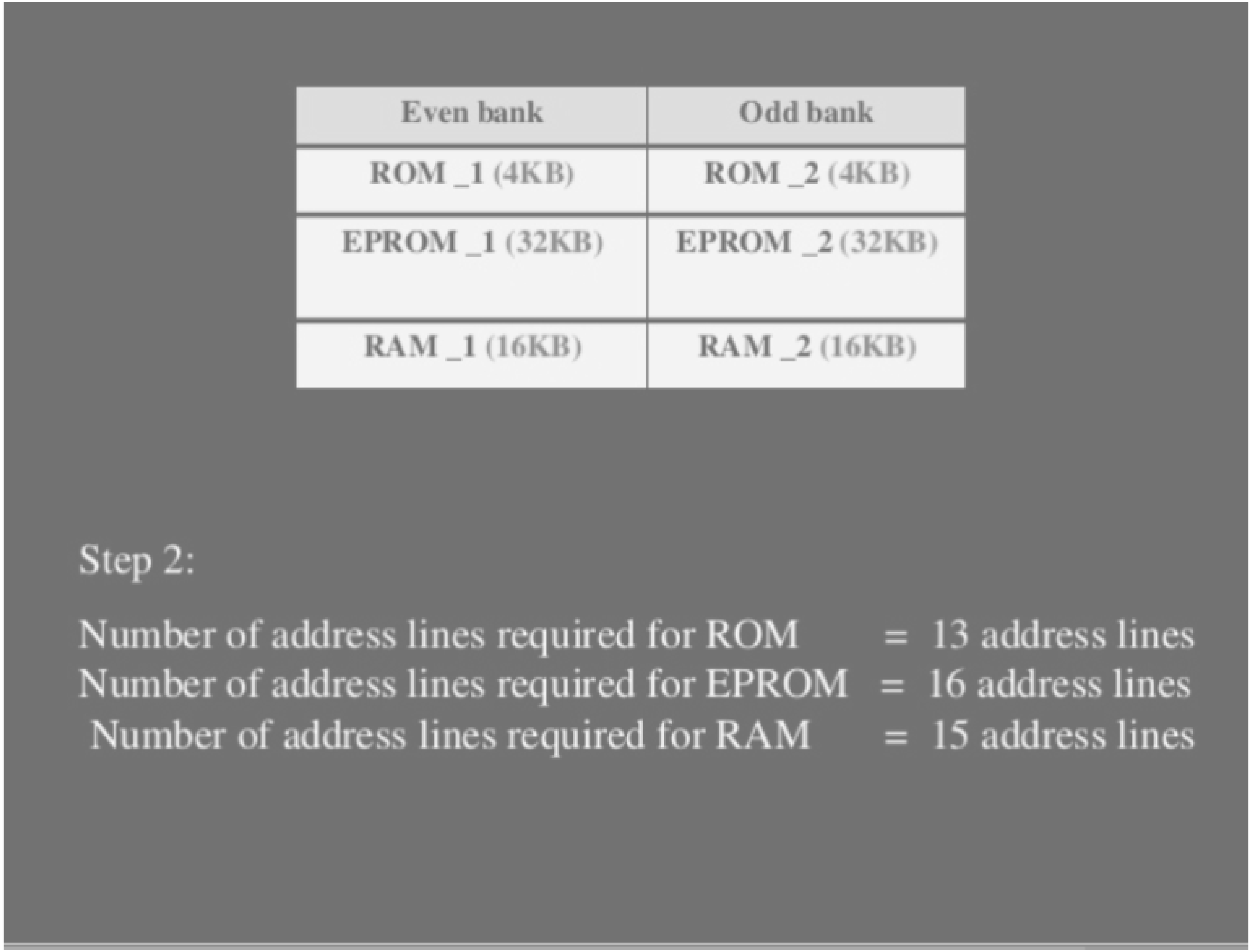
ODD Bank = 1 ICs of 4 KB ROM

Total EPROM memory = 64 KB

EVEV Bank = 1 ICs of 32 KB EPROM
ODD Bank = 1 ICs of 32 KB EPROM

Total RAM memory = 64 KB

EVEV Bank = 1 ICs of 16 KB RAM
ODD Bank = 1 ICs of 16 KB RAM



Even bank	Odd bank
ROM _1 (4KB)	ROM _2 (4KB)
EPROM _1 (32KB)	EPROM _2 (32KB)
RAM _1 (16KB)	RAM _2 (16KB)

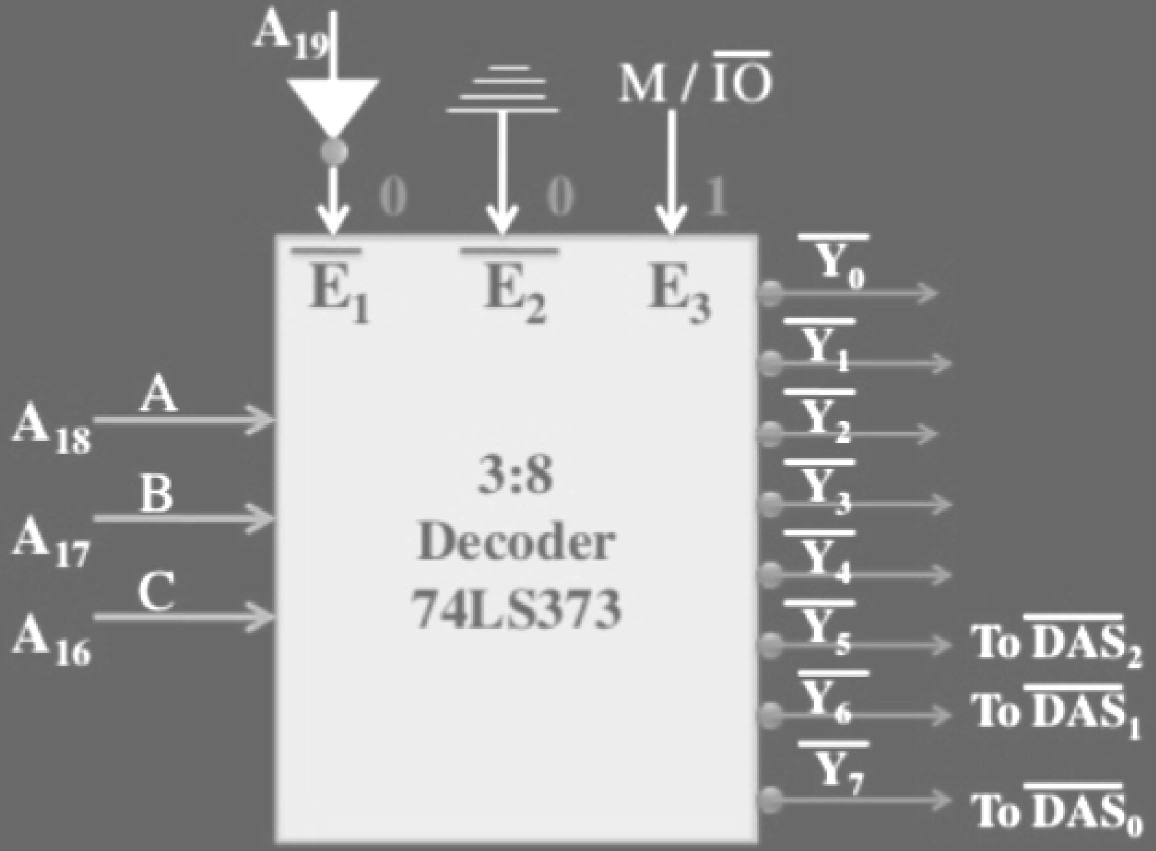
Step 2:

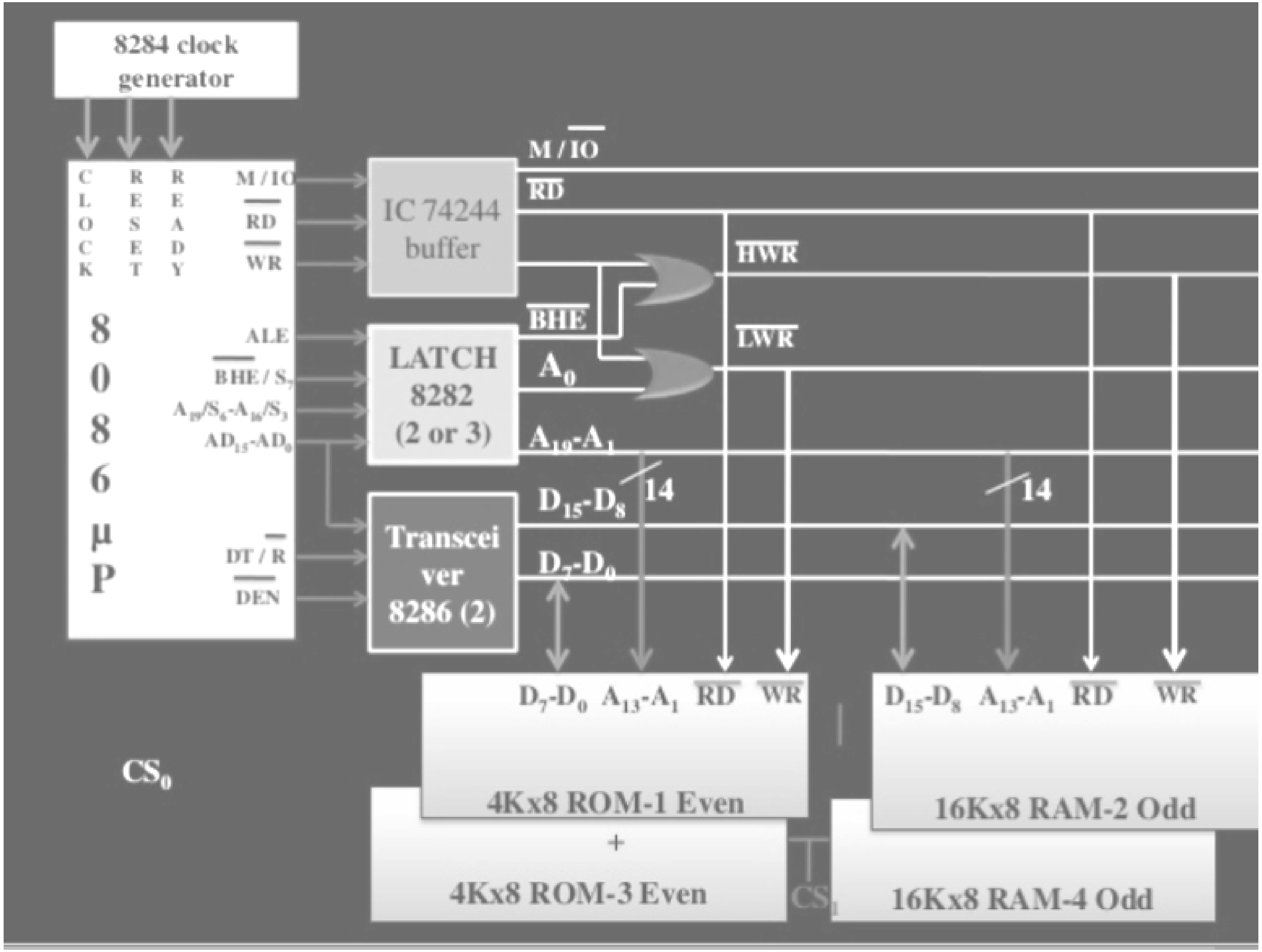
Number of address lines required for ROM = 13 address lines
Number of address lines required for EPROM = 16 address lines
Number of address lines required for RAM = 15 address lines

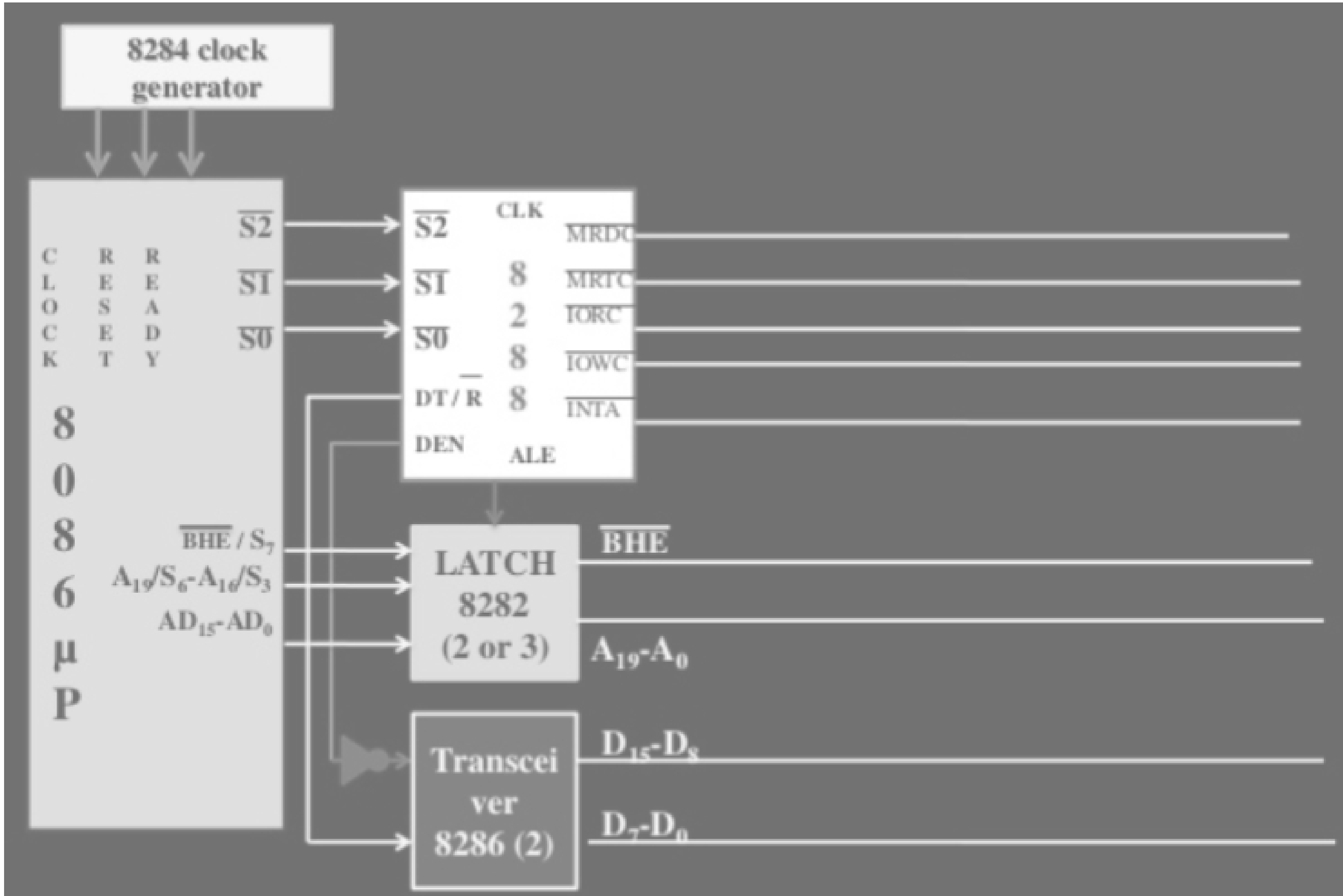
Step_3: Address decoding table

MEMORY IC	HEX ADDRESS	BINARY ADDRESS																			
		A_{19}	A_{18}	A_{17}	A_{16}	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
4 K x 8 ROM-(1)	FFFFE	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0
	FE000	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0
									To ROM IC												
32 K x 8 EPROM-(1)	EFFFE	1	1	1	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0
	E0000	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
						To EPROM IC															
16 K x 8 RAM-(1)	D0000	1	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	D7FFE	1	1	0	1	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0
		To decoder					To RAM IC														

Step_3: Generation of chip select logic

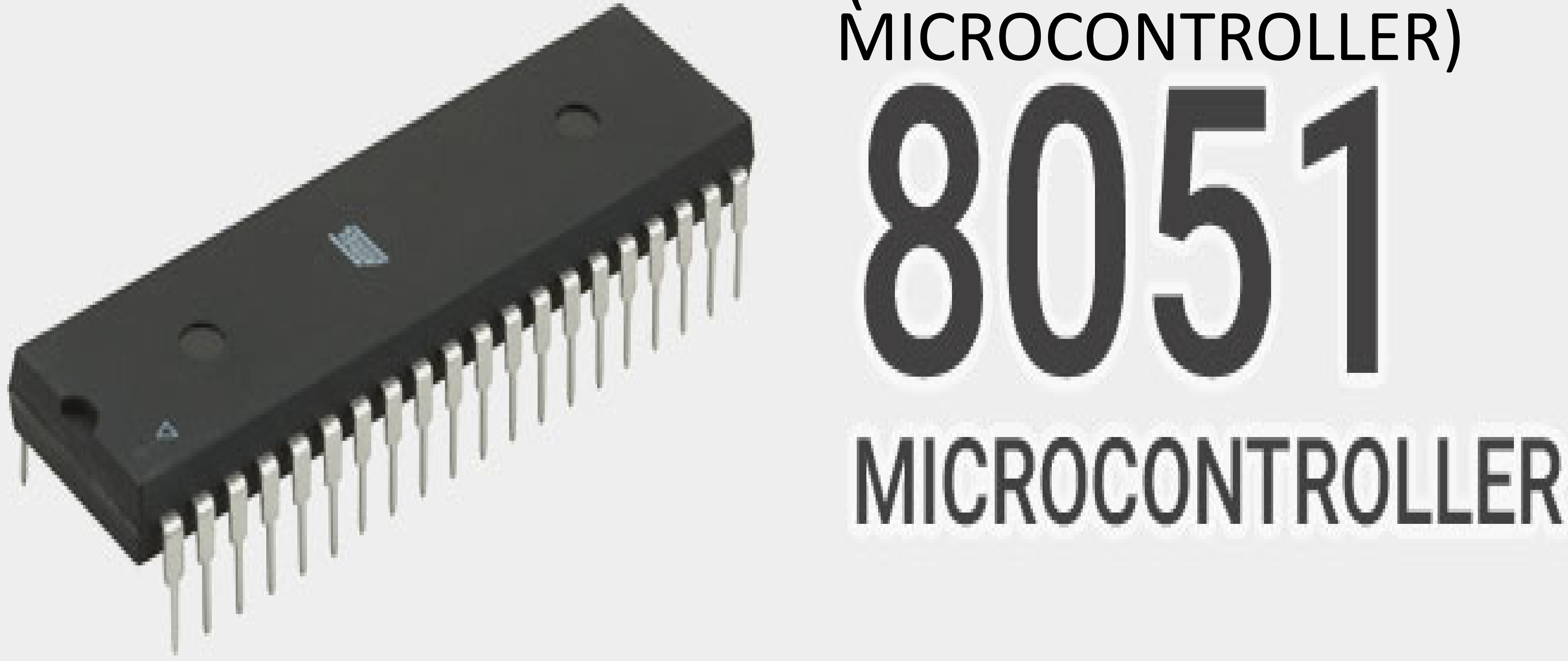



8284 clock generator
CLOCK
RESET
READY
8086 µP
M / IO
RD
WR
ALE
BHE / S7
A19/S6-A16/S3
AD15-AD0
DT / R
DEN
IC 74244 buffer
LATCH 8282 (2 or 3)
Transceiver 8286 (2)
M / IO
RD
HWR
BHE
LWR
A0
A19-A1
14
D15-D8
14
D7-D0
D7-D0
A13-A1
RD
WR
D15-D8
A13-A1
RD
WR
CS0
4Kx8 ROM-1 Even
+
4Kx8 ROM-3 Even
CS1
16Kx8 RAM-2 Odd
16Kx8 RAM-4 Odd


8284 clock generator
CLOCK
RESET
READY
8086 µP
$\overline{S2}$
$\overline{S1}$
$\overline{S0}$
$\overline{BHE}$ / S_7
A_{19}/S_6-A_{16}/S_3
AD_{15}-AD_0
$\overline{S2}$
$\overline{S1}$
$\overline{S0}$
CLK
8288
$DT/\overline{R}$
DEN
ALE
$\overline{MRDC}$
$\overline{MRTC}$
$\overline{IORC}$
$\overline{IOWC}$
$\overline{INTA}$
LATCH 8282 (2 or 3)
$\overline{BHE}$
A_{19}-A_0
Transceiver 8286 (2)
D_{15}-D_8
D_7-D_0

M0DULE-IV

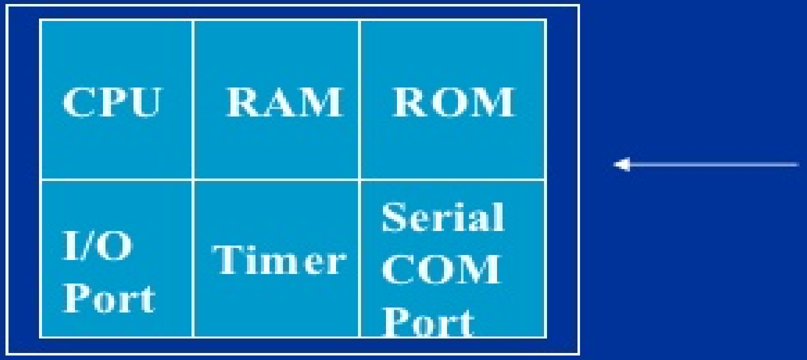
(INTRODUCTION TO MICROCONTROLLER)



- A **microcontroller** is a small and low-cost microcomputer, which is designed to perform the specific tasks of embedded systems like displaying microwave’s information, receiving remote signals, etc.
- The general microcontroller consists of the processor, the memory (RAM, ROM, EPROM), Serial ports, peripherals (timers, counters), etc.
- 8051 microcontroller is designed by Intel in 1981. It is an 8-bit microcontroller. It is built with 40 pins DIP (dual inline package).
- It is an Electronic IC.

Then What is a Microcontroller ?

- A smaller computer
- On-chip RAM, ROM, I/O ports...
- Example : Motorola's 6811, Intel's 8051, Zilog's Z8 and PIC 16X

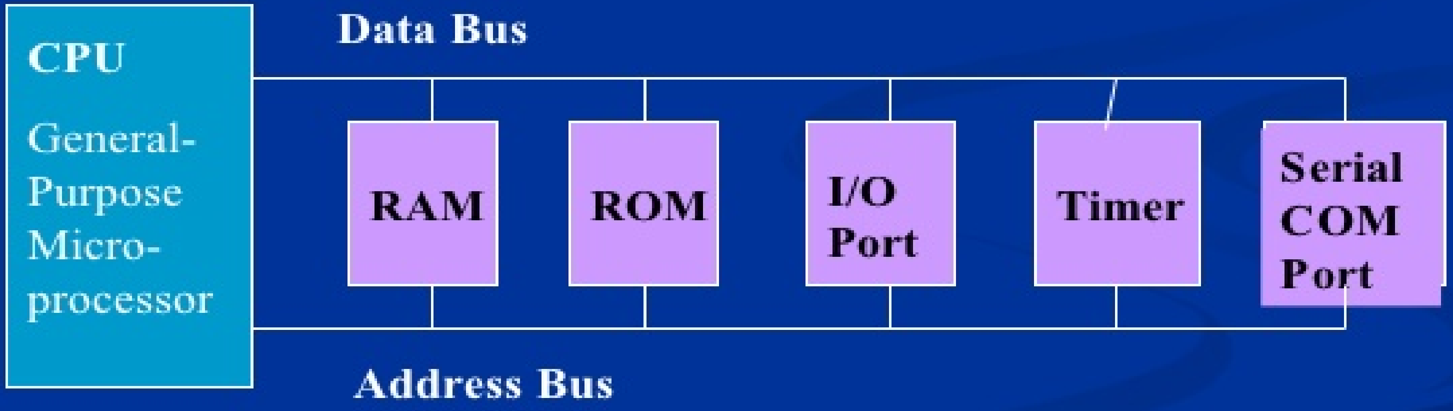


A single chip Microcontroller

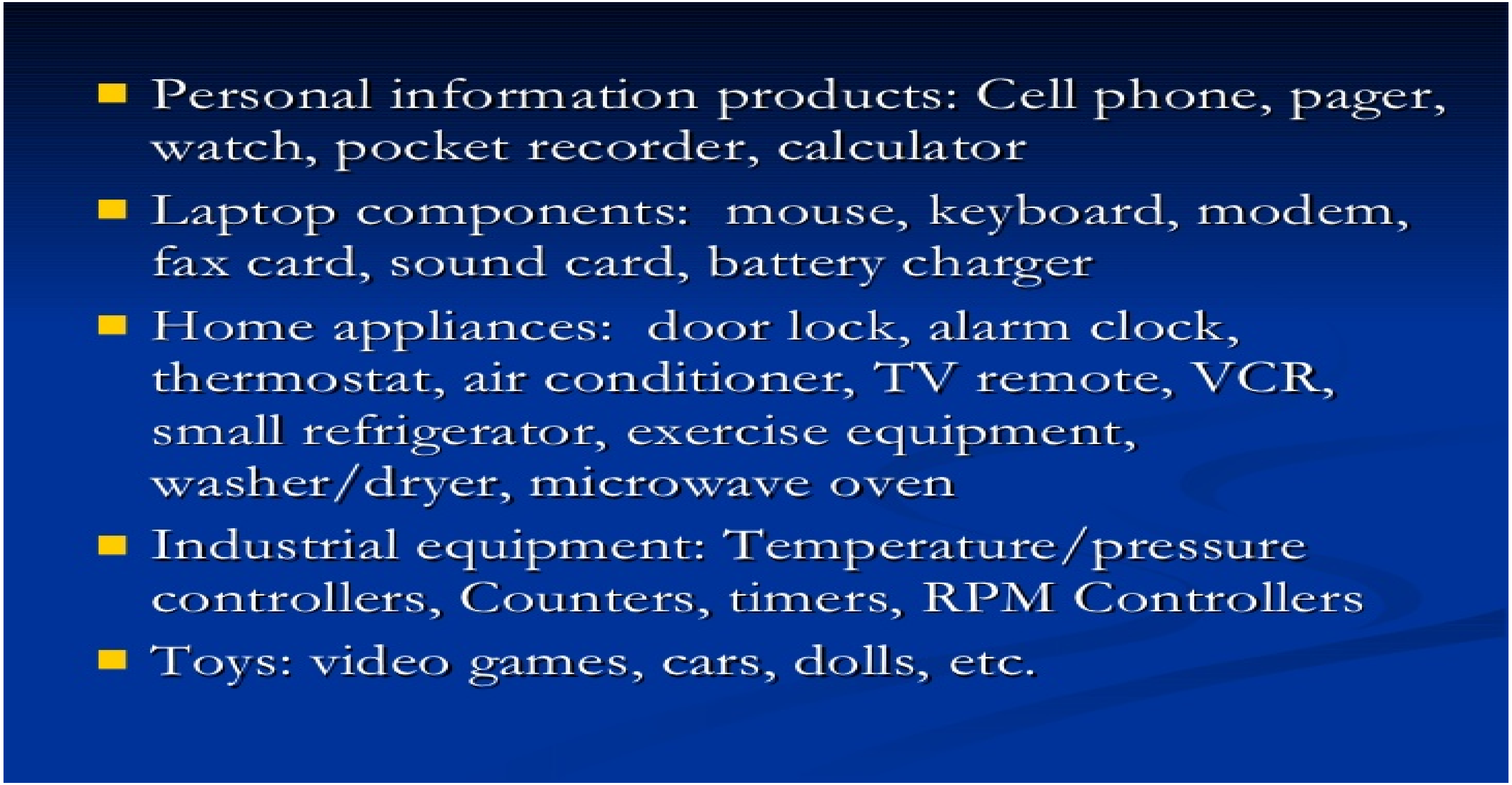
How is it different from a Microprocessor ??

- General-purpose microprocessor

CPU for Computers
No RAM, ROM, I/O on CPU chip itself
Example ： Intel's x86, Motorola's 680x0

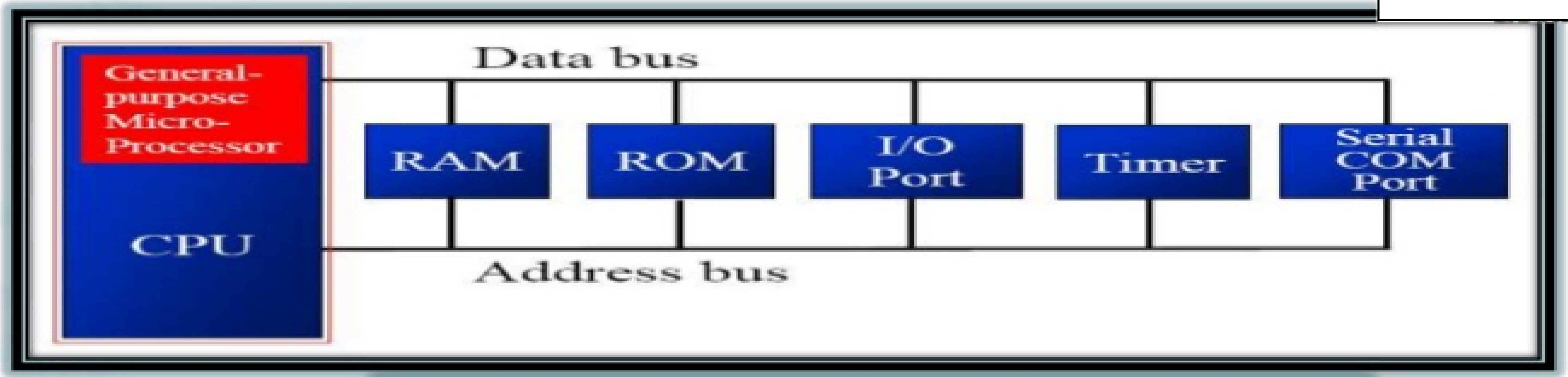


APPLICATIONS OF MICROCONTROLLER



- Personal information products: Cell phone, pager, watch, pocket recorder, calculator
- Laptop components: mouse, keyboard, modem, fax card, sound card, battery charger
- Home appliances: door lock, alarm clock, thermostat, air conditioner, TV remote, VCR, small refrigerator, exercise equipment, washer/dryer, microwave oven
- Industrial equipment: Temperature/pressure controllers, Counters, timers, RPM Controllers
- Toys: video games, cars, dolls, etc.

Microprocessor Vs Microcontroller:



Data bus

General-purpose Micro-Processor

CPU

RAM

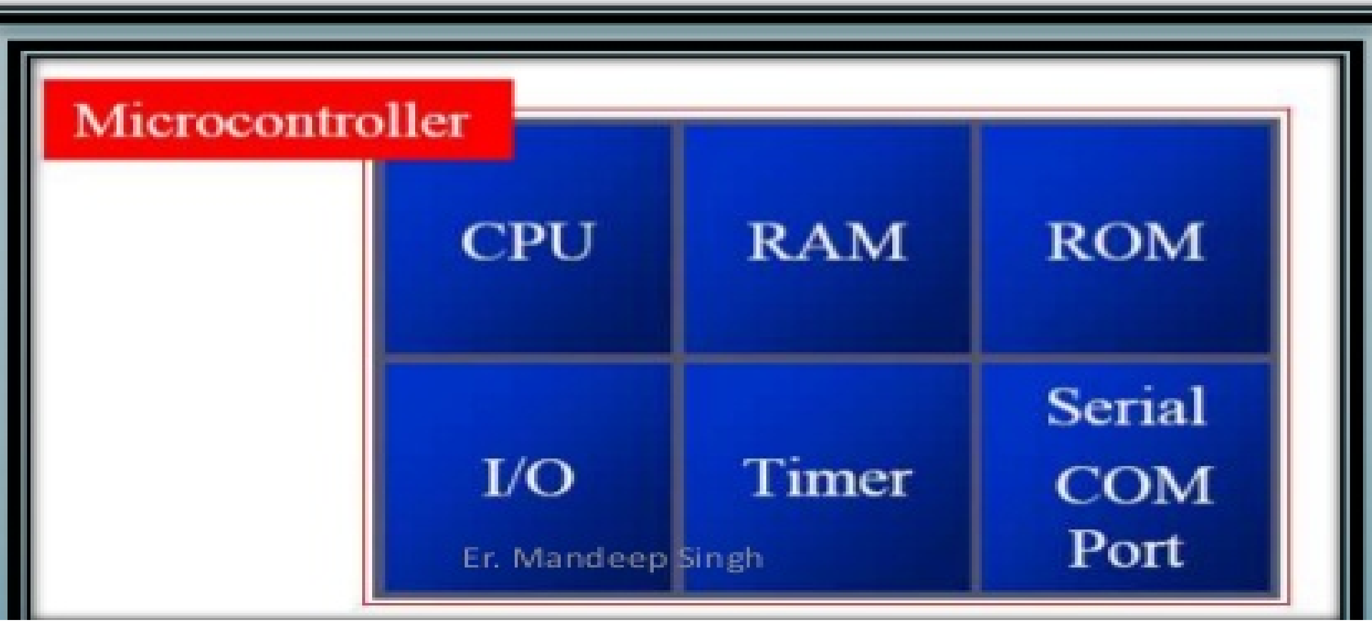
ROM

I/O Port

Timer

Serial COM Port

Address bus



Microcontroller

CPU

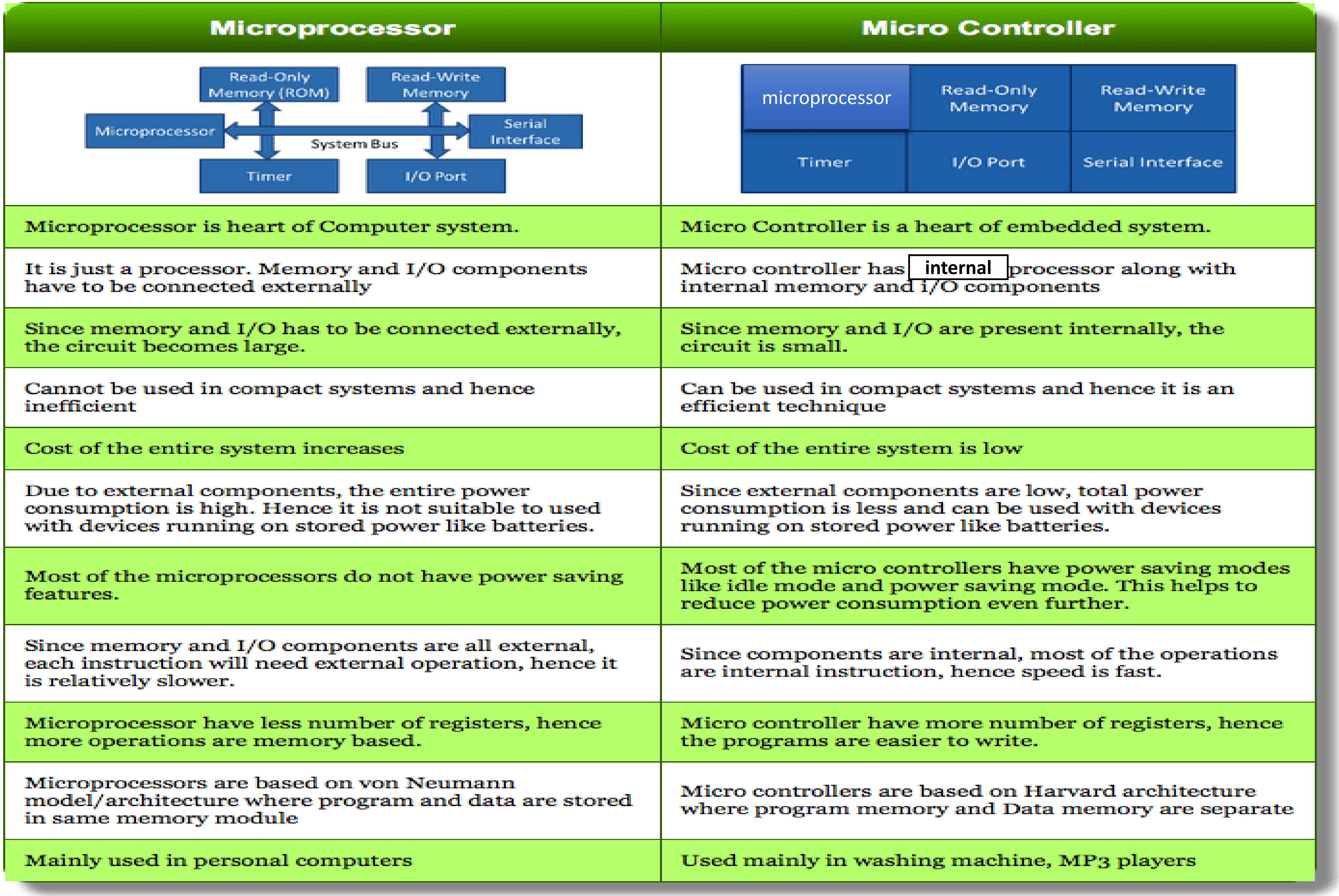
RAM

ROM

I/O

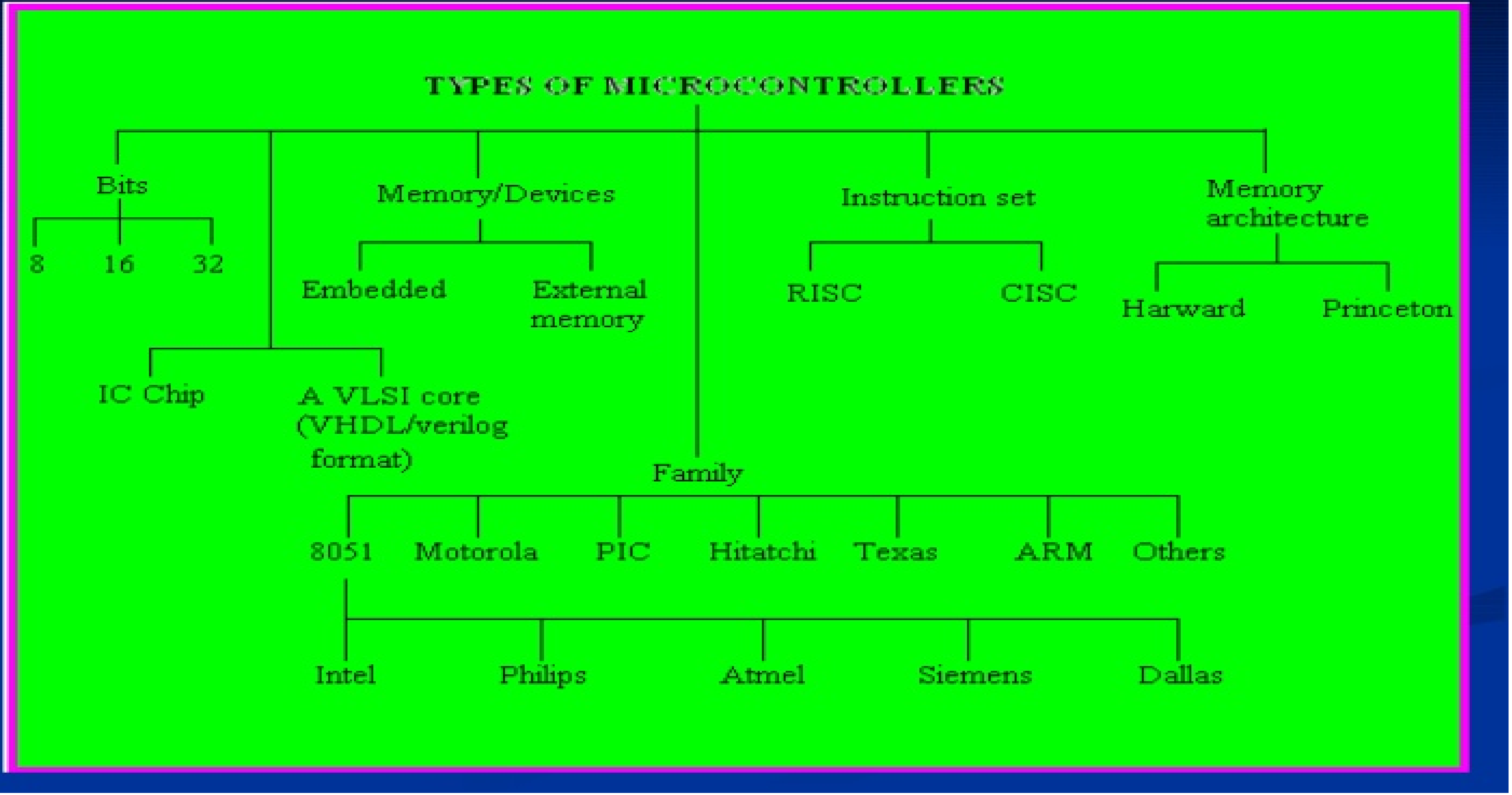
Timer

Serial COM Port



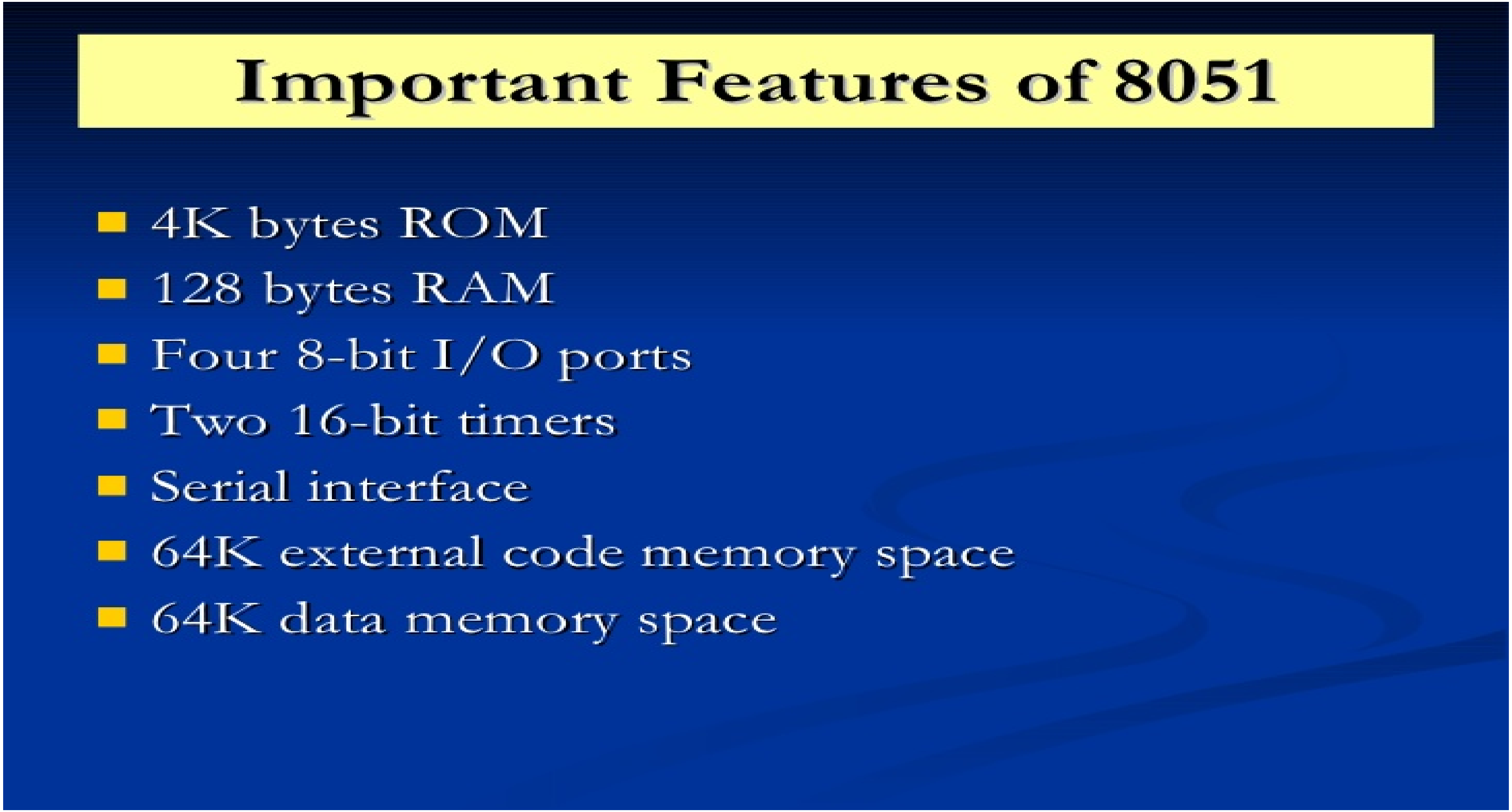
Microprocessor	Micro Controller
Read-Only Memory (ROM) Read-Write Memory Microprocessor System Bus Serial Interface Timer I/O Port	microprocessor Read-Only Memory Read-Write Memory Timer I/O Port Serial Interface
Microprocessor is heart of Computer system.	Micro Controller is a heart of embedded system.
It is just a processor. Memory and I/O components have to be connected externally	Micro controller has internal processor along with internal memory and i/O components
Since memory and I/O has to be connected externally, the circuit becomes large.	Since memory and I/O are present internally, the circuit is small.
Cannot be used in compact systems and hence inefficient	Can be used in compact systems and hence it is an efficient technique
Cost of the entire system increases	Cost of the entire system is low
Due to external components, the entire power consumption is high. Hence it is not suitable to used with devices running on stored power like batteries.	Since external components are low, total power consumption is less and can be used with devices running on stored power like batteries.
Most of the microprocessors do not have power saving features.	Most of the micro controllers have power saving modes like idle mode and power saving mode. This helps to reduce power consumption even further.
Since memory and I/O components are all external, each instruction will need external operation, hence it is relatively slower.	Since components are internal, most of the operations are internal instruction, hence speed is fast.
Microprocessor have less number of registers, hence more operations are memory based.	Micro controller have more number of registers, hence the programs are easier to write.
Microprocessors are based on von Neumann model/architecture where program and data are stored in same memory module	Micro controllers are based on Harvard architecture where program memory and Data memory are separate
Mainly used in personal computers	Used mainly in washing machine, MP3 players

Types of Microcontrollers



TYPES OF MICROCONTROLLERS

- Bits
  - 8
  - 16
  - 32
- IC Chip
- A VLSI core (VHDL/verilog format)
- Memory/Devices
  - Embedded
  - External memory
- Family
  - 8051
    - Intel
    - Philips
    - Atmel
    - Siemens
    - Dallas
  - Motorola
  - PIC
  - Hitatchi
  - Texas
  - ARM
  - Others
- Instruction set
  - RISC
  - CISC
- Memory architecture
  - Harward
  - Princeton



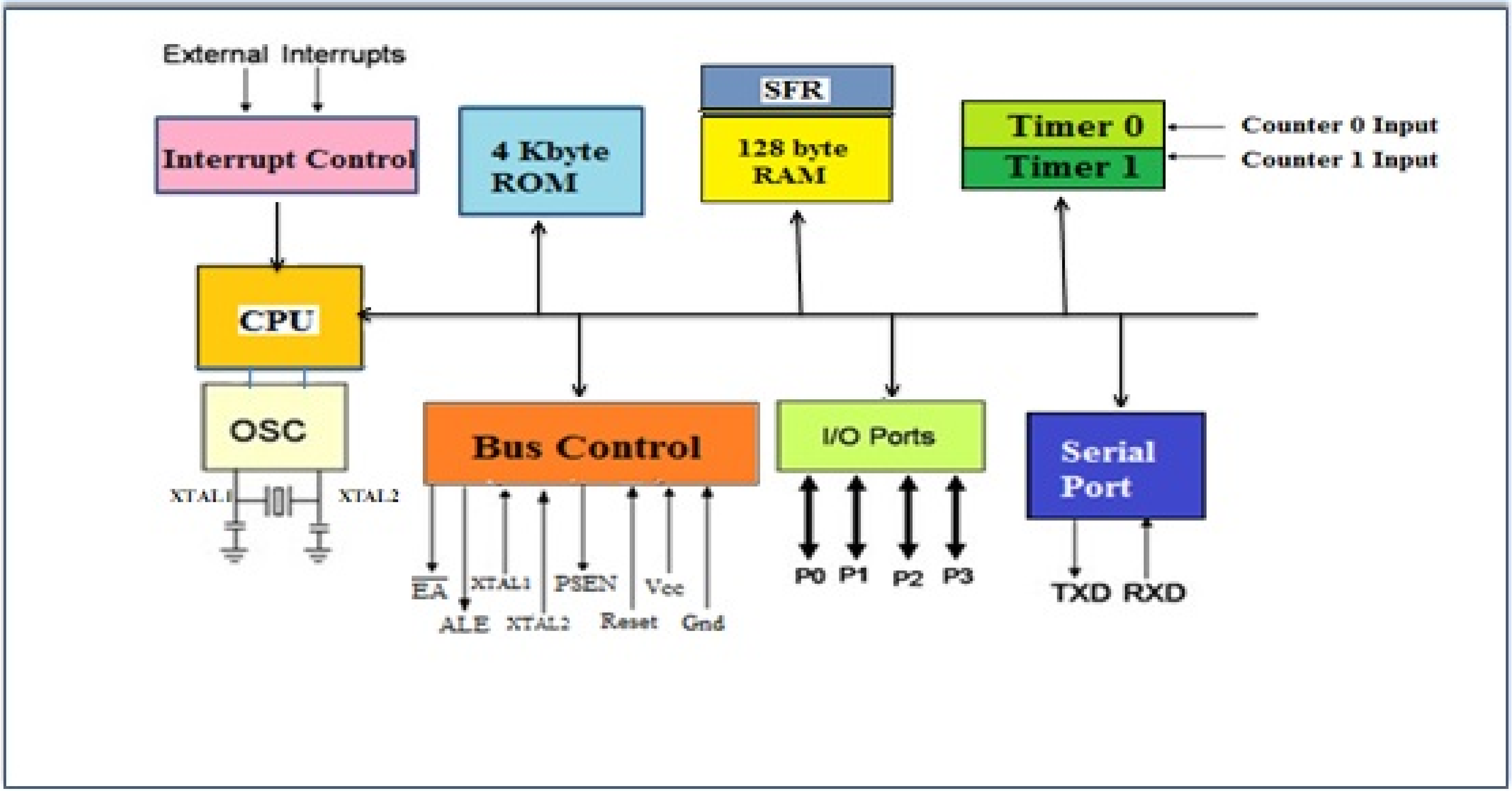
Important Features of 8051

- 4K bytes ROM
- 128 bytes RAM
- Four 8-bit I/O ports
- Two 16-bit timers
- Serial interface
- 64K external code memory space
- 64K data memory space

MCS-51 "Family" of Microcontollers

Feature	8031	8051	8052	8751
ROM	NO	4kB	8kB	4kB UV Eprom
RAM (Bytes)	128	128	256	128
TIMERS	2	2	3	2
I/O PINS	32	32	32	32
SERIAL PORTS	1	1	1	1
INTERRUPT SOURCES	6	6	8	6

Block Diagram of 8051



- 8051 microcontroller is designed by Intel in 1981. It is an 8-bit microcontroller. It is built with 40 pins DIP (dual inline package), 4kb of ROM storage and 128 bytes of RAM storage, 2 16-bit timers.
- It consists of are four parallel 8-bit ports, which are programmable as well as addressable as per the requirement.
- An on-chip crystal oscillator is integrated in the microcontroller having crystal frequency of 12 MHz

- 32 I/O Pins (Input / Output Pins) – Arranged as 4 Ports: P0, P1, P2 and P3.
- 8- bit Stack Pointer (SP) and Processor Status Word (PSW).
- 16 – bit Program Counter (PC) and Data Pointer (DPTR).
- Two 16 – bit Timers / Counters – T0 and T1.
- Control Registers – SCON, PCON, TCON, TMOD, IP and IE.
- Serial Data Transmitter and Receiver for Full – Duplex Operation – SBUF.
- Interrupts: Two External and Three Internal.
- Oscillator and Clock Circuit.

