

# Computer Organization and Architecture



Dr. Pradip Kumar Sahu  
Department of Information Technology

# Lecture of Module 1

## **Introduction**

# Overview

- ▶ **Computer Architecture**
- ▶ **Commuter Organization**
- ▶ **Structure and Function**
- ▶ **Functional units**
- ▶ **Basic Operational Concept**
- ▶ **Registers**
- ▶ **Bus Interconnection**
- ▶ **Bus Structure**
- ▶ **Harvard Architecture**
- ▶ **Von Neumann Architecture**
- ▶ **IAS Architecture**

# What is a Computer?



A computer is an Electronic device that can be programmed to process information, data etc. to yield meaningful results.



# Computer Architecture

## Computer Organization

- Attributes of a system visible to the programmer
- Have a direct impact on the logical execution of a program

- Instruction set, number of bits used to represent various data types, I/O mechanisms, techniques for addressing memory

Computer Architecture

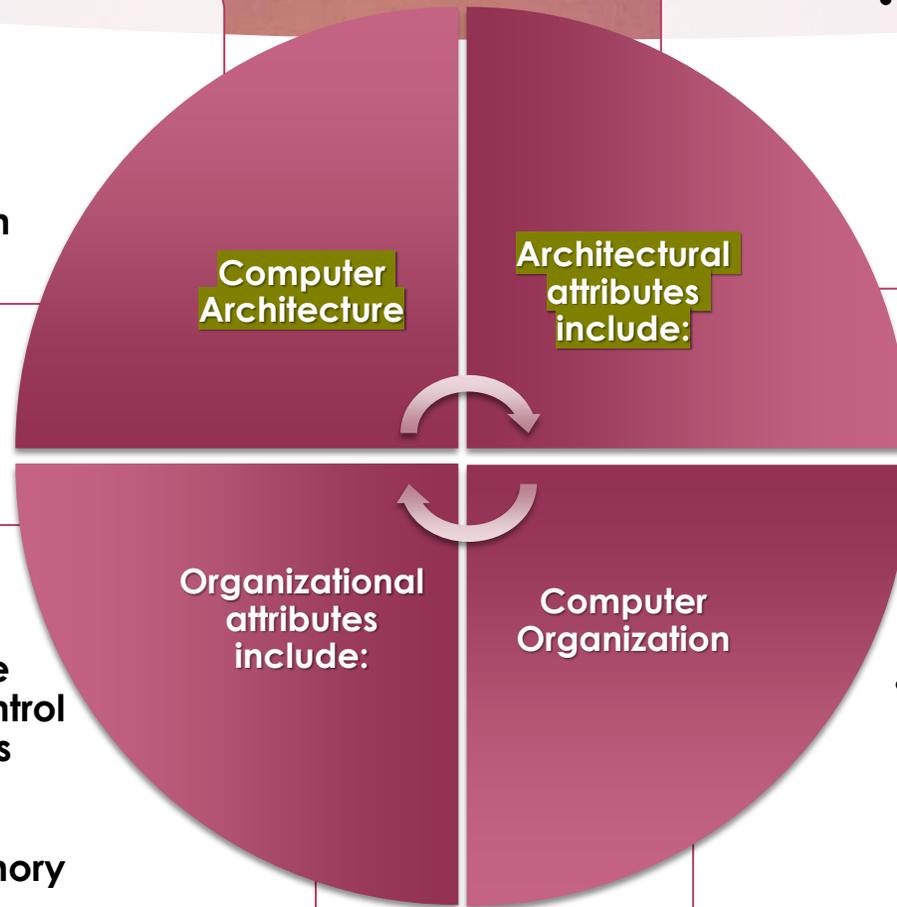
Architectural attributes include:

Organizational attributes include:

Computer Organization

- Hardware details transparent to the programmer, control signals, interfaces between the computer and peripherals, memory technology used

- The operational units and their interconnections that realize the architectural specifications



- **Whether a computer system can execute multiply instructions?**

Architectural Issue

Organizational Issue

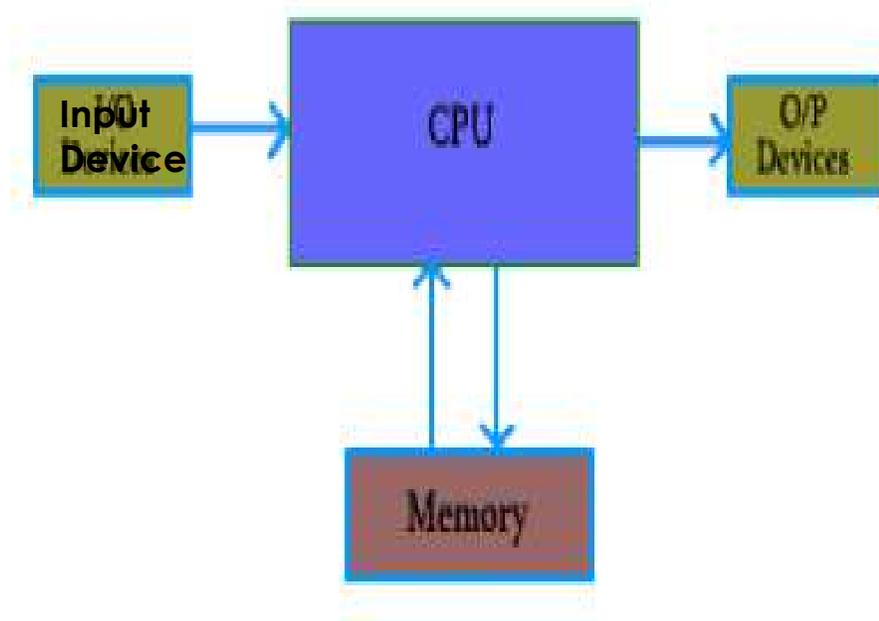
- **Whether to have a special multiply circuit? Or**
- **To have a method that makes repeated use of the add unit?**

Architectural Issue

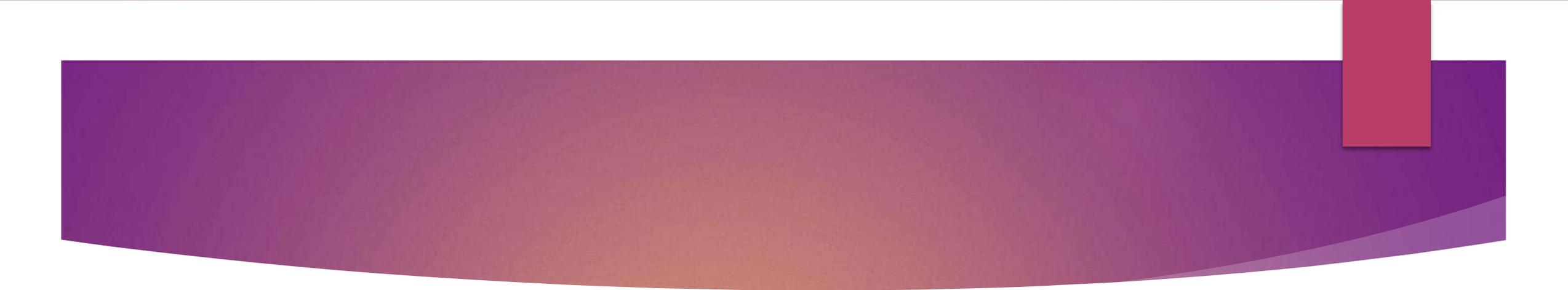
Organizational decision depends on:

- **The frequency of use of the multiply instruction**
- **The relative speed of the two approaches**
- **The cost and size of the special multiply unit**

# Structure and Function



- ▶ Structure
  - ▶ The way in which components relate to each other
- ▶ Function
  - ▶ The operation of individual components as part of the structure



- ▶ Hierarchical system

- ▶ Set of interrelated subsystems

- ▶ Hierarchical nature of complex systems is essential to both their design and their description

- ▶ Designer need only deal with a particular level of the system at a time

- ▶ Concerned with structure and function at each level

- ▶ Structure

- ▶ The way in which components relate to each other

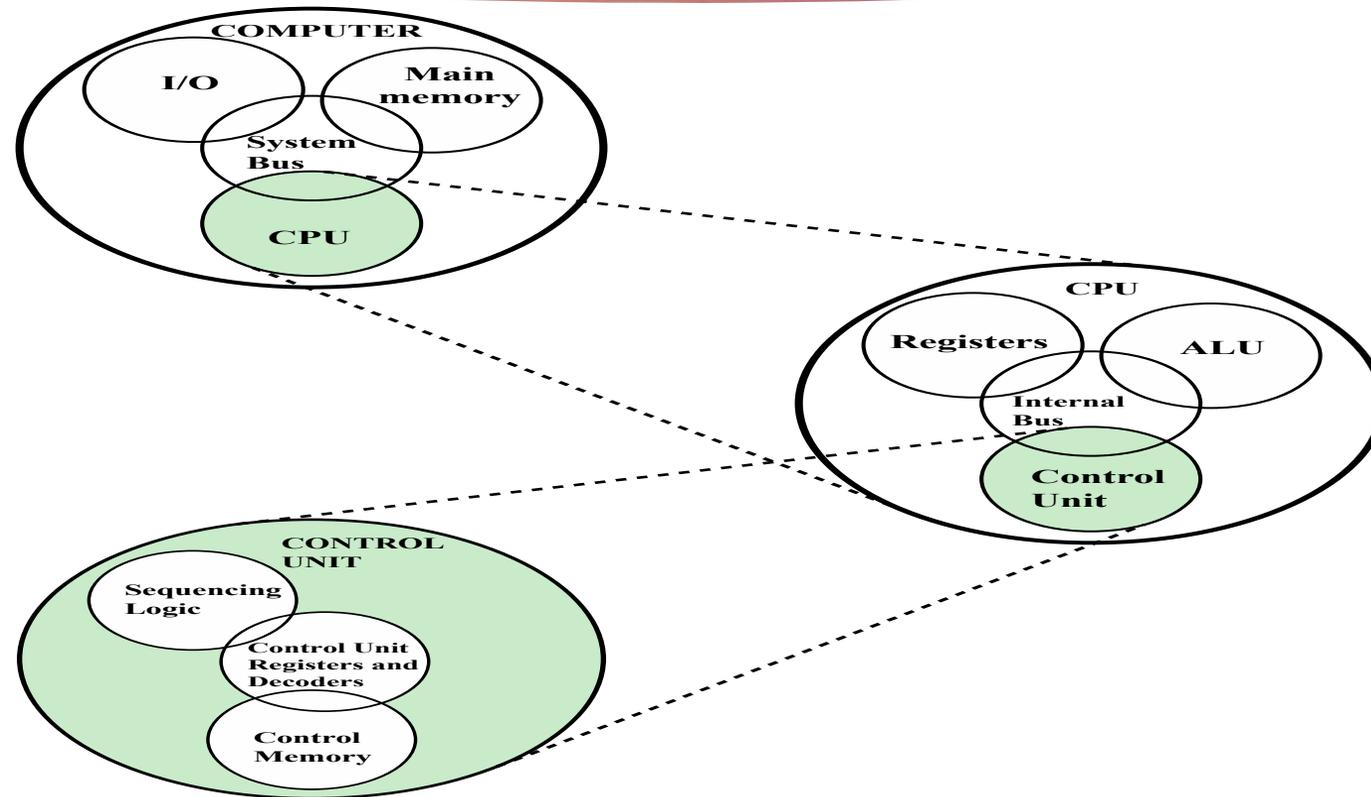
- ▶ Function

- ▶ The operation of individual components as part of the structure

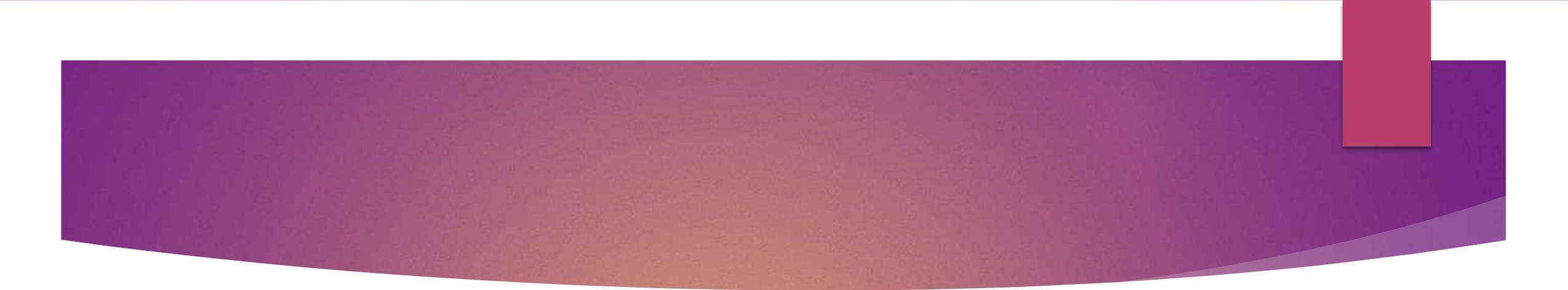
# Function

- ▶ There are four basic functions that a computer can perform:
  - ▶ **Data processing**
    - ▶ Data may take a wide variety of forms and the range of processing requirements is broad
  - ▶ **Data storage**
    - ▶ Short-term
    - ▶ Long-term
  - ▶ **Data movement**
    - ▶ Input-output (I/O) - when data are received from or delivered to a device (peripheral) that is directly connected to the computer
    - ▶ Data communications – when data are moved over longer distances, to or from a remote device
  - ▶ **Control**
    - ▶ A control unit manages the computer's resources and cares the performance of its functional parts in response to instructions

# Structure



**Figure 1.1 A Top-Down View of a Computer**

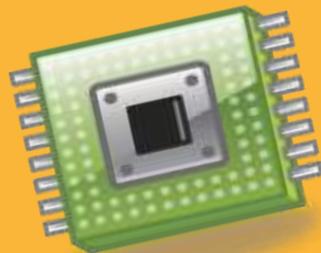
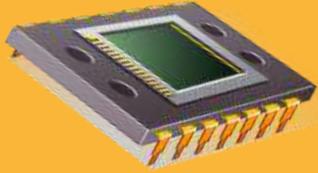


There are four main structural functional units of the computer:

- ✦ CPU – controls the operation of the computer and performs its data processing functions
- ✦ Main Memory – stores data
- ✦ I/O – moves data between the computer and its external environment
- ✦ System Interconnection – some mechanism that provides for communication among CPU, main memory, and I/O

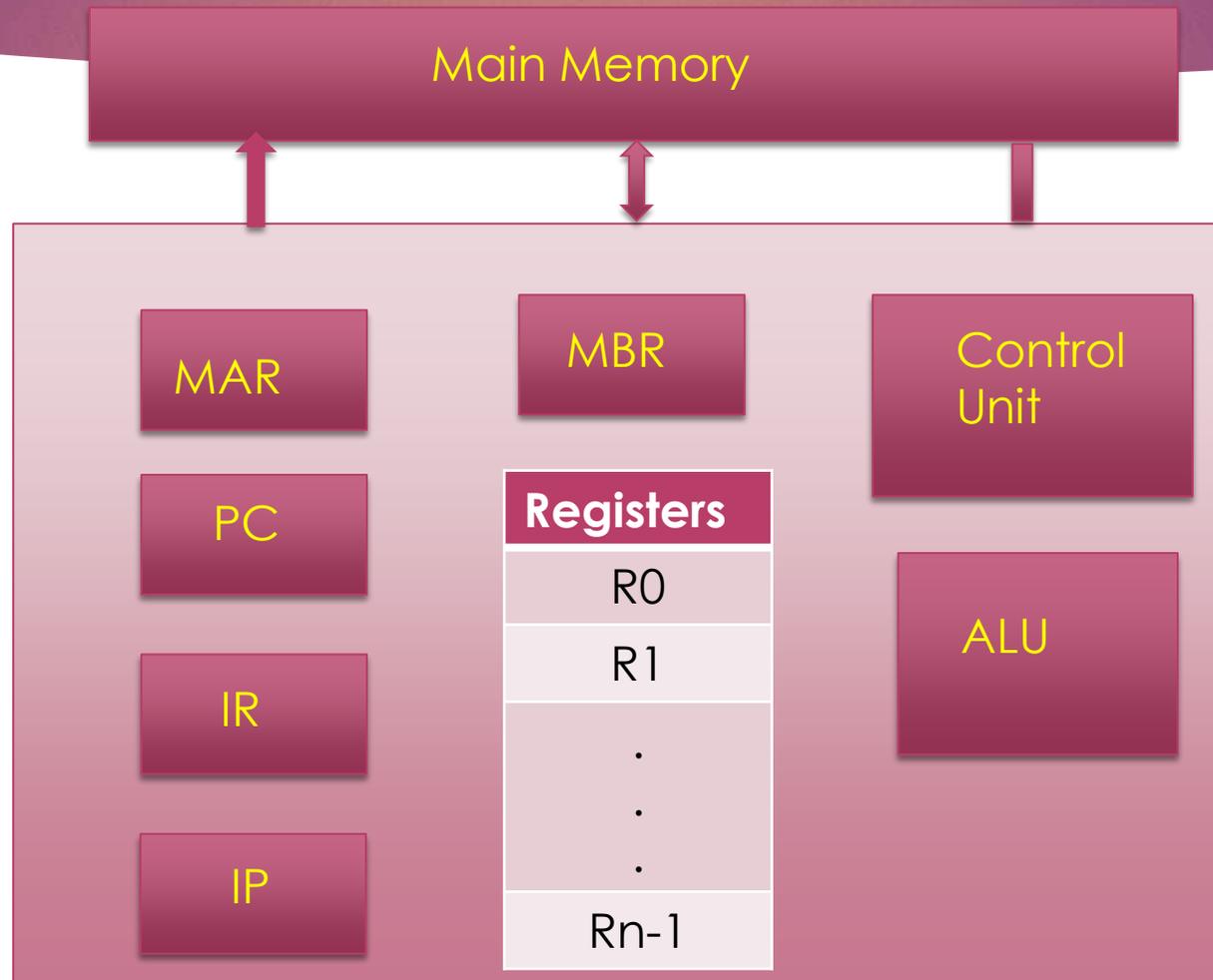
# CPU

Major structural components:



- ▶ **Control Unit**
  - ▶ Controls the operation of the CPU and hence the computer
- ▶ **Arithmetic and Logic Unit (ALU)**
  - ▶ Performs the computer's data processing function
- ▶ **Registers**
  - ▶ Provide storage internal to the CPU
- ▶ **CPU Interconnection**
  - ▶ Some mechanism that provides for communication among the control unit, ALU, and registers

# Basic Operational Concept



# Registers

## Memory buffer register (MBR)

- Contains a word to be stored in memory or sent to the I/O unit
- Or is used to receive a word from memory or from the I/O unit

## Memory address register (MAR)

- Specifies the address in memory of the word to be written from or read into the MBR

## Instruction register (IR)

- Contains the current opcode of the instruction being executed

## Instruction Pointer (IP)

- Holds the address of the current instruction on which processing is going on

## Program counter (PC)

- Contains the address of the next instruction to be fetched from memory

## General Purpose Registers (GPR)

- Arrays of registers present in the processor

# Other Registers

- ▶ **Stack Pointer (SP):** It is a memory pointer. It points to a memory location called STACK. The beginning of the Stack is defined by loading the starting address to the Stack pointer.
- ▶ **Base Register:** It stores the base address of the memory. Any address of any data is calculated logically after finding the address which is in the base register
- ▶ **Temporary Register (TR):** Holds temporary data during processing.
- ▶ **Flag Register(FR):** Shows the status of the system during processing. It is affected by ALU operations. It consists of different Flag bits.



# Flag Register

- ▶ **CF:** If there is a carry then flag bit is set, otherwise reset.
- ▶ **PF:** If even numbers 1's in the result then flag is set, otherwise reset.
- ▶ **AC:** If carry is generated from D3 during operation and passes to D4 then set, otherwise reset.
- ▶ **ZF:** If the result is zero then flag is set, otherwise reset.
- ▶ **SF:** If D7 is 1 then flag is set, otherwise reset (Signed number).
- ▶ **TF:** Trap is a non-maskable interrupt. When non-maskable interrupt is generated then flag is set.
- ▶ **IF:** If any interrupt is generated then flag is set.
- ▶ **DF:** When memory is accessed from lower location to higher location the flag is set. When memory is accessed from higher location to lower location the flag is reset.
- ▶ **OF:** When any overflow takes place during arithmetic or logical operations in Register, Stack, Queue, Array etc. then flag is set showing the overflow condition of the current operation.

# Bus Interconnection

- ▶ If a computer is to achieve a reasonable speed of operation, it must be organized so that all units can handle one full word of data at a given time.
- ▶ When a word of data is transferred between units, all its bits are transmitted in parallel.
- ▶ This requires a considerable number of wires (lines) to establish the necessary connections.
- ▶ **BUS:** A collection of wires that connects several devices to carry the information to or from different units of the system is called as BUS.

## Three types Bus

- **Data Bus**
- **Address Bus**
- **Control Bus**

The interconnection structure must support the following types of transfers:

Memory  
to  
processor

Processor  
reads an  
instruction or  
a unit of data  
from memory

Processor  
to  
memory

Processor  
writes a unit  
of data to  
memory

I/O to  
processor

Processor  
reads data  
from an I/O  
device via an  
I/O module

Processor  
to I/O

Processor  
sends data to  
the I/O  
device

I/O to or  
from  
memory

An I/O  
module is  
allowed to  
exchange  
data directly  
with memory  
without going  
through the  
processor  
using direct  
memory  
access

# Bus Interconnection

A communication pathway connecting two or more devices

- Key characteristic is that it is a shared transmission medium

Signals transmitted by any one device are available for reception by all other devices attached to the bus

- If two devices transmit during the same time period their signals will overlap and become garbled

Typically consists of multiple communication lines

- Each line is capable of transmitting signals representing binary 1 and binary 0

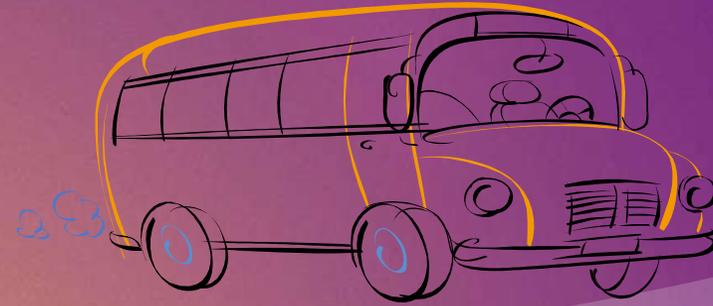
Computer systems contain a number of different buses that provide pathways between components at various levels of the computer system hierarchy

*System bus*

- A bus that connects major computer components (processor, memory, I/O)

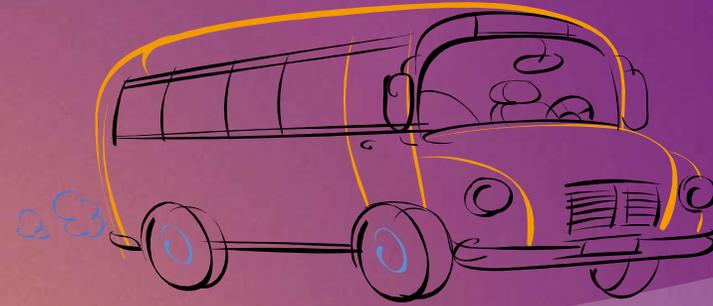
The most common computer interconnection structures are based on the use of one or more system buses

# Data Bus



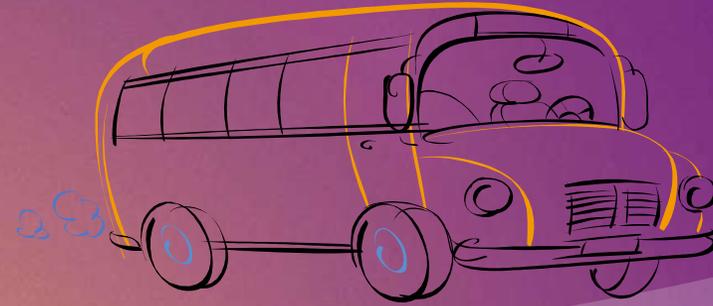
- ▶ Data lines that provide a path for moving data among system modules
- ▶ Number of wires depends on type of data transfer and word length
- ▶ May consist of 32, 64, 128, or more separate lines for parallel communication
- ▶ One line is required for serial communication
- ▶ The number of lines is referred to as the *width* of the data bus
- ▶ The number of lines determines how many bits can be transferred at a time (word length)
- ▶ The width of the data bus is a key factor in determining overall system performance
- ▶ **Direction is Bidirectional**

# Address Bus



- ▶ Used to designate the source or destination of the data
  - ▶ If the processor wishes to read or write a word of data from or to memory it puts the address of the desired word on the address lines
- ▶ Width determines the maximum possible memory capacity of the system
- ▶ Also used to address I/O ports
  - ▶ Used to select a I/O port
- ▶ **Direction is unidirectional**

# Control Bus



- ▶ All the functions of the system must be synchronized and controlled
- ▶ This is the function of control unit which provides control signals through buses
- ▶ Used to control the access and the use of the data and address lines
- ▶ Because the data and address lines are shared by all components there must be a means of controlling their use
- ▶ Control signals transmit both command and timing information among system modules
- ▶ Timing signals indicate the validity of data and address information
- ▶ Command signals specify operations to be performed
- ▶ Each line of the bus indicates a particular control signal
- ▶ A particular control line may be unidirectional or bidirectional but collectively as a bus no concept of direction

# Bus Structure

- ▶ According to the connection mechanism of different functional units the Bus structures are of two types

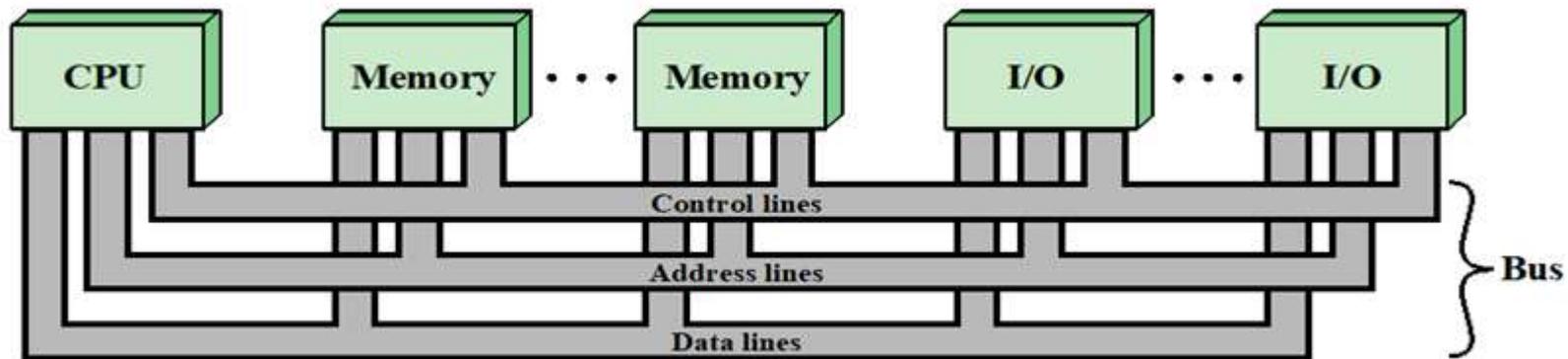
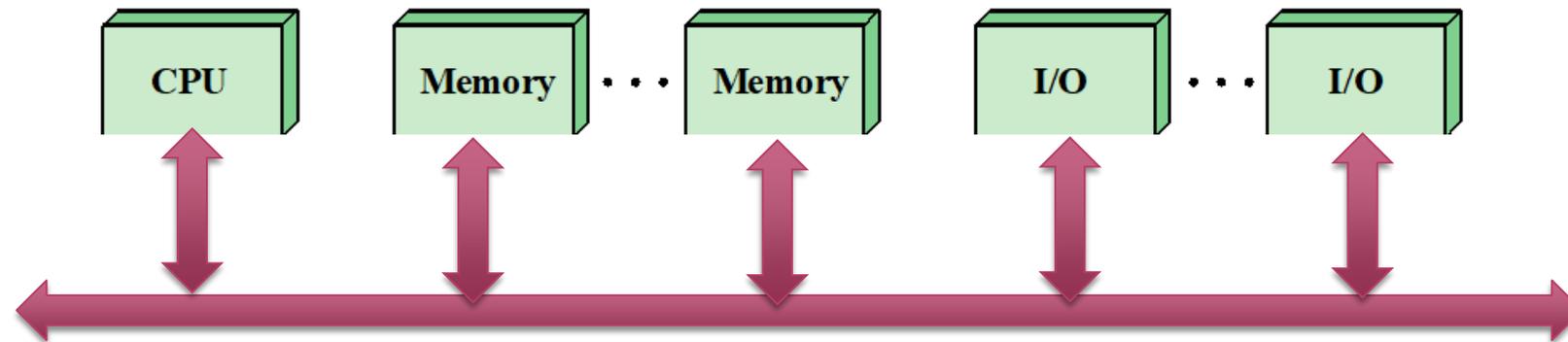
Single Bus structure

Multi-Bus structure

Single Bus structure:

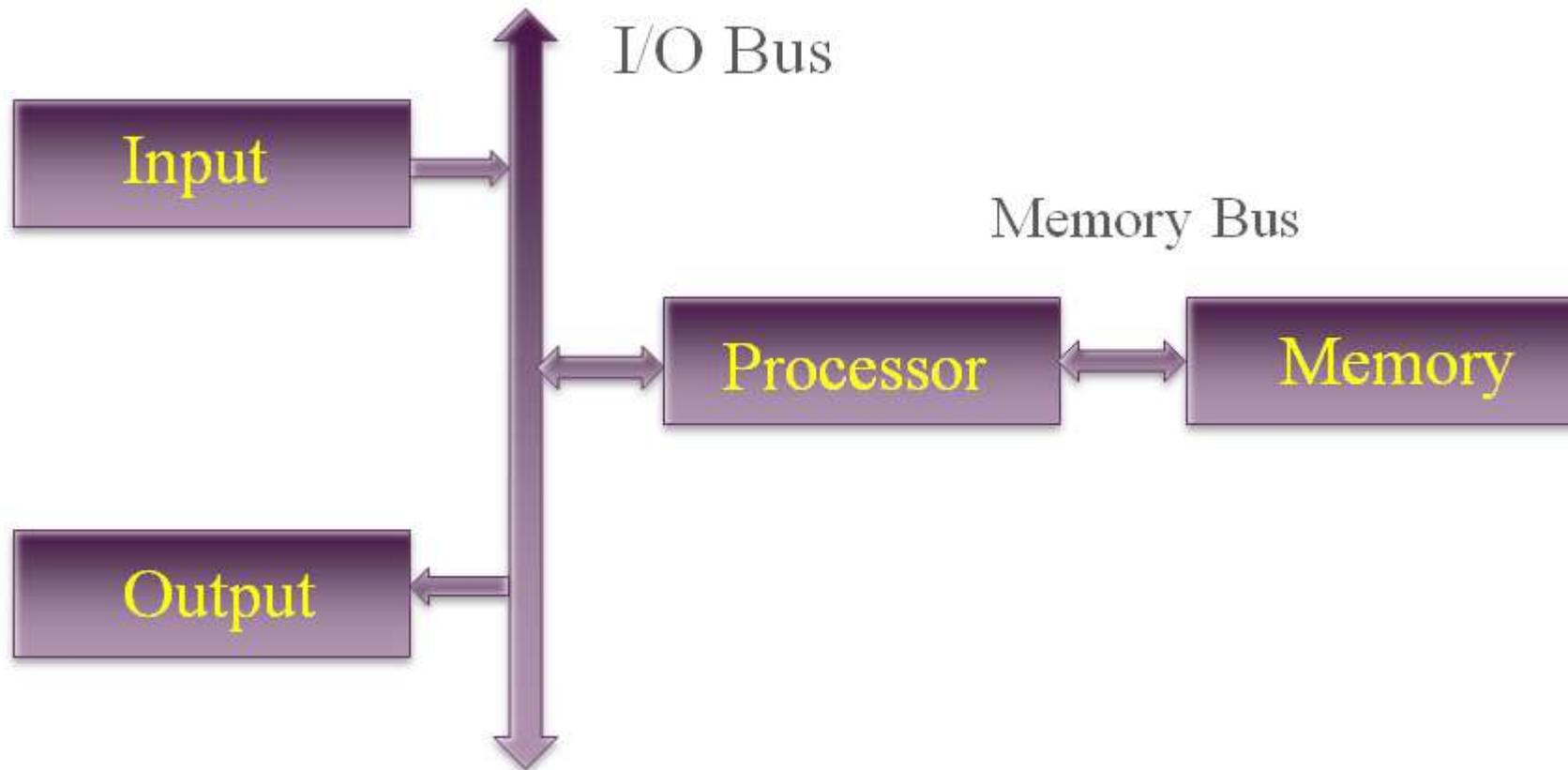
- ▶ All units are connected to single I/O bus
- ▶ At any given time two units can actively use the bus
- ▶ Bus control is used to arbitrate multiple requests for use of bus
- ▶ Flexibility for attaching peripheral devices
- ▶ Low hardware complexity
- ▶ Low cost
- ▶ But, slower data transfer

# Single Bus structure

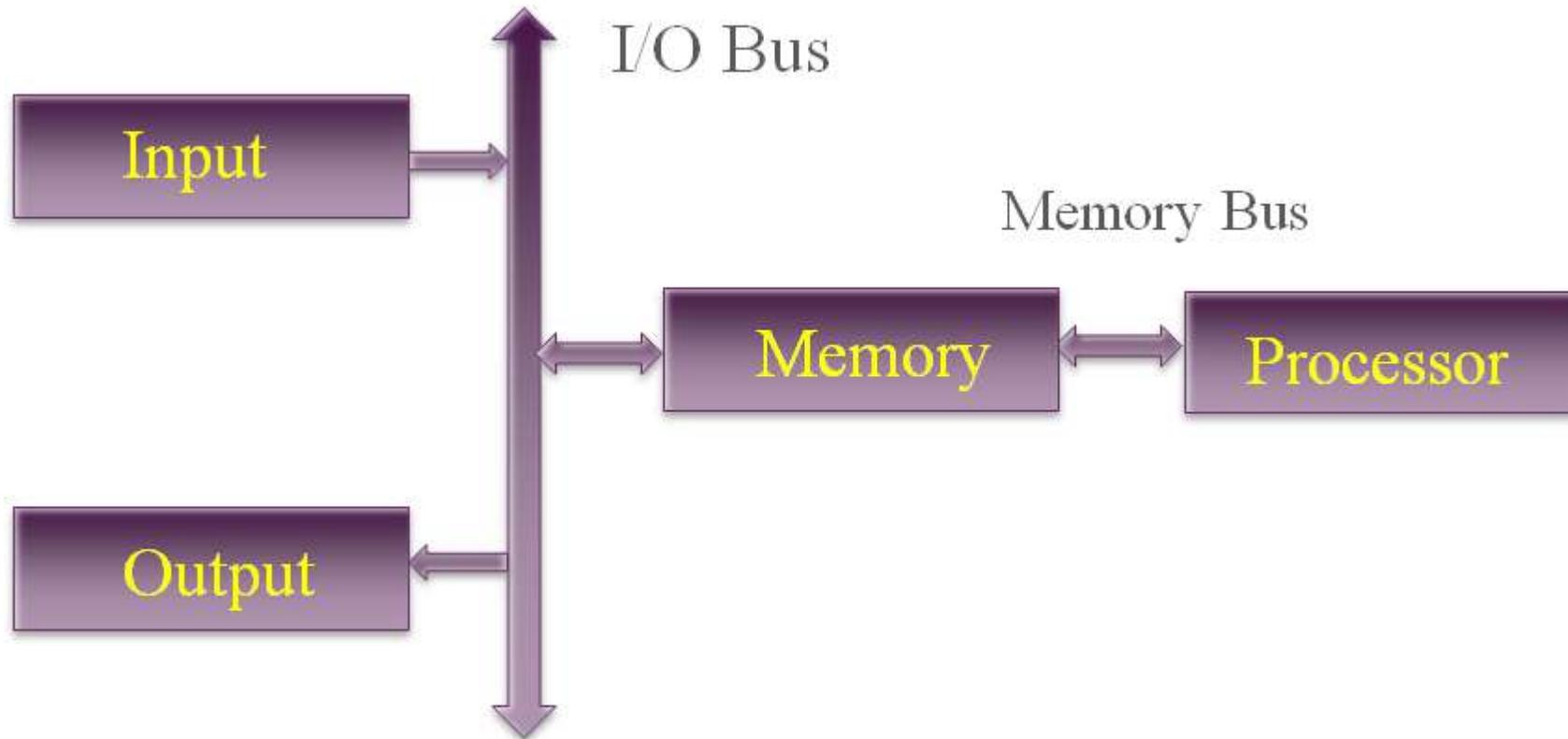


Bus Interconnection Scheme

# Multi Bus structure:



- ▶ It is a simplest Multi bus (two bus) computer
- ▶ The processor interacts with memory through **memory bus**
- ▶ Input and output functions are handled over an **I/O bus**
- ▶ Data passes to memory for processing through the processor
- ▶ I/O transfers are usually under direct control of the processor
- ▶ Processor initiates the transfer and monitors their progress until completion
- ▶ In this architecture the processor sit ideally after initiating the I/O operations till completion
- ▶ Wastage of CPU time which degrades the performance
- ▶ So, another multi bus architecture has been developed to enhance the performance of the system



- ▶ It is another Multi bus (two bus) architecture
- ▶ Here, the position of memory and processor interchanged
- ▶ I/O transfers are performed directly to or from memory
- ▶ But, memory can not control the I/O transfer
- ▶ So, a control circuitry as part of the I/O equipment is necessary
- ▶ That control circuitry is a special purpose processor called as **Peripheral Processor** or **Secondary Processor** or **I/O Channel** which controls the I/O transfer
- ▶ The main processor initiates I/O transfer by passing required information to the I/O channel
- ▶ The I/O channel then takes over and controls the actual transfer of data
- ▶ During I/O operations now the main processor is free and it can perform other CPU operations
- ▶ So, the performance enhanced

# Design of practical Computer

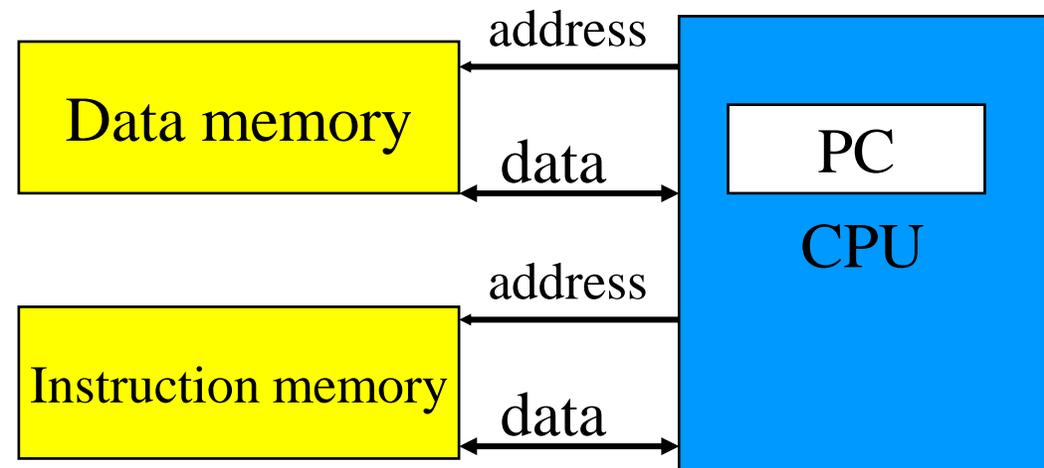
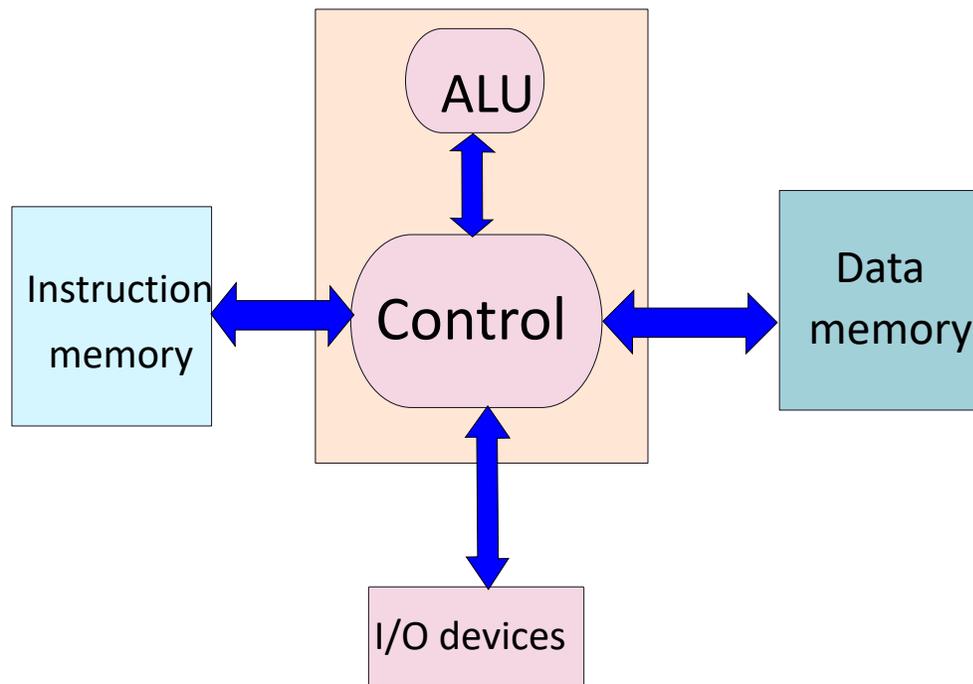
Two proposed architectural design

Harvard Architecture

Von Neumann Architecture

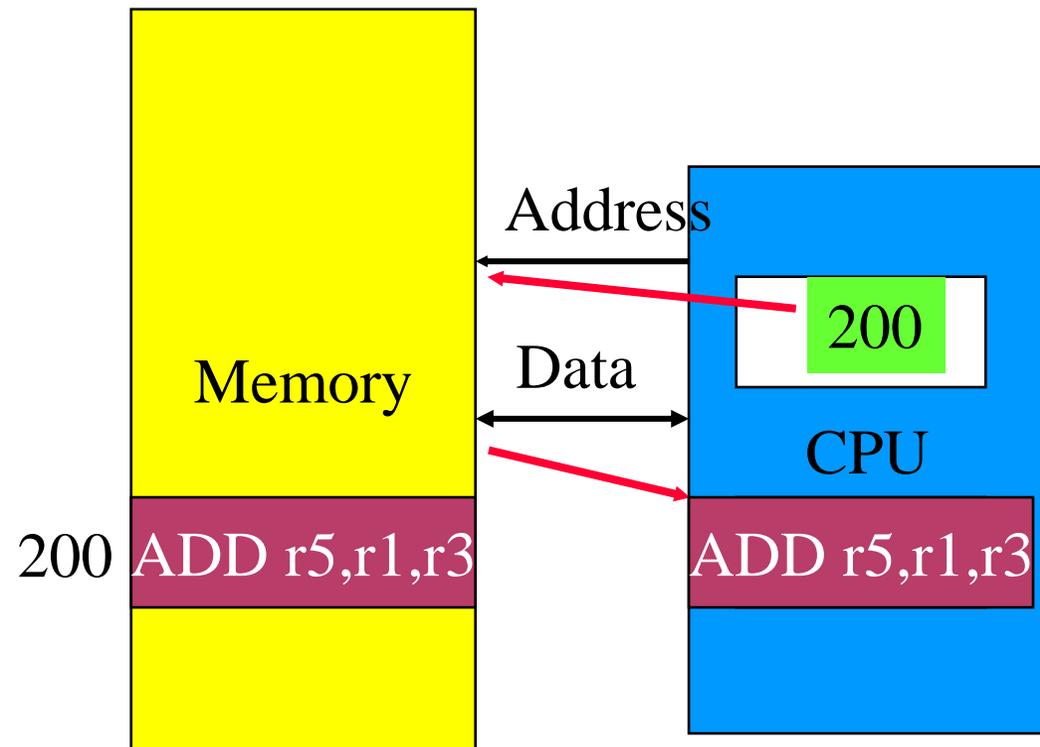
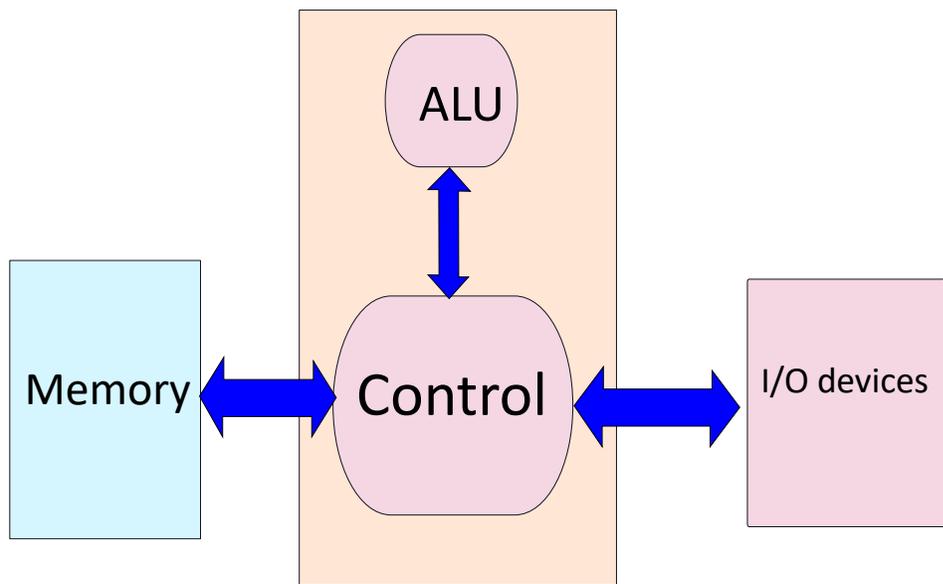
# Harvard Architecture

- ▶ Separate data and instruction memories



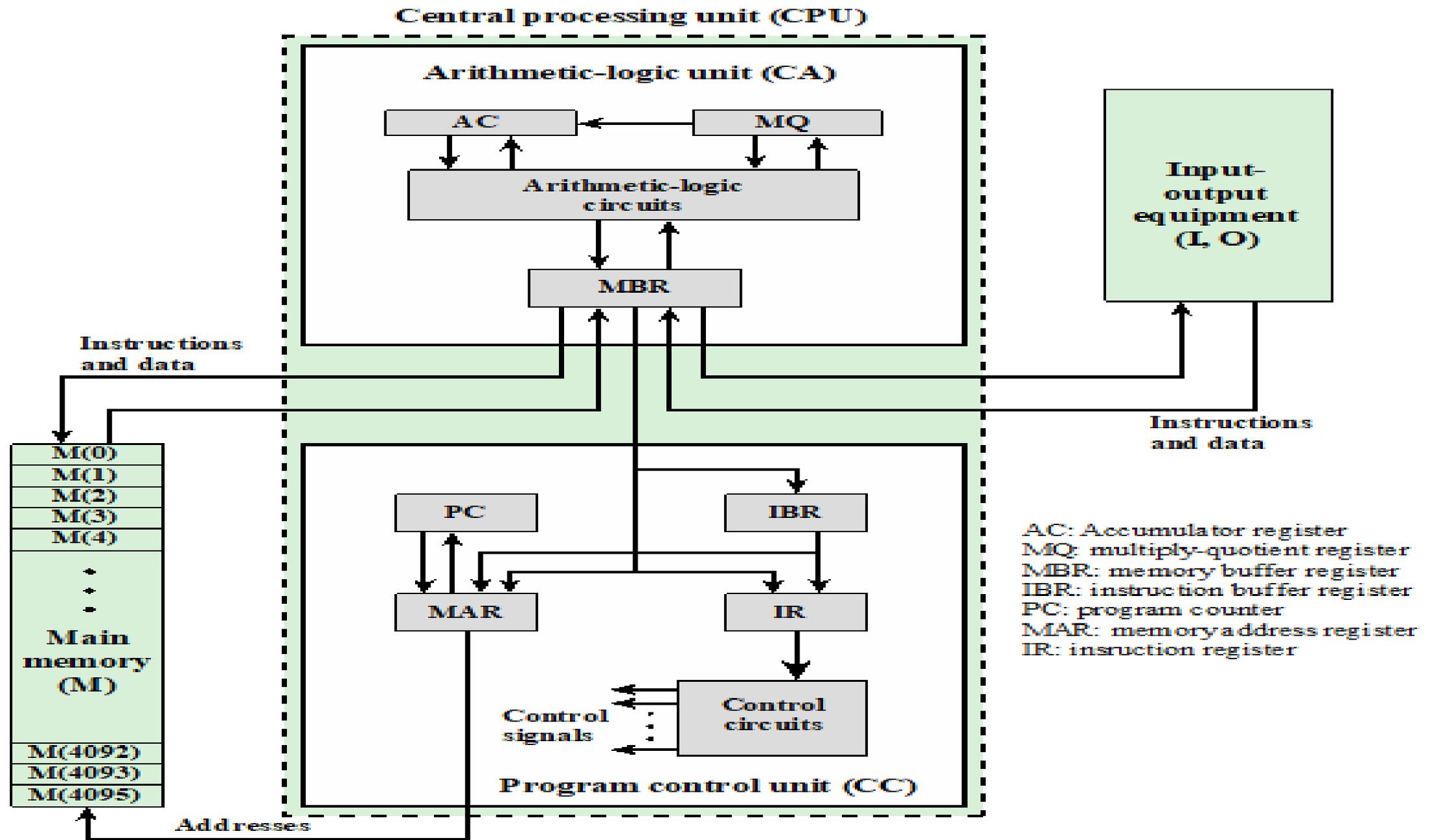
# Von Neumann Architecture

- ▶ Contemporary computer designs are based on concepts developed by John Von Neumann at the Institute for Advanced Studies, Princeton
- ▶ Referred to as the *Von Neumann Architecture* and is based on three key concepts:
  - ▶ **Stored Program Architecture**: Data and instructions are stored in a single read-write memory
  - ▶ The contents of this memory are addressable by location, without regard to the type of data contained there
  - ▶ Execution occurs in a sequential fashion (unless explicitly modified) from one instruction to the next
  - ▶ Example is **IAS Computer** developed by John Von Neumann and group at the **Institute for Advanced Studies**, Princeton

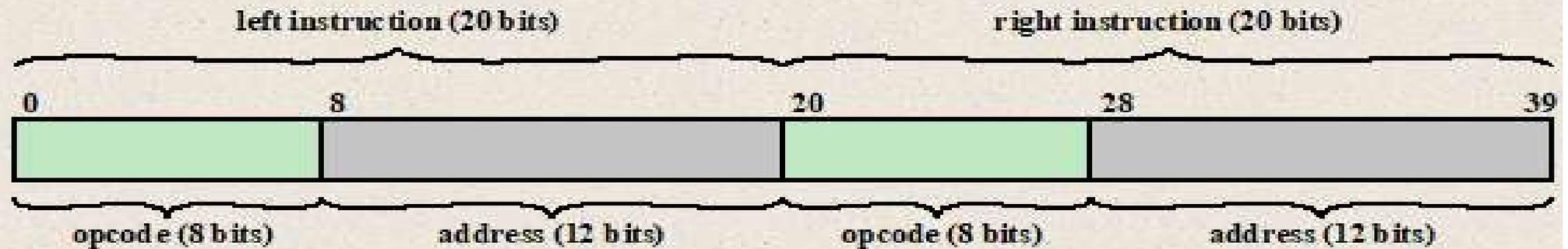
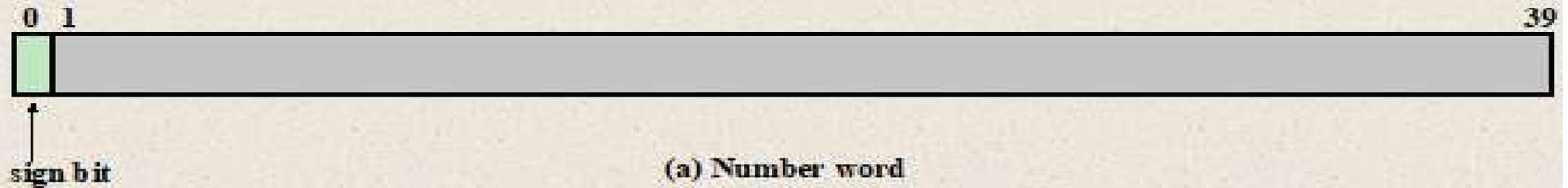


# IAS Computer

- ▶ IAS computer
  - ▶ Fundamental design approach was the stored program concept
    - ▶ Attributed to the mathematician John von Neumann
    - ▶ First publication of the idea was in 1945
  - ▶ Design began at the Institute for Advanced Studies, Princeton
  - ▶ Completed in 1952
  - ▶ Prototype of all subsequent general-purpose computers



# IAS Instruction format



# Registers

## Memory buffer register (MBR)

- Contains a word to be stored in memory or sent to the I/O unit
- Or is used to receive a word from memory or from the I/O unit

## Memory address register (MAR)

- Specifies the address in memory of the word to be written from or read into the MBR

## Instruction register (IR)

- Contains the 8-bit opcode instruction being executed

## Instruction buffer register (IBR)

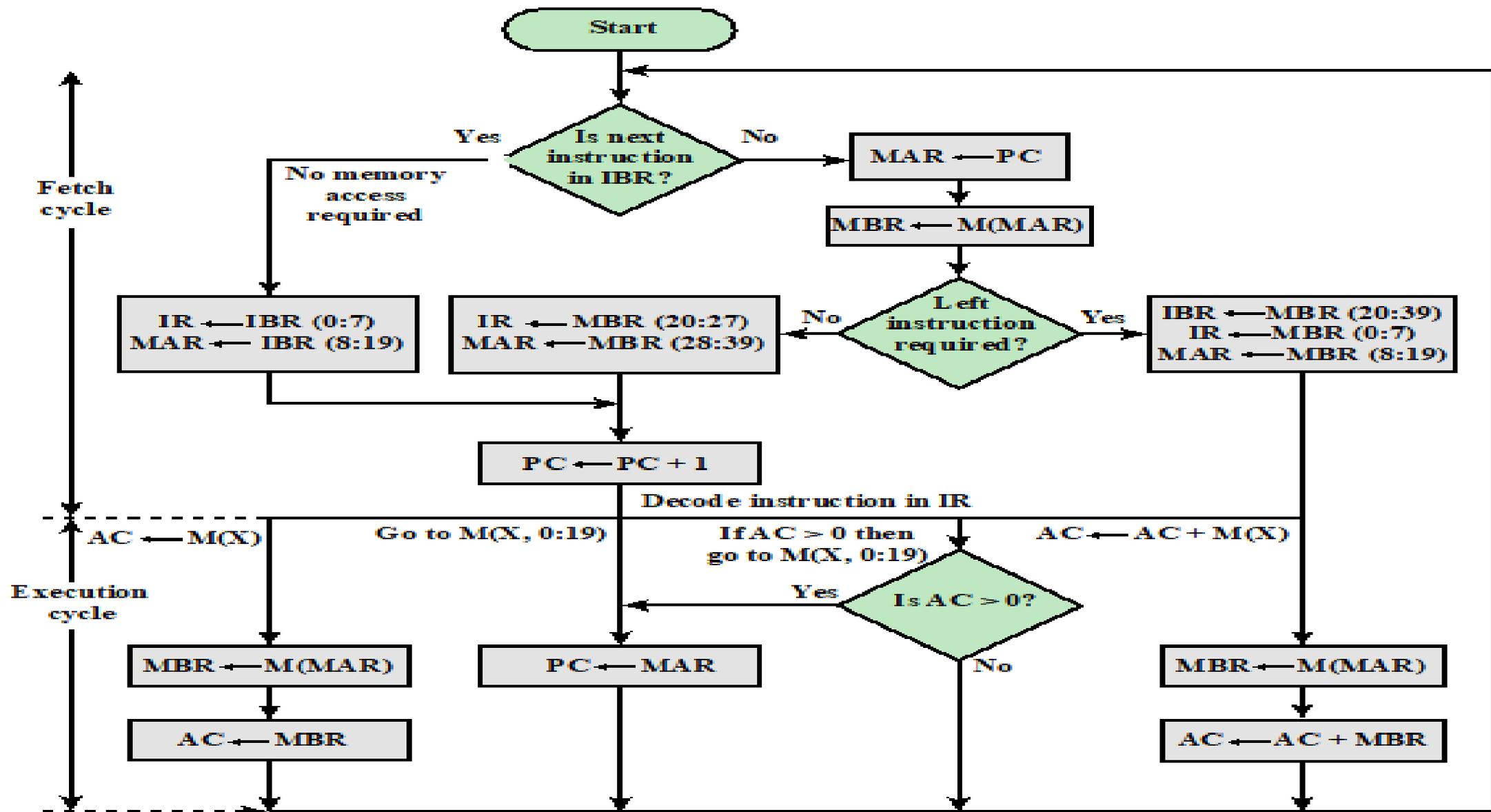
- Employed to temporarily hold temporarily the right-hand instruction from a word in memory

## Program counter (PC)

- Contains the address of the next instruction pair to be fetched from memory

## Accumulator (AC) and multiplier quotient (MQ)

- Hold operands and results of ALU operations. The most significant 40 bits are stored in the AC and the least significant in the MQ

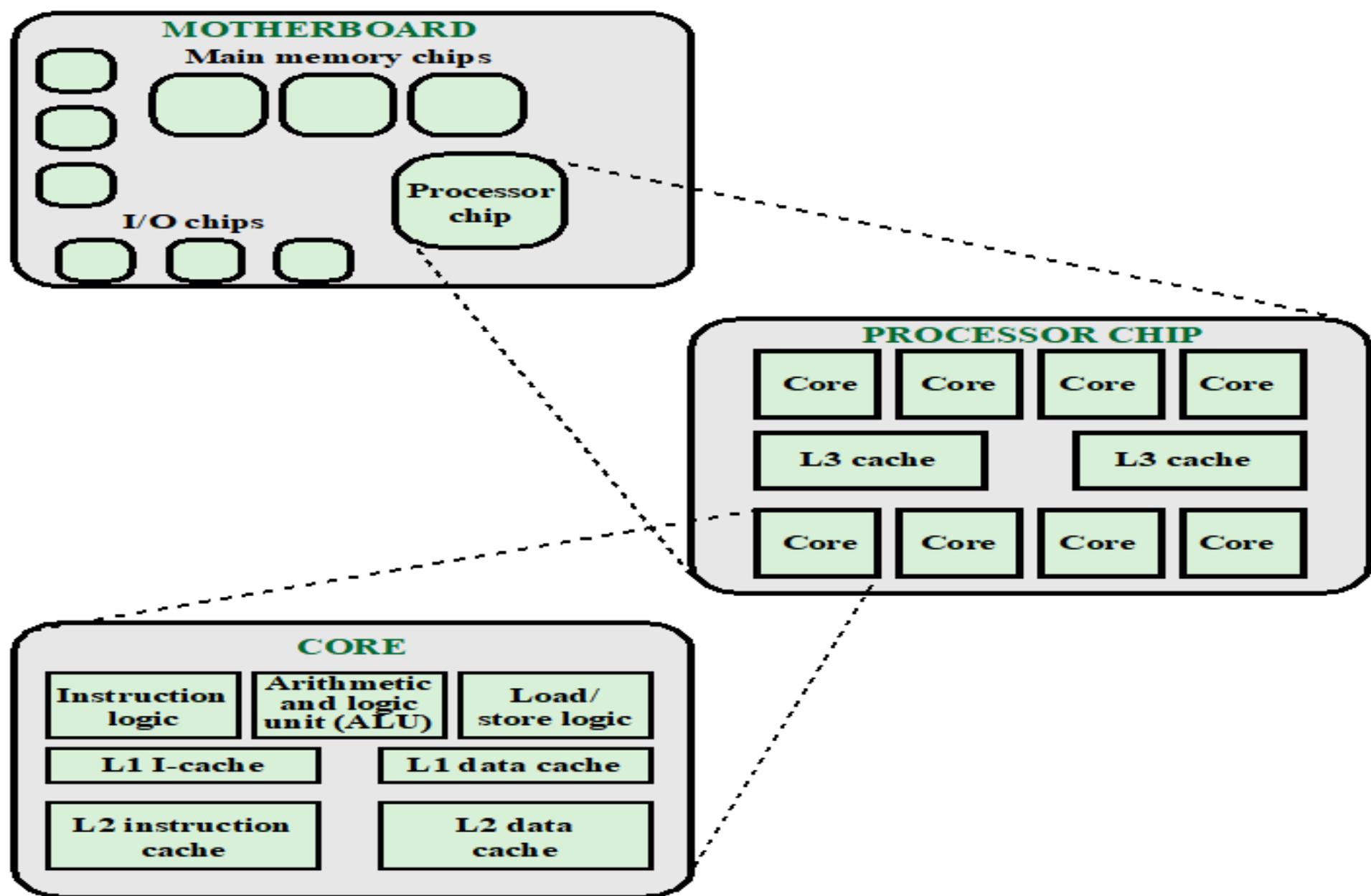


M(X) = contents of memory location whose address is X  
 (i:j) = bits i through j

**Partial Flowchart of IAS Operation**

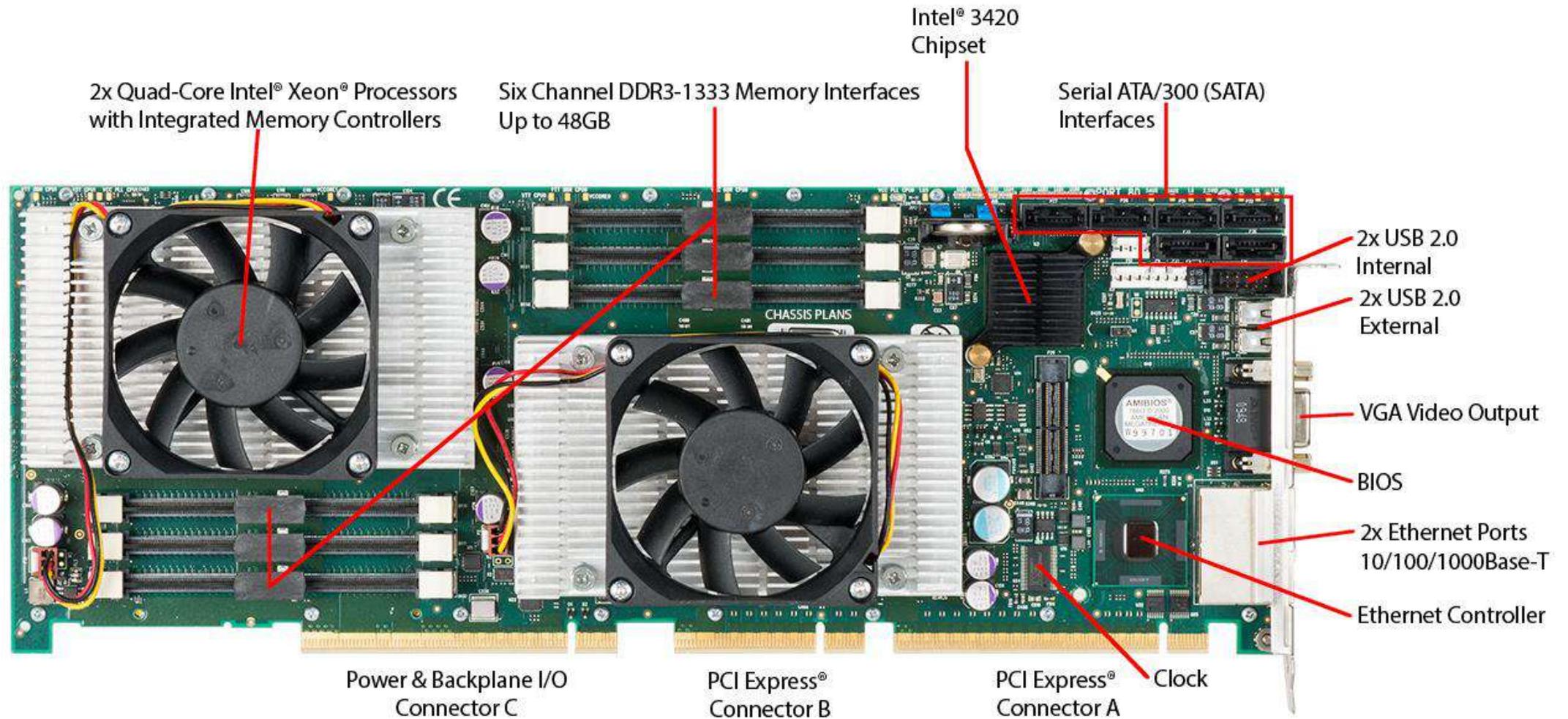
# IAS Instruction Set

Instruction Type	Opcode	Symbolic Representation	Description
Data transfer	00001010	LOAD MQ	Transfer contents of register MQ to the accumulator AC
	00001001	LOAD MQ,M(X)	Transfer contents of memory location X to MQ
	00100001	STOR M(X)	Transfer contents of accumulator to memory location X
	00000001	LOAD M(X)	Transfer M(X) to the accumulator
	00000010	LOAD -M(X)	Transfer -M(X) to the accumulator
	00000011	LOAD  M(X)	Transfer absolute value of M(X) to the accumulator
Unconditional branch	00000100	LOAD - M(X)	Transfer - M(X)  to the accumulator
	00001101	JUMP M(X,0:19)	Take next instruction from left half of M(X)
Conditional branch	00001110	JUMP M(X,20:39)	Take next instruction from right half of M(X)
	00001111	JUMP+ M(X,0:19)	If number in the accumulator is nonnegative, take next instruction from left half of M(X)  <i>JU MP + M(X ,20: 39)</i>  <i>If number in the accumulator is nonnegative, take next instruction from right half of M(X)</i>
Arithmetic	00000101	ADD M(X)	Add M(X) to AC; put the result in AC
	00000111	ADD  M(X)	Add  M(X)  to AC; put the result in AC
	00000110	SUB M(X)	Subtract M(X) from AC; put the result in AC
	00001000	SUB  M(X)	Subtract  M(X)  from AC; put the remainder in AC
	00001011	MUL M(X)	Multiply M(X) by MQ; put most significant bits of result in AC, put least significant bits in MQ
	00001100	DIV M(X)	Divide AC by M(X); put the quotient in MQ and the remainder in AC
	00010100	LSH	Multiply accumulator by 2; i.e., shift left one bit position
00010101	RSH	Divide accumulator by 2; i.e., shift right one position	
Address modify	00010010	STOR M(X,8:19)	Replace left address field at M(X) by 12 rightmost bits of AC
	00010011	STOR M(X,28:39)	Replace right address field at M(X) by 12 rightmost bits of AC



**Simplified View of Major Elements of a Multicore Computer**

# Motherboard with Two Intel Quad-Core Xeon Processors





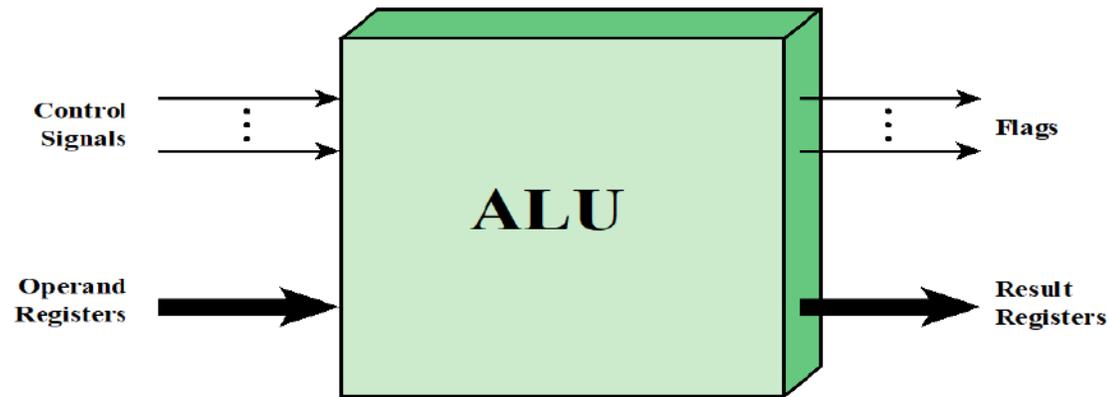
# **Computer Arithmetic**

# Overview

- ▶ **Arithmetic Operations**
- ▶ **Binary Arithmetic**
- ▶ **Signed Binary Numbers**
- ▶ **Decimal Arithmetic operation**
- ▶ **Floating point representation**
- ▶ **Floating point Arithmetic**
- ▶ **General Multiplication**
- ▶ **Booth Multiplication**
- ▶ **Array Multiplier**
- ▶ **Division**

# Arithmetic & Logic Unit (ALU)

- ▶ Part of the computer that actually performs arithmetic and logical operations on data
- ▶ All of the other elements of the computer system are there mainly to bring data into the ALU for it to process and then to take the results back out
- ▶ Based on the use of simple digital logic devices that can store binary digits and perform simple Boolean logic operations



**ALU Inputs and Outputs**

# Arithmetic Operations

## Addition

- ▶ Follow same rules as in decimal addition, with the difference that when sum is 2 indicates a carry (not a 10)
- ▶ Learn new carry rules
  - ▶  $0+0 = \text{sum } 0 \text{ carry } 0$
  - ▶  $0+1 = 1+0 = \text{sum } 1 \text{ carry } 0$
  - ▶  $1+1 = \text{sum } 0 \text{ carry } 1$
  - ▶  $1+1+1 = \text{sum } 1 \text{ carry } 1$

<b>Carry</b>	<b>1</b>	<b>1</b>	<b>1</b>	<b>1</b>	<b>1</b>	<b>0</b>
<b>Augend</b>	<b>0</b>	<b>0</b>	<b>1</b>	<b>0</b>	<b>0</b>	<b>1</b>
<b>Addend</b>	<b>0</b>	<b>1</b>	<b>1</b>	<b>1</b>	<b>1</b>	<b>1</b>
<b>Result</b>	<b>1</b>	<b>0</b>	<b>1</b>	<b>0</b>	<b>0</b>	<b>0</b>

$$\begin{array}{r} 111 \\ 0101 \\ +1011 \\ \hline 10000 \end{array}$$

Carry Values

## Subtraction

- ▶ Learn new borrow rules
  - ▶  $0-0 = 1-1 = 0$  borrow 0
  - ▶  $1-0 = 1$  borrow 0
  - ▶  $0-1 = 1$  borrow 1

The rules of the decimal base applies to binary as well. To be able to calculate  $0-1$ , we have to “borrow one” from the next left digit.

$$\begin{array}{r} 12 \\ 0202 \\ 1010 \\ -0111 \\ \hline 0011 \end{array}$$

# Binary Subtraction

- ▶ 1's Complement Method
- ▶ 2's Complement Method

## 1's Complement Method

Example:  $1010100 - 1000100$

1's complement of  $1000100$  is  $0111011$

$$\begin{array}{r}
 1010100 \\
 + 0111011 \\
 \hline
 10001111 \\
 \hline
 0010000
 \end{array}$$

+1

If Carry, result is positive.  
Add carry to the partial result

Example:  $1000100 - 1010100$

1's complement of  $1010100$  is  $0101011$

$$\begin{array}{r}
 1000100 \\
 + 0101011 \\
 \hline
 1101111 \\
 = -0010000
 \end{array}$$

If no Carry, result is negative.  
Magnitude is 1's complement of the result

# Binary Subtraction

- ▶ 1's Complement Method
- ▶ 2's Complement Method

## 2's Complement Method

Example:  $1010100 - 1000100$

2's complement of  $1000100$  is  $0111100$

If Carry, result is positive.  
Discard the carry

$$\begin{array}{r} 1010100 \\ + 0111100 \\ \hline 10010000 \end{array}$$

$$0010000$$

Example:  $1000100 - 1010100$

2's complement of  $1010100$  is  $0101100$

If no Carry, result is negative.  
Magnitude is 2's complement of the result

$$\begin{array}{r} 1000100 \\ + 0101100 \\ \hline 1110000 \\ = -0010000 \end{array}$$

# Signed Binary Numbers

- ▶ When a signed binary number is positive
  - The MSB is '0' which is the sign bit and rest bits represents the magnitude
- ▶ When a signed binary number is negative
  - The MSB is '1' which is the sign bit and rest of the bits may be represented by three different ways
    - ❖ Signed magnitude representation
    - ❖ Signed 1's complement representation
    - ❖ Signed 2's complement representation

# Signed Binary Numbers

	<u>-9</u>	<u>+9</u>
Signed magnitude representation	1 1001	0 1001
Signed 1's complement representation	1 0110	0 1001
Signed 2's complement representation	1 0111	0 1001
	<u>-0</u>	<u>+0</u>
Signed magnitude representation	1 0000	0 0000
Signed 1's complement representation	1 1111	0 0000
Signed 2's complement representation	-None-	0 0000

# Range of Binary Number

## Binary Number of n bits

- ▶ General binary number:  $(2^n - 1)$
- ▶ Signed magnitude binary number:  $-(2^{n-1} - 1)$  to  $+(2^{n-1} - 1)$
- ▶ Signed 1's complement binary number:  $-(2^{n-1} - 1)$  to  $+(2^{n-1} - 1)$
- ▶ Signed 2's complement binary number:  $-(2^{n-1})$  to  $+(2^{n-1} - 1)$

# Signed Binary Number Arithmetic

- ▶ Add or Subtract two signed binary number including its sign bit either signed 1's complement method or signed 2's complement method
- ▶ The 1's complement and 2's complement rules of general binary number is applicable to this
- It is important to decide how many bits we will use to represent the number
- Example: Representing +5 and -5 on 8 bits:
  - +5: 00000101
  - -5: 10000101
- *So, the very first step we have to decide on the number of bits to represent number*

# Decimal Subtraction

- ▶ 9's Complement Method
- ▶ 10's Complement Method

## 9's Complement Method

Example:  $72532 - 3250$

9's complement of 03250 is

$$99999 - 03250 = 96749$$

If Carry, result is positive.  
Add carry to the partial result

$$\begin{array}{r} 72532 \\ + 96749 \\ \hline 169281 \\ \phantom{1} \rightarrow +1 \\ \hline 69282 \end{array}$$

Example:  $3250 - 72532$

9's complement of 72532 is

$$99999 - 72532 = 27467$$

If no Carry, result is negative.  
Magnitude is 9's complement of the result

$$\begin{array}{r} 03250 \\ + 27467 \\ \hline 30717 \\ \rightarrow \\ = -69282 \end{array}$$

# Decimal Subtraction

- ▶ 9's Complement Method
- ▶ 10's Complement Method

Example:  $72532 - 3250$

10's complement of 03250 is

$$100000 - 03250 = 96750$$

## 10's Complement Method

If Carry, result is positive.  
Discard the carry

$$\begin{array}{r} 72532 \\ + 96750 \\ \hline \end{array}$$

$$\mathbf{1}69282$$

Result is 69282

Example:  $3250 - 72532$

10's complement of 72532 is

$$100000 - 72532 = 27468$$

If no Carry, result is negative.  
Magnitude is 10's complement of the result

$$\begin{array}{r} 03250 \\ + 27468 \\ \hline 30718 \\ = -69282 \end{array}$$

# BCD Addition Rules

## BCD addition

Add two numbers as same as binary addition

Case 1: If the result is less than or equals to 9 and carry is zero then it is valid BCD.

Case 2: If result is greater than 9 and carry is zero then add 6 in four bit combination.

Case 3: If result is less than or equals to 9 but carry is 1 then add 6 in four bit combination.

BCD	1	1		
	0001	1000	0100	184
	+0101	0111	0110	+576
Binary sum	0111	10000	1010	
Add 6		0110	0110	
BCD sum	0111	0110	0000	760

# Comparing Binary and BCD Sums

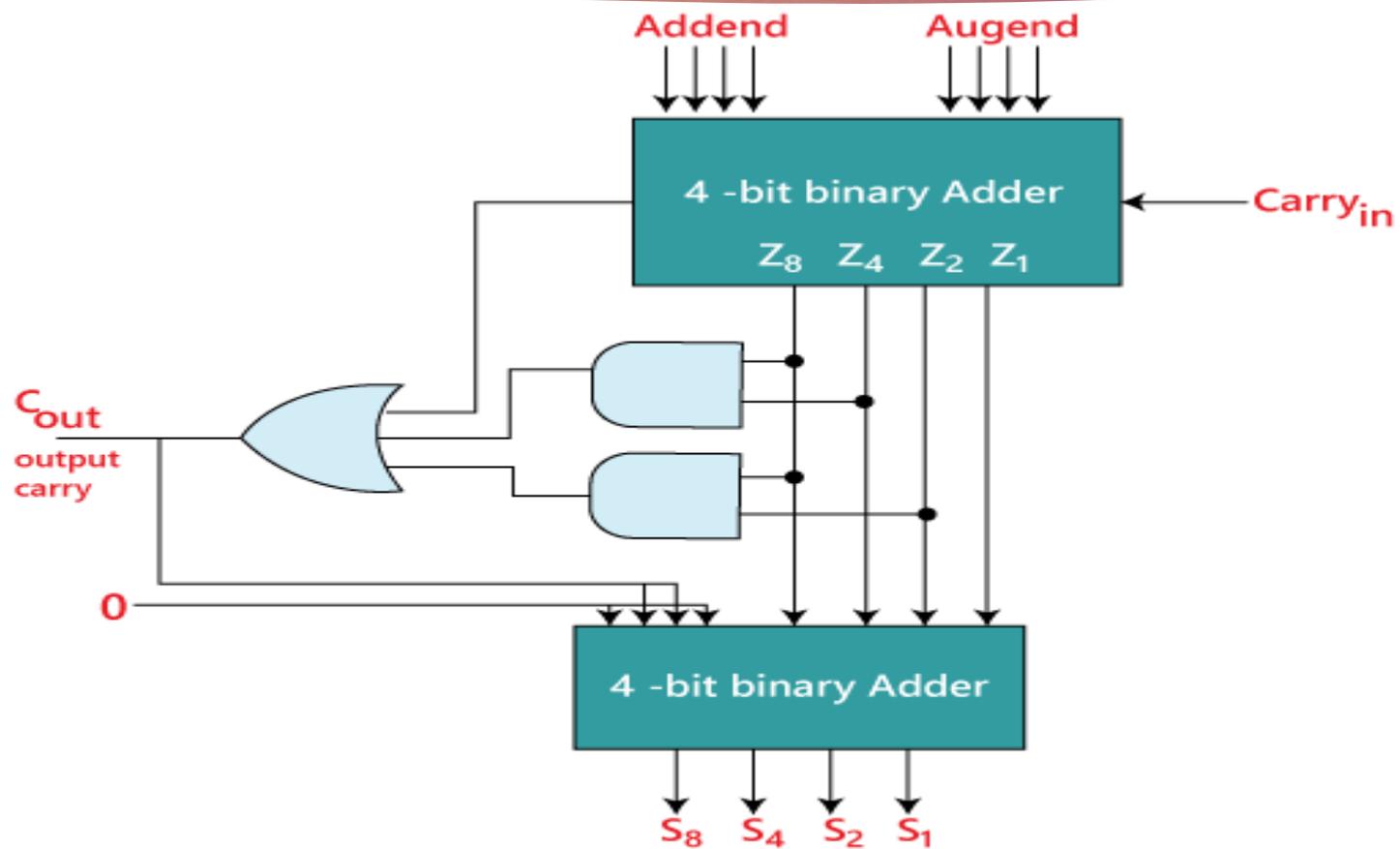
Binary Sum						BCD Sum						Decimal
K	Z <sub>8</sub>	Z <sub>4</sub>	Z <sub>2</sub>	Z <sub>1</sub>		C	S <sub>8</sub>	S <sub>4</sub>	S <sub>2</sub>	S <sub>1</sub>		
0	0	0	0	0	<b>S A M E  C O D E</b>	0	0	0	0	0		0
0	0	0	0	1		0	0	0	0	1		1
0	0	0	1	0		0	0	0	1	0		2
.	.	.	.	.		.	.	.	.	.		.
.	.	.	.	.		.	.	.	.	.		.
.	.	.	.	.		.	.	.	.	.		.
.	.	.	.	.		.	.	.	.	.		.
0	1	0	0	0		0	1	0	0	0		8
0	1	0	0	1		0	1	0	0	1		9
<b>10 to 19 Binary and BCD codes are not the same</b>												
0	1	0	1	0		1	0	0	0	0		10
0	1	0	1	1		1	0	0	0	1		11
0	1	1	0	0		1	0	0	1	0		12
0	1	1	0	1		1	0	0	1	1		13
0	1	1	1	0		1	0	1	0	0		14
0	1	1	1	1		1	0	1	0	1		15
1	0	0	0	0		1	0	1	1	0		16
1	0	0	0	1		1	0	1	1	1		17
1	0	0	1	0		1	1	0	0	0		18
1	0	0	1	1		1	1	0	0	1		19

# BCD Adder

- ▶ In the previous table Decimal sum from **0 to 9**, the Binary sum same as BCD sum. So, no conversion is needed.
- ▶ Apply correction if the Decimal sum is between **10-19**.
  - ❖ The correction is needed (Decimal sum **16-19**) when the binary sum has an output carry  $K = 1$
  - ❖ The correction is needed (Decimal sum **10-15**) when  $Z_8 = 1$  and either  $Z_4 = 1$  or  $Z_2 = 1$ .
- ▶ So, the condition for a correction and an output carry can be expressed by the Boolean function:

$$C = K + Z_8Z_4 + Z_8Z_2$$

- ▶ When  $C = 1$ , it is necessary to add 0110 to the binary sum to get BCD sum and provide an output carry for the next stage.



Block diagram of a BCD adder

# Cascading of BCD Adders

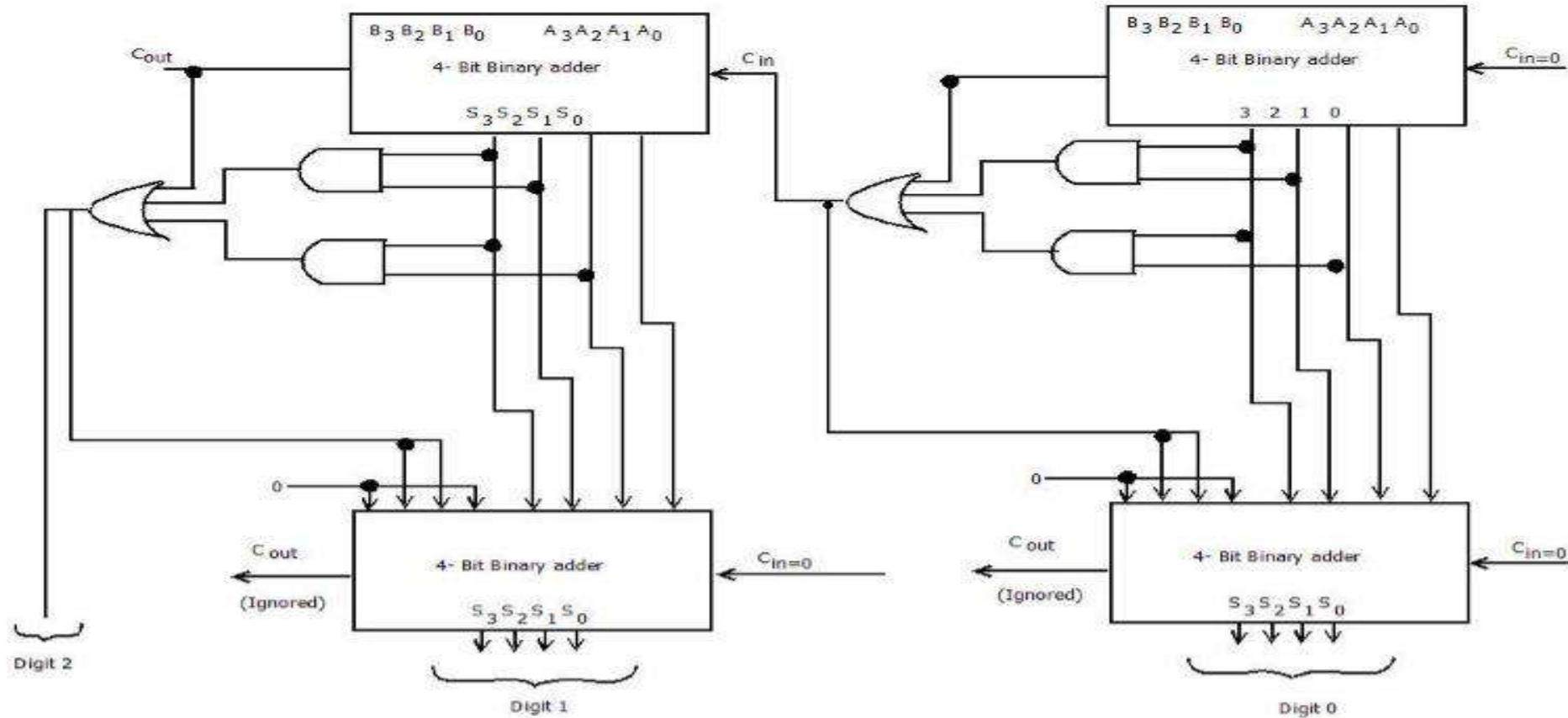


Fig : 8 - Bit BCD Adder

# BCD Subtraction Rules

Let two BCD numbers are A and B.  
B to be subtracted from A.

## RULES:

- Add 9's Complement of B to A
- If result  $> 9$ , Correct by adding 0110
- If carry is generated at most significant position then the result is positive and the End around carry must be added
- If carry is not generated at most significant position then the result is negative and the result is 9's complement of original result

BCD number (d)	Binary equivalent of BCD number				9's complement of BCD (9 - d)	Binary equivalent of 9's complement number			
	w	x	y	z		C <sub>3</sub>	C <sub>2</sub>	C <sub>1</sub>	C <sub>0</sub>
0	0	0	0	0	9	1	0	0	1
1	0	0	0	1	8	1	0	0	0
2	0	0	1	0	7	0	1	1	1
3	0	0	1	1	6	0	1	1	0
4	0	1	0	0	5	0	1	0	1
5	0	1	0	1	4	0	1	0	0
6	0	1	1	0	3	0	0	1	1
7	0	1	1	1	2	0	0	1	0
8	1	0	0	0	1	0	0	0	1
9	1	0	0	1	0	0	0	0	0

# Example

## Regular Subtraction

(a)

$$\begin{array}{r} 8 \\ - 2 \\ \hline 6 \end{array}$$

(b)

$$\begin{array}{r} 28 \\ - 13 \\ \hline 15 \end{array}$$

(c)

$$\begin{array}{r} 18 \\ - 24 \\ \hline -6 \end{array}$$

## 9's Complement Subtraction

(a)

$$\begin{array}{r} 8 \\ + 7 \text{ 9's complement of 2} \\ \hline 5 \\ \text{\textcircled{1}} \downarrow + 1 \text{ Add carry to result} \\ \hline 6 \end{array}$$

(b)

$$\begin{array}{r} 28 \\ + 86 \text{ 9's complement of 13} \\ \hline 14 \\ \text{\textcircled{1}} \downarrow + 1 \text{ Add carry to the result} \\ \hline 15 \end{array}$$

(c)

$$\begin{array}{r} 18 \\ + 75 \text{ 9's complement of 24} \\ \hline 93 \text{ 9's complement of result} \\ \downarrow \text{(No carry indicates that the} \\ \text{answer is negative and in} \\ \text{complement form)} \\ -06 \end{array}$$

E.G.  $8 - 3 = 8 + [9\text{'s COMP. OF } 3]$   
 $= 8 + 6$

$$\begin{array}{r} 1000 \\ 0110 \\ \hline 1110 \leftarrow \text{INVALID } (>9) \\ 0110 \leftarrow \text{CORRECTION} \\ \text{\textcircled{1}} 0100 \\ \downarrow \text{\textcircled{1}} \leftarrow \text{END AROUND CARRY} \\ \hline 0101 = 5 \end{array}$$

(b)  $3 - 8 = -5$

$$\begin{array}{r} 0011 \\ 0001 \\ \hline 0100 \end{array}$$

**NO CARRY >>> NEGATIVE**  
 9's COMP. OF 0100 = 0101 = -5

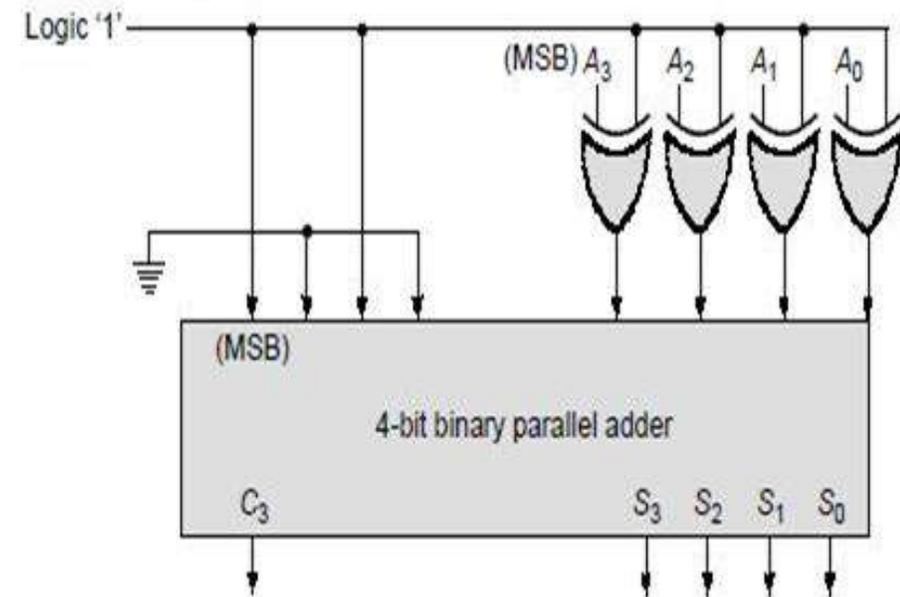
(c)  $87 - 39 >>> 87 + [9\text{'s COMP OF } 39]$

$$\begin{array}{r} 87 \quad 1000 \quad 0111 \\ 60 \quad 0110 \quad 0000 \\ \hline \text{INVALID} \rightarrow 1110 \quad 0111 \\ 0110 \\ \text{\textcircled{1}} 0100 \quad 0111 \\ \downarrow \text{\textcircled{1}} \leftarrow \\ \hline 0100 \quad 1000 \\ = \quad 4 \quad 8 \end{array}$$

# 9's Complement Circuit

- 9's complement of 2 is 7
- Binary equivalent of 2 is 0010
- 1's complement of 0010 is 1101
- Then, 
$$\begin{array}{r} 1101 \\ + 1010 \\ \hline = 0111 \end{array}$$
 which is Binary equivalent of 7
- If carry discard it.

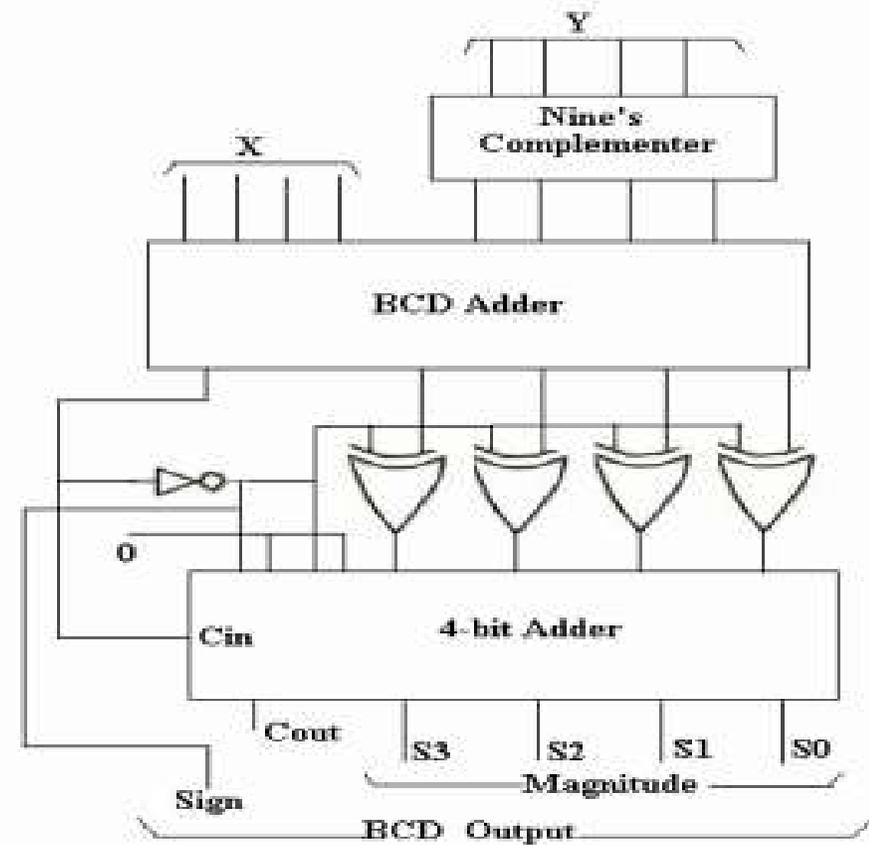
- 9's complement of 3 is 6
- Binary equivalent of 3 is 0011
- 1's complement of 0011 is 1100
- Then, 
$$\begin{array}{r} 1100 \\ + 1010 \\ \hline = 0110 \end{array}$$
 which is Binary equivalent of 6
- **If carry discard it.**



# BCD Subtractor Circuit

## RULES:

- Add 9's Complement of B to A
- If result  $> 9$ , Correct by adding 0110
- If carry is generated at most significant position then the result is positive and the End around carry must be added
- If carry is not generated at most significant position then the result is negative and the result is 9's complement of original result



# Floating Point Number

- ▶ Floating point number can be represented as

$$m \times r^e$$

- ▶  $m$  is mantissa,  $e$  is exponent and  $r$  is radix
- ▶ Let the decimal number is 6132.789, which can be represented as  $0.6132789 \times 10^4$
- ▶ Let the binary number is 1001.110, which can be represented as  $0.1001110 \times 2^4$  or can be represented as  $1.001110 \times 2^3$

# Floating Point Arithmetic

## ► Addition/Subtraction

- Align the radix point first to make the exponent equal before addition or subtraction
- Add or Subtract mantissa
- Normalize the result by adjusting the exponent
- $(A \times E^n) \pm (B \times E^n) = (A \pm B) E^n$

## ► Multiplication

- $(A \times E^m) \times (B \times E^n) = (A \times B) E^{m+n}$

## ► Division

- $(A \times E^m) \div (B \times E^n) = (A \div B) E^{m-n}$

# Addition (Decimal FP)

- Consider a 4-digit decimal example
  - $9.999 \times 10^1 + 1.610 \times 10^{-1}$
- **1. Align decimal points**
  - Adjust exponent
  - $9.999 \times 10^1 + 0.016 \times 10^1$
- **2. Add mantissa**
  - $9.999 \times 10^1 + 0.016 \times 10^1 = 10.015 \times 10^1$
- **3. Normalize result & check for over/underflow**
  - $1.0015 \times 10^2$
- **4. Round and renormalize if necessary**
  - $1.002 \times 10^2$

# Addition (Binary FP)

- Now consider a 4-digit binary example
  - $1.000_2 \times 2^{-1} + -1.110_2 \times 2^{-2}$  ( $0.5 + -0.4375$ )
- **1. Align binary points**
  - Adjust exponent
  - $1.000_2 \times 2^{-1} + -0.111_2 \times 2^{-1}$
- **2. Add mantissa**
  - $1.000_2 \times 2^{-1} + -0.111_2 \times 2^{-1} = 0.001_2 \times 2^{-1}$
- **3. Normalize result & check for over/underflow**
  - $1.000_2 \times 2^{-4}$
- **4. Round and renormalize if necessary**
  - $1.000_2 \times 2^{-4}$  (no change) = 0.0625

# Multiplication (Decimal FP)

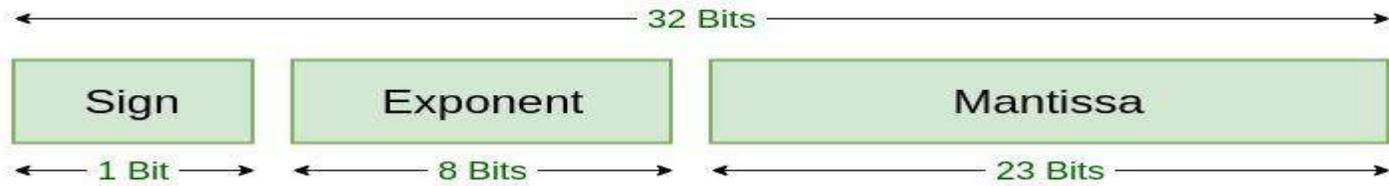
- Consider a 4-digit decimal example
  - $1.110 \times 10^{10} \times 9.200 \times 10^{-5}$
- **1. Add exponents**
  - For biased exponents, subtract bias from sum
  - New exponent =  $10 + -5 = 5$
- **2. Multiply mantissa**
  - $1.110 \times 9.200 = 10.212 \Rightarrow 10.212 \times 10^5$
- **3. Normalize result & check for over/underflow**
  - $1.0212 \times 10^6$
- **4. Round and renormalize if necessary**
  - $1.021 \times 10^6$
- **5. Determine sign of result from signs of operands**
  - $+1.021 \times 10^6$

# Multiplication (Binary FP)

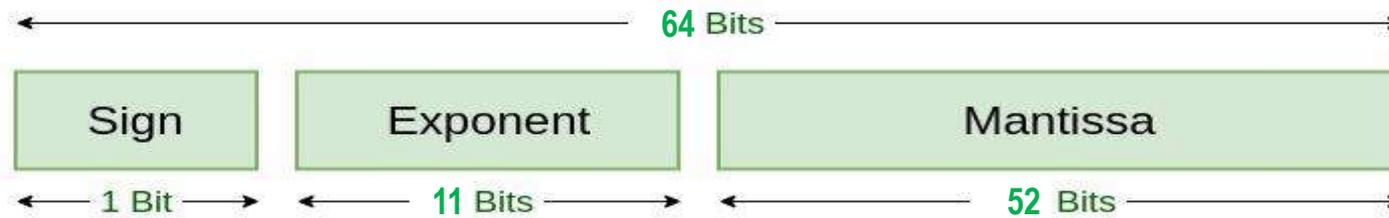
- Now consider a 4-digit binary example
  - $1.000_2 \times 2^{-1} \times -1.110_2 \times 2^{-2}$  ( $0.5 \times -0.4375$ )
- **1. Add exponents**
  - Unbiased:  $-1 + -2 = -3$
  - Biased:  $(-1 + -2 + 127) = -3 + 127$
- **2. Multiply mantissa**
  - $1.000_2 \times 1.110_2 = 1.110_2 \Rightarrow 1.110_2 \times 2^{-3}$
- **3. Normalize result & check for over/underflow**
  - $1.110_2 \times 2^{-3}$
- **4. Round and renormalize if necessary**
  - $1.110_2 \times 2^{-3}$  (no change)
- **5. Determine sign: + ve  $\times$  - ve = -ve**
  - $-1.110_2 \times 2^{-3} = -0.21875$

# Floating Point Standard

- The IEEE Standard for Floating-Point (IEEE 754) is a technical standard for floating-point representation which was defined in 1985 by the **Institute of Electrical and Electronics Engineers (IEEE)**.
- Developed in response to divergence of representations
  - Portability issues for scientific code
- Now almost universally adopted
- Two representations
  - Single precision (32-bit)
  - Double precision (64-bit)



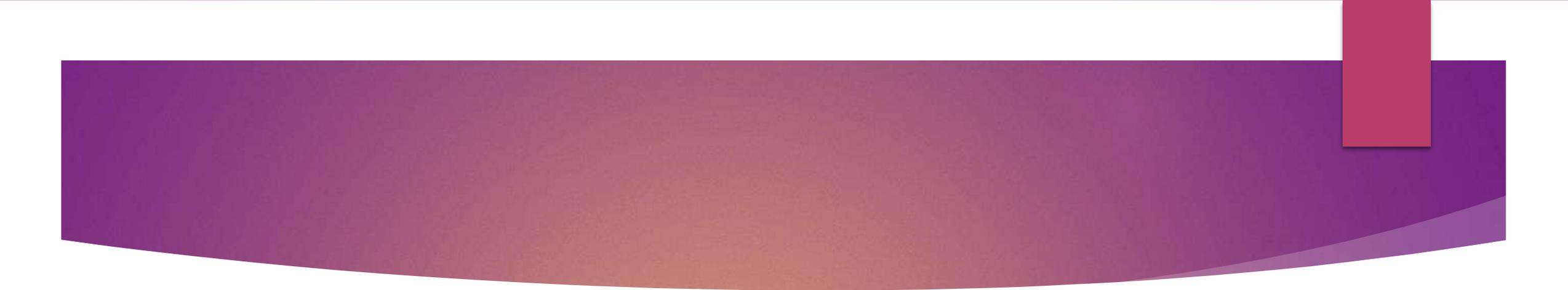
Single Precision  
IEEE 754 Floating-Point Standard



Double Precision  
IEEE 754 Floating-Point Standard

- Normalize significand:  $1.0 \leq |\text{significand}| < 2.0$ 
  - Always has a leading pre-binary-point 1 bit, so no need to represent it explicitly (hidden bit)
  - Significand is Fraction with the “1.” restored
- Exponent: excess representation: actual exponent + Bias
  - Ensures exponent is unsigned
  - Single precision: Bias = 127
  - Double precision: Bias = 1023

- 
- ▶ In the CPU, a 32-bit floating point number is represented using IEEE single precision standard format as follows:
  - ▶ S | EXPONENT | MANTISSA
  - ▶ where S is one bit, the EXPONENT is 8 bits, and the MANTISSA is 23 bits.
  - The *mantissa* represents the leading significant bits in the number.
  - The *exponent* is used to adjust the position of the binary point (like "decimal" point)
  - ▶ The mantissa is said to be normalized when it is expressed as a value between 1 and 2. i.e., the mantissa would be in the form 1.xxxx.

- 
- ▶ The leading integer of the binary representation is not stored. Since it is always a 1, it can be easily restored
  - ▶ The "S" bit is used as a sign bit and indicates whether the value represented is positive or negative
  - ▶ 0 for positive, 1 for negative
  - ▶ If a number is smaller than 1, normalizing the mantissa will produce a negative exponent
  - ▶ But 127 is added to all exponents in the floating point representation, allowing all exponents to be represented by a positive number

# Single Precision

- ▶ **Example 1.** Represent the decimal value 2.5 in 32-bit floating point format.

$$2.5 = 10.1b$$

- ▶ In normalized form, this is:  $1.01 \times 2^1$
- ▶ The mantissa:  $M = 010000000000000000000000$   
(23 bits without the leading 1)
- ▶ The exponent:  $E = 1 + 127 = 128 = 10000000b$
- ▶ The sign:  $S = 0$  (the value stored is positive)
- ▶ So,  $2.5 =$  **0** **10000000** **010000000000000000000000**  
**Sign** **Exponent** **Mantissa**

▶ **Example 2:** Represent the number - 0.00010011b in floating point form.

▶  $0.00010011b = 1.0011 \times 2^{-4}$  in normalized form

▶ Mantissa:  $M = 001100000000000000000000$

▶ Exponent:  $E = -4 + 127 = 123 = 01111011b$

▶  $S = 1$  (as the number is negative)

▶ Result: **1 01111011 001100000000000000000000**

**Sign Exponent Mantissa**

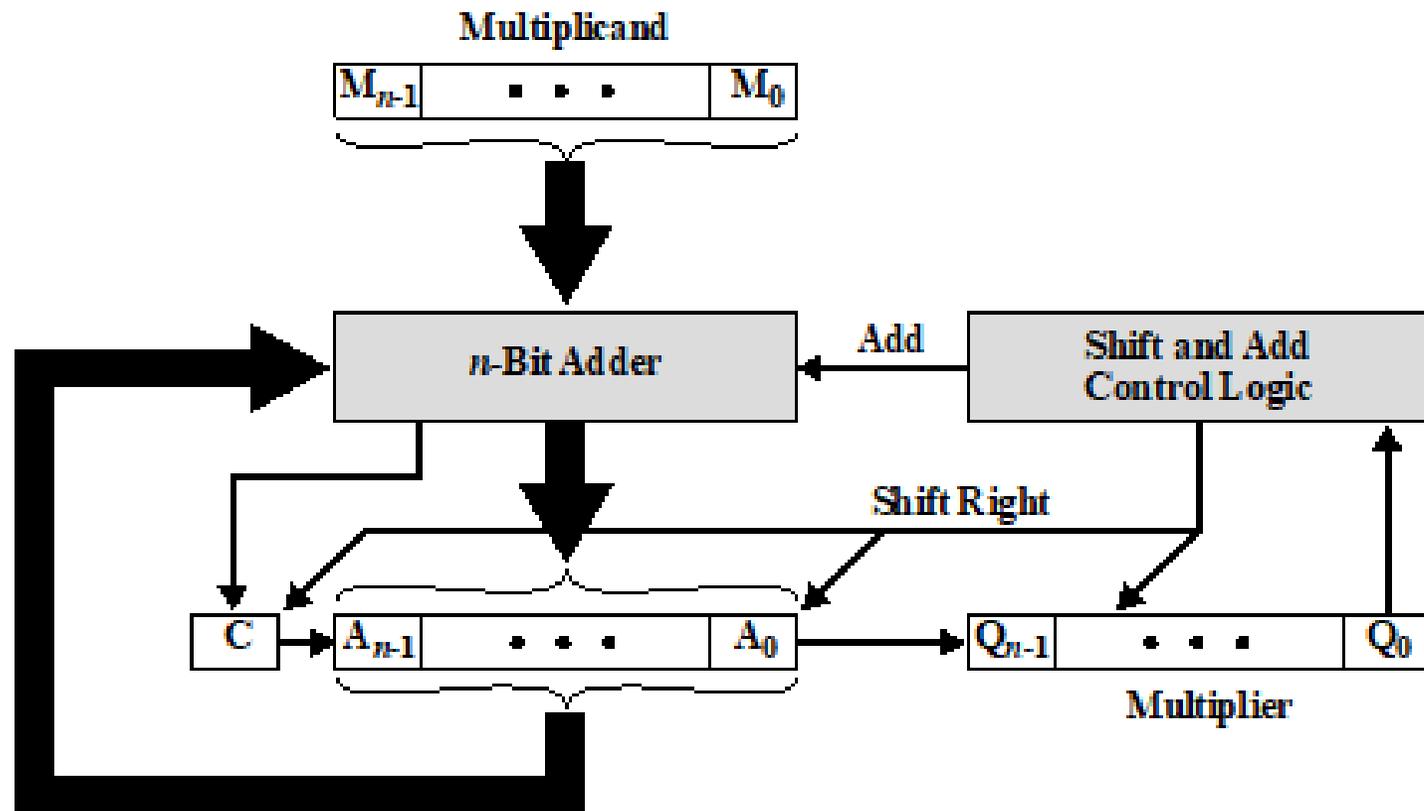


# Multiplication

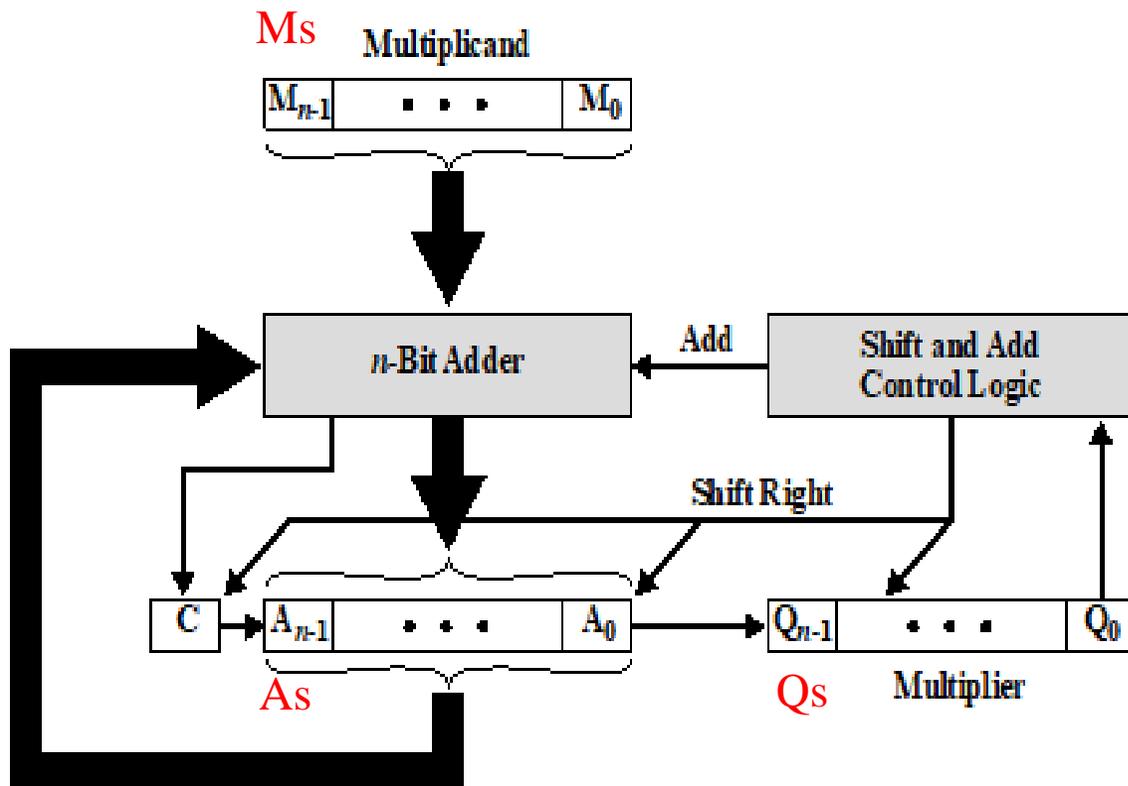
1011	<b>Multiplicand (11)</b>
× 1101	<b>Multiplier (13)</b>
<hr/>	
1011	} <b>Partial products</b>
0000	
1011	
1011	
<hr/>	
10001111	<b>Product (143)</b>

**Multiplication of Unsigned Binary Integers**

# Hardware Diagram



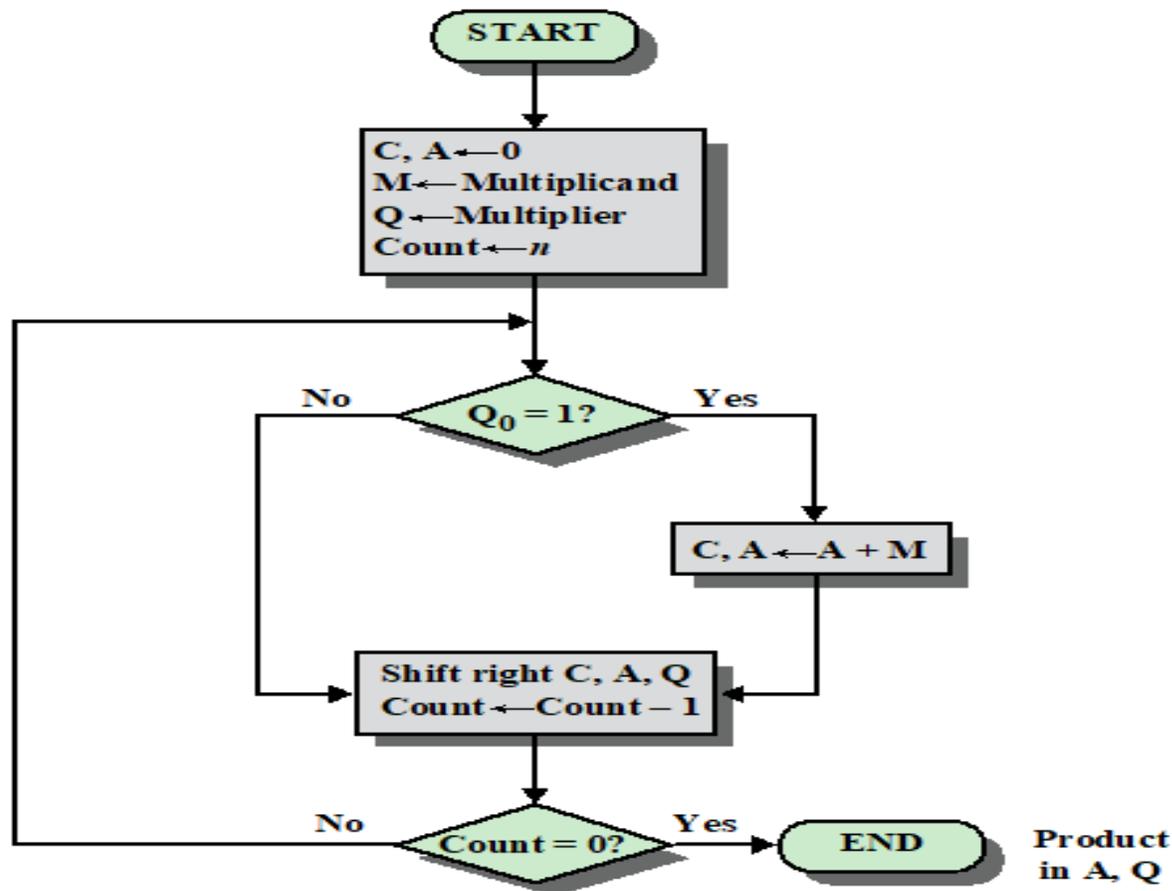
# Hardware Diagram



The sign of the product is determined from the signs of the Multiplicand and Multiplier.

- If they are alike, Sign of the product is Positive.
- If they are unlike, Sign of the product is Negative
- So,  $As$  will be equal to  $Ms$  EX-OR with  $Qs$

# General Multiplication



C	A	Q	M		
0	0000	1101	1011	Initial Values	
0	1011	1101	1011	Add	} First Cycle
0	0101	1110	1011	Shift	
0	0010	1111	1011	Shift	} Second Cycle
0	1101	1111	1011	Add	} Third Cycle
0	0110	1111	1011	Shift	
1	0001	1111	1011	Add	} Fourth Cycle
0	1000	1111	1011	Shift	

# Booth Multiplication

- ▶ **Booth's multiplication algorithm** is a multiplication algorithm that multiplies two signed binary numbers in 2's complement notation.
- ▶ The algorithm was invented by Andrew Donald Booth in 1950.
- ▶ It is used to speed up the performance of the multiplication process. It is very efficient too.
- ▶ If string of 0's or string of 1's are there in the multiplier that requires no operation only shift.
- ▶ Consider a general multiplier consisting of a block of 1s surrounded by 0s. For example, 00111110. The product is given by:  
$$M \times 00111110 = M \times (2^5 + 2^4 + 2^3 + 2^2 + 2^1) = M \times 62$$
 where, M is the multiplicand
- ▶ The number of operations can be reduced to two by rewriting the same as  
$$M \times 00111110 = M \times (2^6 - 2^1) = M \times 62$$
- ▶ This one is Booth Multiplication.

# Example

- ▶ Let the multiplication is  $M \times +14$

In signed 2's complement representation  $+14 = 0\ 000\ 1110$

Which is  $M \times 0\ 000\ 1110 = M \times (2^4 - 2^1) = M \times (16 - 2) = M \times +14$

- ▶ Let the multiplication is  $M \times -14$

In signed 2's complement representation  $-14 = 1\ 111\ 0010$

Which is  $M \times 1\ 111\ 0010 = M \times (-2^4 + 2^2 - 2^1) = M \times (-16 + 4 - 2) = M \times -14$

# Algorithm

- ▶ As in all multiplication schemes, Booth algorithm also requires examination **of the multiplier bits** from LSB to MSB and shifting of the partial product.
- ▶ Prior to the shifting, the multiplicand may be added to the partial product, subtracted from the partial product, or left unchanged according to following rules:
  - The multiplicand is subtracted from the partial product upon encountering the first least significant 1 in a string of 1's in the multiplier
  - The multiplicand is added to the partial product upon encountering the first 0 (provided that there is a previous '1') in a string of 0's in the multiplier.
  - The partial product does not change when the multiplier bit is identical to the previous multiplier bit, that is string of 0s or string of 1s.