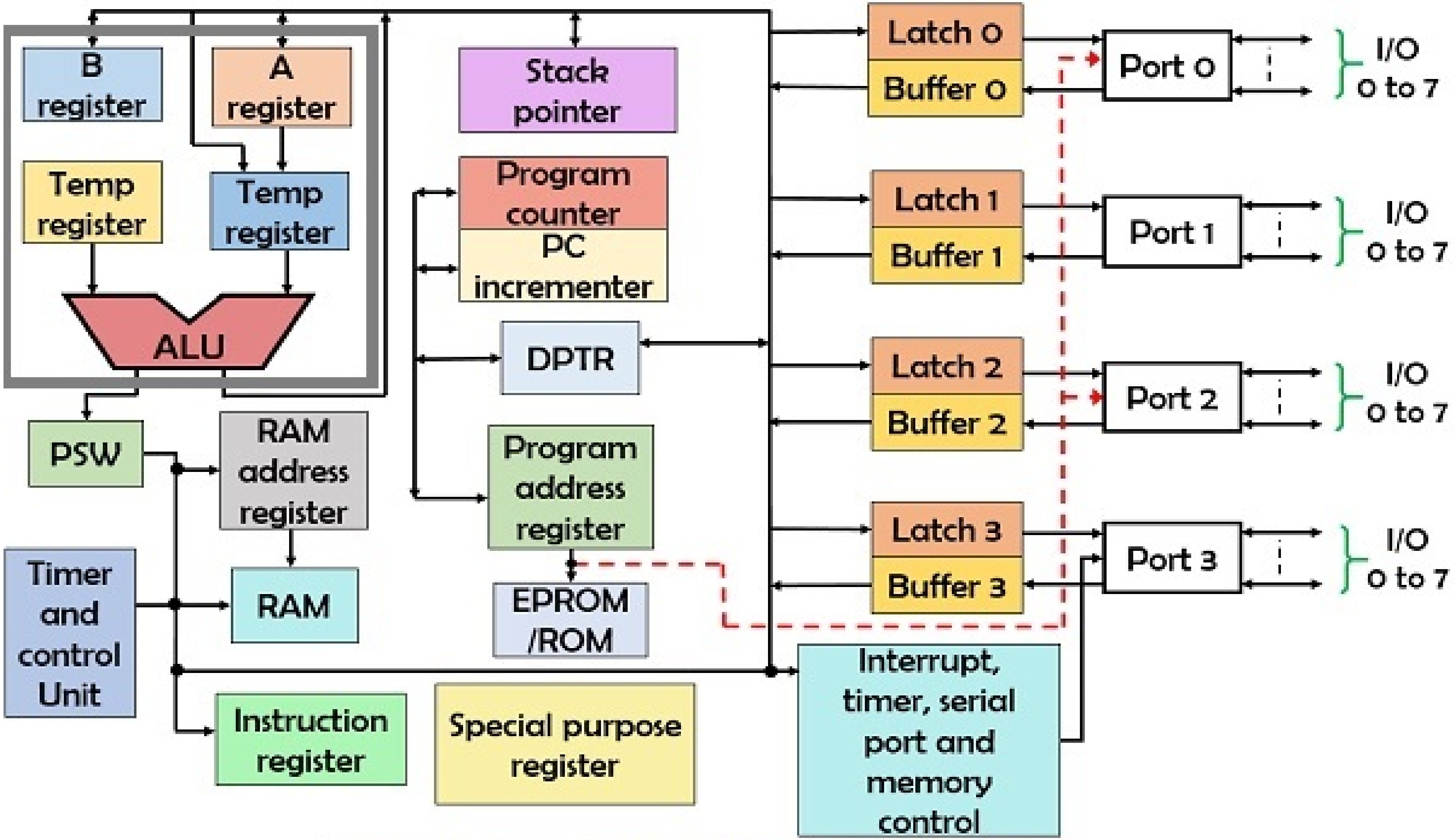
- **CPU (Central Processing Unit)**
- It is the heart of the Microcontroller that mainly comprises of an Arithmetic Logic Unit (ALU) and a Control Unit (CU) and other important components. The CPU is the primary device in communicating with peripheral devices like Memory, Input and Output.
- **Clock Generator (Oscillator)**
- A clock signal allows the operations inside the microcontroller and other parts to be synchronous. A Clock Generator is an integral part of the Microcontroller's Architecture and the user has to provide an additional Timing Circuit in the form of a Crystal.

- **Input and Output Ports**
- I/O Ports or Input / Output Ports provide the microcontroller, a physical connection to the outside world. Input Ports provide a gateway for passing on the data from the outside world with the help of sensors.
- The data from the input ports is manipulated (depending on the application) and will determine the data on the output port.
- Output Ports allow microcontroller to control external devices (like motors and LEDs). Generally, all ports in microcontrollers have dual functionality i.e. they can act as both input and output port (not at the same time though).

- **Memory –**
- A Microcontroller needs program memory to store program/instructions to perform defined tasks. This memory is termed as ROM. Furthermore the Microcontroller also requires data memory to store the operands/data on a temporary basis. This memory is known as RAM. The 8051 Microcontroller is built with 4 Kb on-chip Read Only Memory (ROM) and 128 bytes Random Access Memory (RAM).
- **Address Bus –**
- A bus of the Microcontroller can be defined as a group of wire which can act as a medium for the transfer of data. There are two buses present in the 8051 Microcontroller. While we are already aware of the Data Bus, let us know about the Address Bus of the 8051 Microcontroller. The address bus, which is used to address memory locations, is 16-bit wide. Furthermore, the address bus can also be used to transfer data from the CPU (Central Processing Unit) to the memory. Hence, for obvious reasons the address bus is unidirectional.

- **Interrupts –**
- The most powerful attribute of the 8051 Microcontroller is the concept of Interrupts. The interrupt is a mechanism to –
- **Temporarily suspend the ongoing program,**
- **Pass the control to a subroutine,**
- **Execute the subroutine,**
- **Resume the ongoing/main program.**
- Interrupts can be of various types, such as, Software and Hardware interrupts, Non-maskable and maskable interrupts, etc. Now the 8051 Microcontroller incorporates five interrupts. These are :
- **INT0** – External Hardware Interrupt.
- **TF0** – Timer 0 Overflow Interrupt.
- **INT1** – External Hardware Interrupt.
- **TF1** – Timer 1 Overflow Interrupt.
- **R1/T1** – Serial communication Interrupt.

- **Input/Output Ports –**
- The 8051 Microcontroller needs to be connected to the peripheral devices in order to control their operations. The I/O Ports are responsible for the connection of the Microcontroller to its peripheral devices. There are total Four 8-bit Input/Output Ports present in this Microcontroller.
- Additionally, these are some important features of 8051 microcontroller given as follows :
- **Two 16-bit Timers and Counters.**
- **A Data Pointer and a Program Counter of 16-bit each.**
- **128 User defined Flags.**
- **Four Register banks.**
- **31 General Purpose Registers which are of 8-bit each.**
- **Pin diagram of 8051 Microcontroller –**



Architecture of 8051 Microcontroller

Electronics Desk

Operation of each block:-

- ACCUMULATOR(ACC):- it is used for data transfer and arithmetic operations. After any operation result is stored in ACC and can be accessed through its SFR address of 0E0H.

- B REGISTER:- it is used to store the upper 8 bit result of multiplication and divisions. It is used as temporary register and can be accessed through its SFR address of 0F0H.

. **ALU:**

It is 8 bit unit. It performs arithmetic operation as addition, subtraction, multiplication, division,
increment and decrement

. It performs logical operations like AND, OR and EX-OR operations

.Program counter(PC):

The Program Counter (PC) is a 2-byte address which tells the 8051 where the next instruction to execute is found in memory.

It is used to hold 16 bit address of internal RAM, external RAM or external ROM locations.

When the 8051 is initialized PC always starts at 0000h and is incremented each time an instruction is executed.

It is important to note that PC isn't always incremented by one and never decremented.

Operation of each block:-

- STACK POINTER(SP):- This is an 8 bit register. SP is incremented before the data is stored onto the stack using PUSH/CALL instructions execution. During PUSH, first SP is incremented and then copy the data. In the POP operation, initially copy the data and then decrement the SP.

- DATA POINTER(DTPR):- DTPR is a 16 bit register. It consists of higher byte (DPH) and a lower byte (DPL). DPTR is very useful for string operations and look up table operations. With a 16 bit DPTR, a maximum of 64K of off chip data memory and 64k of off chip memory can be addressed.

- **Stack pointer(SP):** It is 8-bit register. It is byte addressable. Its address is 81H. It is used to hold the internal RAM memory location addresses which are used as stack memory. When the data is to be placed on stack by push instruction, the content of stack pointer is incremented by 1, and when data is retrieved from stack, content of stack of stack pointer is decremented by 1.

Internal RAM

Internal RAM has memory 128-byte. See 8051 hardware for further internal RAM design.

Internal RAM is organized into three distinct areas: 32 bytes working registers from address 00h to 1Fh 16 bytes bit

addressable occupies RAM byte address 20h to 2Fh, altogether 128 addressable bits General purpose RAM from 30h to 7Fh.

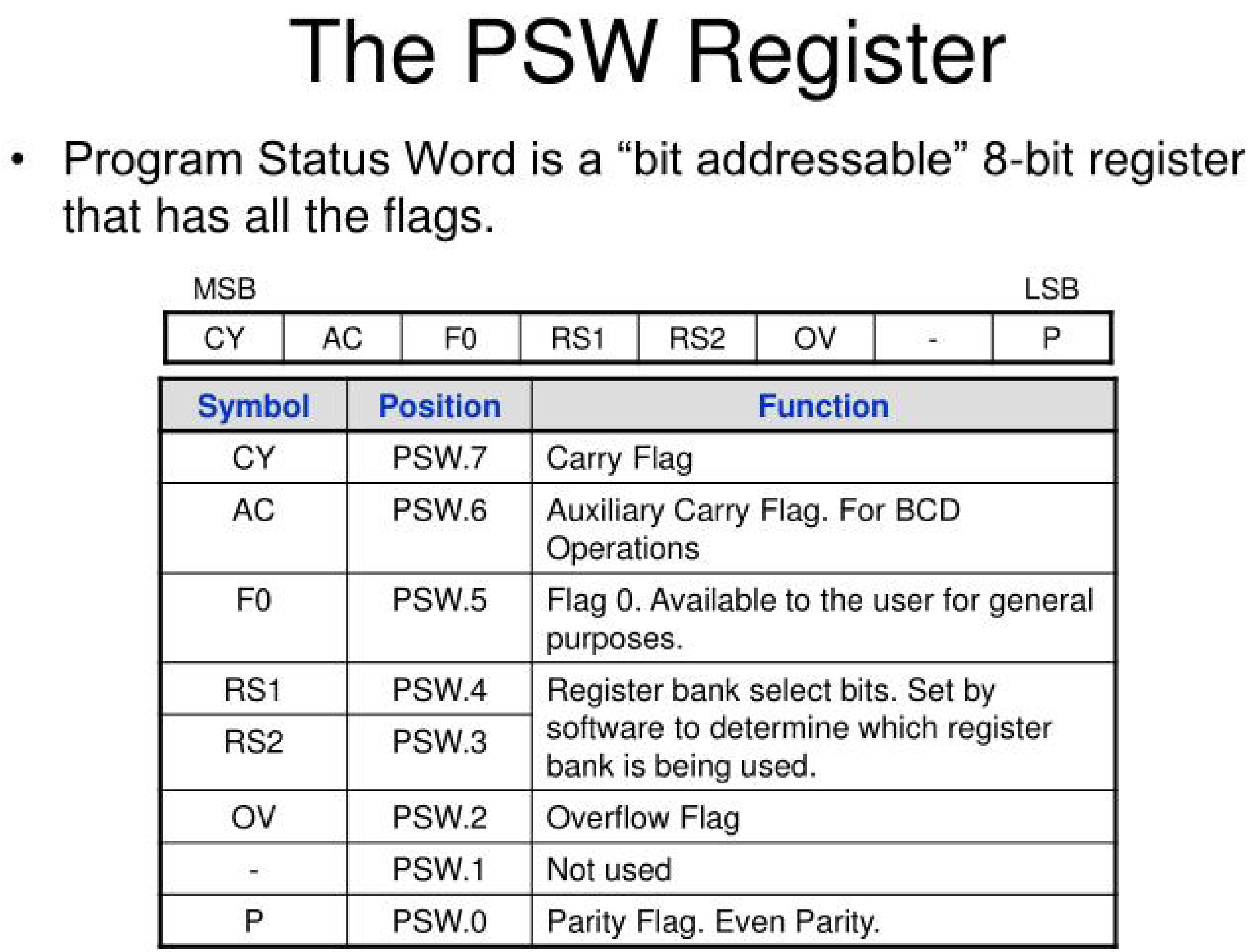
Internal ROM

Data memory and program code memory both are in different physical memory but both have the same addresses.

An internal ROM occupied addresses from 0000h to 0FFFh. PC addresses program codes from 0000h to 0FFFh.

Program addresses higher than 0FFFh that exceed the internal ROM capacity will cause 8051 architecture to

fetch codes bytes from external program memory.



The PSW Register

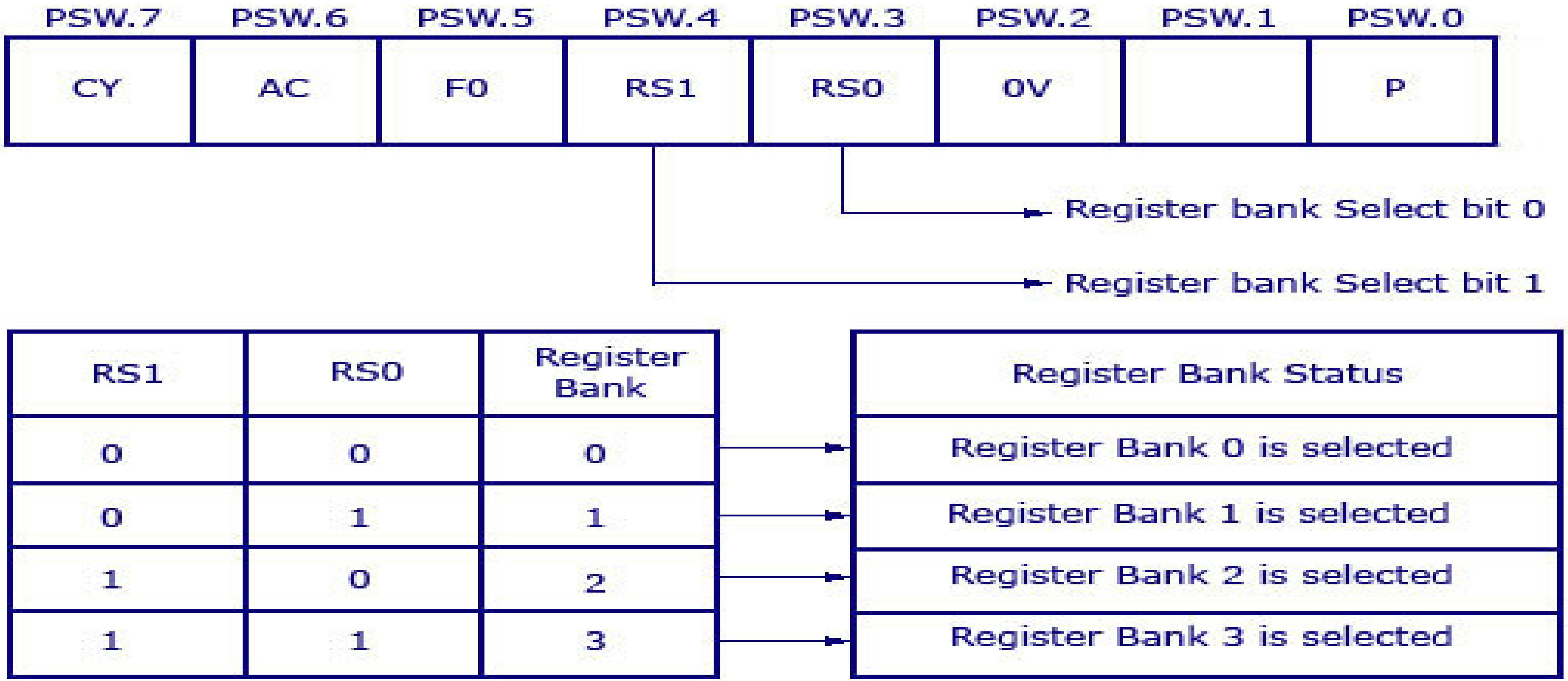
- Program Status Word is a “bit addressable” 8-bit register that has all the flags.

MSB							LSB
CY	AC	F0	RS1	RS2	OV	-	P

Symbol	Position	Function
CY	PSW.7	Carry Flag
AC	PSW.6	Auxiliary Carry Flag. For BCD Operations
F0	PSW.5	Flag 0. Available to the user for general purposes.
RS1	PSW.4	Register bank select bits. Set by software to determine which register bank is being used.
RS2	PSW.3	
OV	PSW.2	Overflow Flag
-	PSW.1	Not used
P	PSW.0	Parity Flag. Even Parity.

- Program status word (PSW): this is a special function register and consists of different status bits that reflect the current state of microcontroller.
- It contains carry(CY) , axillary carry(AC) and two registors bank select bits (RS1 and RS0), over flow flag(OV) , a parity bit(P) and 2 user defined status flags.

Processor Status Word



PSW.7	PSW.6	PSW.5	PSW.4	PSW.3	PSW.2	PSW.1	PSW.0
CY	AC	F0	RS1	RS0	0V		P

Register bank Select bit 0

Register bank Select bit 1

RS1	RS0	Register Bank
0	0	0
0	1	1
1	0	2
1	1	3

Register Bank Status
Register Bank 0 is selected
Register Bank 1 is selected
Register Bank 2 is selected
Register Bank 3 is selected

www.CircuitsToday.com

Oscillator and clock generator:

All operations in a microcontroller are synchronized by the help of an oscillator clock.

The oscillator clock generates the clock pulses by which all internal operations are synchronized.

A resonant network connected through pins XTAL1 and XTAL2 forms up an oscillator.

For this purpose a quartz crystal and capacitors are employed.

The crystal run at specified maximum and minimum frequencies typically at 1 MHz to 16 MHz.

Special function Registers(SFR):

8051 microcontroller has 11 SFR divided in 4 groups:

A. Timer/Counter register: 8051 microcontroller has 2-16 bit Timer/counter registers called Timer-reg-T0 And
Timer/counter Reg-T1.Each register is 16 bit register divide into lower and higher byte register as shown below:

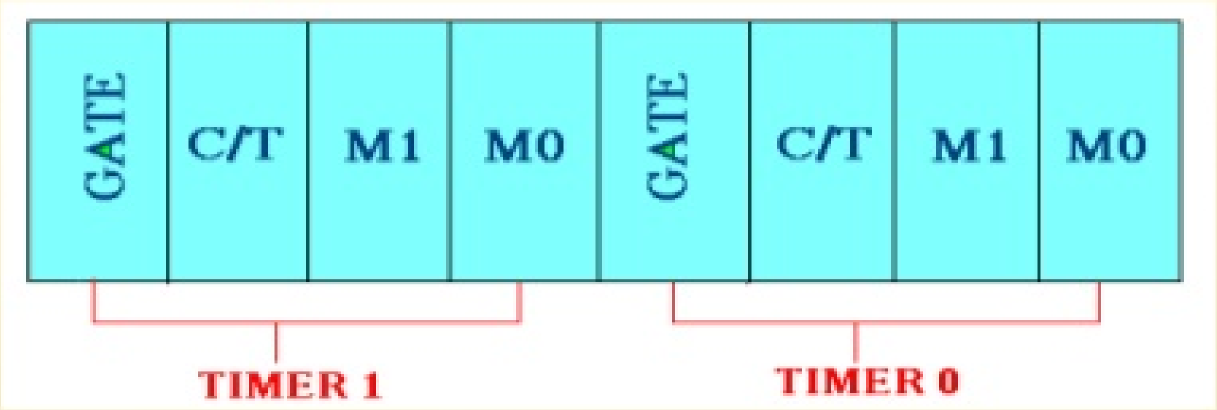
These register are used to hold initial no. of count. All of the 4 register are byte addressable.

1. **Timer control register:** 8051 microcontroller has two 8-bit timer control register i.e. TMOD and TCON register.

   1) TMOD Register: it is 8-bit register. Its address is 89H. It is byte addressable.
   It used to select mode and control operation of time by writing control word.

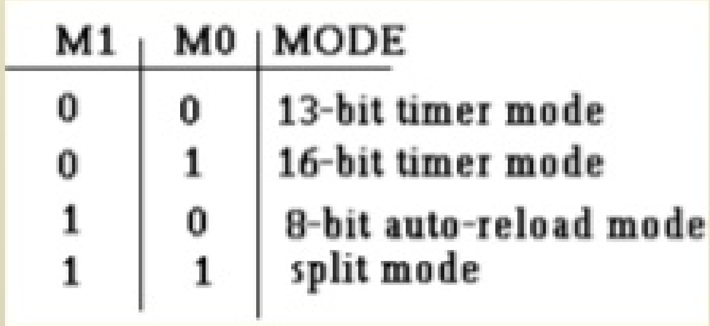
   2). TCON register: It is 8-bit register. Its address is 88H. It is byte addressable.
   Its MSB 4-bit are used to control operation of timer/ counter and LSB 4-bit are used for external interrupt control.

TMOD Register:



GATE	C/T	M1	M0	GATE	C/T	M1	M0
TIMER 1				TIMER 0			

- **Gate :** ***When set, timer only runs while INT(0,1) is high.***
- ***C/T :*** ***Counter/Timer select bit.***
- ***M1 :*** ***Mode bit 1.***
- ***M0 :*** ***Mode bit 0.***



M1	M0	MODE
0	0	13-bit timer mode
0	1	16-bit timer mode
1	0	8-bit auto-reload mode
1	1	split mode

TCON Register:

TF1	TR1	TF0	TR0	IE1	IT1	IE0	IT0

- *TF: Overflow flag*
  - *Set by hardware on Timer/Counter overflow*
  - *Cleared by hardware when processor vectors to interrupt routine*
- *TR: Run control bit*
  - *Set/Cleared by software to turn Timer/Counter on/off*
- *IE: Interrupt Edge flag*
  - *Set by hardware when external interrupt edge detected*
  - *Cleared when interrupt processed*
- *IT: Interrupt Type control bit*
  - *Set/Cleared by software to specify*
    falling edge/low level triggered external interrupts

- ***TF1: Timer 1 overflow flag.*** ***TR1: Timer 1 run control bit.***
- ***TF0: Timer 0 overflag.*** ***TR0: Timer 0 run control bit.***
- ***IE1: External interrupt 1 edge flag.*** ***IT1: External interrupt 1 type flag.***
- ***IE0: External interrupt 0 edge flag.*** ***IT0: External interrupt 0 type flag.***

2.. Serial data register: 8051 micro controller has 2 serial data register viz. SBUF and SCON.

1. Serial buffer register (SBUF): it is 8-bit register. It is byte addressable .

Its address is 99H. It is used to hold data which is to be transferred serially.

2. Serial control register (SCON): it is 8-bit register. It is bit/byte addressable.

Its address is 98H. The 8-bit loaded into this register controls the operation

of serial communication.

3. Interrupt register: 8051 µC has 2 8-bit interrupt register

.1. Interrupt enable register (IE): it is 8-bit register. It is bit/byte addressable. Its address is A8H.

it is used to enable and disable function of interrupt.

2. Interrupt priority register (IP): It is 8-bit register. It is bit/byte addressable.

Its address is B8H.it is used to select low or high level priority of each individual interrupts.

4. Power control register (PCON): it is 8-bit register. It is byte addressable .Its address is 87H.

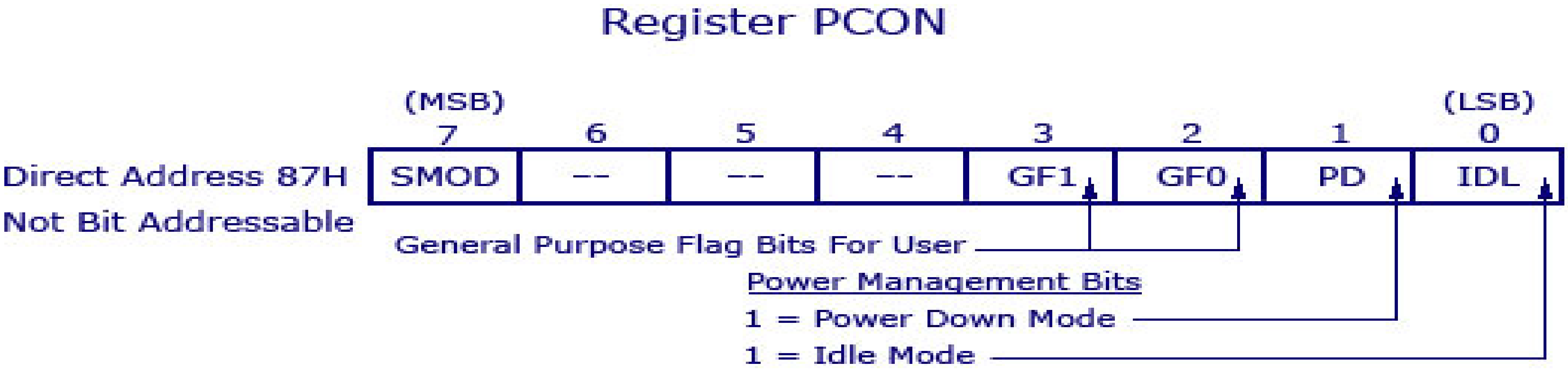
its bits are used to control mode of power saving circuit, either idle or power down mode

and also one bit is used to modify baud rate of serial communication.

4. Power control register (PCON): it is 8-bit register. It is byte addressable .Its address is 87H.

its bits are used to control mode of power saving circuit, either idle or power down mode

and also one bit is used to modify baud rate of serial communication.



Register PCON

	(MSB) 7	6	5	4	3	2	1	(LSB) 0
Direct Address 87H Not Bit Addressable	SMOD	--	--	--	GF1	GF0	PD	IDL

General Purpose Flag Bits For User

Power Management Bits

1 = Power Down Mode

1 = Idle Mode

www.CircuitsToday.com

Operation of each block:-

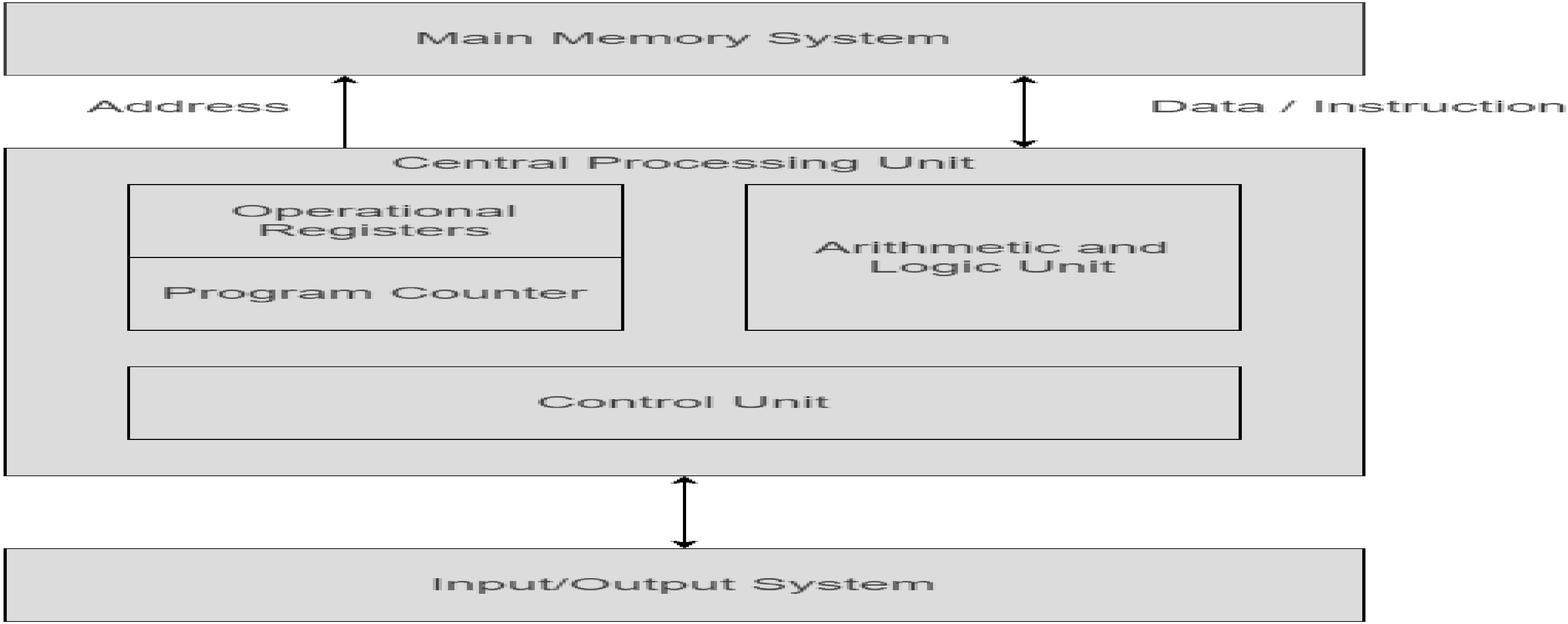
PORT0, PORT1, PORT2, PORT3 LATCHES AND DRIVERS:- Each latch and corresponding drivers of port 0-3 is allotted to the corresponding on chip I/O port.

Difference between Von Neumann and Harvard Architecture

- **Von Neumann Architecture**

- Von Neumann Architecture is a digital computer architecture whose design is based on the concept of stored program computers where program data and instruction data are stored in the same memory. This architecture was designed by the famous mathematician and physicist **John Von Neumann** in 1945.

Von Neumann Architecture

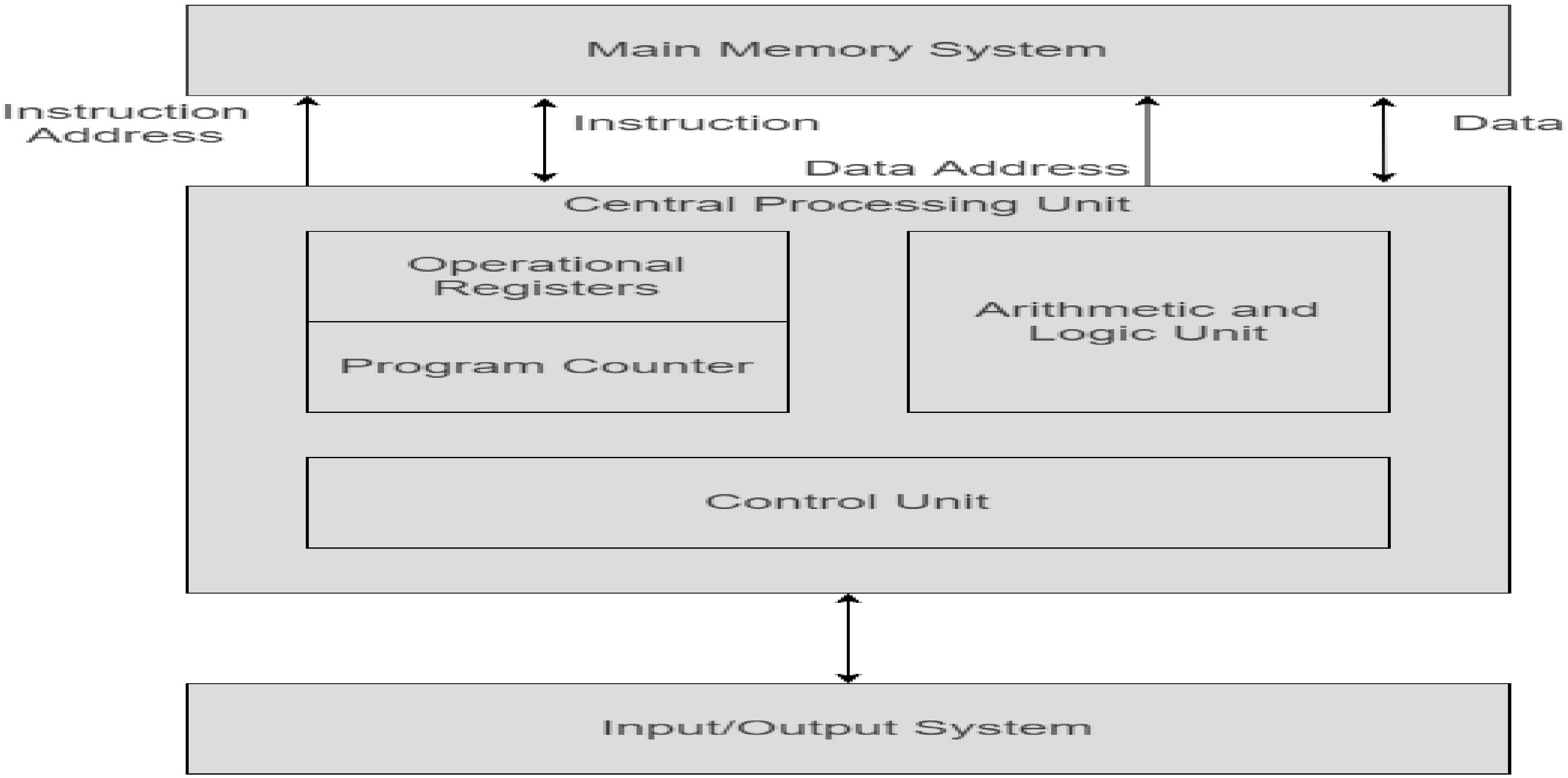


Von Neumann Architecture

Harvard Architecture

- Harvard Architecture is the digital computer architecture whose design is based on the concept where there are separate storage and separate buses (signal path) for instruction and data. It was basically developed to overcome the bottleneck of Von Neumann Architecture.

Harvard Architecture



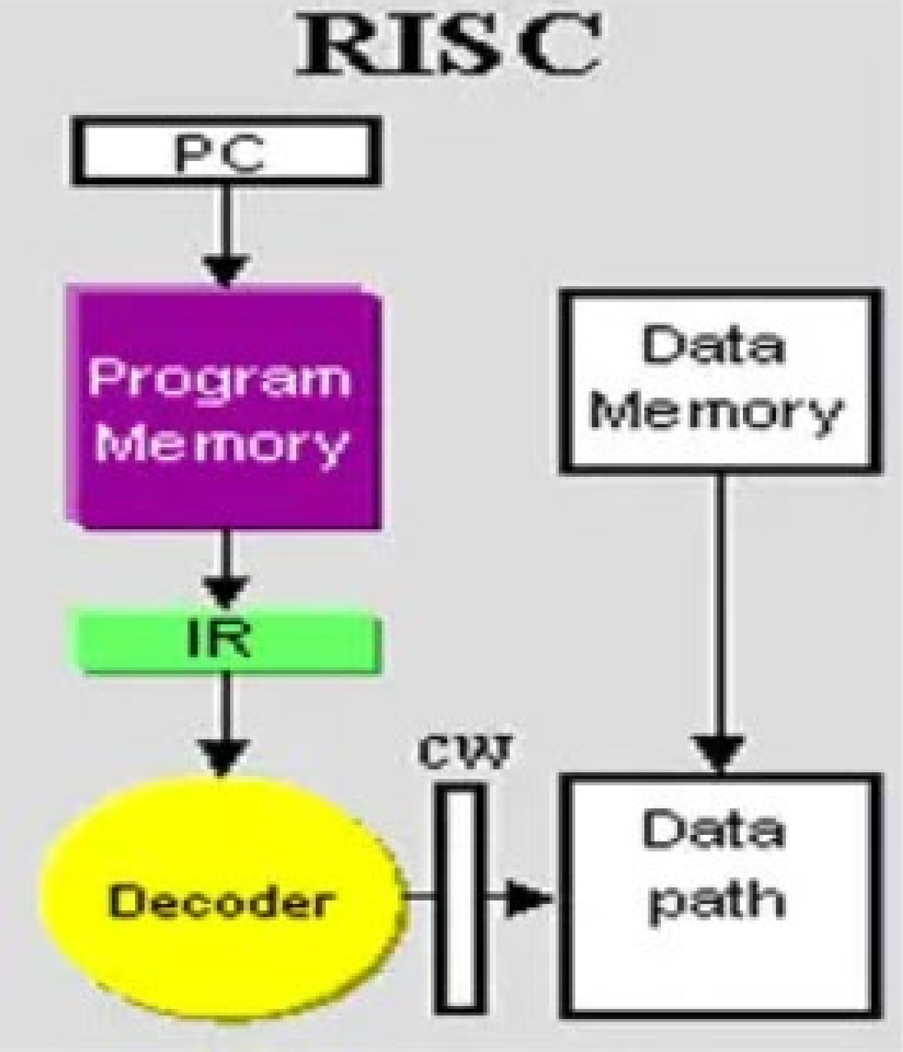
Harvard Architecture

Difference between Von Neumann and Harvard Architecture :

It is ancient computer architecture based on stored program computer concept.	It is modern computer architecture based on Harvard Mark I relay based model.
Same physical memory address is used for instructions and data.	Separate physical memory address is used for instructions and data.
There is common bus for data and instruction transfer.	Separate buses are used for transferring data and instruction.
Two clock cycles are required to execute single instruction.	An instruction is executed in a single cycle.
It is cheaper in cost.	It is costly than van neumann architecture.
CPU can not access instructions and read/write at the same time.	CPU can access instructions and read/write at the same time.
It is used in personal computers and small computers.	It is used in micro controllers and signal processing.

RISC Architecture Basics

- The word RISC stands for ‘Reduced Instruction Set Computer’.
- It is such a design of the CPU that follows simple instructions and is really speedy.
- Basically, it is a subset of a number of instructions. In simple words, each command performs a really simple and small jobs.
- In such a computer, the set of instructions is simple and easy to implement.
- Therefore, it becomes easy to implement such commands that are really complex and difficult to execute as single instructions. Every instruction is of almost the same length.
- In short, it divides complex instructions into simple instructions using Piplelining. Pipelining is a multi stage process to execute instructions.
- Normally,it can execute a single instruction in one machine cycle.



Simple instructions
One instruction = One CW
PM longer

Pipelining in RISC

- This method is a pipelining which is mainly increase the speed of the RISC machines.
- It is a very crucial technique. Reduced Instruction Set Computer is a Architecture which is designed in such a way that it carries out only a few commands in parallel simultaneously. Due to the small size if the instructions, the chips used in this sort of architecture need a very few number of transistors.
- In RISC very less decoding is required. Plus, the data types in the hardware are also less. The general purpose register is the same one for all.
- The instruction set is uniform. And the addressing nodes are really simple.
- Finally when a job is being performed, RISC saves the number of cycles in which it is being executed by eliminating the unnecessary part of the code.

Characteristic of RISC Architecture

- There are a lot of characteristics related to the RISC architecture, some of them are as follows:

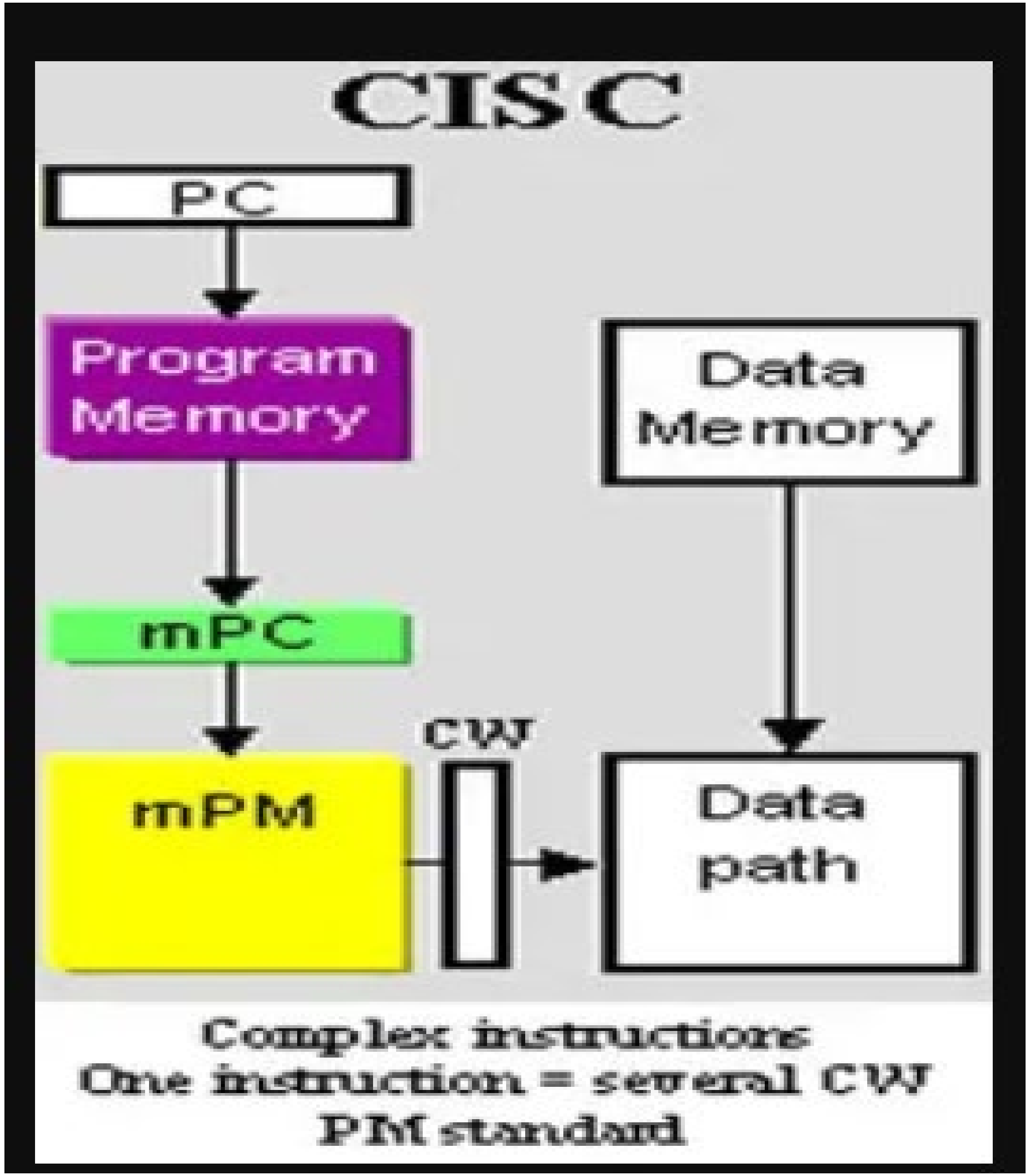
1. Simple set of instructions which are easy to decode and implement.
2. The size of one instruction comes under the size of a single word.
3. Only one clock cycle is required to execute a single instruction, so it is a fast process.
4. The quantity of general purpose register is greater.
5. The addressing modes are quite simple.
6. The variable data types are very less.
7. Its main idea is to achieve pipelining

.

- Example – Let’s suppose we are to perform addition operation on two 8-bit numbers:
- The load command will be used to load the data in the registers and then the addition operator will be used on them and the result will be stored in the location of the output.
-

CISC Architecture Introductin

- The word CISC is abbreviated as ‘’Complex Instruction Set Computer”.
- It is such a design of the CPU that executes a job using only a single command. The command contains multi-step operations that program want to execute.
- Moreover, CISC machines have relatively smaller programs.
- Whereas, the number of compound instruction size is huge.
- Therefore, it requires a lot of time in execution. In this type of architecture, each instruction set is very well protected in various steps.
- This means that there are extra three hundred instructions related to each set of instruction. Due to this, the instructions take time in their execution.
- Their time may vary from two to ten machine cycles, depending on the size of the instruction set. Furthermore, CISC architecture doesn’t implement pipelining normally as it is hard to.



Complex instructions
One instruction = several CW
PM standard

Main Features of CISC

- If we see from the prospective of compilers, CISC machines are good for them.
- Because the range of innovative instructions are easily obtained in a single instruction set. They execute the compound instructions in only a single and complex set of instructions.
- CISC is able to get processes at low-level. Hence, it is easier this way to have addressing nodes that are huge and a lot of different data types in machine hardware.
- Despite of all this CISC works less efficiently than the way RISC works.
- This is mainly because CISC is unable to remove the portion of the code that is not required and so a lot of cycles are wasted by them when the instruction set is executed.
- Plus, their microprocessor chips are very difficult to manufacture and program. They are really complex.

Characteristic of CISC

- There are a lot of characteristics related to the CISC architecture, some of them are as follows:

1. The instruction set is complex. Hence. is its decoding.
2. Instructions are normally large due to their complexity. Instructions are normally bigger than one word size.
3. Usually, the compound instructions take greater time than a single clock cycle in their execution.
4. The number of general purpose registers are less. Because this, it performs most operations in the memory itself.
5. The addressing modes are normally complex.
6. The data types are numerous.

- Example – Let’s suppose we are to perform addition operation on two 8-bit numbers:
- Only one instruction is used for the execution of this operation. The ADD operation will simply perform the required task. All the tasks will be done by this single command.

RISC vs CISC

This very equation is normally used to check the performance of any compute

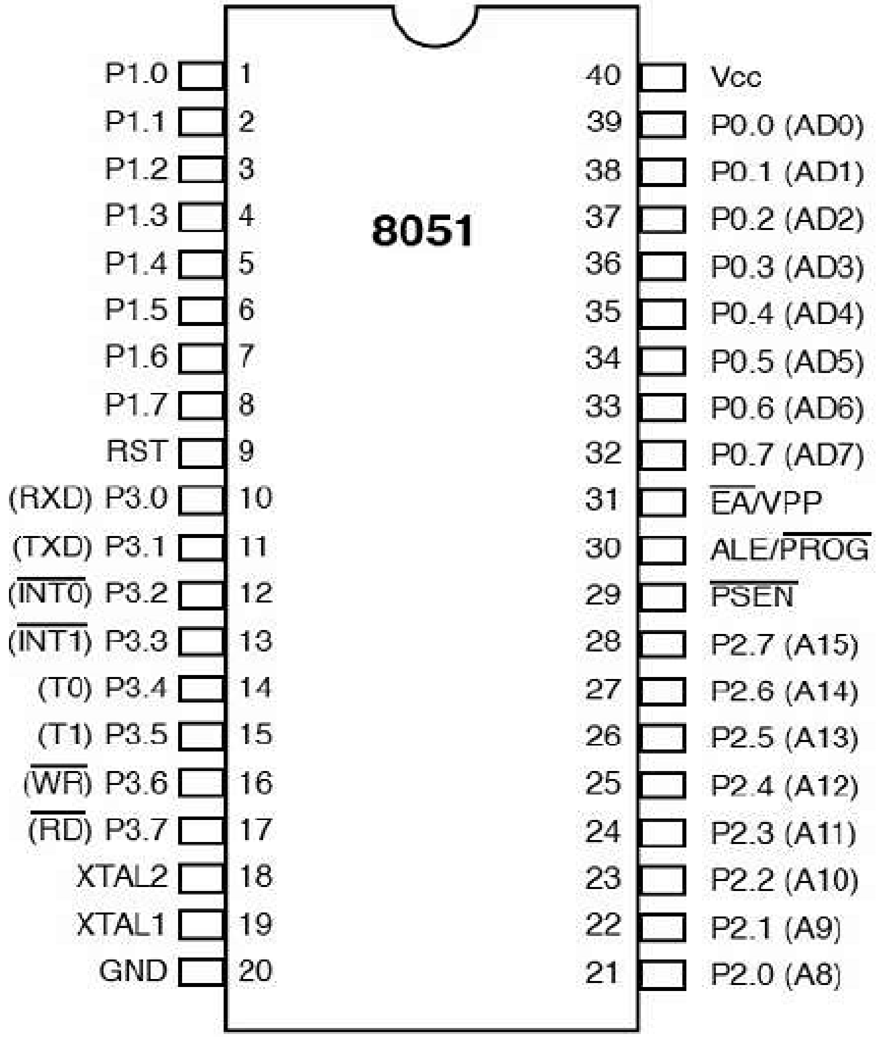
$$\frac{\text{time}}{\text{program}} = \frac{\text{time}}{\text{cycle}} \times \frac{\text{cycles}}{\text{instruction}} \times \frac{\text{instructions}}{\text{program}}$$

This formula clearly tells that the performance of a RISC based architecture is way better than the one operating using CISC architecture. CISC and RISC are two entirely different types of computer architectures. Some of their differences are as follows:

Difference between RISC and CISC Comparison Chart

CISC	RISC
Memory unit is present to implement the instructions	There is no memory unit and registers store data
It is microprogramming unit	It has a complex design of compiler
Its compiler design is easy	Compiler design is complex
Its calculations are slower yet precise	Perform mathematical calculations faster
Their decoding is difficult	Decoding of its instructions is easier
Instructions are complex so it takes time in execution	It is faster as its instructions are simple
External memory mandatory requirement	No external memory requirment
Pipelining is difficult to implement	Pipelining is easy to implement
Their processors often stall	There is no stalling normally
Code expansion is easier	The expansion of code can be an issue
Utilize more disk space	Consumes less disk space
Its examples include:include VAX, PDP-11, Motorola 68k and your desktop PCs	Its examples include:DEC Alpha, ARC, AMD 29k, Atmel AVR, Intel i860, Blackfin, i960, Motorola 88000, MIPS, PA-RISC, Power, SPARC, SuperH, and ARM

The pin diagram of 8051 microcontroller looks as follows



- **Pins 1 to 8** − These pins are known as Port 1. This port doesn’t serve any other functions. It is internally pulled up, bi-directional I/O port.
- **Pin 9** − It is a RESET pin, which is used to reset the microcontroller to its initial values.
- **Pins 10 to 17** − These pins are known as Port 3. This port serves some functions like interrupts, timer input, control signals, serial communication signals RxD and TxD, etc.
- **Pins 18 & 19** − These pins are used for interfacing an external crystal to get the system clock.
- **Pin 20** − This pin provides the power supply to the circuit.
- **Pins 21 to 28** − These pins are known as Port 2. It serves as I/O port. Higher order address bus signals are also multiplexed using this port.
- **Pin 29** − This is PSEN pin which stands for Program Store Enable. It is used to read a signal from the external program memory.
- **Pin 30** − This is EA pin which stands for External Access input. It is used to enable/disable the external memory interfacing.
- **Pin 31** − This is ALE pin which stands for Address Latch Enable. It is used to demultiplex the address-data signal of port.
- **Pins 32 to 39** − These pins are known as Port 0. It serves as I/O port. Lower order address and data bus signals are multiplexed using this port.
- **Pin 40** − This pin is used to provide power supply to the circuit.

Microcontrollers 8051 Input Output Ports

- 8051 microcontrollers have 4 I/O ports each of 8-bit, which can be configured as input or output. Hence, total 32 input/output pins allow the microcontroller to be connected with the peripheral devices.
- **Pin configuration**, i.e. the pin can be configured as 1 for input and 0 for output as per the logic state.
    - **Input/Output (I/O) pin** − All the circuits within the microcontroller must be connected to one of its pins except P0 port because it does not have pull-up resistors built-in.
    - **Input pin** − Logic 1 is applied to a bit of the P register. The output FE transistor is turned off and the other pin remains connected to the power supply voltage over a pull-up resistor of high resistance.
- **Port 0** − The P0 (zero) port is characterized by two functions −
    - When the external memory is used then the lower address byte (addresses A0A7) is applied on it, else all bits of this port are configured as input/output.
    - When P0 port is configured as an output then other ports consisting of pins with built-in pull-up resistor connected by its end to 5V power supply, the pins of this port have this resistor left out.

- Input Configuration
- If any pin of this port is configured as an input, then it acts as if it “floats”, i.e. the input has unlimited input resistance and in-determined potential.
- Output Configuration
- When the pin is configured as an output, then it acts as an “open drain”. By applying logic 0 to a port bit, the appropriate pin will be connected to ground (0V), and applying logic 1, the external output will keep on “floating”.
- In order to apply logic 1 (5V) on this output pin, it is necessary to build an external pullup resistor.
- **Port 1**
- P1 is a true I/O port as it doesn’t have any alternative functions as in P0, but this port can be configured as general I/O only. It has a built-in pull-up resistor and is completely compatible with TTL circuits.
- **Port 2**
- P2 is similar to P0 when the external memory is used. Pins of this port occupy addresses intended for the external memory chip. This port can be used for higher address byte with addresses A8-A15. When no memory is added then this port can be used as a general input/output port similar to Port 1.
- **Port 3**
- In this port, functions are similar to other ports except that the logic 1 must be applied to appropriate bit of the P3 register.
- Pins Current Limitations
- When pins are configured as an output (i.e. logic 0), then the single port pins can receive a current of 10mA.
- When these pins are configured as inputs (i.e. logic 1), then built-in pull-up resistors provide very weak current, but can activate up to 4 TTL inputs of LS series.
- If all 8 bits of a port are active, then the total current must be limited to 15mA (port P0: 26mA).
- If all ports (32 bits) are active, then the total maximum current must be limited to 71mA.

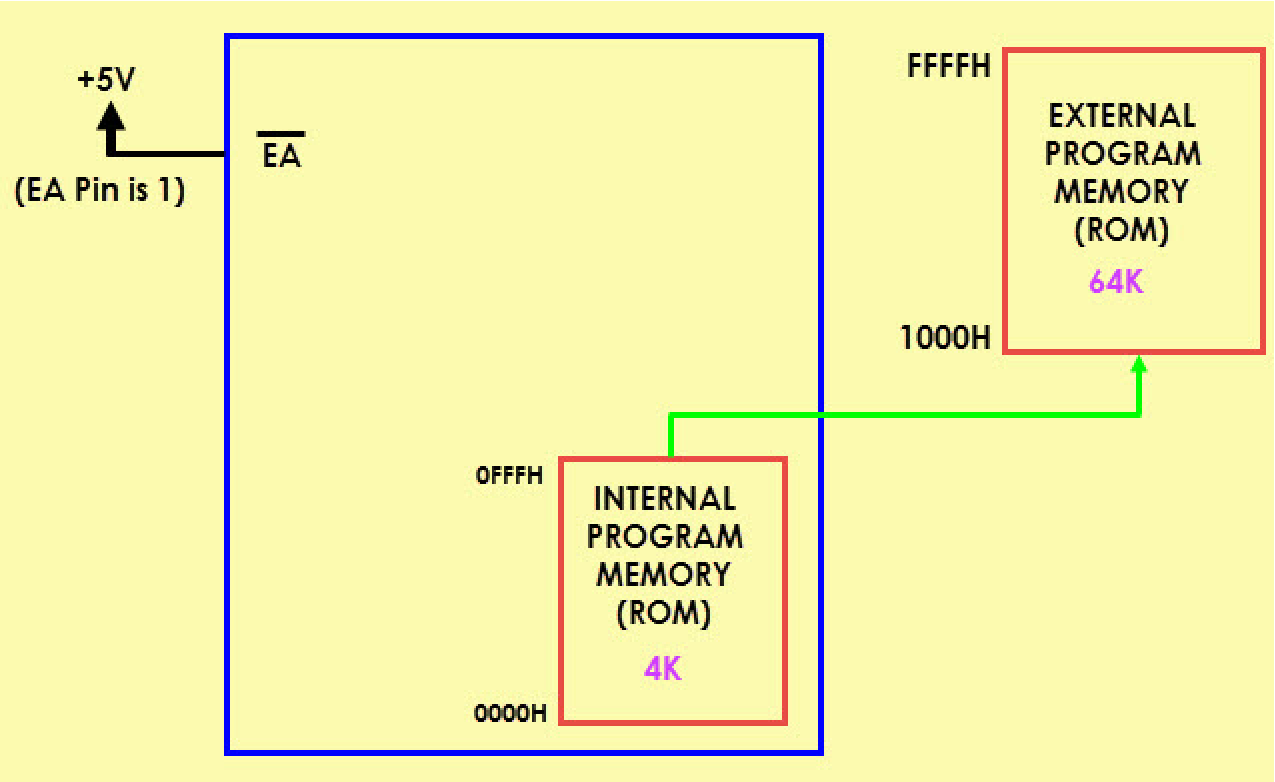
8051 Microcontroller Memory Organization

- **8051 Microcontroller Memory Organization**. The 8051 Microcontroller Memory is separated in **Program Memory (ROM) and Data Memory (RAM)**. The Program Memory of the 8051 Microcontroller is used for storing the program to be executed i.e. instructions. The Data Memory on the other hand, is used for storing temporary variable data and intermediate ...

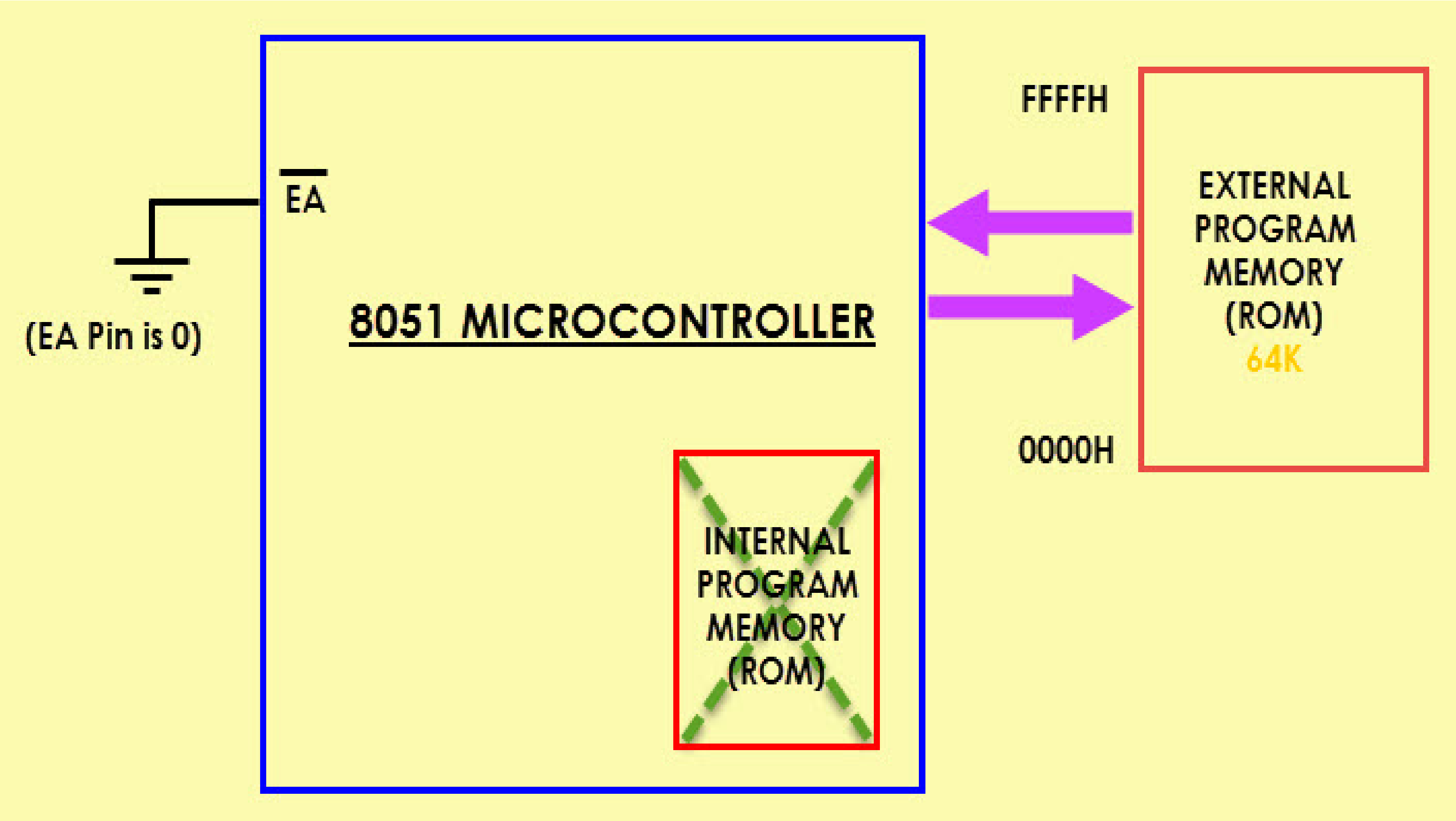
PROGRAM MEMORY (ROM) OF 8051 MICROCONTROLLER

- In 8051 Microcontroller, the code or instructions to be executed are stored in the Program Memory, which is also called as the ROM of the Microcontroller. The original 8051 Microcontroller by Intel has 4KB of internal ROM.
- In case of 4KB of Internal ROM, the address space is 0000H to 0FFFH. If the address space i.e. the program addresses exceed this value, then the CPU will automatically fetch the code from the external Program Memory.
- For this, the External Access Pin (EA Pin) must be pulled HIGH i.e. when the EA Pin is high, the CPU first fetches instructions from the Internal Program Memory in the address range of 0000H to 0FFFFH and if the memory addresses exceed the limit, then the instructions are fetched from the external ROM in the address range of 1000H to FFFFH.As shown in fig

INTERNAL ROM AND EXTERNAL ROM ORGANIZATION OF 8051



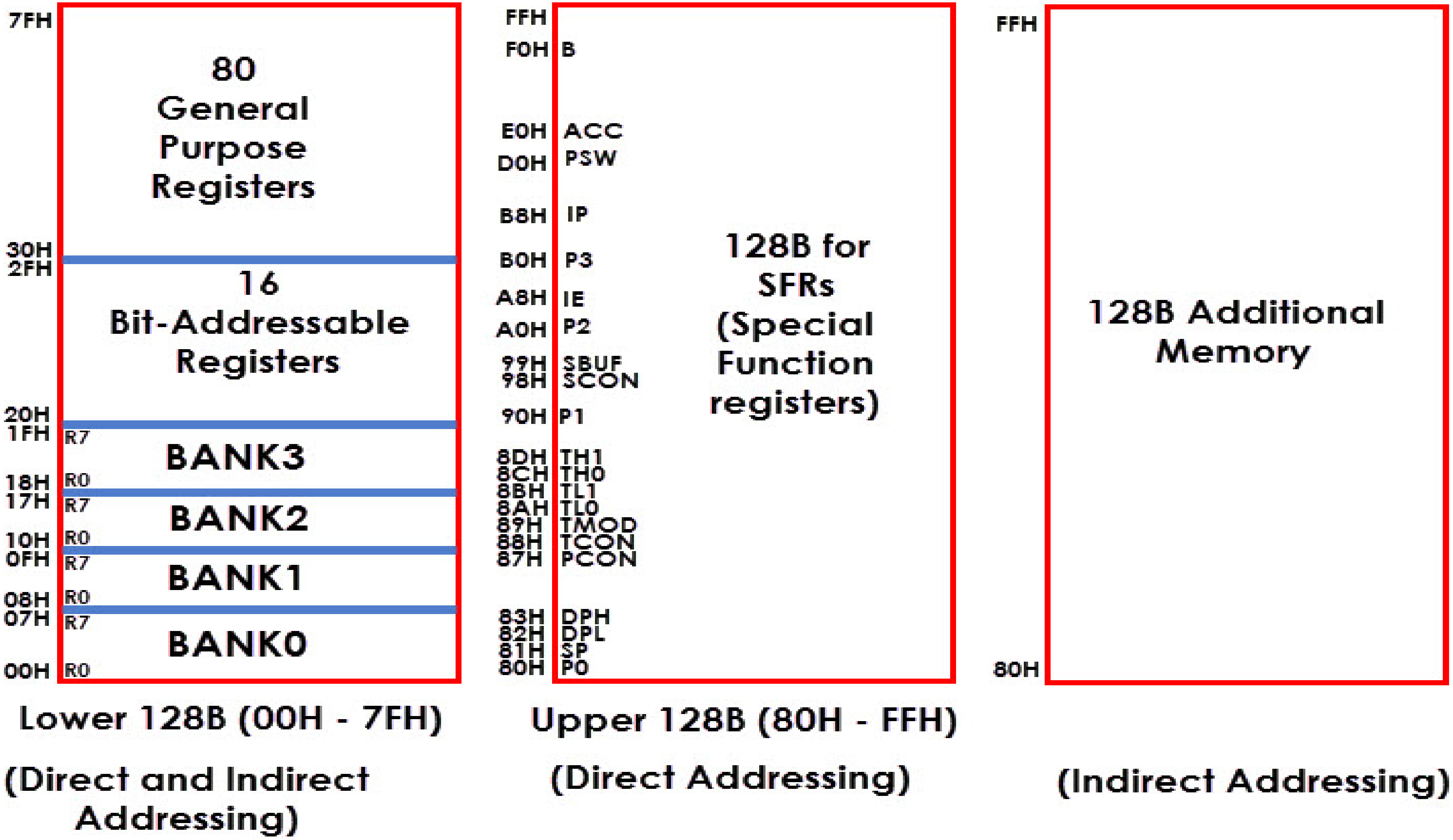
There is another way to fetch the instructions: ignore the Internal ROM and fetch all the instructions only from the External Program Memory (External ROM). For this scenario, the EA Pin must be connected to GND. In this case, the memory addresses of the external ROM will be from 0000H to FFFFH.as shown in fig



DATA MEMORY (RAM) OF 8051 MICROCONTROLLER

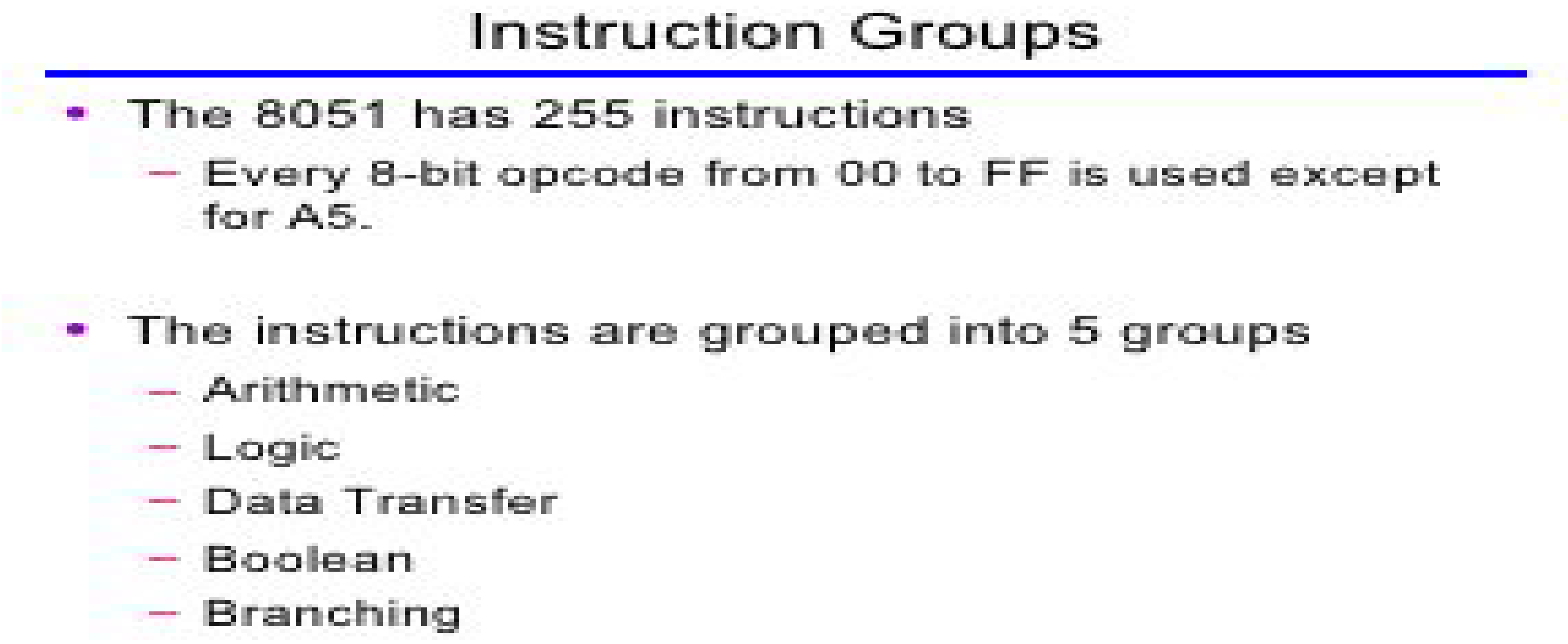
- The Data Memory or RAM of the 8051 Microcontroller stores temporary data and intermediate results that are generated and used during the normal operation of the microcontroller. Original Intel’s 8051 Microcontroller had 128B of internal RAM.
- But almost all modern variants of 8051 Microcontroller have 256B of RAM. In this 256B, the first 128B i.e. memory addresses from 00H to 7FH is divided in to Working Registers (organized as Register Banks), Bit – Addressable Area and General Purpose RAM (also known as Scratchpad area).
- In the first 128B of RAM (from 00H to 7FH), the first 32B i.e. memory from addresses 00H to 1FH consists of 32 Working Registers that are organized as four banks with 8 Registers in each Bank.
- The 4 banks are named as Bank0, Bank1, Bank2 and Bank3. Each Bank consists of 8 registers named as R0 – R7. Each Register can be addressed in two ways: either by name or by address.
- To address the register by name, first the corresponding Bank must be selected. In order to select the bank, we have to use the RS0 and RS1 bits of the Program Status Word (PSW) Register (RS0 and RS1 are 3rd and 4th bits in the PSW Register).
- The next 16B of the RAM i.e. from 20H to 2FH are Bit – Addressable memory locations. There are totally 128 bits that can be addressed individually using 00H to 7FH or the entire byte can be addressed as 20H to 2FH.

The final 80B of the internal RAM i.e. addresses from 30H to 7FH, is the general purpose RAM area which are byte addressable.
These lower 128B of RAM can be addressed directly or indirectly.



- The upper 128B of the RAM i.e. memory addresses from 80H to FFH is allocated for Special Function Registers (SFRs). SFRs control specific functions of the 8051 Microcontroller. Some of the SFRs are I/O Port Registers (P0, P1, P2 and P3), PSW (Program Status Word), A (Accumulator), IE (Interrupt Enable), PCON (Power Control), etc.
- SRFs Memory addresses are only direct addressable. Even though some of the addresses between 80H and FFH are not assigned to any SFR, they cannot be used as additional RAM area.
- In some microcontrollers, there is an additional 128B of RAM, which share the memory address with SFRs i.e. 80H to FFH. But, this additional RAM block is only accessed by indirect addressing.

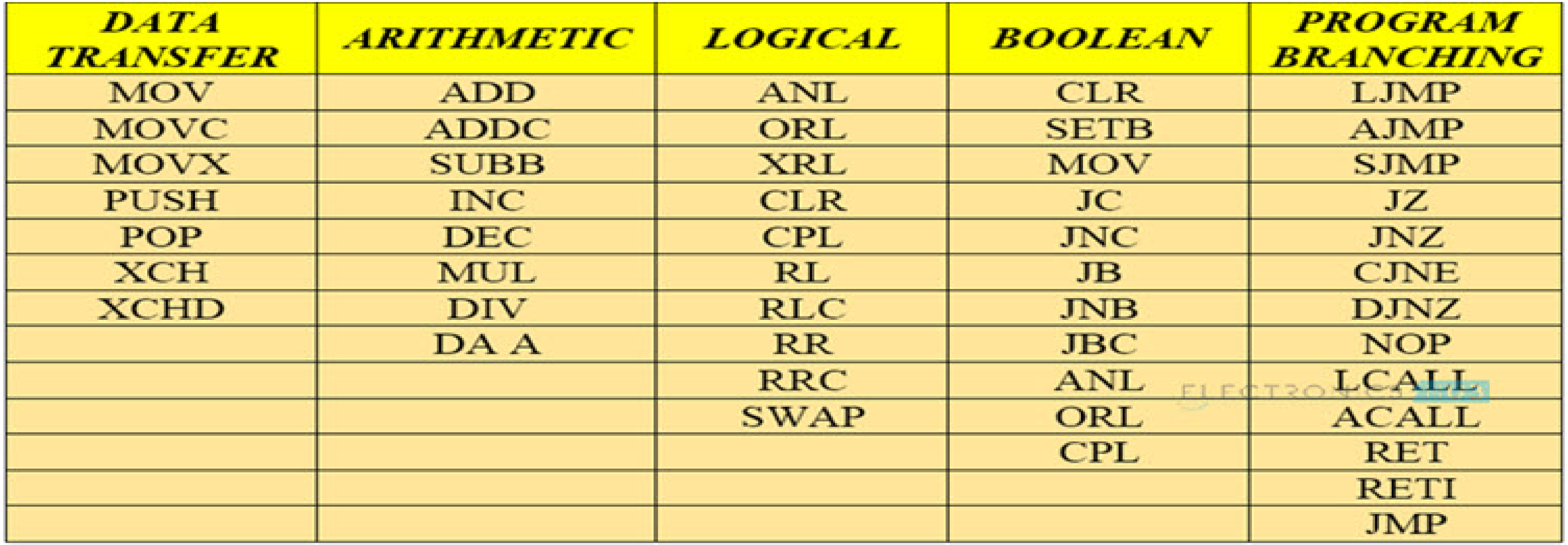
INSTRUCTION SET OF 8051 MICROCONTROLLER



Instruction Groups

- The 8051 has 255 instructions
  - Every 8-bit opcode from 00 to FF is used except for A5.

- The instructions are grouped into 5 groups
  - Arithmetic
  - Logic
  - Data Transfer
  - Boolean
  - Branching

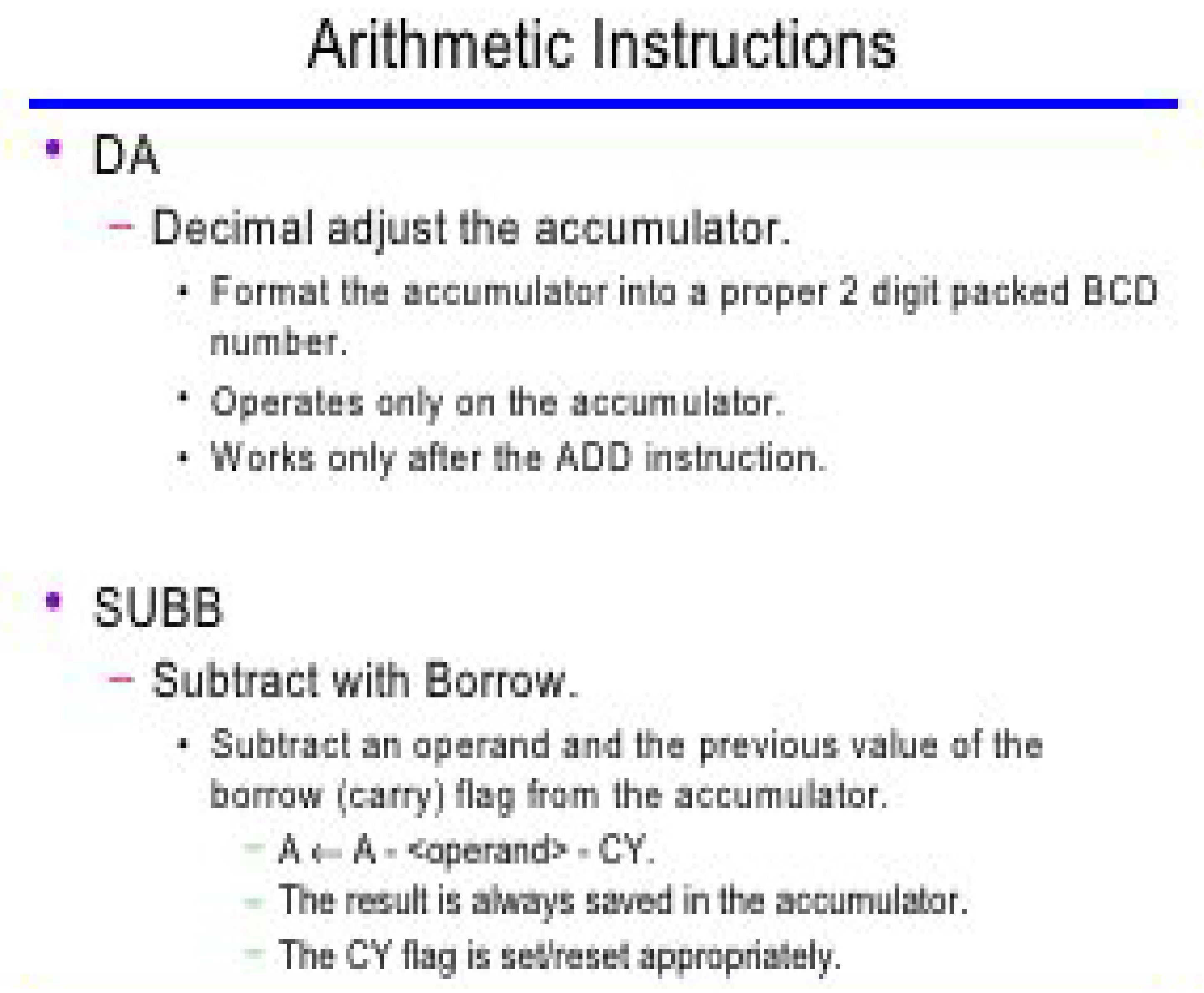


DATA TRANSFER	ARITHMETIC	LOGICAL	BOOLEAN	PROGRAM BRANCHING
MOV	ADD	ANL	CLR	LJMP
MOVC	ADDC	ORL	SETB	AJMP
MOVX	SUBB	XRL	MOV	SJMP
PUSH	INC	CLR	JC	JZ
POP	DEC	CPL	JNC	JNZ
XCH	MUL	RL	JB	CJNE
XCHD	DIV	RLC	JNB	DJNZ
	DA A	RR	JBC	NOP
		RRC	ANL	LCALL
		SWAP	ORL	ACALL
			CPL	RET
				RETI
				JMP

8051 MICROCONTROLLER INSTRUCTION SET

Arithmetic Instructions

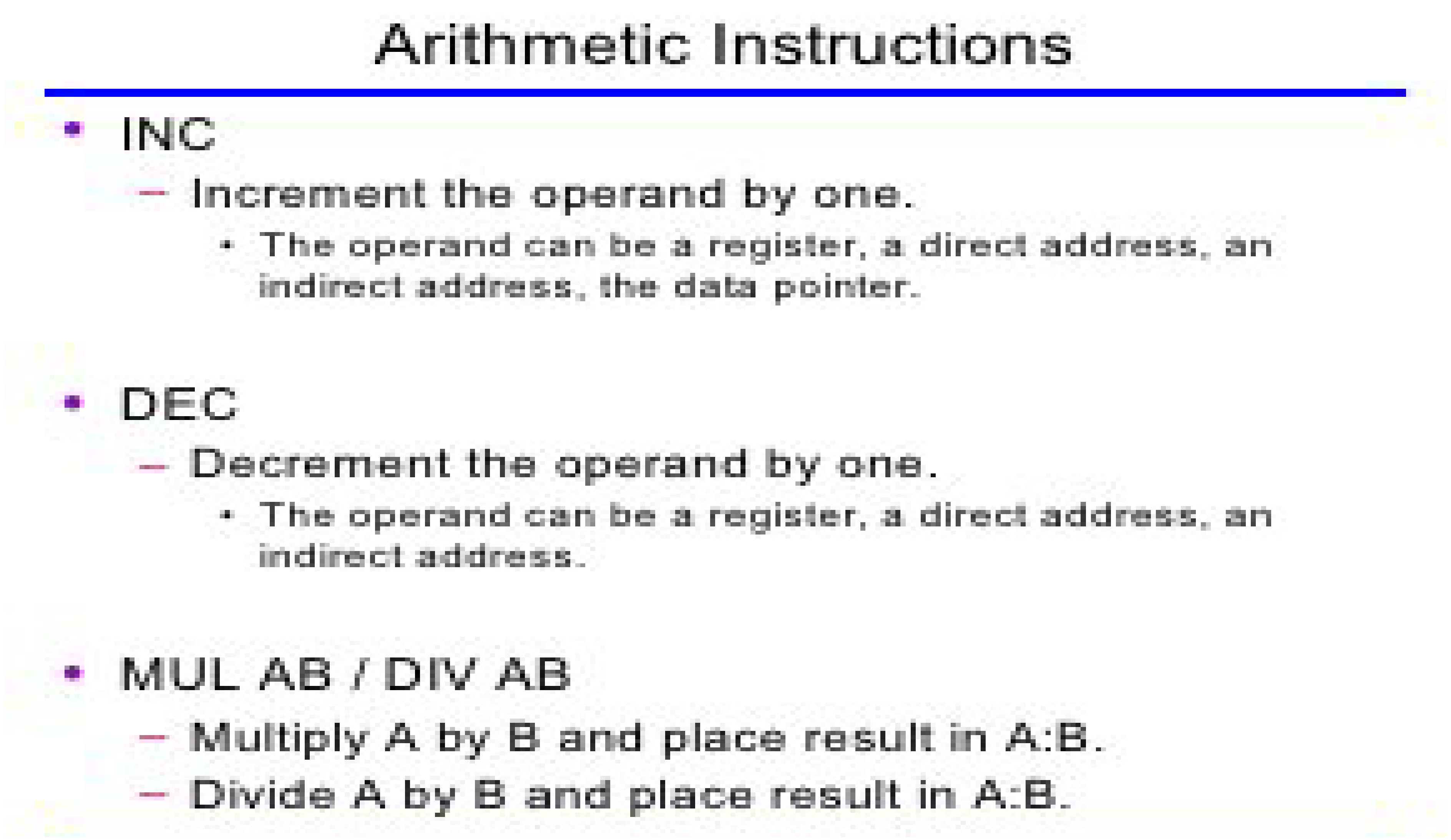
- ADD
  - 8-bit addition between the accumulator (A) and a second operand.
    - The result is always in the accumulator.
    - The CY flag is set/reset appropriately.
- ADDC
  - 8-bit addition between the accumulator, a second operand and the previous value of the CY flag.
    - Useful for 16-bit addition in two steps.
    - The CY flag is set/reset appropriately.



Arithmetic Instructions

- DA
  - Decimal adjust the accumulator.
    - Format the accumulator into a proper 2 digit packed BCD number.
    - Operates only on the accumulator.
    - Works only after the ADD instruction.

- SUBB
  - Subtract with Borrow.
    - Subtract an operand and the previous value of the borrow (carry) flag from the accumulator.
      - A ← A - <operand> - CY.
      - The result is always saved in the accumulator.
      - The CY flag is set/reset appropriately.



Arithmetic Instructions

- INC
  - Increment the operand by one.
    - The operand can be a register, a direct address, an indirect address, the data pointer.
- DEC
  - Decrement the operand by one.
    - The operand can be a register, a direct address, an indirect address.
- MUL AB / DIV AB
  - Multiply A by B and place result in A:B.
  - Divide A by B and place result in A:B.

Logical Operations

- XRL
  - Works on bytes only.
- CPL / CLR
  - Complement / Clear.
  - Work on the accumulator or a bit.
    - CLR P1.2

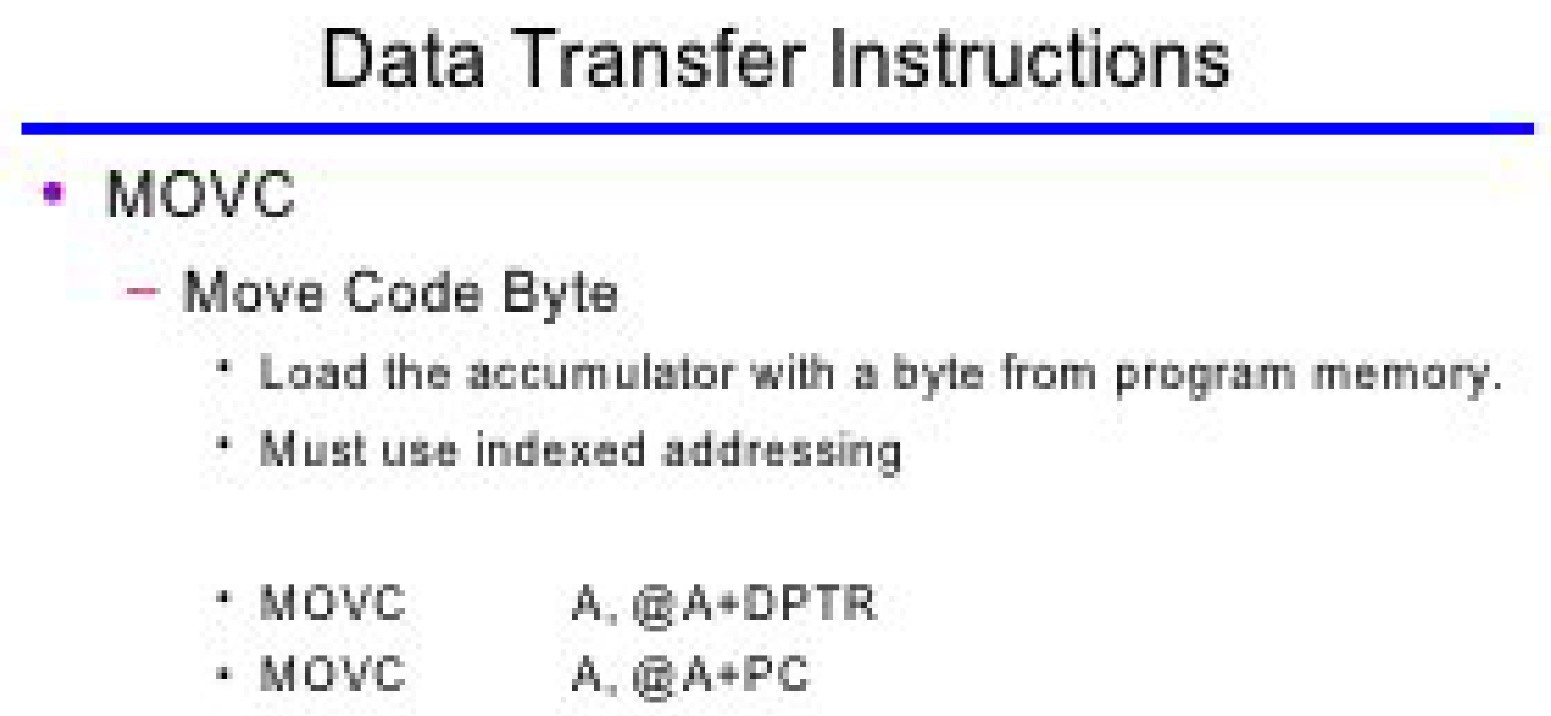
Logical Operations

- RL / RLC / RR / RRC
  - Rotate the accumulator.
    - RL and RR without the carry
    - RLC and RRC rotate through the carry.
- SWAP A
  - Swap the upper and lower nibbles of the accumulator.
- No compare instruction.
  - Built into conditional branching instructions.

Data Transfer Operations

- MOV
  - 1-bit data transfer involving the CY flag
    - MOV C, bit
    - MOV bit, C

- MOV
  - 16-bit data transfer involving the DPTR
    - MOV DPTR, #data



Data Transfer Instructions

- MOVC
  - Move Code Byte
    - Load the accumulator with a byte from program memory.
    - Must use indexed addressing

    - MOVC A, @A+DPTR
    - MOVC A, @A+PC

Data Transfer Instructions

- PUSH / POP
  - Push and Pop a data byte onto the stack.
  - The data byte is identified by a direct address from the internal RAM locations.
    - PUSH DPL
    - POP 40H

Data Transfer Instructions

- XCH
  - Exchange accumulator and a byte variable
    - XCH A, Rn
    - XCH A, direct
    - XCH A, @Ri

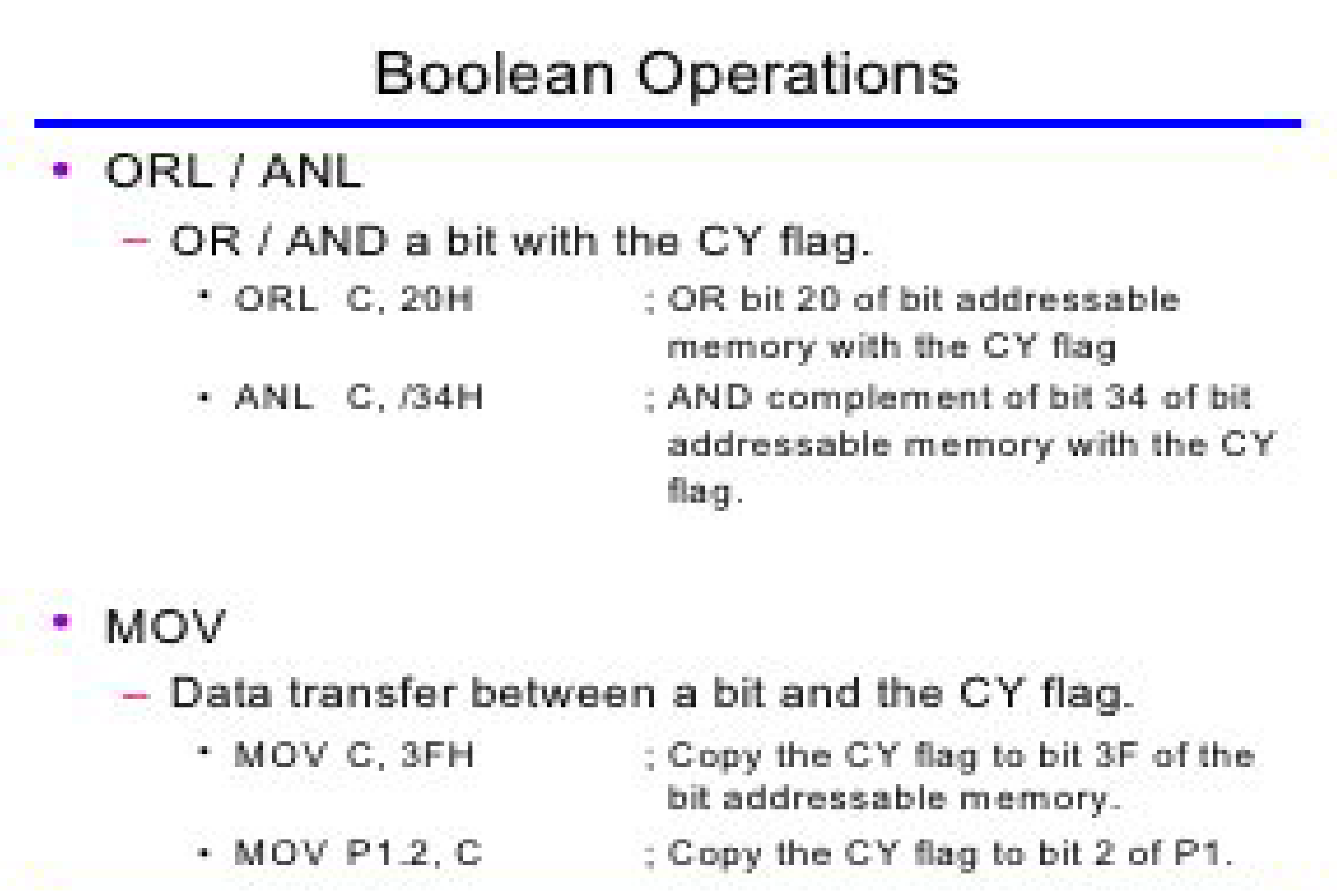
- XCHD
  - Exchange lower digit of accumulator with the lower digit of the memory location specified.
    - XCHD A, @Ri
    - The lower 4-bits of the accumulator are exchanged with the lower 4-bits of the internal memory location identified indirectly by the index register.
    - The upper 4-bits of each are not modified.