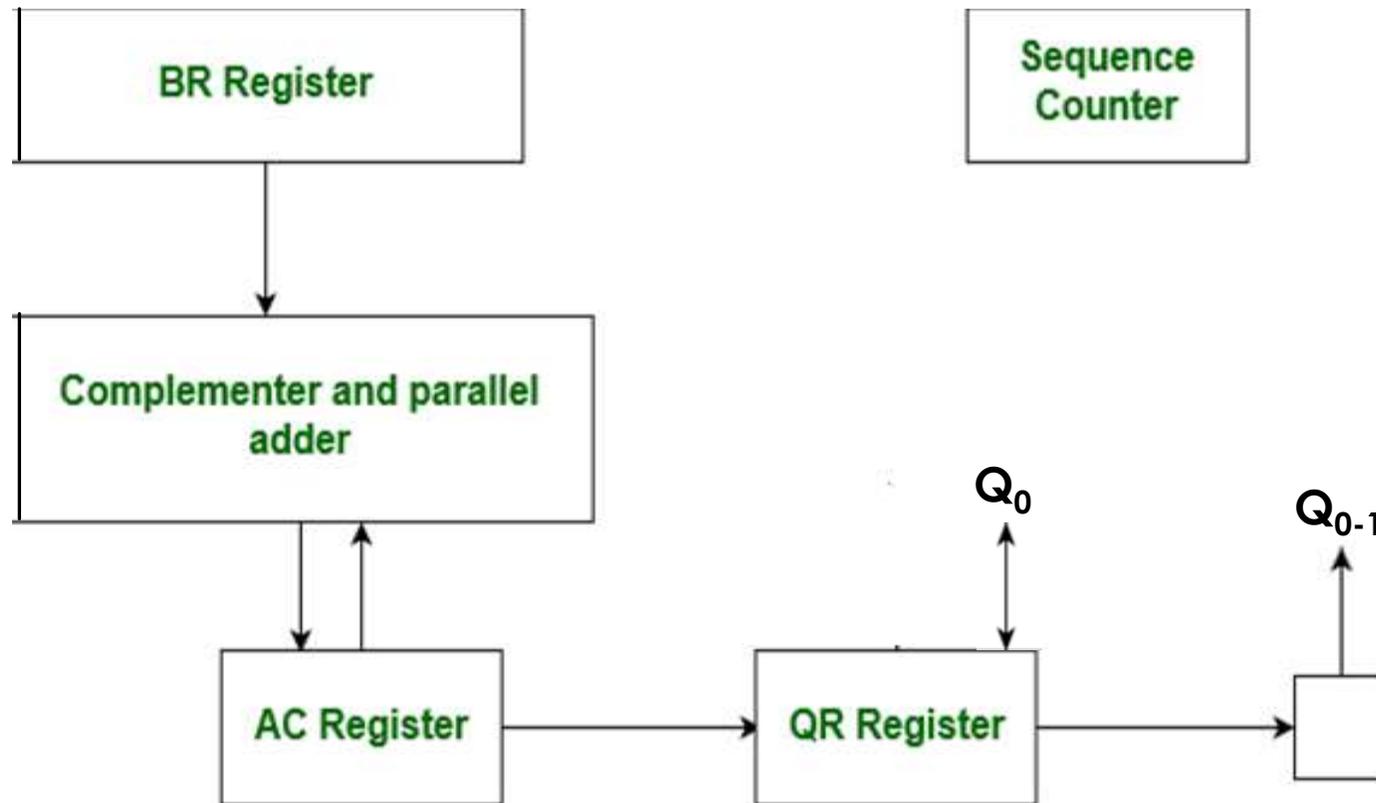
# Arithmetic Shift Right

- ▶ In Booth Multiplication Algorithm Shift Right is **Arithmetic shift right**

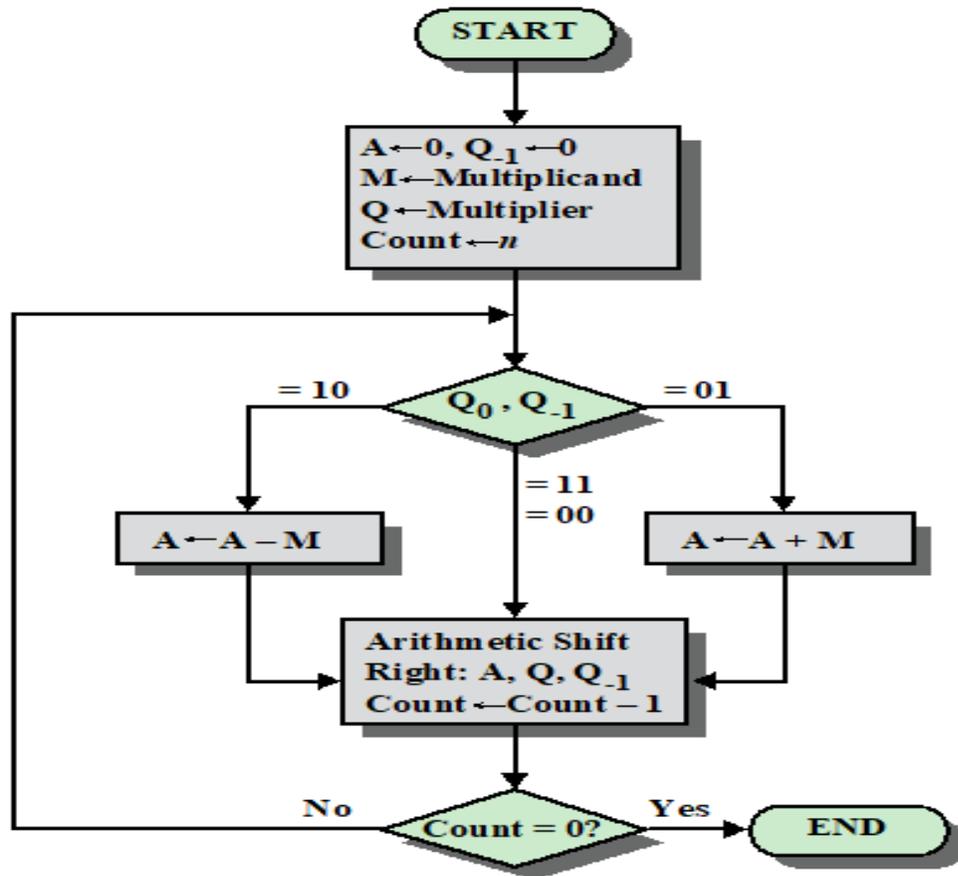
## Example:

- ▶ Let the number is 1001
- ▶ Shift right is 0100
- ▶ But, Arithmetic shift right is 1100
- ▶ Let the number is 0101
- ▶ Arithmetic shift right of this number is 0010

# Hardware Diagram



# Booth Multiplication Algorithm

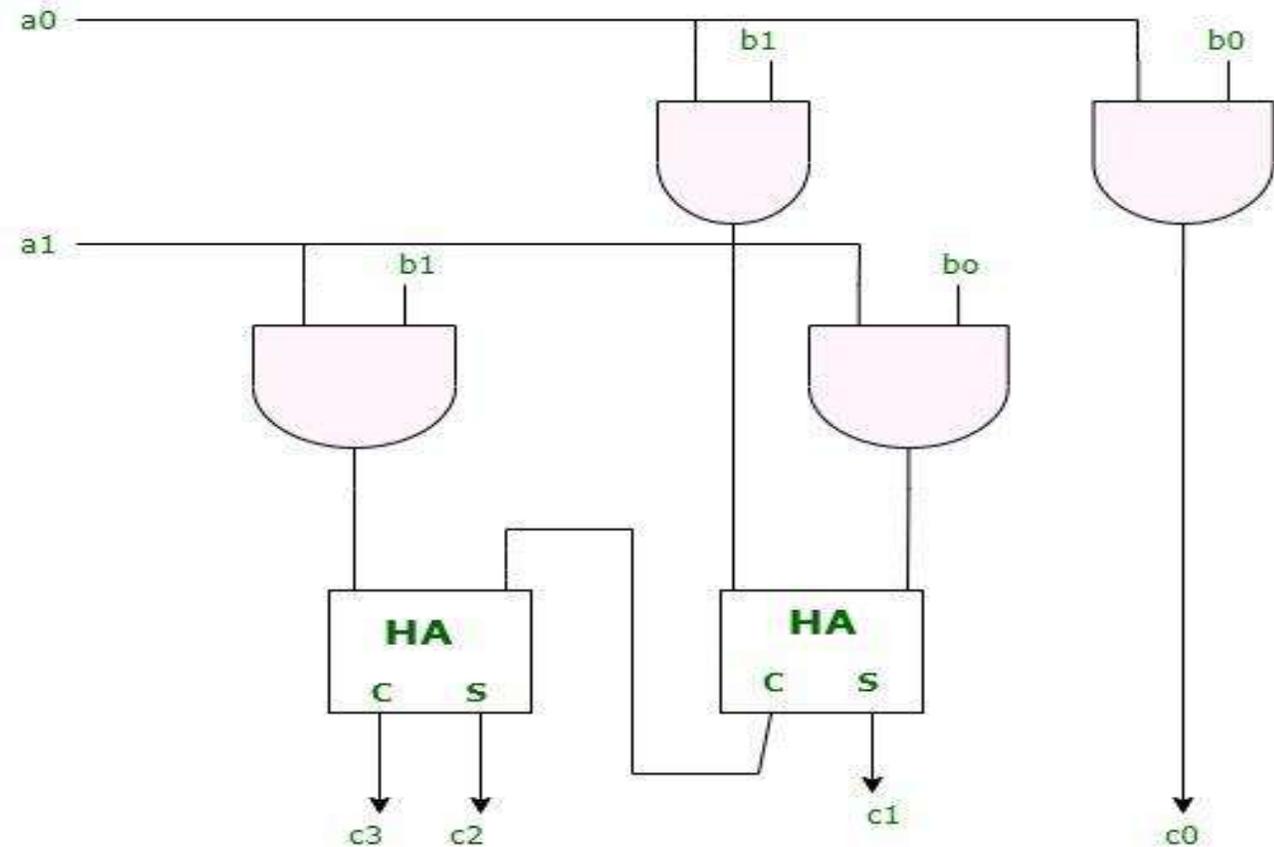


A	Q	Q <sub>-1</sub>	M	
0000	0011	0	0111	Initial Values
1001	0011	0	0111	A ← A - M } First Cycle
1100	1001	1	0111	
1110	0100	1	0111	Shift } Second Cycle
0101	0100	1	0111	
0010	1010	0	0111	Shift
0001	0101	0	0111	Shift } Fourth Cycle

**Example of Booth's Algorithm (7× 3)**

# Multiplier Design

$$\begin{array}{r} a1\ a0 \\ \times\ b1\ b0 \\ \hline a1b0\ a0b0 \\ a1b1\ a0b1 \\ \hline \end{array}$$



# Array Multiplier

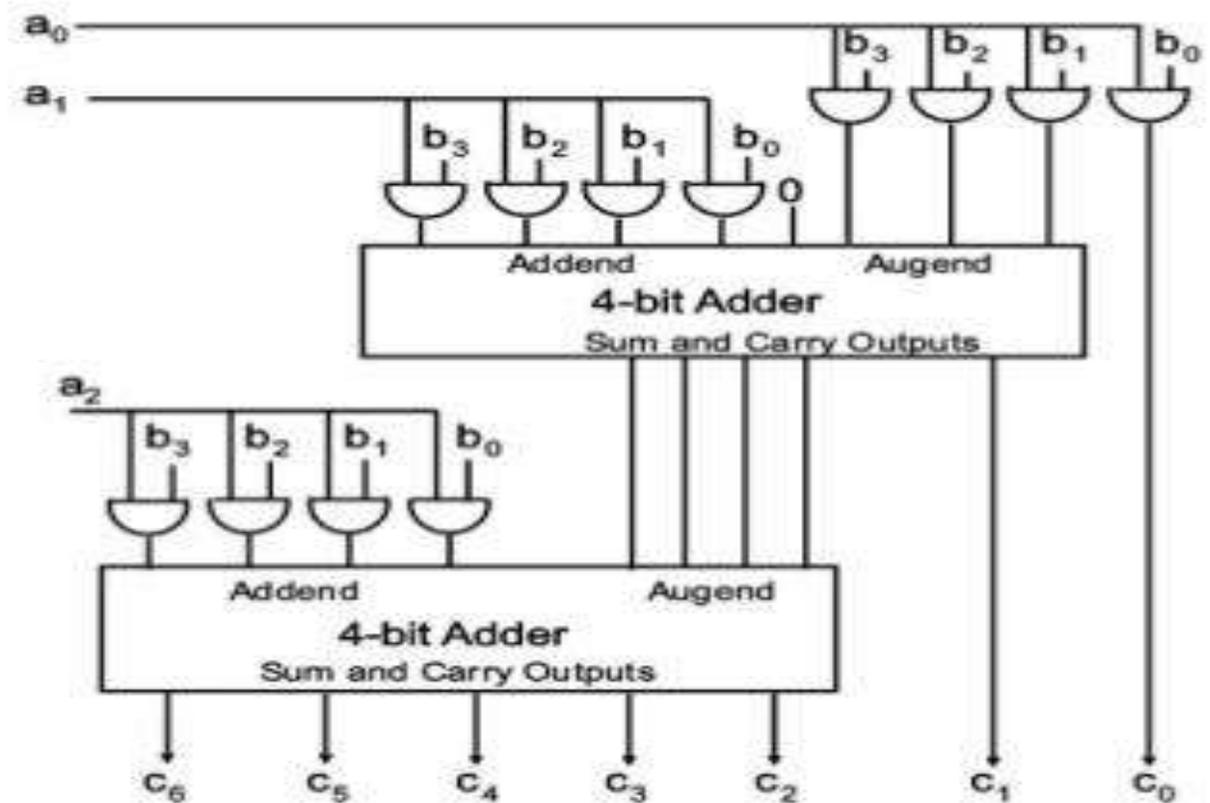
$$\begin{array}{r}
 \phantom{a_2} b_3 b_2 b_1 b_0 \\
 \times \phantom{a_2} a_2 a_1 a_0 \\
 \hline
 a_0 b_3 \ a_0 b_2 \ a_0 b_1 \ a_0 b_0 \\
 a_1 b_3 \ a_1 b_2 \ a_1 b_1 \ a_1 b_0 \\
 a_2 b_3 \ a_2 b_2 \ a_2 b_1 \ a_2 b_0 \\
 \hline
 c_6 \ c_5 \ c_4 \ c_3 \ c_2 \ c_1 \ c_0
 \end{array}$$

J = Multiplicand

K = Multiplier

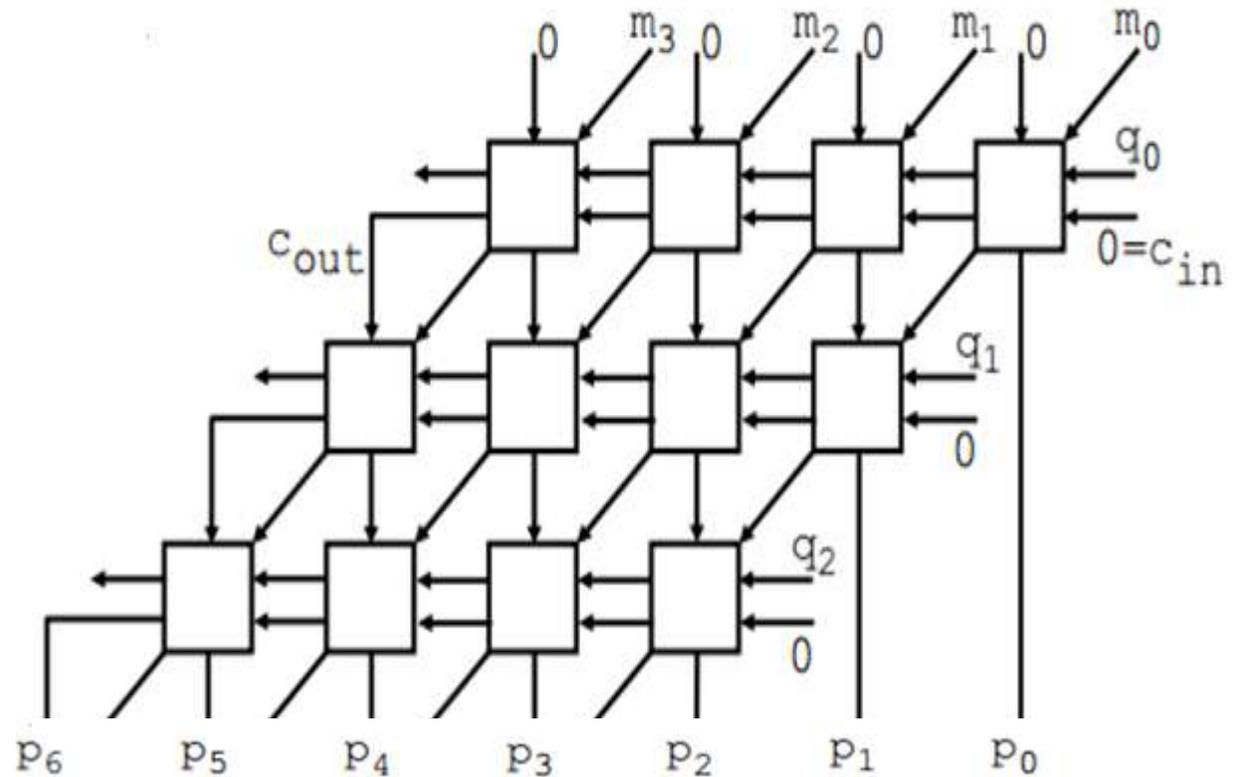
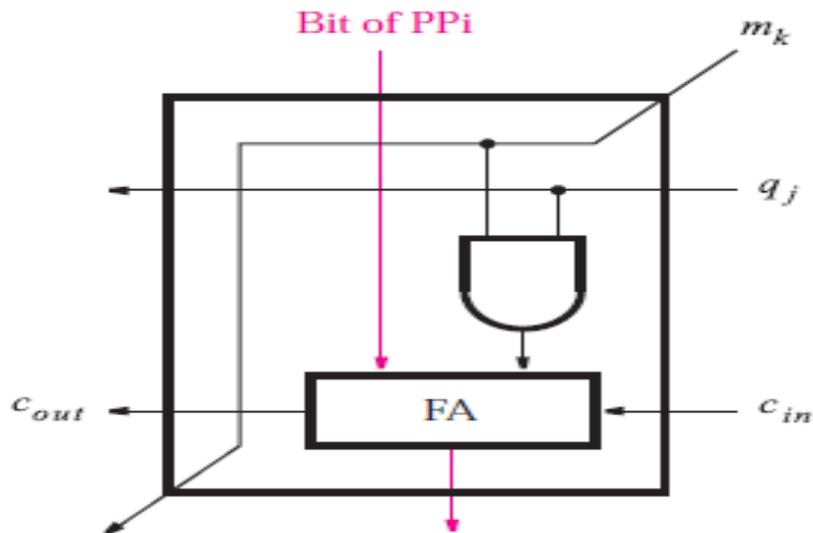
AND gate required = JK nos.

(K-1) nos. of J-bit Adder required



# Array Multiplier

$$\begin{array}{r}
 m_3 \ m_2 \ m_1 \ m_0 \\
 \times \quad q_2 \ q_1 \ q_0 \\
 \hline
 m_3q_0 \ m_2q_0 \ m_1q_0 \ m_0q_0 \\
 m_3q_1 \ m_2q_1 \ m_1q_1 \ m_0q_1 \\
 m_3q_2 \ m_2q_2 \ m_1q_2 \ m_0q_2 \\
 \hline
 P_6 \ P_5 \ P_4 \ P_3 \ P_2 \ P_1 \ P_0
 \end{array}$$



# Lecture of Module 2

## **Instruction Set Architecture**

# Overview

- ▶ **Introduction**
- ▶ **Instruction Format**
- ▶ **Different types of Instruction format**
- ▶ **Addressing Modes**
- ▶ **Types of Instruction**
- ▶ **Instruction Cycle**

# Introduction

- ▶ A program is a set of instructions that specify the operations, operands and sequence of operations by which processing has to occur.
- ▶ A computer instruction is a binary code that specifies a sequence of micro-operations.
- ▶ Instruction code together with data stored in the computer.
- ▶ The collection of different instructions that the processor can execute is referred to as the processor's *instruction set*.
- ▶ Every processor has its own specific Instruction Set.
- ▶ Each instruction must contain the information required by the processor for execution.
- ▶ The ability to store and execute instruction, the stored program concept is most important property of general purpose computer.
- ▶ The instruction code is a group of bits that instruct the computer to perform specific operation.

# Elements of Instruction

## Operation code (opcode)

- Specifies the operation to be performed. The operation is specified by a binary code, known as the operation code, or *opcode*

## Source operand reference

- The operation may involve one or more source operands, that is, operands that are inputs for the operation

## Result operand

- The operation may produce a result

## Next instruction reference

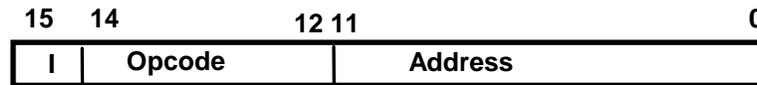
- This tells the processor where to fetch the next instruction after the execution of this instruction is complete

# General Instruction Format

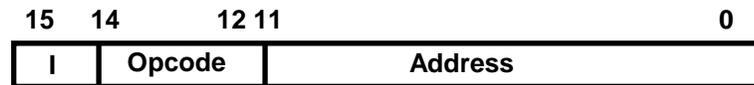


- ▶ General instruction format consists of two fields
  - Opcode
  - Operand
- ▶ The number of bits in opcode or operational code depends on the number of operations available in the Instruction set.
- ▶ When the Opcode decoded in the Control unit, it issues Control signals to read the operand and initiates micro-operation to perform complete operation of that instruction.
- ▶ The Operand/Address part may be the data or address of data or Register or I/O device etc.
- ▶ This also specifies where the result to be stored.
- ▶ The number of bits in operand field depends on the number of registers in that processor or address bits required to locate memory location.

# Example



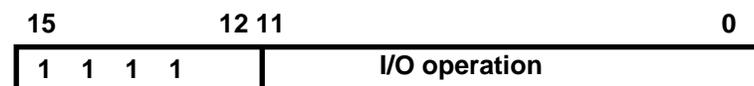
**Memory-Reference Instructions (OP-code = 000 ~ 110)**



**Register-Reference Instructions (OP-code = 111, I = 0)**



**Input-Output Instructions (OP-code = 111, I = 1)**



- ▶ Let the instruction code having 16 bits
- ▶ 12 bits for Operand/Address, 3 bits for Opcode and 1 bit for mode (I) bit
- ▶ If I = 0 Direct addressing memory reference instruction
- ▶ If I = 1 Indirect addressing memory reference instruction
- ▶ Opcode 111 are for Register or I/O reference instruction
- ▶ If I = 0, it is Register reference instruction
- ▶ If I = 1, it is I/O reference instruction

- ▶ Opcodes are represented by abbreviations called *mnemonics*
- ▶ Examples:
  - ▶ ADD      Add
  - ▶ SUB      Subtract
  - ▶ MUL      Multiply
  - ▶ DIV      Divide
  - ▶ LOAD     Load data from memory
  - ▶ STOR     Store data to memory
- ▶ Operands are also represented symbolically
- ▶ Each symbolic opcode has a fixed binary representation
  - ▶ The programmer specifies the location of each symbolic operand

<i>Symbol</i>	<i>Description</i>
<b>AND</b> <b>ADD</b> <b>LDA</b> <b>STA</b> <b>BUN</b> <b>BSA</b> <b>ISZ</b>	<b>AND</b> memory word to AC <b>Add</b> memory word to AC <b>Load</b> AC from memory <b>Store</b> content of AC into memory <b>Branch</b> unconditionally <b>Branch</b> and save return address <b>Increment</b> and skip if zero
<b>CLA</b> <b>CLE</b> <b>CMA</b> <b>CME</b> <b>CIR</b> <b>CIL</b> <b>INC</b> <b>SPA</b> <b>SNA</b> <b>SZA</b> <b>SZE</b> <b>HLT</b>	<b>Clear</b> AC <b>Clear</b> E <b>Complement</b> AC <b>Complement</b> E <b>Circulate</b> right AC and E <b>Circulate</b> left AC and E <b>Increment</b> AC <b>Skip</b> next instr. if AC is positive <b>Skip</b> next instr. if AC is negative <b>Skip</b> next instr. if AC is zero <b>Skip</b> next instr. if E is zero <b>Halt</b> computer
<b>INP</b> <b>OUT</b> <b>SKI</b> <b>SKO</b> <b>ION</b> <b>IOF</b>	<b>Input</b> character to AC <b>Output</b> character from AC <b>Skip</b> on input flag <b>Skip</b> on output flag <b>Interrupt</b> on <b>Interrupt</b> off

# Instruction Format

- ▶ A computer will usually have a variety of Instruction Code format.
- ▶ It is the function of Control unit within the CPU to interpret each instruction code and provide necessary control functions needed to process the instruction.
- ▶ The bits of instructions are divided into groups called fields.
- ▶ The most common fields found in instruction format are:
  - An Operation code field (Opcode)
  - Operand or Address fields
  - A mode field that specifies the way the operand or effective address of the operand is determined
  - Other special fields are sometimes employed under certain circumstances
- ▶ Computers may have instructions of several lengths containing varying number of address fields.
- ▶ The number of address field in the instruction format of a computer depends on the internal organization of its registers.

- ▶ Most of the computers fall into one of three types of CPU organization.
  - Single Accumulator organization
  - General Register organization
  - STACK organization

#### Single Accumulator organization

ADD X             $AC \leftarrow AC + M[X]$

#### General Register organization

ADD R1, R2, R3     $R1 \leftarrow R2 + R3$

ADD R1, R2         $R1 \leftarrow R1 + R2$

ADD R1, X          $R1 \leftarrow R1 + M[X]$

#### STACK organization

PUSH A

PUSH B

ADD

# Different types of Instruction format

- ▶ Some computers may follow one or more than one or all mentioned features in their structure.
- ▶ There are various types of Instruction format according to the number of Address fields.
  - Three-address instruction
  - Two-address instruction
  - One-address instruction
  - Zero-address instruction

## Example:

Let the expression is  $X = (A + B) \times (C + D)$

## Three-address instruction

ADD R1, A, B             $R1 \leftarrow M[A] + M[B]$

ADD R2, C, D             $R2 \leftarrow M[C] + M[D]$

MUL X, R1, R2            $M[X] \leftarrow R1 \times R2$

Expression is  $X = (A + B) \times (C + D)$

### Two-address instruction

MOV R1, A	$R1 \leftarrow M[A]$
ADD R1, B	$R1 \leftarrow R1 + M[B]$
MOV R2, C	$R2 \leftarrow M[C]$
ADD R2, D	$R2 \leftarrow R2 + M[D]$
MUL R1, R2	$R1 \leftarrow R1 \times R2$
MOV X, R1	$M[X] \leftarrow R1$

### One-address instruction

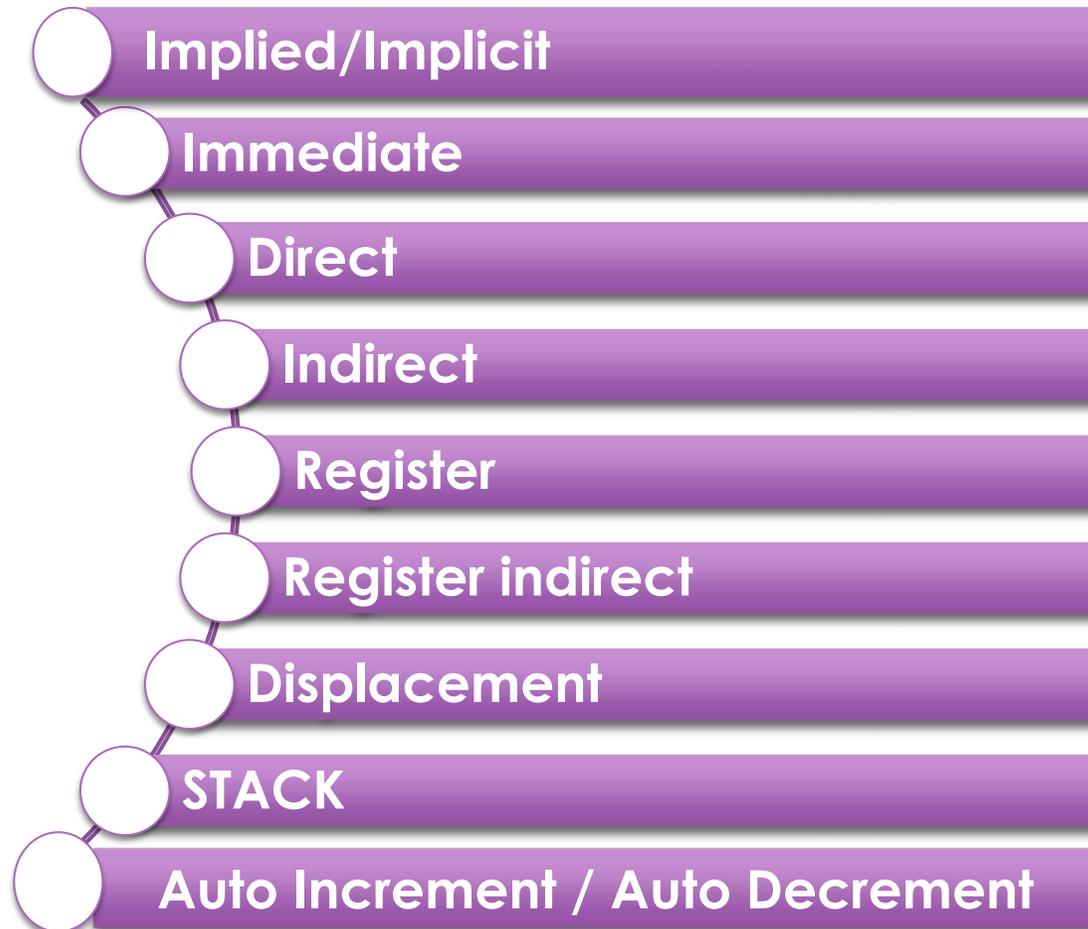
LOAD A	$AC \leftarrow M[A]$
ADD B	$AC \leftarrow AC + M[B]$
STOR T	$M[T] \leftarrow AC$
LOAD C	$AC \leftarrow M[C]$
ADD D	$AC \leftarrow AC + M[D]$
MUL T	$AC \leftarrow AC \times M[T]$
STOR X	$M[X] \leftarrow AC$

Expression is  $X = (A + B) \times (C + D)$

Zero-address instruction

PUSH	A	Top $\leftarrow$ A
PUSH	B	Top $\leftarrow$ B
ADD		Top $\leftarrow$ (A + B)
PUSH	C	Top $\leftarrow$ C
PUSH	D	Top $\leftarrow$ D
ADD		Top $\leftarrow$ (C + D)
MUL		Top $\leftarrow$ (A + B) $\times$ (C + D)
POP	X	M[X] $\leftarrow$ Top

# Addressing Modes



- In an instruction format the way the operands are chosen during program execution is dependent on the Addressing mode of the instruction.
- Before any operand is referenced, the addressing mode specifies the operand or address of the operand.
- Various types of Addressing modes are there as stated.
- A particular Instruction Set Architecture may follow some or all of these addressing modes to find the effective address of the operand.

# Implied/Implicit Addressing mode

- ▶ No address field required.



Instruction format

- ▶ The operand is specified within the instruction implicitly.
- ▶ All the instructions that uses accumulator implicitly within the instruction called implied addressing mode.
- ▶ CLA – Clear the content of Accumulator
- ▶ CMA – Complement the content of the Accumulator
- ▶ No memory reference other than the instruction fetch.

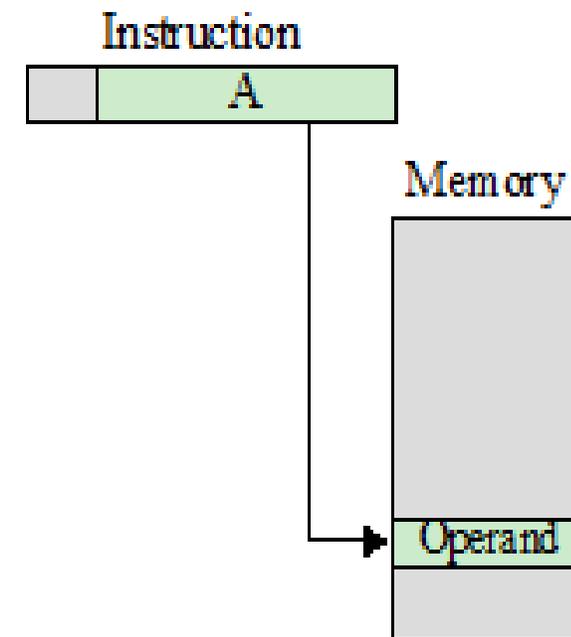
# Immediate Addressing Mode

- ▶ Simplest form of addressing.
- ▶ Operand = A
- ▶ The required operand is present in the instruction.
- ▶ This mode can be used to define and use constants or set initial values of variables.
- ▶ No memory reference other than the instruction fetch is required to obtain the operand, thus saving one memory or cache cycle in the instruction cycle.
- ▶ Size of the data restricted to the size of the Operand/Address field. It may be less than the word length, which is a drawback.



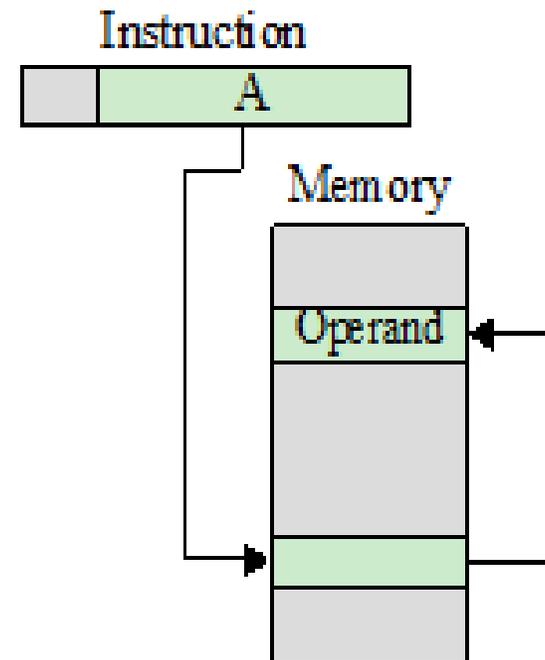
# Direct Addressing Mode

- ▶ In the instruction format the address field contains the effective address of the operand.
- ▶ Effective address (EA) = address field A
- ▶ It was common in earlier generations of computers.
- ▶ Except instruction fetching, it requires only one memory reference for data reading.
- ▶ No special calculation necessary for effective address.
- ▶ Limitation is that it provides only a limited address space.



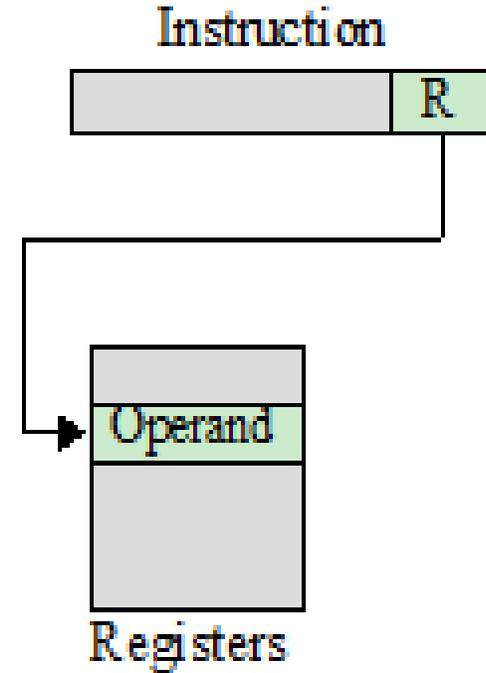
# Indirect Addressing Mode

- ▶ In the instruction format the Operand/Address field contains an address where the address of the data is present.
- ▶ Effective address (EA) = [A]
- ▶ Reference to the address of a word in memory which contains a full-length address of the operand.
- ▶ For a word length of  $N$  an address space of  $2^N$  is now available, which resolves the limitation of Direct addressing.
- ▶ Instruction execution requires two memory references to fetch the operand. One to get its address and a second to get its value.



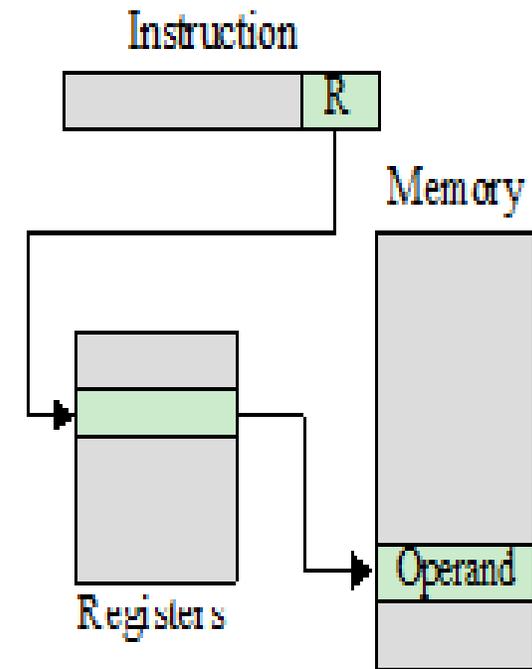
# Register Addressing Mode

- ▶ In the instruction format the Operand/Address field specifies a register rather than a memory address which contains the required operand.
- ▶  $EA = R$
- ▶ Only a small Operand/Address field is needed in the instruction to specify a particular register.
- ▶ No memory references rather register reference to find the required data.
- ▶ The address space is very limited



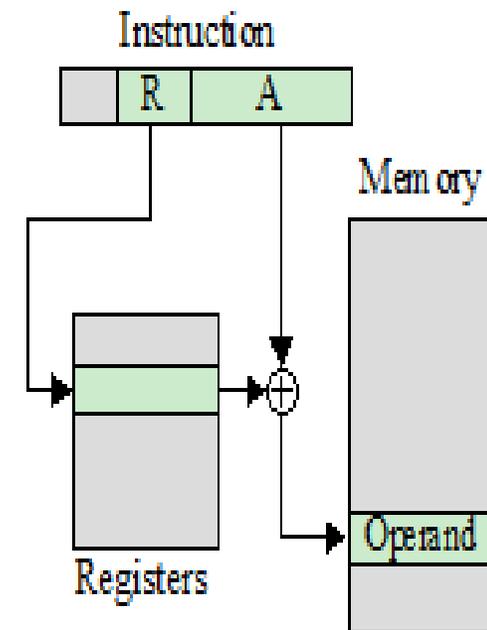
# Register Indirect Addressing Mode

- ▶ In the instruction format the Operand/Address field specifies a register which contains the address of an operand in the memory location.
- ▶ Analogous to indirect addressing
  - ▶ difference is whether the address field refers to a memory location or a register.
  - ▶ Uses one less memory reference than indirect addressing.
  - ▶ Instruction format may be smaller than indirect addressing.
- ▶  $EA = [R]$
- ▶ Address space limitation of the address field is overcome by having that field refer to a word-length location containing an address.
- ▶ Before using a Register Indirect mode instruction, the programmer must ensure that the memory address of the operand is placed in the processor register with a previous instruction.



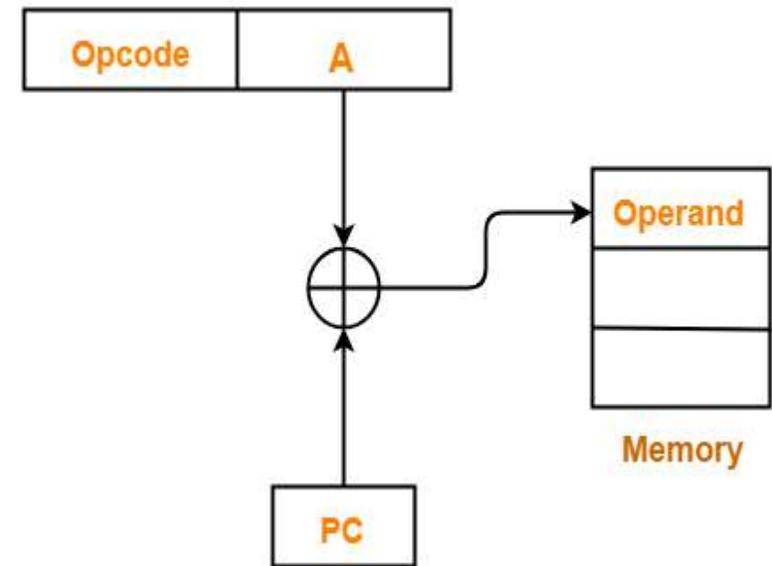
# Displacement Addressing Mode

- ▶ Very powerful addressing mode
- ▶ Combines the capabilities of register indirect addressing and direct addressing
- ▶  $EA = [R] + A$
- ▶ The instruction may have two address fields
  - ▶ The value contained in one address field (value = A) is used directly
  - ▶ The other address field refers to a register whose contents are added to A to produce the effective address
- ▶ Most common uses are:
  - ▶ Relative addressing
  - ▶ Base-register addressing
  - ▶ Indexed addressing



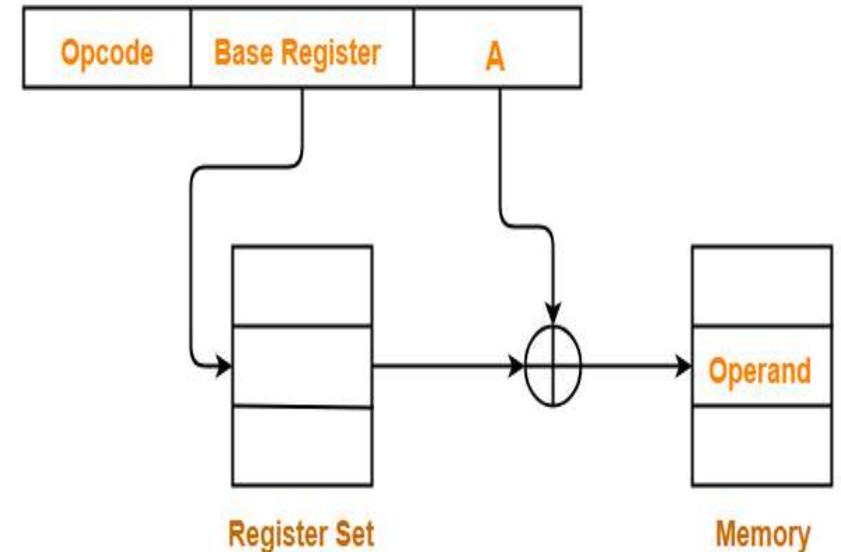
# Relative Addressing Mode

- ▶ The referenced register is the Program Counter (PC).
- ▶ The next instruction address (PC) is added to the address field A to produce the effective address (EA).
- ▶ Typically the address field is treated as a two's complement number for this operation.
- ▶ The effective address is a displacement relative to the address of the instruction.
- ▶ Exploits the concept of locality.
- ▶ Saves address bits in the instruction if most memory references are relatively near to the instruction being executed.
- ▶ Also called Limit Addressing Mode.



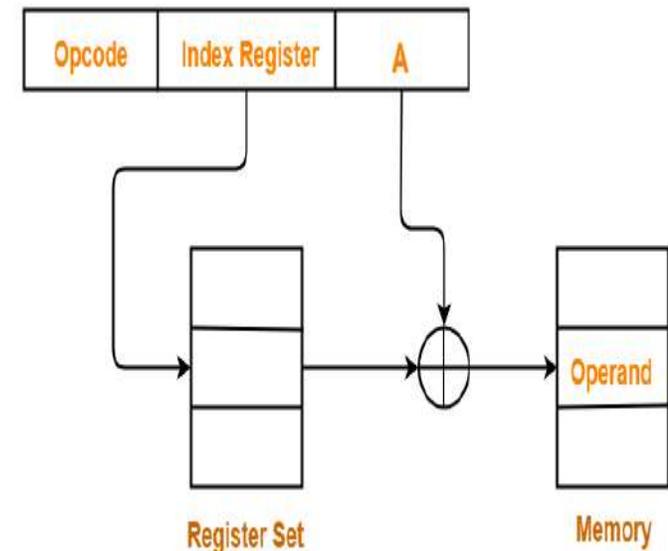
# Base-register Addressing Mode

- ▶ In this type of addressing mode the content of the Base register is added to the address part, that is, A of the instruction to obtain effective address.
- ▶ The Base register contains a main memory address and the address field contains a displacement from that address.
- ▶ Exploits the locality of memory references.
- ▶ Convenient means of implementing segmentation.
- ▶ In some implementations a single segment base register is employed.
- ▶ In others the programmer may choose a register to hold the base address of a segment and the instruction must reference it.



# Indexed Addressing Mode

- ▶ Index Register is a special CPU register contains an Index value.
- ▶ The address field (A) references a main memory address and the index register contains a positive displacement from that address.
- ▶ The method of calculating the EA is the same as for base-register addressing.
- ▶ An important use is to provide an efficient mechanism for performing iterative operations.
- ▶ Autoindexing
  - ▶ Automatically increment or decrement the index register after each reference to it
  - ▶  $EA = A + (XR)$
  - ▶  $(XR) \leftarrow (XR) + 1$



# STACK Addressing Mode

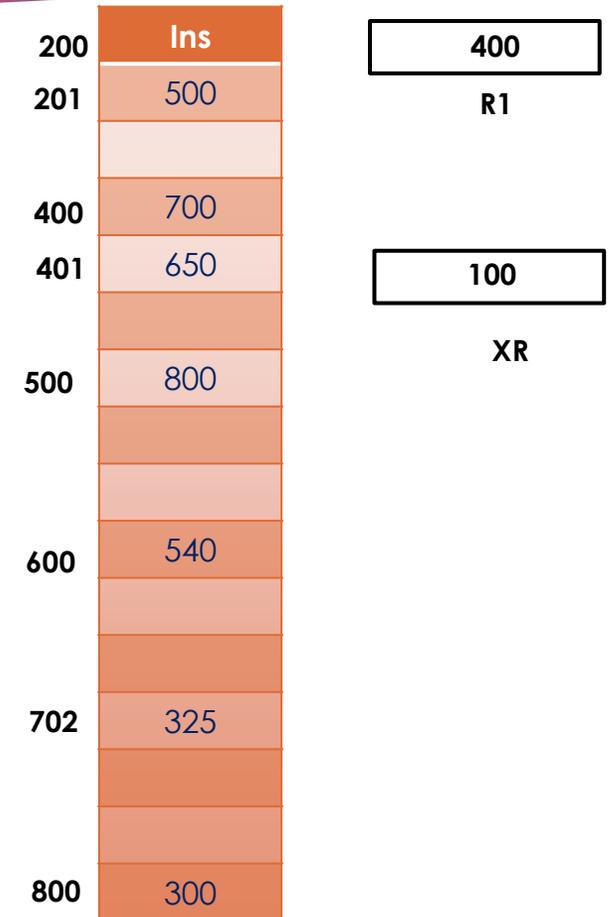
- ▶ A stack is a linear array of locations.
- ▶ Items are appended to the top of the stack so that the block is partially filled.
- ▶ Associated with the Stack Pointer (SP) whose value is the address of the top of the stack.
  - ▶ The stack pointer is a register
  - ▶ Thus references to stack locations in memory are in fact register indirect addresses
- ▶ This is also a form of implied addressing mode.
- ▶ In this mode, operand is at the top of the stack.
- ▶ For example: `ADD`, this instruction will `POP` top two operands from the stack, add them, and will then `PUSH` the result to the top of the stack.

# Auto increment/Auto decrement Addressing

- ▶ It is similar to Register or Register Indirect Addressing mode.
- ▶ If it Register addressing mode then the content of the specified register is incremented or decremented by 1.
- ▶ If it Register Indirect addressing mode then the address of the operand is incremented or decremented by 1.
- ▶ To access consecutive location the address value is stored in the register and then incremented or decremented

# Example:

- ▶ An instruction is stored at location 200 with address field at location 201. The address field has the value 500. A processor register R1 contains 400. Evaluate the effective address for different addressing modes if XR value is 100.
  - Direct Addressing Mode:  $EA = 500, AC = 800$
  - Indirect Addressing Mode:  $EA = 800, AC = 300$
  - Immediate Addressing Mode:  $EA = 201, AC = 500$
  - Register Addressing Mode:  $EA = No, AC = 400$
  - Register Indirect Addressing Mode:  $EA = [R1] = 400, AC = 700$
  - Relative Addressing Mode:  $EA = [PC] + 500 = 702, AC = 325$
  - Indexed Addressing Mode:  $EA = [XR] + 500 = 600, AC = 540$
  - Auto increment (Register addressing):  $EA = No, AC = 401$
  - Auto increment (Register indirect addressing):  $EA = 401, AC = 650$



# Types of Instruction

- ▶ A computer provides an extensive set of instructions to give the user flexibility to carry out various computational tasks.
- ▶ The instruction set for different processors differ from each other mostly in the way the operands are determined and mode field.
- ▶ The actual operations available in the instruction set are not very different from one computer to another.
- ▶ The binary code of the Opcode may differs from processor to processor, even for the same operation.
- ▶ Most computer instructions can be classified into following categories.
  - Data transfer instructions
  - Data manipulation instructions
    - Arithmetic instructions
    - Logical and bit manipulation instructions
    - Shift instructions
  - Program control instructions

## Data Transfer Instructions

- ▶ Data transfer instructions move data from one place to another without changing the data content.
- ▶ Most common transfers are between memory and processor registers, between processor registers and I/O, and between processor registers themselves.
- ▶ LOAD, STOR, MOV, XCHG, IN, OUT, PUSH, POP etc.

## Data manipulation instructions

- ▶ Data manipulation instructions perform operations on data and provide the computational capabilities for the computer.
  - Arithmetic instructions
  - Logical and bit manipulation instructions
  - Shift instructions

## Arithmetic Instructions

- ▶ Basic arithmetic operations are addition, subtraction, multiplication and division.
- ▶ Most computers provide instructions for all four operations.
- ▶ Some small computers have only addition and possibly subtraction instructions.
- ▶ The multiplication and division must then be performed by help of addition and other instructions.
- ▶ ADD, SUB, MUL, DIV, INC, DEC, ADDC, SUBB etc.

## Logical and Bit manipulation Instructions

- ▶ AND, OR, XOR, CLR, COM, CLRC, SETC, COMC etc.

## Shift Instructions

- ▶ Instructions to shift the content of a register or accumulator are quite useful.
- ▶ Shift operations may specify either logical shifts or arithmetic shifts or rotate type operations.
- ▶ In either case the shift may be to the left or to the right.
- ▶ SHR, SHL, ASHR, ASHL, RAR, RAL, RRC, RLC etc.

## Program Control instructions

- ▶ Instructions are always stored in successive memory locations.
- ▶ Instructions are fetched from memory and executed.
- ▶ Just after fetching of an instruction, the Program counter is incremented for the next instruction in sequence.
- ▶ On the other hand, a program control type of instruction, when executed may change the address value in the Program counter and cause the flow of control to be altered.
- ▶ The change in value of the Program counter due to program control instruction cause a break in sequence of instruction execution.
- ▶ BR, JUMP, JC, JNC, JZ, JNZ, JP, JN, SKP, CALL, RET etc.

# Instruction Cycle

- ▶ An instruction is a command given to the computer to perform specific operation on given data.
- ▶ To perform a particular task a sequence of instructions are written, called a program.
- ▶ Generally instruction and data are stored in the memory.
- ▶ The necessary steps that a CPU carries out to fetch an instruction and required data and then to process it constitute an Instruction Cycle.
- ▶ Each Instruction cycle is subdivided into a sequence of sub cycles or phases.
- ▶ In Basic Computer, an instruction passes through following sub cycles:
  1. Fetch an instruction from memory
  2. Decode the instruction
  3. Read the required Operand
  4. Entertain the Interrupt if it is generated
  5. Execute the instruction

*Note:* Every different processor has its own (different) instruction cycle

# Instruction Cycle

Includes the following stages:

**Fetch**

**Read the next instruction from memory into the processor**

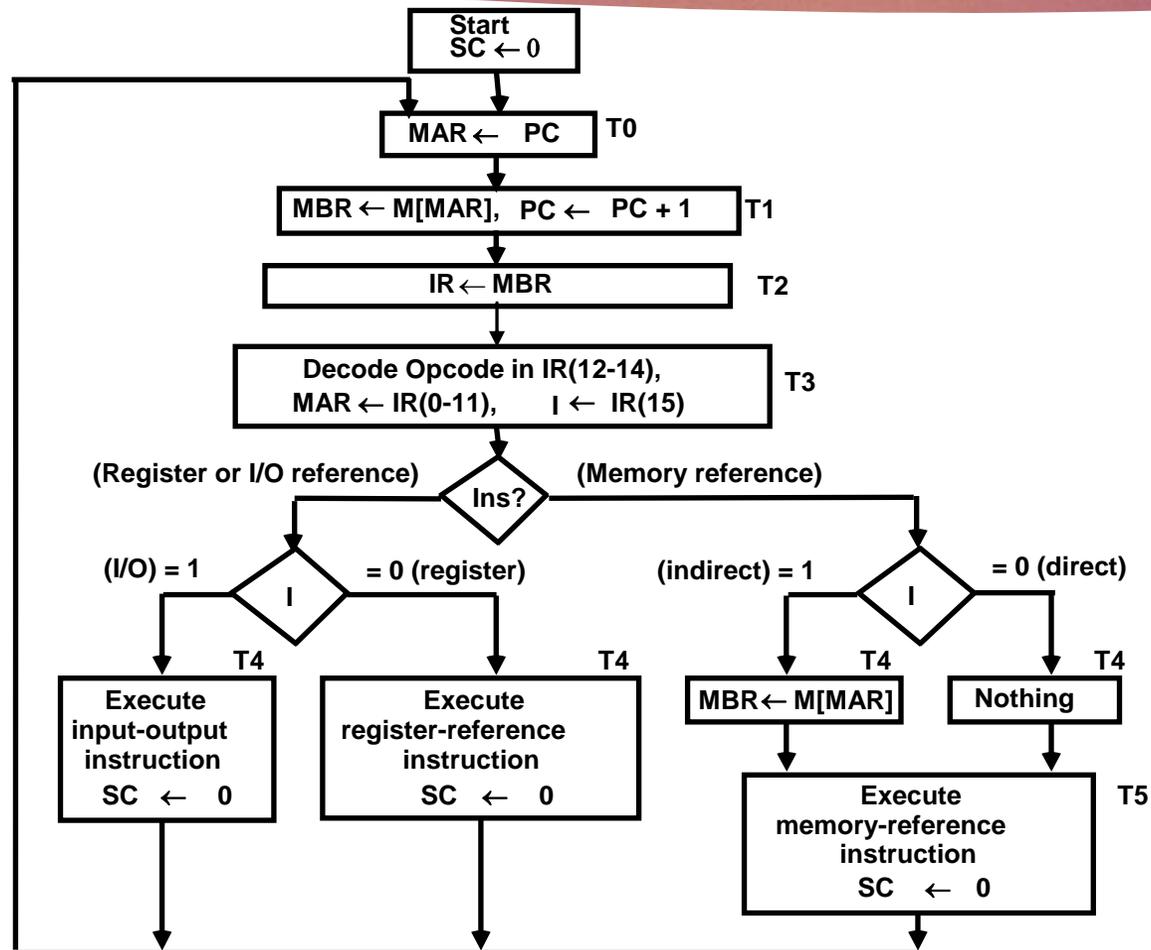
**Execute**

**Interpret the opcode and perform the indicated operation**

**Interrupt if generated**

**If interrupts are enabled and an interrupt has occurred, save the current process state and service the interrupt**

# Instruction Cycle Flow Diagram



T0:  $MAR \leftarrow PC$

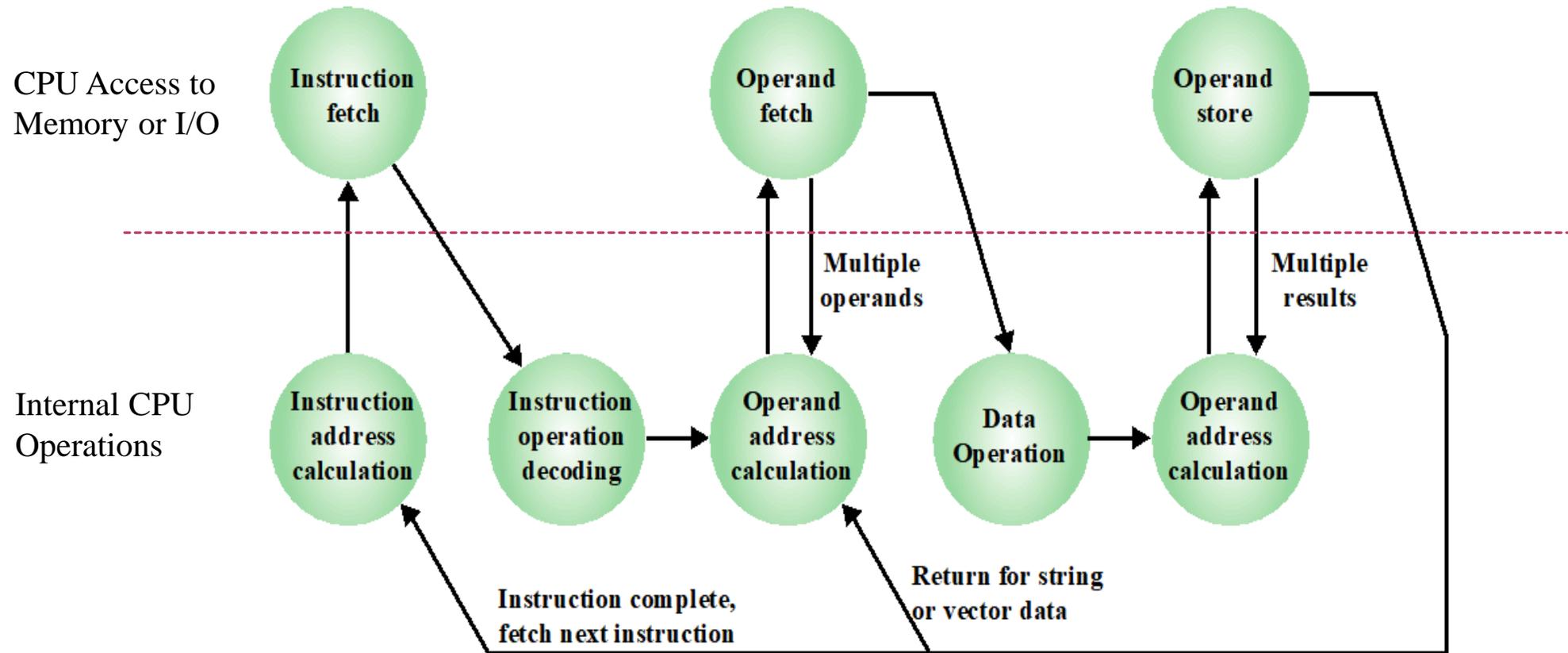
T1:  $MBR \leftarrow M[MAR], PC \leftarrow PC + 1$

T2:  $IR \leftarrow MBR$

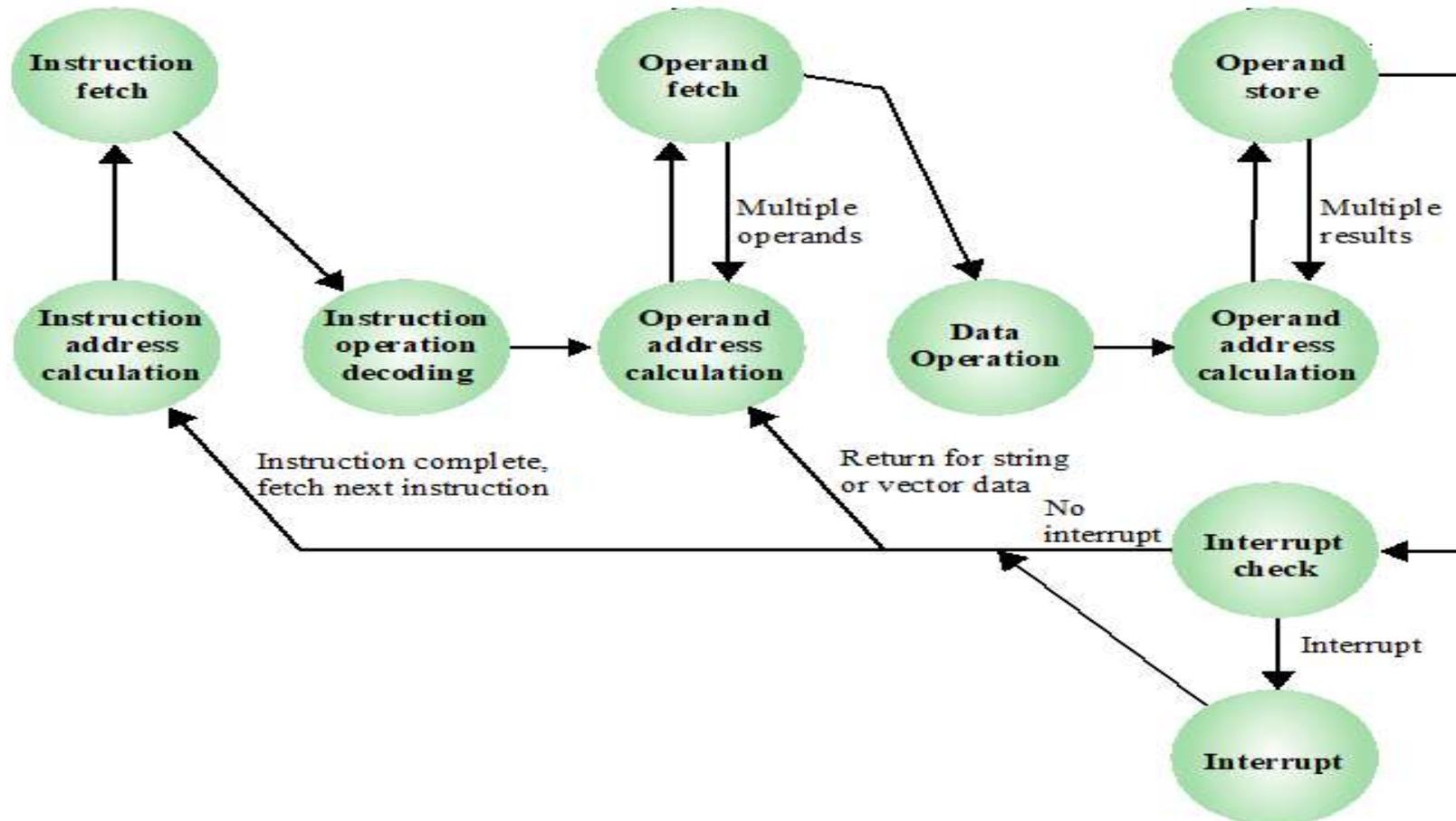
T3: Decode

T4, T5: Execute

# Instruction Cycle State Diagram



# Instruction Cycle State Diagram with Interrupt





# **CPU Organization**

# Overview

- ▶ **Introduction**
- ▶ **Single Bus organization (Data Path) inside Processor**
- ▶ **Register Transfers**
- ▶ **Control Sequences**
- ▶ **Fetching a Word from Memory**
- ▶ **Storing a Word in Memory**
- ▶ **Control Sequences for Execution of a Complete Instruction**
- ▶ **Multi Bus organization (Data Path) inside Processor**
- ▶ **Organization of Basic Control Unit**
- ▶ **Hardwired Control Unit**
- ▶ **Microprogrammed Control Unit**
- ▶ **Stack Organization**
- ▶ **Revers Polish Notation (RPN)**
- ▶ **Evaluation of Arithmetic Expression using RPN**
- ▶ **Subroutine**

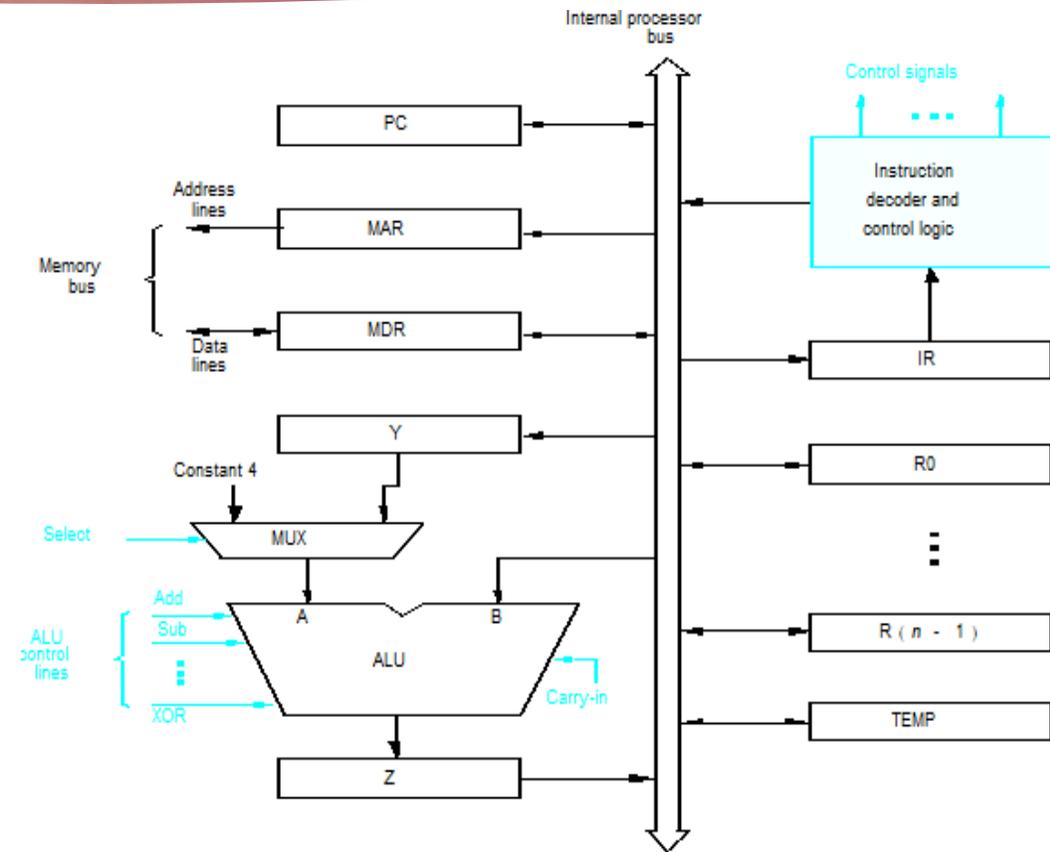
# Introduction

- ▶ Processor fetches one instruction at a time and perform the required operation.
- ▶ Instructions are fetched from successive memory locations until a Branch, Call or a Jump instruction is encountered.
- ▶ Processor keeps track of the address of the memory location containing the next instruction to be fetched using Program Counter (PC).
- ▶ After fetching from memory instruction is stored in Instruction Register (IR) for decoding.
  1. Fetch the contents of the memory location pointed to by the PC.
  2. Assuming that the memory is byte addressable, and each instruction comprises 4 bytes, increment the contents of the PC by 4.  $PC \leftarrow [PC] + 4$
  3. Carry out the operations specified by the instruction.

**Steps 1 and 2 constitute the fetch phase, and step 3 constitutes the execution phase.**

# Single Bus organization (Data Path) inside Processor

- ▶ Generally there are two sub units within the processor, that is, Control Unit and Datapath.
- ▶ Physically there is hardly any difference between the Control unit and Datapath.
- ▶ The hardware of both subunits are tightly coupled in a single physical unit.
- ▶ The Datapath includes the internal path for movement of data within the Processor between ALU and registers and other hardware like temporary storage, latches, multiplexers, demultiplexers, decoders, counters, delay logics etc.
- ▶ Other buses are external buses that connects memory and I/O devices.