Boolean Operations

- This group of instructions is associated with the single-bit operations of the 8051.
- This group allows manipulating the individual bits of bit addressable registers and memory locations as well as the CY flag.
  - The P, OV, and AC flags cannot be directly altered.
- This group includes:
  - Set, clear, and, or complement, move.
  - Conditional jumps.

Boolean Operations

- CLR
  - Clear a bit or the CY flag.
    - CLR P1.1
    - CLR C
- SETB
  - Set a bit or the CY flag.
    - SETB A.2
    - SETB C
- CPL
  - Complement a bit or the CY flag.
    - CPL 40H ; Complement bit 40 of the bit addressable memory



Boolean Operations

- ORL / ANL
  - OR / AND a bit with the CY flag.
    - ORL C, 20H ; OR bit 20 of bit addressable memory with the CY flag
    - ANL C, /34H ; AND complement of bit 34 of bit addressable memory with the CY flag.
- MOV
  - Data transfer between a bit and the CY flag.
    - MOV C, 3FH ; Copy the CY flag to bit 3F of the bit addressable memory.
    - MOV P1.2, C ; Copy the CY flag to bit 2 of P1.

Boolean Operations

- JC / JNC
  - Jump to a relative address if CY is set / cleared.
- JB / JNB
  - Jump to a relative address if a bit is set / cleared.
    - JB ACC.2, <label>
- JBC
  - Jump to a relative address if a bit is set and clear the bit.

Branching Instructions

- The 8051 provides four different types of unconditional jump instructions:
  - Short Jump – SJMP
    - Uses an 8-bit signed offset relative to the 1st byte of the next instruction.
  - Long Jump – LJMP
    - Uses a 16-bit address.
    - 3 byte instruction capable of referencing any location in the entire 64K of program memory.

Branching Instructions

- Absolute Jump – AJMP
  - Uses an 11-bit address.
  - 2 byte instruction
    - The upper 3-bits of the address combine with the 5-bit opcode to form the 1st byte and the lower 8-bits of the address form the 2nd byte.
  - The 11-bit address is substituted for the lower 11-bits of the PC to calculate the 16-bit address of the target.
    - The location referenced must be within the 2K Byte memory page containing the AJMP instruction.

- Indirect Jump – JMP
  - JMP @A + DPTR

Branching Instructions

- The 8051 provides 2 forms for the CALL instruction:
  - Absolute Call – ACALL
    - Uses an 11-bit address similar to AJMP
    - The subroutine must be within the same 2K page.
  - Long Call – LCALL
    - Uses a 16-bit address similar to LJMP
    - The subroutine can be anywhere.

  - Both forms push the 16-bit address of the next instruction on the stack and update the stack pointer.

Branching Instructions

- The 8051 provides 2 forms for the return instruction:
  - Return from subroutine – RET
    - Pop the return address from the stack and continue execution there.
  - Return from ISV – RETI
    - Pop the return address from the stack.
    - Restore the interrupt logic to accept additional interrupts at the same priority level as the one just processed.
    - Continue execution at the address retrieved from the stack.
    - The PSW is not automatically restored.

Branching Instructions

- The 8051 supports 5 different conditional jump instructions.
  - ALL conditional jump instructions use an 8-bit signed offset.
  - Jump on Zero – JZ / JNZ
    - Jump if the A == 0 / A != 0
      - The check is done at the time of the instruction execution.
  - Jump on Carry – JC / JNC
    - Jump if the C flag is set / cleared.

Branching Instructions

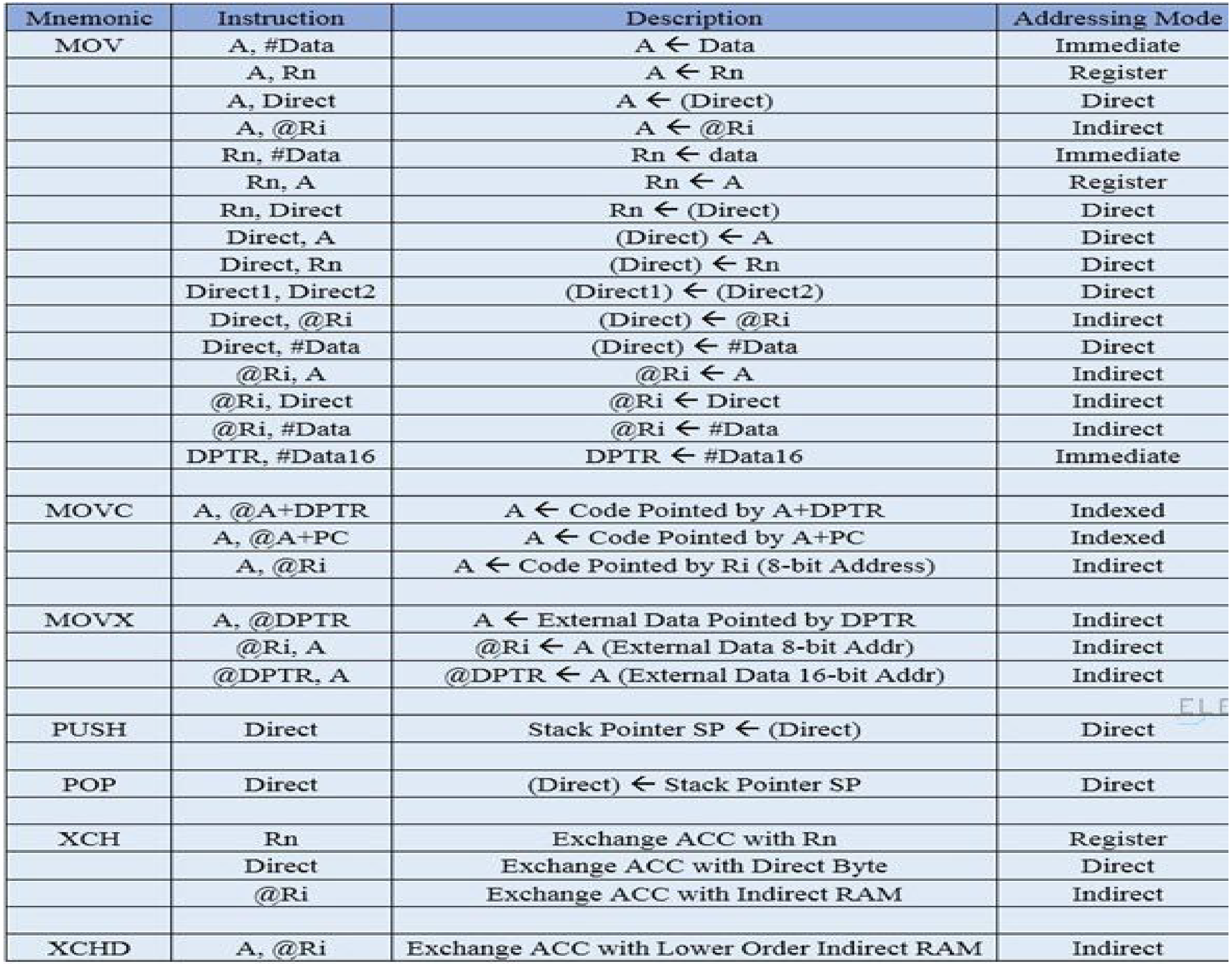
- Jump on Bit – JB / JNB
  - Jump if the specified bit is set / cleared.
  - Any addressable bit can be specified.

- Jump if the Bit is set then Clear the bit – JBC
  - Jump if the specified bit is set.
  - Then clear the bit.

Branching Instructions

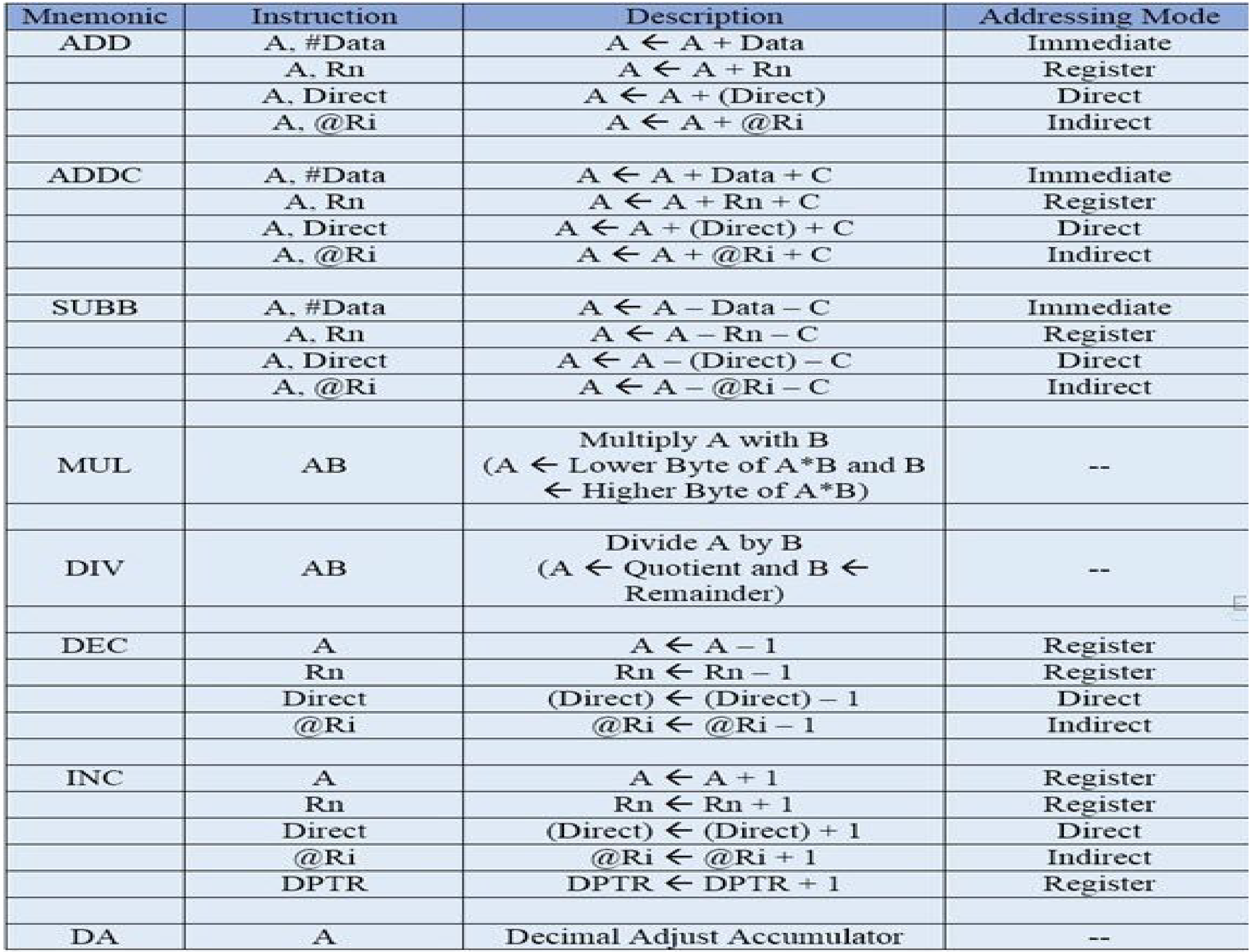
- Decrement and Jump if Not Zero – DJNZ
  - Decrement the first operand by 1 and jump to the location identified by the second operand if the resulting value is not zero.
    - DJNZ Rn, rel
    - DJNZ direct, rel
- No Operation
  - NOP

DATA TRANSFER INSTRUCTIONS



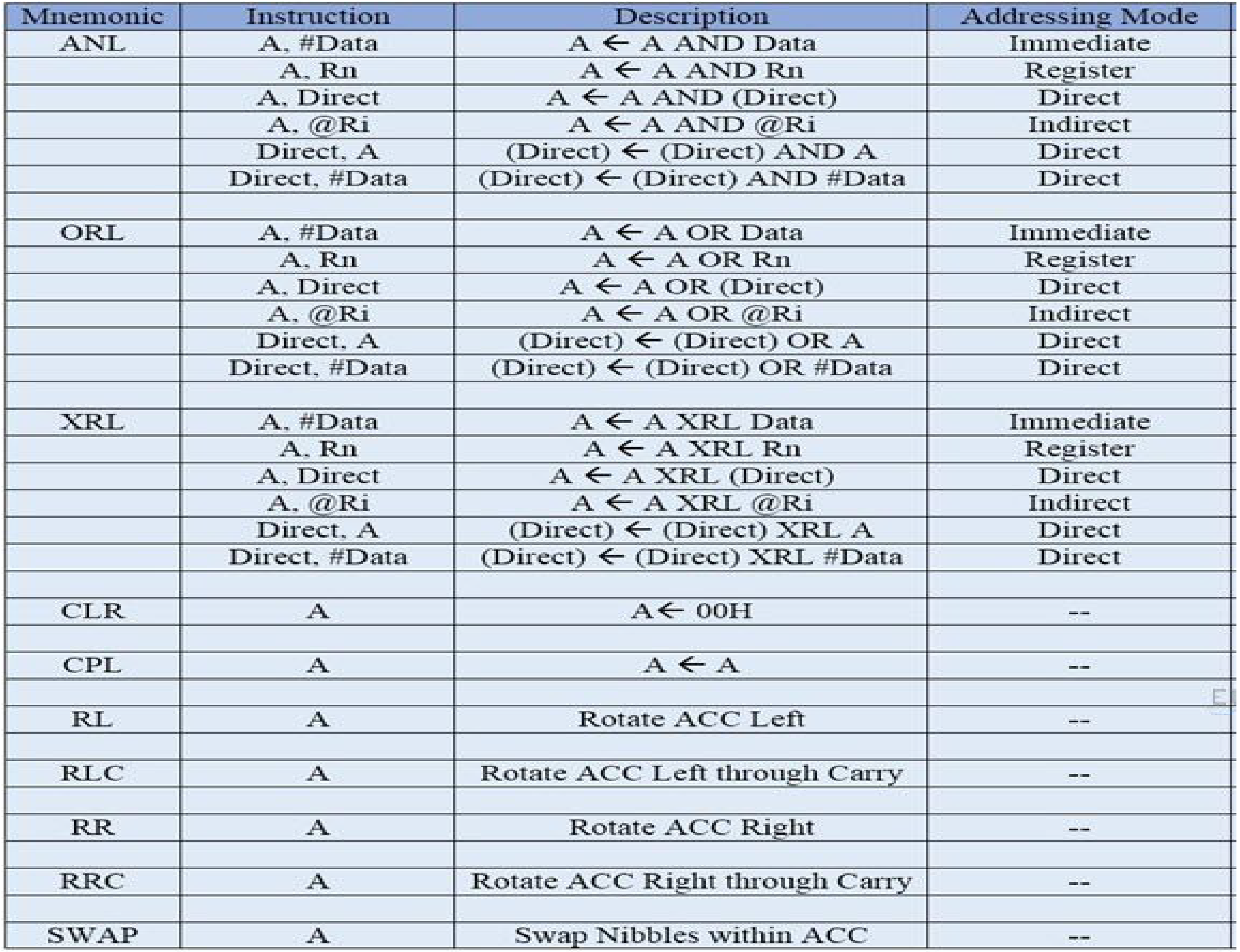
Mnemonic	Instruction	Description	Addressing Mode
MOV	A, #Data	A ← Data	Immediate
	A, Rn	A ← Rn	Register
	A, Direct	A ← (Direct)	Direct
	A, @Ri	A ← @Ri	Indirect
	Rn, #Data	Rn ← data	Immediate
	Rn, A	Rn ← A	Register
	Rn, Direct	Rn ← (Direct)	Direct
	Direct, A	(Direct) ← A	Direct
	Direct, Rn	(Direct) ← Rn	Direct
	Direct1, Direct2	(Direct1) ← (Direct2)	Direct
	Direct, @Ri	(Direct) ← @Ri	Indirect
	Direct, #Data	(Direct) ← #Data	Direct
	@Ri, A	@Ri ← A	Indirect
	@Ri, Direct	@Ri ← Direct	Indirect
	@Ri, #Data	@Ri ← #Data	Indirect
	DPTR, #Data16	DPTR ← #Data16	Immediate
MOVC	A, @A+DPTR	A ← Code Pointed by A+DPTR	Indexed
	A, @A+PC	A ← Code Pointed by A+PC	Indexed
	A, @Ri	A ← Code Pointed by Ri (8-bit Address)	Indirect
MOVX	A, @DPTR	A ← External Data Pointed by DPTR	Indirect
	@Ri, A	@Ri ← A (External Data 8-bit Addr)	Indirect
	@DPTR, A	@DPTR ← A (External Data 16-bit Addr)	Indirect
PUSH	Direct	Stack Pointer SP ← (Direct)	Direct
POP	Direct	(Direct) ← Stack Pointer SP	Direct
XCH	Rn	Exchange ACC with Rn	Register
	Direct	Exchange ACC with Direct Byte	Direct
	@Ri	Exchange ACC with Indirect RAM	Indirect
XCHD	A, @Ri	Exchange ACC with Lower Order Indirect RAM	Indirect

ARITHMATIC INSTRUCTIONS



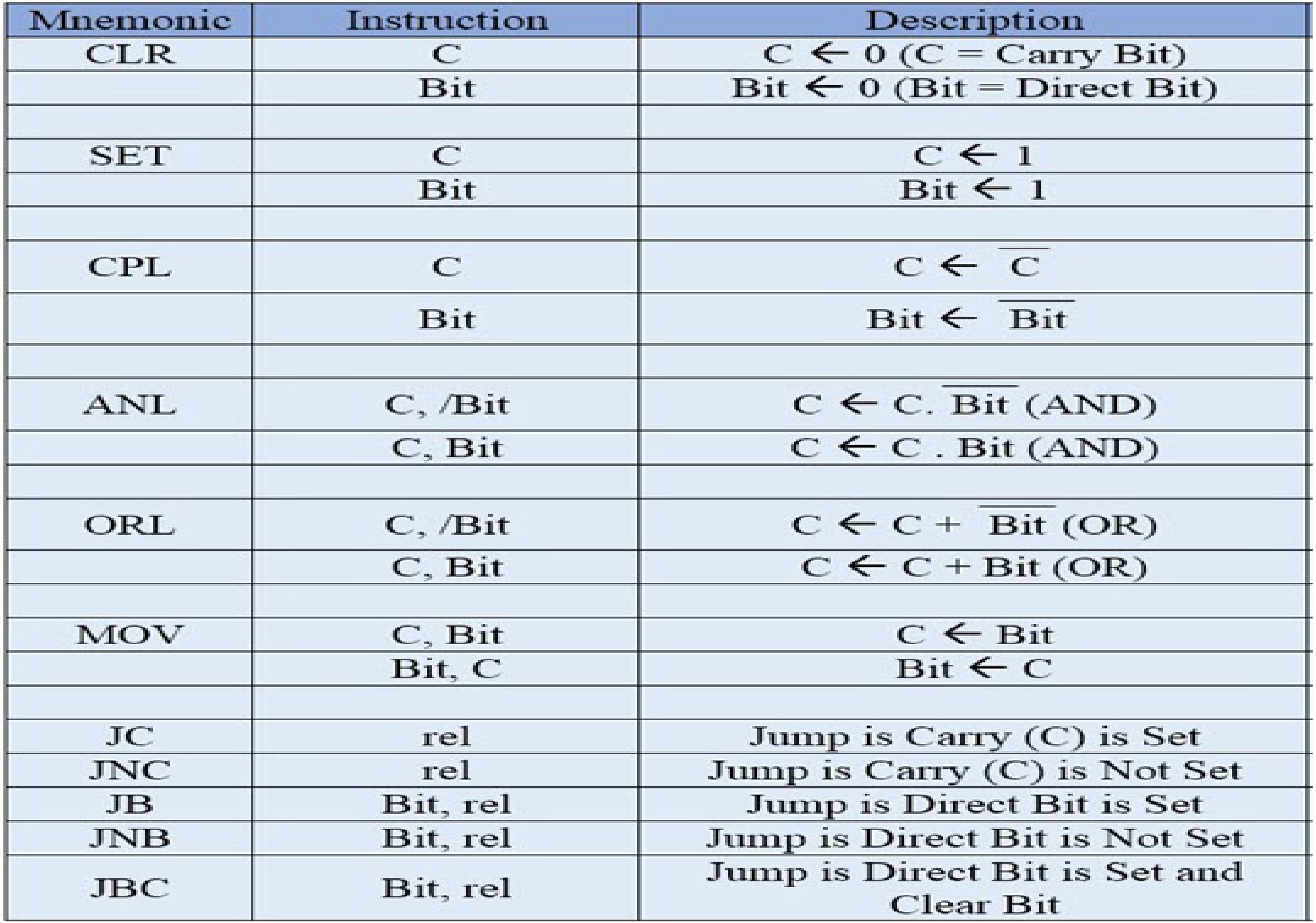
Mnemonic	Instruction	Description	Addressing Mode
ADD	A, #Data	A ← A + Data	Immediate
	A, Rn	A ← A + Rn	Register
	A, Direct	A ← A + (Direct)	Direct
	A, @Ri	A ← A + @Ri	Indirect
ADDC	A, #Data	A ← A + Data + C	Immediate
	A, Rn	A ← A + Rn + C	Register
	A, Direct	A ← A + (Direct) + C	Direct
	A, @Ri	A ← A + @Ri + C	Indirect
SUBB	A, #Data	A ← A – Data – C	Immediate
	A, Rn	A ← A – Rn – C	Register
	A, Direct	A ← A – (Direct) – C	Direct
	A, @Ri	A ← A – @Ri – C	Indirect
MUL	AB	Multiply A with B (A ← Lower Byte of A*B and B ← Higher Byte of A*B)	--
DIV	AB	Divide A by B (A ← Quotient and B ← Remainder)	--
DEC	A	A ← A – 1	Register
	Rn	Rn ← Rn – 1	Register
	Direct	(Direct) ← (Direct) – 1	Direct
	@Ri	@Ri ← @Ri – 1	Indirect
INC	A	A ← A + 1	Register
	Rn	Rn ← Rn + 1	Register
	Direct	(Direct) ← (Direct) + 1	Direct
	@Ri	@Ri ← @Ri + 1	Indirect
	DPTR	DPTR ← DPTR + 1	Register
DA	A	Decimal Adjust Accumulator	--

LOGICAL INSTRUCTIONS



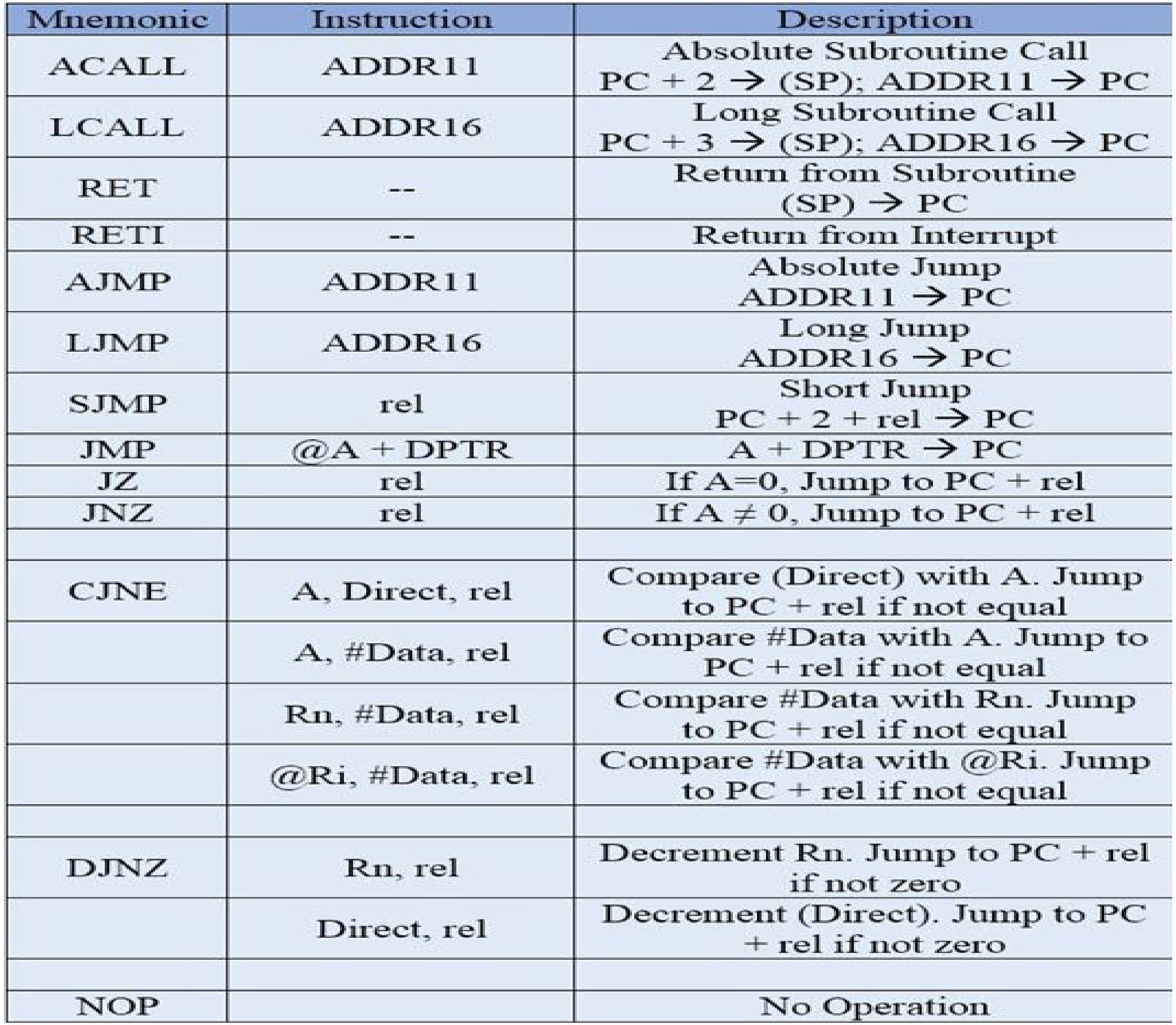
Mnemonic	Instruction	Description	Addressing Mode
ANL	A, #Data	A ← A AND Data	Immediate
	A, Rn	A ← A AND Rn	Register
	A, Direct	A ← A AND (Direct)	Direct
	A, @Ri	A ← A AND @Ri	Indirect
	Direct, A	(Direct) ← (Direct) AND A	Direct
	Direct, #Data	(Direct) ← (Direct) AND #Data	Direct
ORL	A, #Data	A ← A OR Data	Immediate
	A, Rn	A ← A OR Rn	Register
	A, Direct	A ← A OR (Direct)	Direct
	A, @Ri	A ← A OR @Ri	Indirect
	Direct, A	(Direct) ← (Direct) OR A	Direct
	Direct, #Data	(Direct) ← (Direct) OR #Data	Direct
XRL	A, #Data	A ← A XRL Data	Immediate
	A, Rn	A ← A XRL Rn	Register
	A, Direct	A ← A XRL (Direct)	Direct
	A, @Ri	A ← A XRL @Ri	Indirect
	Direct, A	(Direct) ← (Direct) XRL A	Direct
	Direct, #Data	(Direct) ← (Direct) XRL #Data	Direct
CLR	A	A← 00H	--
CPL	A	A ← A	--
RL	A	Rotate ACC Left	--
RLC	A	Rotate ACC Left through Carry	--
RR	A	Rotate ACC Right	--
RRC	A	Rotate ACC Right through Carry	--
SWAP	A	Swap Nibbles within ACC	--

BOOLEAN INSTRUCTIONS



Mnemonic	Instruction	Description
CLR	C	C ← 0 (C = Carry Bit)
	Bit	Bit ← 0 (Bit = Direct Bit)
SET	C	C ← 1
	Bit	Bit ← 1
CPL	C	C ← $\overline{\text{C}}$
	Bit	Bit ← $\overline{\text{Bit}}$
ANL	C, /Bit	C ← C. $\overline{\text{Bit}}$ (AND)
	C, Bit	C ← C . Bit (AND)
ORL	C, /Bit	C ← C + $\overline{\text{Bit}}$ (OR)
	C, Bit	C ← C + Bit (OR)
MOV	C, Bit	C ← Bit
	Bit, C	Bit ← C
JC	rel	Jump is Carry (C) is Set
JNC	rel	Jump is Carry (C) is Not Set
JB	Bit, rel	Jump is Direct Bit is Set
JNB	Bit, rel	Jump is Direct Bit is Not Set
JBC	Bit, rel	Jump is Direct Bit is Set and Clear Bit

PROGRAMME BRANCHING INSTRUCTIONS



Mnemonic	Instruction	Description
ACALL	ADDR11	Absolute Subroutine Call PC + 2 → (SP); ADDR11 → PC
LCALL	ADDR16	Long Subroutine Call PC + 3 → (SP); ADDR16 → PC
RET	--	Return from Subroutine (SP) → PC
RETI	--	Return from Interrupt
AJMP	ADDR11	Absolute Jump ADDR11 → PC
LJMP	ADDR16	Long Jump ADDR16 → PC
SJMP	rel	Short Jump PC + 2 + rel → PC
JMP	@A + DPTR	A + DPTR → PC
JZ	rel	If A=0, Jump to PC + rel
JNZ	rel	If A ≠ 0, Jump to PC + rel
CJNE	A, Direct, rel	Compare (Direct) with A. Jump to PC + rel if not equal
	A, #Data, rel	Compare #Data with A. Jump to PC + rel if not equal
	Rn, #Data, rel	Compare #Data with Rn. Jump to PC + rel if not equal
	@Ri, #Data, rel	Compare #Data with @Ri. Jump to PC + rel if not equal
DJNZ	Rn, rel	Decrement Rn. Jump to PC + rel if not zero
	Direct, rel	Decrement (Direct). Jump to PC + rel if not zero
NOP		No Operation

ADDRESSING MODES OF 8051

- **Addressing mode:** The way instruction is provided
- 8051 provides total 5 addressing modes
- They are 1) Immediate
- 2)Direct
- 3)Register
- 4)Register Indirect
- 5)Indexed
- 1)) **Immediate addressing mode**: In this addressing mode the source is constant
- Immediate data must be proceeded by sign" # "
- This addressing mode can be used to load information into any one of regiter.
- **syntax** mov A, #25H; load 25h into A
- mov Rn, #30; Load decimal value into Rn register(n=0 to 7)
- mov DPTR, #4521H; load DPTR=4521H

- 2)Direct Addressing mode: This mode used to access RAM locations of 8051(30-7F)

- This mode the direct address of memory location is provided in instruction to fetch the operand. Only internal RAM and SFR's address can be used in this type of instruction.
- **syntax** : MOV A, 30H ; Content of RAM address 30H is copied into Accumulator.
- : MOV 30H , A; Content of Accumulator. is copied into RAM address 30H

- **3)** Register Addressing modes: Register addressing mode involves use of registers to hold the data to be manipulated
- **Syntax:**   mov A,Rn;   copy the content of Rn into A
- Mov Rn, A;   copy the content of A into Rn
- ADD A,Rn,;   add the content of Rn to A
-

- 4) Register Indirect Addressing modes:Here the address of memory location is indirectly provided by a register. The '@' sign indicates that the register holds the address of memory location i.e. fetch the content of memory location whose address is provided in register.

syntax: MOV A,@R0 => Copy the content of memory location whose address is given in R0 register.

• 5). Indexed Addressing mode:

- This addressing mode is basically used for accessing data from look up table. Here the address of memory is indexed i.e. added to form the actual address of memory.

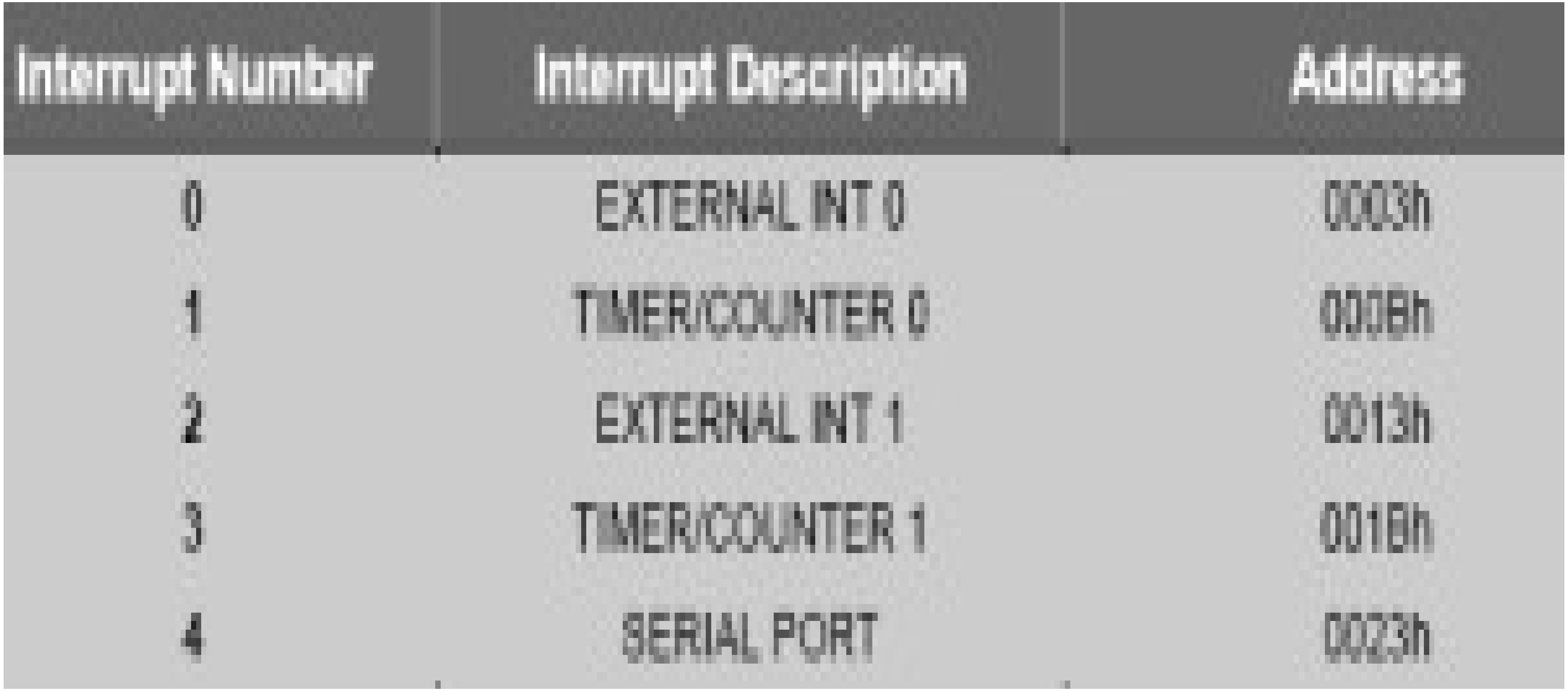
syntax: MOVC A,@A+DPTR => here 'C' means Code. Here the content of A register is added with content of DPTR and the resultant is the address of memory location from where the data is copied to A register.

Types of Interrupts in 8051 Microcontroller

- The 8051 microcontroller can recognize five different events that cause the main program to interrupt from the normal execution. These five sources of interrupts in 8051are:

1. Timer 0 overflow interrupt- TF0
2. Timer 1 overflow interrupt- TF1
3. External hardware interrupt- INT0
4. External hardware interrupt- INT1
5. Serial communication interrupt- RI/TI

The Timer and Serial interrupts are internally generated by the microcontroller, whereas the external interrupts are generated by additional interfacing devices or switches that are externally connected to the microcontroller. These external interrupts can be edge triggered or level triggered. When an interrupt occurs, the microcontroller executes the interrupt service routine so that memory location corresponds to the interrupt that enables it. The Interrupt corresponding to the memory location is given in the interrupt vector table below.

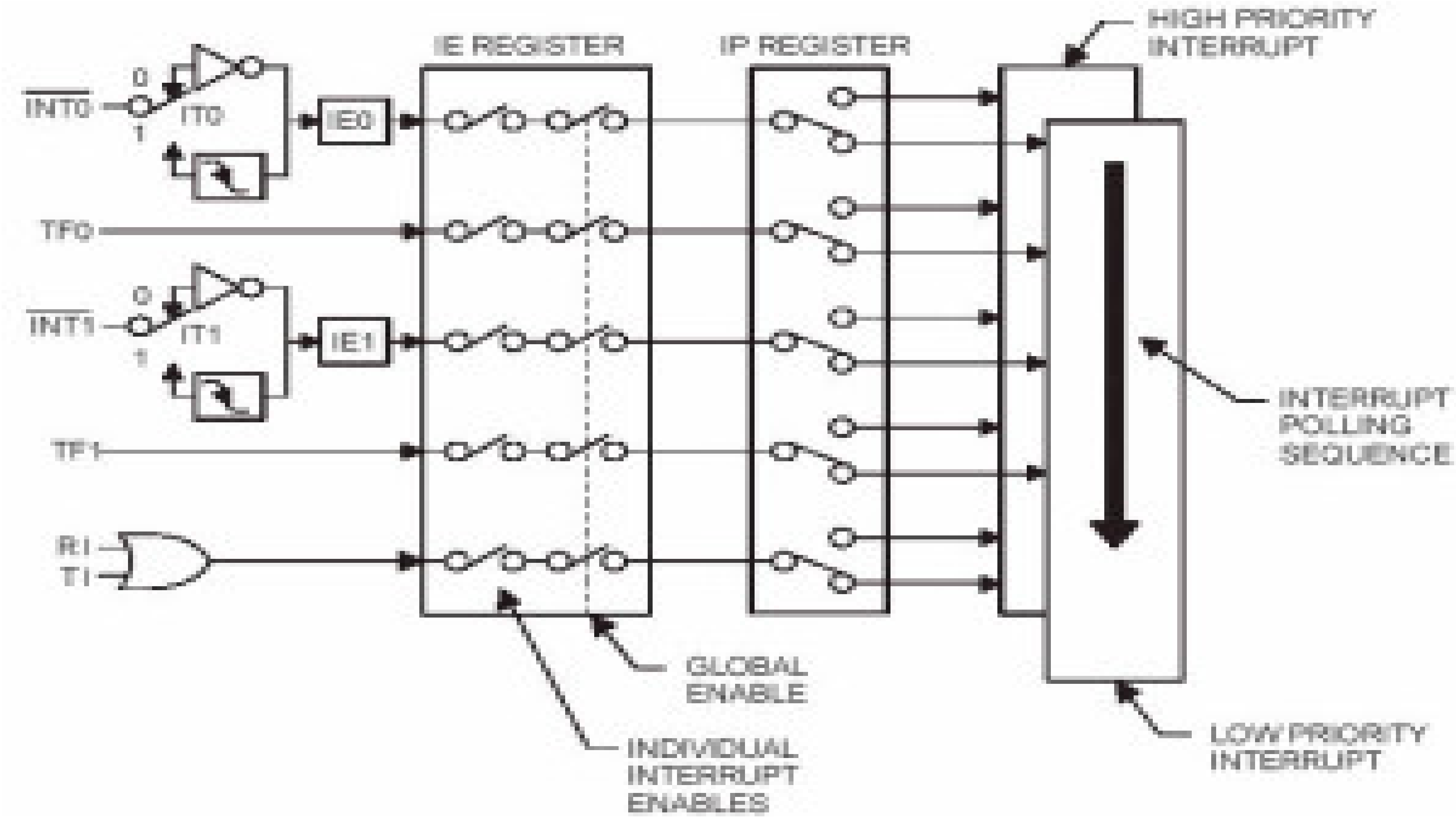


Interrupt Number	Interrupt Description	Address
0	EXTERNAL INT 0	0003h
1	TIMER/COUNTER 0	000Bh
2	EXTERNAL INT 1	0013h
3	TIMER/COUNTER 1	001Bh
4	SERIAL PORT	0023h

Interrupt Structure of 8051 Micro controller

Upon ‘RESET’ all the interrupts get disabled, and therefore, all these interrupts must be enabled by a software. In all these five interrupts, if anyone or all are activated, this sets the corresponding interrupt flags as shown in the figure. All these interrupts can be set or cleared by bit in some special function register that is Interrupt Enabled (IE), and this in turn depends on the priority, which is executed by IP interrupt priority register.

interrupt structure of 8051 microcontroller



Interrupt Enable (IE) Register

- **Interrupt Enable (IE) Register:** This register is responsible for enabling and disabling the interrupt. It is a bit addressable register in which EA must be set to one for enabling interrupts. The corresponding bit in this register enables particular interrupt like timer, external and serial inputs. In the below IE register, bit corresponding to 1 activates the interrupt and 0 disables the interrupt.

Interrupt Enable (IE) Register

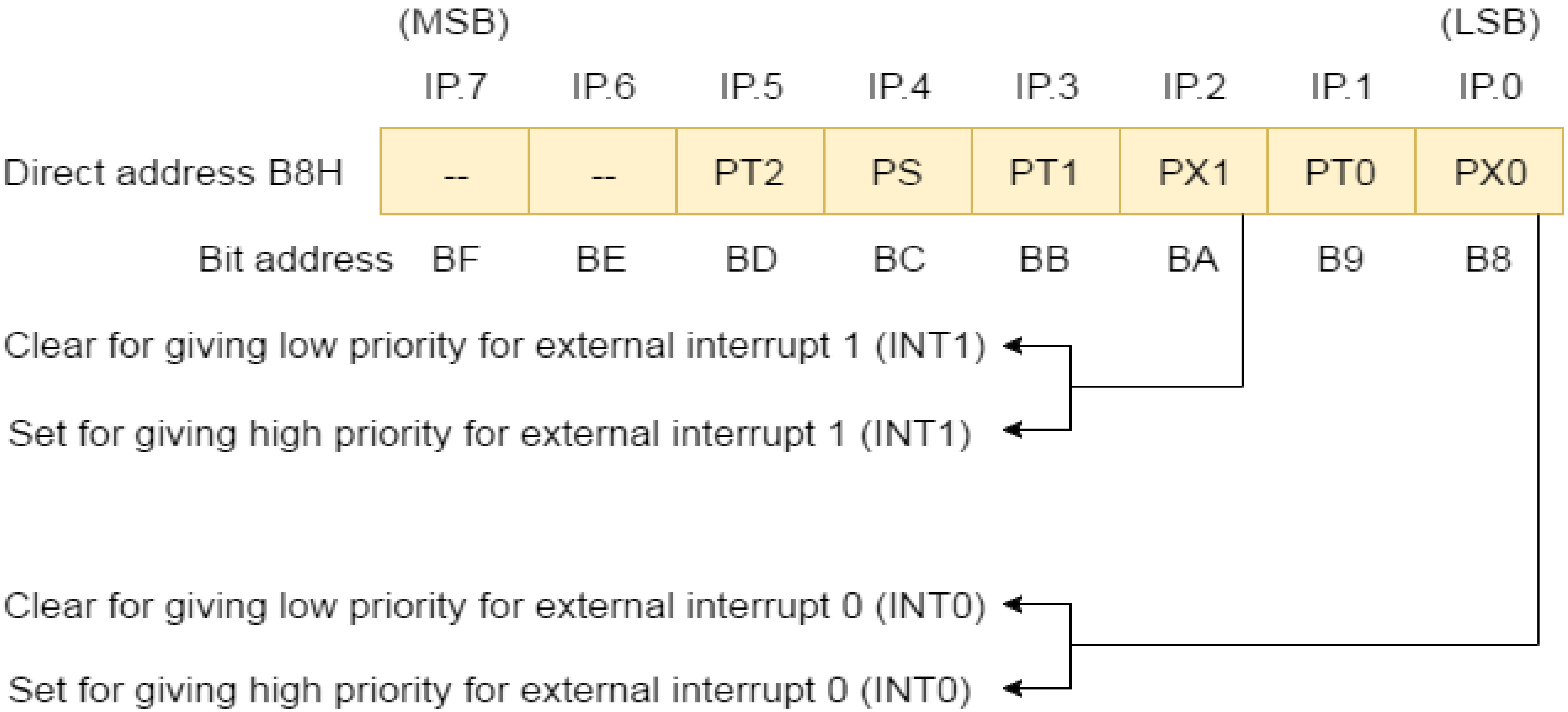
EA	--	--	ES	ET1	EX1	ET0	EX0

EA	IE.7	Disables all interrupts, If EA=0, no interrupt will be acknowledged. If EA=1, interrupt source is individually enable or disabled by setting or clearing its enable bit.
--	IE.6	Not implemented, reserved for future use*.
--	IE.5	Not implemented, reserved for future use*.
ES	IE.4	Enable or disable the Serial port interrupt.
ET1	IE.3	Enable or disable the Timer 1 overflow interrupt.
EX1	IE.2	Enable or disable External interrupt 1.
ET0	IE.1	Enable or disable the Timer 0 overflow interrupt.
EX0	IE.0	Enable or disable External interrupt 0.

Interrupt Priority Register

- **Interrupt Priority Register (IP):** It is also possible to change the priority levels of the interrupts by setting or clearing the corresponding bit in the Interrupt priority (IP) register as shown in the figure. This allows the low priority interrupt to interrupt the high-priority interrupt, but prohibits the interruption by another low-priority interrupt. Similarly, the high-priority interrupt cannot be interrupted. If these interrupt priorities are not programmed, the microcontroller executes in predefined manner and its order is INT0, TF0, INT1, TF1, and SI.

Interrupt Priority Register



	(MSB)							(LSB)
	IP.7	IP.6	IP.5	IP.4	IP.3	IP.2	IP.1	IP.0
Direct address B8H	--	--	PT2	PS	PT1	PX1	PT0	PX0
Bit address	BF	BE	BD	BC	BB	BA	B9	B8

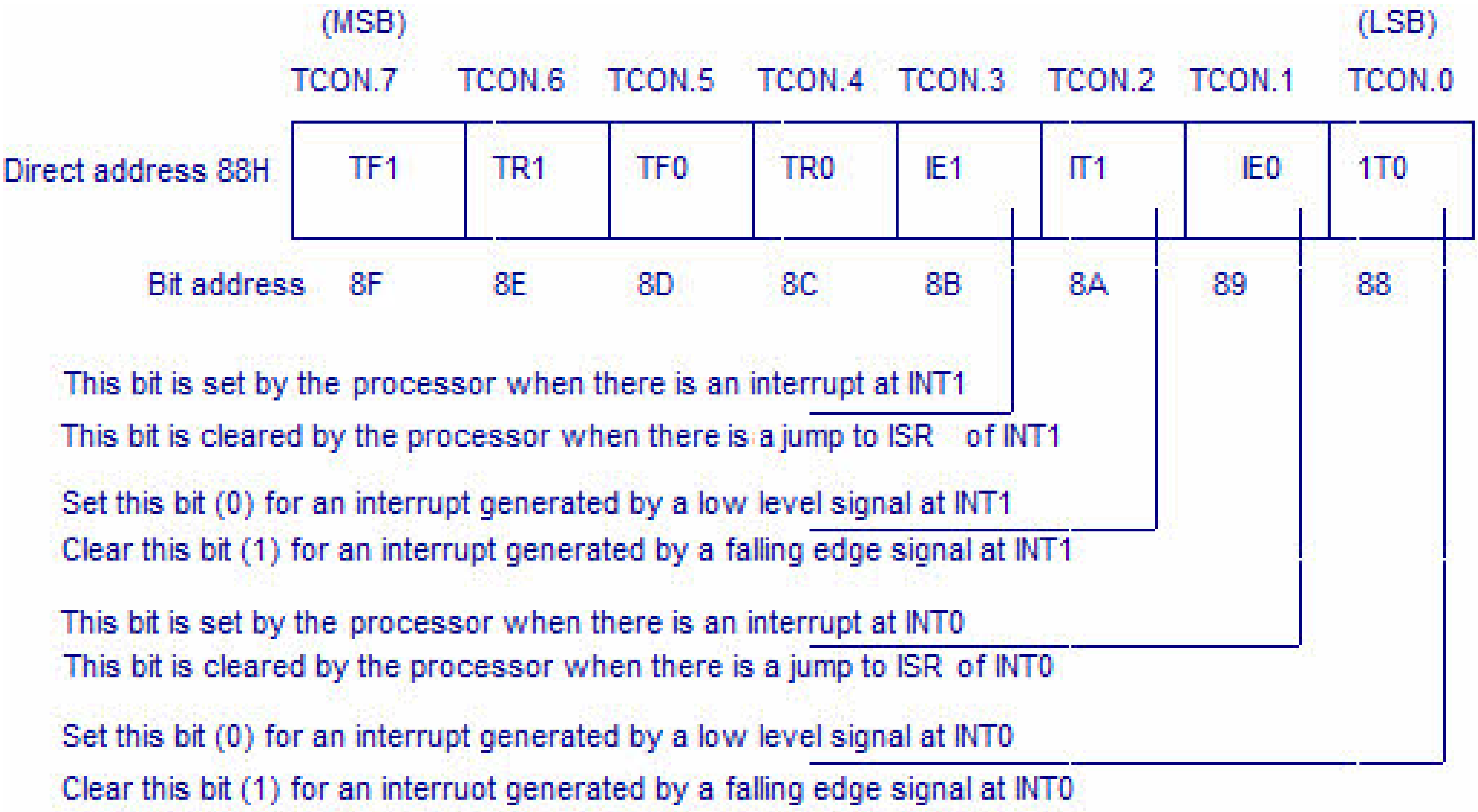
Clear for giving low priority for external interrupt 1 (INT1)

Set for giving high priority for external interrupt 1 (INT1)

Clear for giving low priority for external interrupt 0 (INT0)

Set for giving high priority for external interrupt 0 (INT0)

- **TCON Register:** In addition to the above two registers, the TCON register specifies the type of external interrupt to the 8051 microcontroller, as shown in the figure. The two external interrupts, whether edge or level triggered, specify by this register by a set, or cleared by appropriate bits in it. And, it is also a bit addressable register.



	(MSB)							(LSB)
	TCON.7	TCON.6	TCON.5	TCON.4	TCON.3	TCON.2	TCON.1	TCON.0
Direct address 88H	TF1	TR1	TF0	TR0	IE1	IT1	IE0	1T0
Bit address	8F	8E	8D	8C	8B	8A	89	88

This bit is set by the processor when there is an interrupt at INT1
This bit is cleared by the processor when there is a jump to ISR of INT1

Set this bit (0) for an interrupt generated by a low level signal at INT1
Clear this bit (1) for an interrupt generated by a falling edge signal at INT1

This bit is set by the processor when there is an interrupt at INT0
This bit is cleared by the processor when there is a jump to ISR of INT0

Set this bit (0) for an interrupt generated by a low level signal at INT0
Clear this bit (1) for an interruot generated by a falling edge signal at INT0

Interrupt Programming in 8051

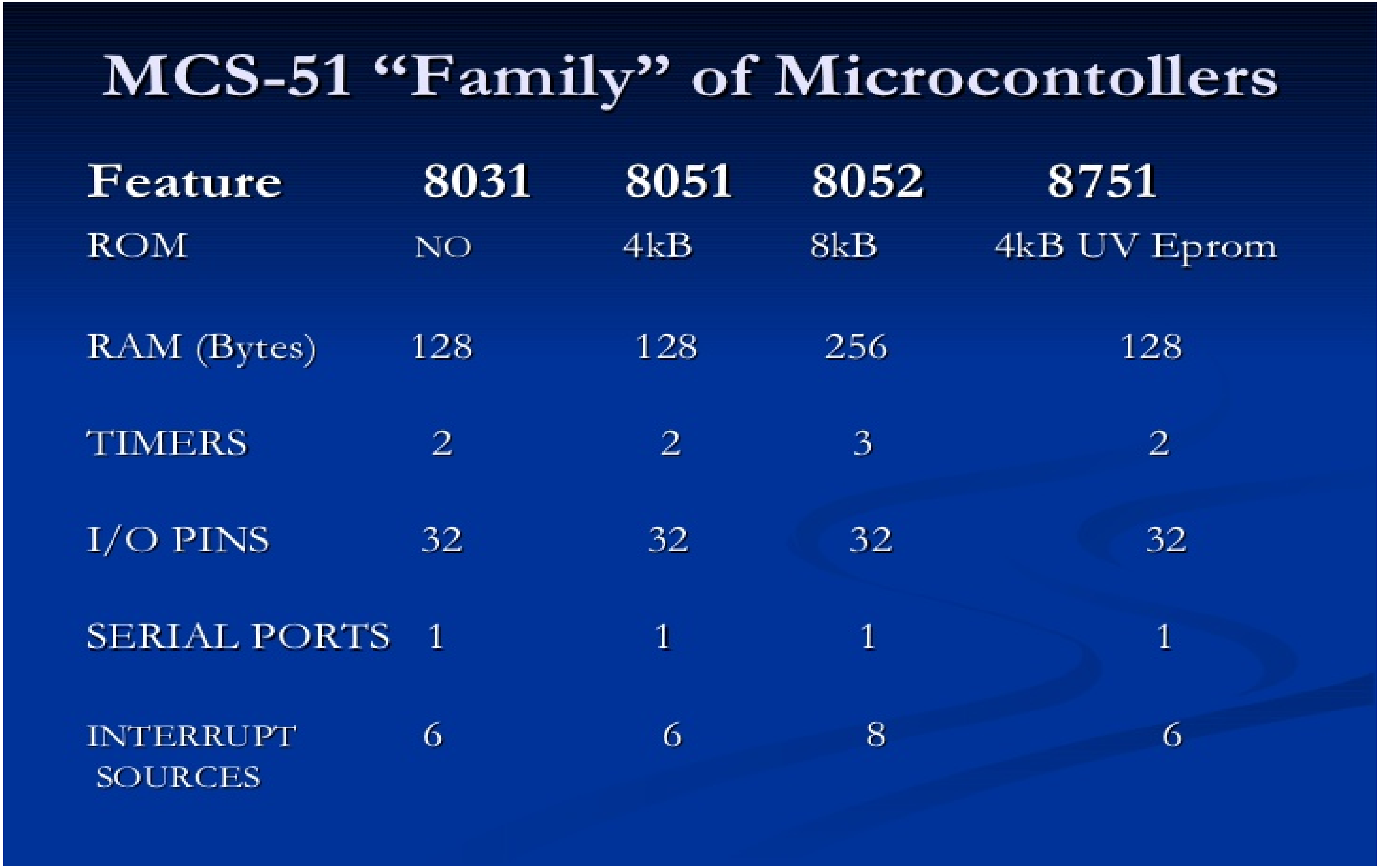
- **1.Timer Interrupt Programming**
- Timer 0 and Timer 1 interrupts are generated by the timer register bits TF0 and TF1. These interrupts programming by C code involves:
- Selecting the timer by configuring TMOD register and its mode of operation.
- Choosing and loading the initial values of TLx and THx for appropriate modes.
- Enabling the IE registers and corresponding timer bit in it.
- Setting the timer run bit to start the timer.
- Writing the subroutine for the timer for time required and clear timer value TRx at the end of subroutine.

- **2.External Hardware Interrupt Programming**
- 8051 microcontrollers consists of two external hardware interrupts: INT0 and INT1 as discussed earlier. These are enabled at pin 3.2 and pin 3.3. These can be edge triggered or level triggered. In level triggering, the low at pin 3.2 enables the interrupt, while at pin 3.2 – the high to low transition enables the edge triggered interrupt. This edge triggering or level triggering is decided by the TCON register that has been discussed above. The programming procedure in 8051 is as follows:
- Enable the corresponding bit of external interrupt in IE register.
- If it is level triggering, just write the subroutine appropriate to this interrupt, or else enable the TCON register bit corresponding to the edge triggered interrupt – whether it is INT0 or INT1.

- **3.Serial Communication Interrupt Programming**
- Serial communication interrupts come into picture when there is a need to send or receive data. Since one interrupt bit is set for both TI (Transfer Interrupt) and RI (Receiver Interrupt) flags, Interrupt Service routine must examine these flags to know the actual interrupt.
- The logical OR operation of these two flags (RI ands TI) causes this interrupt, and it is cleared by the software alone. Here, a special register SCON is used for controlling communication operation by enabling the corresponding bits in it.
- Configure the IE register for enabling serial interrupt
- Configure the SCON register for receiving or transferring operation
- Write subroutine for this interrupt with appropriate function and clear TI or RI flags with in this routine.

MODULE-V
8051 REAL TIME CONTROL

- A **microcontroller** is a small and low-cost microcomputer, which is designed to perform the specific tasks of embedded systems like displaying microwave’s information, receiving remote signals, etc.
- The general microcontroller consists of the processor, the memory (RAM, ROM, EPROM), Serial ports, peripherals (timers, counters), etc.
- 8051 microcontroller is designed by Intel in 1981. It is an 8-bit microcontroller. It is built with 40 pins DIP (dual inline package).
- It is an Electronic IC.



MCS-51 “Family” of Microcontollers

Feature	8031	8051	8052	8751
ROM	NO	4kB	8kB	4kB UV Eprom
RAM (Bytes)	128	128	256	128
TIMERS	2	2	3	2
I/O PINS	32	32	32	32
SERIAL PORTS	1	1	1	1
INTERRUPT SOURCES	6	6	8	6

- 8051 microcontroller is designed by Intel in 1981. It is an 8-bit microcontroller. It is built with 40 pins DIP (dual inline package), 4kb of ROM storage and 128 bytes of RAM storage, 2 16-bit timers.
- It consists of are four parallel 8-bit ports, which are programmable as well as addressable as per the requirement.
- An on-chip crystal oscillator is integrated in the microcontroller having crystal frequency of 12 MHz

- 32 I/O Pins (Input / Output Pins) – Arranged as 4 Ports: P0, P1, P2 and P3.
- 8- bit Stack Pointer (SP) and Processor Status Word (PSW).
- 16 – bit Program Counter (PC) and Data Pointer (DPTR).
- Two 16 – bit Timers / Counters – T0 and T1.
- Control Registers – SCON, PCON, TCON, TMOD, IP and IE.
- Serial Data Transmitter and Receiver for Full – Duplex Operation – SBUF.
- Interrupts: Two External and Three Internal.
- Oscillator and Clock Circuit.

- **Interrupts –**
- The most powerful attribute of the 8051 Microcontroller is the concept of Interrupts. The interrupt is a mechanism to –
- **Temporarily suspend the ongoing program,**
- **Pass the control to a subroutine,**
- **Execute the subroutine,**
- **Resume the ongoing/main program.**
- Interrupts can be of various types, such as, Software and Hardware interrupts, Non-maskable and maskable interrupts, etc. Now the 8051 Microcontroller incorporates five interrupts. These are :
- **INT0** – External Hardware Interrupt.
- **TF0** – Timer 0 Overflow Interrupt.
- **INT1** – External Hardware Interrupt.
- **TF1** – Timer 1 Overflow Interrupt.
- **R1/T1** – Serial communication Interrupt.

- **Input/Output Ports –**
- The 8051 Microcontroller needs to be connected to the peripheral devices in order to control their operations. The I/O Ports are responsible for the connection of the Microcontroller to its peripheral devices. There are total Four 8-bit Input/Output Ports present in this Microcontroller.
- Additionally, these are some important features of 8051 microcontroller given as follows :
- **Two 16-bit Timers and Counters.**
- **A Data Pointer and a Program Counter of 16-bit each.**
- **128 User defined Flags.**
- **Four Register banks.**
- **31 General Purpose Registers which are of 8-bit each.**
- **Pin diagram of 8051 Microcontroller –**

Special function Registers(SFR):

8051 microcontroller has 11 SFR divided in 4 groups:

A. Timer/Counter register: 8051 microcontroller has 2-16 bit Timer/counter registers called Timer-reg-T0 And
Timer/counter Reg-T1.Each register is 16 bit register divide into lower and higher byte register as shown below:
These register are used to hold initial no. of count. All of the 4 register are byte addressable.

1. **Timer control register:** 8051 microcontroller has two 8-bit timer control register i.e. TMOD and TCON register.

    1) TMOD Register: it is 8-bit register. Its address is 89H. It is byte addressable.
    It used to select mode and control operation of time by writing control word.

    2). TCON register: It is 8-bit register. Its address is 88H. It is byte addressable.
    Its MSB 4-bit are used to control operation of timer/ counter and LSB 4-bit are used for external interrupt control.

TMOD Register:

GATE	C/T	M1	M0	GATE	C/T	M1	M0
TIMER 1				TIMER 0			

- ***Gate :*** *When set, timer only runs while INT(0,1) is high.*
- ***C/T :*** *Counter/Timer select bit.*
- ***M1 :*** *Mode bit 1.*
- ***M0 :*** *Mode bit 0.*

M1	M0	MODE
0	0	13-bit timer mode
0	1	16-bit timer mode
1	0	8-bit auto-reload mode
1	1	split mode

TCON Register:

TF1	TR1	TF0	TR0	IE1	IT1	IE0	IT0

- *TF: Overflow flag*
  - *Set by hardware on Timer/Counter overflow*
  - *Cleared by hardware when processor vectors to interrupt routine*
- *TR: Run control bit*
  - *Set/Cleared by software to turn Timer/Counter on/off*
- *IE: Interrupt Edge flag*
  - *Set by hardware when external interrupt edge detected*
  - *Cleared when interrupt processed*
- *IT: Interrupt Type control bit*
  - *Set/Cleared by software to specify falling edge/low level triggered external interrupts*

- ***TF1: Timer 1 overflow flag.*** ***TR1: Timer 1 run control bit.***
- ***TF0: Timer 0 overflag.*** ***TR0: Timer 0 run control bit.***
- ***IE1: External interrupt 1 edge flag.*** ***IT1: External interrupt 1 type flag.***
- ***IE0: External interrupt 0 edge flag.*** ***IT0: External interrupt 0 type flag.***

2.. Serial data register: 8051 micro controller has 2 serial data register viz. SBUF and SCON.

1. Serial buffer register (SBUF): it is 8-bit register. It is byte addressable .

Its address is 99H. It is used to hold data which is to be transferred serially.

2. Serial control register (SCON): it is 8-bit register. It is bit/byte addressable.

Its address is 98H. The 8-bit loaded into this register controls the operation

of serial communication.

3. Interrupt register: 8051 μC has 2 8-bit interrupt register

.1. Interrupt enable register (IE): it is 8-bit register. It is bit/byte addressable. Its address is A8H.

it is used to enable and disable function of interrupt.

2. Interrupt priority register (IP): It is 8-bit register. It is bit/byte addressable.

Its address is B8H.it is used to select low or high level priority of each individual interrupts.

4. Power control register (PCON): it is 8-bit register. It is byte addressable .Its address is 87H.

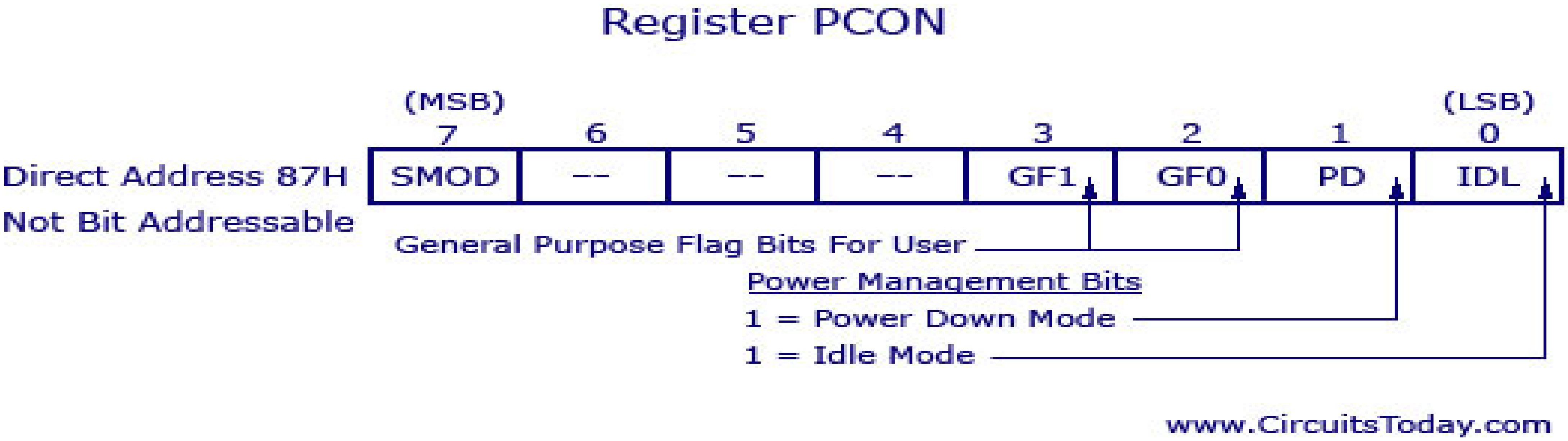
its bits are used to control mode of power saving circuit, either idle or power down mode

and also one bit is used to modify baud rate of serial communication.

4. Power control register (PCON): it is 8-bit register. It is byte addressable .Its address is 87H.

its bits are used to control mode of power saving circuit, either idle or power down mode

and also one bit is used to modify baud rate of serial communication.



Register PCON

	(MSB) 7	6	5	4	3	2	1	(LSB) 0
Direct Address 87H Not Bit Addressable	SMOD	--	--	--	GF1	GF0	PD	IDL

General Purpose Flag Bits For User

Power Management Bits

1 = Power Down Mode

1 = Idle Mode

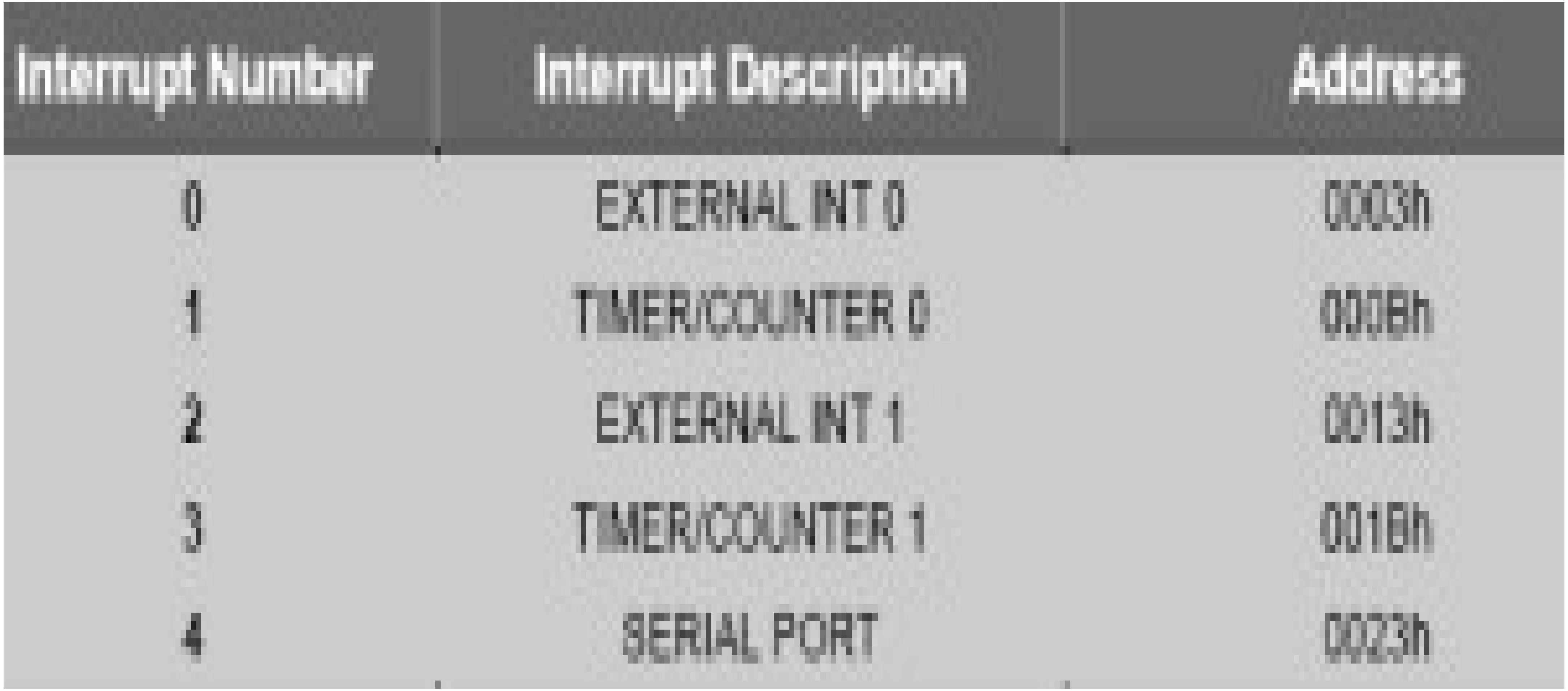
www.CircuitsToday.com

Types of Interrupts in 8051 Microcontroller

- The 8051 microcontroller can recognize five different events that cause the main program to interrupt from the normal execution. These five sources of interrupts in 8051are:

1. Timer 0 overflow interrupt- TF0
2. Timer 1 overflow interrupt- TF1
3. External hardware interrupt- INT0
4. External hardware interrupt- INT1
5. Serial communication interrupt- RI/TI

The Timer and Serial interrupts are internally generated by the microcontroller, whereas the external interrupts are generated by additional interfacing devices or switches that are externally connected to the microcontroller. These external interrupts can be edge triggered or level triggered. When an interrupt occurs, the microcontroller executes the interrupt service routine so that memory location corresponds to the interrupt that enables it. The Interrupt corresponding to the memory location is given in the interrupt vector table below.

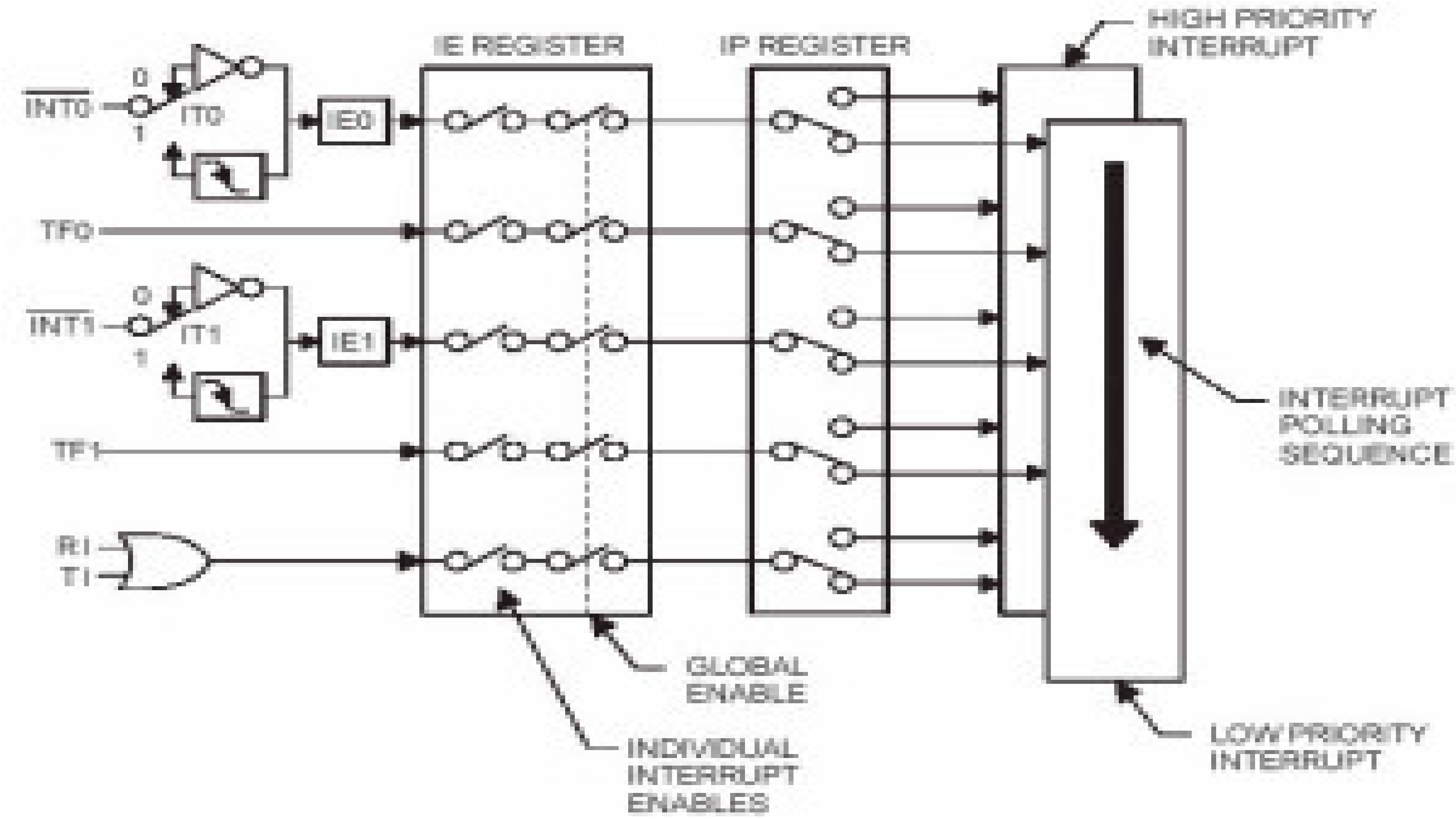


Interrupt Number	Interrupt Description	Address
0	EXTERNAL INT 0	0003h
1	TIMER/COUNTER 0	000Bh
2	EXTERNAL INT 1	0013h
3	TIMER/COUNTER 1	001Bh
4	SERIAL PORT	0023h

Interrupt Structure of 8051 Micro controller

Upon ‘RESET’ all the interrupts get disabled, and therefore, all these interrupts must be enabled by a software. In all these five interrupts, if anyone or all are activated, this sets the corresponding interrupt flags as shown in the figure. All these interrupts can be set or cleared by bit in some special function register that is Interrupt Enabled (IE), and this in turn depends on the priority, which is executed by IP interrupt priority register.

interrupt structure of 8051 microcontroller



Interrupt Enable (IE) Register

- **Interrupt Enable (IE) Register:** This register is responsible for enabling and disabling the interrupt. It is a bit addressable register in which EA must be set to one for enabling interrupts. The corresponding bit in this register enables particular interrupt like timer, external and serial inputs. In the below IE register, bit corresponding to 1 activates the interrupt and 0 disables the interrupt.

Interrupt Enable (IE) Register

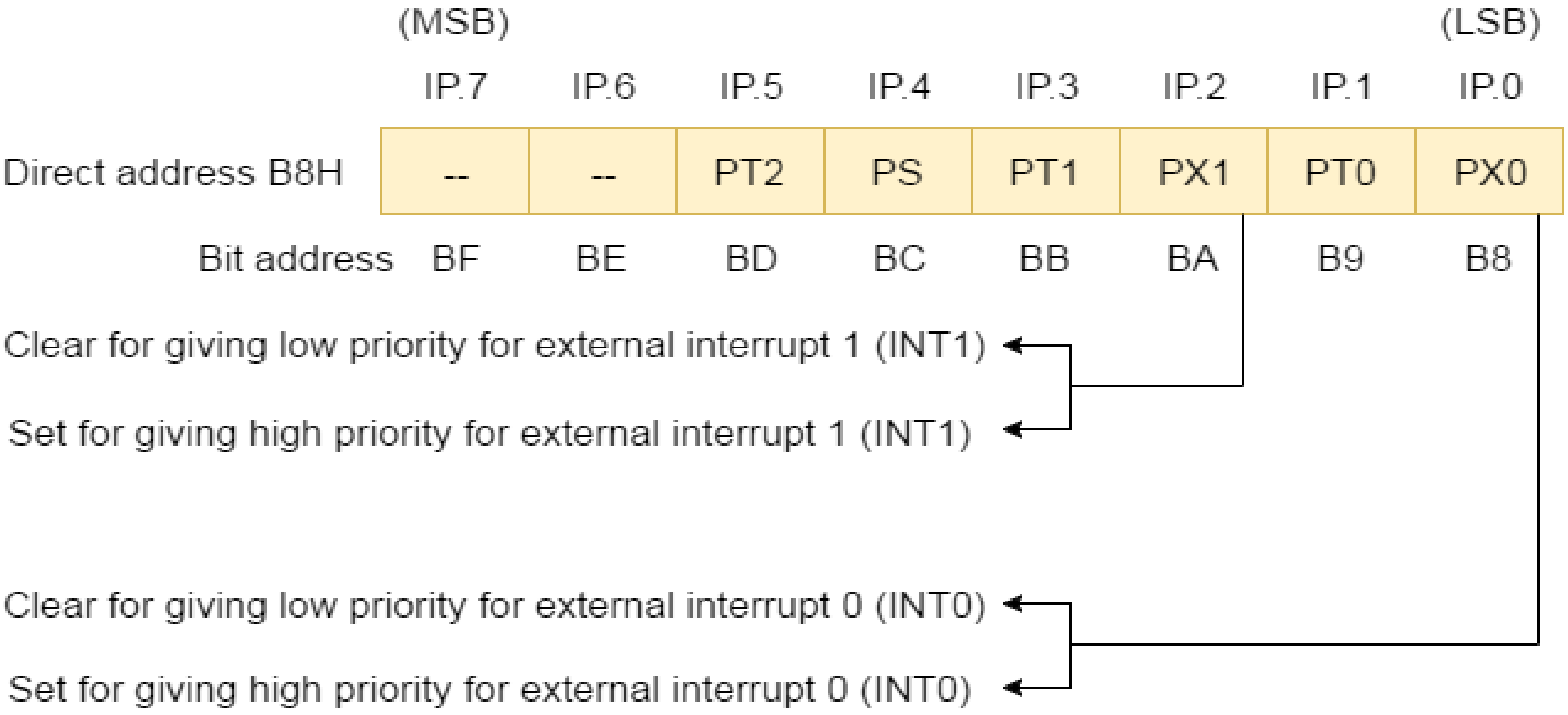
EA	--	--	ES	ET1	EX1	ET0	EX0

EA	IE.7	Disables all interrupts, If EA=0, no interrupt will be acknowledged. If EA=1, interrupt source is individually enable or disabled by setting or clearing its enable bit.
--	IE.6	Not implemented, reserved for future use*.
--	IE.5	Not implemented, reserved for future use*.
ES	IE.4	Enable or disable the Serial port interrupt.
ET1	IE.3	Enable or disable the Timer 1 overflow interrupt.
EX1	IE.2	Enable or disable External interrupt 1.
ET0	IE.1	Enable or disable the Timer 0 overflow interrupt.
EX0	IE.0	Enable or disable External interrupt 0.

Interrupt Priority Register

- **Interrupt Priority Register (IP):** It is also possible to change the priority levels of the interrupts by setting or clearing the corresponding bit in the Interrupt priority (IP) register as shown in the figure. This allows the low priority interrupt to interrupt the high-priority interrupt, but prohibits the interruption by another low-priority interrupt. Similarly, the high-priority interrupt cannot be interrupted. If these interrupt priorities are not programmed, the microcontroller executes in predefined manner and its order is INT0, TF0, INT1, TF1, and SI.

Interrupt Priority Register



	(MSB)							(LSB)
	IP.7	IP.6	IP.5	IP.4	IP.3	IP.2	IP.1	IP.0
Direct address B8H	--	--	PT2	PS	PT1	PX1	PT0	PX0
Bit address	BF	BE	BD	BC	BB	BA	B9	B8

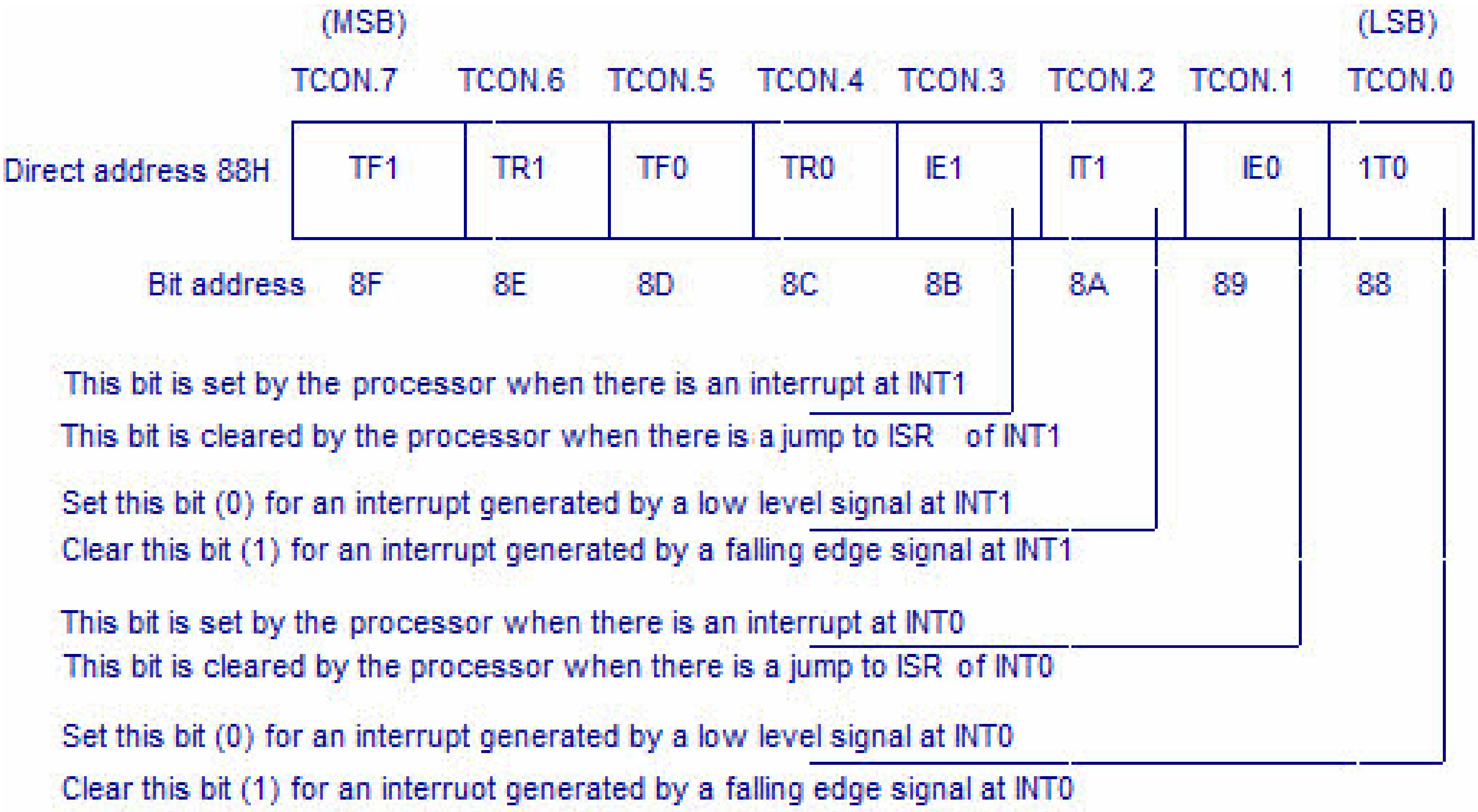
Clear for giving low priority for external interrupt 1 (INT1)

Set for giving high priority for external interrupt 1 (INT1)

Clear for giving low priority for external interrupt 0 (INT0)

Set for giving high priority for external interrupt 0 (INT0)

- **TCON Register:** In addition to the above two registers, the TCON register specifies the type of external interrupt to the 8051 microcontroller, as shown in the figure. The two external interrupts, whether edge or level triggered, specify by this register by a set, or cleared by appropriate bits in it. And, it is also a bit addressable register.



	(MSB)							(LSB)
	TCON.7	TCON.6	TCON.5	TCON.4	TCON.3	TCON.2	TCON.1	TCON.0
Direct address 88H	TF1	TR1	TF0	TR0	IE1	IT1	IE0	1T0
Bit address	8F	8E	8D	8C	8B	8A	89	88

This bit is set by the processor when there is an interrupt at INT1
This bit is cleared by the processor when there is a jump to ISR of INT1

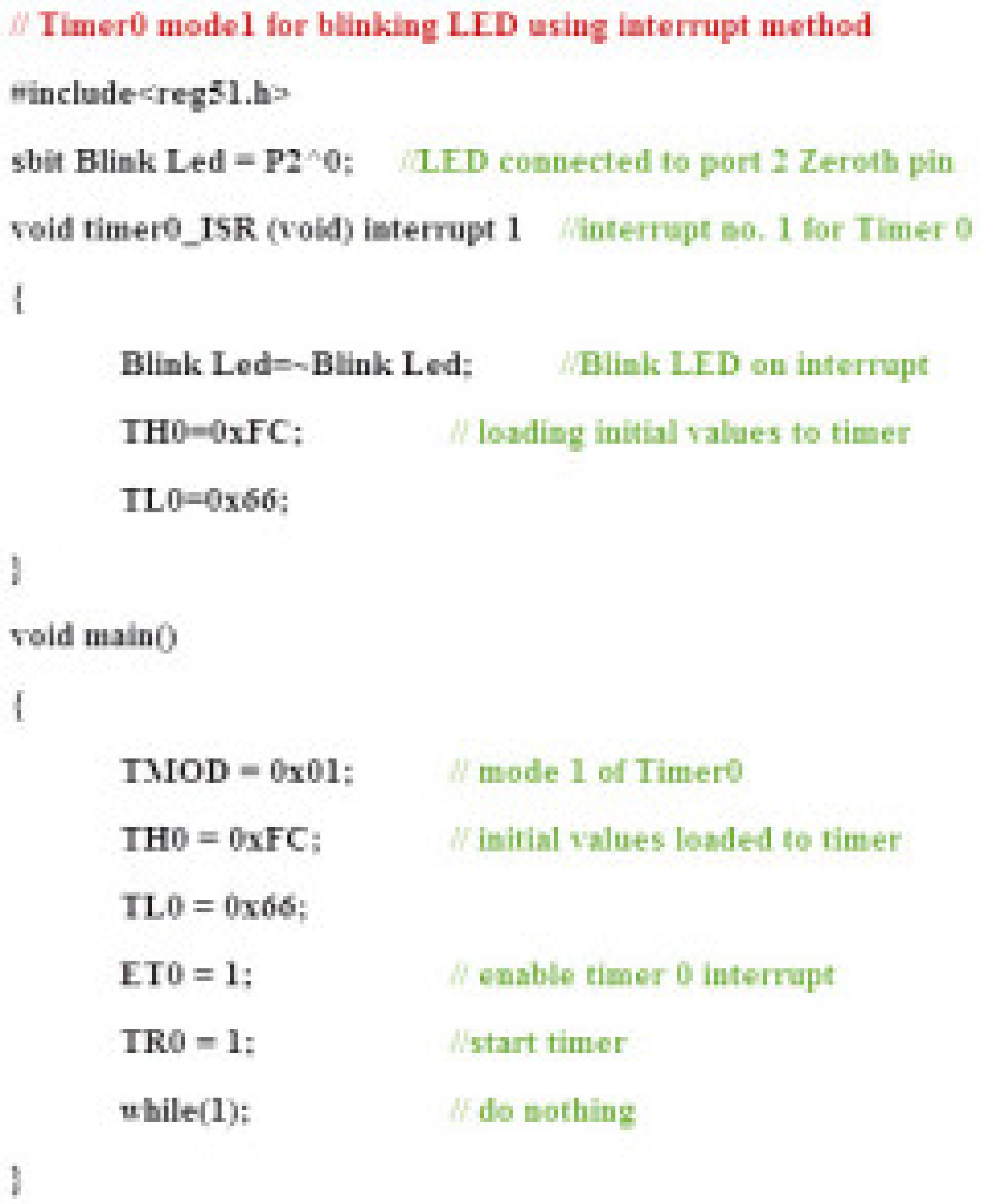
Set this bit (0) for an interrupt generated by a low level signal at INT1
Clear this bit (1) for an interrupt generated by a falling edge signal at INT1

This bit is set by the processor when there is an interrupt at INT0
This bit is cleared by the processor when there is a jump to ISR of INT0

Set this bit (0) for an interrupt generated by a low level signal at INT0
Clear this bit (1) for an interruot generated by a falling edge signal at INT0

Interrupt Programming in 8051

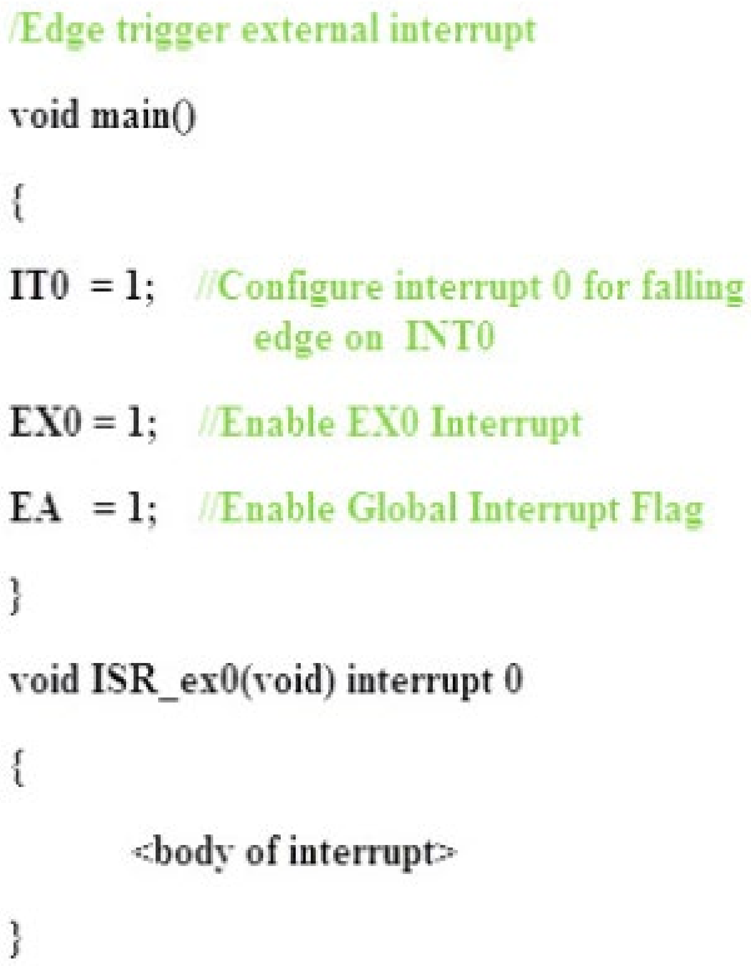
- **1.Timer Interrupt Programming**
- Timer 0 and Timer 1 interrupts are generated by the timer register bits TF0 and TF1. These interrupts programming by C code involves:
- Selecting the timer by configuring TMOD register and its mode of operation.
- Choosing and loading the initial values of TLx and THx for appropriate modes.
- Enabling the IE registers and corresponding timer bit in it.
- Setting the timer run bit to start the timer.
- Writing the subroutine for the timer for time required and clear timer value TRx at the end of subroutine.



```
// Timer0 mode1 for blinking LED using interrupt method

#include<reg51.h>

sbit Blink Led = P2^0;     //LED connected to port 2 Zeroth pin

void timer0_ISR (void) interrupt 1     //interrupt no. 1 for Timer 0

{

        Blink Led=~Blink Led;          //Blink LED on interrupt

        TH0=0xFC;                 // loading initial values to timer

        TL0=0x66;

}

void main()

{

        TMOD = 0x01;          // mode 1 of Timer0

        TH0 = 0xFC;           // initial values loaded to timer

        TL0 = 0x66;

        ET0 = 1;              // enable timer 0 interrupt

        TR0 = 1;              //start timer

        while(1);             // do nothing

}
```

- **2.External Hardware Interrupt Programming**
- 8051 microcontrollers consists of two external hardware interrupts: INT0 and INT1 as discussed earlier. These are enabled at pin 3.2 and pin 3.3. These can be edge triggered or level triggered. In level triggering, the low at pin 3.2 enables the interrupt, while at pin 3.2 – the high to low transition enables the edge triggered interrupt. This edge triggering or level triggering is decided by the TCON register that has been discussed above. The programming procedure in 8051 is as follows:
- Enable the corresponding bit of external interrupt in IE register.
- If it is level triggering, just write the subroutine appropriate to this interrupt, or else enable the TCON register bit corresponding to the edge triggered interrupt – whether it is INT0 or INT1.