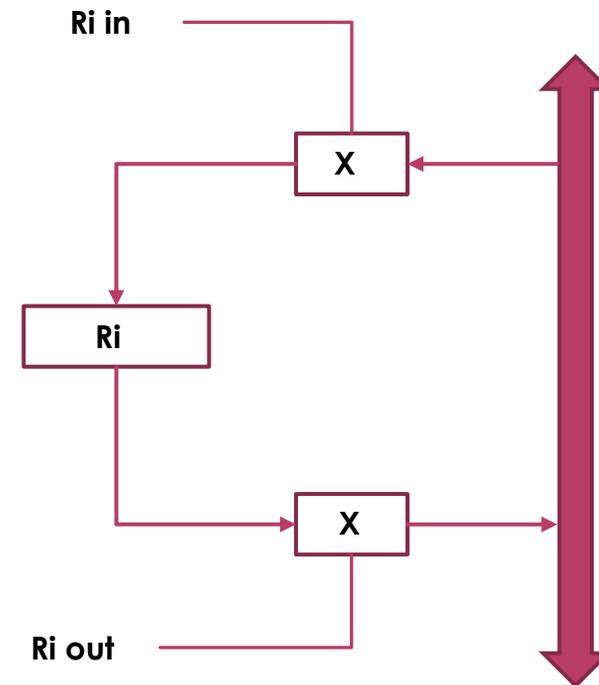


Single-bus organization of the datapath inside a processor.

- ▶ The ALU and all the registers are interconnected via a single common bus. This bus is internal to the processor and should not be confused with the external bus that connects the processor to the memory and I/O devices.
- ▶ The data and address lines of the external memory bus are connected to the internal processor bus via the memory data register (MDR), and the memory address register (MAR), respectively.
- ▶ The input of the MAR is connected to the internal bus, and its output is connected to the external bus.
- ▶ The number and use of the processor registers R0 through R(n-1) vary from one processor to another.
- ▶ Three registers, Y, Z, and TEMP are transparent to the programmer. They are used by the processor for temporary storage during execution of some instructions.
- ▶ The multiplexer MUX selects either the output of register Y or a constant value 4 to be provided as input of A of the ALU. The constant 4 is used to increment the content of the program counter (PC).
- ▶ The instruction decoder and the control logic unit is responsible for implementing the actions specified by the instruction loaded in the IR register.

# Register Transfers

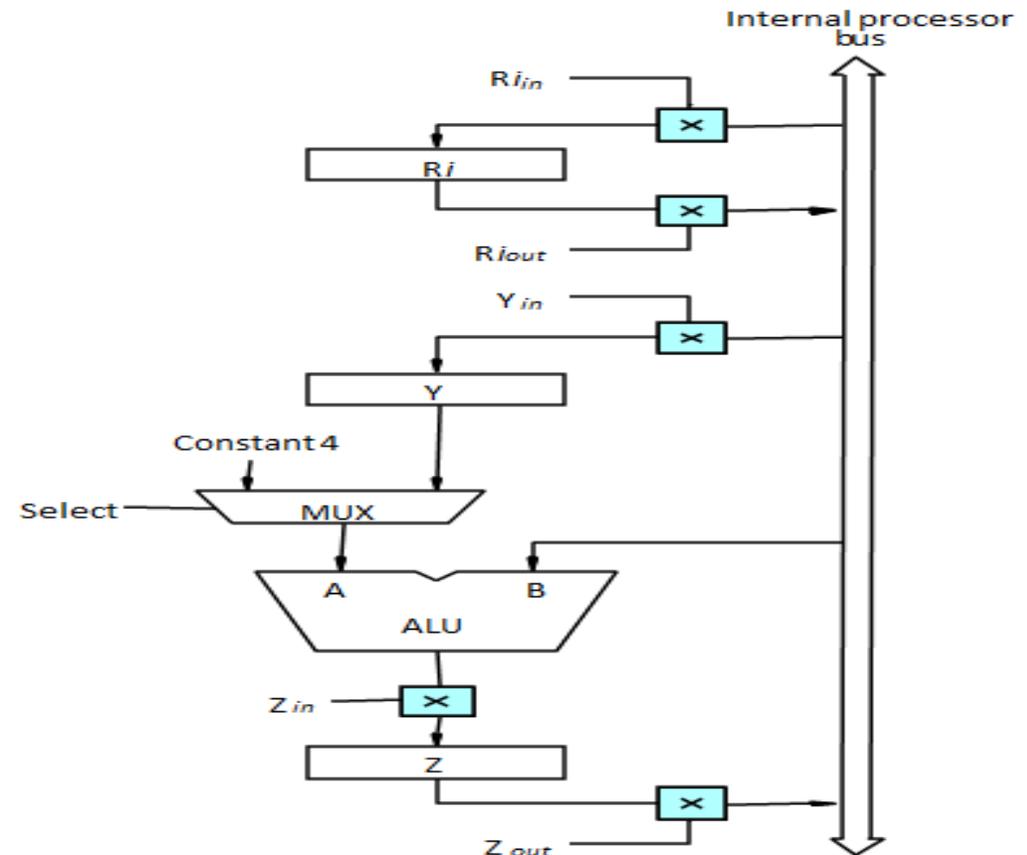
- ▶ Instruction execution involves a sequence of steps in which data are transferred from one register to another.
- ▶ For each register, two control signals are used to place the content of that register on the bus or from the bus to the register.
- ▶ The input and output of register  $R_i$  are connected to the bus via switches controlled by the signal  $Ri_{in}$  and  $Ri_{out}$  respectively.
- ▶ When  $Ri_{in} = 1$ , the data on the bus are loaded into  $R_i$ .
- ▶ When  $Ri_{out} = 1$ , the contents of register  $R_i$  are placed on the bus.
- ▶ While  $Ri_{out} = 0$ , the bus can be used for transferring data from other registers.
- ▶ All the operations and data transfers within the processor take place within time periods defined by the processor clock.



# Control Sequences

- ▶ The ALU is a combinational circuit that has no internal storage.
- ▶ ALU gets the two operands from MUX and bus. The result is temporarily stored in register Z.
- ▶ A sequence of operations to add the contents of register R1 to R2 and store the result in R3 is shown below.

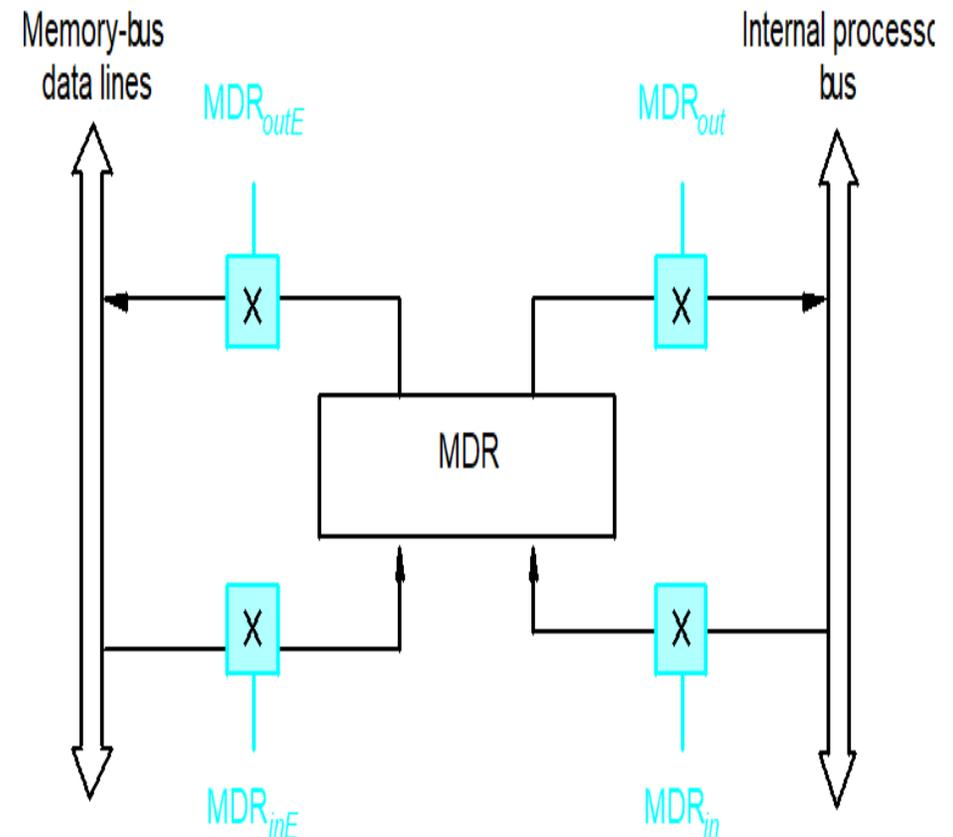
1.  $R1_{out}, Y_{in}$
2.  $R2_{out}, \text{Select}Y, \text{Add}, Z_{in}$
3.  $Z_{out}, R3_{in}$



Input and output gating for the registers

# Fetching a Word from Memory

- ▶ To fetch a word from the memory, the processor has to specify the address of the memory location
- ▶ Address into MAR; issue Read operation; data into MDR.
- ▶ It has four control signals.
- ▶  $MDR_{in}$  and  $MDR_{out}$  control the connection to the internal bus.
- ▶  $MDR_{inE}$  and  $MDR_{outE}$  control the connection to the external bus.



- ▶ During memory Read and Write operations, the timing of internal processor operations must be coordinated with the response of the addressed device on the memory bus.
- ▶ The processor completes one internal data transfer in one clock cycle.
- ▶ The speed of the operation of I/O devices are slower than the processor speed and also different device speed is different from each other.
- ▶ To accommodate this, the processor waits until it receives an indication that the requested operation has been completed.
- ▶ A control signal called Memory-Function-Complete (MFC) is used for this purpose.
- ▶ During Read or Write operation to or from memory, the other operations Wait for MFC signal.
- ▶ WMFC is a control signal that causes processor control circuitry to wait for arrival of MFC signal.

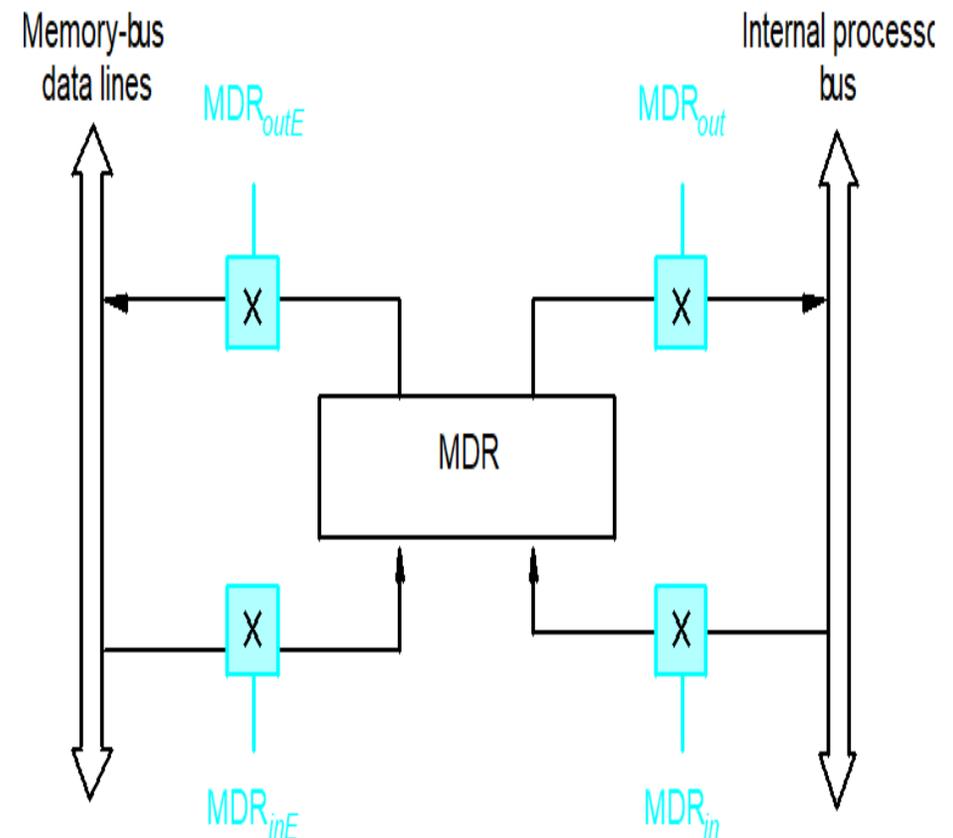
**Example:** Consider an instruction MOV R2, [R1]

► The actions needed for memory read are:

1.  $MAR \leftarrow [R1]$
2. Start a Read operation on the memory bus
3. Wait for the MFC response from the memory
4. Load MDR from the memory bus
5.  $R2 \leftarrow MDR$

► Control Sequences are:

1.  $R1_{out}$ ,  $MAR_{in}$ , Read
2.  $MDR_{inE}$ , WMFC
3.  $MDR_{out}$ ,  $R2_{in}$



# Storing a Word in Memory

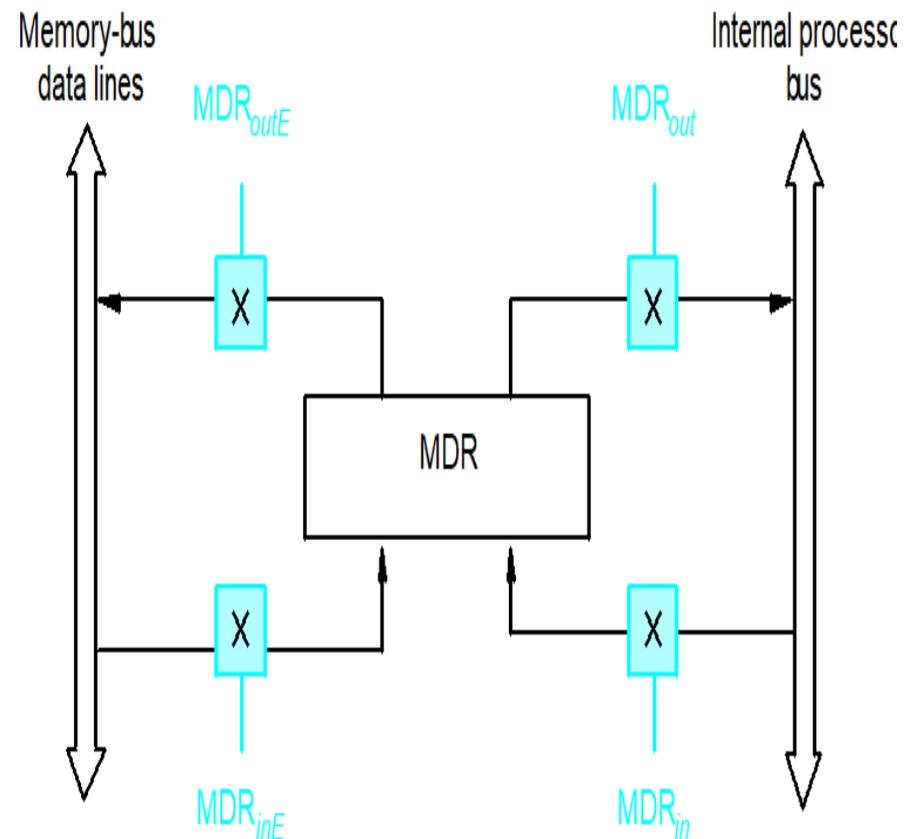
**Example:** Consider an instruction Move [R1], R2

► The actions needed for memory write are:

1.  $MAR \leftarrow [R1]$
2.  $MDR \leftarrow R2$
3. Start a write operation on the memory bus
4. Wait for the MFC response from the memory
5. Store memory bus from MDR

► Control Sequences are:

1.  $R1_{out}, MAR_{in}$
2.  $R2_{out}, MDR_{in}, Write$
3.  $MDR_{out}, E, WMFC$



# Execution of a Complete Instruction

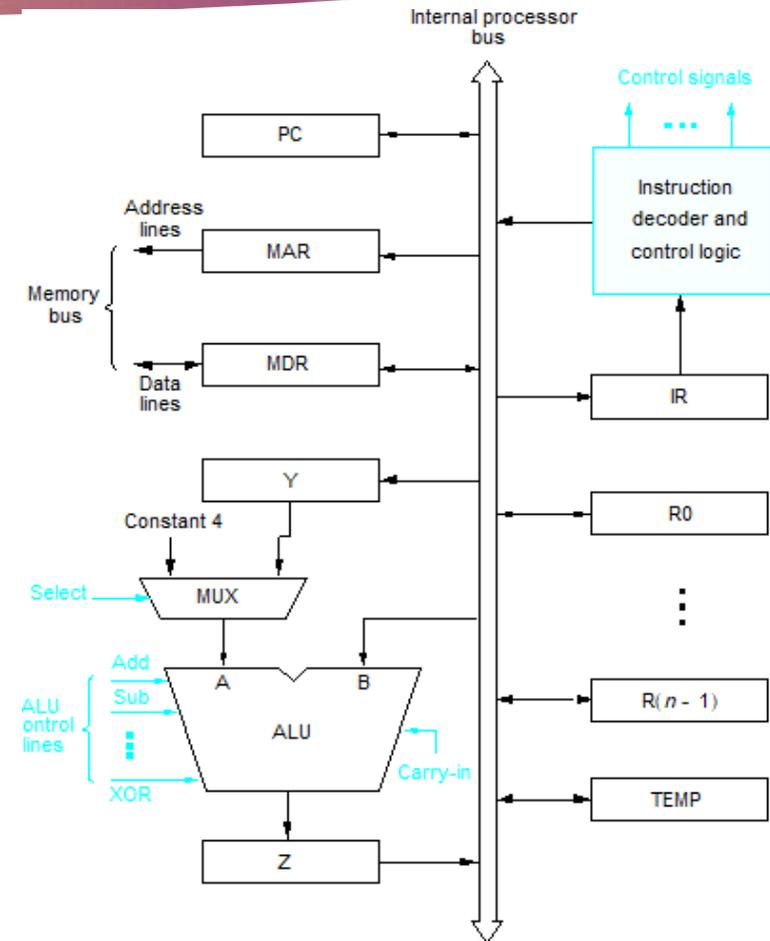
**Example:** Consider an instruction ADD R1, [R3]

- ▶ The actions needed for execution of complete instruction are:
  1. Fetch the instruction
  2. Read the operand (the contents of the memory location pointed to by R3)
  3. Perform the addition
  4. Load the result into R1

**Example:** Consider an instruction ADD R1, [R3]

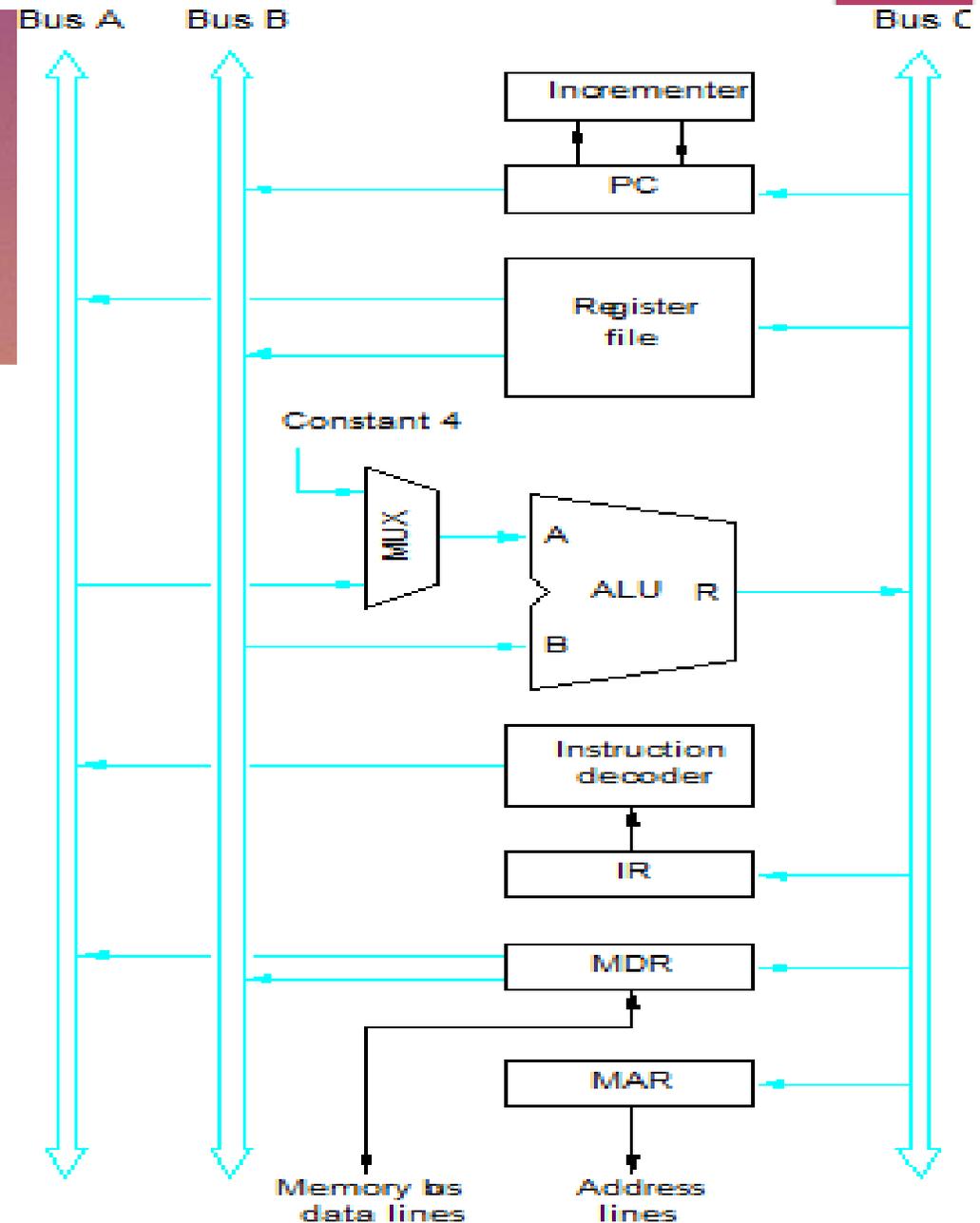
▶ Control Sequences are:

1.  $PC_{out}$ ,  $MAR_{in}$ , Read, Select4, Add,  $Z_{in}$
  2.  $Z_{out}$ ,  $PC_{in}$ , WMFC
  3.  $MDR_{inE}$ ,  $MDR_{out}$ ,  $IR_{in}$
  4.  $R3_{out}$ ,  $MAR_{in}$ , Read
  5.  $R1_{out}$ ,  $Y_{in}$ , WMFC
  6.  $MDR_{inE}$ ,  $MDR_{out}$ , SelectY, Add,  $Z_{in}$
  7.  $Z_{out}$ ,  $R1_{in}$ , End
- ▶ Sequence 1, 2 and 3 for Opcode fetching and PC increment for next instruction.
- ▶ Sequence 4 for locating memory location to read operand.
- ▶ Sequence 5 for reading operand from register.
- ▶ Sequence 6 for operand reading from memory to ALU and arithmetic operation.

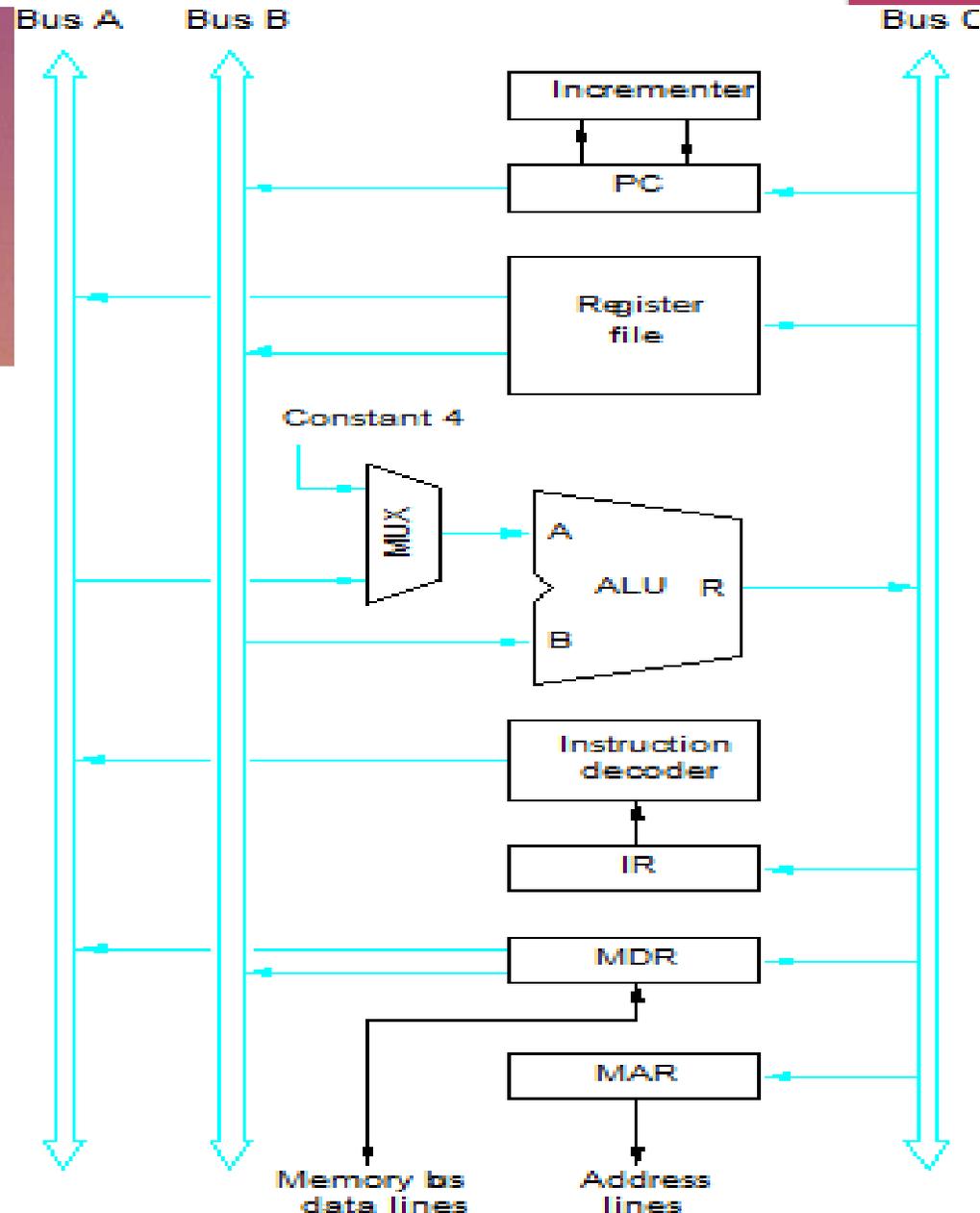


# Multi Bus organization (Data Path)inside Processor

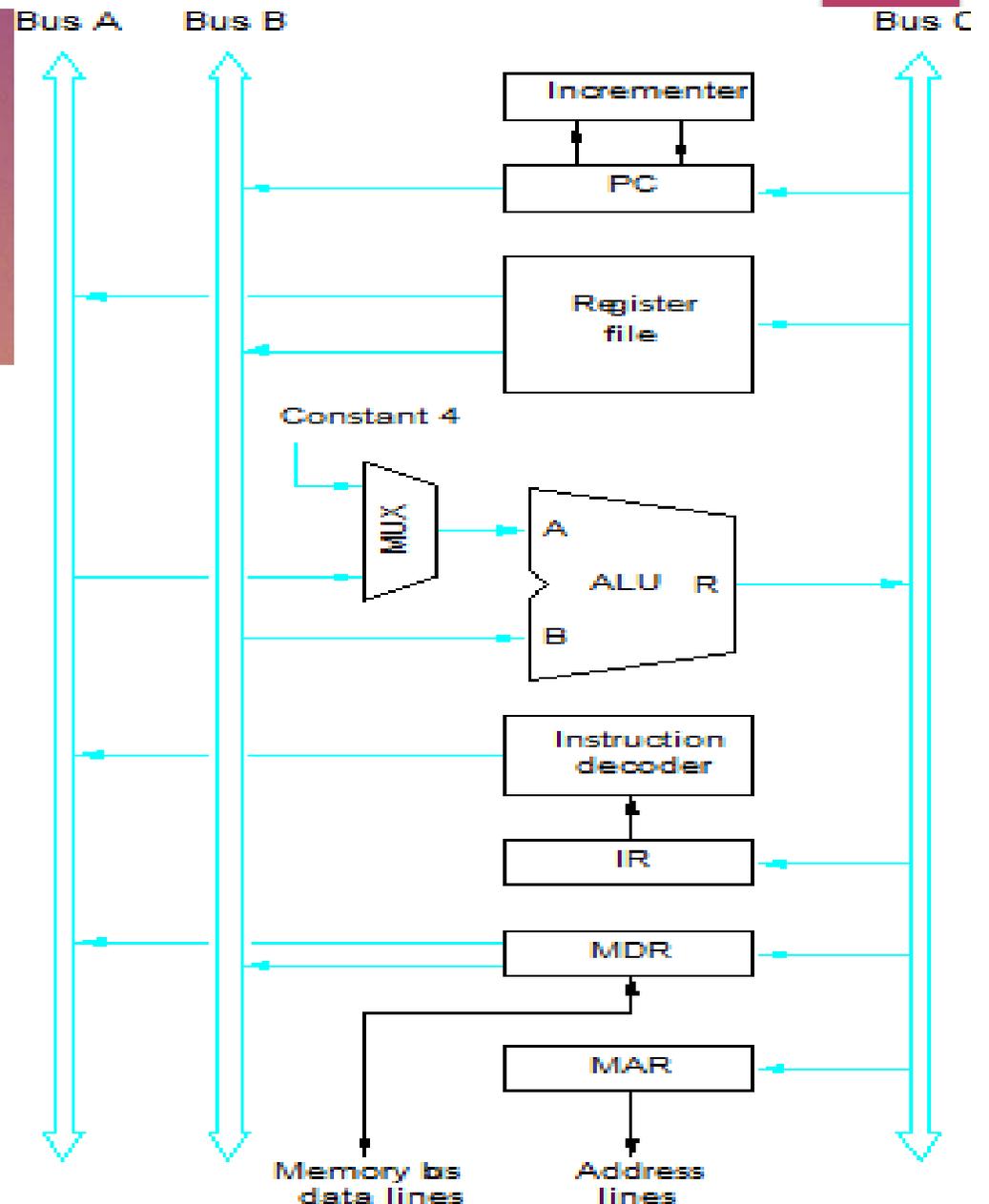
- ▶ With a single bus organization, the resulting control sequences are quite long because only one data item can be transferred over the bus in a single clock cycle.
- ▶ To reduce the number steps needed, most commercial processors provide multiple internal paths that enable several transfers to take place in parallel.
- ▶ All general-purpose registers are combined into a single block called the register file.
- ▶ The register file is said to have three ports. There are two outputs, allowing the contents of two different registers to be accessed simultaneously and have their contents placed on buses A and B.



- ▶ The third port allows the data on the bus C to be loaded into a third register during the same clock cycle.
- ▶ Buses A and B are used to transfer the source operands to the A and B inputs of the ALU.
- ▶ The result is transferred to the destination over bus C.
- ▶ If needed, the ALU may simply pass one of its two input operands unmodified to bus C.
- ▶ The ALU control signals for such an operation may be called as  $R = A$  or  $R = B$ .



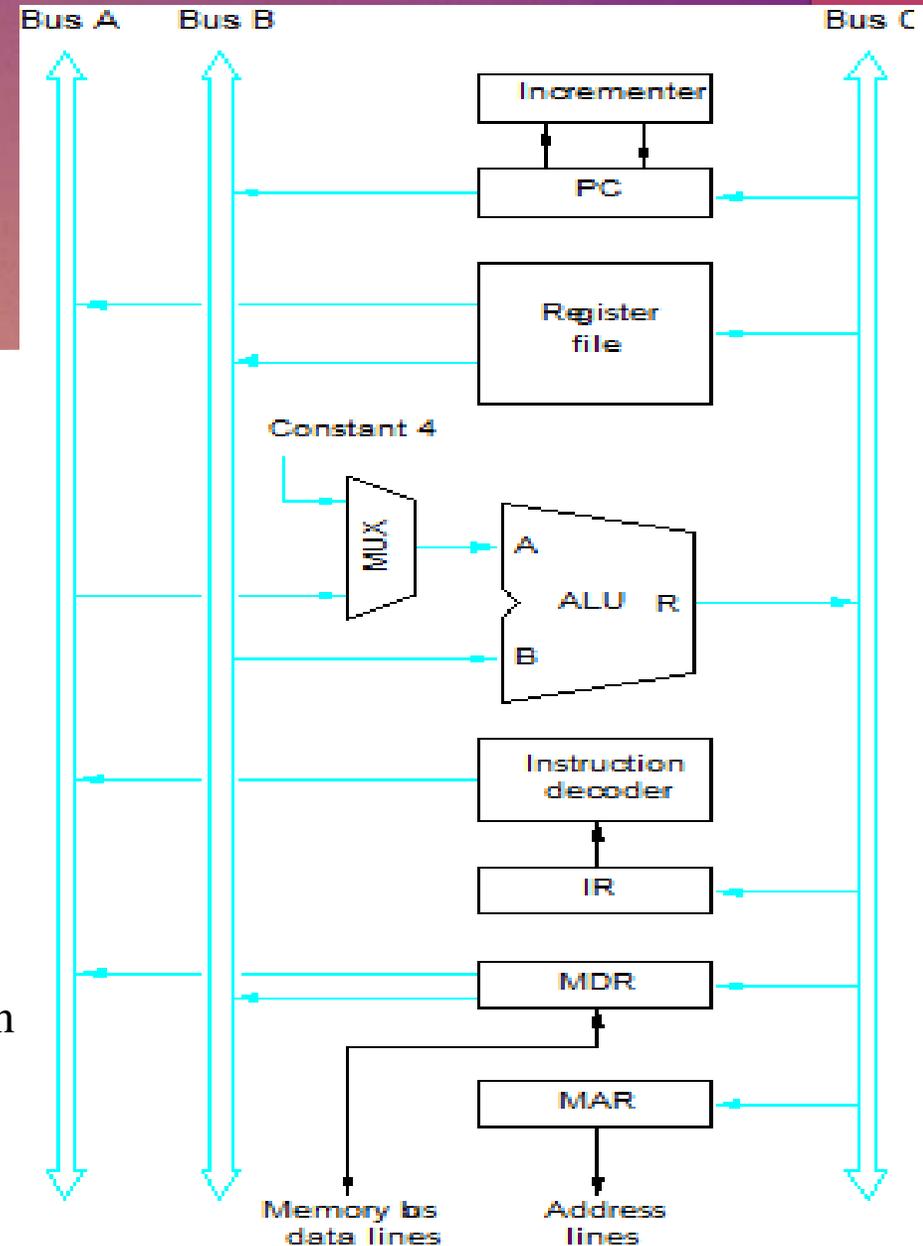
- ▶ The three-bus arrangement obviates the need for the registers Y and Z (available in the single-bus organization).
- ▶ Another feature is the introduction of the Incrementor unit, which is used to increment the PC by 4.
- ▶ Using the Incrementor eliminates the need to add 4 to the PC using the main ALU.
- ▶ The source for the constant 4 at the ALU input multiplexer is still useful. It can be used to increment other addresses, such as the memory addresses in LoadMultiple and StoreMultiple instruction.



► Add R6, R5, R4

Step	Action
1	$PC_{out}$ , $R=B$ , $MAR_{in}$ , Read, IncPC
2	WMFC
3	$MDR_{inE}$ , $MDR_{outB}$ , $R=B$ , $IR_{in}$
4	$R4_{outA}$ , $R5_{outB}$ , SelectA, Add, $R6_{in}$ , End

Control sequence for the instruction for the three-bus organization



# Control Unit

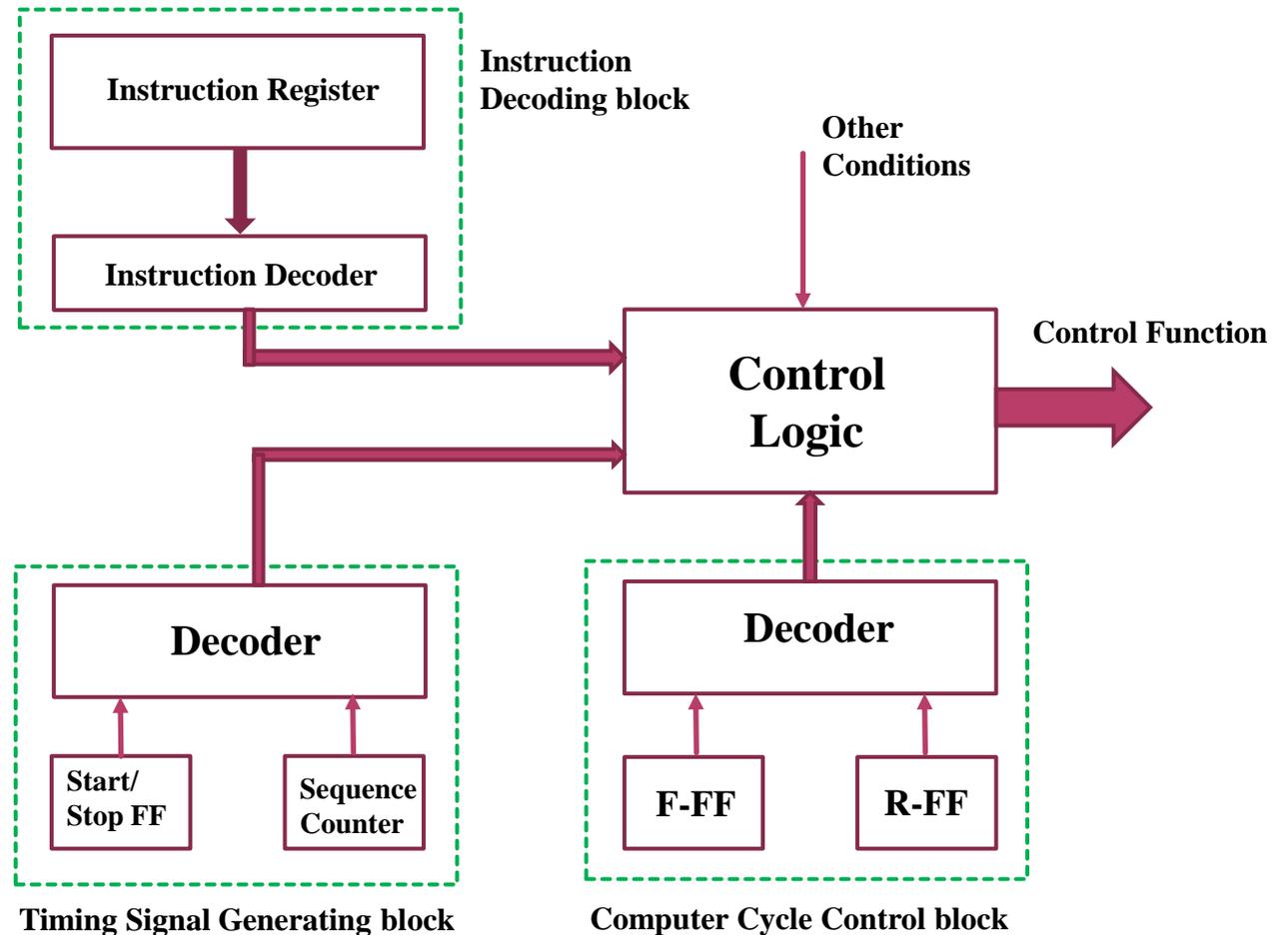
- ▶ All the operations in a computer system must be coordinated in some synchronized way, which is the task of control unit.
- ▶ This unit effectively the nerve center of the system.
- ▶ In concept it is reasonable to think of a control unit as a well defined physically separate unit that interacts with other parts of the system but, in practice much of the control circuitry is physically distributed throughout the machine.

# Organization of basic Control Unit

- ▶ The organization of basic control unit of a system consists of four blocks.
  - Instruction Decoding block
  - Timing Signal Generating block
  - Computer Cycle Control block
  - Control Logic

## Instruction Decoding block

- ▶ It consists of an Instruction register and Instruction decoder.
- ▶ It decodes the instruction and provides necessary information to the Control Logic.



### Timing Signal Generating block

- ▶ It is essentially consists of a Sequence Counter and a Start/Stop flip flop.
- ▶ Start/Stop flip flop is used to enable the decoder.
- ▶ The Decoder provides necessary timing signals to the Control Logic.

### Computer Cycle Control block

- ▶ This block consists of two flip flops **F** and **R** and a Decoder.
- ▶ The computer cycle is determined by this circuit depending on the following Truth Table.

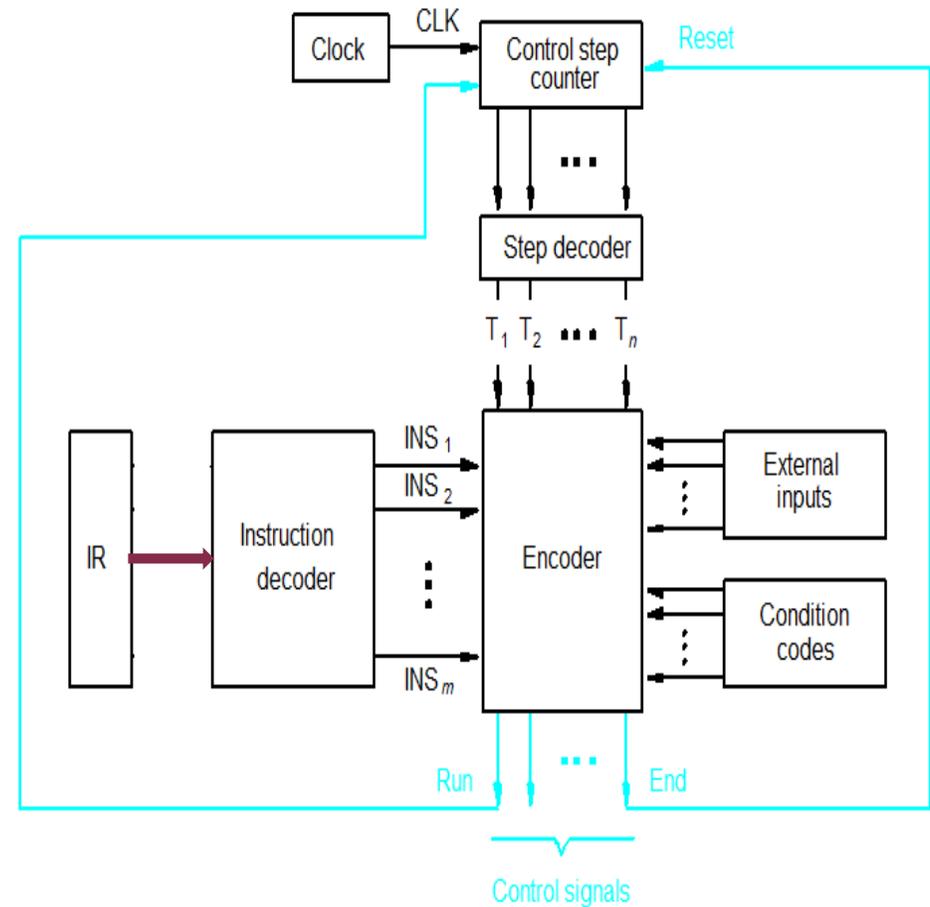
F	R	Cycle
0	0	Fetch Cycle
0	1	Operand Cycle
1	0	Execute Cycle
1	1	Interrupt Cycle

## Control Logic

- ▶ It is a complex circuit which receives inputs from Instruction decoding block, Timing signal generating block and Computer cycle control block.
- ▶ It also takes some control conditions from outside and generates appropriate Control Function or Control word (CW).
  
- ▶ After decoding the instructions the Control unit must have some means of generating control signals needed to complete the operations.
- ▶ The computers are designed by implementing a wide variety of technologies to perform this.
- ▶ Most of these technologies, however, fall into two categories.
  - Hardwired Control
  - Microprogrammed Control

# Hardwired Control Unit

- ▶ Each step in sequence is completed in one clock period.
- ▶ A counter may be used to keep track of the control steps.
- ▶ Each state, or count, of this counter corresponds to one control step.
- ▶ The required control signals are determined by the following information.
  - Contents of the instruction register and decoder
  - Contents of the condition code flags
  - External input signals, such as MFC and interrupt requests
  - Signals from Step decoder



- ▶ The step decoder provides a separate signal line for each step, or time slot, in the control sequence.
- ▶ Similarly, the output of the instruction decoder consists of a separate line for each machine instruction.
- ▶ For any instruction loaded in the IR, one of the output lines  $INS_1$  through  $INS_m$  is set to 1, and all other lines are set to 0.
- ▶ The input signals to the encoder block are combined to generate the individual control signals like  $Y_{in}$ ,  $PC_{out}$ , Add, End, and so on.
- ▶ The control hardware can be viewed as a state machine that changes from one state to another in every clock cycle, depending on the contents of the instruction register, the condition codes, and the external inputs.
- ▶ The outputs of the state machine are the control signals from which all or some of the lines are activated.
- ▶ The sequence of operations carried out by this machine is determined by the wiring of the logic elements, hence the name “hardwired”.

## Advantages

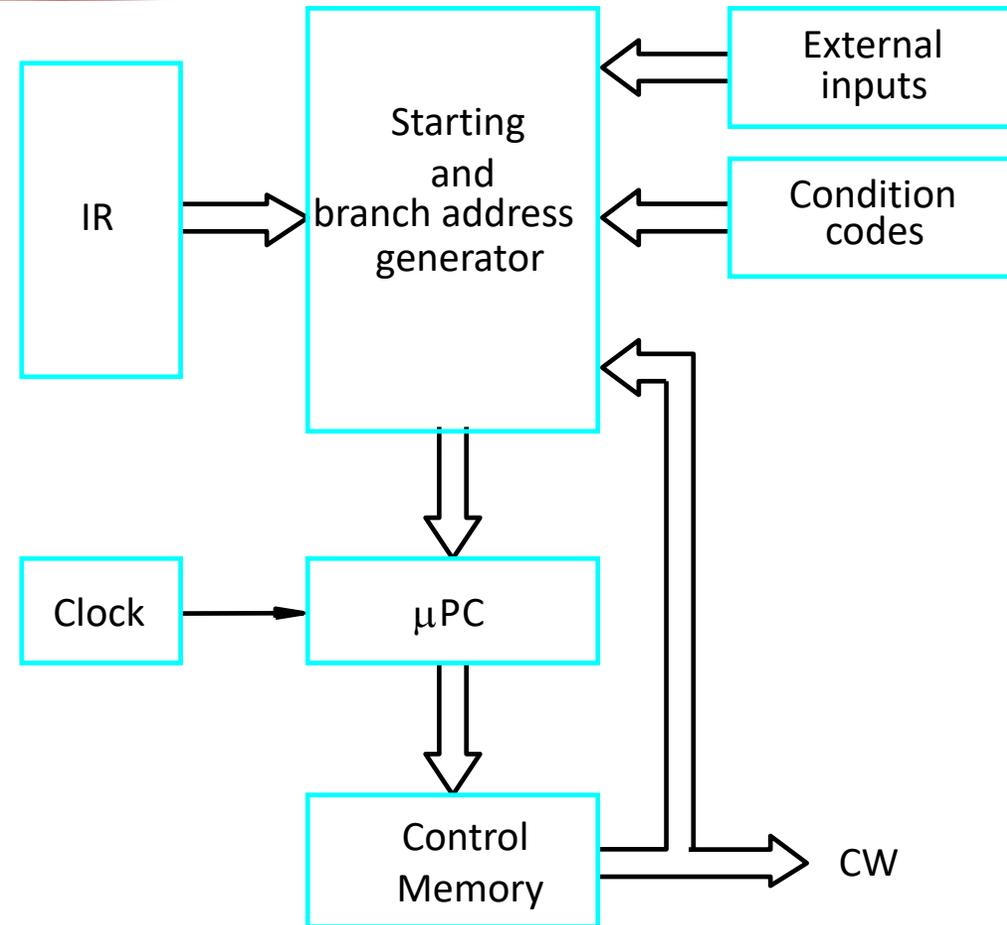
- ▶ A hardwired control unit works fast.
- ▶ The combinational circuits generate the control signal based on the input signals status. The delay between the output generation to the input availability depends on the number of gates in the path and propagation delay of each gate in the path.

## Disadvantages

- ▶ If the CPU has a large number of control points, the control unit design becomes very complex.
- ▶ The design does not give any flexibility.
- ▶ If any modification instruction set is required, it is extremely difficult to make the correction.

# Microprogrammed Control Unit

- ▶ Microprogramming is a method of Control unit design in which the control signal selection and sequencing information is stored in ROM or RAM, called Control Memory (CM).
- ▶ The control signal to be activated at any time are specified by a microinstruction which is fetched from the Control memory.
- ▶ Each microinstruction also provides necessary information for microoperation sequencing.
- ▶ A set of microinstructions forms Microprogram.
- ▶ Microprogram can be changed relatively easily by changing the contents of Control Memory (CM).
- ▶ So, Microprogrammed control unit is more flexible than the Hardwired control unit.



- ▶ The instruction is loaded to the Instruction Register (IR).
- ▶ The address field of the microinstruction contains the next address and that to be used when branching condition is satisfied.
- ▶ The Microprogram Counter ( $\mu$ PC) provides the next microinstruction address when no branching is needed.
- ▶ The starting address or branch address is generated by taking condition code, external inputs and CW into consideration.
- ▶ By decoding the microinstruction appropriate control signals, that is, Control Word (CW) is generated.
- ▶ Generally the control memory is ROM, it can not be modified.
- ▶ For flexibility now writeable control memory (WCM) is used.
- ▶ So, a processor with writeable control memory (WCM) in the control unit is said to be dynamically micro-programmable because the control memory content can be altered.

## Advantages

- ▶ Less design complexity
- ▶ Flexible due to WCM
- ▶ A given CPU instruction set can be easily modified by changing the microprogram.

## Disadvantages

- ▶ A microprogram control unit works slow than Hardwired control unit.

## Microprogram

- ❑ Program stored in control memory that generates all the control signals required to execute the instruction set correctly
- ❑ Consists of microinstructions

## Microinstruction

- ❑ Contains a control word and a sequencing word
  - Control Word - All the control information required for one clock cycle
  - Sequencing Word - Information needed to decide the next microinstruction address

## Control Memory (Control Storage: CS)

Storage in the microprogrammed control unit to store the microprogram

## Writeable Control Memory (Writeable Control Storage: WCS)

- ❑ CS whose contents can be modified
  - Allows the microprogram can be changed
  - Instruction set can be changed or modified

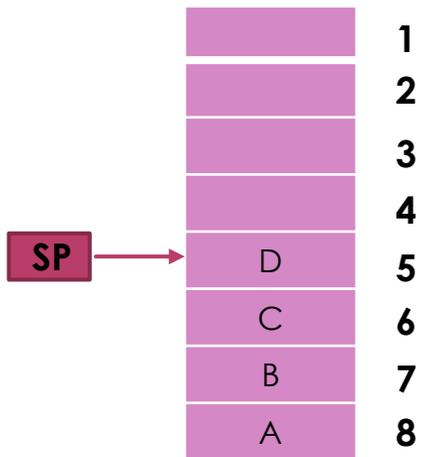
## Dynamic Microprogramming

- ❑ Computer system whose control unit is implemented with a microprogram in WCS
- ❑ Microprogram can be changed by a systems programmer or a user

# STACK

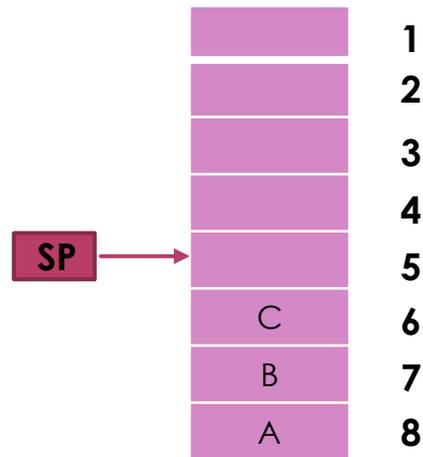
- ▶ Stack is a contiguous memory location or collection of finite number of registers defined by the user that stores information.
- ▶ If it is contiguous memory locations, called memory stack.
- ▶ If it is collection of finite number of registers, called register stack.
- ▶ The stack is a reserved area of the memory where we can store temporary information.
- ▶ The item stored last into the stack will be retrieved first. So, the operation is called LIFO.
- ▶ The register that hold the address for the stack, that is, top of the stack is called Stack pointer (SP).
- ▶ The two operations of a stack are the insertion and deletion of item.
- ▶ The operation of insertion is called PUSH.
- ▶ The operation of deletion is called POP.
- ▶ These operations are simulated by incrementing or decrementing the Stack Pointer register.

- ▶ A stack may be **Full stack** or **Empty stack**.
- ▶ It may be **Descending stack** or **Ascending stack**.



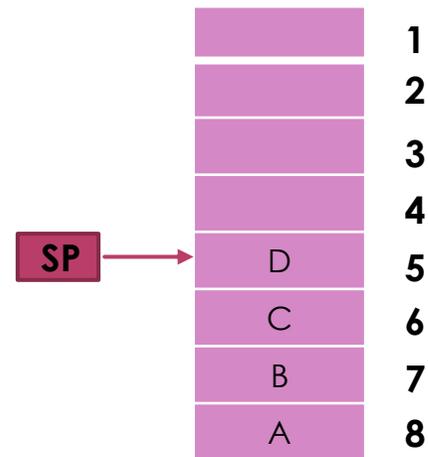
**Full Stack**

$SP \leftarrow SP - 1$   
PUSH



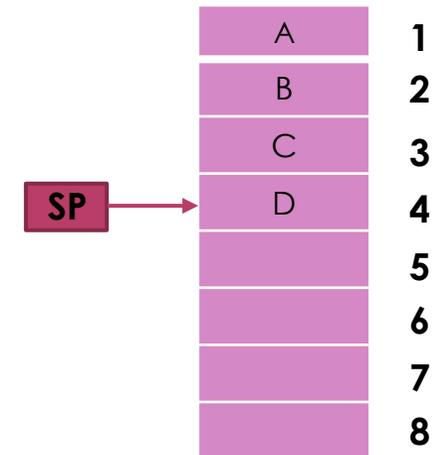
**Empty Stack**

PUSH  
 $SP \leftarrow SP - 1$



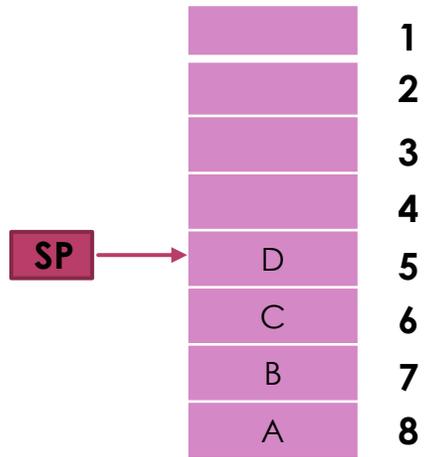
**Descending Stack**

$SP \leftarrow SP - 1$   
PUSH



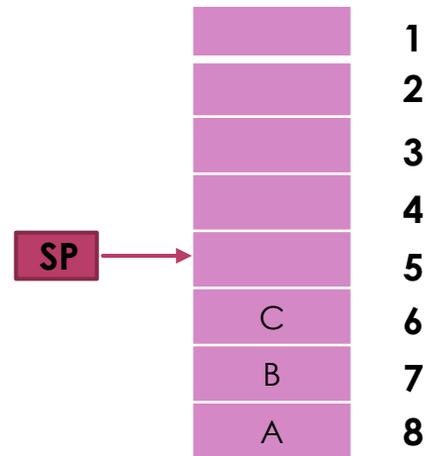
**Ascending Stack**

$SP \leftarrow SP + 1$   
PUSH



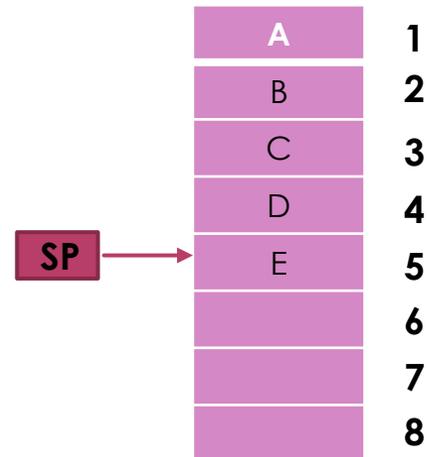
**Full Descending Stack**

$SP \leftarrow SP - 1$   
 PUSH



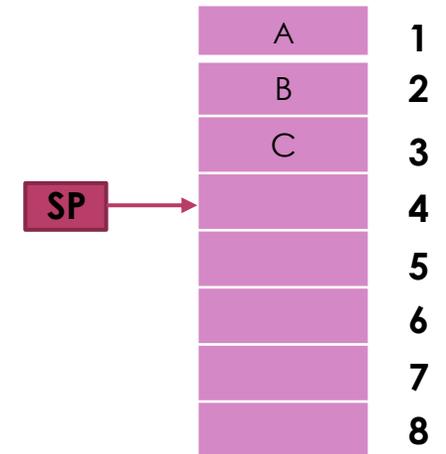
**Empty Descending Stack**

PUSH  
 $SP \leftarrow SP - 1$



**Full Ascending Stack**

$SP \leftarrow SP + 1$   
 PUSH



**Empty Ascending Stack**

PUSH  
 $SP \leftarrow SP + 1$

# Polish Notation

- ▶ The way to write arithmetic expression is known as **notation**.
- ▶ An arithmetic expression can be written in three different but equivalent notations, that is, without changing the essence or output of an expression.
- ▶ These notations are –
  - Infix Notation
  - Prefix (**Polish**) Notation
  - Postfix (**Reverse-Polish**) Notation
- ▶ **Infix**, **Prefix** and **Postfix** notations are three different but equivalent notations of writing algebraic expressions.
- ▶ While writing an arithmetic expression using infix notation, the operator is placed between the operands.
- ▶ For example,  $A+B$ ; here, plus operator is placed between the two operands A and B.
- ▶ Although it is easy to write expressions using infix notation, computers find it difficult to parse as they need a lot of information to evaluate the expression.
- ▶ Information is needed about operator precedence, associativity rules, and brackets which overrides these rules.
- ▶ So, computers work more efficiently with expressions written using prefix or postfix notations.

# Reverse Polish Notation

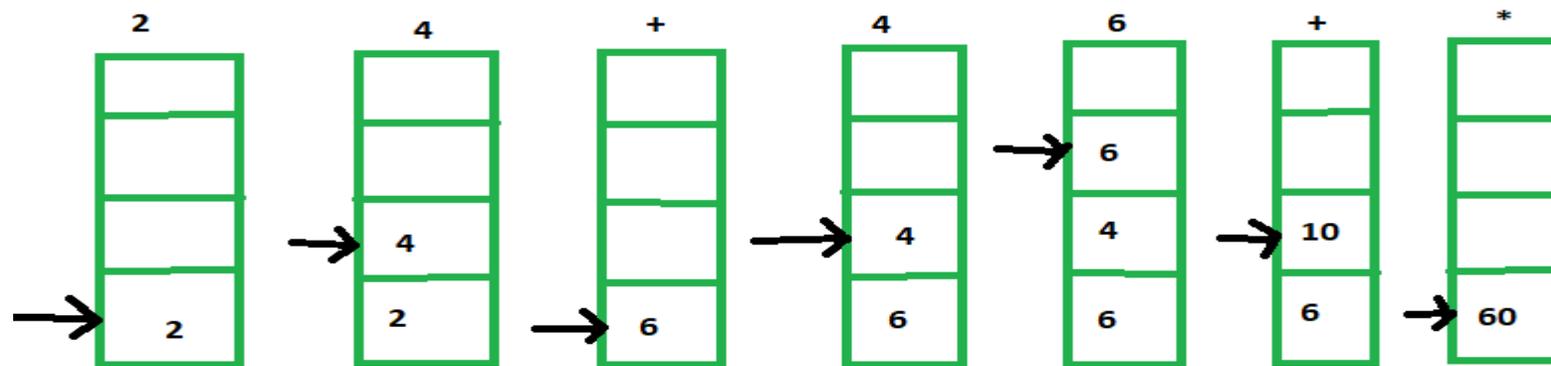
- ▶ **Postfix** notation was given by Jan Łukasiewicz who was a Polish logician, mathematician, and philosopher.
- ▶ His aim was to develop a parenthesis-free prefix notation (also known as **Polish notation**) and a postfix notation which is known as **Reverse Polish Notation or RPN**.
- ▶ In postfix notation, the operator is placed after the operands. For example, if an expression is written as  $A+B$  in infix notation, the same expression can be written as  $AB+$  in postfix notation.
- ▶ The order of evaluation of a postfix expression is always from left to right.
- ▶ The expression  $(A + B) * C$  is written as:  $AB+C*$  in the postfix notation.
- ▶ A postfix operation does not even follow the rules of operator precedence.
- ▶ No parenthesis required.
- ▶ The operator which occurs first in the expression is operated first on the operands.
- ▶ For example, given a postfix notation  $AB+C*$ . While evaluation, addition will be performed prior to multiplication.

- 
- ▶ Now we need to calculate the value of the arithmetic operations by using stack.
  - ▶ The procedure for getting the result is:
    1. Convert the expression in Reverse Polish notation (post-fix notation).
    2. Push the operands into the stack in the order they appear.
    3. When any operator encounter then pop two topmost operands for executing the operation.
    4. After execution push the result into the stack.
    5. After the complete execution of expression the final result remains on the top of the stack.

# Evaluation of Arithmetic Expression

## Example

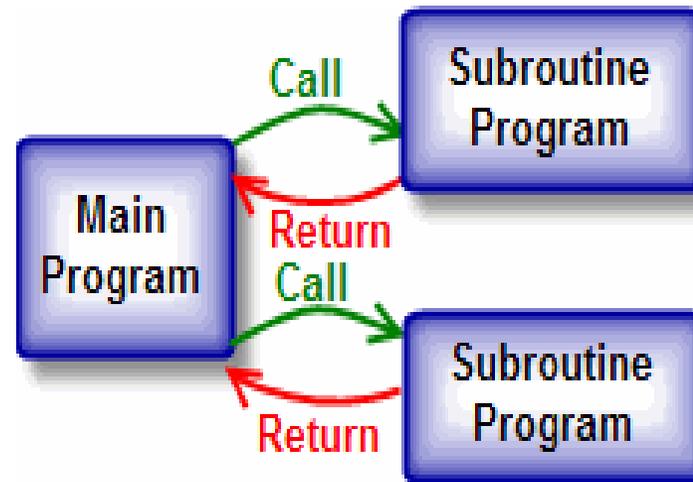
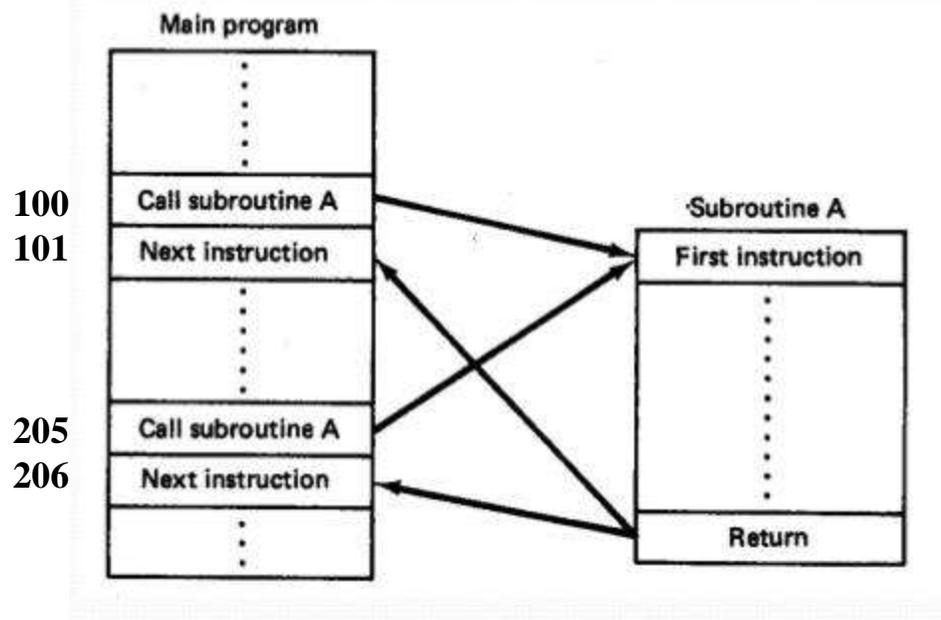
- ▶ Infix notation:  $(2+4) * (4+6)$
- ▶ Post-fix notation:  $2\ 4\ +\ 4\ 6\ +\ *$
- ▶ Result: 60
- ▶ The stack operations for this expression evaluation is shown below:



Stack operations to evaluate  $(2+4)*(4+6)$

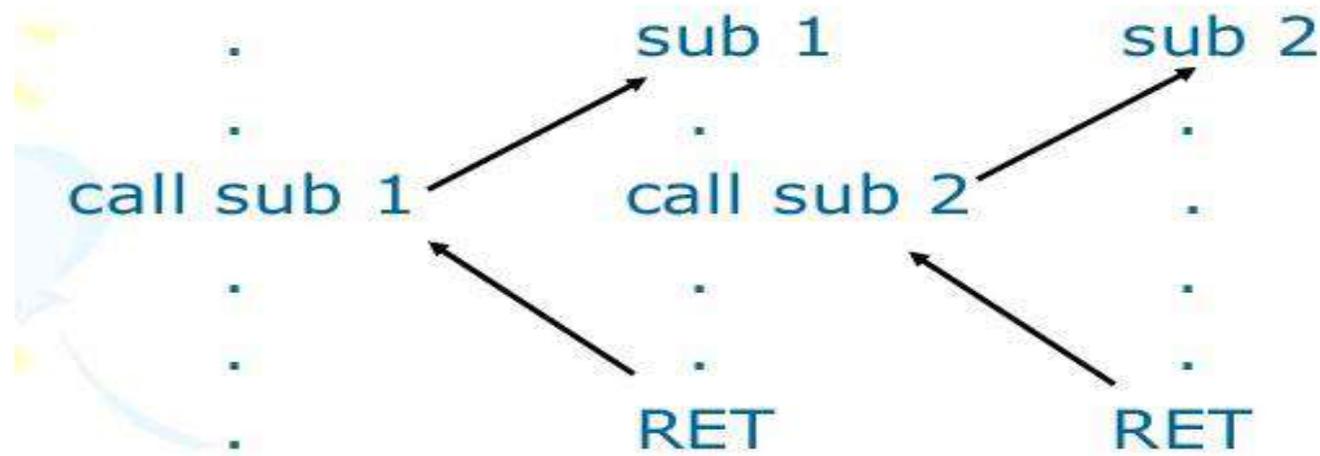
# Subroutine

- ▶ In programming, a **subroutine** is a sequence of instructions that performs a specific task, packaged as a unit.
- ▶ This unit can then be used in programs wherever that particular task should be performed.
- ▶ Subroutines may be defined within programs, or separately in libraries that can be used by many programs.
- ▶ In different programming languages, a subroutine may be called a **routine**, **subprogram**, **function**, **method**, or **procedure**.
- ▶ So, a subroutine is a group of instructions that will be used repeatedly in different locations of the program.
- ▶ Rather than repeat the same instructions several times, they can be grouped into a subroutine that is called from the different locations.

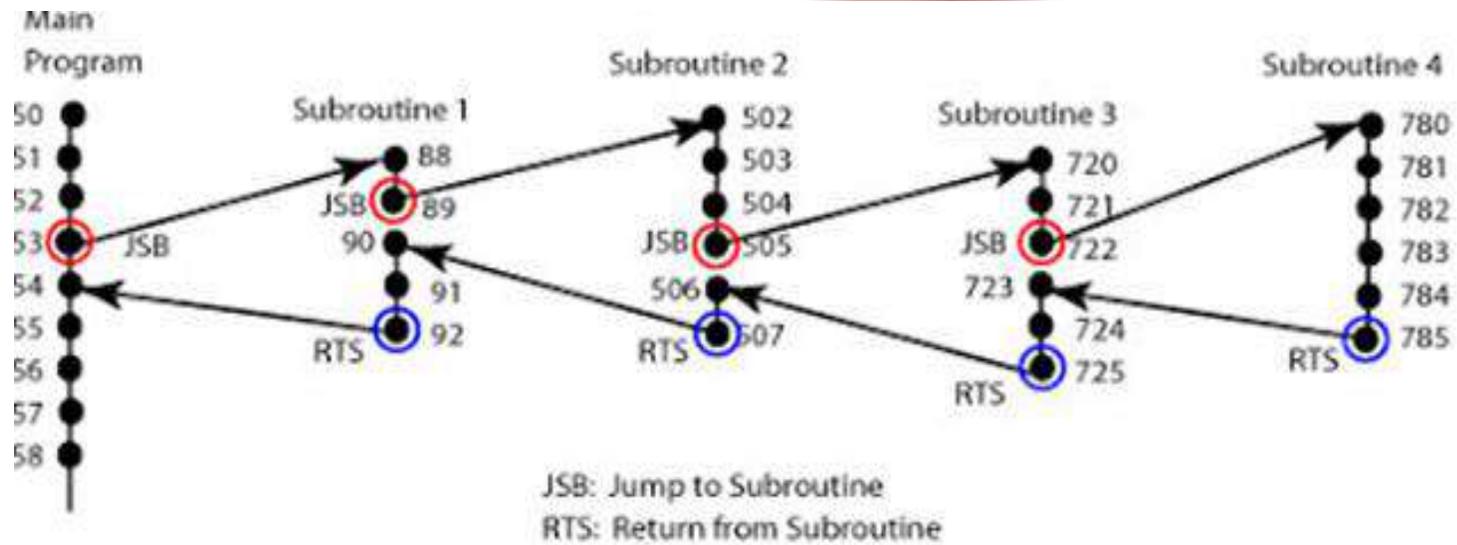


# Nested Subroutine

- ▶ A nested subroutine is a subroutine that is called from other subroutine.
- ▶ Stack operations can be very useful at subroutine entry and exit to avoid losing register contents if other subroutines are called.
- ▶ At the start of a subroutine, content of PC required to be stored on the stack, and at exit they can be popped off again.



# Example



54

90
54

506
90
54

723
506
90
54

# Lecture of Module 3

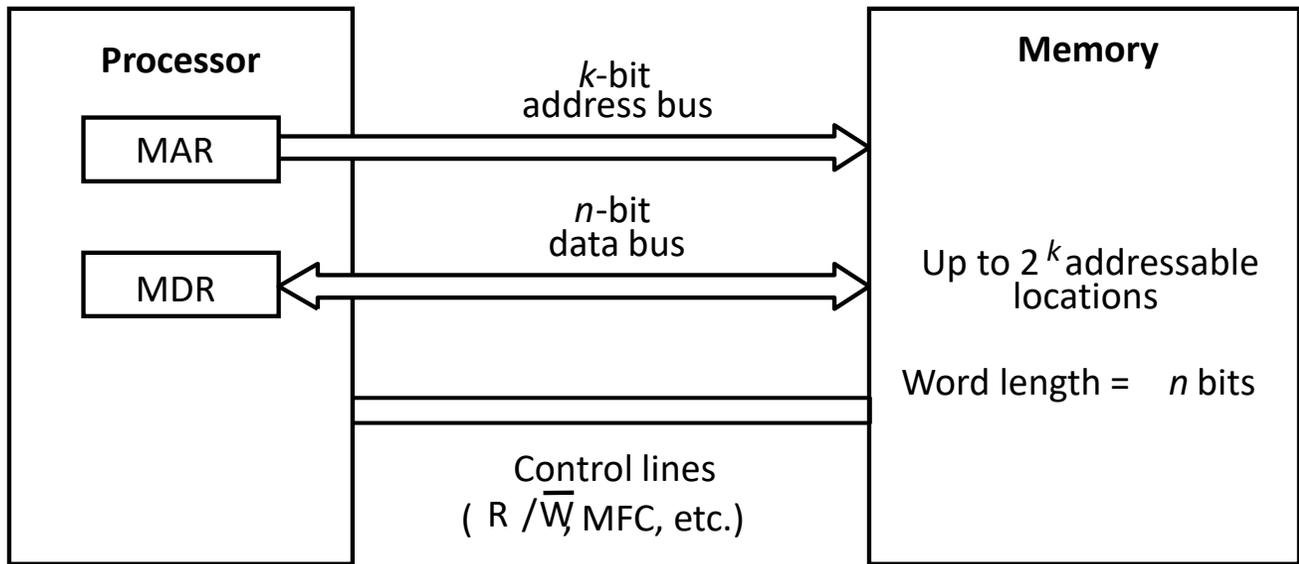
## **Memory Organization**

# Overview

- ▶ **Introduction**
- ▶ **Characteristics of Memory System**
- ▶ **Memory Hierarchy**
- ▶ **Memory Classification**
- ▶ **Semiconductor Memories**
- ▶ **Memory Cell Organization and Operation**
- ▶ **Magnetic Disk**
- ▶ **Magnetic Tape**

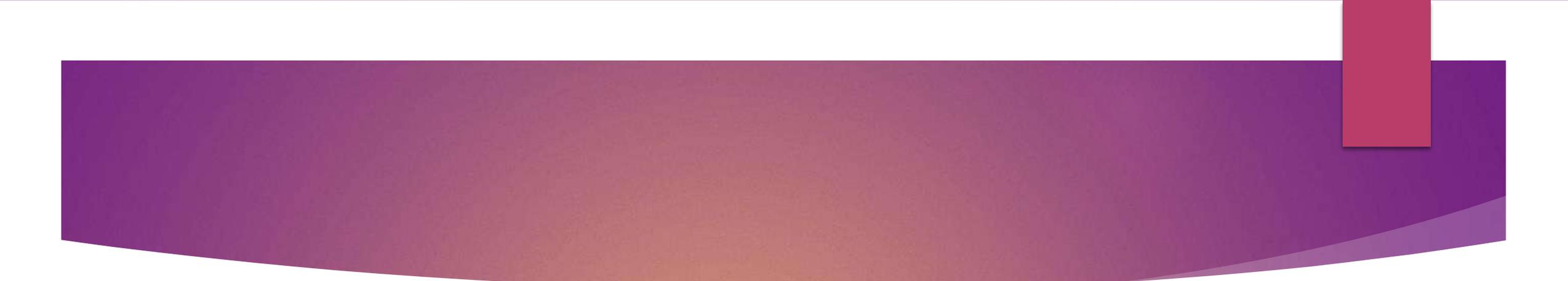
# Introduction

- ▶ One of the major advantages of computer is its storage capacity where huge amount of information can be stored.
- ▶ But how this information is represented and stored?
- ▶ Now we are going to learn about the various data storage devices.
- ▶ A memory is just like a human brain.
- ▶ It is used to store data and instructions.
- ▶ Computer memory is the storage space in the computer, where the data and instructions are stored.
- ▶ Each location of the memory has a unique address, which varies from zero to memory size minus one.
- ▶ For example, if the computer has 64k words, then this memory unit has  $64 * 1024 = 65536$  memory locations. The address of these locations varies from 0 to 65535.



# Characteristics of Memory System

<b>Location</b> Internal (e.g. processor registers, cache, main memory) External (e.g. optical disks, magnetic disks, tapes)	<b>Performance</b> Access time Cycle time Transfer rate
<b>Capacity</b> Number of words Number of bytes	<b>Physical Type</b> Semiconductor Magnetic Optical Magneto-optical
<b>Unit of Transfer</b> Word Block	<b>Physical Characteristics</b> Volatile/nonvolatile Erasable/nonerasable
<b>Access Method</b> Sequential Direct Random Associative	



▶ **Location**

- ▶ Refers to whether memory is internal and external to the computer
- ▶ Internal memory is often equated with main memory
- ▶ Processor requires its own local memory, in the form of registers
- ▶ Cache is another form of internal memory
- ▶ External memory (Secondary Memory) consists of peripheral storage devices that are accessible to the processor via I/O controllers

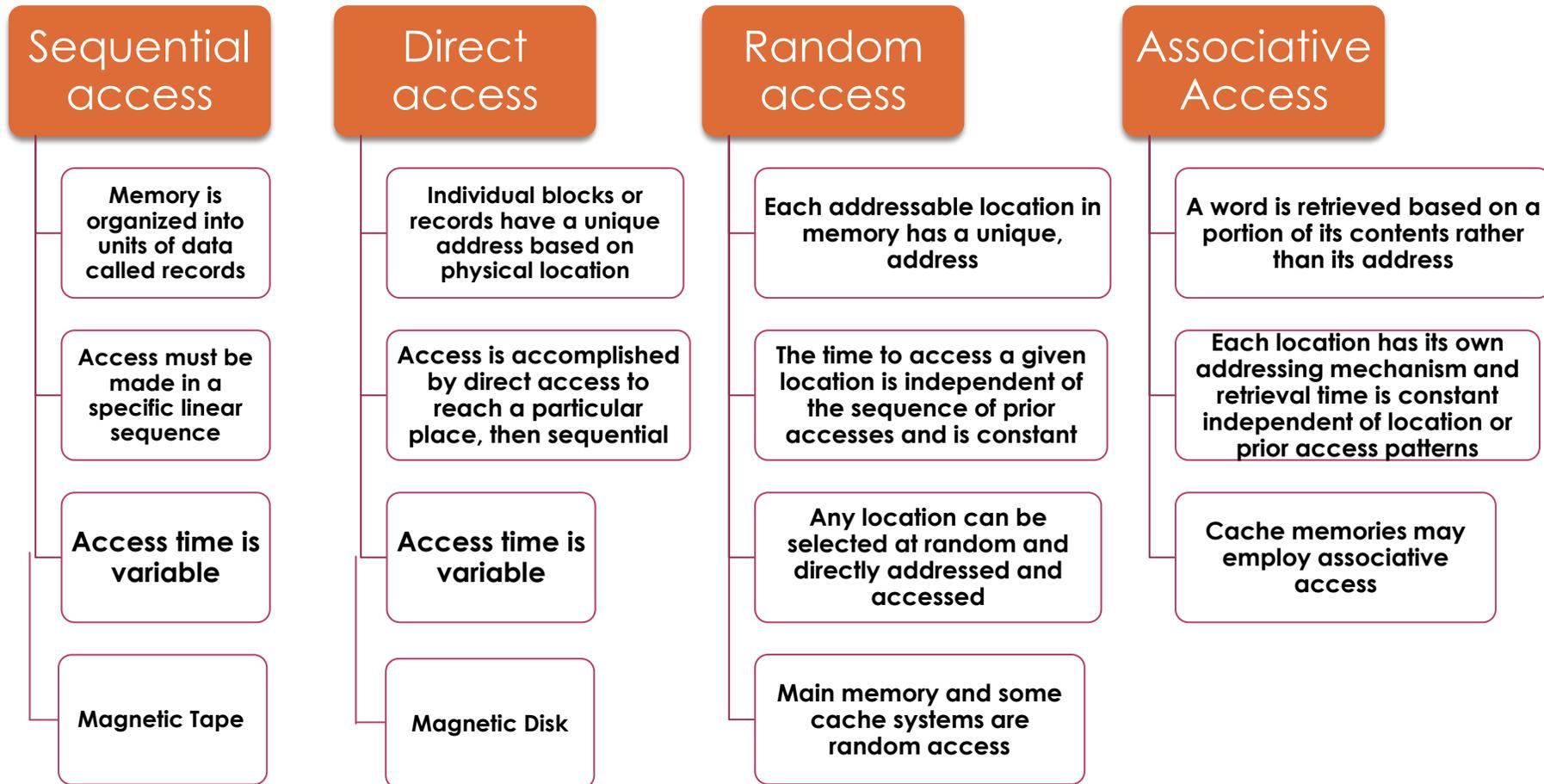
▶ **Capacity**

- ▶ Memory is typically expressed in terms of bytes

▶ **Transfer unit**

- ▶ Byte Addressable, Word Addressable (Main Memory)
- ▶ Block (Secondary Memory)

# Access Methods



# Performance

## Most important characteristics of memory

Three performance parameters are used:

### Access time

For random-access memory it is the time it takes to perform a read or write operation

- For non-random-access memory it is the time it takes to position the read-write mechanism at the desired location and read/write
- Time between initiation of a memory operation and completion

### Memory cycle time

- Access time plus any additional time required before second access can commence
- Time between initiation of two successive memory operations
- Slightly more than the Access time

### Transfer rate

- The rate at which data can be transferred into or out of a memory unit
- For random-access memory it is equal to  $1/(\text{cycle time})$

# Physical Characteristics

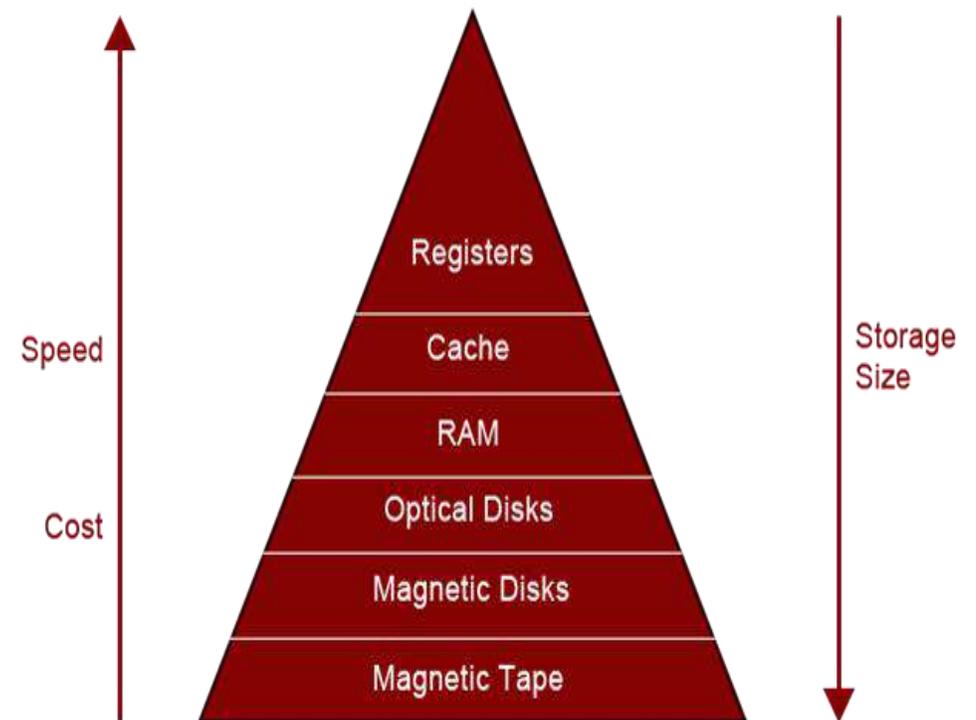
- ▶ Volatile memory
  - ▶ Information decays naturally or is lost when electrical power is switched off
  - ▶ RAM
- ▶ Nonvolatile memory
  - ▶ Once recorded, information remains without deterioration until deliberately changed
  - ▶ No electrical power is needed to retain information
  - ▶ Magnetic-surface memories, ROM

Semiconductor memory may be either volatile or nonvolatile

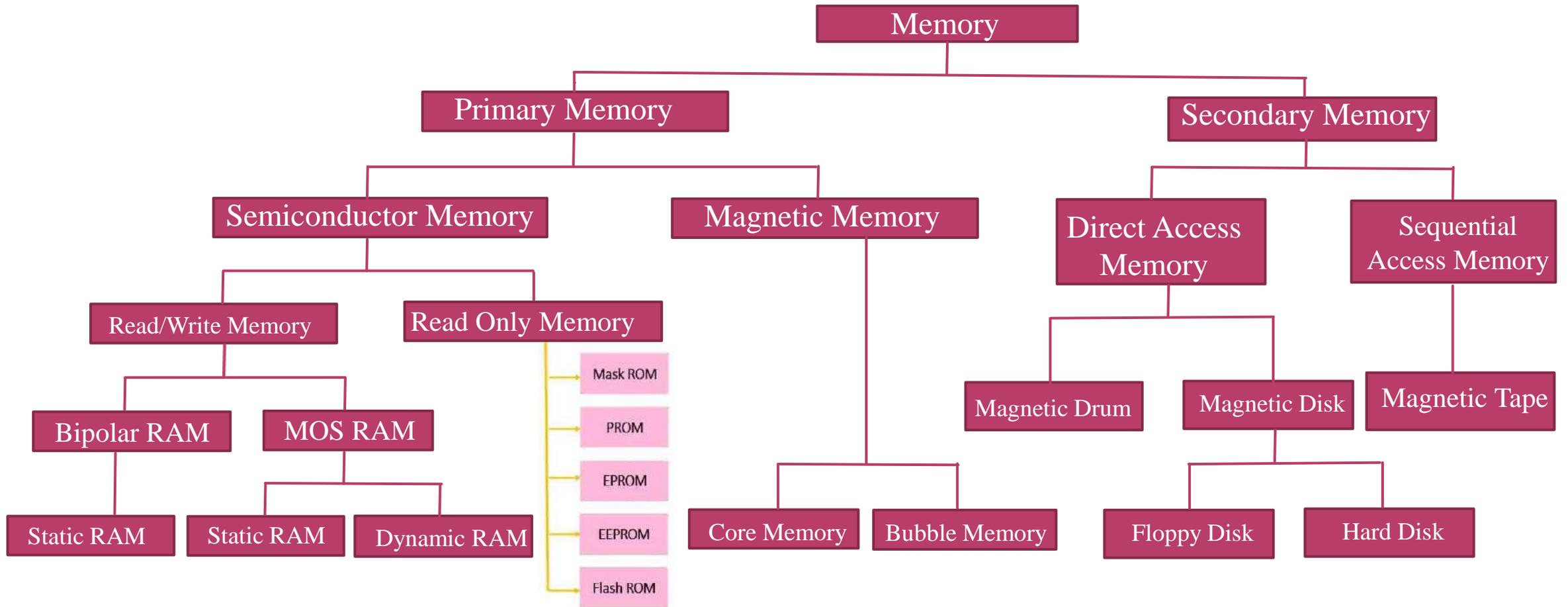
- ▶ Erasable memory
  - Can be altered or erased by the programmer
- ▶ Nonerasable memory
  - ▶ Cannot be altered, except by destroying the storage unit
  - ▶ Semiconductor memory of this type is known as read-only memory (ROM)

# Memory Hierarchy

- ▶ Design constraints on a computer's memory can be summed up by three questions:
  - ▶ How much, how fast, how expensive
- ▶ There is a trade-off among capacity, access time, and cost
  - ▶ Faster access time, greater cost per bit
  - ▶ Greater capacity, smaller cost per bit
  - ▶ Greater capacity, slower access time
- ▶ The way out of the memory dilemma is not to rely on a single memory component or technology, but to employ a memory hierarchy
- ▶ The **memory hierarchy** separates computer storage into a hierarchy based on access time, capacity and cost.



# Memory Classification



# Semiconductor Memory

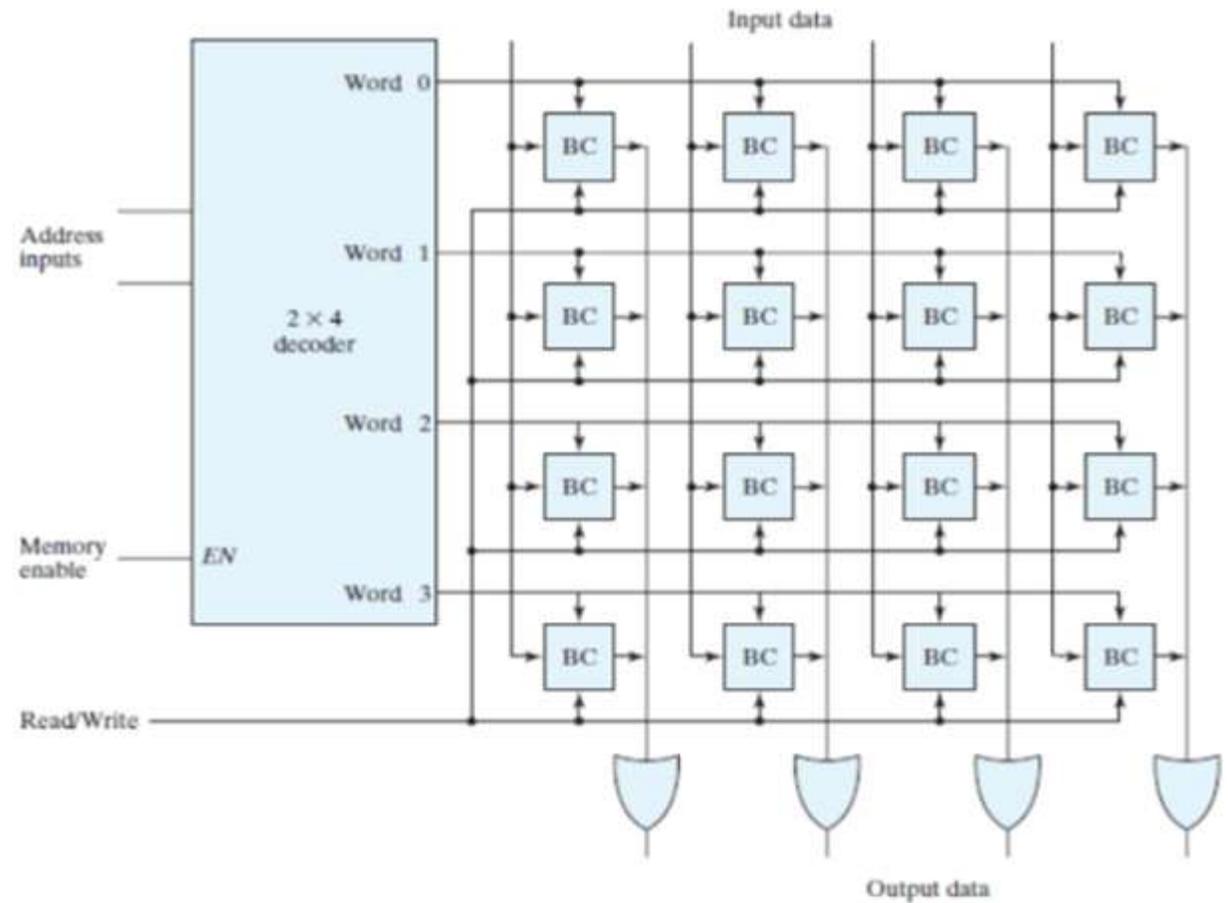
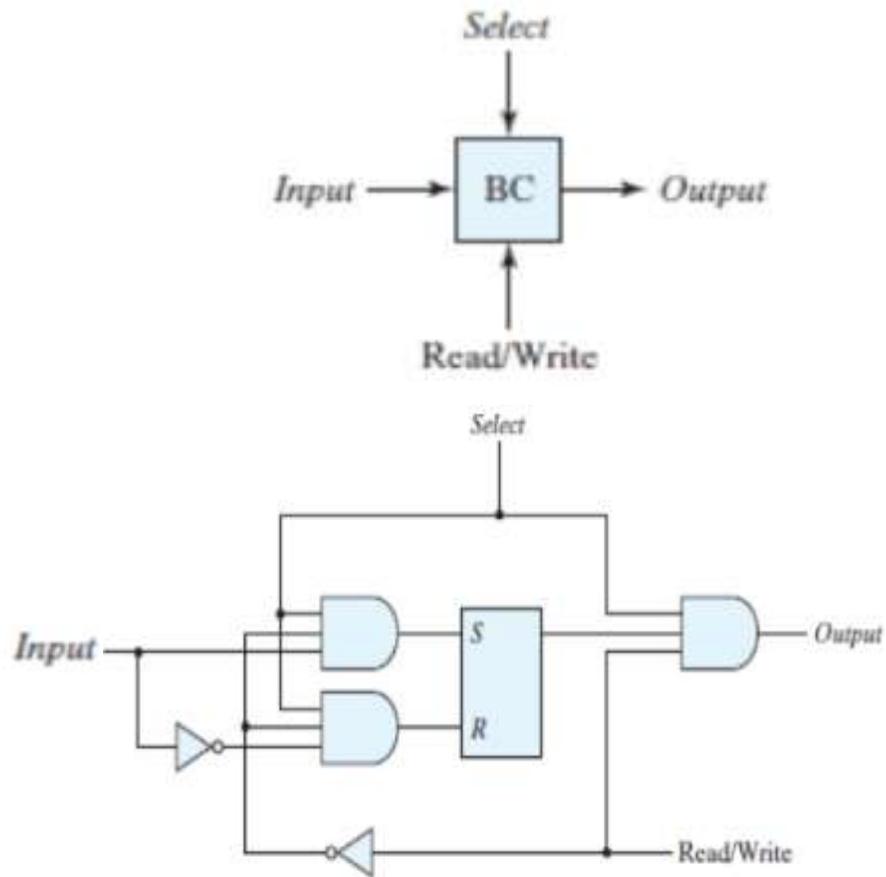
- ▶ **Semiconductor memory** is a semiconductor device used for digital data storage in the computer.
- ▶ It typically refers to Bipolar or MOS memory, where data is stored within memory cell on a silicon integrated circuit (IC) memory chip.
- ▶ There are different types of memory using different semiconductor technologies.
- ▶ The two main types of semiconductor memories are Random Access Memory (RAM) and Read Only Memory (ROM).
- ▶ Most types of semiconductor memory have the property of random access, which means that it takes the same amount of time to access any memory location.
- ▶ So, data can be efficiently accessed in any random order.
- ▶ Semiconductor memory also has much faster access time than other types of data storage.
- ▶ It is used for main memory for the computer (primary storage), to hold data of the computer.

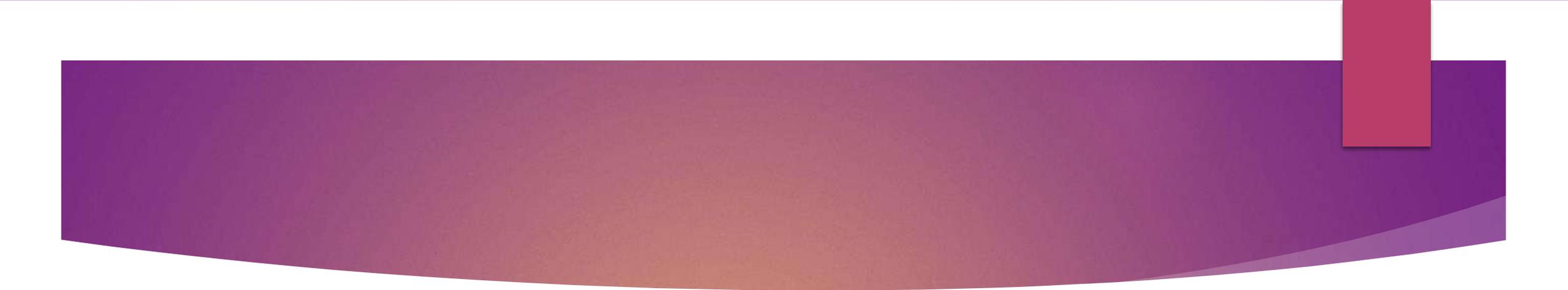
# Random Access Memory (RAM)

- ▶ RAM (Random Access Memory) is the internal memory of the CPU for storing data, program, and results.
- ▶ It is a read/write memory which stores data until the machine is working.
- ▶ As soon as the machine is switched off, data is erased.
- ▶ RAM is volatile, that is, data stored in it is lost when we switch off the computer or if there is a power failure.
- ▶ Hence, a backup Uninterruptible Power Supply (UPS) is often used with computers.
- ▶ RAM is small, both in terms of its physical size and in the amount of data it can hold.
- ▶ Access time in RAM is independent of the address, that is, each storage location inside the memory is as easy to reach as other locations and takes the same amount of time.



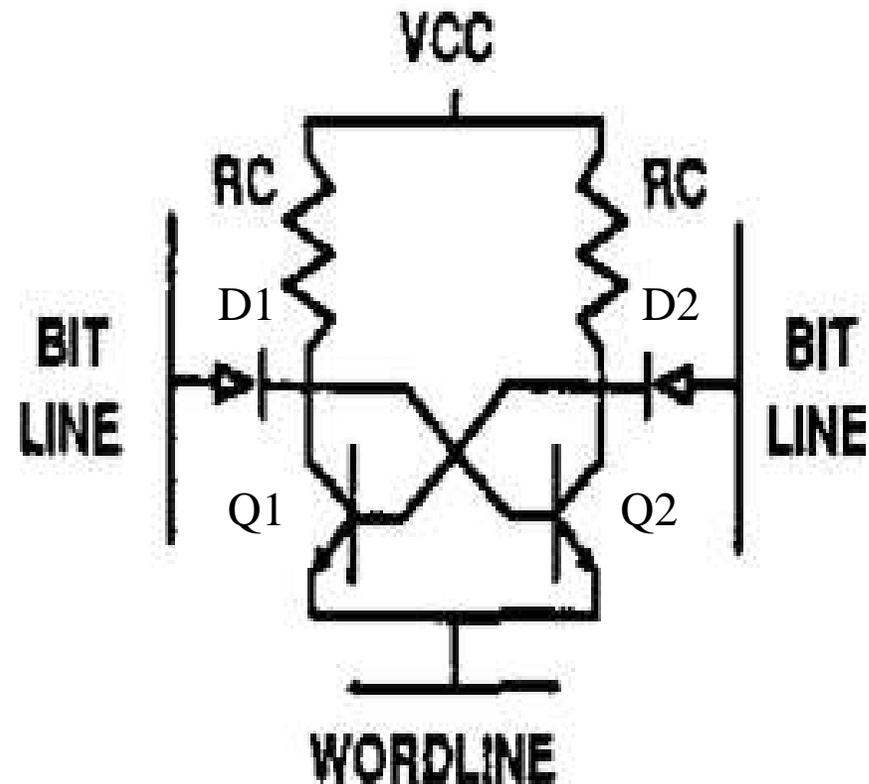
# Structure of RAM



- 
- ▶ According to semiconductor technology, there are two types of RAM.
    - Bipolar RAM - always Static RAM
    - MOS RAM – may be Static RAM or Dynamic RAM
  - ▶ The word **Static** indicates that the memory retains its contents as long as power is being supplied. However, data is lost when the power gets down due to volatile nature.
  - ▶ SRAM chips use a matrix of transistors and no capacitors.
  - ▶ Transistors do not require power to prevent leakage, so SRAM need not be refreshed on a regular basis.
  - ▶ There is extra space in the matrix, hence SRAM uses more chips than DRAM for the same amount of storage space, making the manufacturing costs higher.
  - ▶ SRAM is thus used as cache memory and has very fast access.

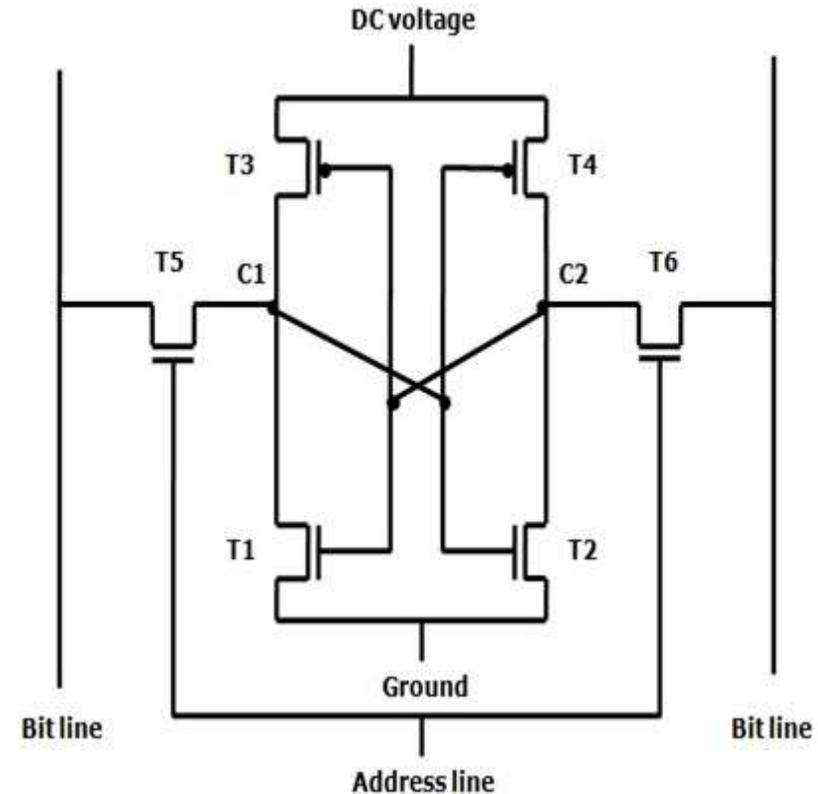
# Static RAM (Bipolar)

- ▶ Bipolar RAM always Static RAM
- ▶ Q1 is on Q2 is off – logic 1 is stored
- ▶ Q2 is on Q1 is off – logic 0 is stored
- ▶ Very high speed of operation
- ▶ But – High power dissipation  
Low bit density  
Complex manufacturing process
- ▶ Used as CACHE memory cell



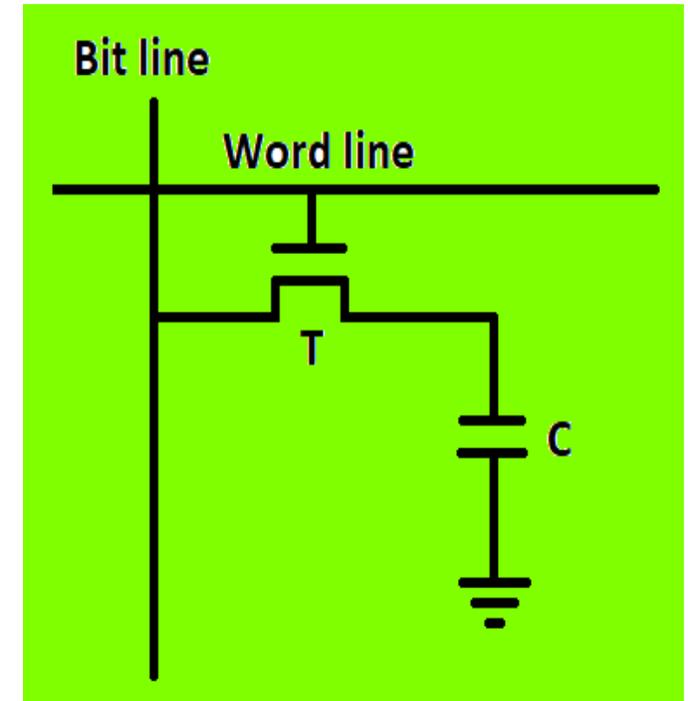
# Static RAM (MOS)

- ▶ This is a random access memory that uses an array of six transistors and is made up of CMOS technology.
- ▶ Its structure consisting of two cross-coupled inverters which hold the data.
- ▶ T1, T4 – Off, T2, T3 – On, So C1 is high and C2 is low, stores logic 1
- ▶ T1, T4 – On, T2, T3 – Off, So C1 is low and C2 is high, stores logic 0
- ▶ All states remain stable until the power voltage is applied with the Direct Current (DC)
- ▶ Higher bit density than Bipolar static RAM
- ▶ Lower power dissipation than Bipolar static RAM
- ▶ Slower speed of operation than Bipolar static RAM



# Dynamic RAM (DRAM)

- ▶ DRAM consists of one transistor and a capacitor.
- ▶ The capacitor stores one bit of data in the form of charge.
- ▶ Presence or absence of charge in a capacitor is interpreted as a binary 1 or 0.
- ▶ It requires refreshing of each memory cell to retain the data as capacitor discharges through time. So, dynamic in nature.
- ▶ For storing, Word line is selected, T is On by applying voltage to the Bit line then the capacitor starts charging.
- ▶ For reading, Word line is selected, the charged capacitor will discharge through Bit line and the sense amplifier compares the voltage in capacitor with threshold voltage.
- ▶ If the voltage more than threshold voltage then logic 1 otherwise logic 0.
- ▶ Requires periodic charge refreshing to maintain data storage.



- ▶ Low cost than SRAM
- ▶ Higher bit density than SRAM
- ▶ So, High storage capacity than SRAM
- ▶ Simple manufacturing process
- ▶ Consumes less power than SRAM
- ▶ Slower speed of operation than SRAM
- ▶ Needs to refresh periodically

Some DRAM chip incorporate refreshing facility and address control circuitry within the memory chip to make it behave similar to Static RAM. So, the dynamic nature of the chip is not visible to the user. This type of DRAM called as **Pseudo-Static RAM (PSRAM)**.

# Types of DRAM

ADRAM – Asynchronous DRAM

EDRAM – Enhanced DRAM

CDRAM – Cache DRAM

RDRAM – Rhombus DRAM

SDRAM – Synchronous DRAM

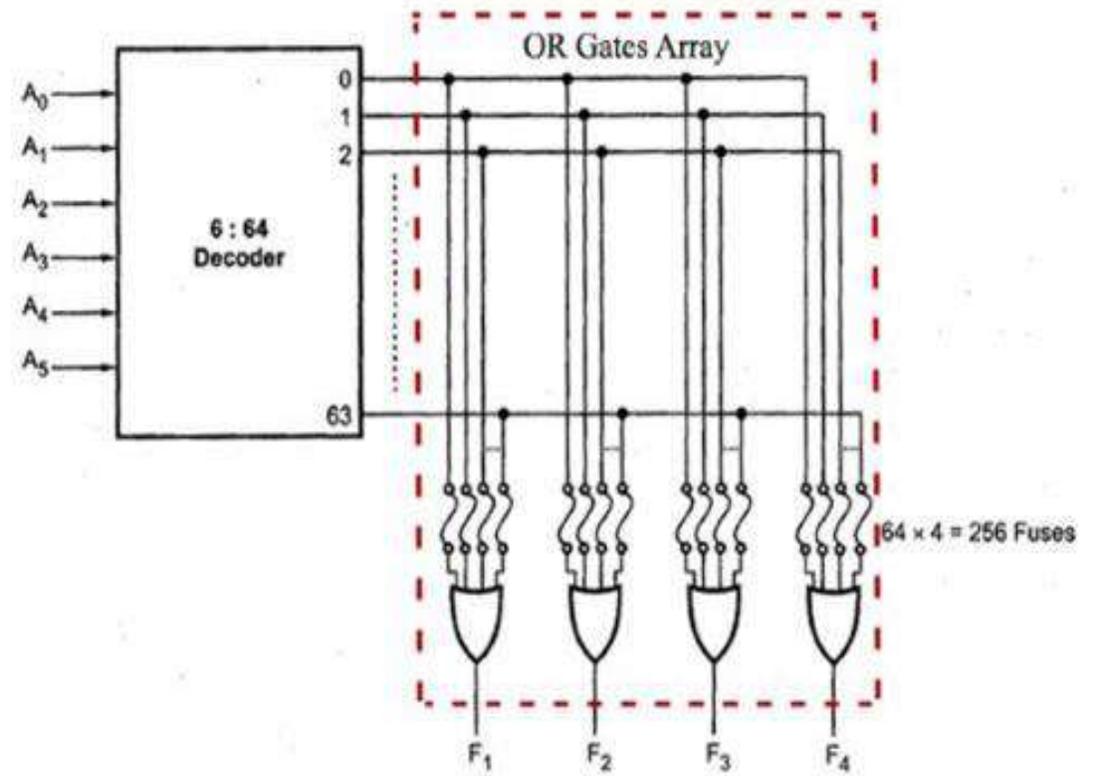
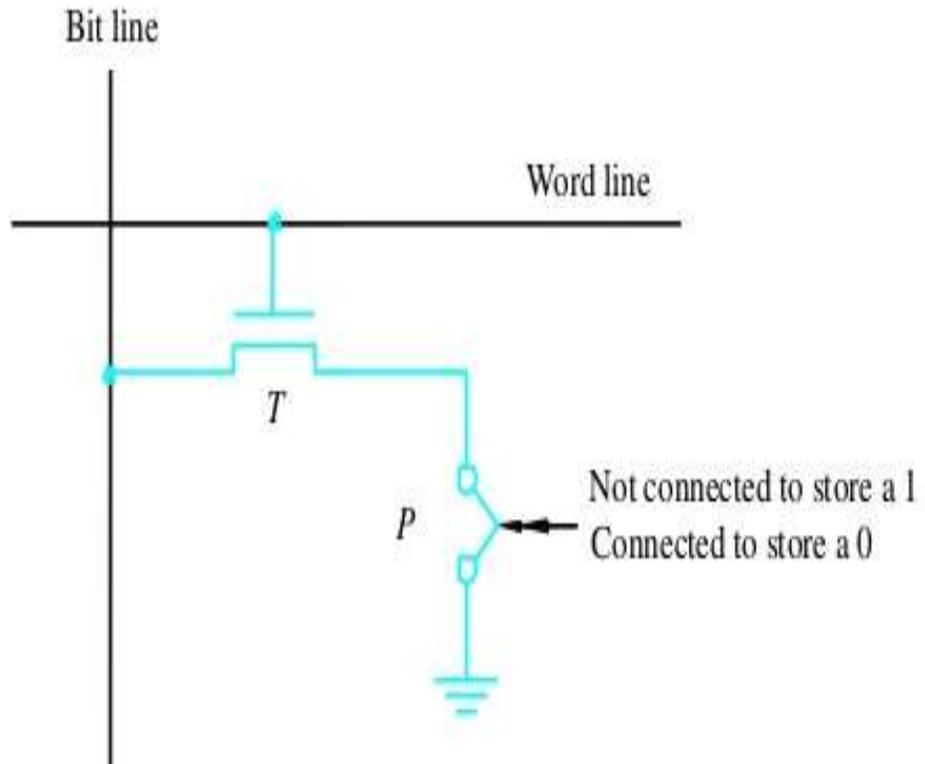
DDR SDRAM – Double Data Rate SDRAM

# Comparison

<u>SRAM</u>	<u>DRAM</u>
1. SRAM has lower access time, so it is faster compared to DRAM.	1. DRAM has higher access time, so it is slower than SRAM.
2. SRAM is costlier than DRAM.	2. DRAM costs less compared to SRAM.
3. SRAM requires constant power supply, which means this type of memory consumes more power.	3. DRAM offers reduced power consumption, due to the fact that the information is stored in the capacitor.
4. Due to complex internal circuitry, less storage capacity is available compared to the same physical size of DRAM memory chip.	4. Due to the small internal circuitry in the one-bit memory cell of DRAM, the large storage capacity is available.
5. SRAM has low packaging density.	5. DRAM has high packaging density.

# Read Only Memory (ROM)

- ▶ ROM stands for **Read Only Memory**.
- ▶ The memory from which we can only read but cannot write on it.
- ▶ This type of memory is non-volatile.
- ▶ The information is stored permanently in such memories during manufacture.
- ▶ Contains a permanent pattern of data that cannot be changed or added.
- ▶ No power source is required to maintain the bit values in memory.
- ▶ Data or program is permanently in main memory and never needs to be loaded from a secondary storage device.
- ▶ Data is actually wired into the chip as part of the fabrication process.
- ▶ A ROM stores such instructions that are required to start a computer.
- ▶ This operation is referred to as **Bootstrap**.



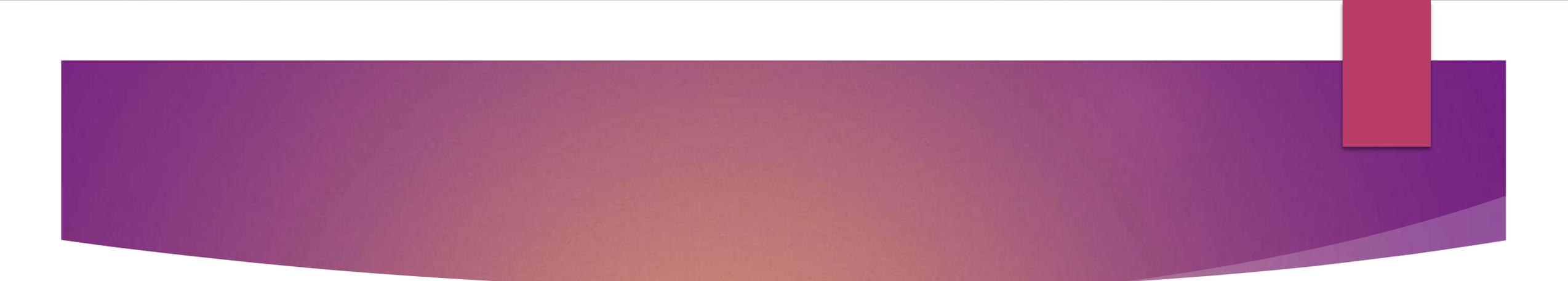
Internal construction of 64x4 ROM

# Types of ROM

- ▶ Mask ROM
- ▶ PROM
- ▶ EPROM
- ▶ EEPROM
- ▶ Flash ROM

**Mask ROM** – In this type of ROM, the specification of the ROM (its contents and their location), is taken by the manufacturer from the customer in tabular form in a specified format and then makes corresponding masks for the paths to produce the desired output . This is costly, as the vendor charges special fee from the customer for making a particular ROM (recommended, only if large quantity of the same ROM is required).

**Uses** – They are used in network operating systems, server operating systems, storing of fonts for laser printers, sound data in electronic musical instruments.

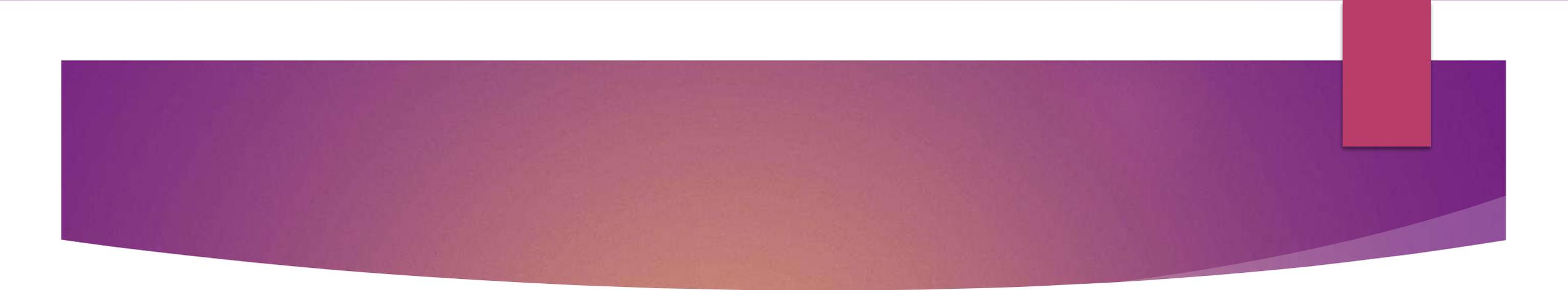


**PROM** – It stands for Programmable Read-Only Memory . It is first prepared as blank memory, and then it is programmed to store the information . The difference between PROM and Mask ROM is that PROM is manufactured as blank memory and programmed after manufacturing, whereas a Mask ROM is programmed during the manufacturing process. To program the PROM, a PROM programmer or PROM burner is used . The process of programming the PROM is called as burning the PROM . Also, the data stored in it cannot be modified, so it is called as one – time programmable device.

**Uses** – They have several different applications, including cell phones, video game consoles, medical devices, and other electronics.

**EPROM** – It stands for Erasable Programmable Read-Only Memory . It overcomes the disadvantage of PROM that once programmed, the fixed pattern is permanent and cannot be altered . If a bit pattern has been established, the PROM becomes unusable, if the bit pattern has to be changed .This problem has been overcome by the EPROM, as when the EPROM is placed under a special ultraviolet light for a length of time, the shortwave radiation makes the EPROM return to its initial state, which then can be programmed accordingly . Again for erasing the content, PROM programmer or PROM burner is used.

**Uses** – Before the advent of EEPROMs, some micro-controllers, like some versions of Intel 8048, the Freescale 68HC11 used EPROM to store their program .

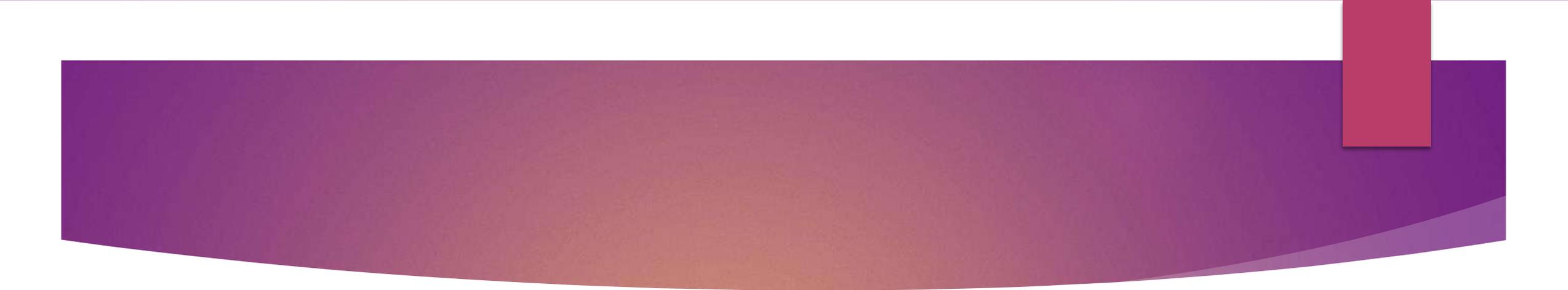


**EEPROM** – It stands for Electrically Erasable Programmable Read-Only Memory . It is similar to EPROM, except that in this, the EEPROM is returned to its initial state by application of an electrical signal, in place of ultraviolet light . Thus, it provides the ease of erasing, as this can be done, even if the memory is positioned in the computer. It erases or writes one byte of data at a time .

**Uses** – It is used for storing the computer system BIOS.

**Flash ROM** – It is an enhanced version of EEPROM .The difference between EEPROM and Flash ROM is that in EEPROM, only 1 byte of data can be deleted or written at a particular time, whereas, in flash memory, blocks of data (usually 512 bytes) can be deleted or written at a particular time . So, Flash ROM is much faster than EEPROM .

**Uses** – Many modern PCs have their BIOS stored on a flash memory chip, called as flash BIOS and they are also used in modems as well.



▶ **Advantages of ROM**

- It is non-volatile, meaning data which was set by the manufacture will function as expected.
- Due to being static, they don't need a refreshing time.
- In comparison to RAM, the circuitry is simpler.
- Data can be stored permanently.

▶ **Disadvantages of ROM:**

- ROM is a read only memory unit, so it can't be modified.
- If any changes are required, it's not possible.

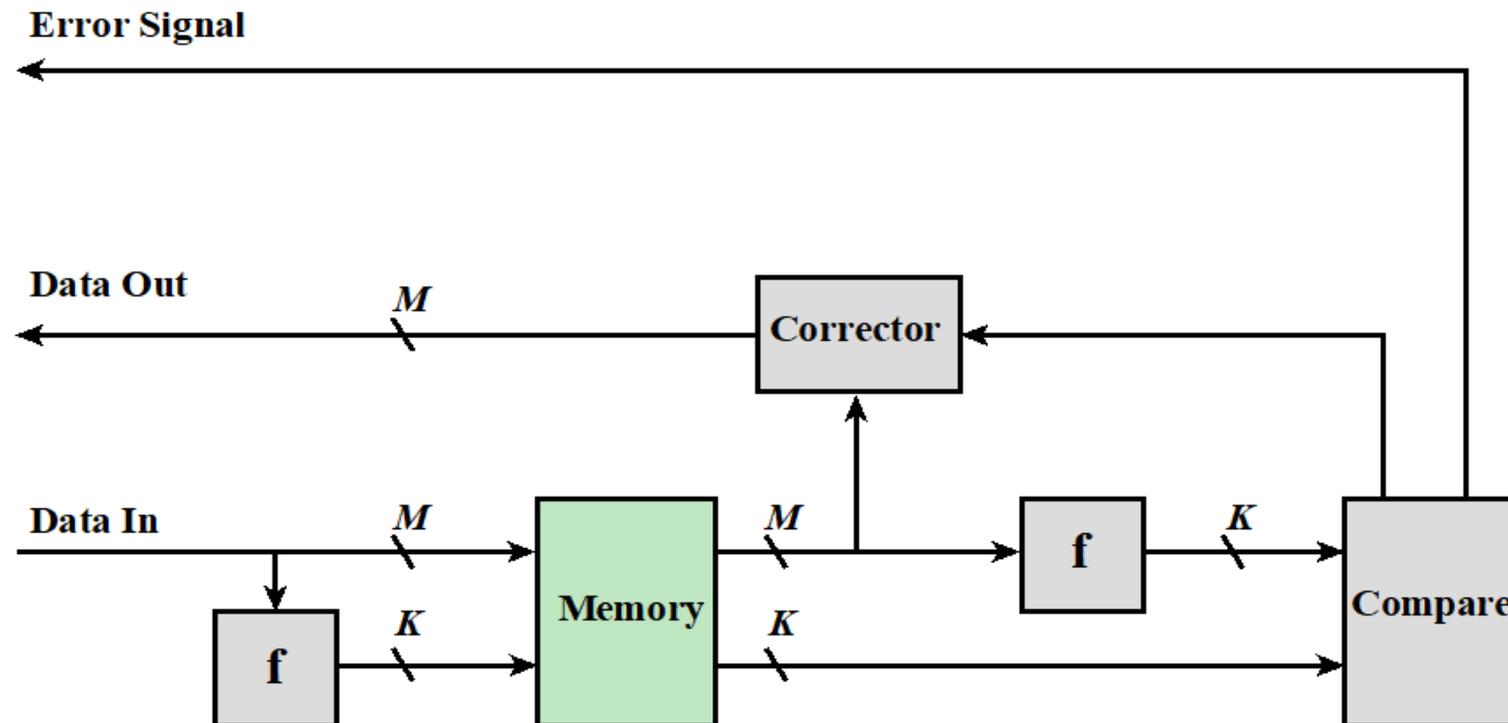
# Comparison

<b>RAM</b>	<b>ROM</b>
Random Access Memory	Read Only Memory
Volatile memory	Non volatile memory
If the system is turned off, the information will be deleted	If the system is turned off, the information it carries will still be on the memory, meaning that the system can retrieve it again when the system is switched on
Requires power to store data	Doesn't require power to store data
RAM is a temporary storage unit to store files	ROM is used to store BIOS which don't change
Chips often range from 1 to 256 GB	Chips often range from 4 to 8 MB
Temporary memory	Permanent memory
Complex Circuit	Simpler Circuit

# Error Detection and Correction

- ▶ Two types of error may be encountered in semiconductor memory.
- ▶ **Hard Failure**
  - ▶ Permanent physical defect so that memory cell or cells affected cannot reliably store data but become stuck at 0 or 1 or switch erratically between 0 and 1.
  - ▶ Can be caused by:
    - ▶ Harsh environmental effect
    - ▶ Manufacturing defects
    - ▶ Wear and tear etc.
- ▶ **Soft Error**
  - ▶ It is a random, non-destructive event that alters the contents of one or more memory cells without damaging the memory.
  - ▶ Can be caused by:
    - ▶ Power supply problems
    - ▶ Alpha particles results from radioactive decay

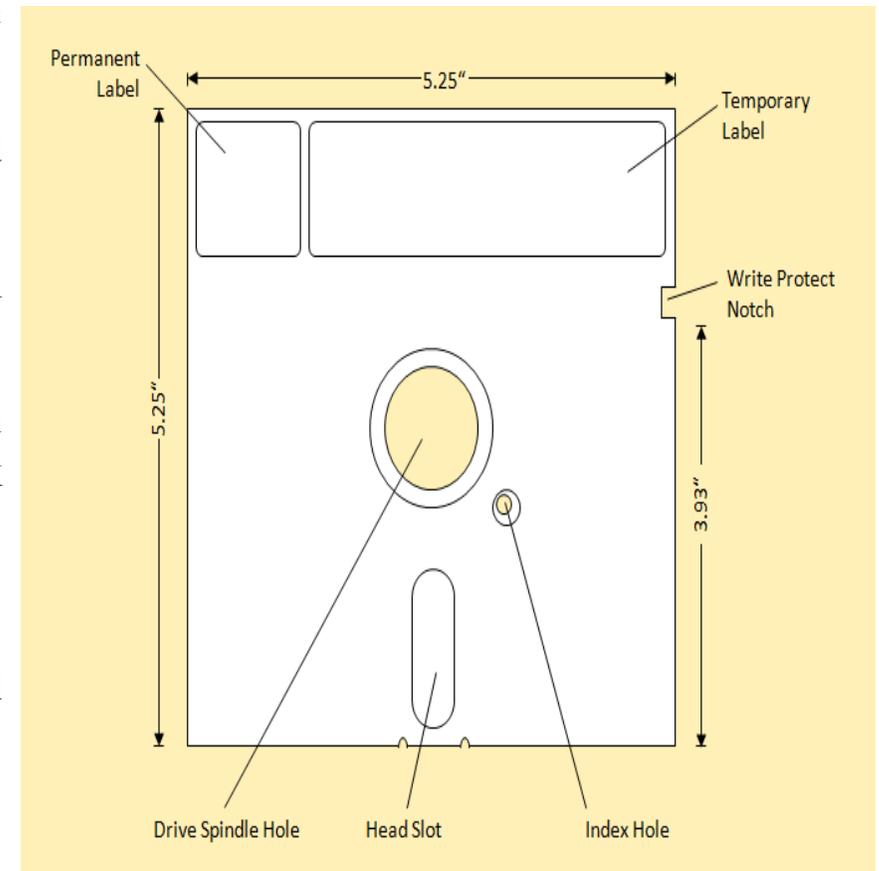
So, error detection and correction is required.