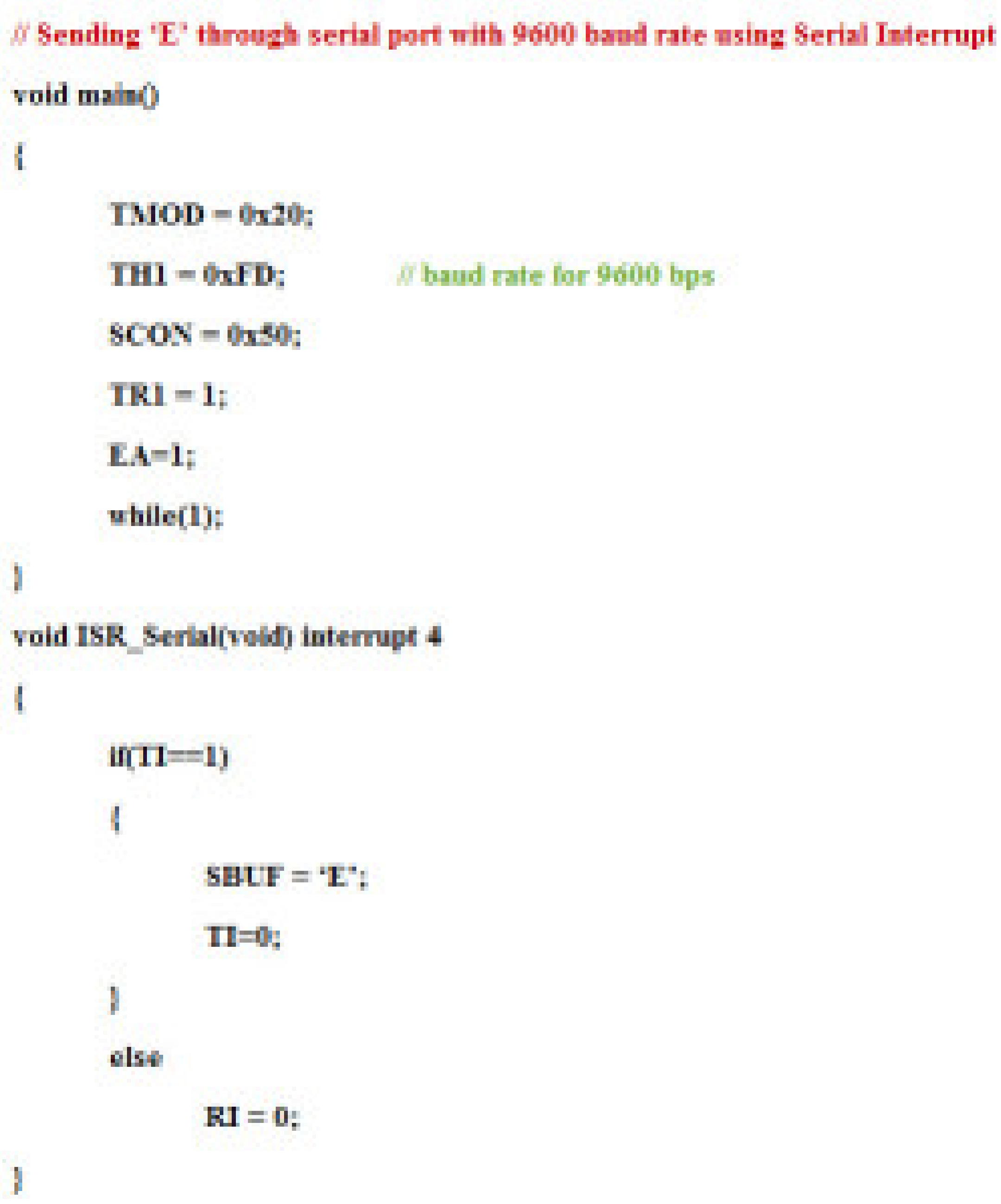
External Hardware Interrupt Programming



```
/Edge trigger external interrupt

void main()

{

IT0  = 1;    //Configure interrupt 0 for falling
                   edge on  INT0

EX0 = 1;    //Enable EX0 Interrupt

EA   = 1;    //Enable Global Interrupt Flag

}

void ISR_ex0(void) interrupt 0

{

          <body of interrupt>

}
```

External Hardware Interrupt Programming

- **3.Serial Communication Interrupt Programming**
- Serial communication interrupts come into picture when there is a need to send or receive data. Since one interrupt bit is set for both TI (Transfer Interrupt) and RI (Receiver Interrupt) flags, Interrupt Service routine must examine these flags to know the actual interrupt.
- The logical OR operation of these two flags (RI ands TI) causes this interrupt, and it is cleared by the software alone. Here, a special register SCON is used for controlling communication operation by enabling the corresponding bits in it.
- Configure the IE register for enabling serial interrupt
- Configure the SCON register for receiving or transferring operation
- Write subroutine for this interrupt with appropriate function and clear TI or RI flags with in this routine.



```
// Sending 'E' through serial port with 9600 baud rate using Serial Interrupt
void main()
{
        TMOD = 0x20;
        TH1 = 0xFD;            // baud rate for 9600 bps
        SCON = 0x50;
        TR1 = 1;
        EA=1;
        while(1);
}
void ISR_Serial(void) interrupt 4
{
        if(TI==1)
        {
                SBUF = 'E';
                TI=0;
        }
        else
                RI = 0;
}
```

- Serial interrupt programming

Interfacing

Interfacing is one of the important concepts in **microcontroller 8051** because the **microcontroller** is a CPU that can perform some operation on a data and gives the output. However to perform the operation we need an input device to enter the data and in turn output device displays the results of the operation.

Interfacing is the process of connecting devices together so that they can exchange the information and that proves to be easier to write the programs. There are different type of input and output devices as for our requirement such as LEDs, LCDs, 7segment, keypad, motors and other devices.

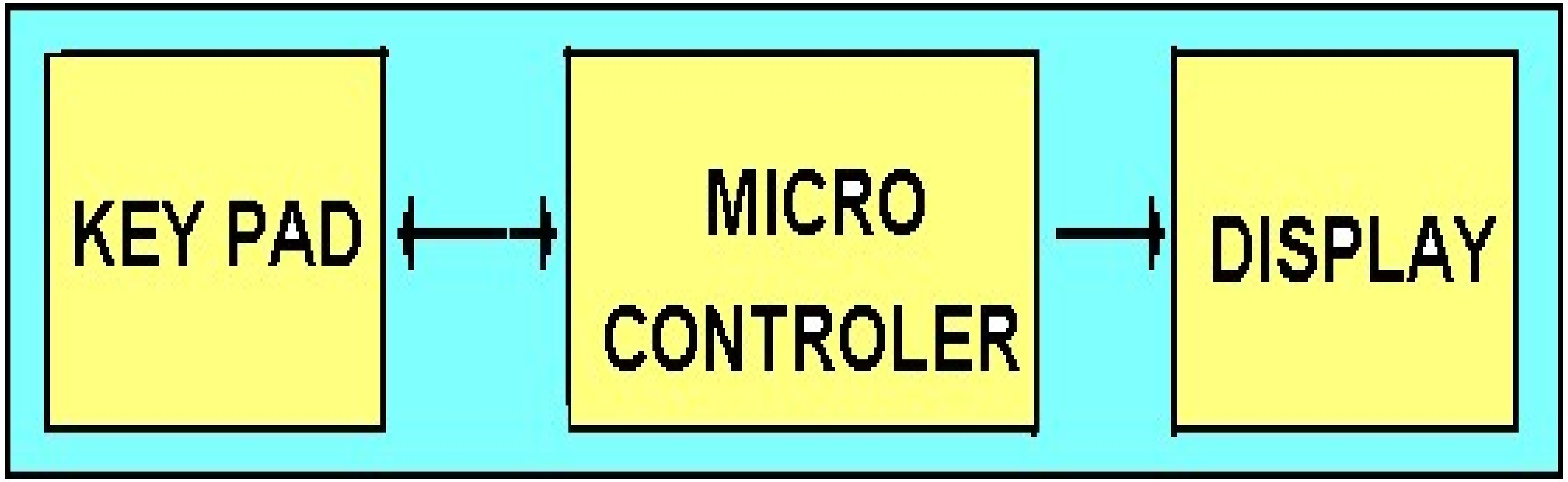
Interfacing the Keyboard to 8051 microcontroller

- The key board here we are interfacing is a matrix keyboard. This key board is designed with a particular rows and columns. These rows and columns are connected to the microcontroller through its ports of the micro controller 8051. We normally use 8*8 matrix key board. So only two ports of 8051 can be easily connected to the rows and columns of the key board.
- When ever a key is pressed, a row and a column gets shorted through that pressed key and all the other keys are left open. When a key is pressed only a bit in the port goes high. Which indicates microcontroller that the key is pressed. By this high on the bit key in the corresponding column is identified.

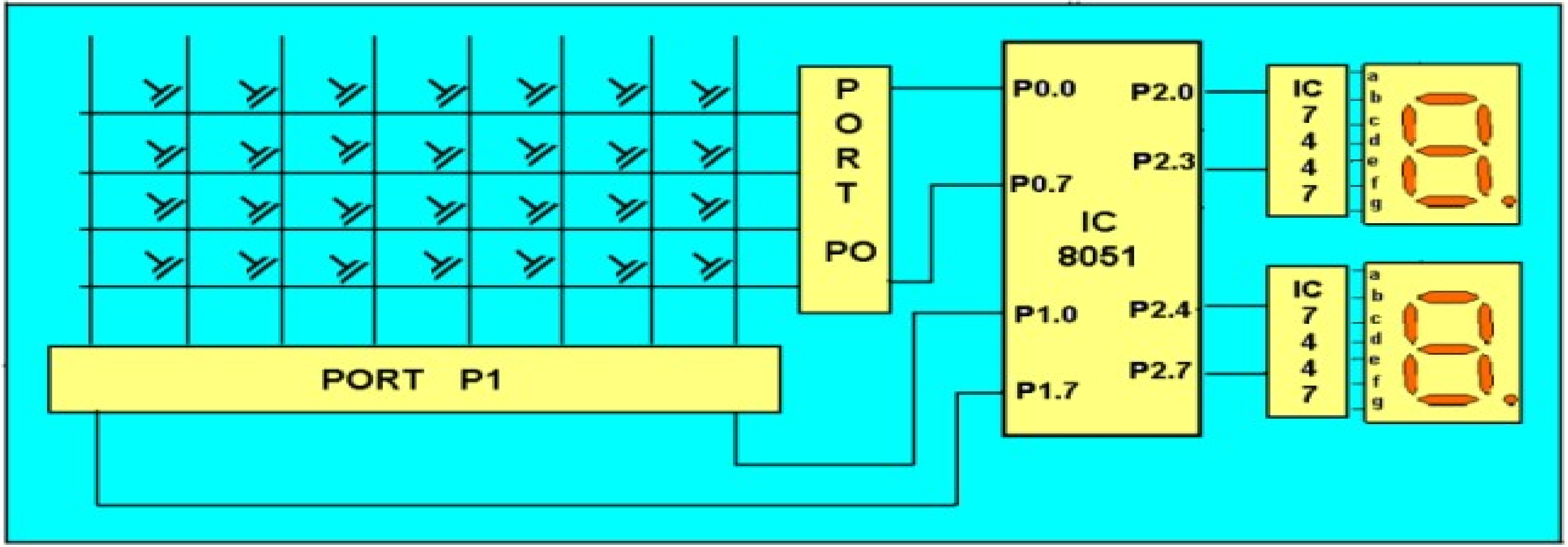
- Once we are sure that one of key in the key board is pressed next our aim is to identify that key. To do this we firstly check for particular row and then we check the corresponding column the key board.
- To check the row of the pressed key in the keyboard, one of the row is made high by making one of bit in the output port of 8051 high . This is done until the row is found out. Once we get the row next out job is to find out the column of the pressed key. The column is detected by contents in the input ports with the help of a counter. The content of the input port is rotated with carry until the carry bit is set.

- The contents of the counter is then compared and displayed in the display. This display is designed using a seven segment display and a BCD to seven segment decoder IC 7447.
- The BCD equivalent number of counter is sent through output part of 8051 displays the number of pressed key.

Circuit diagram of *INTERFACING KEY BOARD TO 8051.*



Keyboard is organized in a matrix of rows and columns as shown in the figure. The microcontroller accesses both rows and columns through the port.



1.The 8051 has 4 I/O ports P0 to P3 each with 8 I/O pins, P0.0 to P0.7,P1.0 to P1.7, P2.0 to P2.7, P3.0 to P3.7. The one of the port P1 (it understood that P1 means P1.0 to P1.7) as an I/P port for microcontroller 8051, port P0 as an O/P port of microcontroller 8051 and port P2 is used for displaying the number of pressed key.

2.Make all rows of port P0 high so that it gives high signal when key is pressed.

2. See if any key is pressed by scanning the port P1 by checking all columns for non zero condition.

3. If any key is pressed, to identify which key is pressed make one row high at a time.

4. Initiate a counter to hold the count so that each key is counted

5. Check port P1 for nonzero condition. If any nonzero number is there in [accumulator], start column scanning by following step 9.
6. Otherwise make next row high in port P1.
7. Add a count of 08h to the counter to move to the next row by repeating steps from step 6.
8. If any key pressed is found, the [accumulator] content is rotated right through the carry until carry bit sets, while doing this increment the count in the counter till carry is found.
9. Move the content in the counter to display in data field or to memory location
10. To repeat the procedures go to step 2.

Program to interface matrix keyboard to microcontroller 8051

Start of main program: to check that whether any key is pressed

```
start:  mov a,#00h
        mov p1,a          ;making all rows of port p1 zero
        mov a,#0fh
        mov p1,a          ;making all rows of port p1 high
press:  mov a,p2
        jz press          ;check until any key is pressed
```

after making sure that any key is pressed

```
        mov a,#01h        ;make one row high at a time
        mov r4,a
        mov r3,#00h       ;initiating counter
next:   mov a,r4
        mov p1,a          ;making one row high at a time
        mov a,p2          ;taking input from port A
        jnz colscan       ;after getting the row jump to check
                           column
        mov a,r4
        rl a              ;rotate left to check next row
        mov r4,a
        mov a,r3
        add a,#08h        ;increment counter by 08 count
        mov r3,a
        sjmp next         ;jump to check next row
```

- *after identifying the row to check the colomn following steps are followed*
-

```
    colscan:    mov r5,#00h
         in:    rrc a           ;rotate right with carry until get the carry
                jc out          ;jump on getting carry
                inc r3          ;increment one count
                jmp in
        out:    mov a,r3
                da a            ;decimal adjust the contents of counter
                                     before display
                mov p2,a
                jmp start       ;repeat for check next key
```

Alphanumeric Displays

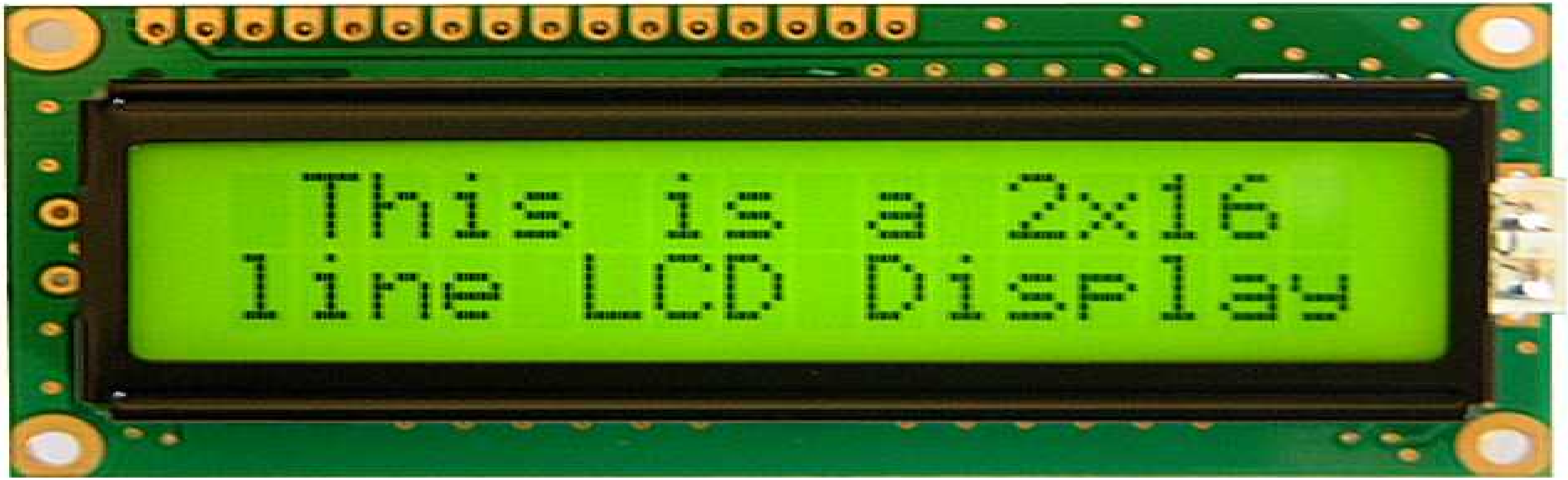
[redacted]

1 x 16 Display

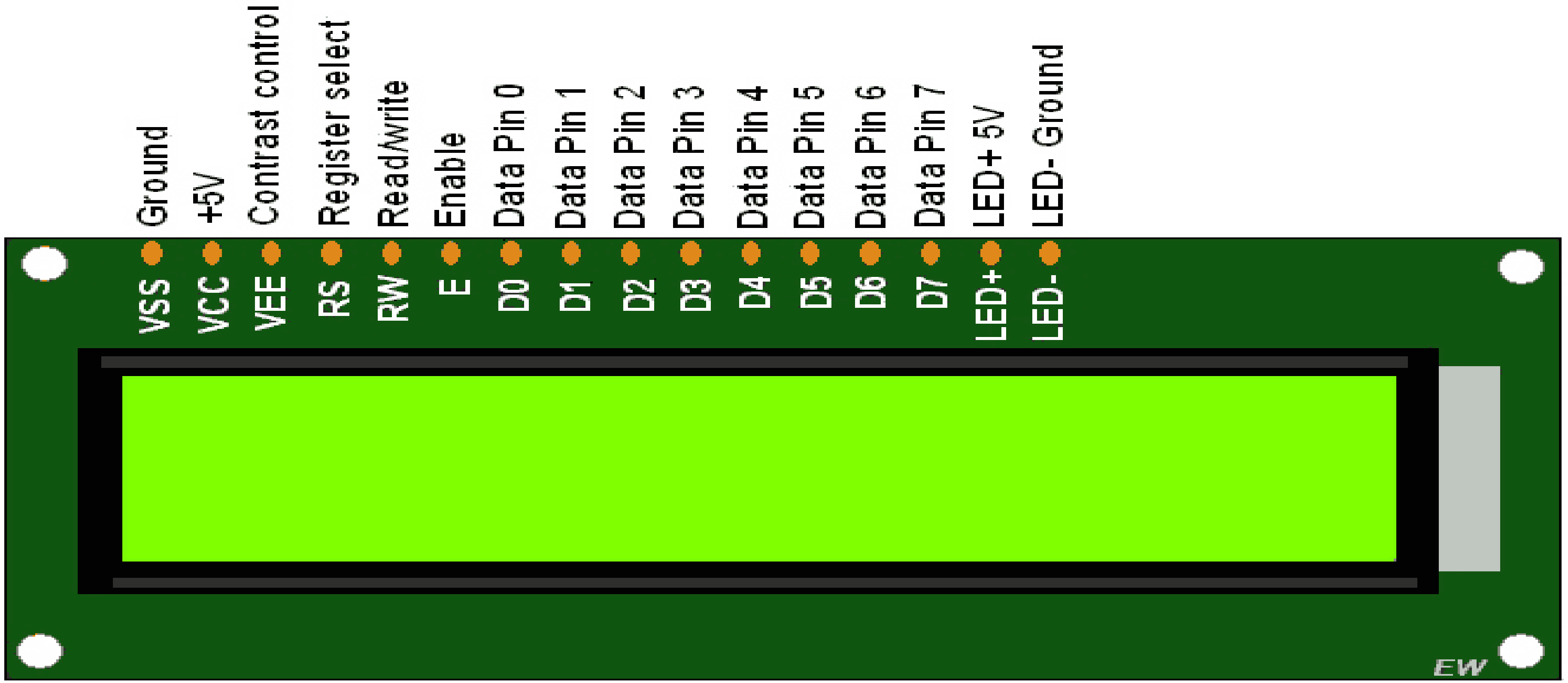
2 x 16 Display

4 x 16 Display

LCD DIS PLAY



LCD INTERFACING WITH 8051 MICROCONTROLLER



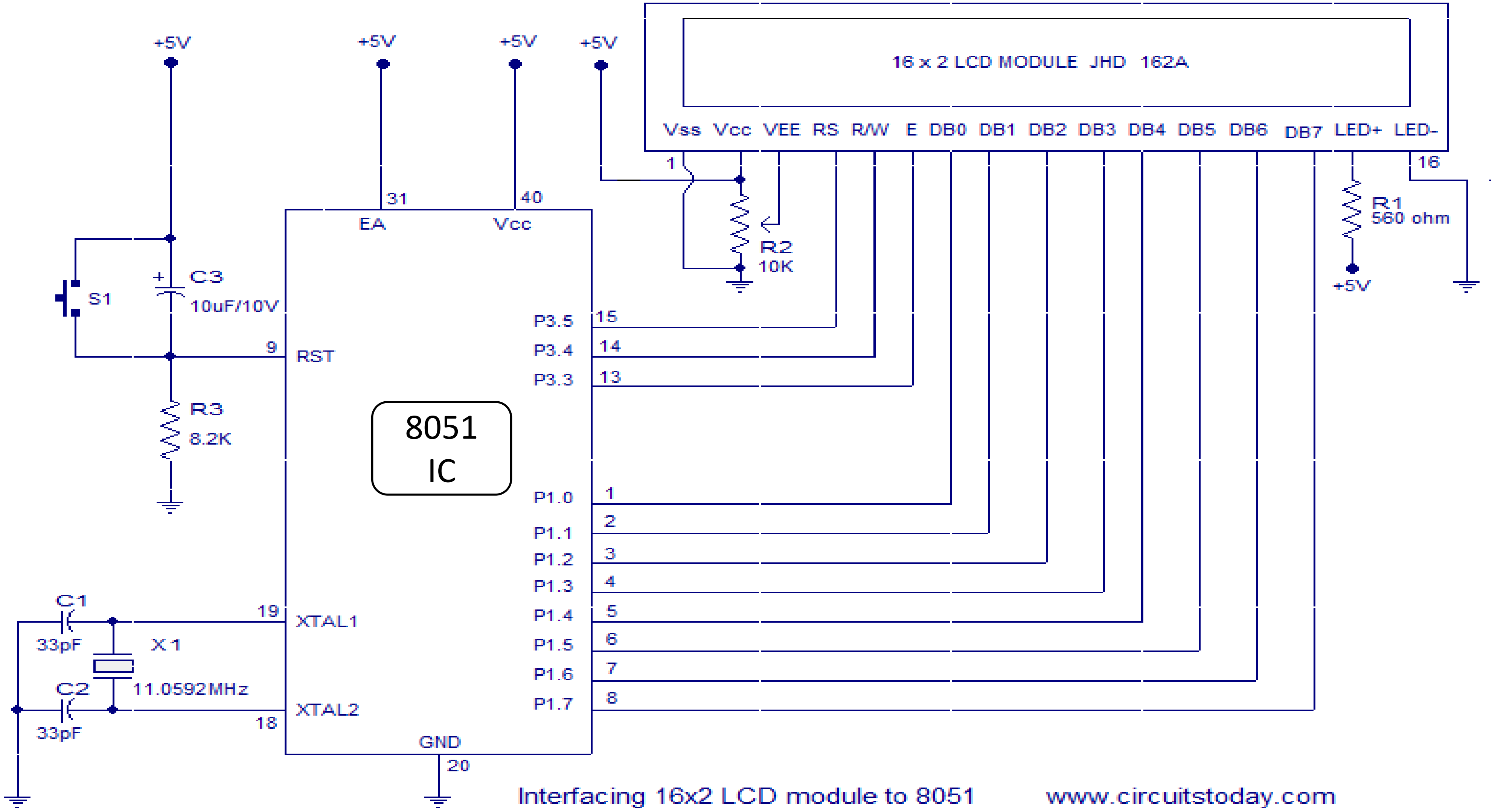
- **A Brief Note on 16×2 LCD**
  16×2 Liquid Crystal Display which will display the 32 characters at a time in two rows (16 characters in one row). Each character in the display is of size 5×7 pixel matrix. This matrix differs for different 16×2 LCD modules,. There are 16 pins in the LCD module, the pin configuration us given below

PIN NO	NAME	FUNCTION
1	VSS	Ground pin
2	VCC	Power supply pin of 5V
3	VEE	Used for adjusting the contrast commonly attached to the potentiometer.
4	RS	RS is the register select pin used to write display data to the LCD (characters), this pin has to be high when writing the data to the LCD. During the initializing sequence and other commands this pin should low.
5	R/W	Reading and writing data to the LCD for reading the data R/W pin should be high (R/W=1) to write the data to LCD R/W pin should be low (R/W=0)
6	E	Enable pin is for starting or enabling the module. A high to low pulse of about 450ns pulse is given to this pin.
7	DB0	
8	DB1	
9	DB2	
10	DB3	
11	DB4	DB0-DB7 Data pins for giving data(normal data like numbers characters or command data) which is meant to be displayed
12	DB5	
13	DB6	
14	DB7	
15	LED+	Back light of the LCD which should be connected to Vcc
16	LED-	Back light of LCD which should be connected to ground.

So by reading the above table you can get a brief idea how to display a character. For displaying a character you should enable the enable pin (pin 6) by giving a pulse of 450ns, after enabling the pin6 you should select the register select pin (pin4) in write mode. To select the register select pin in write mode you have to make this pin high (RS=1), after selecting the register select you have to configure the R/W to write mode that is R/W should be low (R/W=0).

- Follow these simple steps for displaying a character or data
- **E=1;** enable pin should be high
- **RS=1;** Register select should be high
- **R/W=0;** Read/Write pin should be low.
- To send a command to the LCD just follows these steps:
- **E=1;** enable pin should be high
- **RS=0;** Register select should be low
- **R/W=0;** Read/Write pin should be low.

Interfacing 16×2 LCD with 8051 Circuit Diagram



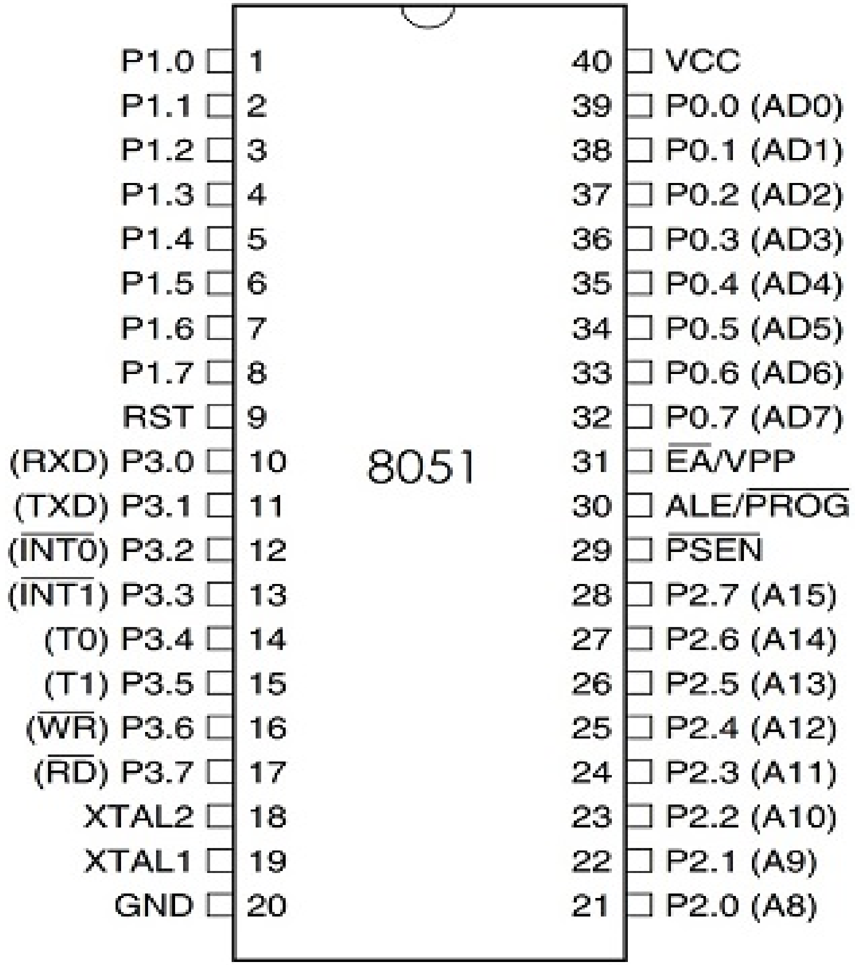
Interfacing 16x2 LCD module to 8051 www.circuitstoday.com

Commands: There are some preset commands which will do a specific task in the LCD. These commands are very important for displaying data in LCD.
The list of commands given below:

COMMAND	FUNCTION
0F	For switching on LCD, blinking the cursor.
1	Clearing the screen
2	Return home.
4	Decrement cursor
6	Increment cursor
E	Display on and also cursor on
80	Force cursor to beginning of the first line
C0	Force cursor to beginning of second line
38	Use two lines and 5x7 matrix
83	Cursor line 1 position 3
3C	Activate second line
0C3	Jump to second line position 3
0C1	Jump to second line position1

EXTERNAL MEMORY INTERFACING WITH 8051 MICROCONTROLLER

PIN DIAGRAMME OF 8051 MICROCONTROLLER



40 - PIN DIP

- **PSEN: PIN 29**

1. This is an output pin.
2. PSEN stands for “program store enable.”
3. Connect this PSEN pin to the OE pin of the ROM to enable access to data.
4. It is an active low output signal.
5. It is used to enable/read external program memory (ROM).
   1. When [PSEN] = 0, then external program memory becomes enabled, and microcontroller reads the content of external memory location.
   2. Therefore, it is connected to (OE) of external ROM. It is activated twice every external ROM memory cycle.
   3. When [PSEN] = 1, then the data cannot be read from any external program memory, then the microcontroller has to depend on the on-chip ROM to store the program code.

- **ALE: PIN 30**

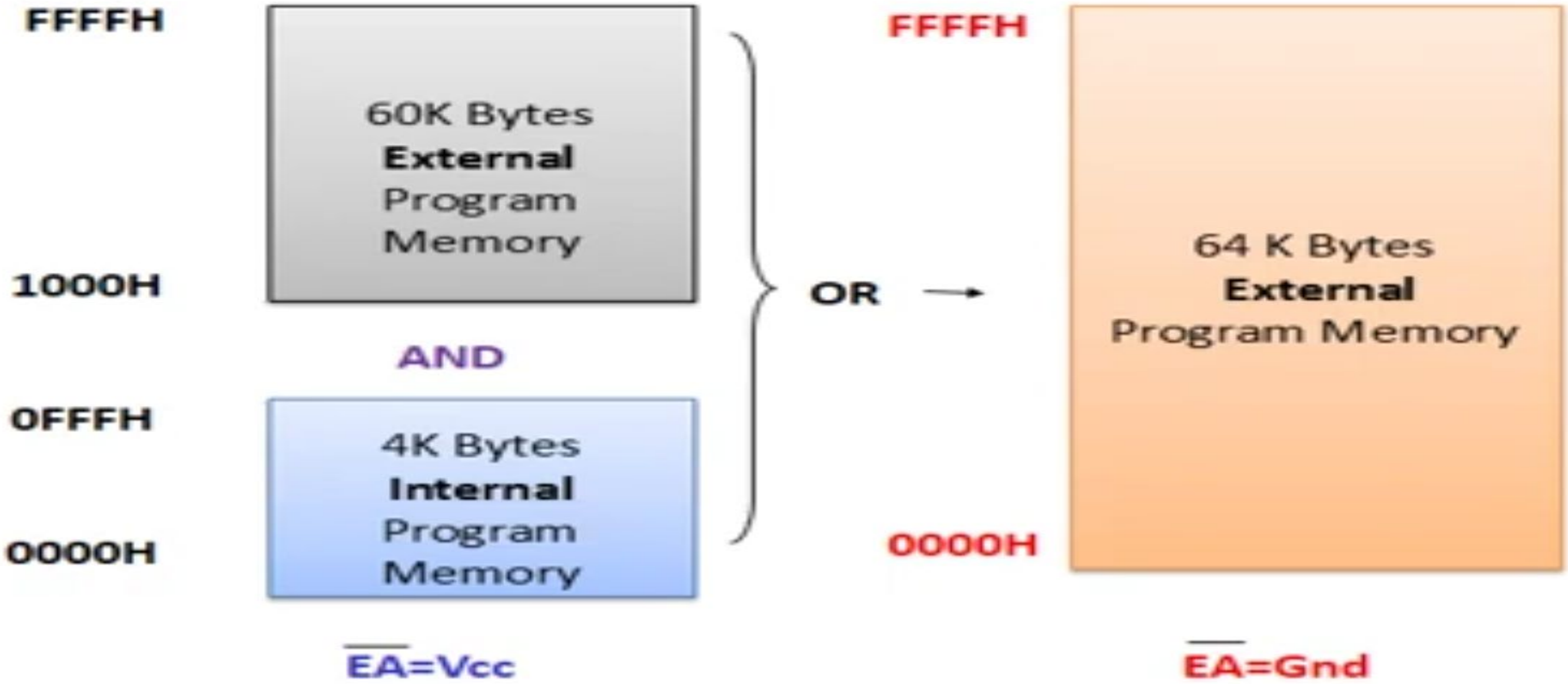
1. Port 0 of 8051 can be used to access the address bus and the data bus.
2. The ALE pin is used for de-multiplexing the address and the data by connecting to the G-pin of the 74LS373 latch.
3. It is also Active High

- **EA: PIN 31**

1. EA is the External Access pin of 8051 microcontrollers.
2. The EA pin is connected to GND to indicate that the code is stored completely in the external program memory (64kB).
3. To use both on-chip ROM (4kB) and external ROM (60kB) together, the EA pin is connected to the +5V VCC supply.
4. The overline represents active-low operation, i.e. Turns ON when a low pulse/signal is provided.

- **RD: P3.7**
- RD is used as a read control signal pin.
- **WR: P3.6**
- WR is used as a write control signal pin.

8051 Program Memory



FFFFH

60K Bytes **External** Program Memory

1000H

AND

0FFFH

4K Bytes **Internal** Program Memory

0000H

$\overline{EA}$=Vcc

OR →

FFFFH

64 K Bytes **External** Program Memory

0000H

$\overline{EA}$=Gnd

- All memory chips have one or more than one pins called the Chip Select (CS) pins (Chip Enable (CE) pins). These pins are commonly active-low pins, and we have to activate it to access the chip it belongs to.
- In connecting a memory chip to the 8051, note the following points:

1. The data bus of the 8051 is connected directly to the data pins of the memory chip.
2. Control signals RD (read) and WR (memory write) from the 8051 are connected to the OE (output enable) and WE (write enable) pins of the memory chip.
3. In the case of the address buses, while the lower bits of the address from the 8051 go directly to the memory chip address pins, the upper ones are used to activate the CS/CE pin of the memory chip via an additional decoding circuitry. The latter is known as Chip Select Logic.

- The 74LS138 latch this is chip select logic ckt

- **74LS138 as a decoder circuit**

1. The three inputs A, B, and C generate eight active low outputs Y0 – Y7
2. We connect each of the Y output to CS of a memory chip, allowing us control over eight memory blocks via a single 74LS138.
3. In the 74LS138, the inputs to A, B, and C activates the output. Also, there are three additional inputs, G2A, G2B, and G1.
4. G2A and G2B are the enable input pins and are active low. G1 is the enable input pin that is active high.
5. If anyone of the inputs G1, G2A, or G2B is not connected to an address signal, they must be activated permanently either by VCC or ground, depending on the activation level.

- **Port 0 and Port 2**

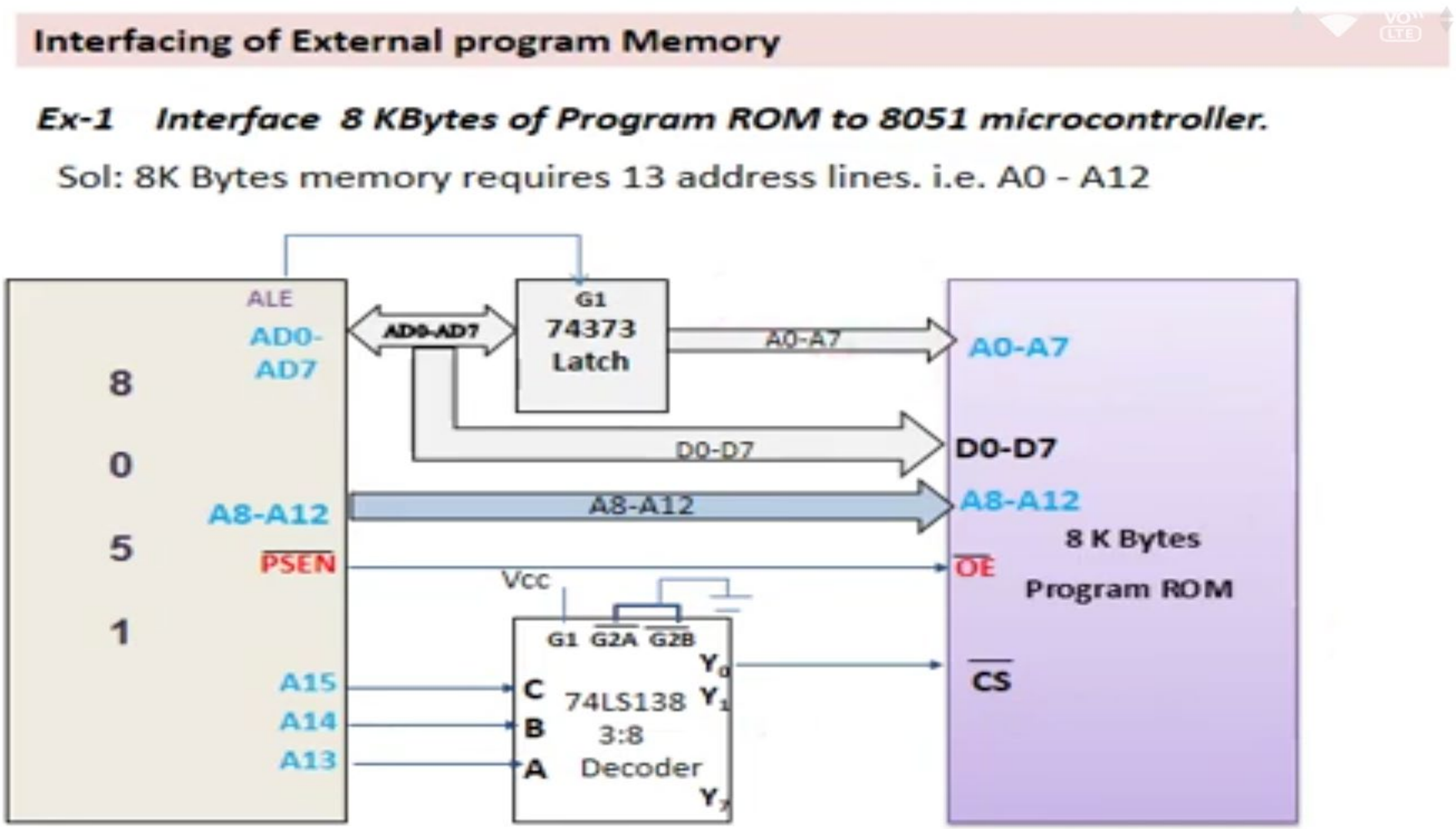
1. These ports provide a 16-bit address to access External Memory.
2. P0: Multiplexed lower order address/data bus: AD0-AD7.
3. P2: Higher-order address bus: A8-A15.
4. When ALE = 0, P0 facilitates data path
5. When ALE = 1, P0 facilitates address path
6. We can address 2^16 = 64Kb memory (i.e. 64Kb code memory and 64Kb data memory) by these 16 address lines from A0 to A15.
7. To extract the address from the P0 pins, we connect P0 to a 74LS373 and use the ALE pin to latch the address.

- **STEPS to interface external program ROM with 8051**

1. **Step 1**: Connect EA pin to ground
2. **Step 2**: Connect the PSEN to the CE and OE.
3. **Step 3**: Then, Port 2 (P2.0 – P2.7) to A8 – A12 pins of ext. ROM.
4. **Step 4**: Connect ALE to G of 74LS373 latch to enable it.
5. **Step 5**: Next, connect the OC of 74LS373 to GND.
6. **Step 6**: Connect Port 0 (P0.0 – P0.7), which consists of both address and data multiplexed into Port 0 to 1D – 8D pins of 74LS373 latch to demultiplex it and 1Q – 8Q of the latch to A0 – A7 of ext. ROM.
7. **Step 7**: Connect Port 0 (P0.0 – P0.7) to D0 – D7 of the ext. ROM.
8. **Step 8**: VPP of ext. ROM to VCC.

- Now you might be wondering what does Ax, Dx, or ADx mean.
- ADx- Multiplexed address and data lines.
- Ax – The address lines determine the location from which the data is to be accessed or be sent.
- Dx- The data lines are used to send/receive the data to/from the external memory.
- Let’s take an example. Suppose we want to activate the chip connected to output Y0 of the decoder.
- Based on the simple working of a decoder, we know that the values of A, B, and C pins (A13, A14,A15) need to be 1 each. So for ABC = “000” we select the Y0 output, and the chip connected to it is now accessible.
- But what range of addressable memory does this occupy? That will depend on the values of the entire 16-bit address line taken as a whole. Note that A15 also has to be 0 at all times since it is connected to G2A, the enable input pin that’s active low. With these four values fixed (A15A14A13A12 = “0001”), we can now vary the remaining pins from 0 to 1 each to get the final address range accessible via output pin Y0.

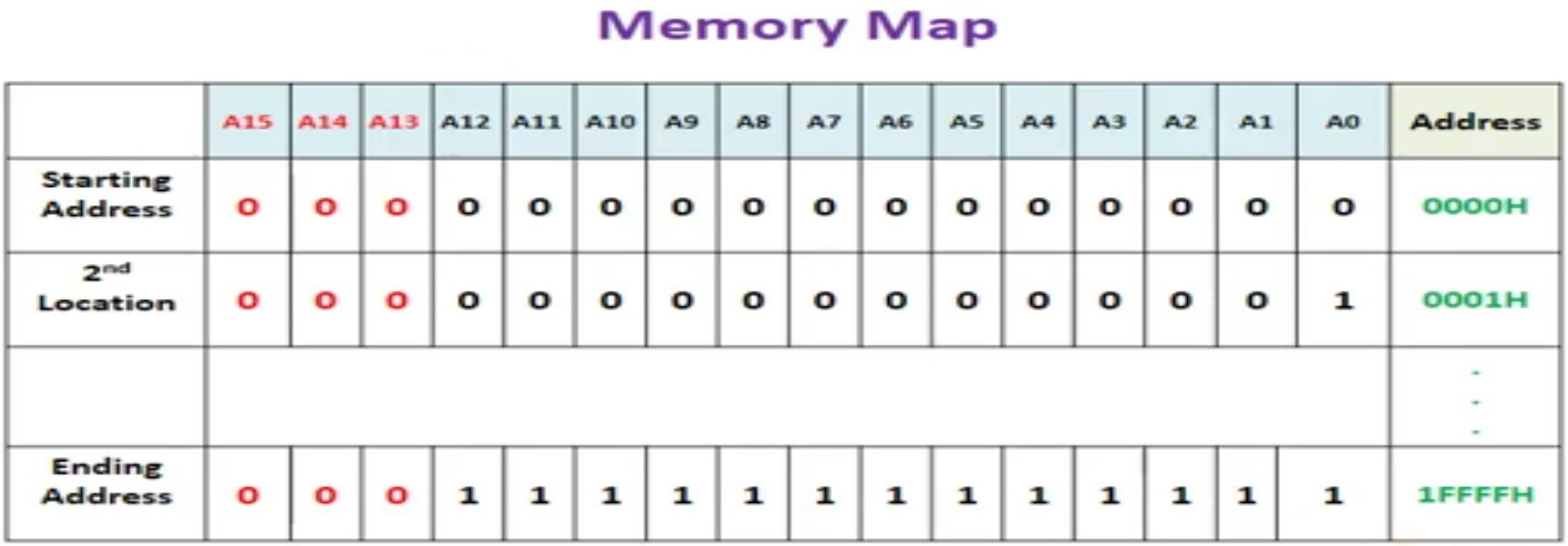
Circuit diagram to interface external program ROM with 8051



Interfacing of External program Memory

Ex-1 Interface 8 KBytes of Program ROM to 8051 microcontroller.

Sol: 8K Bytes memory requires 13 address lines. i.e. A0 - A12

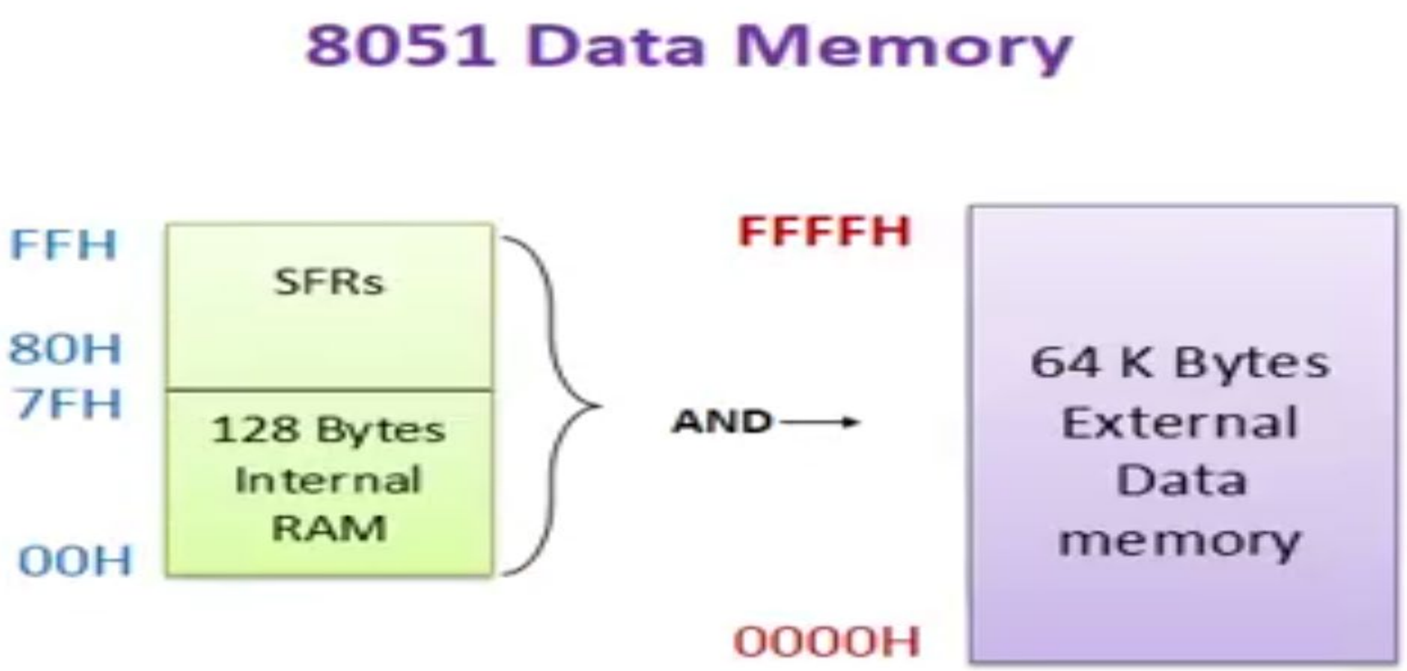


Memory Map

	A15	A14	A13	A12	A11	A10	A9	A8	A7	A6	A5	A4	A3	A2	A1	A0	Address
Starting Address	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0000H
2nd Location	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0001H
																	. . .
Ending Address	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1FFFFH

From the above table, we can see that the address range of Y0 is from 0000H to 1FFFH.