# Magnetic Disk

- ▶ **Magnetic Disk** is type of secondary memory which is a flat disc covered with magnetic coating to hold information.
- ▶ A disk is a circular *plate* constructed of nonmagnetic material, called the *substrate*, coated with a magnetizable material.
  - ▶ Traditionally the substrate has been an aluminium or aluminium alloy material.
  - ▶ Recently glass substrates have been introduced for improvement in the uniformity of the magnetic film surface to increase disk reliability.
- ▶ It is used to store various programs and files.
- ▶ The polarized information in one direction is represented by 1, and other direction is indicated by 0.
- ▶ There are two types of Magnetic Disk memory

**Floppy Disk** and **Hard Disk**



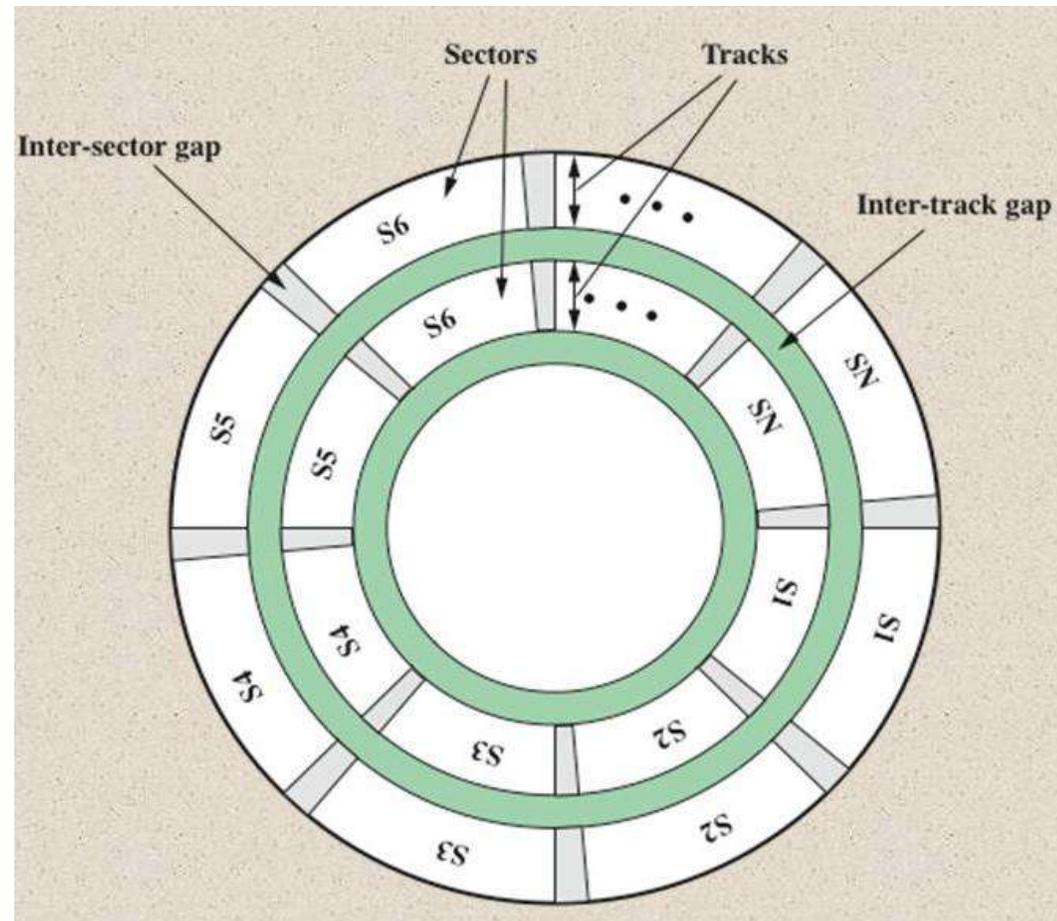
# Floppy Disk

## Hard Sectoring

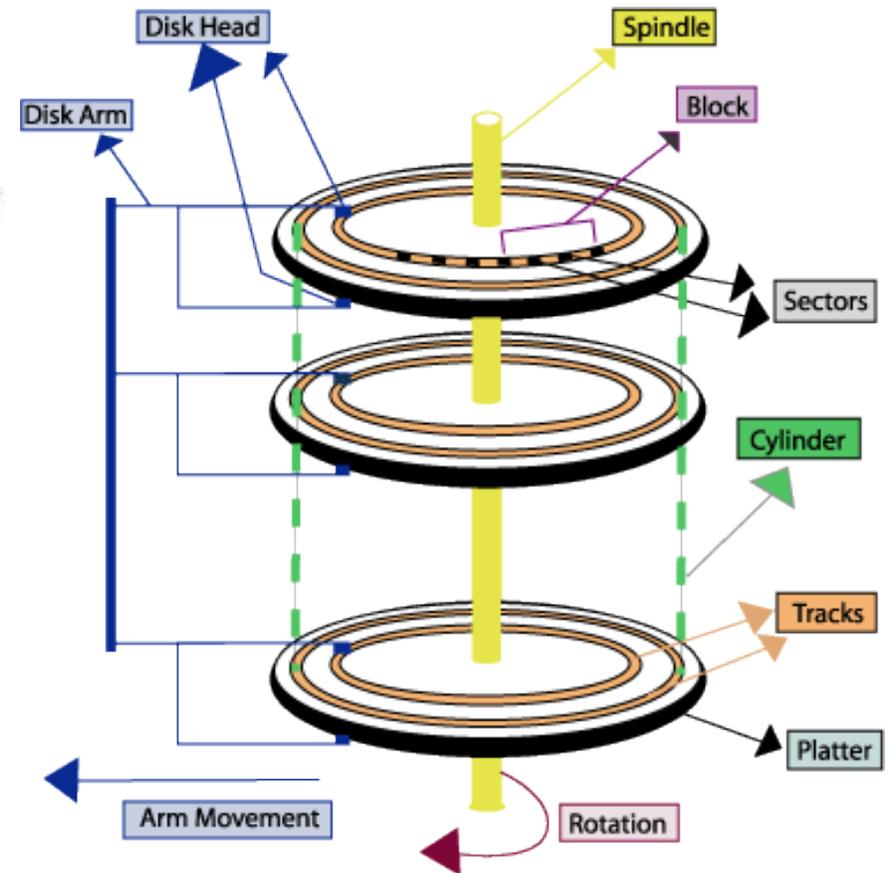
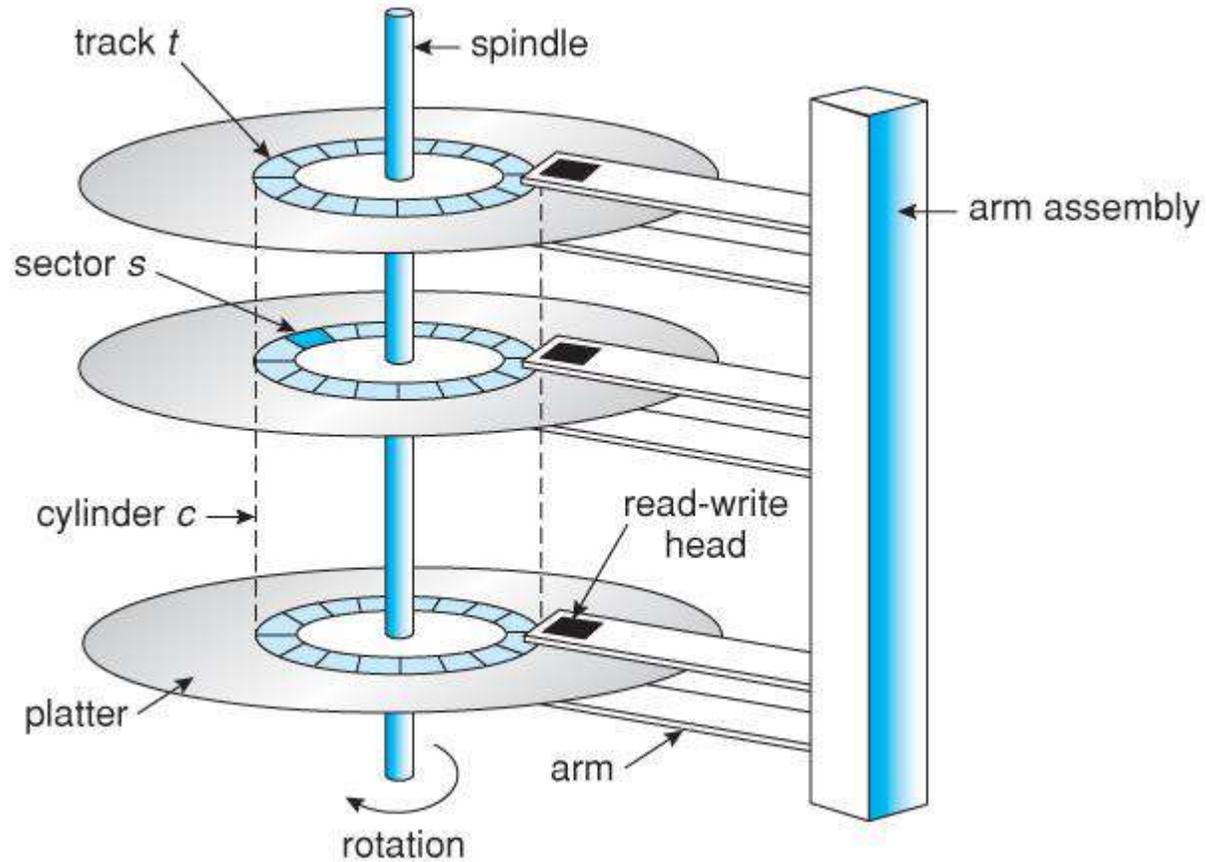
- ▶ The number of Sectors on each Track is physically fixed while manufacturing.
- ▶ The beginning of each sector is identified by a sector hole punched on the disk.
- ▶ Sector is fixed, no flexibility.

## Soft Sectoring

- ▶ Number Sectors per Track chosen by software.
- ▶ Information about sector is written on each sector.
- ▶ Flexible



# Hard Disk



# Disk structures

## ▶ Tracks-

- ▶ circular areas of the disk
- ▶ Data first written to outer most track
- ▶ Over 1000 on a hard disk

## ▶ Sectors-

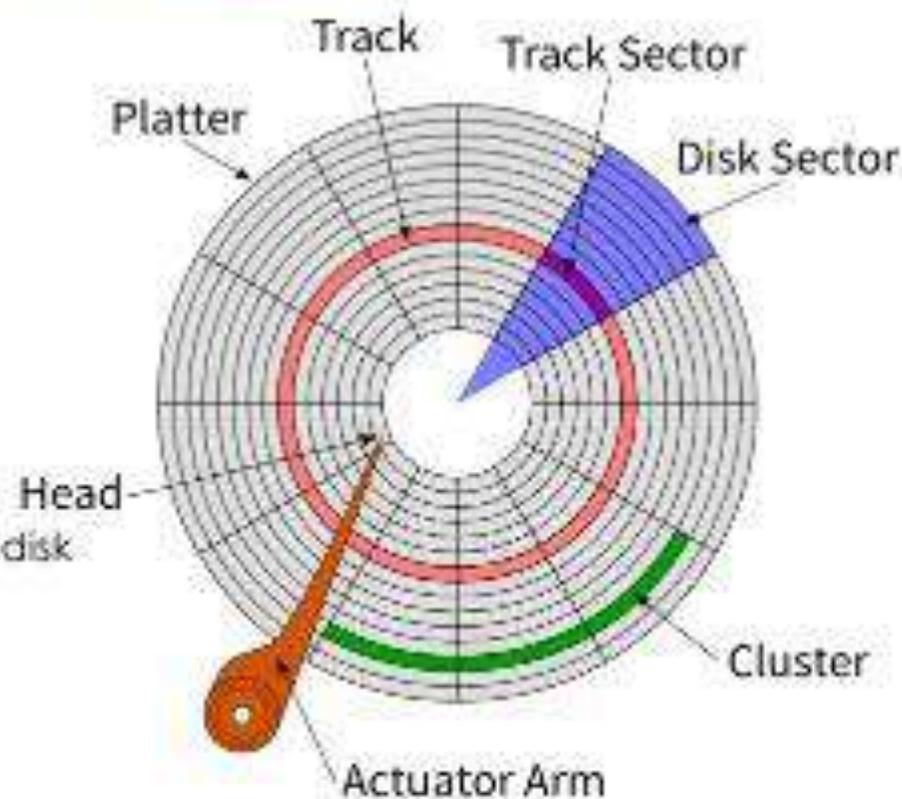
- ▶ Divides tracks sections
- ▶ Sectors are the smallest physical storage units on a disk-
- ▶ Each sector stores 512 bytes of data

## ▶ Cylinders -

- ▶ Logical groupings of the same track on each disk surface in a disk unit
- ▶ Track create cylinder inside the hard disk.

## ▶ Clusters (Blocks) -

- ▶ Groups of sectors used by operating system
- ▶ 64 sectors in one cluster



- 
- ▶ Magnetic disk are less expensive than RAM and can store large amounts of data.
  - ▶ Data access rate is slower than main memory.
  - ▶ Data can be modified or can be deleted easily in the magnetic disk memory.

### Advantages:

These are economical memory.

The easy and direct access of data possible.

It can store large amounts of data.

It has better data transfer rate than magnetic tapes.

It has less prone to corruption of data as compared to tapes.

### Disadvantages:

More expensive than magnetic tape memories.

It need clean and dust free environment to store.

These are not suitable for sequential access.

# Disk Performance Parameters

- ▶ When the disk drive is operating the disk rotate at constant speed.
- ▶ To read or write the head must be positioned at the desired track and at the beginning of the desired sector on the track.
  - ▶ Track selection involves moving the head in a movable-head system or electronically selecting one head on a fixed-head system.
  - ▶ Once the track is selected, the disk controller waits until the appropriate sector rotates to line up with the head.
- ▶ **Seek time**
  - ▶ On a movable-head system, the time it takes to position the head at the proper track.
  - ▶ For fixed head system, it is zero.
  - ▶ **Average Seek time is  $N/3$** , where  $N$  = Number of Track or Cylinder.

## ▶ **Rotational delay (*latency*)**

- ▶ After finding the Track, the time it takes to reach the head at the beginning of the required sector.
- ▶ **Average Latency time is half of the rotation time.**

## ▶ **Access time**

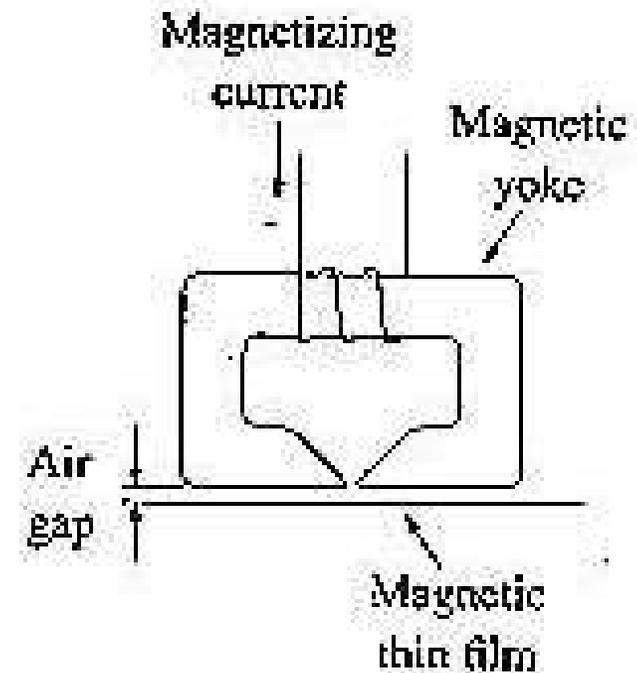
- ▶ The sum of the seek time and the rotational delay (latency).
- ▶ The time it takes to get into position to read or write.
- ▶ So, **Access time = Seek time + Latency time**

## ▶ **Transfer time**

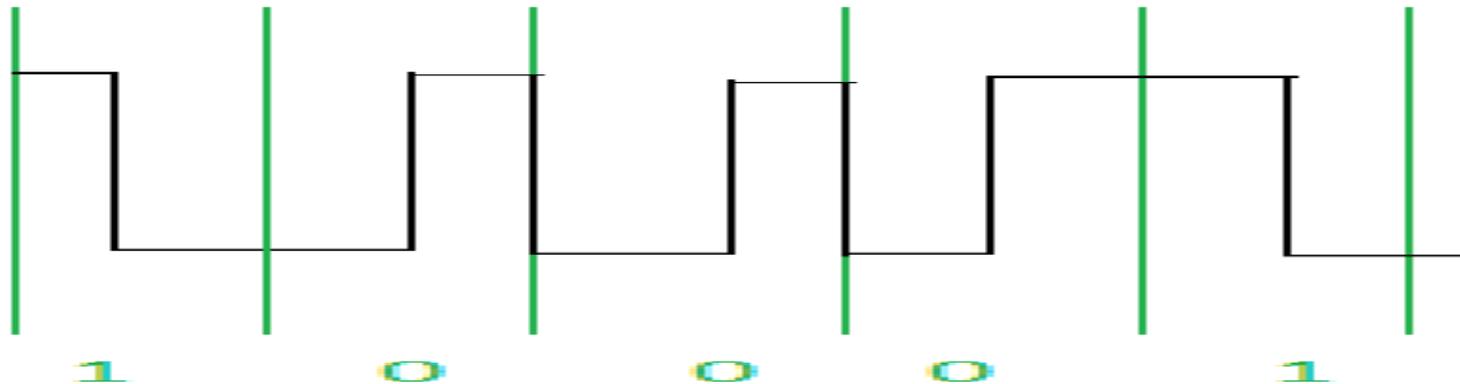
- ▶ Once the head is in position, the read or write operation is then performed as the sector is under the head.
- ▶ Time required to move the data from secondary storage device to the processor memory.

# Read/Write Mechanism

- ▶ The magnetic surface is placed just below the yoke coil.
- ▶ The magnetic surface is mounted on a rotatory device and rotates at a uniform speed.
- ▶ Digital Information can be stored on Magnetic film by applying current pulses of suitable polarity to the magnetizing coils.
- ▶ This causes the magnetization of the film in the area immediately underneath the head by inducing magnetic field just below the yoke.
- ▶ According to the generated flux polarization the bit pattern is stored over the surface.
- ▶ The same method is used for reading the information.



- ▶ For reading, when head reaches at the required place over the magnetic surface from where an information to be retrieved, it senses the polarity of the magnetic field present at that particular spot and sends appropriate signal through the coil.
- ▶ This particular technique for reading/writing is known as **Manchester Encoding**.



# Problem:

A disk system has 16 data recording surfaces with 1024 tracks per surface. There are 16 sectors per track, each containing 1024 bytes. The diameter of inner cylinder is 6 inches and outer cylinder is 10 inches. Find out

- Capacity of disk
- Transfer rate if rotational speed is 3000 rpm
- Latency time and average latency time
- Track density
- Maximum bit density (linear and areal)

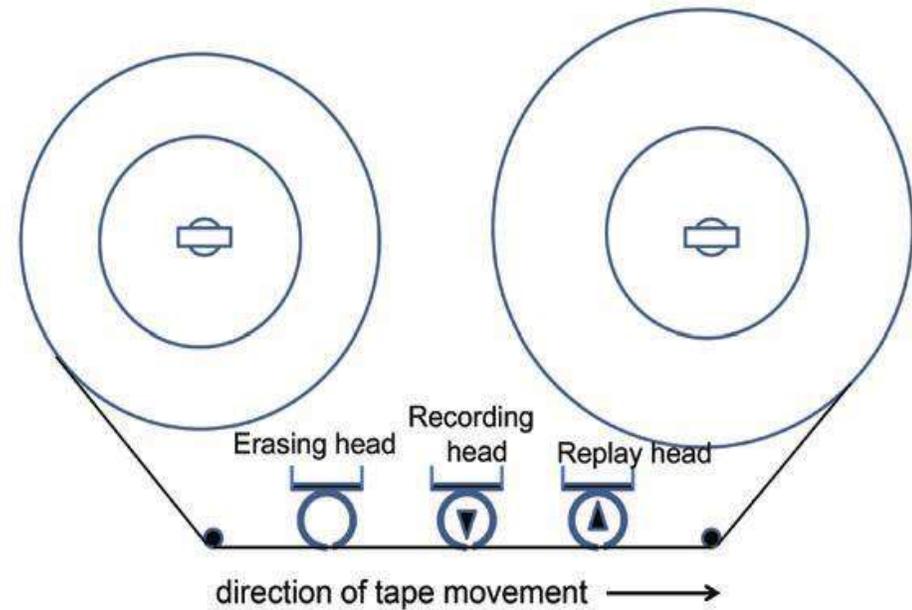
# Magnetic Tape



- ▶ Tape systems use the same reading and recording techniques as disk systems.
- ▶ The tape is made of a thin magnetizable coating on a long narrow strip of plastic film.
- ▶ Coating may consist of particles of pure metal in special binders or vapor-plated metal films.
- ▶ Data on the tape are structured as a number of parallel tracks running lengthwise.
- ▶ Reading and writing is serial and data are laid out as a sequence of bits along each track.
- ▶ Data are read and written in contiguous blocks called *physical records*.



- ▶ Blocks on the tape are separated by gaps referred to as *inter-record gaps*.
- ▶ Magnetic tape is usually recorded on only one side.
- ▶ The opposite side is a *substrate* to give the tape strength and flexibility.
- ▶ Data read/write speed is slower because of sequential access.
- ▶ The width of the ribbon varies from 4mm to 1 Inch and it has storage capacity 100 MB to 200 GB.





► **Advantages:**

1. These are inexpensive, that is, low cost memories.
2. It provides backup or archival storage.
3. It can be used for larger files.
4. It can be used for copying from disk files.
5. It is a reusable memory.

► **Disadvantages:**

1. Sequential access is the disadvantage, means it does not allow access randomly or directly.
2. Data access rate is slower.
3. It requires caring to store, that is, vulnerable humidity, dust free, and suitable environment.
4. It stored data cannot be easily updated or modified, that is, difficult to make updates on data.

# Overview

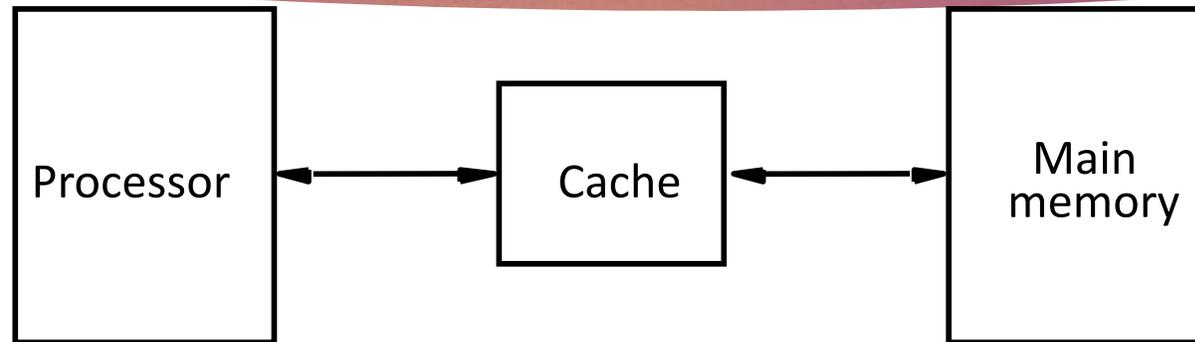
- ▶ **Introduction**
- ▶ **CACHE memory**
- ▶ **Memory Management**
- ▶ **Virtual Memory**
- ▶ **Associative Memory**
- ▶ **Memory Interleaving**

# Introduction

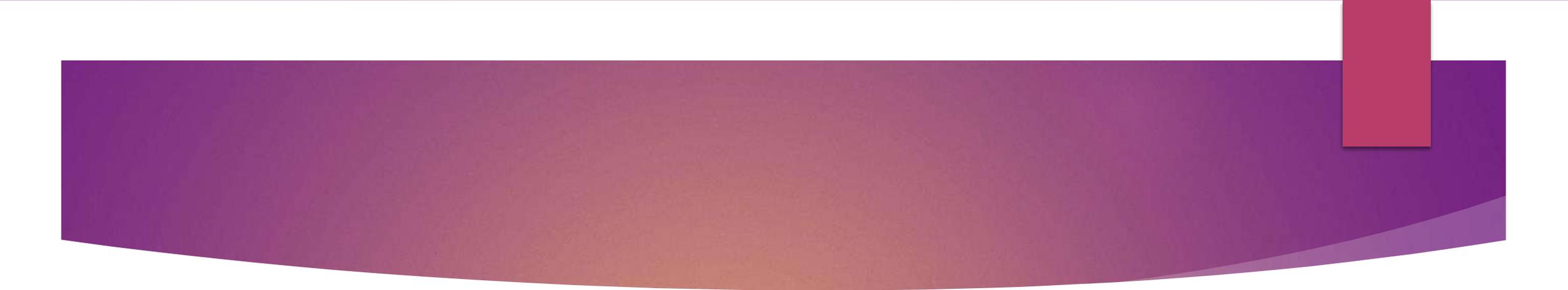
- ▶ All computer manufacturer want their systems to run as fast as possible.
- ▶ Number of advanced techniques currently being investigated to improve system performance.
- ▶ To make computer faster, high performance memory techniques should be implemented.
- ▶ The performance of the memory generally improved due to following techniques.
  - ❑ Cache Memory
  - ❑ Virtual Memory
  - ❑ Associative Memory
  - ❑ Memory Interleaving

# Cache Memory

- ▶ Processor is much faster than the main memory.
  - As a result, the processor has to spend much of its time waiting while instructions and data are being fetched from the main memory.
  - Major obstacle towards achieving good performance.
- ▶ Speed of the main memory cannot be increased beyond a certain point.
- ▶ Cache memory is an architectural arrangement which makes the main memory appear faster to the processor than it really is.
- ▶ A special very high speed memory called Cache which is used to increase the speed of processing by making a current program and data available to the CPU at a rapid rate.
- ▶ Generally it is employed in computer system to compensate the speed difference between the main memory access time and high speed processor logic.
- ▶ So, the Cache is located between Main memory and the Processor.

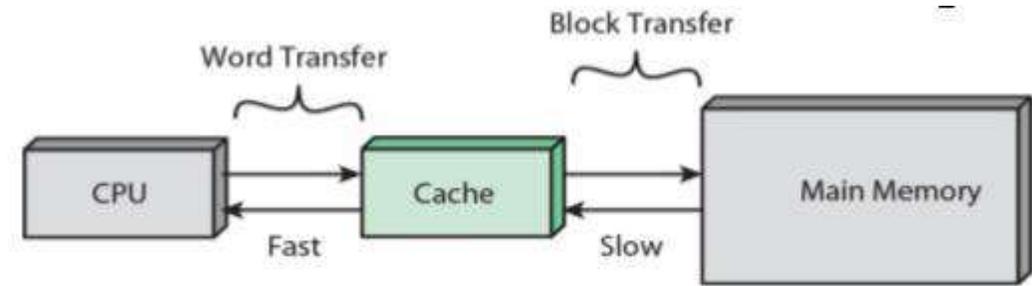


- ▶ Cache memory access time is close to the processor logic cycle time.
- ▶ It is also used to store segments of program currently being needed and temporary data frequently needed in the present calculation.
- ▶ Sometimes it is found in a computer program a particular part is executed repeatedly, while other parts are less frequently.
- ▶ These instructions may be the ones in a loop, nested loop or few procedures/functions calling each other repeatedly.
- ▶ This is called “**locality of reference**”.

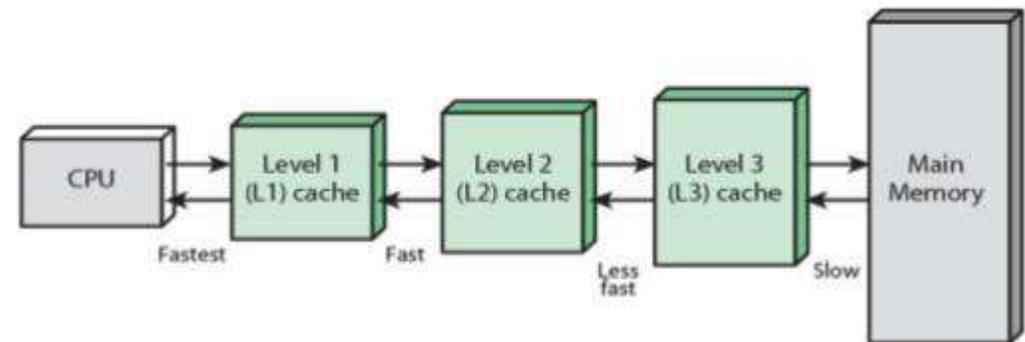
- 
- ▶ There are two types of Locality of reference.
  - ▶ **Temporal locality of reference:**
    - A memory location that is referenced is likely to be accessed again in the near future.
    - Data/instruction brought into cache expecting it to be used again.
  - ▶ **Spatial locality of reference:**
    - Memory locations near the last access are likely to be accessed in the near future.
    - Instructions/data with addresses close to a recently instruction/data are likely to be executed soon.
  - ▶ If the active segment of the program can be placed in a fast memory, then the total execution time can be significantly reduced.
  - ▶ The fundamental idea of Cache organization is that by keeping the most frequently accessed instructions and data in the fast Cache memory.

# Levels of Cache

- ▶ The basic technique that works very effectively is to introduce separate Cache for Instruction and Data.
- ▶ This is called as **Split Cache** which is **Level 1** Cache.
- ▶ Today many memory systems are more complicated and additional Cache called **Level 2** Cache, may reside between instruction and data cache and main memory.
- ▶ In fact there may one more level of cache, that is, **Level 3** cache in more sophisticated systems.
- ▶ Level 2 and Level 3 are **unified Cache**.

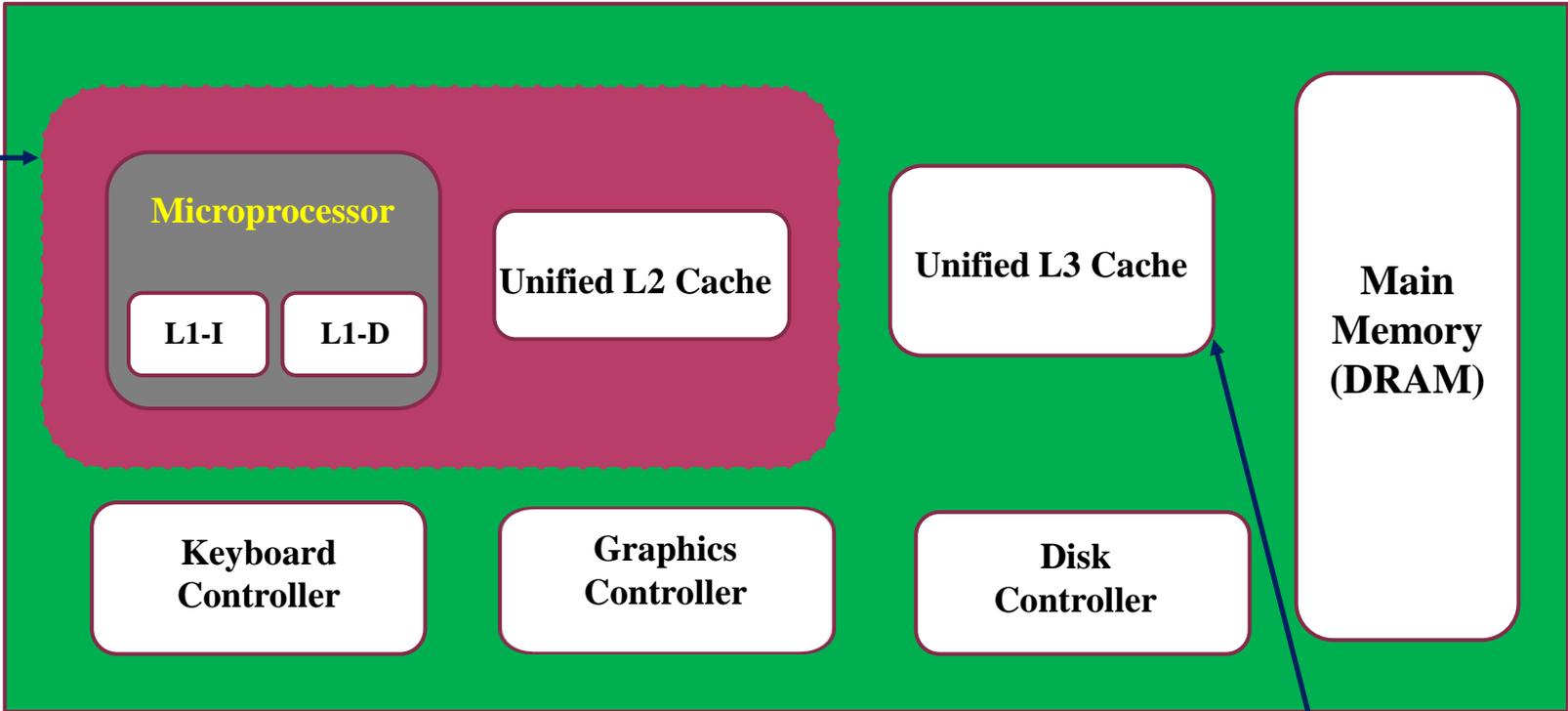


(a) Single cache



(b) Three-level cache organization

CPU Package



Board level Cache (SRAM)

- ▶ The processor within itself contains a small instruction and data cache, typically **16KB to 64KB**.
- ▶ There is Level 2 cache, which is not within the processor, but may be included in the CPU package.
- ▶ This Cache is generally unified containing both instruction and data, typically **512KB to 1MB**.
- ▶ The Level 3 Cache is on the Processor Board and consists of few megabytes of SRAM, which is much faster than Main DRAM Memory.
- ▶ Caches are inclusive, with the full contents of the Level 1 Cache being in the Level 2 and the full contents of the Level 2 Cache being in the Level 3 Cache.

# Operations

- ▶ When the CPU needs to access memory, the Cache is examined.
- ▶ If the word is found in the Cache, it reads from the Cache memory.
- ▶ If the word is not found in the Cache, the main memory is accessed to read the required word.
- ▶ A block of words containing the one just accessed is then transferred from the main memory to Cache memory.
- ▶ In this manner some data are transferred to the Cache from the main memory so that future references to memory find the required words in the fast Cache memory.

# Cache Hit

- ▶ Existence of a Cache is transparent to the processor.
- ▶ The processor issues Read and Write requests in the same manner.
- ▶ If the data is in the cache, it is called a Read or Write **hit**.
- ▶ Read hit:
  - The data is obtained from the cache.
- ▶ Write hit:
  - Cache has a replica of the contents of the main memory.
  - Contents of the cache and the main memory may be updated simultaneously. This is the **write-through** protocol.
  - Update the contents of the cache, and mark it as updated by setting a bit known as the **dirty bit** or **modified** bit. The contents of the main memory are updated when this block is replaced.
  - This is **write-back** or **copy-back** protocol.

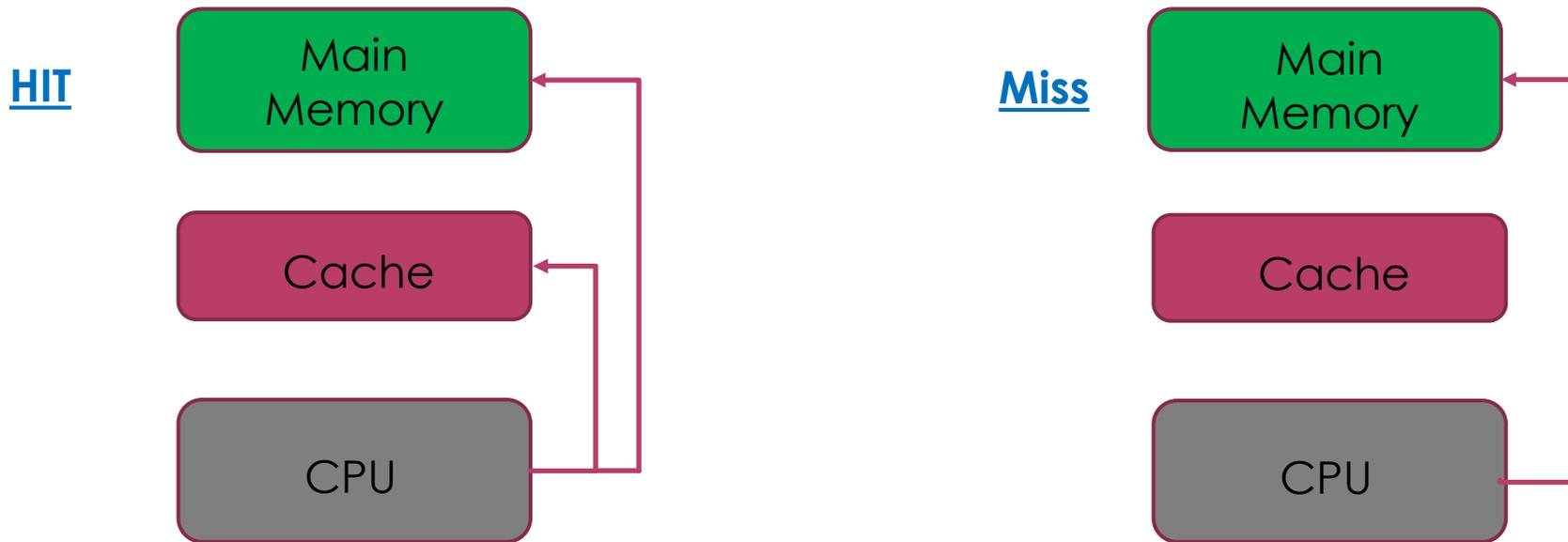
# Cache Miss

- ▶ If the data is not present in the cache, then a **Read miss** or **Write miss** occurs.
- ▶ If Read miss:
  - Block of words containing this requested word is transferred from the memory.
  - After the block is transferred, the desired word is forwarded to the processor.
  - The desired word may also be forwarded to the processor as soon as it is transferred without waiting for the entire block to be transferred. This is called **load-through** or **early-restart**.
- ▶ Write-miss:
  - Write-through protocol is used, then the contents of the main memory are updated directly.
  - If write-back protocol is used, the block containing the addressed word is first brought into the cache. The desired word is overwritten with new information.
- ▶ The performance of Cache is measured in terms of a quantity called **Hit Ratio**.
- ▶ **Hit Ratio = Hit / Hit + Miss** (CPU references to memory)

# Write Policy

- ▶ An important aspect of cache organization is concerned with memory write requests.
- ▶ When the CPU find a word in Cache during a read operation, the main memory is not involved in the transfer.
- ▶ If the operation is a write, there are two ways the system can proceed for writing.
  - ❑ Write through
  - ❑ Write back

# Write through

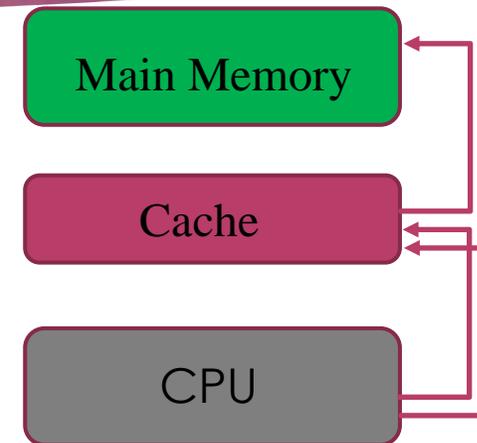


- ▶ Update the main memory with every main memory write operation with the Cache memory being updated in parallel if it contains the word at specified address.
- ▶ The main memory always contains same data as Cache.
- ▶ The data reside in main memory is valid at all time.

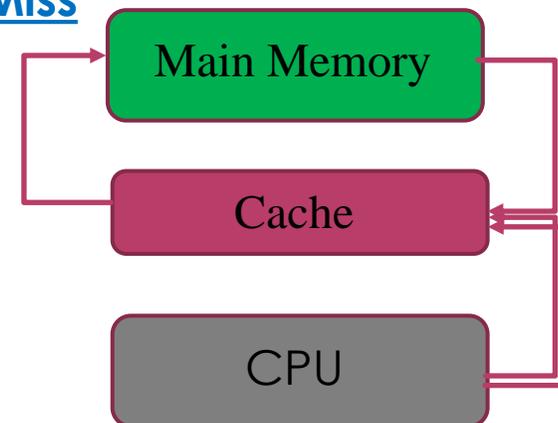
# Write back

- ▶ Problem of write through is repeated write operation.
- ▶ Write back minimizes memory write.
- ▶ Updates are made only in the Cache.
- ▶ When an update occurs, an **Update/Dirty bit** associated with the slot is set.
- ▶ When that block is replaced, it is written back to main memory if and only if the Update/Dirty bit associated with the slot is set.
- ▶ The problem with write back is that the portions of main memory data not valid data.

HIT



Miss

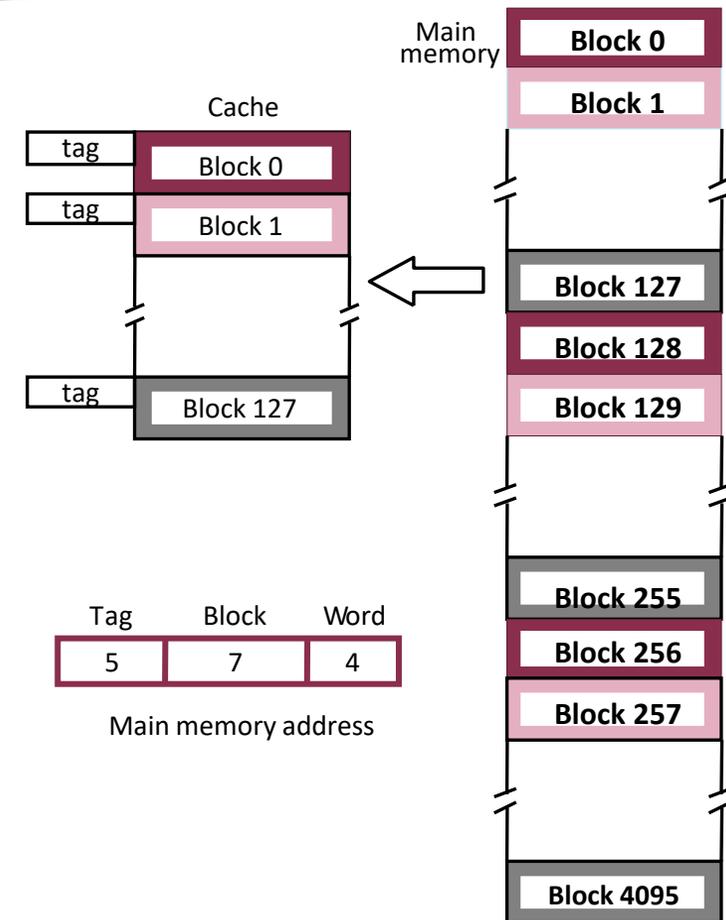


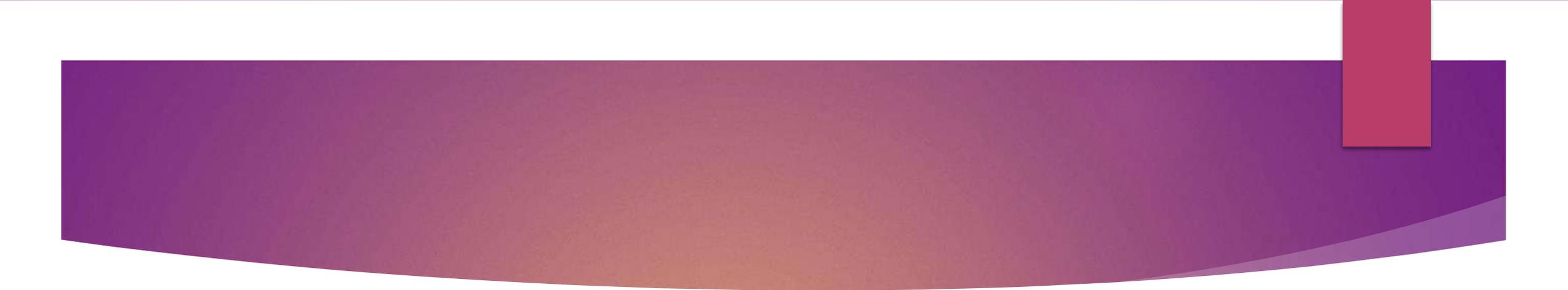
# Cache Mapping

- ▶ Processor issues a Read request, a block of words is transferred from the main memory to the cache, one word at a time.
- ▶ Subsequent references to the data in this block of words are found in the cache.
- ▶ At any given time, only some blocks in the main memory are held in the cache.
- ▶ Which blocks in the main memory are in the cache is determined by a **Mapping Function**.
- ▶ Mapping functions determine how memory blocks are placed in the cache.
- ▶ Three mapping functions:
  - Direct mapping
  - Associative mapping
  - Set-associative mapping.

# Direct Mapping

- ▶ A simple processor example:
  - Main memory has 64K words.
  - Main memory is addressable by a 16-bit address.
  - Size of cache is 2048 (2K) words.
  - Cache is addressable by a 11-bit address.
- ▶ Now **Index will be LS11-bits and Tag will be MS 5-bits.**
  - Number of bits in Index field is equal to the number of bits required for accessing Cache.
  - Cache consisting of blocks of 16 words each.
  - Total number of Blocks will be 128.
- ▶ Now Index bit will be divided as Block bits and Word bits.
- ▶ **Block will be 7-bits and Word will be 4-bits.**
- ▶ Main memory has 4K blocks of 16 words each.

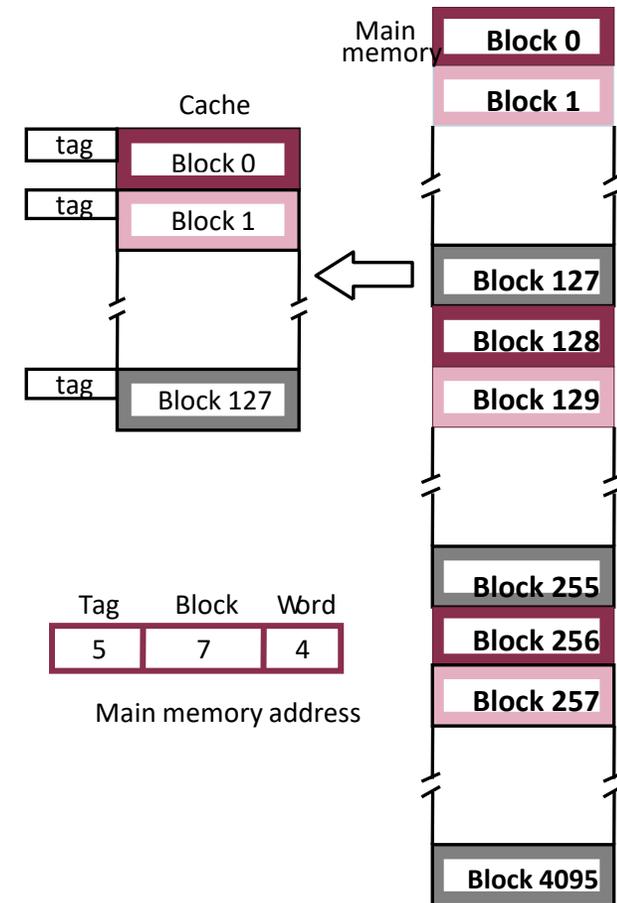


- 
- ▶ Each word in the Cache consists of data word and its Tag bits stored alongside the data bits.
  - ▶ When the CPU generates a memory request, the Index field is used for the address to access the Cache.
  - ▶ The Tag field generated during addressing is compared with the Tag bits in word read from the Cache.
  - ▶ If two Tags match, then there is a Hit.
  - ▶ If no match, there is a Miss and the required word read from the memory and then the word is stored in Cache with the new Tag.

- Block  $j$  of the main memory maps to  $j \bmod 128$  of the cache.

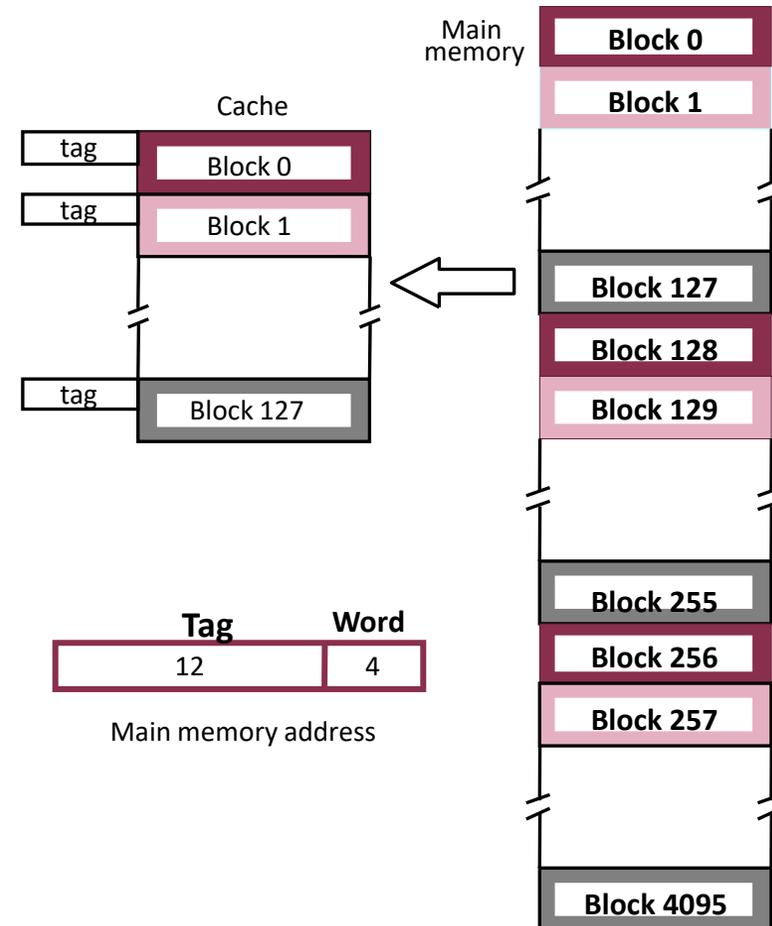
0 maps to 0, 129 maps to 1

- More than one memory block is mapped onto the same position in the cache.
- May lead to contention for cache blocks even if the cache is not full.
- Resolve the contention by allowing new block to replace the old block, leading to a trivial replacement algorithm.
- Memory address is divided into three fields:
  - Low order 4 bits determine one of the 16 words in a block.
  - When a new block is brought into the cache, the next 7 bits determine which cache block this new block is placed in.
  - High order 5 bits determine which of the possible 32 blocks is currently present in the cache. These are tag bits.
- Simple to implement but not very flexible.



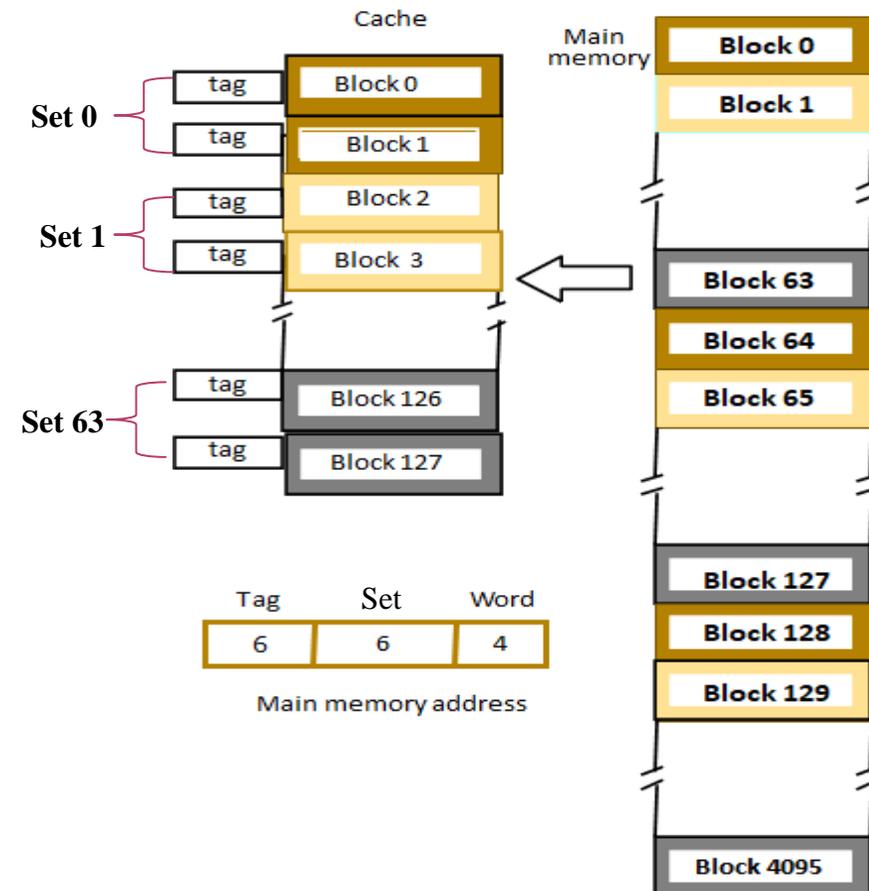
# Associative Mapping

- ▶ This is fastest and flexible.
- ▶ Uses Associative Memory.
- ▶ Main memory block can be placed into any cache position.
- ▶ Memory address is divided into two fields.
- ▶ Lower 4-bits identify the word within a Block.
- ▶ Higher 12-bit is the Tag bit identify a memory block when it is resident in the Cache.
- ▶ So, in Associative Cache each word in the Cache consists of data word and its Tag bits stored alongside the data bits.
- ▶ This is very expensive.
- ▶ Flexible and uses Cache space efficiently.
- ▶ Replacement algorithm is used to replace an existing block in the Cache when the Cache is full.



# Set-Associative Mapping

- ▶ The problem of Direct Mapping is that two words with same Index but different Tag values can not reside in the Cache memory at a time.
- ▶ Set associative addresses the problem of possible thrashing in the direct mapping method.
- ▶ In Set-Associative two or more words of memory under the same Index address can reside.
- ▶ It is a combination of Direct and associative mapping.
- ▶ Blocks of Cache are grouped into Set.
- ▶ Hence, the contention problem of Direct mapping is somehow solved by having few choices for block placement.
- ▶ *Number of blocks per set is a design parameter.*
  - *One extreme is to have all the blocks in one set, requiring no set bits (fully associative mapping).*
  - *Other extreme is to have one block per set, is the same as direct mapping.*



# Replacement Policy

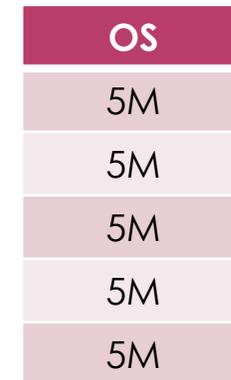
- ▶ When the cache is full, and a block of words needs to be transferred from the main memory, some block of words in the cache must be replaced.
- ▶ This is determined by a **Replacement Algorithm**.
- ▶ Cache performance is greatly affected by properly choosing data that is unlikely to be referenced again.
- ▶ The most common replacement algorithms are:
  - ❑ Random replacement
  - ❑ First-in-first-out (FIFO)
  - ❑ Least Recently Used (LRU)
  - ❑ Least Frequently Used (LFU)

- 
- ▶ With a Random replacement policy any block can be replaced randomly.
  - ▶ In case of FIFO, replace the block in the set which has been in the cache for longest time. It is easily implemented as a round-robin or circular buffer technique.
  - ▶ In case of LRU, replace the block in the set which has been in the Cache for longest period with no references.
  - ▶ In case of LFU, replace the block from the Cache which has fewest references.

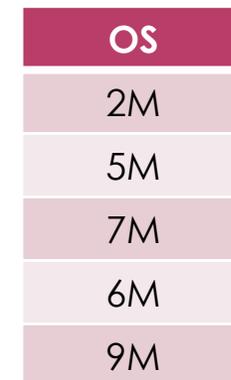
# Memory Management

- ▶ In a uni-processor system, main memory is divided in to two parts.
  - ❑ One part for OS (Resident Monitor)
  - ❑ Other part for the program currently being executed (User part)
- ▶ In multiprogramming the user part of the memory is subdivided to accommodate multiple processes.
- ▶ The subdivision is carried out dynamically.
- ▶ The memory needs to be allocated efficiently to pack as many processes in to the memory as possible.
- ▶ This is known as **Memory Management**.

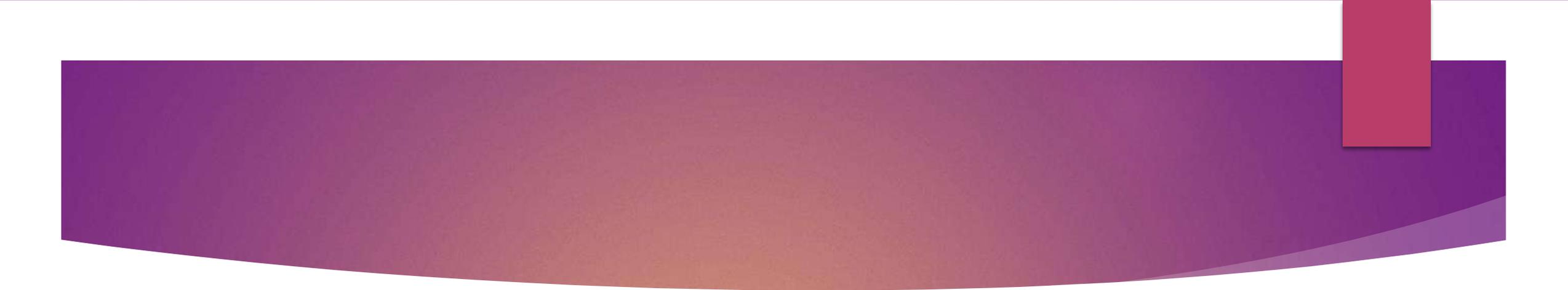
- ▶ Generally there are two types partitioning available for memory.
- ▶ One is **Fixed Partitioning** and another one is **Variable Partitioning**.
- ▶ Fixed partitioning is of two types.
- ▶ One is **Equal size partitioning** and another one is **Unequal size partitioning**.
- ▶ The process brought into the memory and placed into the smallest available partition.
- ▶ In most of the cases process may not require exactly as much memory as provided by the partition.
- ▶ So, there will be wastage of memory space.



Equal size partitioning



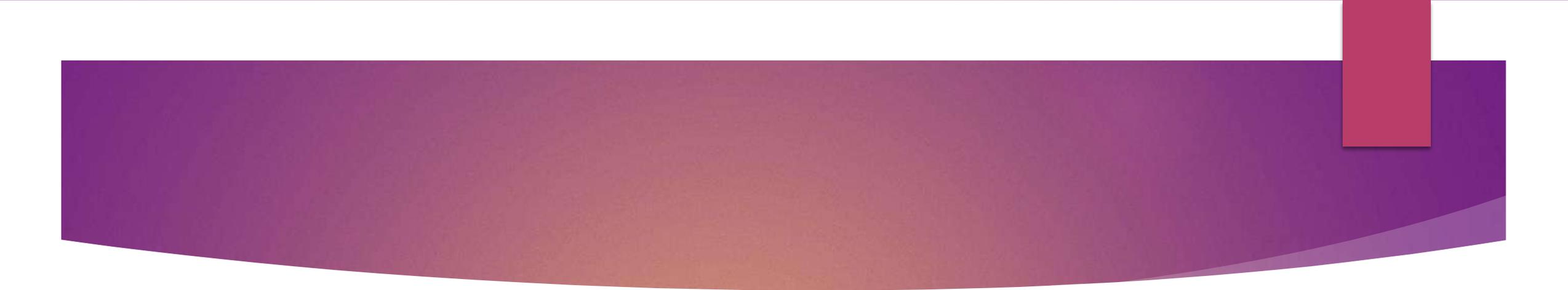
Unequal size partitioning

- 
- ▶ A more efficient approach is **Variable Partitioning**.
  - ▶ When the process is brought into memory, it is allocated exactly as much memory as it requires and no more.
  - ▶ This method starts out well, but eventually it leads to a situation in which there are lot of small holes in memory.
  - ▶ At time goes the memory became more and more fragmented.
  - ▶ Unused space in a partition is called as **Internal Fragmentation**.
  - ▶ Unused of a partition is called **External Fragmentation**.
  - ▶ So, the memory utilization will be declined.

- ▶ The technique to overcome this is **Compaction**.
- ▶ From time to time the OS shifts the processes in the memory and places all the free memory space together in one block.
- ▶ This is called **Compaction**.
- ▶ This is a time consuming procedure and wastage of processor time.
- ▶ If compaction is adopted, the process may be shifted in main memory.
- ▶ The instruction and branch address of the instruction also shifted.
- ▶ This can be solved by **Logical Addressing**.
- ▶ Logical address is expressed as a location relative to the beginning of the program.
- ▶ Instructions in the program contains only **logical address**.
- ▶ **Physical address** is the actual location in the memory.

- ▶ When the processor executes a process, it automatically converts from logical to physical address by adding the current starting location of the process which is called its **Base Address** to each logical address.
- ▶ Both fixed and variable partitioning are not efficient.
- ▶ So, **Paging**.
- ▶ The memory is partitioned in to relatively small fixed size chunks, known as **Frame**.
- ▶ The processes are also divided to small fixed size chunks of same size of frames.
- ▶ The chunk of processes is called as **Page**.

- ▶ The pages are **swapped in** and **swapped out** to or from the frames of the memory.
- ▶ Usually this enhances the performance of the system.
- ▶ This leads to the concept of **Virtual Memory**.
- ▶ Each page of the process brought in to the memory, when it is needed, that is on demand, called **Demand Paging**.
- ▶ During the processing if a page reference is required, but that is not in main memory then **Page fault** occurs.
- ▶ Then the OS brings the required page to main memory.

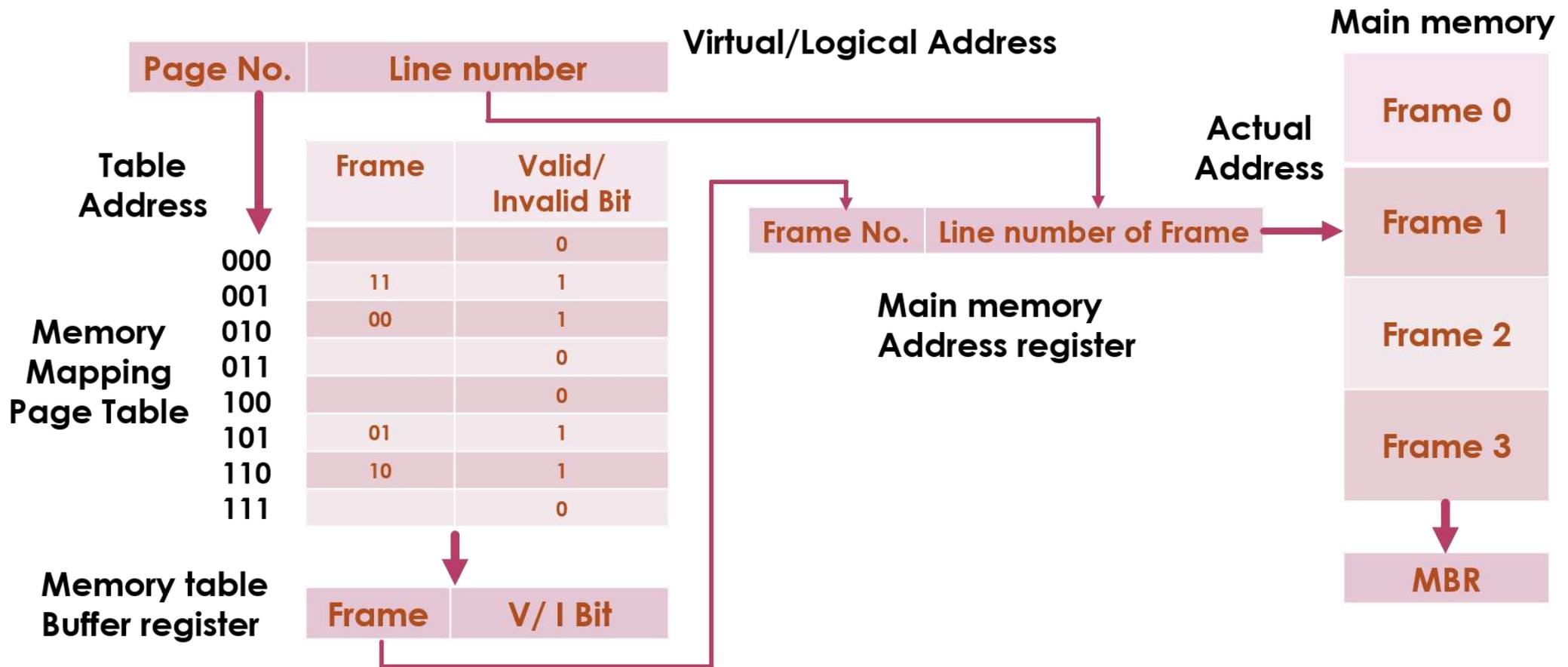
- 
- ▶ Paging is invisible to the programmer.
  - ▶ There is another way in which addressable memory can be subdivided, known as **Segmentation**.
  - ▶ It is visible to the programmer and provides a convenience for organizing the programs and data.
  - ▶ It allows the programmer to view memory as consisting of multiple address space or **segments**.

# Virtual Memory

- ▶ Program and data are stored in auxiliary memory.
- ▶ Portions of the program or data that are brought in to the main memory as they are needed by the CPU.
- ▶ Virtual memory is a technique that allows execution of processes that may not be completely in the main memory.
- ▶ Virtual memory concept is used to give programmer the illusion that they have very large memory, even though the computer actually has a relatively small main memory.
- ▶ Each address is referenced by the CPU goes through an address mapping from the **virtual address** to a **physical address** in main memory.

- 
- ▶ An address used by the programmer called **Virtual address** or **Logical address**.
  - ▶ Set of virtual addresses called as **Address space**.
  - ▶ An address in main memory is called as **location** or **Physical address**.
  - ▶ Set of physical addresses called as **Memory space**.
  - ▶ The physical memory is divided in to equal size called as **Frame** or **Block**.

- ▶ Let a computer address space is 8K and memory space is 4K.
- ▶ Let split the address space in to 8 pages and each page size is 1K.
- ▶ The virtual address will be 13 bits.
- ▶ These 13 bits are for specifying the pages and the lines within the page.
- ▶ So, for 8 pages 3 bit are required and rest 10 bits LSB for the line number of each page.
- ▶ The memory space is divided in to the size of pages as 1K so, 4 frames or blocks.
- ▶ The actual address is 12 bits.
- ▶ For 4 frames, 2 bits are required and rest 10 bits LSB for the line number of each frame.
- ▶ So, the line address in address space and memory space is same.
- ▶ Only the translation is required from page to frame, that is, from 3 bits to 2 bits.

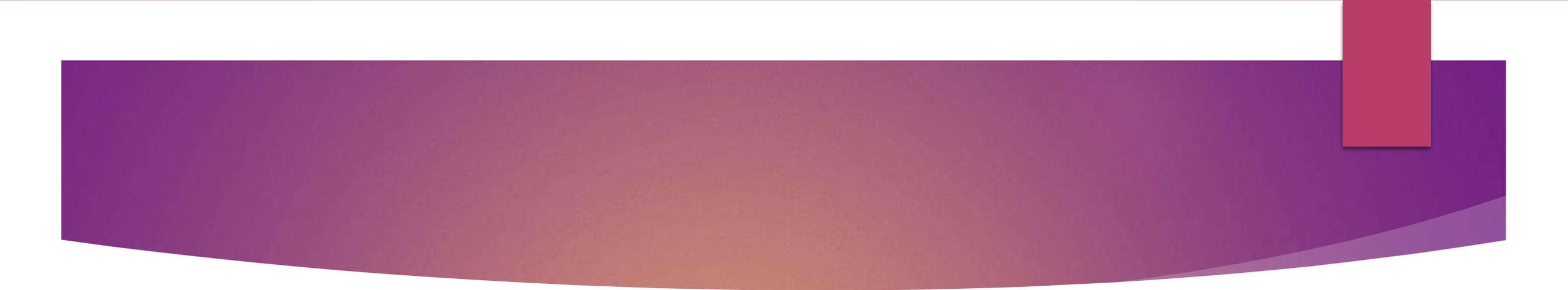


# Page Replacement Algorithms

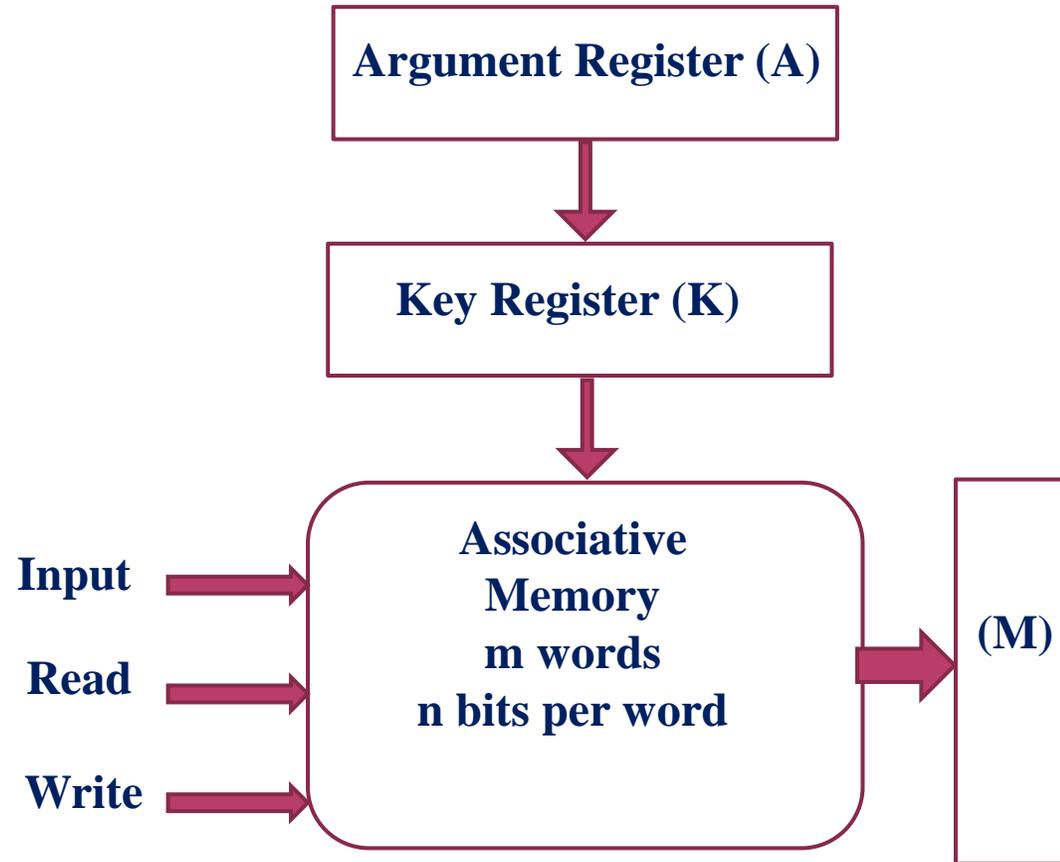
- ▶ If a page is required for processing and that is not in main memory and the main memory is full, in that case page replacement is required.
- ▶ Various page replacement algorithms are:
  - ❑ First-in-first-out (FIFO)
  - ❑ Least Recently Used (LRU)
  - ❑ Optimal Algorithm
- ▶ In case of LRU, replace the page that has not been used for longest period of time.
- ▶ In case of optimal algorithm, replace the page that will not be used for the longest period of time.

# Associative Memory

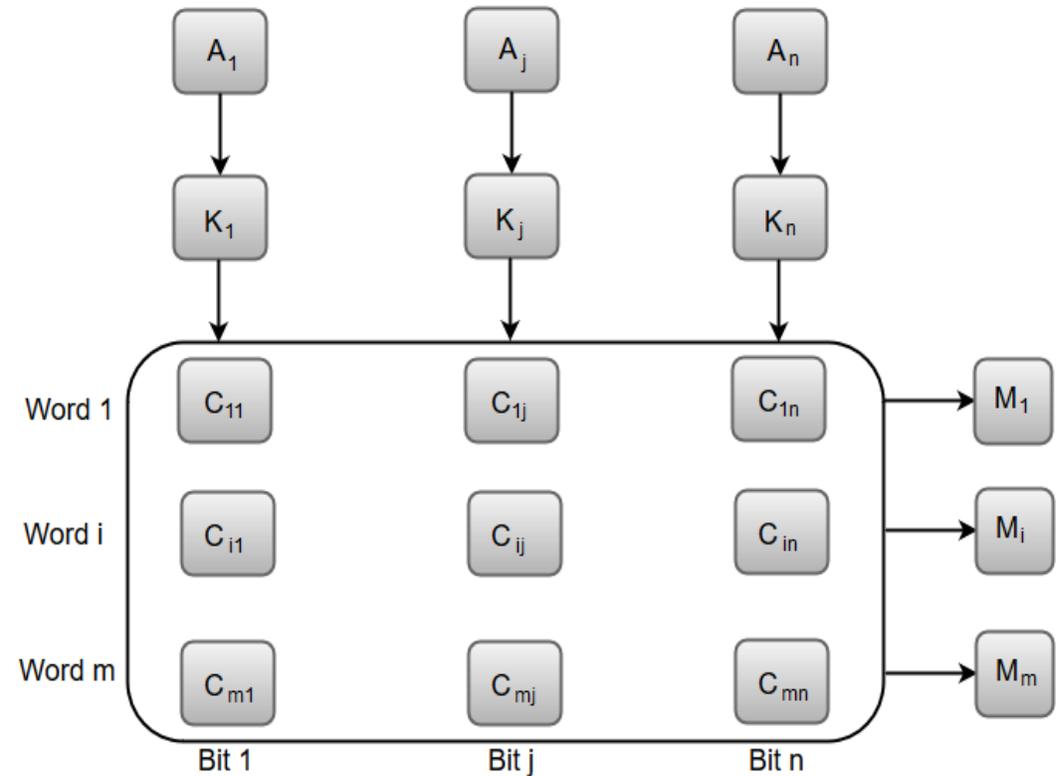
- ▶ The time required to find an item stored in memory can be reduced considerably if stored data can be identified for access by the content of data itself rather than by an address.
- ▶ The **Associative Memory (AM)** is called as **Content Addressable Memory (CAM)** which addresses data by content rather than by address.
- ▶ When a word is written in an associative memory no address is needed.
- ▶ The memory is also capable of finding unused locations to store that word.

- 
- ▶ When a word to be read from an associative memory the content of word or part of the word is specified.
  - ▶ The memory locates all words which match the specified content and mark them for reading.
  - ▶ Then at a time all marked content can be read.
  - ▶ So, associative memory known as **Content addressable memory** or **Parallel search memory** or **Multiaccess memory**.

# Hardware Organization

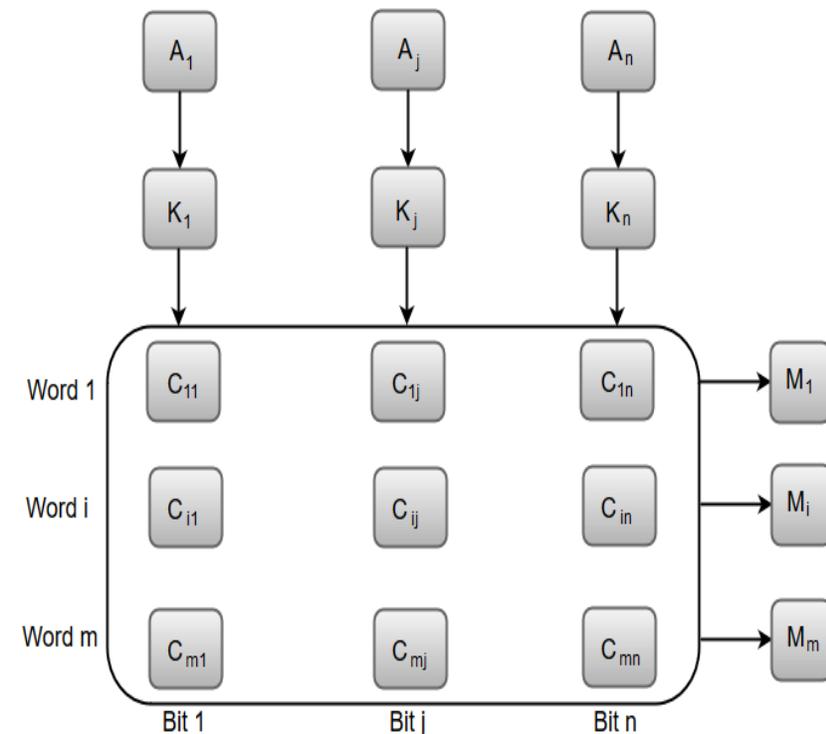


Associative memory of m word, n cells per word:



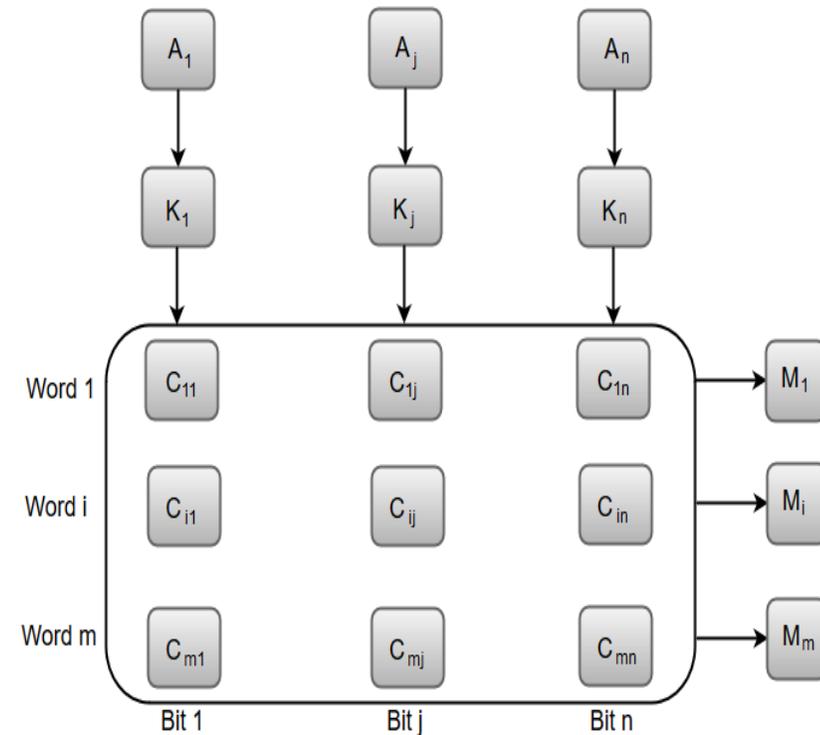
- ▶ Each word in the memory is compared in parallel with the contents of the **Argument register**.
- ▶ The **Key register** or **Masking register** provides a mask for choosing a particular field or key in the argument word.
- ▶ The entire argument is compared with each memory word if the Key register contains all 1's.
- ▶ Otherwise only those bits in the argument that have 1's in their corresponding position of the Key register are compared.

Associative memory of m word, n cells per word:



- ▶ Thus, masking register provides a mask for identifying whole or piece of information which specifies how the reference to the memory is made.
- ▶ The words that matches the bits of the Argument register taking in to consideration of 1's in Key register set the corresponding bit in the **Match register**.
- ▶ After the matching process, those bits in the Match register that have been set indicates that their corresponding word have been matched.

Associative memory of m word, n cells per word:



- ▶ Reading is accomplished by accessing to the memory for those words corresponding bits in the Match register have been set.

### Match Example:

$$A = 101111100$$

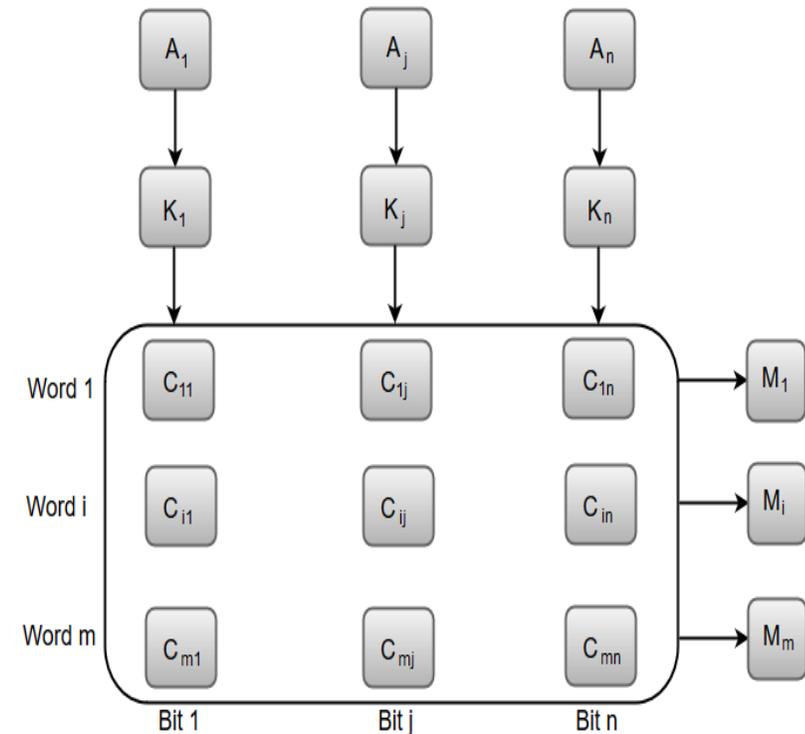
$$K = 111000000$$

$$\text{Word 1} = 100111100$$

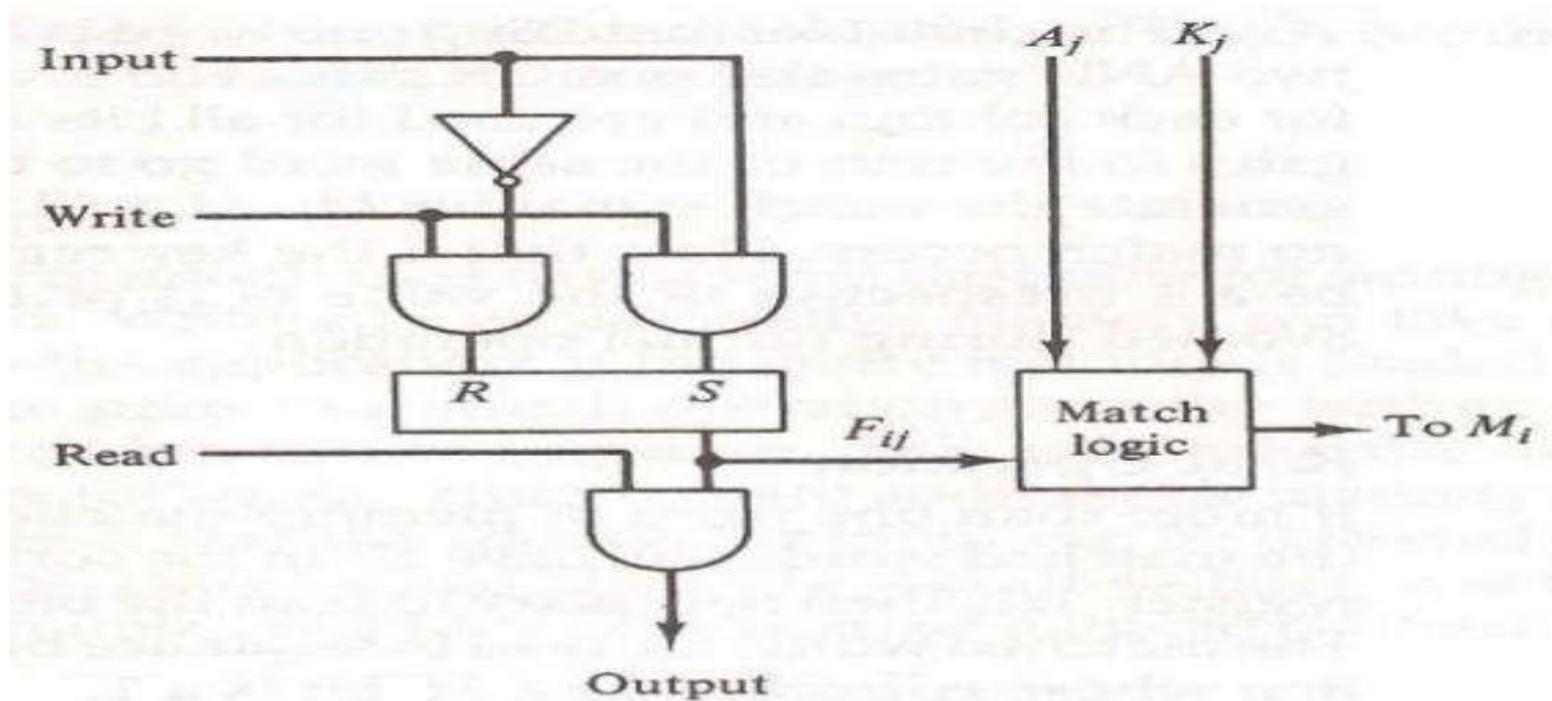
$$\text{Word 2} = 101000001$$

- ▶ Which word will match?

Associative memory of m word, n cells per word:



# Internal Organization of each Cell



$A_j$  will be compared with  $F_{ij}$  for matching when  $K_j = 1$

# Match Logic

- ▶ Comparison logic for two numbers required to find matching.
- ▶ Neglecting Key or Mask bits, *word i* is equal to the argument A if

$$A_j = F_{ij} \text{ for } j = 1, 2, 3, \dots, n$$

- ▶ Two bits are equal if both are either 1 or 0.
- ▶ The equality of two bits can be expressed logically by the Boolean function

$$x_j = A_j F_{ij} + A'_j F'_{ij} \quad x_j = 1 \text{ if both are equal else } 0$$

- ▶ For *word i* to be equal to the argument A must all  $x_j$  variables equal to 1.
- ▶ This is the condition for setting the corresponding match bit  $M_i$  to 1.
- ▶ So, the Boolean function will be  $M_i = x_1 x_2 x_3 \dots x_n$ . *This is a AND operation.*

▶ Now the key bit  $K_j$  is included in the comparison logic.

▶ When  $K_j = 1$   $A_j$  and  $F_{ij}$  needs comparison

$K_j = 0$   $A_j$  and  $F_{ij}$  need not be compared

$$x_j + K'_j = \begin{cases} x_j, & \text{if } K_j = 1 \\ 1, & \text{if } K_j = 0 \end{cases}$$

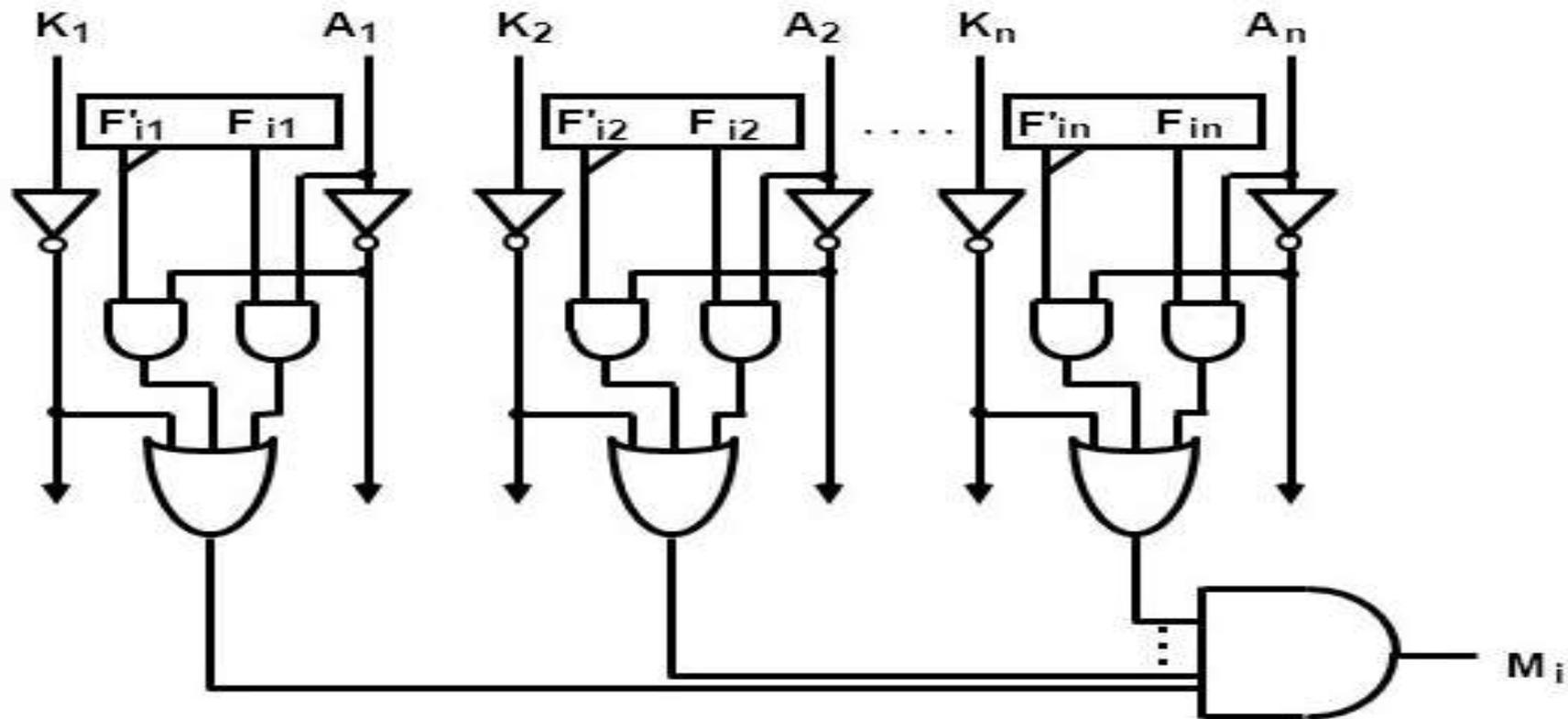
▶ Thus, now the  $x_j$  term can be rewritten as  $(x_j + K'_j)$ .

▶ Now the match logic  $M_i$  for *word*  $i$  will be

$$M_i = (x_1 + K'_1) (x_2 + K'_2) \dots (x_n + K'_n)$$

- ▶ If  $K_j = 1$  the terms will be compared and will be either  $0$  or  $1$ .
- ▶ When all terms will be  $1$  then only match will occur and  $M_i$  will be  $1$ .
- ▶ If  $K_j = 0$  all terms in the expression will be  $1$  but this condition can be avoided as all bits of Key word can not be  $0$ .
- ▶ So, now  $M_i = \prod_{j=1}^n A_j F_{ij} + A'_j F'_{ij} + K'_j$
- ▶ This is the expression for one word, that is *word i*.
- ▶ m numbers of such function needed to compare m numbers of words as  $i = 1, 2, 3, \dots, m$ .

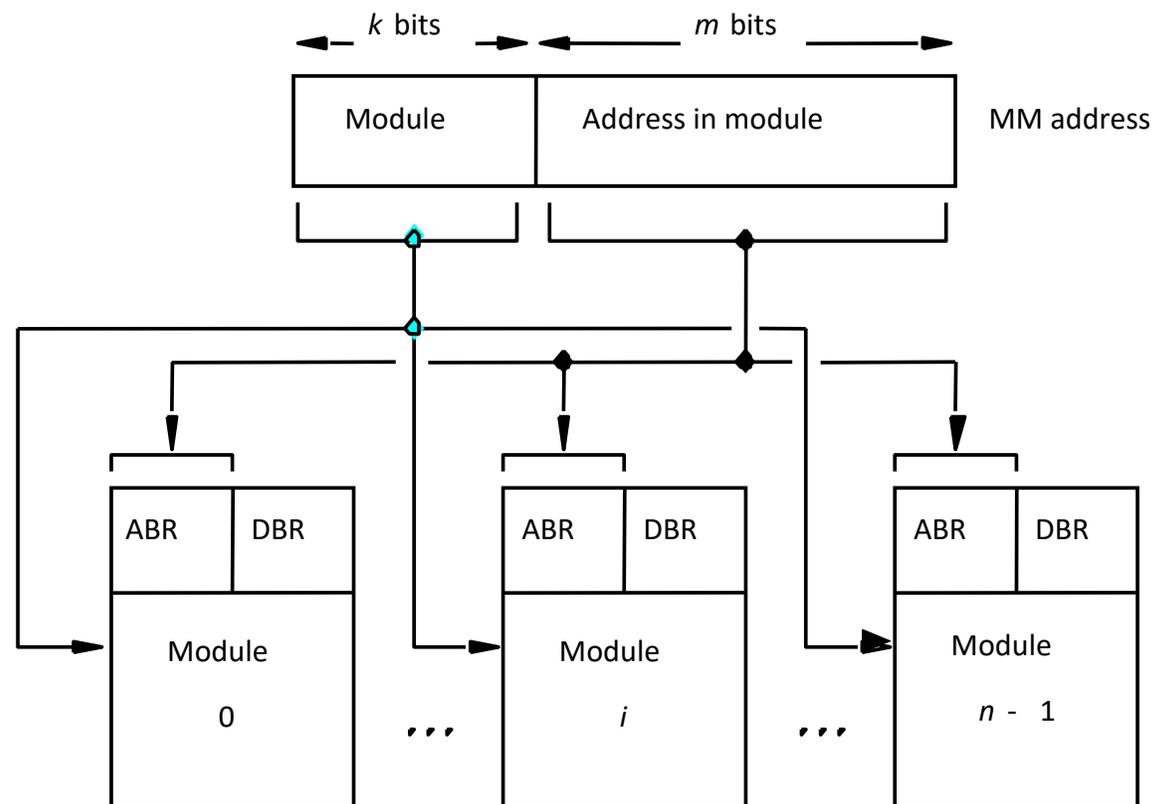
# Diagram of Match Logic



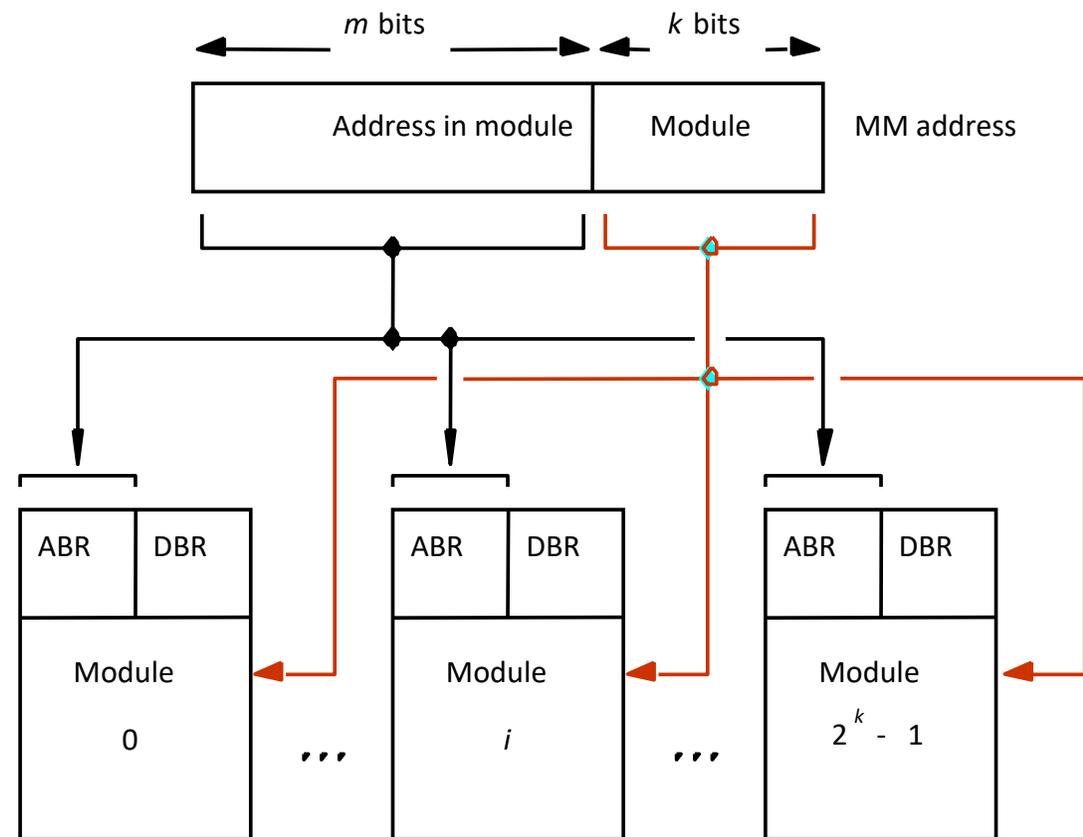
# Memory Interleaving

- ▶ There may be simultaneous access to memory from two or more sources due to parallel processing.
- ▶ The memory can be partitioned into a number of modules connected to a common memory address and data bus.
- ▶ Each module has its own address buffer register (ABR) and data buffer register (DBR).
- ▶ Due to its personal Address register and Data register for each module the average rate of transmission of data is increased.

- **Consecutive words are placed in a module.**
- **High-order  $k$  bits of a memory address determine the module.**
- **Low-order  $m$  bits of a memory address determine the word within a module.**
- **When a block of words is transferred from main memory to cache, only one module is busy at a time.**
- **One module kept busy by CPU at a time.**



- Arrange addressing so that successive words in the address space are placed in consecutive modules.
- When requests for memory access involve consecutive addresses, the access will be to consecutive modules.
- While transferring a block of data, several memory modules can be kept busy at the same time.
- Since parallel access to these modules is possible, the average rate of fetching words from the Main Memory can be increased.
- Higher utilization of memory system.



# Lecture of Module 4

## **Input/Output Organization**

# Overview

- ▶ **Peripheral Devices**
- ▶ **Accessing I/O Devices**
- ▶ **Input/Output Interface**
- ▶ **Types of Data Transfer**
  - ❑ **Asynchronous Parallel Data Transfer**
  - ❑ **Asynchronous Serial Data Transfer**
- ▶ **Asynchronous Communication Interface**
- ▶ **Interrupt**
- ▶ **Modes of Data Transfer**
  - ❑ **Programmed I/O**
  - ❑ **Interrupt driven I/O**
    - **Priority Interrupt**
  - ❑ **Direct Memory Access (DMA)**
- ▶ **I/O Channel and Processor**

# Peripheral Devices

- ▶ Input/Output devices that are connected to a system are called **Peripheral devices**.
- ▶ These devices are designed to read information into or out of the memory or CPU upon command from the CPU and are considered to be the part of computer system.
- ▶ For example: *Keyboards, display units, printers* etc. are common peripheral devices.

## **There are three types of peripherals:**

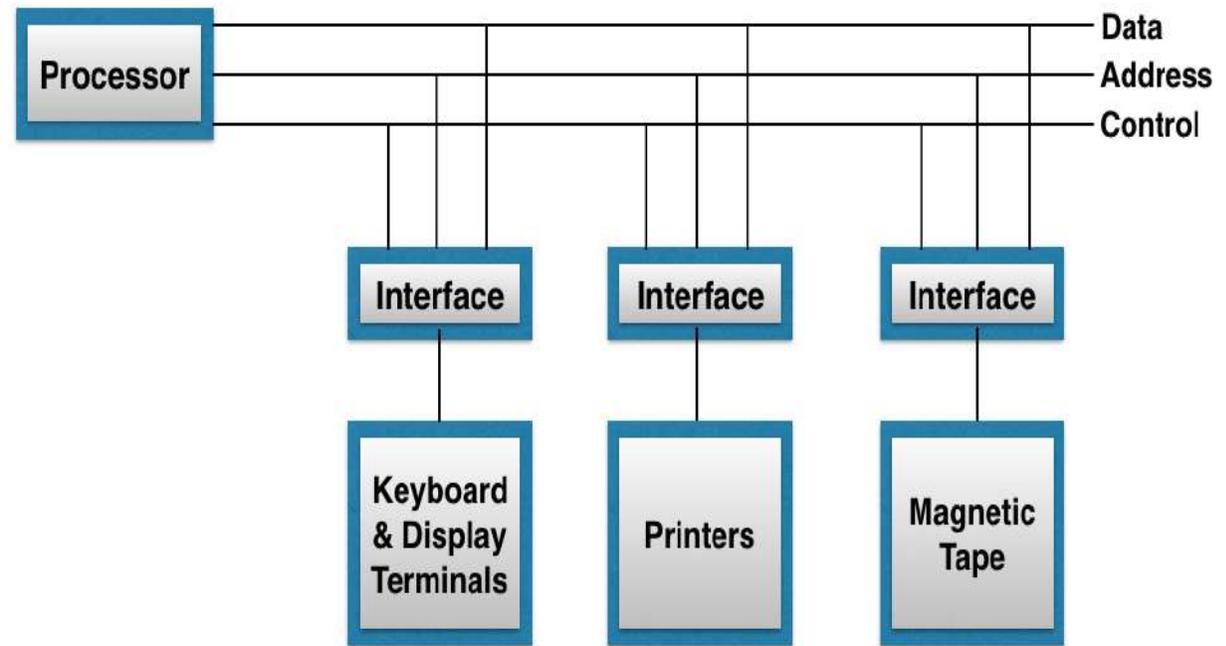
1. **Input peripherals:** Allows user input from the outside world to the computer. Example: Keyboard, Mouse etc.
2. **Output peripherals:** Allows information output from the computer to the outside world. Example: Printer, Monitor etc.
3. **Input-Output peripherals:** Allows both input (from outside world to computer), as well as, output (from computer to the outside world). Example: External memory devices.

# Accessing I/O Devices

- ▶ Generally more numbers of devices are connected to a computer.
- ▶ So, some means have to be provided by which a particular device can be selected to participate in a given I/O operation.
- ▶ This can be accomplished through the use of an I/O bus arrangement.
- ▶ I/O bus is the bus to which all I/O devices are connected used for address, data and control signals.
- ▶ Each device is assigned an identifying code or address so that the CPU can select a particular device by placing its address in the address line.
- ▶ Only the device that recognizes its address responds to the CPU.
- ▶ This is called as **I/O mapped I/O**.

- 
- ▶ An alternative arrangement for identifying I/O devices are by assigning them by an unique address within the memory address space of the computer.
  - ▶ It become possible to access I/O device in the same way as any other memory location accessing.
  - ▶ This is known as **Memory mapped I/O**.

- ▶ Any machine instruction and addressing mode that can be used to deal with operands can also refer to an I/O device.
- ▶ The use of memory mapped I/O offers some flexibility in handling I/O operation.
- ▶ All the devices are connected to the I/O bus through a hardware device or circuit is called as **I/O Interface**.



Connection of I/O Bus to I/O Device

# I/O Interface

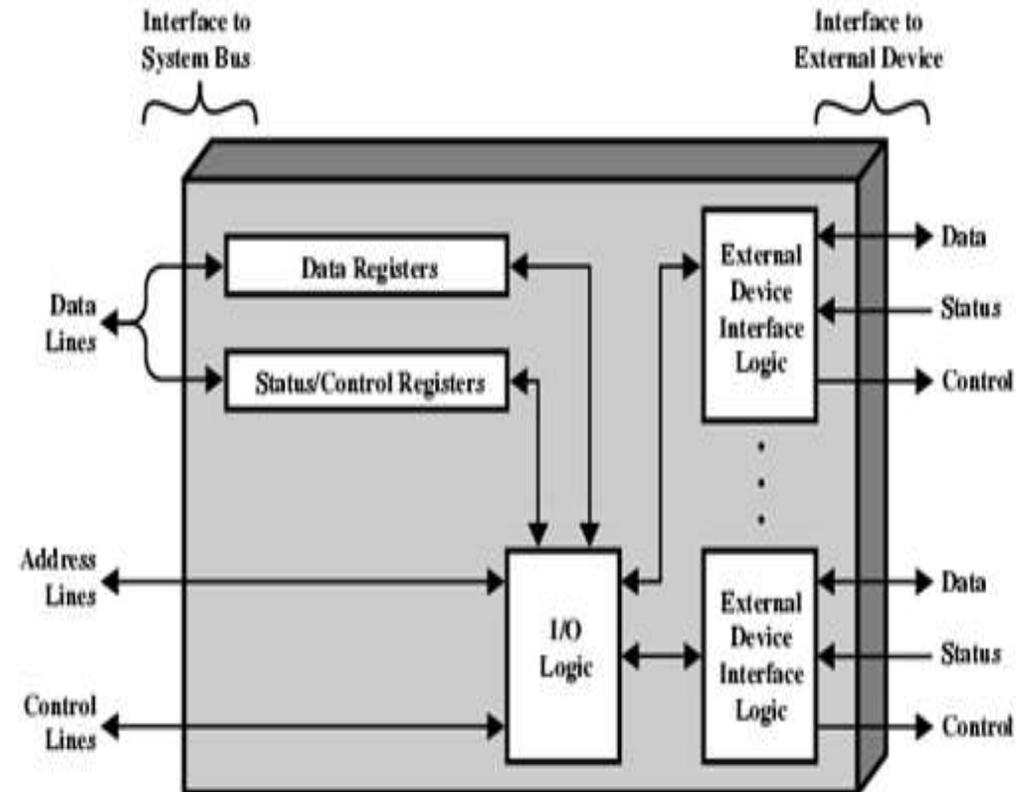
- ▶ Input/Output Interface provides a method for transferring instruction between internal storage device and external I/O device.
- ▶ Peripherals connected to a computer need special communication link for interfacing them with the CPU.
- ▶ The purpose of communication link is to resolve the differences that exists between the CPU and peripherals and among peripherals.

## The major differences are as follows:

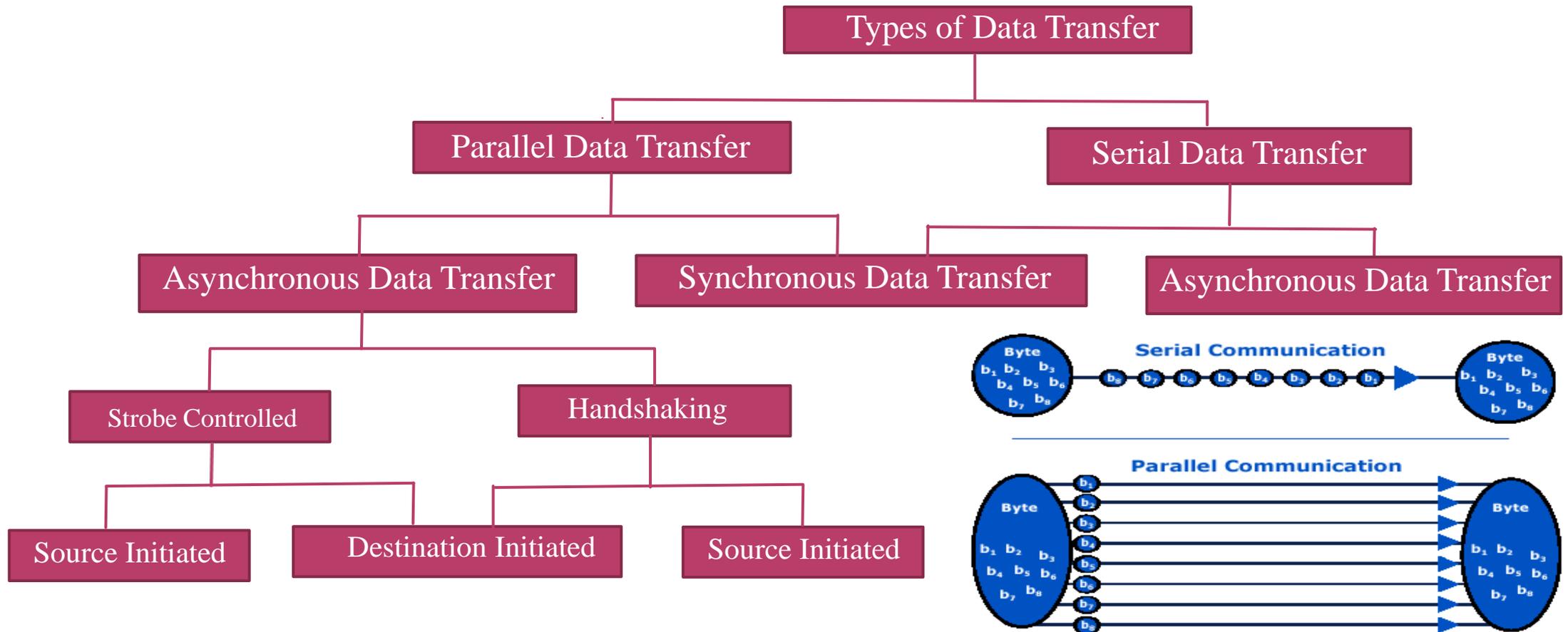
1. Peripheral devices are electromagnetic or electro-mechanical. The CPU is purely electronic device. The nature of operation of peripheral device is different from the operation of CPU. Therefore a conversion of signal values may be required.
2. The data transfer rate of peripherals are usually slower than CPU. So, synchronization mechanism may be needed.
3. The data code and formats in peripherals differ from the word format in the CPU and memory. So, conversion mechanism may be needed.
4. The operating modes of peripherals are different from each other and each must be controlled so as not to disturb the operation of other peripherals connected to CPU.

# Interface Module

- ▶ To resolve these differences there is a need of special hardware component between CPU and peripheral devices to supervise and synchronize all input and output transfers.
- ▶ This component is called as **Interface unit** or **I/O module**.
- ▶ The data register is used to hold data to be transferred.
- ▶ Status register is required to store the status, that is, status or condition of the device for operation.
- ▶ Control circuit is for receiving and sending control signals to or from control unit.



# Types of Data Transfer



<b>Parallel</b>	<b>Serial</b>
<b>1. Each bit has its own path &amp; total message is transmitted at the same time.</b>	<b>1. Each bit in the message is transmitted in sequence one at a time.</b>
<b>2. To transmit n-bit, n-separate wires are used.</b>	<b>2. n-bit is transmitted through one wire.</b>
<b>3. Expensive due to multiple numbers of wires are used to transmit.</b>	<b>3. Inexpensive as compare to parallel because single wire is used to transmit.</b>
<b>4. Speed is faster as data is transmitted through multiple wires.</b>	<b>4. Speed is slow as data is transmitted through single wire.</b>
<b>5. Quite difficult to upgrade the system.</b>	<b>5. Easy to upgrade the system.</b>
<b>6. Suitable for Short distance.</b>	<b>6. Suitable for Long distance.</b>
<b>7. EX- CPU &amp; Memory.(Bus)</b>	<b>7. EX- CPU &amp; I/O.</b>

# Synchronous Data Transfer

- ▶ Both sender and receiver share a common clock.
- ▶ For long distance transmission each unit is driven by separate clock of same frequency.
- ▶ The transmitter transmits block of character along with synchronization information.
- ▶ The receiver decodes and find the synchronization information and keep its clock in step with each other.
- ▶ The transmitter sends data, and the receiver counts the number of bits in the received data.
- ▶ Furthermore, there are no gaps between data, that is, continuous transmission till end.
- ▶ In this method, the timing signals must be accurate to transfer data efficiently.
- ▶ Moreover, this method is faster than asynchronous data transferring.

# Asynchronous Parallel Data Transfer

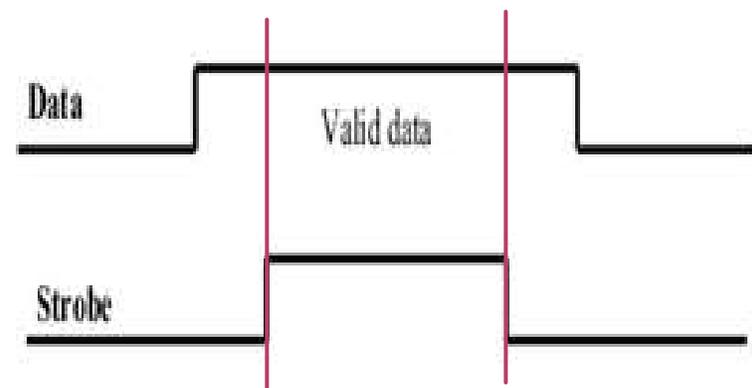
- ▶ Asynchronous parallel data transfer between two independent units require control signals for synchronization during the data transfer.
  1. One way is by means of **strobe pulse** which is supplied by one of the units to other unit when transfer has to occur. This method is known as “**Strobe Control**”.
  2. Another method commonly used is to accompany each data item being transferred with a control signal that indicates the presence of data in the bus. The unit receiving the data item responds with another **signal to acknowledge** receipt of the data. This method of data transfer between two independent units is said to be “**Handshaking**”.
- ▶ Here we consider the transmitting unit as **Source** and receiving unit as **Destination**.
- ▶ The sequence of control during an asynchronous transfer depends on whether the **transfer is initiated by the source or by the destination**.
- ▶ So, it can be further divided as **source initiated** and **destination initiated**.

# Source Initiated Strobe controlled

- ▶ The source unit first places the data on the data bus.
- ▶ After a brief delay to ensure that the data settle to a steady value, the source activates a strobe pulse.
- ▶ The information on data bus and strobe control signal remains in the active state for a sufficient period of time to allow the destination unit to receive the data.
- ▶ The source removes the data from the data bus after it disables its strobe pulse.
- ▶ The strobe signal disabled indicates that the data bus does not contain valid data.
- ▶ New valid data will be available only after the strobe is enabled again.



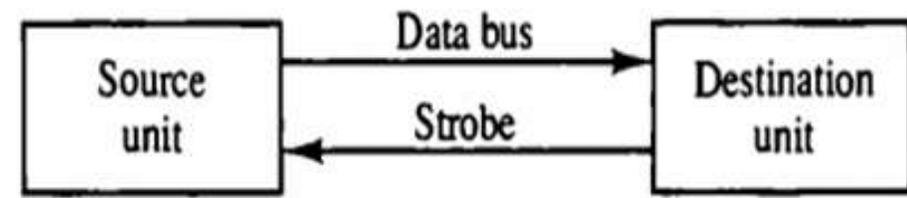
(a) Block Diagram



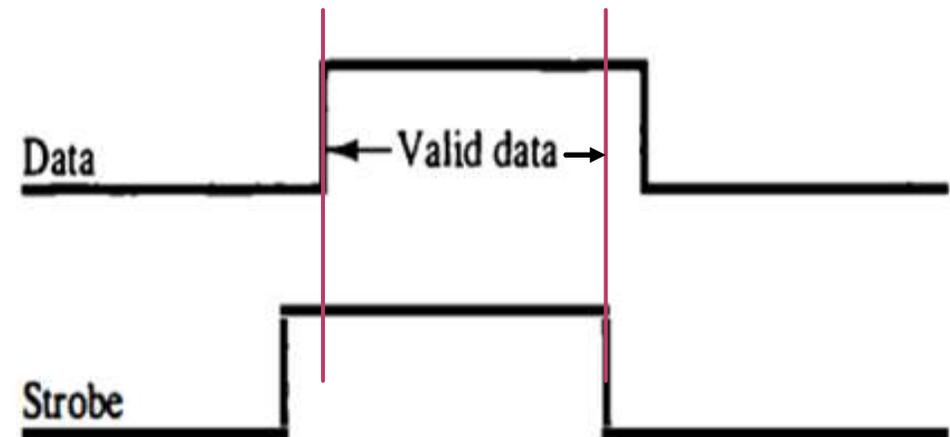
(b) Timing Diagram

# Destination Initiated Strobe controlled

- ▶ The destination unit initiates the data transfer.
- ▶ The destination unit first activates the strobe pulse, informing the source is ready to receive data or informing the source to provide the data.
- ▶ Then the source unit may place the data on the data bus.
- ▶ The data must be valid and remain in the bus long enough for the destination unit to accept it.
- ▶ The destination unit then disables the strobe. And source removes the data from data bus after a time interval.



(a) Block diagram



(b) Timing diagram

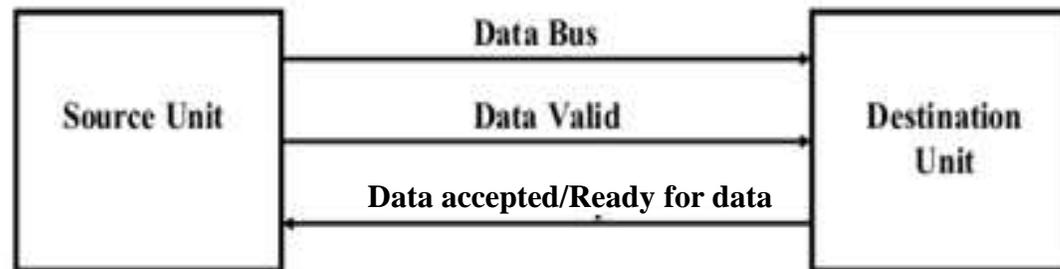
- ▶ In computer, in strobe initiated by source - the strobe may be a **memory-write** control signal from the CPU to a memory unit. The source, CPU, places the word on the data bus and informs the memory unit, which is the destination, that this is a write operation.
- ▶ In the strobe initiated by destination - the strobe may be a **memory-read** control from the CPU to a memory unit. Now the destination is the CPU, initiates the read operation to inform the memory, which is a source unit, to place selected word into the data bus.
- ▶ **The transfer of data between the CPU and an interface unit is similar to the strobe transfer.**

## Disadvantage

- ▶ In source unit that initiates the transfer has no way of knowing whether the destination has actually received the data that was placed on the bus.
- ▶ In a destination unit that initiates the transfer has no way of knowing whether the source unit has actually placed data on the bus.
- ▶ The Handshaking method solves this.

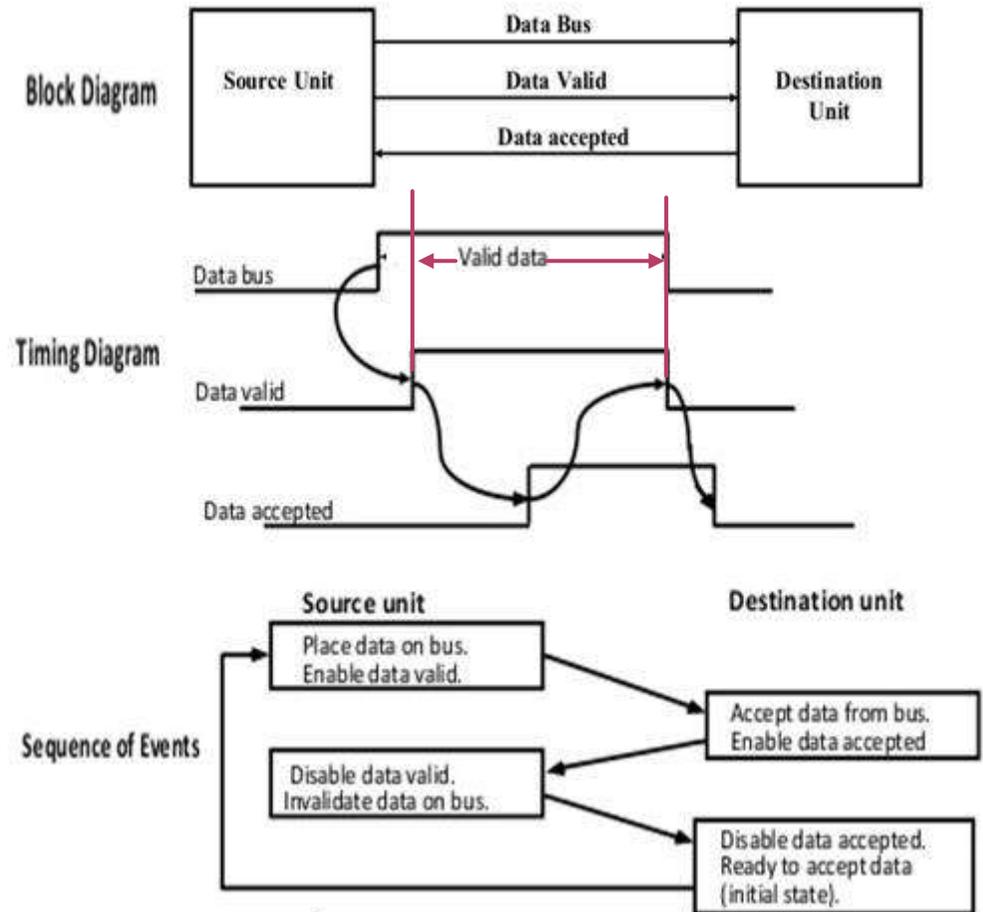
# Handshaking

- ▶ The handshaking method is a two wire controlling method of data transfer.
- ▶ Handshaking method introduce a **second control signal line** that provides a reply to the unit that initiates the transfer.
- ▶ In it, one control line is in the same direction as the data flow in the bus from the source to destination to indicate the presence of valid data on the data bus.
- ▶ The other control line is in the other direction from destination to the source.
- ▶ It is used by the destination unit to inform the source whether the data accepted or not.