EXTERNAL DATA MEMORY OF 8051



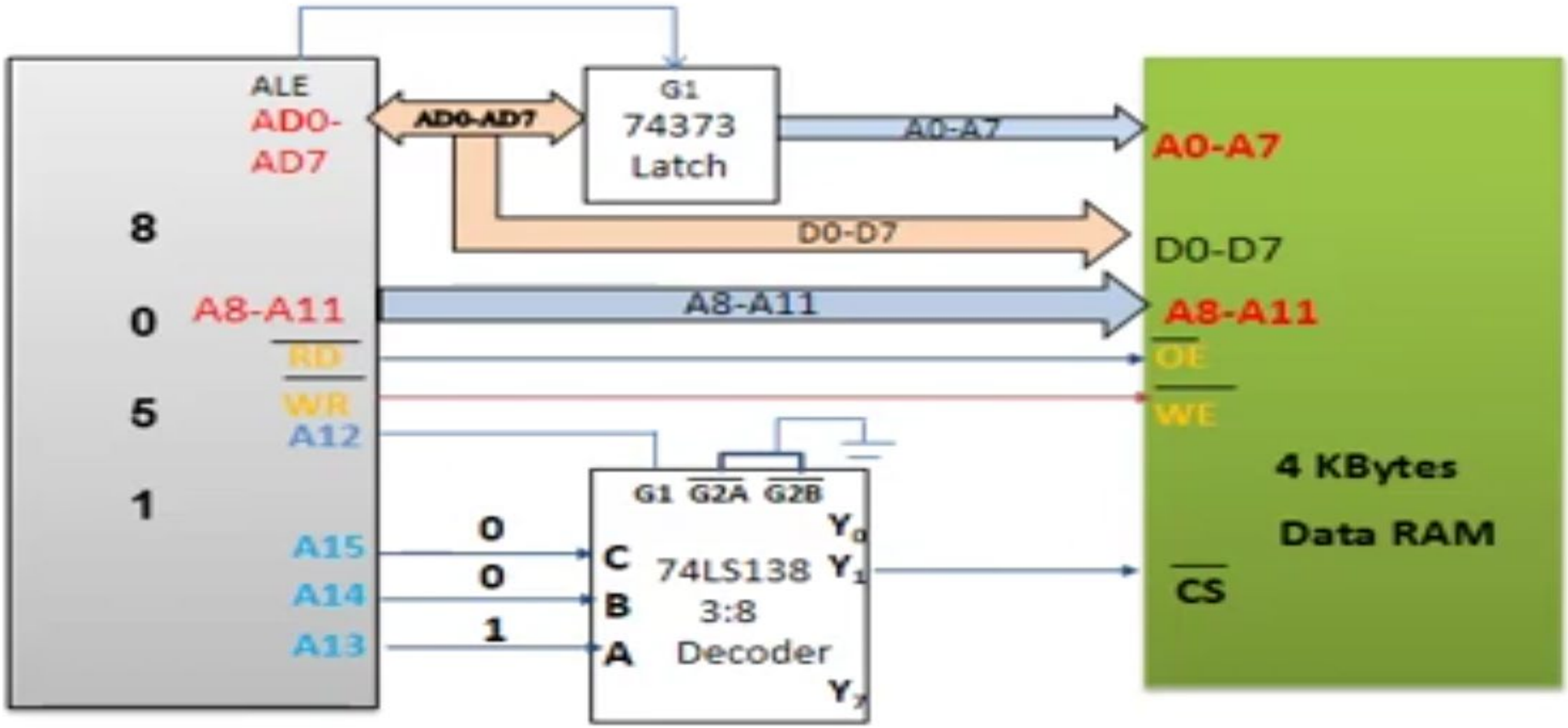
STEPS to interface external RAM with 8051

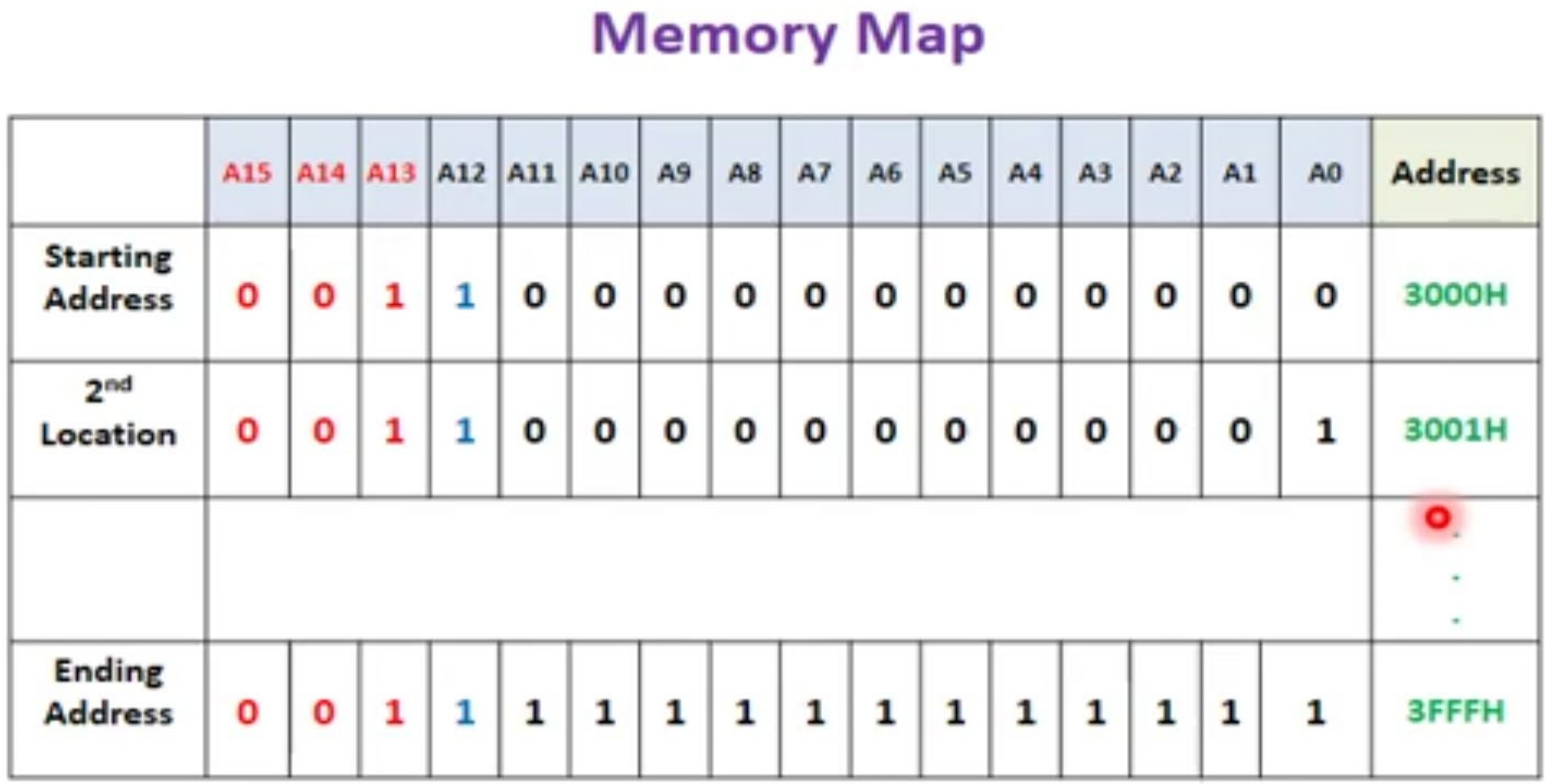
1. **Step 1**: Connect RD to OE of ext. RAM.
2. **Step 2**: Connect WR to WE of ext. RAM.
3. **Step 3**: Connect active low input of NAND gate to CE of external RAM, where the input to NAND gate are address lines A15, A14, and A13. We’ve given 0 1 0 to these lines to access the 8000H location of the external RAM.

Interfacing of External Data Memory

Ex. Interface 4 KBytes of external RAM to 8051 microcontroller.

Sol :4 KBytes memory requires 12 address lines i.e. A0 - A11.





Memory Map

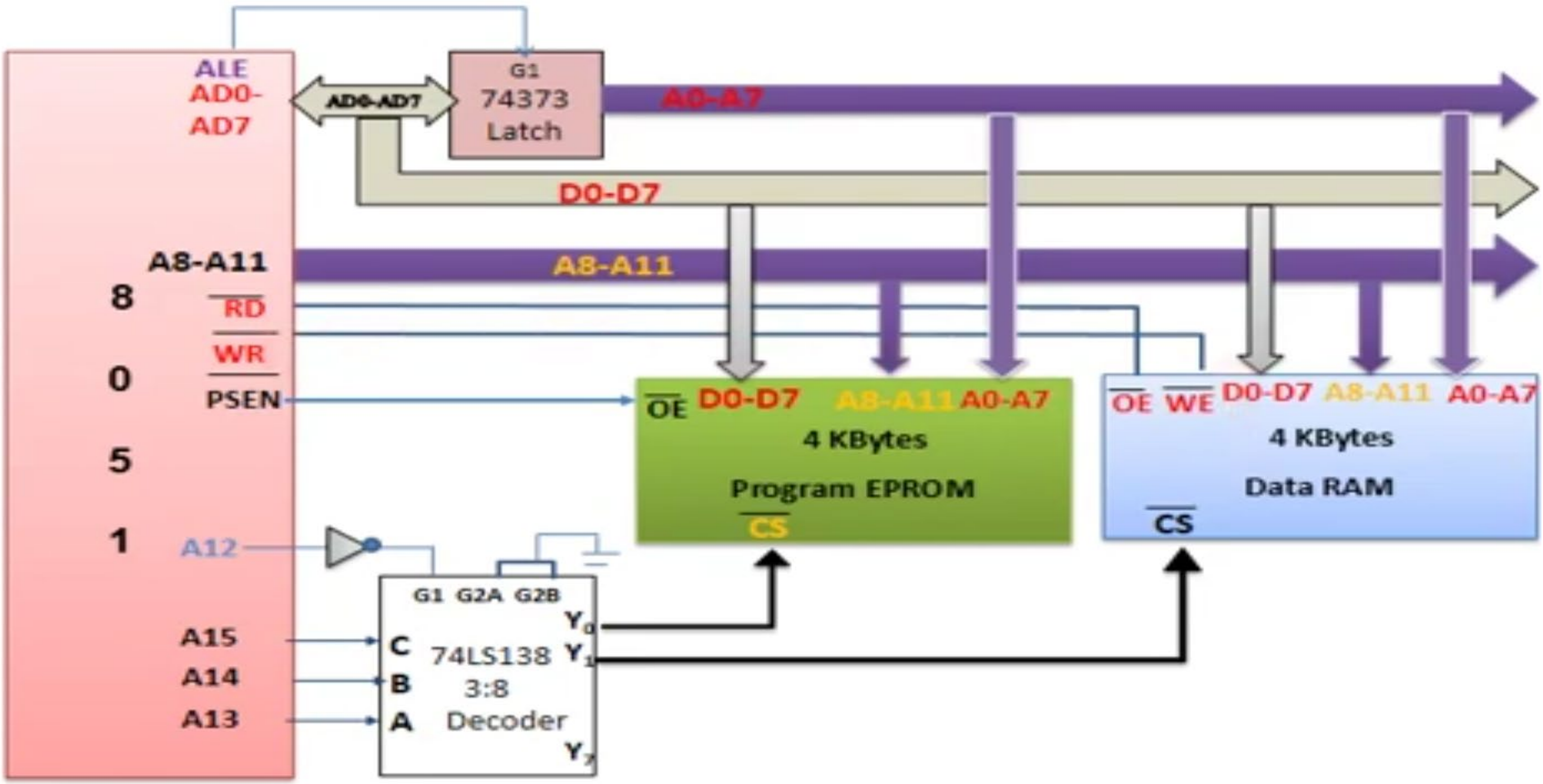
	A15	A14	A13	A12	A11	A10	A9	A8	A7	A6	A5	A4	A3	A2	A1	A0	Address
Starting Address	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	3000H
2nd Location	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	1	3001H
																	. . .
Ending Address	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	3FFFH

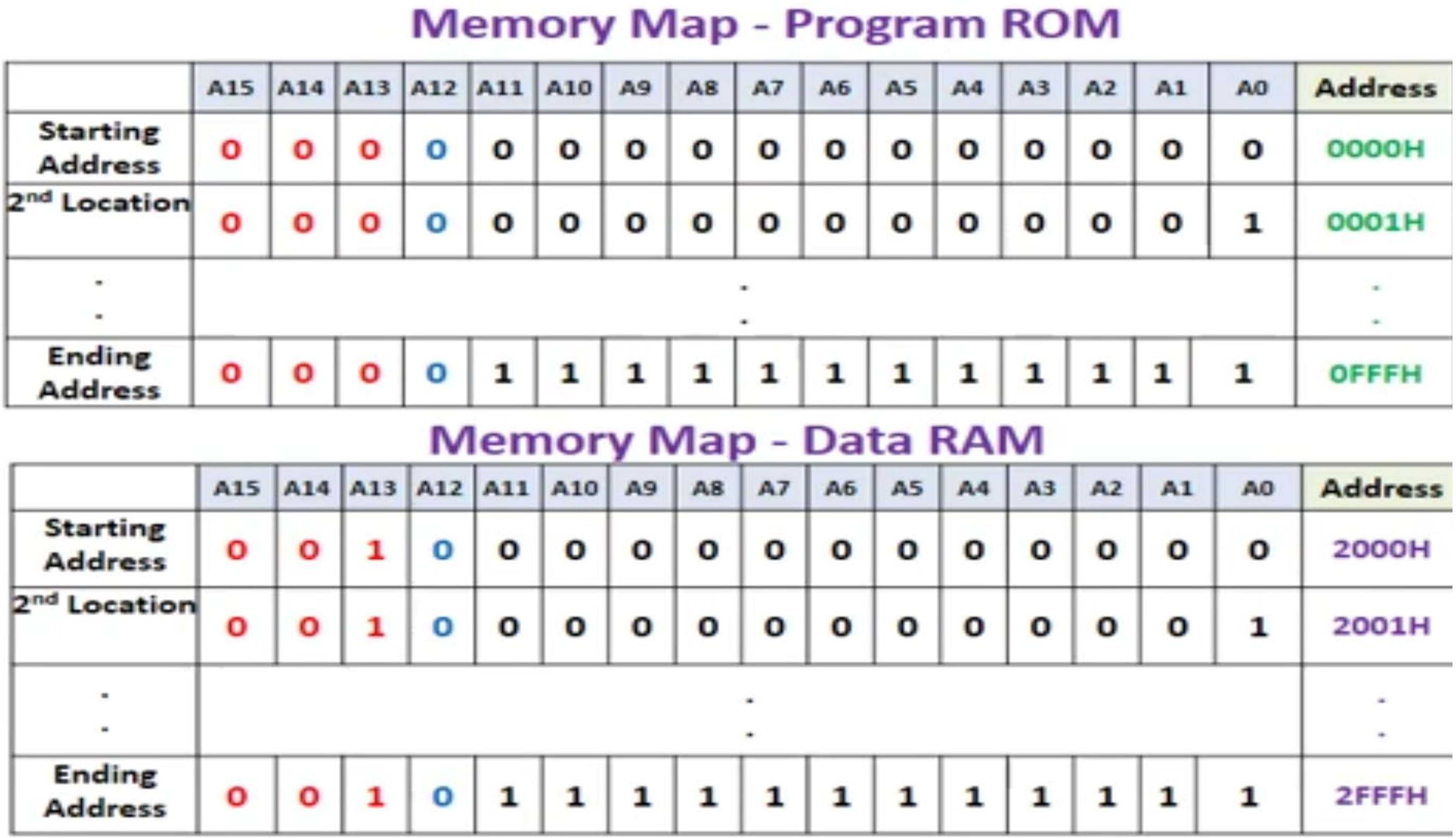
From the above table, we can see that the address range of Y1 is from 3000H to 3FFFH.

- **Interfacing external program ROM, data ROM and external RAM with the 8051**
- Next, let’s interface both program ROM and data RAM to 8051, Let’s say we want to interface 16KB data RAM, 16KB program ROM, and 16KB of data RAM, then we’ll have to follow the following steps:

1. **Step 1**: Calculate the number of address lines required to access 16KB of data, that is 2^{14} = 16KB. Here, we require 14 address lines A0 – A13.
2. **Step 2**: Decide the location of RAM and ROM, here we are going to interface program ROM from 0000H and data RAM from 8000H.
3. **Step 3**: Select the decoder circuit, here we’re going to select 74LS138 decoder.
4. **Step 4**: We do not need a decoder circuit for program ROM, but we have to connect the 74LS138 decoder to data ROM and data RAM.
5. **Step 5**: Connect G1 to VCC, G2A, and G2B to ground.
6. **Step 6**: Connect input A and B to P2.6 and P2.7 respectively, and the input C to ground.
7. **Step 7**: We connect external program and data ROM, for that we can use an AND gate with its input being signal from RD (to access external data space) and PSEN (to access external program space) and output to OE of external ROM.
8. **Step 8**: To interface the external RAM, we connect both RD and WR to WE and OE respectively of external RAM

Interface 4K byte EPROM and 4K byte RAM to 8051





Memory Map - Program ROM

	A15	A14	A13	A12	A11	A10	A9	A8	A7	A6	A5	A4	A3	A2	A1	A0	Address
Starting Address	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0000H
2nd Location	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0001H
. .	. .																. .
Ending Address	0	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	0FFFH

Memory Map - Data RAM

	A15	A14	A13	A12	A11	A10	A9	A8	A7	A6	A5	A4	A3	A2	A1	A0	Address
Starting Address	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	2000H
2nd Location	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	1	2001H
. .	. .																. .
Ending Address	0	0	1	0	1	1	1	1	1	1	1	1	1	1	1	1	2FFFH

ADC Interfacing with 8051

- The data we process in a microcontroller normally deals with digital signals. But there may a situation where we have to deal with external inputs such as analog signals. All most all the input signals from physical sensors are of analog signals. In such cases, we can interface the microcontroller with an external device such as an ADC0808 to convert the analog signal to a digital signal. Because our microcontrollers can only understand 0 and 1. In this article, we look into the details of ADC interfacing with 8051.

- In the present time, there are lots of microcontrollers in the market which has inbuilt ADC with one or more channels, E.g.: PIC18F4550, LPC1768, etc. And by using their ADC registers we can interface. Unfortunately, 8051 doesn’t have an internal module so we will go for an **external ADC**. which is **ADC0808**.

- ADC0808 :

- ADC0808 is a commonly used External 8 bit ADC and it has 28 pins. It can measure up to eight ADC values from 0 to 5 volt since it has eight channels. when voltage reference is +5V, its Step size will be 19.53mV. That is, for every increase of 19.53mV on the input side there will be an increase of 1 bit at the output side.

- **ADC0808** needs an external clock to operate. The ADC needs some specific control signals for its operations like start conversion and bring data to output pins. When the conversion is complete the EOC pins go low to indicate the end of a conversion and that the data is ready to be picked up.

- Features

- Easy interface to all microprocessors

- Operates ratio metrically or with 5 V DC or analog span adjusted voltage reference

- No zero or full-scale adjust required

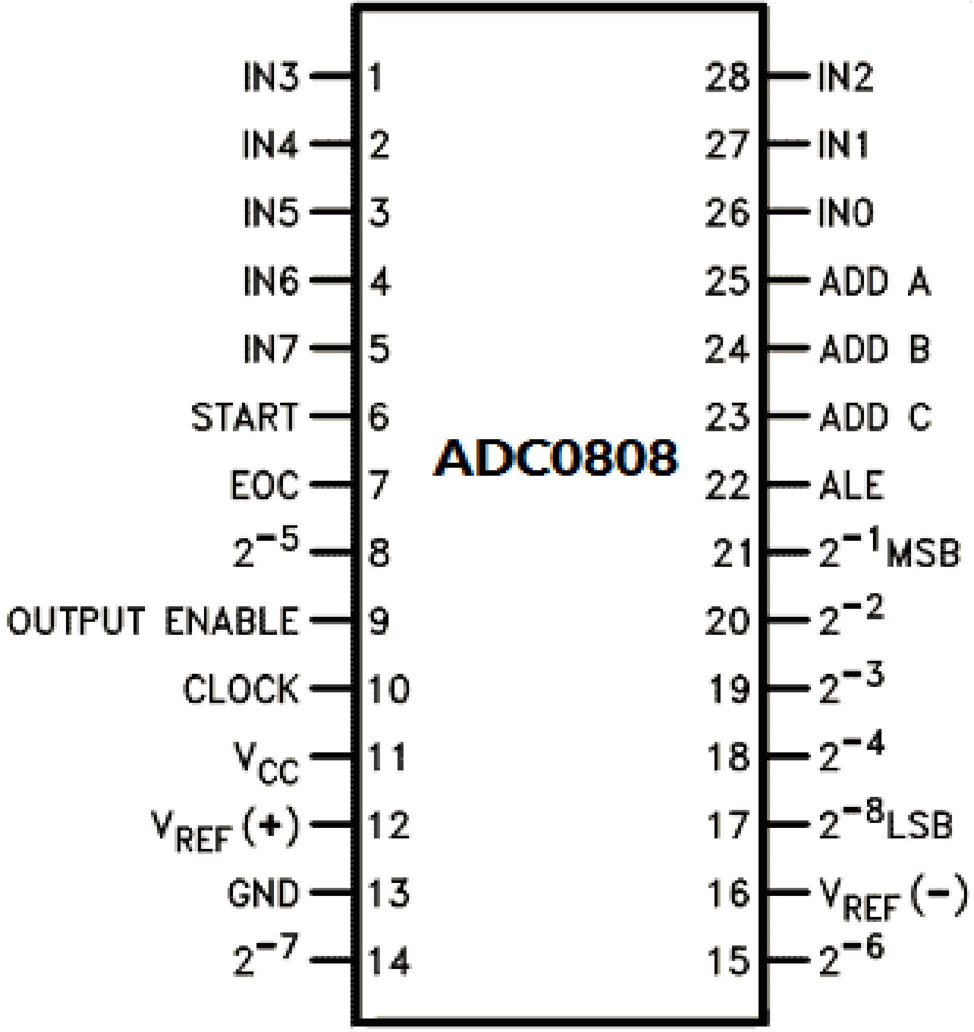
- 8-channel multiplexer with address logic

- 0V to 5V input range with single 5V power supply

- Outputs meet TTL voltage level specifications

28-pin molded chip carrier package

Pin diagram   ADC0808



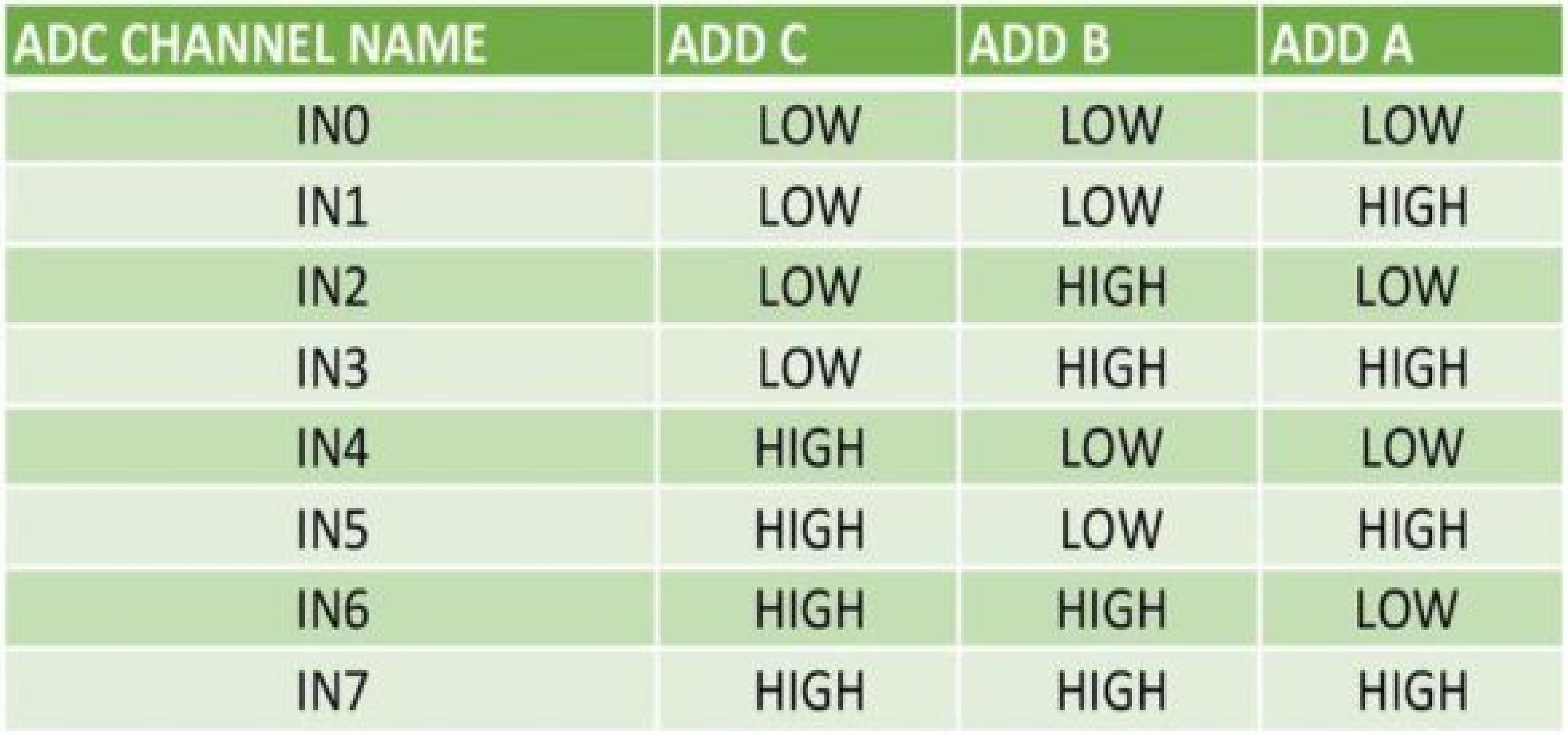
Pin Description

Pin No	Function	Name
1	Analog Input Pin 3	IN3
2	Analog Input Pin 4	IN4
3	Analog Input Pin 5	IN5
4	Analog Input Pin 6	IN6
5	Analog Input Pin 7	IN7
6	Start conversion; input pin; a low to high pulse is given	START
7	End of conversion; output pin; goes low when the conversion is over	EOC
8	Digital output bit	D3
9	Input pin; a low to high pulse brings data to output pins from the internal registers at end of conversion	Output Enable
10	Clock input; to provide external clock	Clock Input
11	Supply voltage; 5V	Vcc
12	Positive reference voltage	Vref+
13	Ground (0v)	GND
14	Digital output bit	D1

17	Digital output bit	D0
18	Digital output bit	D4
19	Digital output bit	D5
20	Digital output bit	D6
21	Digital output bit	D7
22	Address latch enable; Input pin; low to high pulse is required to latch in the address	ALE
23	Address line C	Address C

Channel Selection :

We can select any input channel by using the Address lines ADD A, ADD B and ADD C. As you can see in the below table, We can select the input line IN0 by keeping all three address lines ADD A, ADD B and ADD C Low.



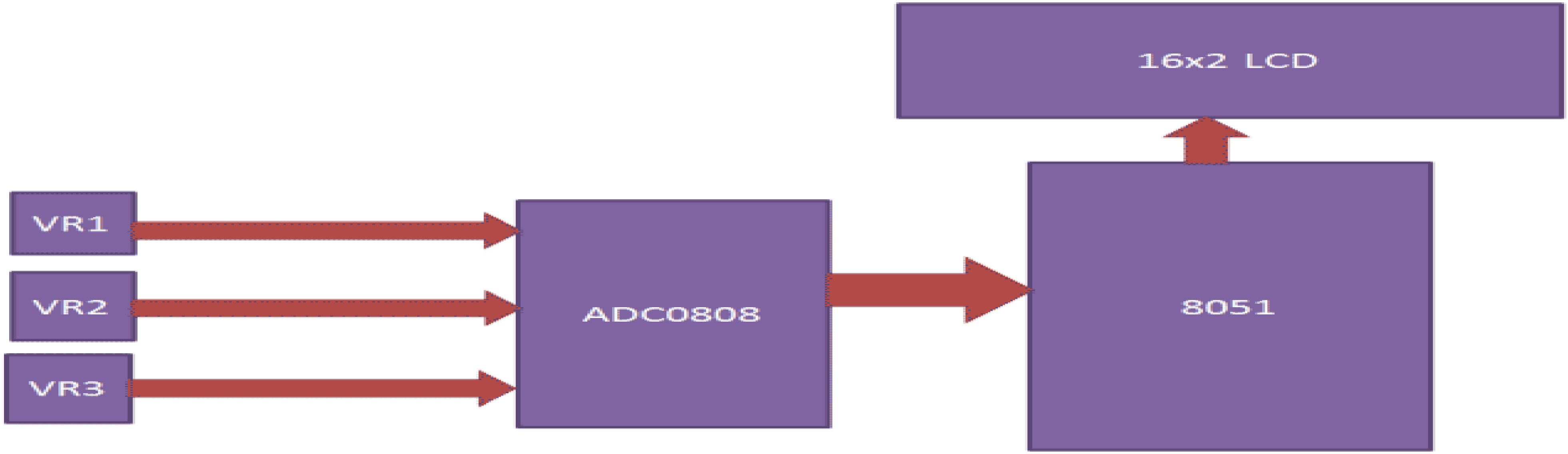
ADC CHANNEL NAME	ADD C	ADD B	ADD A
IN0	LOW	LOW	LOW
IN1	LOW	LOW	HIGH
IN2	LOW	HIGH	LOW
IN3	LOW	HIGH	HIGH
IN4	HIGH	LOW	LOW
IN5	HIGH	LOW	HIGH
IN6	HIGH	HIGH	LOW
IN7	HIGH	HIGH	HIGH

Steps to be followed to interface ADC (ADC0808) with 8051

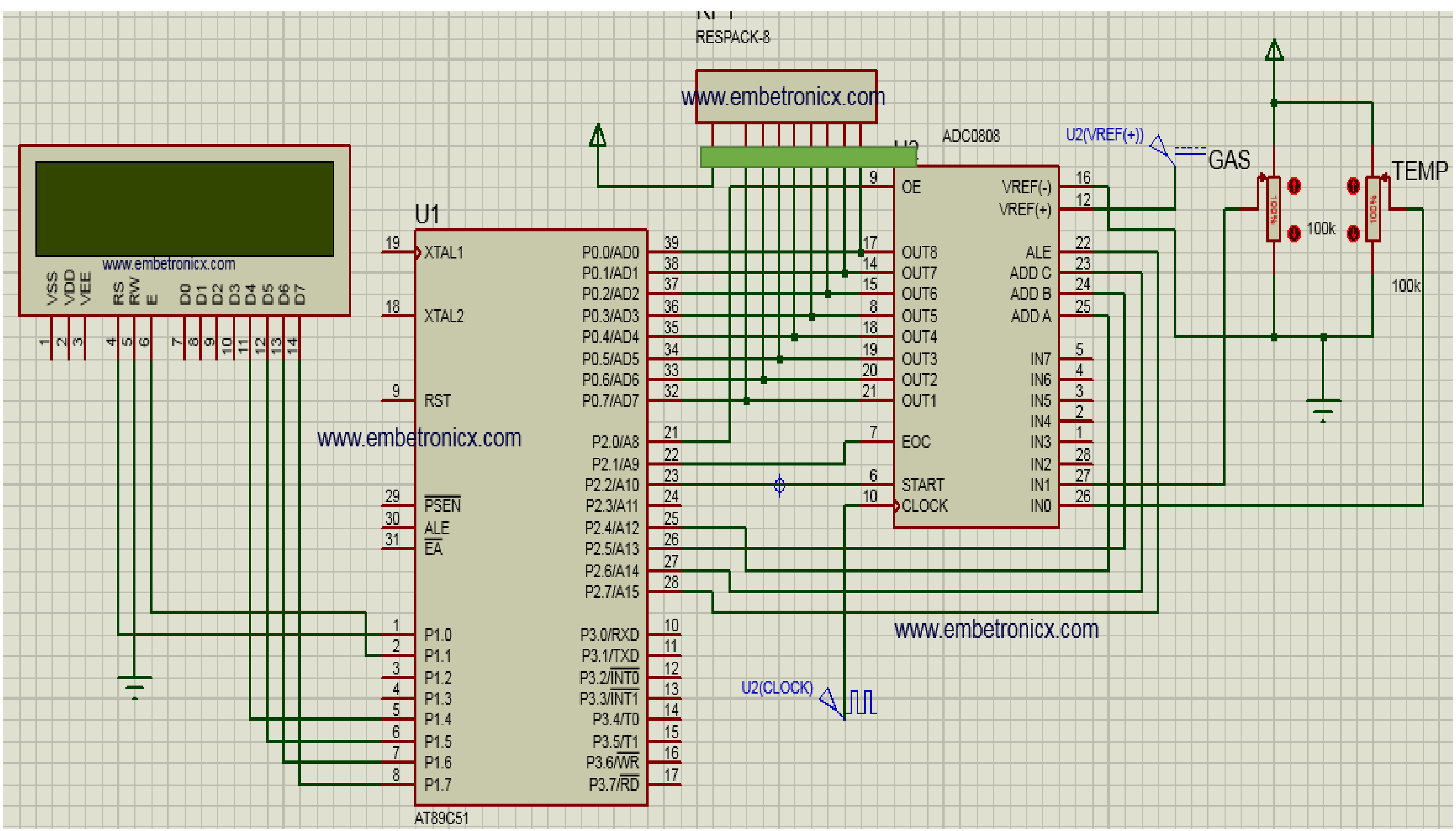
1. Start
2. Select the channel using Address pins.
3. A Low – High transition on ALE to latch in the address.
4. A Low – High transition on Start to reset the ADC’s SAR.
5. A High – Low transition on ALE.
6. A High – Low transition on start to start the conversion.
7. Wait for End of cycle (EOC) pin to become high.
8. Make Output Enable pin High.
9. Take Data from the ADC’s output
10. Make Output Enable pin Low.
11. Stop

Working:

In this project we have interfaced three channels of ADC0808. And for demonstration we have used three variable resistors. When we power the circuit then microcontroller initialize the LCD by using appropriate command, gives clock to ADC chip, selects ADC channel by using address line and send start conversion signal to ADC. After this ADC first reads selected ADC channel input and gives its converted output to microcontroller. Then microcontroller shows its value at Ch1 position in LCD. And then microcontroller changes ADC channel by using address line. And then ADC reads selected channel and send output to microcontroller. And show on LCD as name Ch2. And like wise for other channels.



- **Working of ADC0808** is much similar to working of ADC0804. In this, first microcontroller provides a 500 KHz clock signal to ADC0808, using the Timer 0 interrupt, as ADC requires clock signal to operate. Now microcontroller sends a LOW to HIGH level signal to ALE pin (its active-high pin) of ADC0808 to enable the latch in the address. Then by applying HIGH to LOW Level signal to SC (Start Conversion), ADC starts analog to digital conversion. And then wait for the EOC (End of Conversion) pin to go LOW. When EOC goes LOW, it means analog to digital conversion has been completed and data is ready to use. After this, microcontroller enables the output line by applying a HIGH to LOW signal to OE pin of ADC0808.
- ADC0808 gives ratio metric conversion output at its output pins. And the formula for radiometric conversion is given by:
- $V_{in}/(V_{fs}-V_z)= D_x/(D_{max}-D_{min})$
- WHERE
- V_{in} is input voltage for conversion
- V_{fs} is full scale Voltage
- V_z is zero voltage
- D_x is data point being measure
- D_{max} is Maximum data limit
- D_{min} is Minimum data limit



RP1
RESPACK-8
www.embetronicx.com
U2
ADC0808
U2(VREF(+))
GAS
TEMP
100k
100k
VSS
VDD
VEE
RS
RW
E
D0
D1
D2
D3
D4
D5
D6
D7
1
2
3
4
5
6
7
8
9
10
11
12
13
14
U1
19
XTAL1
18
XTAL2
9
RST
29
PSEN
30
ALE
31
EA
1
P1.0
2
P1.1
3
P1.2
4
P1.3
5
P1.4
6
P1.5
7
P1.6
8
P1.7
AT89C51
P0.0/AD0
39
P0.1/AD1
38
P0.2/AD2
37
P0.3/AD3
36
P0.4/AD4
35
P0.5/AD5
34
P0.6/AD6
33
P0.7/AD7
32
P2.0/A8
21
P2.1/A9
22
P2.2/A10
23
P2.3/A11
24
P2.4/A12
25
P2.5/A13
26
P2.6/A14
27
P2.7/A15
28
P3.0/RXD
10
P3.1/TXD
11
P3.2/INT0
12
P3.3/INT1
13
P3.4/T0
14
P3.5/T1
15
P3.6/WR
16
P3.7/RD
17
9
OE
17
OUT8
14
OUT7
15
OUT6
8
OUT5
18
OUT4
19
OUT3
20
OUT2
21
OUT1
7
EOC
6
START
10
CLOCK
VREF(-)
16
VREF(+)
12
ALE
22
ADD C
23
ADD B
24
ADD A
25
IN7
5
IN6
4
IN5
3
IN4
2
IN3
1
IN2
28
IN1
27
IN0
26
U2(CLOCK)
www.embetronicx.com
www.embetronicx.com
www.embetronicx.com

8051 Interfacing DAC

- Microcontroller are used in wide variety of applications like for measuring and control of physical quantity like temperature, pressure, speed, distance, etc.
- In these systems microcontroller generates output which is in digital form but the controlling system requires analog signal as they don't accept digital data thus making it necessary to use DAC which converts digital data into equivalent analog voltage
- Digital to Analog Converter is a device used to convert digital pulses to analog signals.
- In the figure shown, we use 8-bit DAC 0808. This IC converts digital data into equivalent analog Current. Hence we require an I to V converter to convert this current into equivalent voltage.
- DAC0808 provides 256 discrete voltage (or current) levels of output.

DAC0808

- In the MC1408 (DAC0808), the digital inputs are converted to current (I_{out}), and by connecting a resistor to the I_{out} pin, we convert the result to voltage.
- The total current provided by the I_{out} pin is a function of the binary numbers at the D0 – D7 inputs of the DAC0808 and the reference current (I_{re}f), and is as follows:

- where D0 is the LSB, D7 is the MSB for the inputs, and I_{re}f is the input current that must be applied to pin 14. The I_{re}f current is generally set to 2.0 mA.

$$V0=Vref\left[\frac{D0}{2}+\frac{D1}{4}+\frac{D2}{8}+\frac{D3}{16}+\frac{D4}{32}+\frac{D5}{64}+\frac{D6}{128}+\frac{D7}{256}\right]$$

Ex:
1. IF data =00H [00000000], Vref= 10V

$$V0= 10\left[\frac{0}{2}+\frac{0}{4}+\frac{0}{8}+\frac{0}{16}+\frac{0}{32}+\frac{0}{64}+\frac{0}{128}+\frac{0}{256}\right]$$

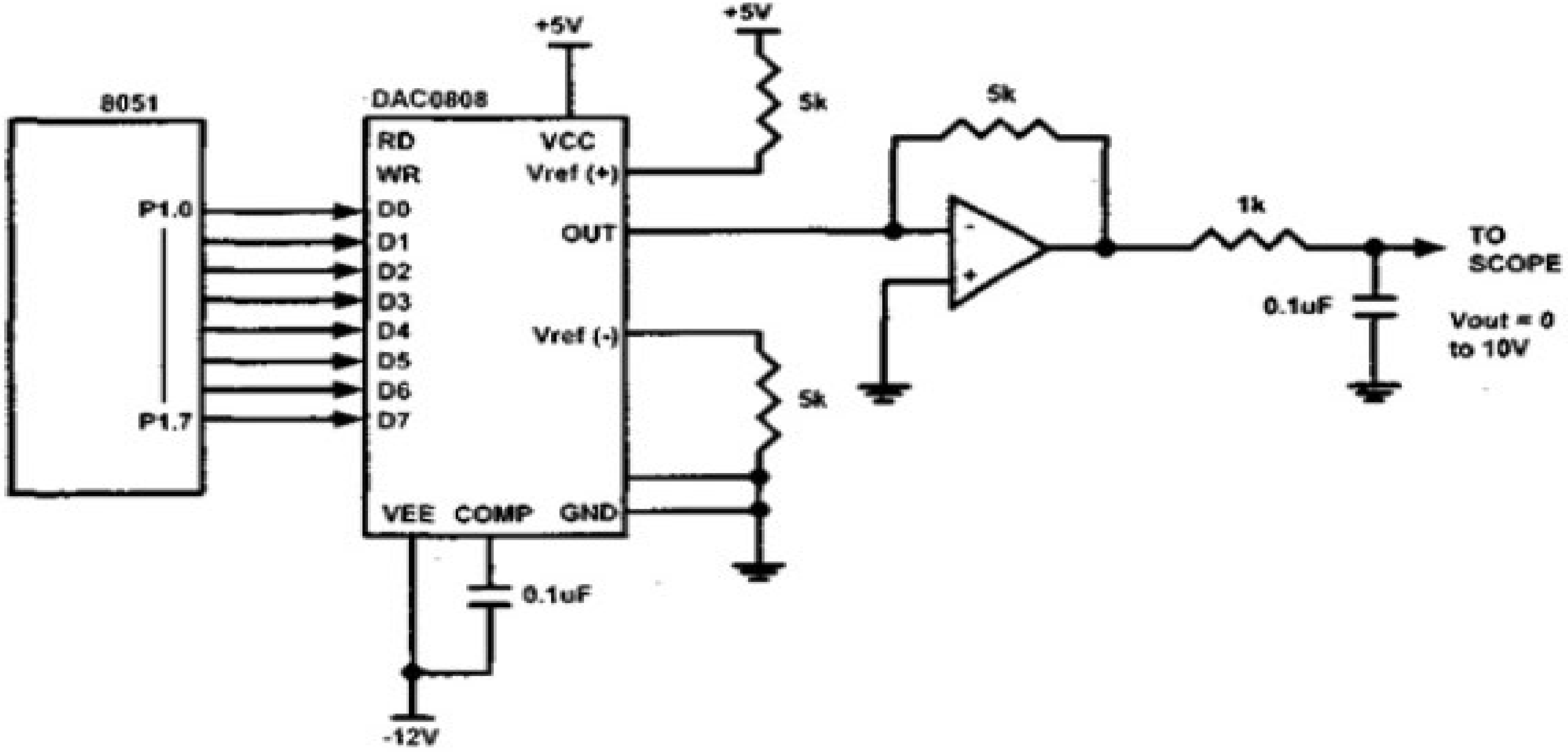
Therefore, V0= 0 Volts.

2. If data is 80H [10000000], Vref= 10V

$$V0=10\left[\frac{1}{2}+\frac{0}{4}+\frac{0}{8}+\frac{0}{16}+\frac{0}{32}+\frac{0}{64}+\frac{0}{128}+\frac{0}{256}\right]$$

Therefore, V0= 5

Interfacing Diagram



SERIAL COMMUNICATION IN 8051

serial communication of data in one of the most widely used microcontroller (8051). Lot of applications require microcontrollers to either accept the data in serial form or o/p the data in serial form.

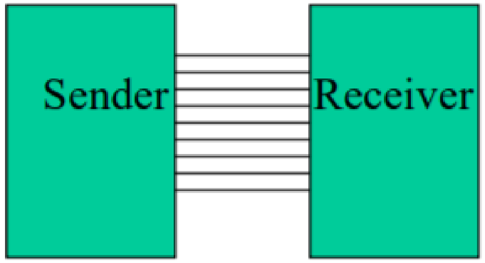
Microcontrollers can communicate data in either parallel form or serial form. In parallel communication, data is transferred over more than one wire for example if 8 wires of one microcontroller are connected to any other peripheral device or another microcontroller then at a particular time 8 data bits are transferred. On the other hand in serial communication, data is transferred in bit by bit manner over a single wire. For example if a microcontroller 1 is transferring data to another microcontroller 2 in serial form then TXD pin of microcontroller 1 will be connected to RXD pin of microcontroller 2 and data is transferred from microcontroller 1 to microcontroller 2 in bit by bit manner over a single wire. If microcontroller 2 wants to send some data to microcontroller 1 then for that TXD pin of microcontroller 2 must be connected to RXD pin of microcontroller 1 and again data bits will be transferred over a single wire. So it is clear that for full duplex communication of data two wires are involved and for simplex communication only one wire will be involved.

Parallel communication requires more number of wires than serial communication but it has higher speed of data transfer as more than one bit is transferred at a given time. Serial communication is preferred when the distance b/w transmitter and receiver is large and it is required to save the cabling cost and reduce h/w complexity but definitely this comes at the cost of reduced speed of data transfer.

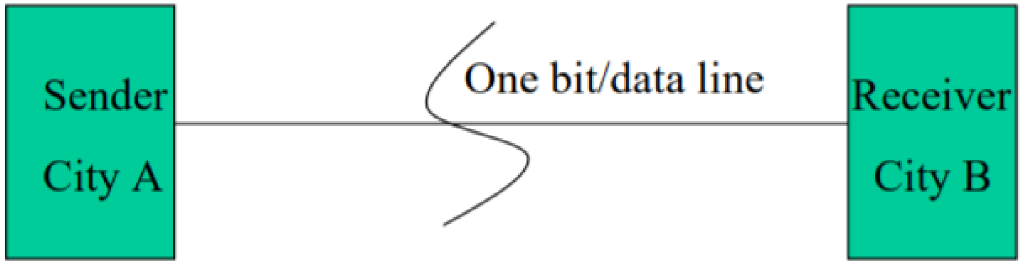
Parallel vs. Serial

- Parallel Communication (Printer)
  - Fast, but distance cannot be great.
  - Expensive

One byte at a time or more



- Serial Communication (Telephone line)
  - Long distance
  - cheaper



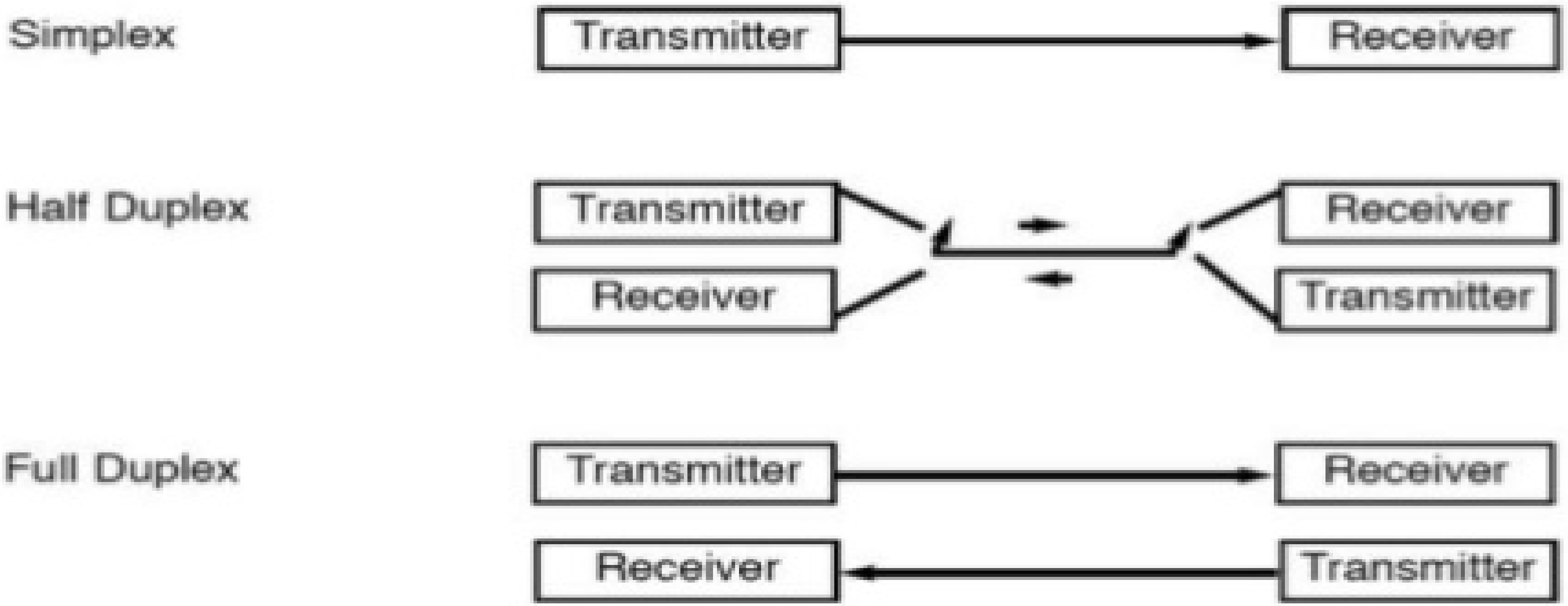
Serial v/s Parallel Communication

Parallel Communication	Serial Communication
Often 8 or more lines (wire conductors) are used to transfer data. Multiple bits are transferred at a time.	The data is sent one bit at a time on a single line (wire)
Preferred for short-distance communication	Preferred over long-distance communication
Costly as more resources are required	Comparatively cheaper
Speed of data transfer is high	Slow
Example: SPI, I2C, UART	Example: PCI

Basics of Serial Communication

- Serial communication uses single data line making it much cheaper
- Enables two computers in different cities to communicate over the telephone
- Byte of data must be converted to serial bits using a parallel-in-serial-out shift register and transmitted over a single data line
- At the receiving end there must be a serial-in-parallel-out shift register
- If transferred on the telephone line, it must be converted to audio tones by modem for short distance

Modes of Serial Communication



- Serial communication can be classified on the basis how transmission occurs.
- 1. Simplex: In simplex, the hardware such that data transfer takes place in only one direction. Ex: Computer to Printer communication
- 2. Half Duplex: The half duplex transmission allows the data transfer in both direction but not simultaneously. Ex: Walkie talkie
- 3. Full Duplex: It allows the data transfer in both direction simultaneously. Ex: Telephone lines Data transfer schemes
- The data in the serial communication may be sent in two formats:

Basics of Serial Communication

- Serial Communication can be
  - ✓ Asynchronous
  - ✓ Synchronous

Synchronous Communication

- Synchronous methods transfer a block of data (characters) at a time
- The events are referenced to a clock
- Example: SPI bus, I2C bus

Asynchronous Communication

- Asynchronous methods transfer a single byte at a time
- There is no clock. The bytes are separated by start and stop bits.
- Example: UART

Basics of Serial Communication

- To support serial communication, special interfaces are built in the microcontroller.
- The microcontrollers use special IC chips called UART (universal asynchronous receiver-transmitter) and USART (universal synchronous asynchronous receiver-transmitter)
- 8051 chip has a built-in UART

Data Framing in Asynchronous Serial Communication

- Data is transmitted in 0s and 1s
- To have a sense of synchronization between transmitter and receiver and to make sense of the data, transmitter and receiver agree on a set of rules i.e protocol, which describes
  - ✓ how the data is packed
  - ✓ how many bits constitute a character
  - ✓ when the data begins and ends

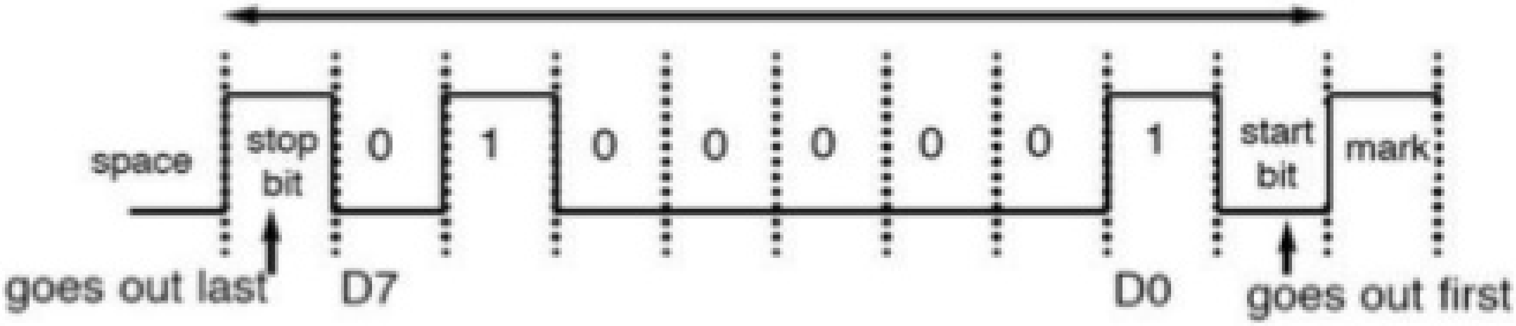
Data Framing in Asynchronous Serial Communication

Start and stop bits

- Each character is placed between start and stop bits. This is called framing.
- Start bit is always one bit, stop bit can be one, two or one and half bits
- In 8051 serial port, when there is no transmission, the TxD line is held high. This is called mark.
- Start bit is always a 0 (low), stop bit(s) is 1 (high)
- LSB is sent out first

Data Framing in Asynchronous Serial Communication

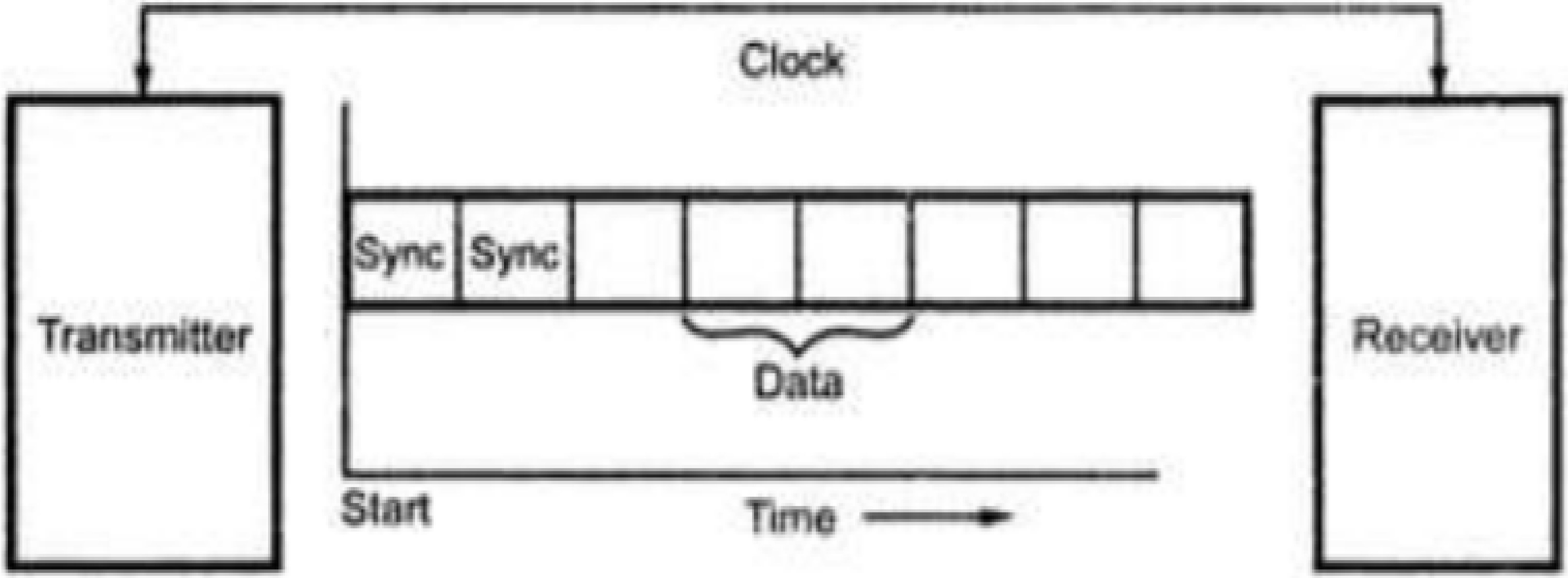
Framing ASCII A



- The transmission begins with a start bit, followed by the LSB(D_0), then the rest of the bits until MSB (D_7), and finally, the one stop bit indicating the end of the character
- When there is no transfer, the signal is 1 (high), which is referred to as *mark*

- Asynchronous: □Asynchronous formats are character oriented.
- In this type the bits or character or data word are sent at constant rate, but characters can come at any rate (asynchronously) as long as they do not overlap.
- When no characters are being sent a line stays high at logic1 called mark, logic0 is called space.
- The beginning of a character is indicated by start bit which is always low.
- This is used to synchronize the transmitter and receiver. □After the start bit the data bits are sent with least significant bit first followed by one or more stop bits (active high).
- The stop bits indicate the end of character.
- The combination of start but, character and stop bits is known as frame. The start and stop bits carry no information, but are required because of asynchronous nature of data.

- **Synchronous:**
- The start and stop bits in each frame of asynchronous format represents wasted overhead bytes that reduce overall character rate. These start and stop bits can be eliminated by synchronizing receiver and transmitter.
- They can be synchronized by having a common clock signal.
- Such a communication is called synchronous serial communication. In this transmission synchronous bits are inserted instead of start and stop bits
- The data rate can be expressed as bit/sec or character/sec.
- The term bit/sec is also called baud rate.



Synchronous transmission format

S.No	Asynchronous	Synchronous
1.	Transmitters and receivers are not synchronized by clock.	Transmitters and receivers are synchronized by clock.
2.	Bits of data are transmitted at constant rate.	Data bits are transmitted with synchronization of clock.
3.	Character may arrive at any rate at receiver.	Character is received at constant rate.
4.	Data transfer is character oriented.	Data transfer takes place in blocks
5.	Start and stop bits are required to establish communication of each character.	Start and stop bits are not required to establish communication of each character. Synchronization bits are required to transfer the data block.
6.	Used in low-speed transmission at about speed less than 20 Kbits/sec.	Used in high speed transmissions.

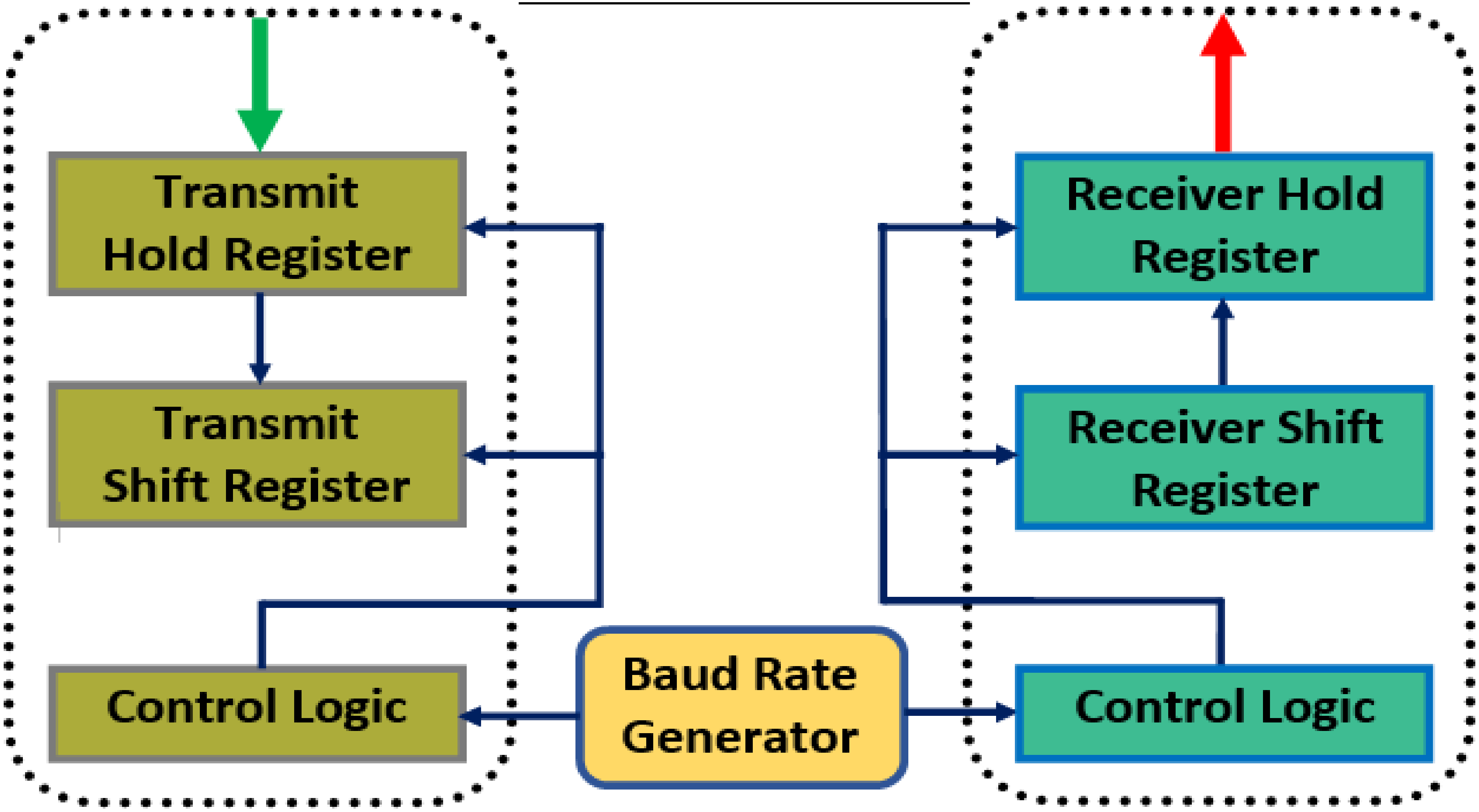
- **Data Transfer**
- In 8051 microcontroller serial communication of data is performed with the help of following special purpose registers-
- SBUF (Serial buffer register)
- SCON (serial control register)
- TMOD (Timer mode register)
- TCON (Timer control register)
- TH1 (Timer1 register higher byte)
- Before going further it is extremely important to first understand these registers and their importance with reference to serial communication.

UART(universal asynchronous transmitter and receiver)

- UART stands for a universal asynchronous transmitter and receiver.
- UART Protocols is a serial communication with two wired protocols.
- The data cable signal lines are labeled as Rx and Tx. Serial communication is commonly used for transmitting and receiving the signal.
- TX and RX are connected between two devices. (eg. USB and computer)

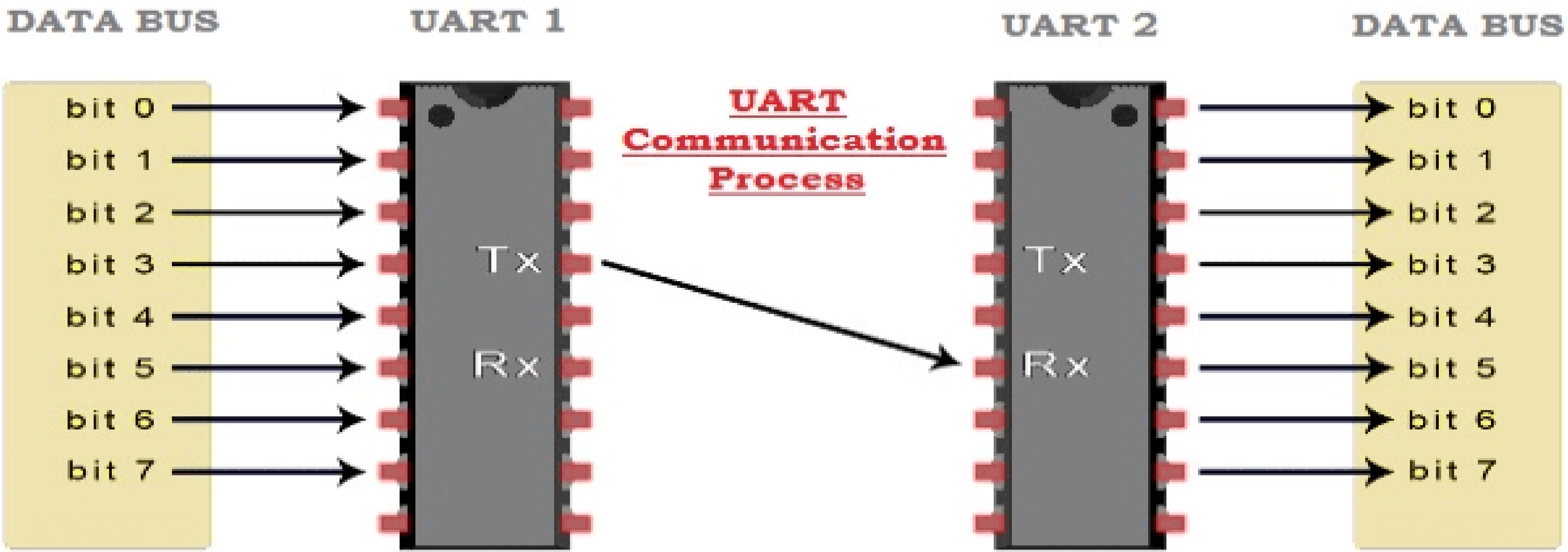
  It is transferred and receives the data serially bit by bit without clock pulses.
- The UART takes bytes of data and sends the individual bits in a sequential manner.
- UART is a half-duplex protocol. Half-duplex means transferring and receiving the data but not at the same time. Most of the controllers have hardware UART on board.
- It uses a single data line for transmitting and receiving the data.
- It has one start bit, 8-bit data and one-stop bit mean the 8-bit data transfer one’s signal is high to low.
- UART commonly built in microcontrollers
- Ex: Emails, SMS, Walkie-talkie.

UART BLOCK DIAGRAM



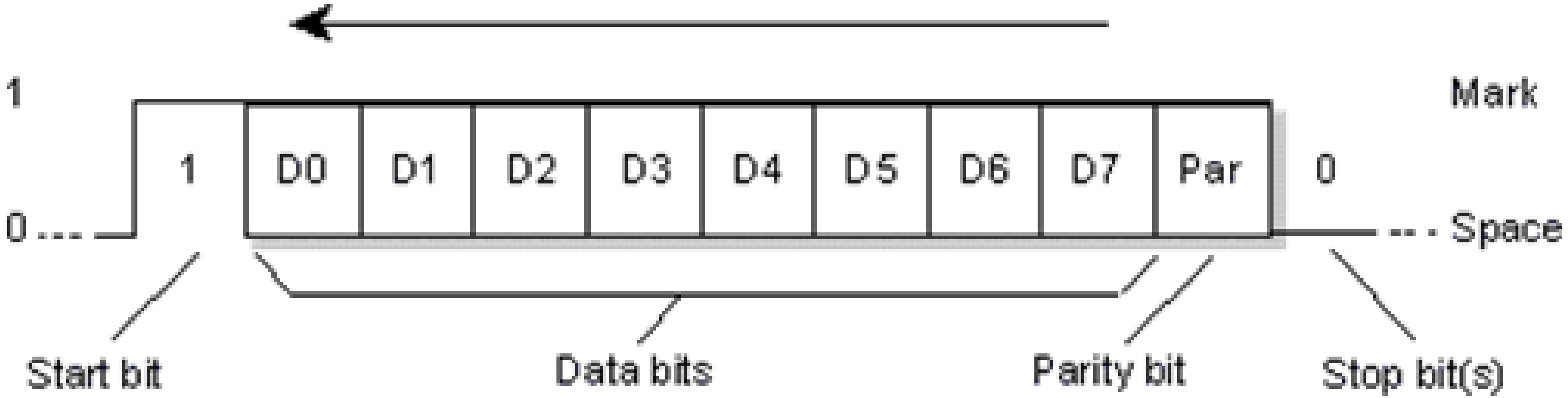
- The Universal Asynchronous Receiver Transmitter (UART) block diagram has two main components.
- They are the receiver and transmitter. These two components are coupled with a baud rate generator.
- This is used mainly for speed generation when the receiver and transmitter section has to receive or transmit data.
- The receiver section consists of shift register, control logic and a receive hold register.
- Likewise, transmitter section also has a shift register, control logic and a transmit hold register.
- The transmitter hold register contains the data to be transmitted. The shift registers in the two components move the data bits left or right till the data transmission or receive operation is completed.
- A write or read logic is used to indicate when the read and write operation must be done.
- The baud rate generator is used to generate speeds ranging from 110 bps to 230400 bps.
- The micro-controllers typically use a baud rate of 9600 bps to 115200 bps.

UART Communication Process



- Universal Asynchronous Receiver Transmitter communication takes place through two mediums i.e. transmitting UART and receiving UART.
- The data flow is from both receiving (Rx) and transmitting (Tx) pins of Universal Asynchronous Receiver Transmitter and only two cables are required for this purpose.
- Universal Asynchronous Receiver Transmitter communication happens asynchronously i.e. clock or other timing signals are absent.
- Instead of that, UART has special start and stop bits that are added to the beginning and end of the data packet respectively. These bits assist the receiving UART in identifying the actual received data.
- The above figure shows a typical Universal Asynchronous Receiver Transmitter (UART) communication process.
- The controlling device transfers data to the transmitting UART through a data bus. This controlling device can be a CPU of a micro-controller or microprocessor, memory units like ROM or RAM. The transmitting UART receives data through parallel mode of communication.
- The data is converted into a data packet by adding the start, stop and parity bits by the Universal Asynchronous Receiver Transmitter (UART). It is then converted from parallel to serial form using a shift register and is transmitted bit wise from the Tx pin.
- This serial data is received by the Rx pin and identifies the actual data through the start and stop bits. Data integrity is verified using the parity bit. The data is again converted into parallel mode using shift register and is dispatched to the controller at the receiving end.

UART Protocol Data Flow



Advantages:

- Only two Wires
- No Clock Signal necessary
- Has Parity bit for error checking
- Structure of data packet can be changed
- Widely used

Disadvantages:

- Size of data frame is limited to a maximum of 9 bits
- Doesnot Support multiple slave or multiple master systems
- Buadrate of each UART must be within 10% of each other

Disadvantages:

- Size of data frame is limited to a maximum of 9 bits
- Doesnot Support multiple slave or multiple master systems
- Buadrate of each UART must be within 10% of each other

Applications:

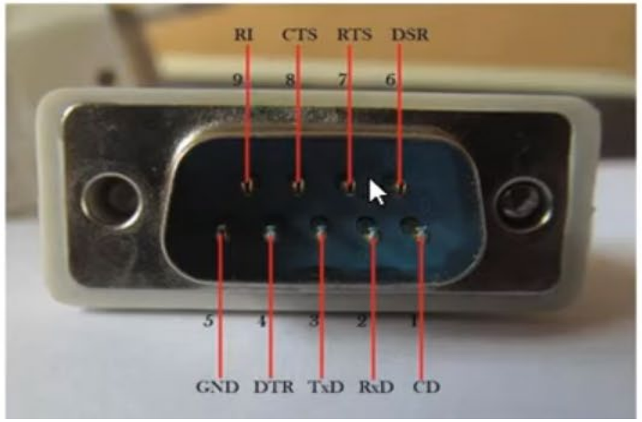
- Commonly included in microcontrollers, used in devices including GPS units,modems,wireless communication and Bluetooth modules and many more..

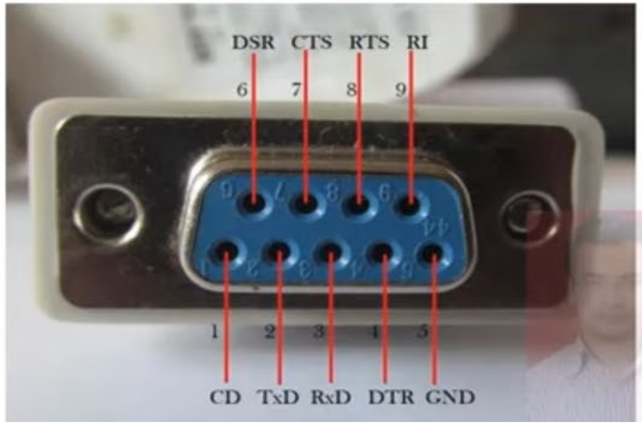
External communication interfaces RS 232

- **RS232 Protocol – Basics**
- RS232 is one of the most widely used techniques to interface external equipment with computers.
- RS232 is a Serial Communication **Recommended Standard Number** developed by the Electronic Industry Association (EIA) and Telecommunications Industry Association (TIA).
- RS232 defines the signals connecting between DTE and DCE. Here, DTE stands for Data Terminal Equipment and an example for DTE is a computer.
- DCE stands for Data Communication Equipment or Data Circuit Terminating Equipment and an example for DCE is a modem.
- RS232 was introduced in 1960's and was originally known as EIA Recommended Standard 232. RS232 is one of the oldest serial communication standards with ensured simple connectivity and compatibility across different manufacturers. Originally, the DTEs in RS32 are electromechanical typewriters and DCEs are modems.
- RS232 uses serial communication, where one bit of data is sent at a time along a single data line. This is contrast to parallel communication, where multiple bits of data are sent at a time using multiple data lines.

RS-232

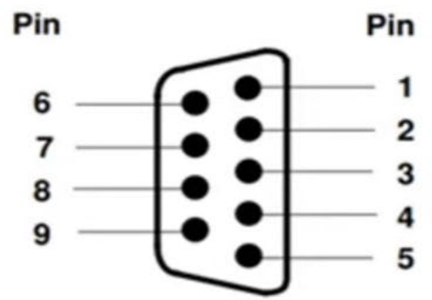
- An RS-232 link was designed to connect the terminal to a modem, which in turn accesses the phone lines that connect to the remote computer.
- But, now a days, an RS-232 port is more likely to connect a PC to an embedded system or to connect two embedded systems.
- The standard designates 25 lines in the interface, but RS-232 ports rarely sup- port more than the nine signals.
- RS232 uses 9 pin (DB9) or 25 pin(DB25) both “D” shaped connectors available in a male or female form both.



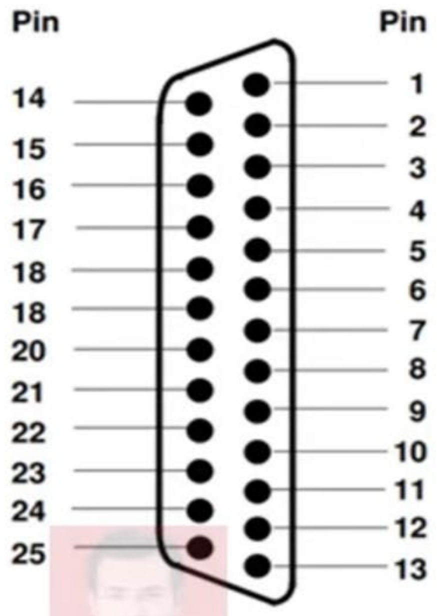


DB9 and DB25 Connectors

Pin Number (9-pin D-sub)	Pin Number (25-pin D-sub)	Signal	Source	Type	Description
1	8	CD	DCE	control	Carrier detect
2	3	RX	DCE	data	Receive data
3	2	TX	DTE	data	Transmit data
4	20	DTR	DTE	control	Data terminal ready
5	7	SG	–	–	Signal ground
6	6	DSR	DCE	control	Data set ready
7	4	RTS	DTE	control	Request to send
8	5	CTS	DCE	control	Clear to send
9	22	RI	DCE	control	Ring Indicator
–	1, 9-19, 21, 23-25	unused	–	–	–

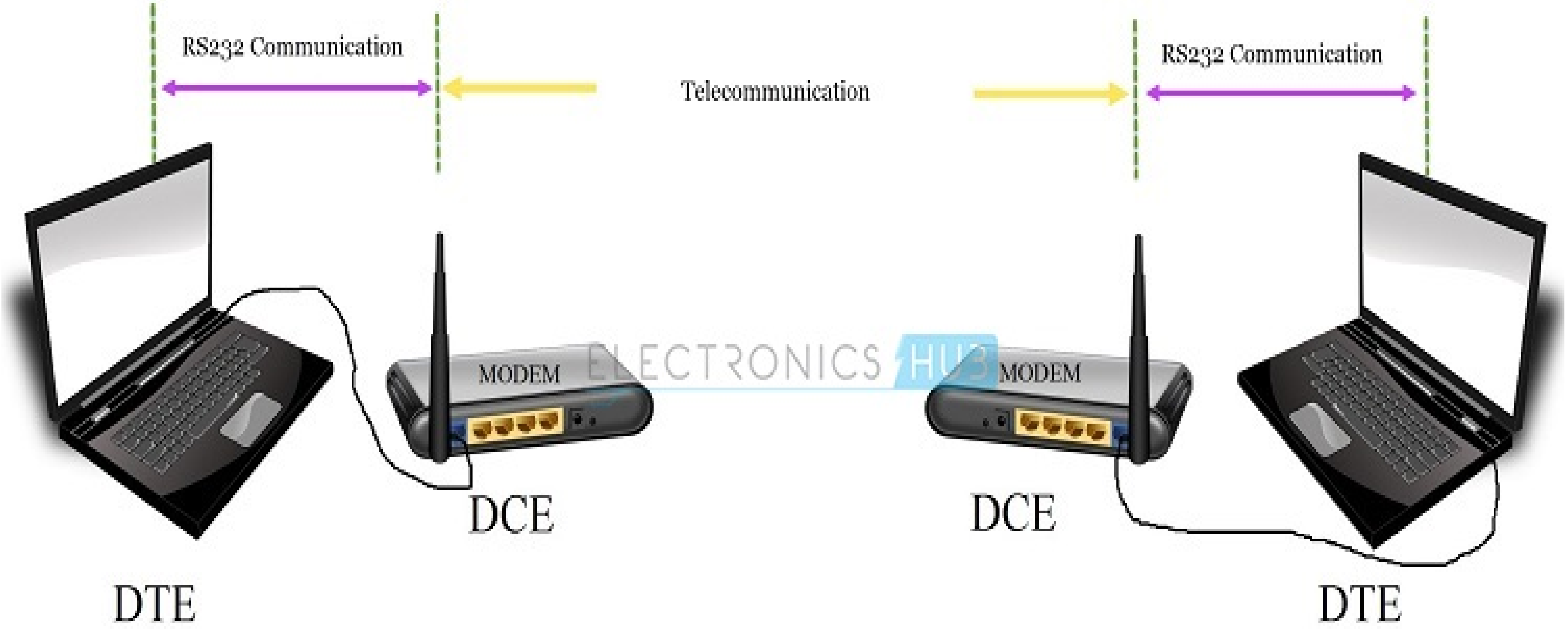


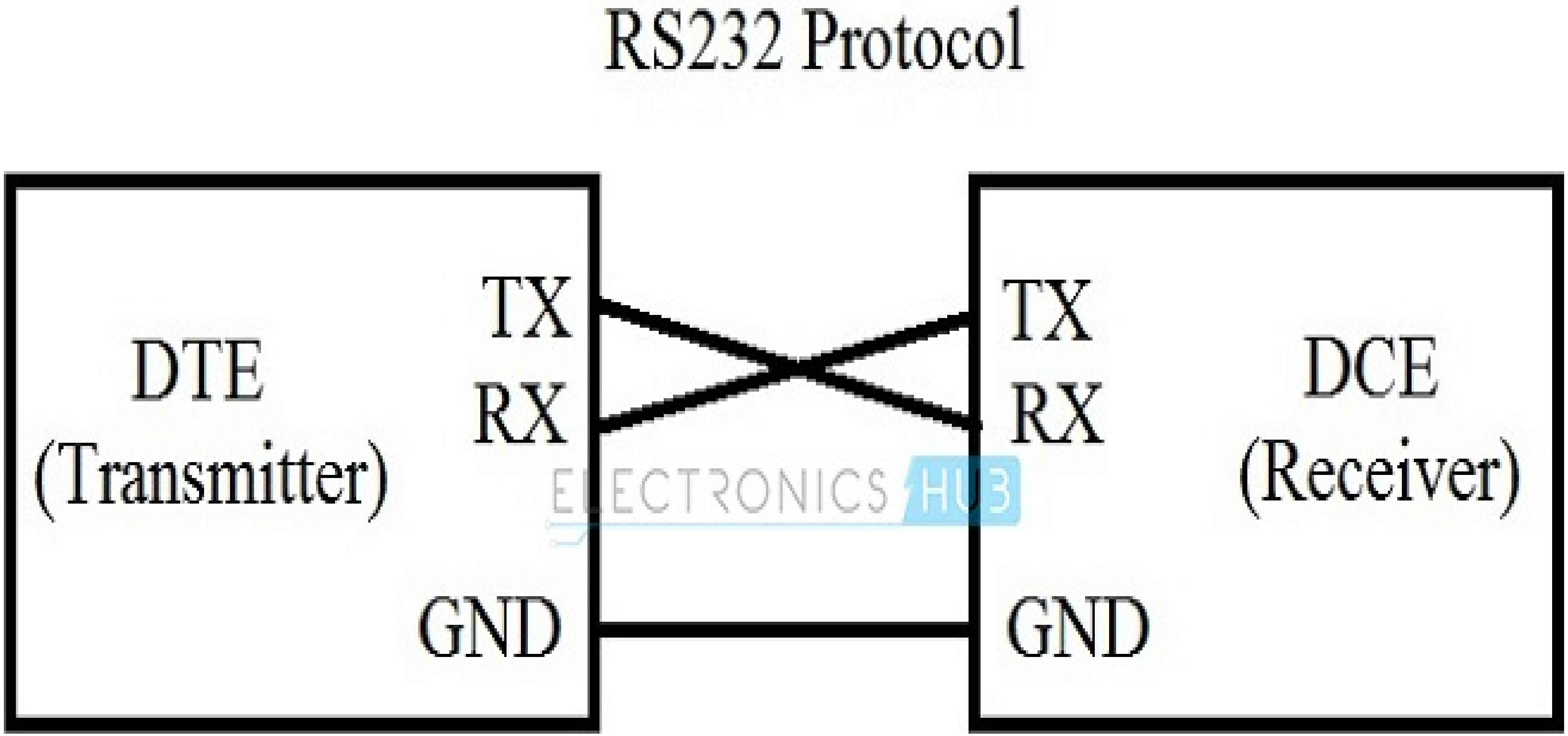
DB9 Connector



DB25 Connector

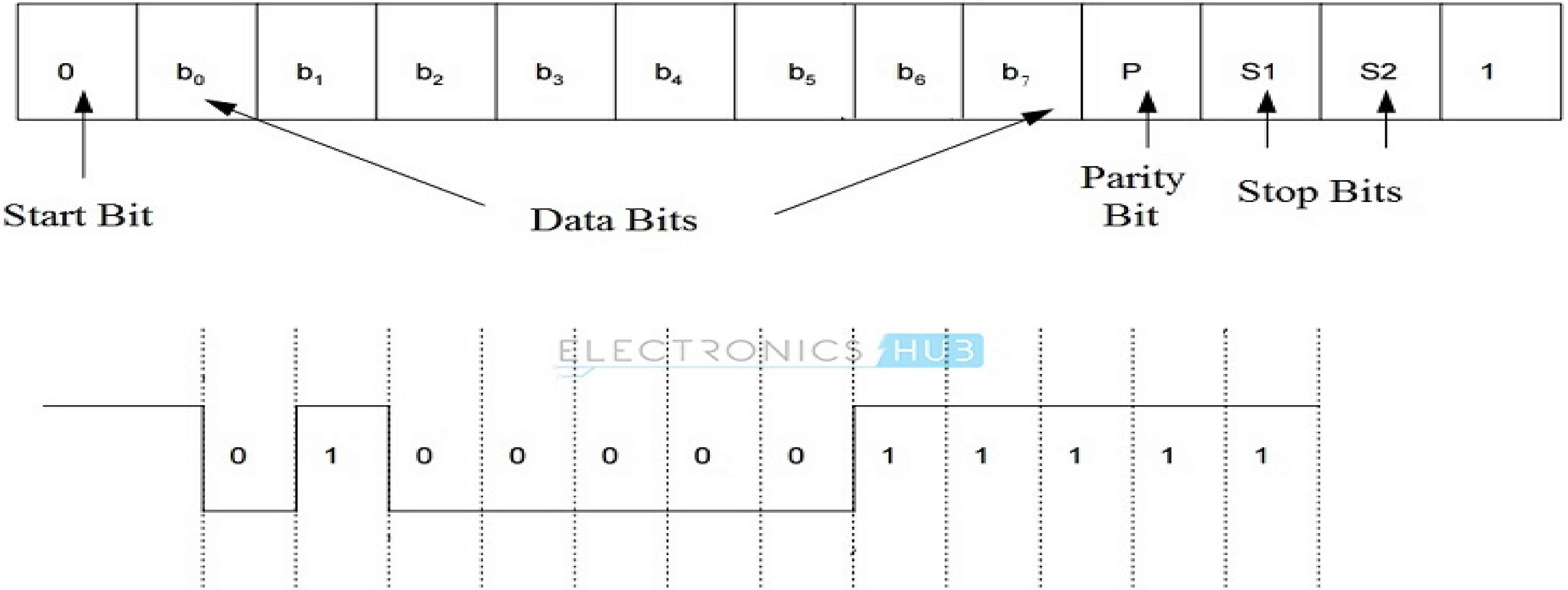
The Computer and Modem communicate with each other using RS232 interface and the communication between the modems is established using telecommunication links.




RS232 Protocol
DTE
(Transmitter)
TX
RX
GND
TX
RX
GND
DCE
(Receiver)

- **How RS232 Works?**
- In RS232, the data is transmitted serially in one direction over a single data line.
- In order to establish two way communication, we need at least three wires (RX, TX and GND) apart from the control signals. A byte of data can transmitted at any time provided the previous byte has already been transmitted.
- RS232 follows asynchronous communication protocol i.e. there is no clock signal to synchronize transmitter and receiver. Hence, it uses start and stop bits to inform the receiver when to check for data.
- There is a delay of certain time between the transmissions of each bit. This delay is nothing but an inactive state i.e. the signal is set to logic ‘1’ i.e. -12V (if you remember, logic ‘1’ in RS232 is -12V and logic ‘0’ is +12V).
- First, the transmitter i.e. the DTE sends a Start bit to the receiver i.e. the DCE to inform it that data transmission starts from next bit. The Start bit is always ‘0’ i.e. +12V. The next 5 to 9 characters are data bits.
- If parity bit is used, a maximum of 8 bits can be transmitted. If parity isn’t used, then 9 data bits can be transmitted. After the data is transmitted, the transmitter sends the stop bits. It can be either 1 bit or 1.5 bits or 2 bits long. The following image shows the frame format of the RS232 protocol.

frame format of the RS232 protocol.



RS-422

- RS-422 uses a twisted pair differential signal (more on this below) for receiving and transmitting data. It runs in full-duplex mode, with each transmission direction using two wires apiece. Putting on our math hats, this means that it requires a total of four wires. It is regularly used in a point to point (two devices talking to each other similar to your standard RS-232 connection) or multi-drop topology. When set up in the latter configuration, it can connect one driver with up to ten receivers on a single bus, often through a daisy chain.

RS-485

- Like RS-422, RS-485 also uses a twisted pair differential signal for receiving and transmitting data. However, this is most commonly done over a single twisted pair, which requires (as you probably guessed) two wires. The RS-485 standard was created to handle the problem of allowing several devices to talk to each other, and accordingly, it supports a multi-point topology. It can manage up to 32 devices, and the use of repeaters can increase this number to 256. This standard allows a reliable signal to persist under a higher load, which makes it a popular interface for industrial applications, including a variety of automation systems.
- RS-485 can run in full-duplex mode and when it does, it essentially becomes RS-422. In fact, you can often take hardware utilizing RS-485 and successfully drop it into an RS-422 setup. However, the reverse is not true, as RS-422 cannot operate in half-duplex mode, and therefore is unable to support multi-point topologies.

RS422 vs RS485

- RS422 and RS485 are hardware only standards. These Standards do not define how data is to be sent, AKA the protocol, nor do they define any speed, AKA Bits Per Second (bps). They exist ONLY to supplement other complete communication protocols, such as RS232, and allow these communication standards extend their communication in terms of physical distance and high speed while maintaining a robust reliability in doing so.
- What both RS422 and RS485 buses standards have in common that they both use hardware differential signalling techniques to allow a combination of high speed communication over long distances who ground potential differences can be non zero. However from a communication direction point of view, RS422 is not compatible with RS485, but RS485 can be made compatible with RS422.
- **FUNCTIONAL KEY BUS DIFFERENCE**
- The biggest difference between RS422 and RS485 is how they communicate with devices on a single pair of differential wires.

  RS422: <u>Each Bus only offers One-way communication.</u> A single transmitting (master) device to one or more receiving (slave) devices on a single pair of wires. Two-way communication requires TWO RS422 buses in parallel. One for each direction. Hence a two-way RS422 bus requires 4 wires.
- RS485: <u>Each Bus offers Two-way communication.</u> Multiple devices can share a single pair of wires. Each device has a transceiver allowing both transmitting and receiving capabilities. This allows point to point two way communication between any given device anywhere on the bus to any OTHER device anywhere on the RS485 bus. Hence two-way communication requires, at a minimum, ONE RS485 bus.
-

- **HARDWARE: RS422 vs RS485 TRANSCEIVER CHIP SIMILARITIES**

  These chips allow a given device to access the RS422 or RS485 bus respectively.   Both chips contains a transceiver. A transceiver contains both a transmitter and receiver in one part. One side of the chip interfaces with standard logic signals from the local device it is connect to while the other side connect to the bus. The transmitter take a digital signal from the device and translates it so it can drive the bus with the digital data. The receivers listens to the bus digital data for the signal on the bus and translates back into a digital signal for the device.

  HARDWARE: RS422 vs RS485 TRANSCEIVER CHIP DIFFERENCES

- **RS422 CHIP.**

  1) The transceivers are fully isolated from each other driving TWO independent one way RS422 bus's in OPPOSITE directions.   The transmitter drives one bus and the receiver receives from a 2nd bus. Hence this is a 4 wire bus.

- 2) At a minimum, two RS422 chips are required to make a complete two way RS422 bus. The transmitter of the first chip drives the receiver of the 2nd chip and the tranmitter of the 2nd chip drives the receiver of the first chip.

- 3) RS422 transmitter and receivers are always on and always connected to the RS422 bus. They cannot be disconnected.

- 4) The transmitter is ALWAYS driving the bus to a known data state (mark or space)

- 5) Each RS422 bus can only be driven by one transmitter.

- 6) Multiple RS422 receivers can monitor the a given RS422 bus.

- # RS485 CHIP:

1) The transceivers are fully connected to each other internally working with ONE RS485 bus.
2) RS485 transceivers have Enable and Disabled functions. The RS485 transmitter and receivers can be DISCONNECTED from the RS485 bus.
3) The ability to disable e tranmitter allows RS85 transceiver can drive the bus.
Unlike RS422 drivers, RS485 drivers have a enable pins on the transceivers which allows the chip to isolate itself from the bus. This is what make bi-directional communication possible.

THE DIFFERENTIAL SIGNALLING ADVANTAGE

RS422 and RS485 require three wires to send data. Two in the form of a twisted pair for data signalling and one for DC ground.

The digital signal data is converted from a standard logic signal reference to ground to become represented by the VOLTAGE POLARITY DIFFERENCE BETWEEN THE TWO SIGNAL WIRES.   Twisted Pair wire offers the properties of low mutual inductance between the two wires.

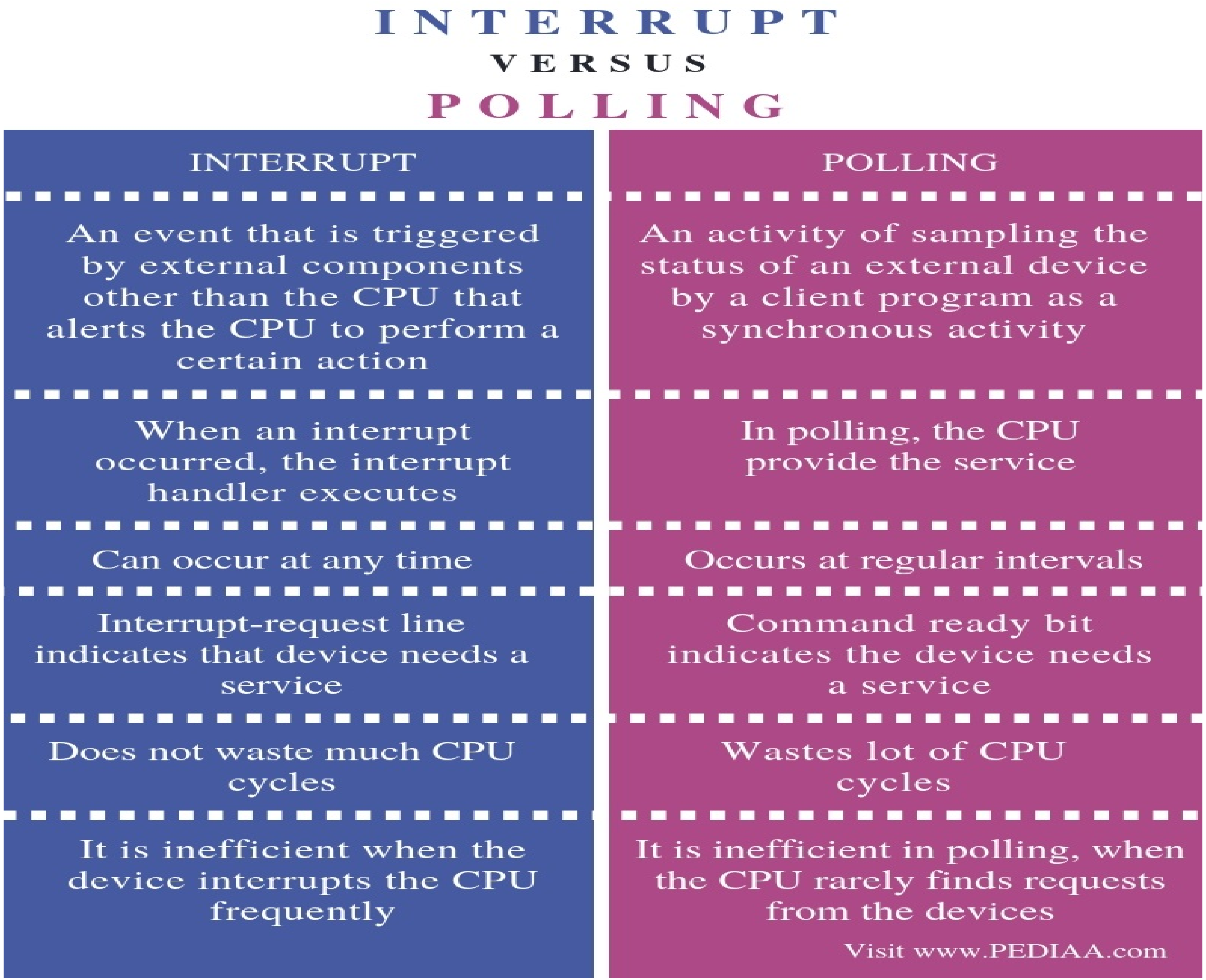
Differential Signalling Performance Benefits
1) takes the ground wire out of the high speed communication path by making no longer Representative of the signal ground. Why?
1a) Eliminate Ground Wire Inductance. Inductance is an electrical property that in high speed communication restricts the speed of communication.
1b) Differences in Earth Ground potentials.   The ground wire's only purpose now is to make the two earth ground potential difference between the devices establish a "common low ground voltage" within the range the RS422 and RS485 chips can operate with without being damaged.   Why? Earth Ground or true 0V is a relative value. It is relative to where your standing on Earth. One can take a volt meter and stick the ground probe into the earth your standing one and hold the positive probe in you hanc and you will measure something close to zero volts.   Take that same meter connect to your same earth ground and extend the positive lead a long distance to the location of the other device and you would read a much LARGER NON ZERO voltage.   should be noted when there are two devices separated by long distance, they will NOT have the same earth ground reference point. It made even worse by high power devices that leak current to Earth Ground.

2) takes the ground wire out of the high speed communication path.

- 2) reject noise from other adjacent wires since the ground is not involved. All noise is common mode which simply means that it appear on both differential wires at the same time.

Difference between interrupt and **polling**

- The main **difference between** interrupt and **polling is** that in interrupt, the device notifies the CPU that it requires attention while, in **polling**, the CPU continuously checks the status **of** the devices to find whether they require attention.
- In brief, an interrupt **is** asynchronous whereas **polling is** synchronous.



INTERRUPT VERSUS POLLING

INTERRUPT	POLLING
An event that is triggered by external components other than the CPU that alerts the CPU to perform a certain action	An activity of sampling the status of an external device by a client program as a synchronous activity
When an interrupt occurred, the interrupt handler executes	In polling, the CPU provide the service
Can occur at any time	Occurs at regular intervals
Interrupt-request line indicates that device needs a service	Command ready bit indicates the device needs a service
Does not waste much CPU cycles	Wastes lot of CPU cycles
It is inefficient when the device interrupts the CPU frequently	It is inefficient in polling, when the CPU rarely finds requests from the devices

Visit www.PEDIAA.com