# Source Initiated Handshaking

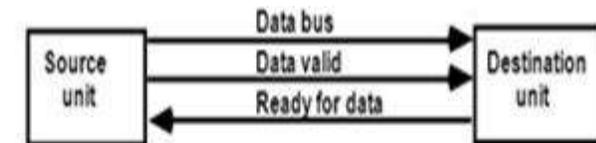
- ▶ Two handshaking lines are "data valid", which is generated by the source unit, and "data accepted", generated by the destination unit.
- ▶ The source initiates a transfer by placing data on the bus.
- ▶ Then enable the data valid signal.
- ▶ The data accepted signal is then activated by destination unit after it accepts the data from the bus.
- ▶ The source unit then disables its data valid signal which invalidates the data on the bus.
- ▶ After this, the destination unit disables its data accepted signal and the system goes into initial state.
- ▶ The source unit does not send the next data item until the destination unit shows its readiness to accept new data by disabling the data accepted signal.



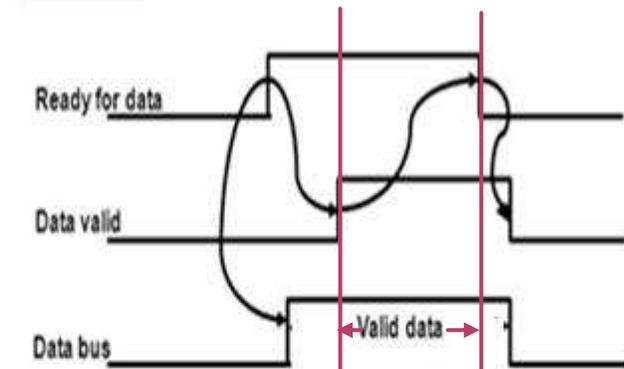
# Destination Initiated Handshaking

- ▶ Two handshaking lines are "data valid", generated by the source unit, and "ready for data" generated by destination unit.
- ▶ Data transfer is initiated by destination, so source unit does not place data on data bus until it receives ready for data signal from destination unit.
- ▶ Enable ready for data.
- ▶ Source Places the data on data bus.
- ▶ Then enable the data valid signal.
- ▶ Destination accepts the data from the data bus.
- ▶ Then the destination disable the ready for data signal after data acceptance.
- ▶ Now the source disable the data valid signal which invalidate the data on data bus and the system goes into initial state.

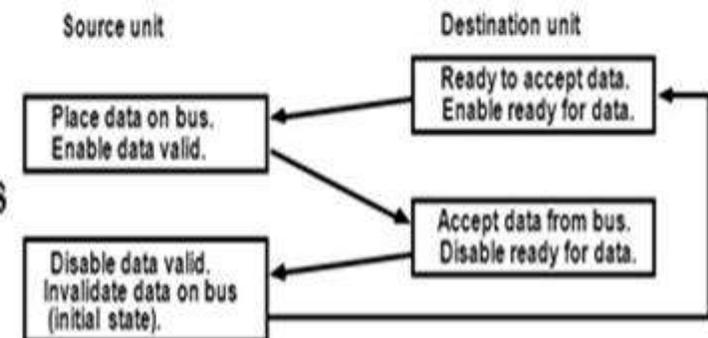
Block Diagram

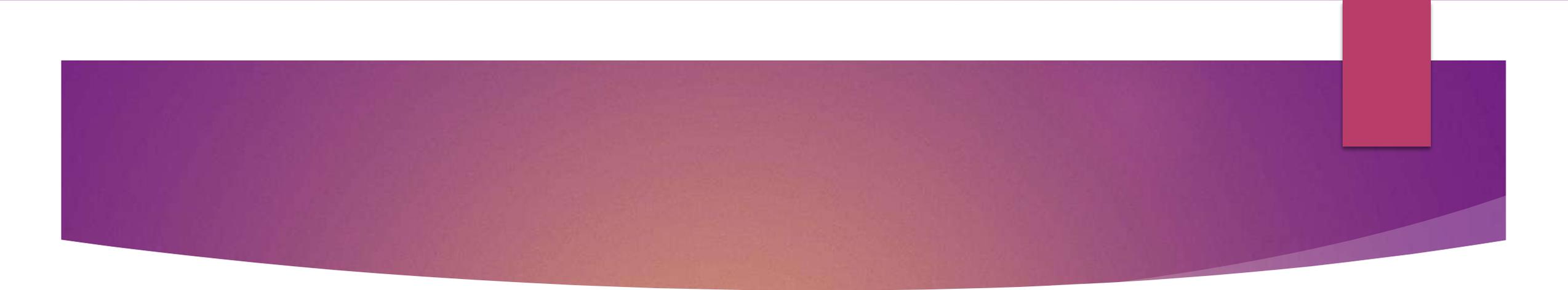


Timing Diagram



Sequence of Events

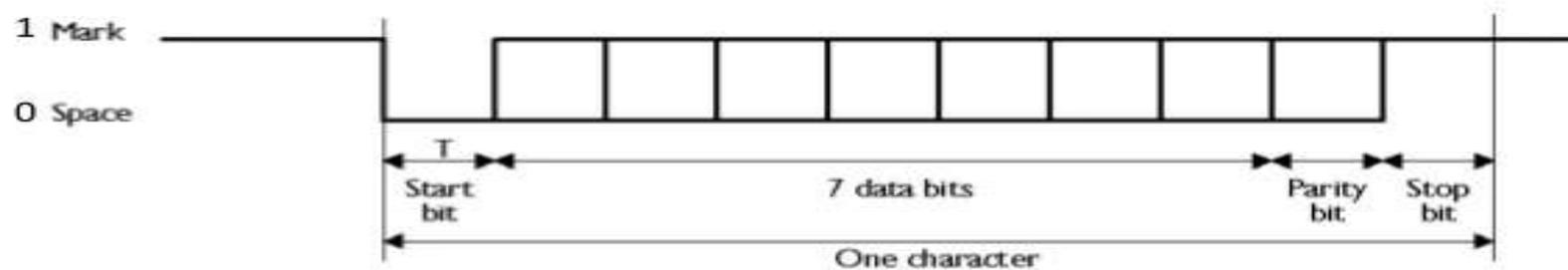


- 
- ▶ Data transfer between an interface and an I/O device is commonly controlled by a set of handshaking lines.
  - ▶ The handshaking scheme provides a high degree of flexibility because the successful completion of a data transfer relies on active participation of both units.
  - ▶ If a unit is faulty or the data transfer can not be performed during specified time the **time out error** is generated.

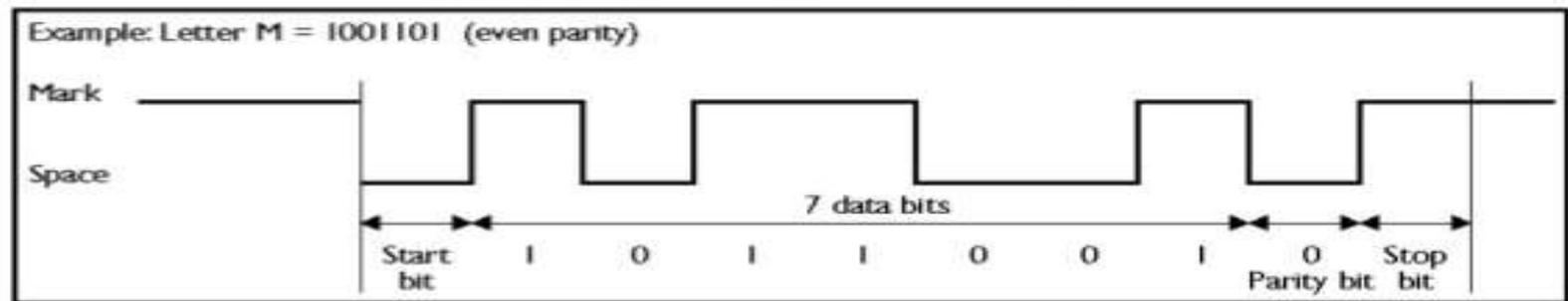
# Asynchronous Serial Data Transfer

- ▶ The transmitter and receiver generates their clock frequencies independently, but they are more or less identical.
- ▶ Generally adopted for low speed transmission.
- ▶ Data transmitted character by character.
- ▶ The time interval between two character is not fixed.
- ▶ The technique adopted for asynchronous serial data transmission is called **FRAMING**.
- ▶ In this technique each character carries the information of **START Bit** and **STOP Bit**.
- ▶ If there is no transmission in the line then the line is maintained in high voltage level – **MARK**.
- ▶ **START Bit** always is voltage level of **Zero (0) – SPACE**.
- ▶ After Start bit the data bits are transmitted followed by **PARITY Bit** and **STOP Bit**.

- ▶ The **Parity bit** may or may not present. If present always it is in either **Low state (0)** or **High state (1)** depending on **Parity generator**.
- ▶ The **Stop bit** may be **one bit** or **more bits**. Always they are in **High state (1)**.
- ▶ So, during transmission the data bits are framed with Start bit and Stop bits, called **Framing**.



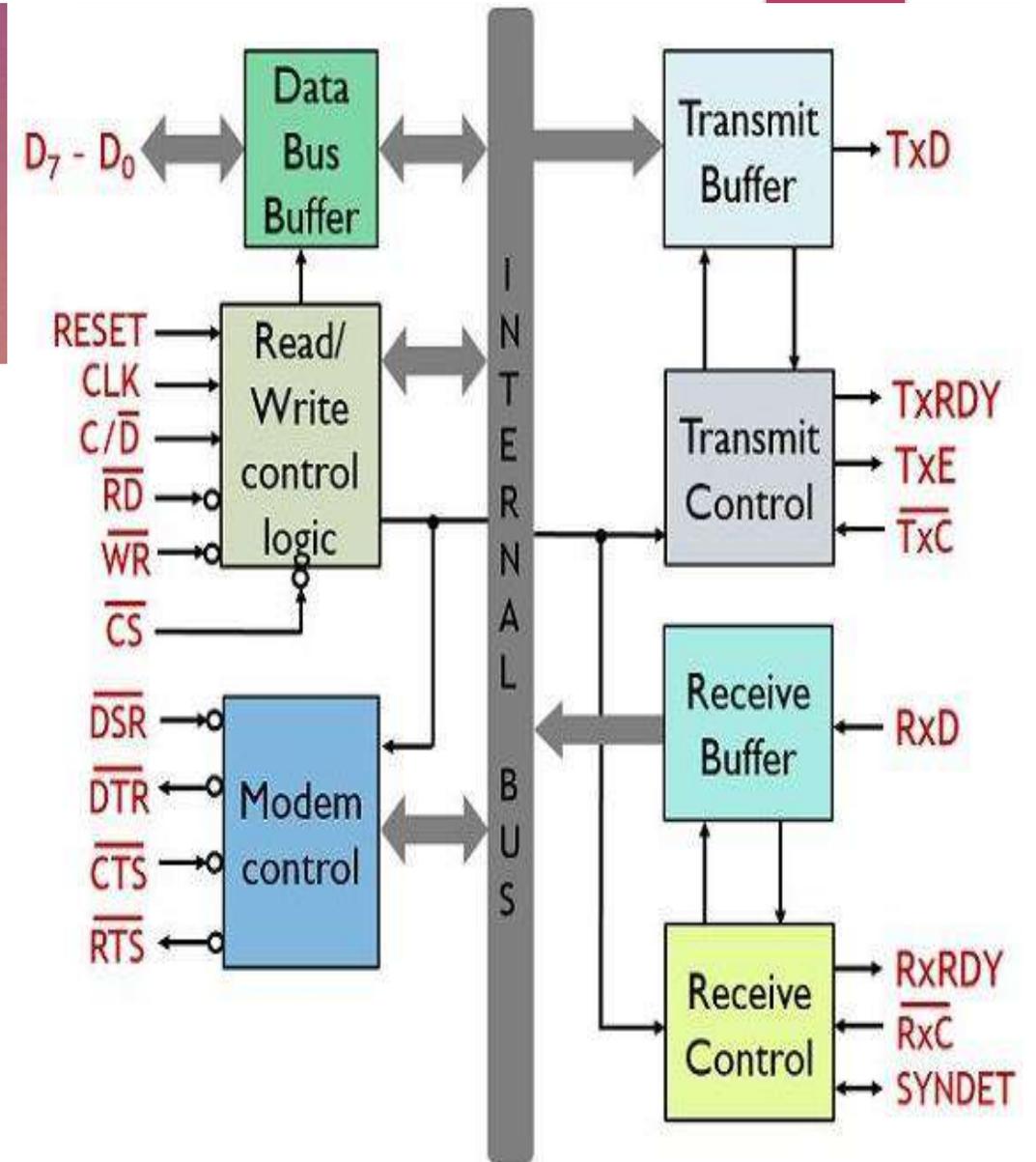
Serial transmission is **little endian** (least significant bit first)



<b>Synchronous Data Transfer</b>	<b>Asynchronous Data Transfer</b>
<b>1. Two units share a common clock.</b>	<b>1. Two units are independent and each unit will have their own private clock.</b>
<b>2. Data transfer between sender and receiver are synchronized by the same clock pulse.</b>	<b>2. Data transfer between sender and receiver is not synchronized by the same clock pulse.</b>
<b>3. Used between the devices that matches in speed.</b>	<b>3. Used between the devices that does not match in speed.</b>
<b>4. The time interval of transmission is constant (due to common clock)</b>	<b>4. The time interval of transmission is random (due to random common clock)</b>
<b>5. Bits are transmitted continuously to keep clock synchronized in both units.</b>	<b>5. Bits are sent only when it is available and the line remains idle when there is information is transmitted.</b>
<b>6. Fast.</b>	<b>6. Slow.</b>
<b>7. Costly.</b>	<b>7. Economical.</b>

# Communication Interface

- ▶ Serial transfer may be synchronous or asynchronous.
- ▶ The IC 8251A is a programmable chip designed for synchronous and asynchronous serial data communication.
- ▶ It is 25 pin DIP.
- ▶ It includes five blocks.
  - Data bus buffer
  - Read/Write control logic
  - Modem control
  - Transmitter
  - Receiver
- ▶ It is also called as Universal Synchronous Asynchronous Receiver and Transmitter (USART).

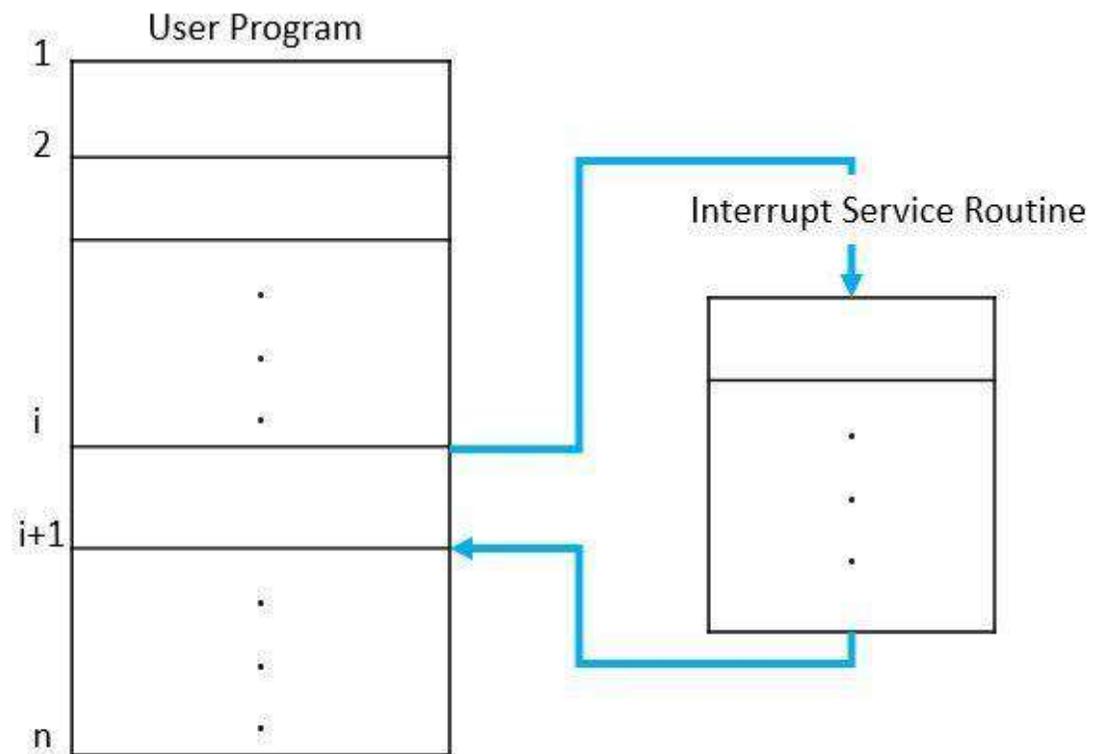


Architecture of 8251 USART

# Interrupt

- ▶ A computer must have some means to coordinate its activities with the external devices connected to it.
- ▶ When accepting characters from a keyboard, the computer needs to know when a new character has been typed.
- ▶ Similarly, during the output operation, it should send a character to a printer if the printer is ready to accept it.
- ▶ The current status of each I/O device connected to a computer is indicated by one or more bits of information.
- ▶ A program may run within the CPU to poll the device by testing the status bits before using an I/O instruction to transfer data.
- ▶ During this polling period the CPU does not perform any useful computation.
- ▶ So, an alternative arrangement is required that when the I/O device will ready, it will alert the CPU.
- ▶ This can be possible by sending a signal called **Interrupt**.

- ▶ An interrupt is a signal emitted by hardware or software that requests the processor to suspend its current execution and service the occurred interrupt.
- ▶ To service the interrupt the processor executes the corresponding interrupt service routine (**ISR**).
- ▶ After the execution of the interrupt service routine, the processor resumes the execution of the suspended program.

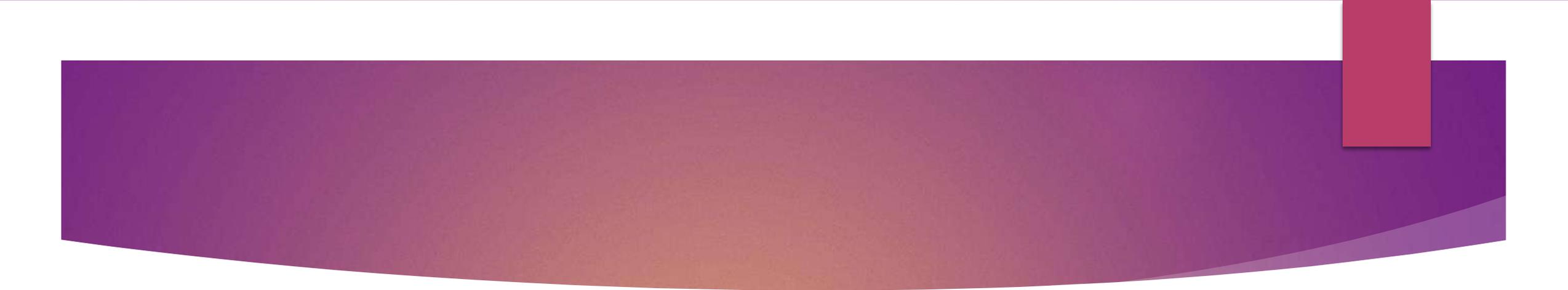


Transfer Control Via Interrupts

# Types of Interrupt

- ▶ There are two types of interrupt.
  - ❑ Hardware Interrupt
  - ❑ Software Interrupt
- ▶ **Hardware Interrupt:** If the signal for the processor is from external or internal device or hardware is called hardware interrupt.
- ▶ **Example:** From keyboard we will press the key to do some action, this pressing of key in keyboard will generate a signal which is given to the processor to do action, such interrupt is hardware interrupt.
- ▶ Hardware interrupt may be generated from external device/hardware or internal device/hardware.
- ▶ So, it may be **External interrupt** or **Internal interrupt**.

- ▶ **External Interrupt:** It comes from I/O devices, timing device, power failure or from any other external hardware.
- ▶ **Example:**
  - I/O device request for transfer of data.
  - I/O device finished transfer of data.
  - Elapse of time of an event – Time out
  - Power failure etc.
- ▶ **Internal Interrupt:** Arises illegal or erroneous use of an instruction or data and any overflow/underflow condition.
- ▶ **Example:**
  - Register overflow/underflow.
  - Stack/Queue overflow/underflow.
  - Attempt to divide by zero.
  - Invalid operation code etc.

- 
- ▶ Internal interrupt is initiated by some exceptional condition caused by the program itself rather than by an external event.
  - ▶ Internal interrupt is synchronous with the program while external interrupt is asynchronous.
  - ▶ If a program is rerun, the internal interrupt will occur in the same place each time. The external interrupt depends on external conditions that are independent of the program being executed at that time.

- ▶ The hardware interrupts may be maskable or non-maskable.
- ▶ **Maskable Interrupt:** The hardware interrupt that can be ignored or delayed for some time if the processor is executing a program with higher priority are termed as maskable interrupts.
- ▶ This type of interrupt is entertained by the CPU just after completion of current execution.
- ▶ **Non-Maskable Interrupt:** The hardware interrupts that can neither be ignored nor delayed and must immediately be serviced by the processor are termed as non-maskable interrupts.
- ▶ When a non-maskable interrupt arises, the CPU suddenly stops the current execution and jumps to entertain the interrupt.

- ▶ **Software Interrupts:** A software interrupt is initiated by executing an instruction.
  - ▶ This type of interrupt can be used by the programmer to initiate an interrupt procedure at any desired point in the program.
  - ▶ Software interrupt is a special CALL instruction that behaves like an interrupt rather than a subroutine call.
- 
- ▶ When an interrupt is initiated the Interrupt Service Routine (ISR) is executed.
  - ▶ The way the processor chooses the branch address of the service routine, the interrupt may be Non vectored interrupt or Vectored interrupt.
  - ▶ **Non vectored:** In this case the branch address is assigned to a fixed location in the memory where corresponding ISR is stored.
  - ▶ **Vectored:** The source of the interrupt that supplies the branch information to the computer.
  - ▶ This information is called the interrupt vector.

# Modes of Data Transfer

- ▶ The data transfer between CPU and I/O devices may be handled in a variety of modes.
- ▶ Some modes use the CPU as an intermediate path, others may transfer data directly to and from the memory unit.
- ▶ Data transfer to and from peripherals may be handled in one of the three possible modes.
  - ❑ Programmed I/O
  - ❑ Interrupt initiated I/O
  - ❑ Direct Memory Access (DMA)

# Programmed I/O

- ▶ Programmed I/O operations are results of I/O operations written in computer program.
- ▶ Each data item transfer is initiated by an instruction in the program.
- ▶ **Example:** In this case, the I/O device does not have direct access to the memory unit. A transfer from I/O device to memory requires the execution of several instructions by the CPU, including an input instruction to transfer the data from device to the CPU and store instruction to transfer the data from CPU to memory.
- ▶ Transferring data under program control requires constant monitoring of the peripheral by the CPU.
- ▶ The CPU stays in the program loop (**polling**) until the I/O unit indicates that it is ready for data transfer.
- ▶ This is a time consuming process since it needlessly keeps the CPU busy.
- ▶ This situation can be avoided by using an interrupt facility.

**X = A + B**

**PUSH A**

**PUSH B**

**ADD**

**STORE X**

# Interrupt Initiated I/O

- ▶ In programmed I/O, the CPU remain busy unnecessarily.
- ▶ This situation can very well be avoided by using an interrupt driven method for data transfer.
- ▶ When the interface determines that the device is ready for data transfer, it generates an **interrupt request** signal to the CPU.
- ▶ In the meantime the CPU can proceed for any other program execution.
- ▶ Upon detection of an external interrupt signal the CPU momentarily stops the execution of current task, the control branches to a service routine to process the required I/O transfer, and then return to the task it was originally performing.
- ▶ But what if multiple devices generate interrupts simultaneously.

# Priority Interrupt

- ▶ In that case, there must have a way to decide which interrupt is to be serviced first.
- ▶ In other words, the priority must be assigned to all the devices for systemic interrupt servicing.
- ▶ The concept of defining the priority among devices so as to know which one is to be serviced first in case of simultaneous requests is called **priority interrupt system**.
- ▶ Generally devices with high speed of data transfer, such as, magnetic disks are assigned high priority than slow devices, such as, Keyboard.
- ▶ When multiple devices interrupt the system at the same time, the service given to the device with highest priority.
- ▶ This could be done with either **software or hardware methods**.

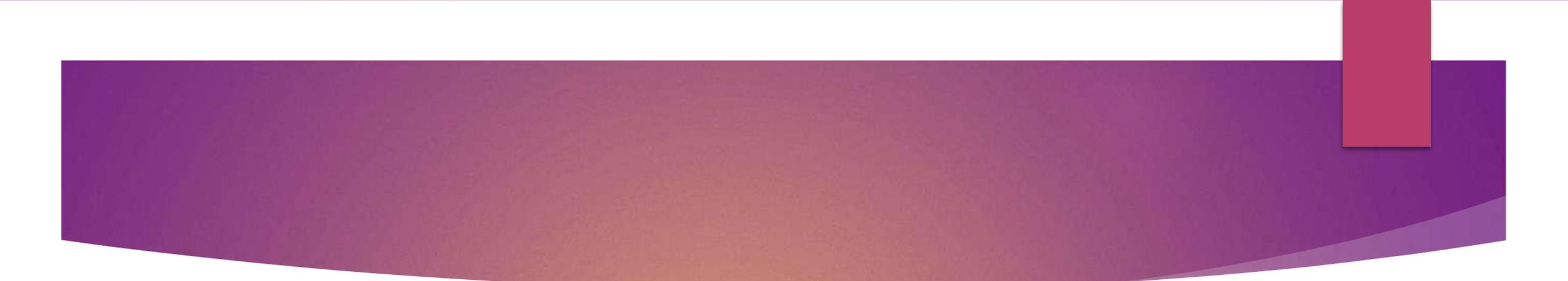
# Software Priority Method

- ▶ In this method a Polling procedure is used to identify the highest priority device by software means.
- ▶ There is one common branch address for all interrupt to execute the ISR.
- ▶ The program that takes care of interrupts begins the execution of the ISR and Polls the interrupt sources in sequence.
- ▶ The order in which they are tested determines the priority of the interrupt and its corresponding device.
- ▶ If the highest priority device interrupt signal is on, then the service is given to that device.
- ▶ Otherwise, the next lower priority source is tested and so on.
- ▶ The major disadvantage of this method is that it is quite slow.
- ▶ To overcome this, we can use hardware priority interrupt.

# Hardware Priority Method

- ▶ A hardware priority interrupt unit functions as an overall manager in an interrupt system environment.
- ▶ To speed up the operation, each interrupt source has its own interrupt vector to access its own ISR directly.
- ▶ So, no polling is required as all the decisions are established by the hardware priority interrupt unit.
- ▶ The hardware priority function can be established by either a **serial** or a **parallel** connection of interrupt lines.
- ▶ The **serial connection** of hardware priority interrupt method is known as **Daisy-Chaining Method**.



- 
- ▶ The daisy-chaining method involves connecting all the devices that can request an interrupt in a serial manner.
  - ▶ This configuration is governed by the priority of the devices.
  - ▶ The device with the highest priority is placed first position (nearer to CPU) followed by lower priority device and so on up to the lowest priority device.
  - ▶ There is an interrupt request line which is common to all the devices and goes into the CPU.
  - ▶ If any of the device raises an interrupt, it places the **LOW (0)** state in the interrupt request line.
  - ▶ When no interrupts are pending, the line is in **HIGH (1)** state.
  - ▶ The CPU responds to an interrupt request by enabling the interrupt acknowledge line.
  - ▶ This signal is received at the **PI (Priority in)** input of device 1.

- ▶ If the device has not requested the interrupt or no pending request, it passes this signal to the next lower priority device through its **PO (priority out)** output as **(PI = 1 & PO = 1)**.
- ▶ However, if the device had requested the interrupt, **(PI = 1 & PO = 0)**.
  - The device consumes the acknowledge signal and block its further use by placing **0** at its **PO (priority out)** output.
  - The device then places its interrupt vector address (VAD) into the processor bus.
  - The device puts its interrupt request signal in **HIGH (1)** state to indicate its interrupt has been taken care of.

NOTE: VAD is the address of the service routine which services that device.

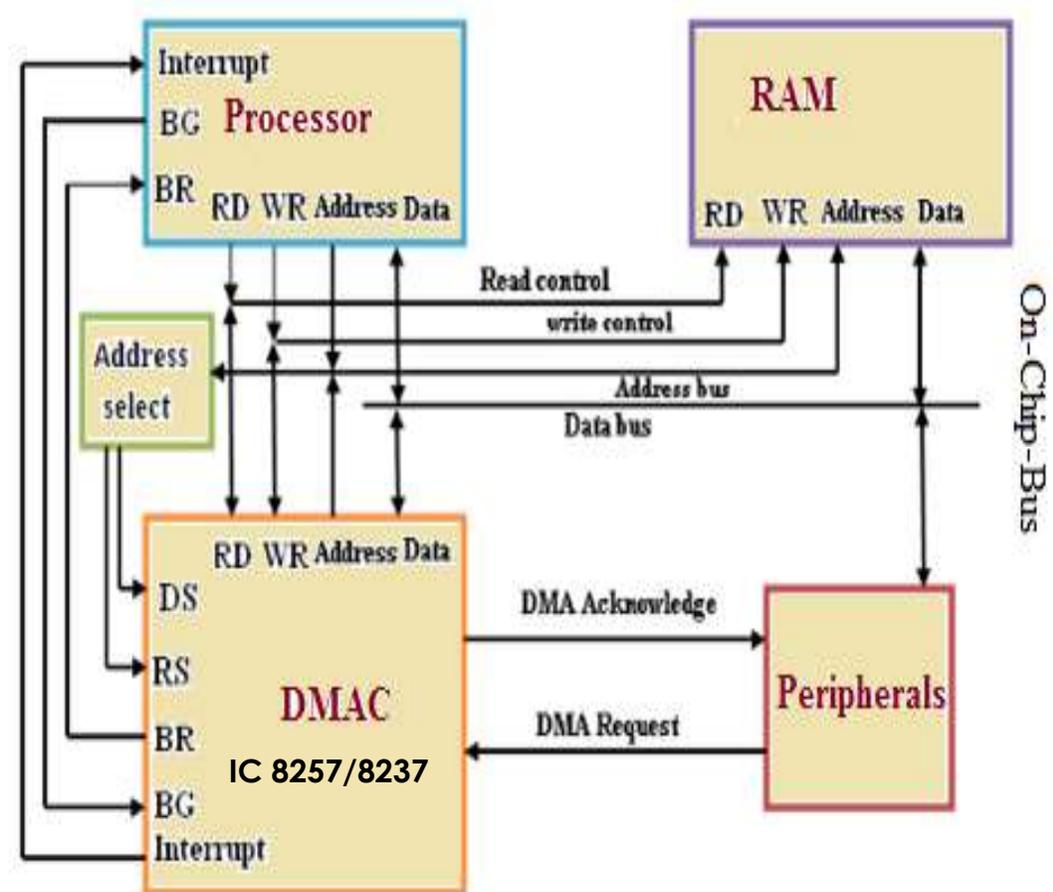
- ▶ If a device gets 0 at its **PI input**, it generates 0 at the **PO** output to tell other devices that acknowledge signal has been blocked, that is, (**PI = 0 & PO = 0**).
- ▶ Therefore, by daisy chain arrangement we have ensured that the highest priority interrupt gets serviced first and have established a hierarchy.
- ▶ **HW: Parallel connection of hardware priority interrupt**

# Direct Memory Access (DMA)

- ▶ The programmed I/O (**CPU has to check for I/O device**) and Interrupt-driven I/O (**CPU doesn't has to check for I/O device, I/O device generate the interrupt signal**) require the active intervention of the processor to transfer data between memory and the I/O module, and data transfer must traverse through the processor.
- ▶ DMA is a process of communication for data transfer between memory and input/output device, controlled by an external circuit called **DMA controller**, without involvement of CPU.
- ▶ DMA is a way to improve processor activity and I/O transfer rate by taking-over the job of transferring data from processor and letting the processor to do other tasks.
- ▶ The DMA controller takes over the buses to manage the transfer directly between the I/O devices and the memory unit.
- ▶ DMA controller is a hardware unit that allows I/O devices to access memory directly without the participation of the processor.

# DMA Controller

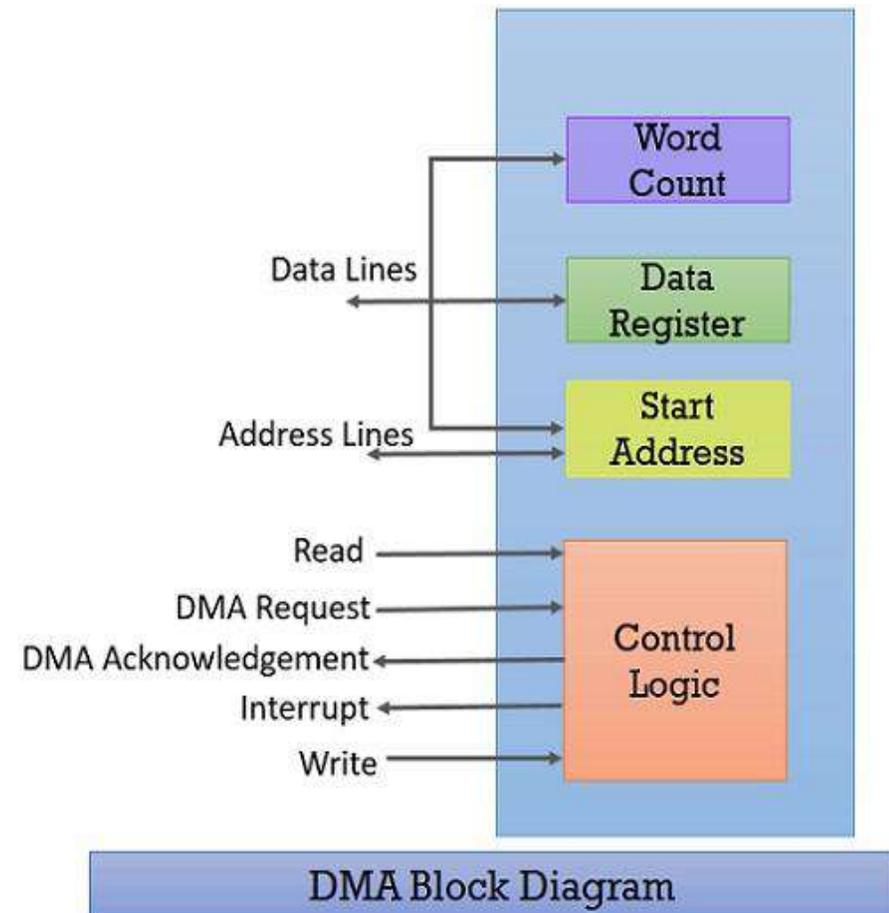
- ▶ Whenever an I/O device wants to transfer the data to or from memory, it sends the **DMA request (DRQ)** to the DMA controller.
- ▶ DMA controller accepts this DRQ and asks the CPU to hold for a few clock cycles by sending it to the CPU **Bus Request (BR)** or **Hold request (HLD)** signal.
- ▶ **Bus Request (BR)** or **Hold request (HLD)**: It is used by the DMA controller to request the CPU to relinquish the control of the buses.



- ▶ After receiving Bus Request signal from DMA controller, the CPU acknowledges it by granting **Bus Grant (BG)** or **Hold acknowledgement (HLDA)** to the DMA Controller.
- ▶ **Bus Grant (BG)** or **Hold acknowledgement (HLDA)**: It is activated by the CPU to inform the DMA controller that the DMA controller can take control of the buses.
- ▶ After receiving the **Hold acknowledgement (HLDA)**, DMA controller **acknowledges I/O device (DACK)** that the data transfer can be performed to or from memory and DMA controller takes the charge of the system bus.
- ▶ When the data transfer is completed, the DMA controller raise an **Interrupt** to the processor that the task of data transfer is finished and the processor can take control over the buses again and it will start processing where it has left.

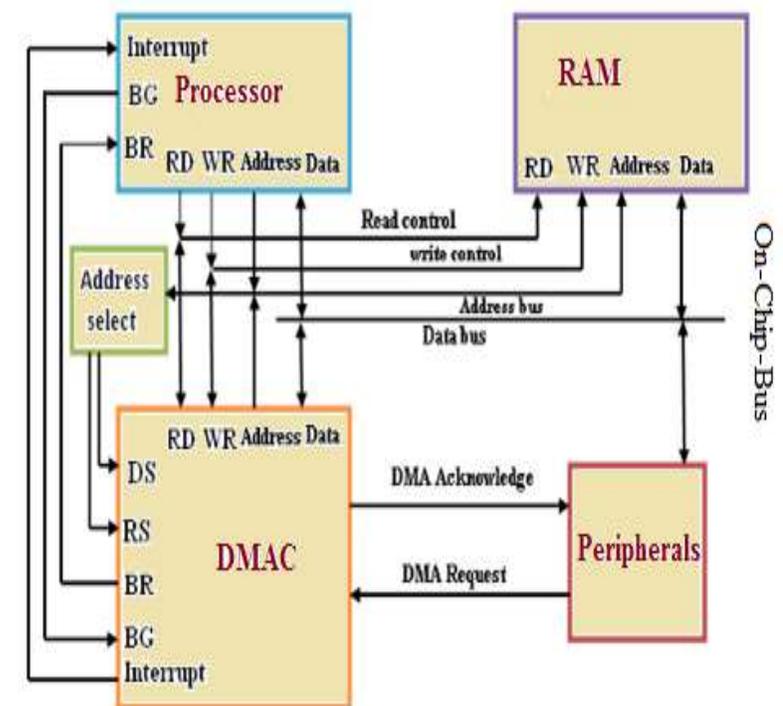
# DMA block diagram

- ▶ The DMA controller has an address register, a word/data count register, data register and a control logic unit.
- ▶ Whenever a processor is requested to read or write a block of data, it instructs the DMA controller by sending the following information.
- ▶ The first information is whether the data has to be read from memory or the data has to write to the memory.
- ▶ It passes this information via read or write control line that is between the processor and DMA controller **Control logic unit**.



- ▶ The processor also provides the **starting address** of the data block in the memory, from where it has to be read or where the data block has to be written in memory.
- ▶ DMA controller stores this in its **address register**. It is also called the **starting address register**.
- ▶ The processor also sends the word count, that is, how many words are to be read or written.
- ▶ The DMA controller stores this information in the **data count** or **word count register**. The register count is decremented by one after each word transfer and internally it is tested for zero.
- ▶ The most important is the **address** of I/O device that wants to read or write data. This information is stored in the **data register**.

- ▶ The CPU will not directly provide the BG to DMA Controller. First the registers in the DMA controller are selected by the CPU through the address bus by enabling the **DS (DMA Select)** and **RS (Register Select)** inputs.
- ▶ Then it will provide the basic information of address register, a word count register and a control register.
- ▶ When the BG (Bus Grant) input is 0, the CPU can communicate with the Data Bus to read from or write to the DMA registers.
- ▶ When BG =1, the CPU relinquish the buses and the DMA controller can communicate directly with the memory by specifying an address in the address bus and activating the RD or WR control.



# DMA Data Transfer

## 1. Burst Mode:

- Here, once the DMA controller gains the charge of the system bus, an entire block of data is transferred in one contiguous sequence.
- Then it releases the system bus to the CPU only after completion of data transfer.
- Till then the CPU has to wait for the system buses. CPU remains inactive relatively long period of time.
- It can perform any work without the requirement of system bus.
- Basically used to transfer large volume of data.
- It is also called as **Block transfer mode**.

## 2. Cycle Stealing Mode:

- In this mode, the data transfer is byte by byte or word by word.
- The DMA controller forces the CPU to stop its operation and relinquish the control over the bus for a short period to DMA controller.
- After the transfer of every byte, the DMA controller releases the bus and then again requests for the bus.
- This will continue to complete the data transfer of entire block.
- In this way, the DMA controller steals the clock cycle for transferring every byte.
- Basically used to transfer very small volume of data.
- More time required to transfer total block of data.
- But, CPU utilization is more than the Burst mode.

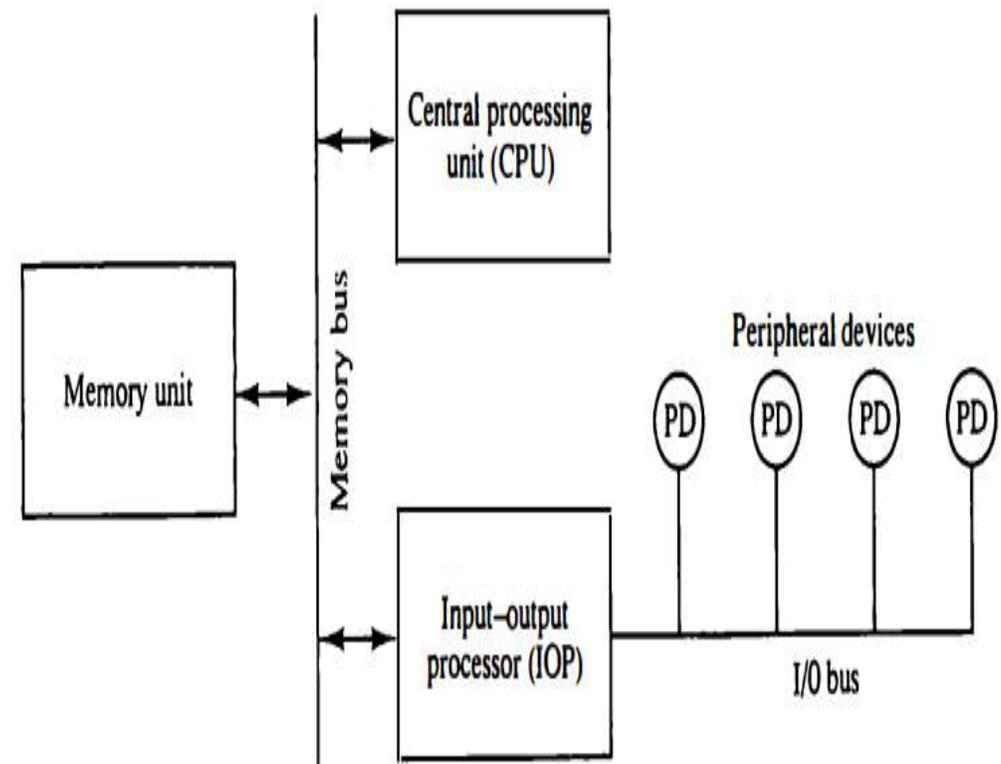
### 3. Transparent Mode

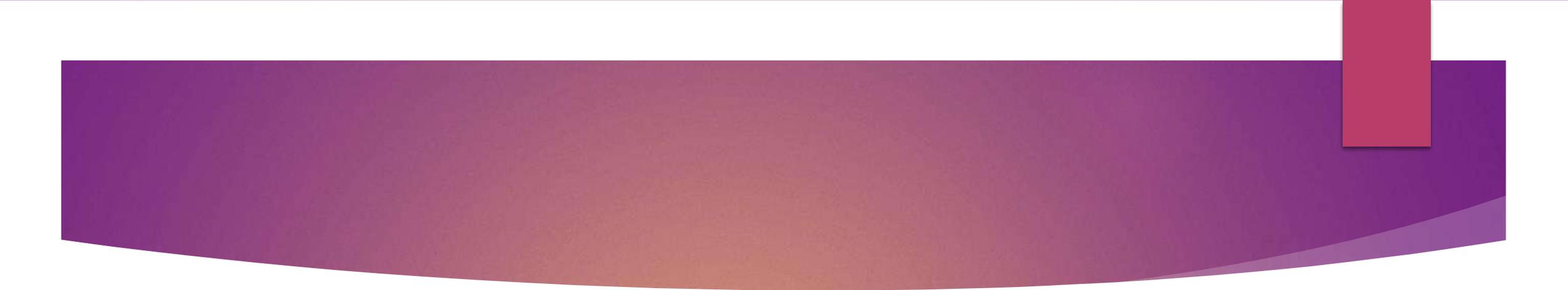
- DMA controller takes charge of system bus for data transfer only when CPU performing operations without using system bus.
- CPU executes the program and DMA controller takes charge of data transfer.
- When the CPU requires the system bus the DMA controller releases the control over buses.
- Complex hardware.

<b>Advantages</b>	- allows a peripheral device to read from/write to memory without going through the CPU
	- allows for faster processing since the processor can be working on something else while the peripheral can be busy with memory
<b>Disadvantages</b>	- requires a DMA controller to carry out the operation, which increases the cost of the system
	- cache coherence problems

# I/O Channel and Processor (IOP)

- ▶ Instead of having each interface communicate with the main processor, a computer may incorporate one or more external processor and assign them to communicate directly with all I/O devices.
- ▶ This secondary processor is called as I/O channel or I/O processor (IOP).
- ▶ The communication between the IOP and the devices is similar to the program control method of transfer.
- ▶ And the communication of IOP with the memory is similar to the direct memory access method.



- 
- ▶ In a large computer, each processor is independent of other processors, and any processor can initiate operations.
  - ▶ The IOP operates independent from CPU and transfer data between peripherals and memory.
  - ▶ The IOP is similar to a CPU except that it is designed to handle the I/O communication.
  - ▶ The CPU can act as master and the IOP act as slave processor.
  - ▶ The CPU assigns the task of initiating operations but it is the IOP, who executes.
  - ▶ CPU instructions provide operations to start an I/O transfer.
  - ▶ The IOP asks for CPU through interrupt.

# Lecture of Module 5

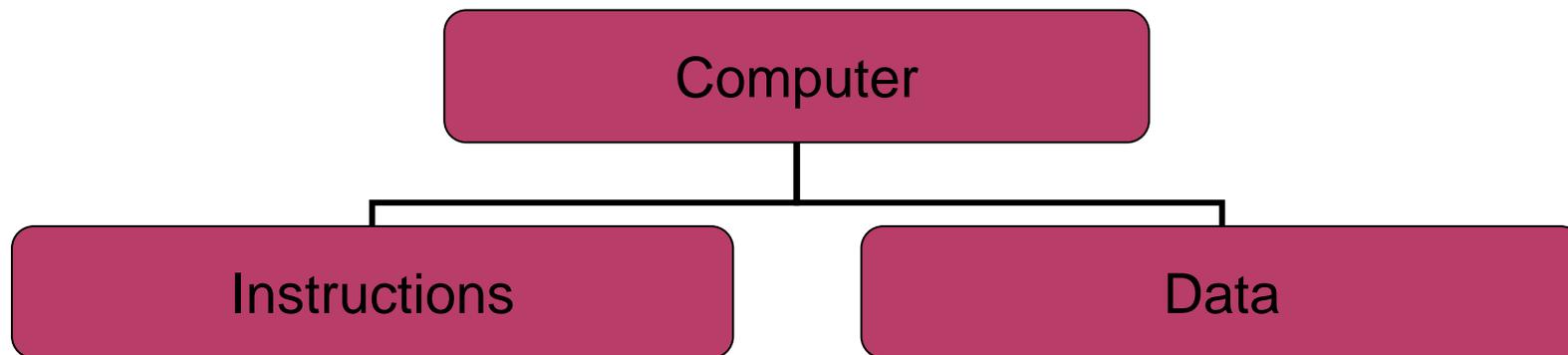
## **Parallel Processing**

# Overview

- ▶ **Introduction**
- ▶ **Flynn's Classification**
- ▶ **Pipeline Architecture**
- ▶ **Pipeline Hazard**
- ▶ **RISC Architecture**
- ▶ **CISC Architecture**

# Introduction

- ▶ Computation is execution of instructions that operates on data.
- ▶ Computer architecture can be classified based on the number of instructions that can be executed and how they operate on data.



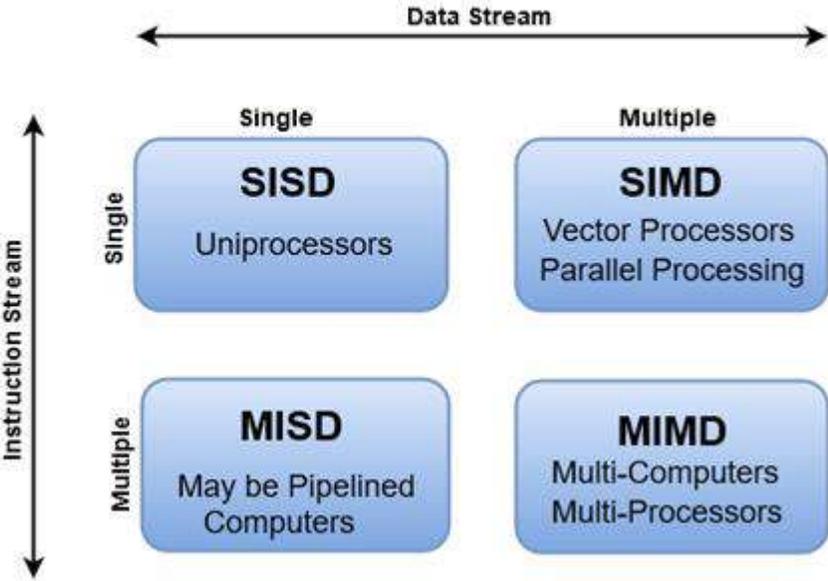
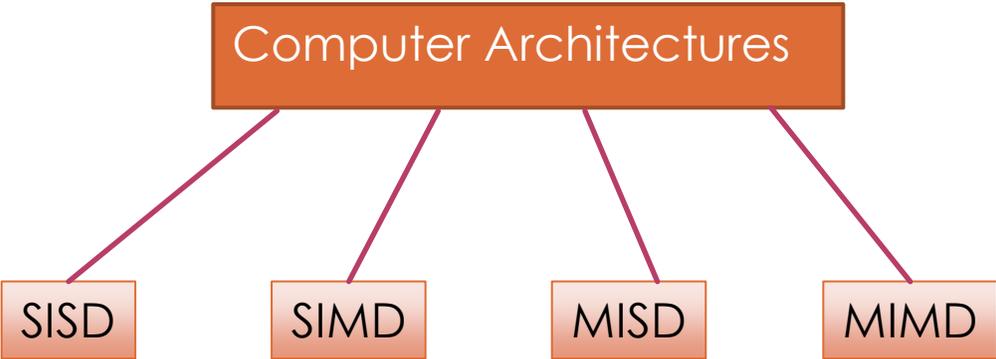
# An Overview of Parallel Processing

- ▶ What is parallel processing?
  - Parallel processing is a method to improve computer system performance by executing two or more instructions simultaneously.
- ▶ The goals of parallel processing.
  - One goal is to reduce the “wall-clock” time or the amount of real time that you need to wait for a problem to be solved.
  - Another goal is to solve bigger problems that might not fit in the limited memory of a single CPU.

# Flynn's Classification

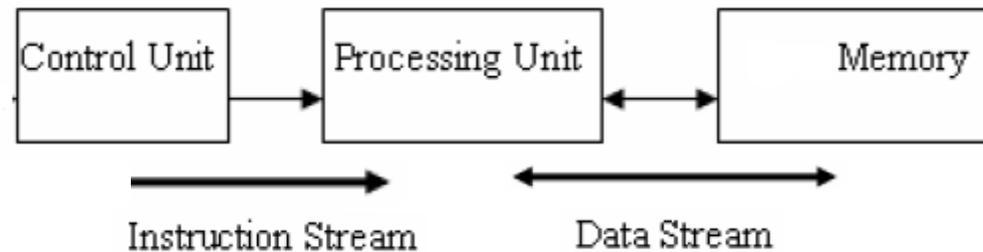
- ▶ In general, digital computer can be classified in to four categories according to the multiplicity of instruction and data stream.
- ▶ The classification was proposed by Michael J. Flynn in 1966.
- ▶ It is the most commonly accepted taxonomy of computer organization.
- ▶ In this classification, computers are classified by whether it processes a single instruction at a time or multiple instructions simultaneously, and whether it operates on one or multiple data sets.
- ▶ Flynn proposed the following categories of computer systems.
  - ❖ Single Instruction Single Data (SISD)
  - ❖ Single Instruction Multiple Data (SIMD)
  - ❖ Multiple Instructions Single Data (MISD)
  - ❖ Multiple Instructions Multiple Data (MIMD)

### Flynn's Classification of Computers



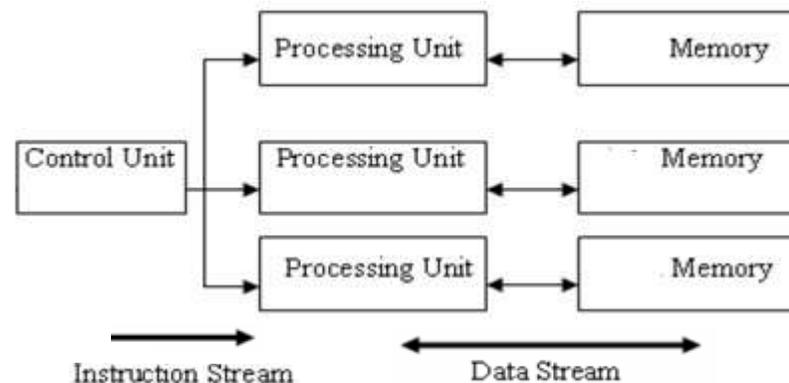
# SISD

- ▶ SISD machines executes a single instruction on individual data values using a single processor.
- ▶ Traditional sequential architecture.
- ▶ Based on traditional Von Neumann uniprocessor architecture, instructions are executed sequentially or serially, one step after the next.
- ▶ Until most recently, most computers are of SISD type.
- ▶ Parallelism can be achieved by Pipelining.



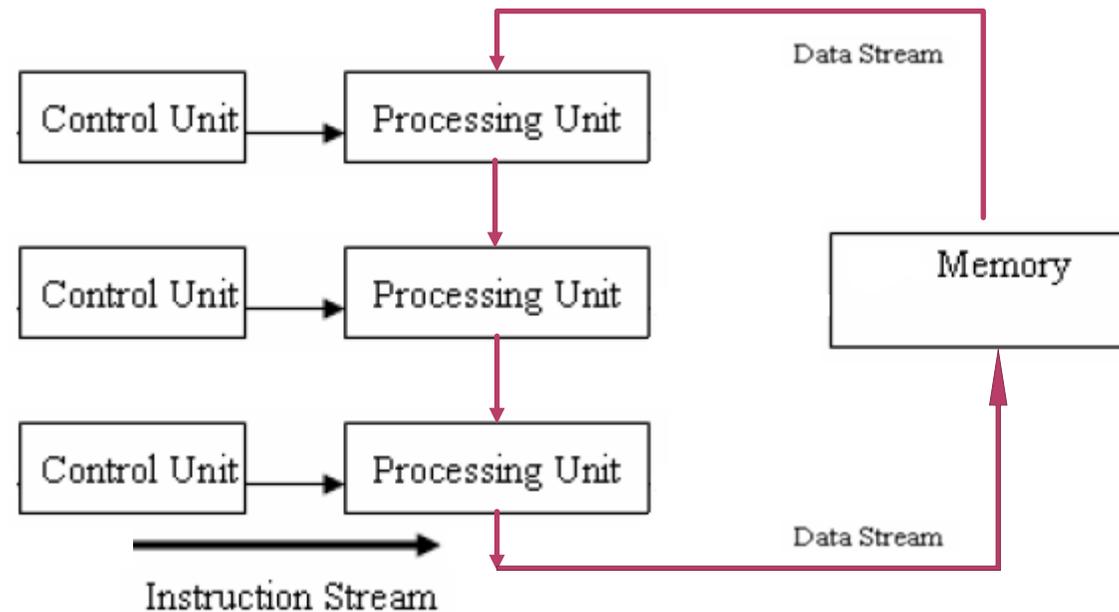
# SIMD

- ▶ An SIMD machine executes a single instruction on multiple data values simultaneously using many processors.
- ▶ Since there is only one instruction, each processor does not have to fetch and decode each instruction. Instead, a single control unit does the fetch and decoding for all processors.
- ▶ SIMD architectures include array processors (ILLIAC-IV).
- ▶ Shared memory system may contain multiple memory modules.



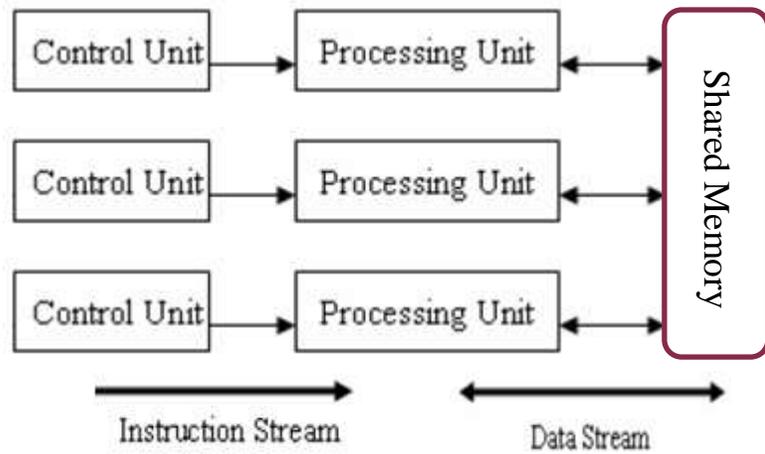
# MISD

- ▶ Multiple operations on same data.
- ▶ The result from one processing unit is input to next processing unit.
- ▶ It is not clear till now whether such machine exists or not.
- ▶ Some people regards Pipeline architecture, Systolic array machine as MISD.

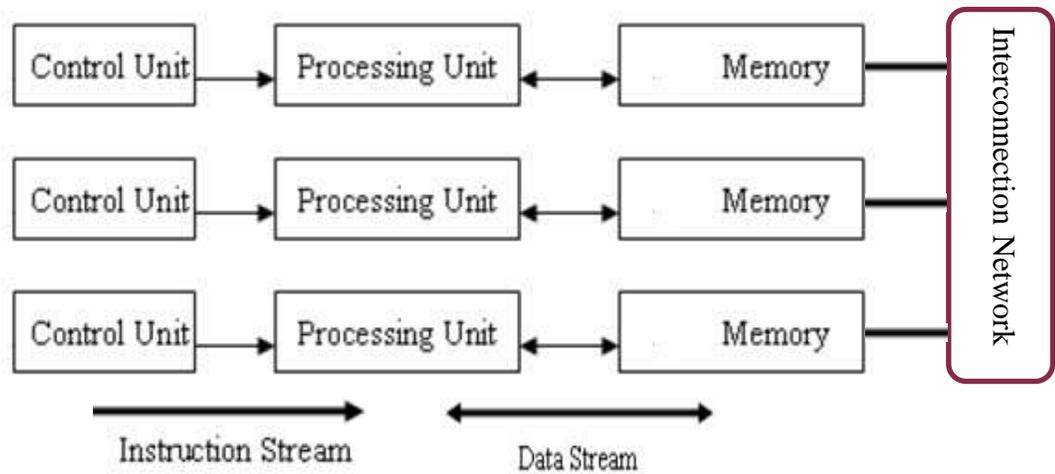


# MIMD

- ▶ Multiple independent CPUs are there.
- ▶ It may execute multiple instructions simultaneously, contrary to SIMD machines.
- ▶ Multiple instruction streams operating on multiple data streams.
- ▶ Each processor must include its own control unit that will assign to the processor parts of a task or a separate task.
- ▶ It may look like multiple SISD (MSISD).
- ▶ MIMD machines are usually referred to as Multiprocessor or Multicomputer.
- ▶ It may have shared memory architecture or distributed memory architecture.
- ▶ It may be tightly coupled or loosely coupled.
- ▶ Example: CRAY X-MP, IBM SP/2, CRAY T3E.



Multiprocessor Architecture  
Shared Memory Architecture  
Tightly Coupled



Multicomputer Architecture  
Distributed Memory Architecture  
Loosely Coupled

# Interconnection Topology

- A system may also be classified by its topology.
- A topology is the pattern of connections between processors.
- The cost-performance tradeoff determines which topologies to use for a multiprocessor system.



Whether parallelism can be achieved in uni-processor system?

Yes

Various techniques are there-

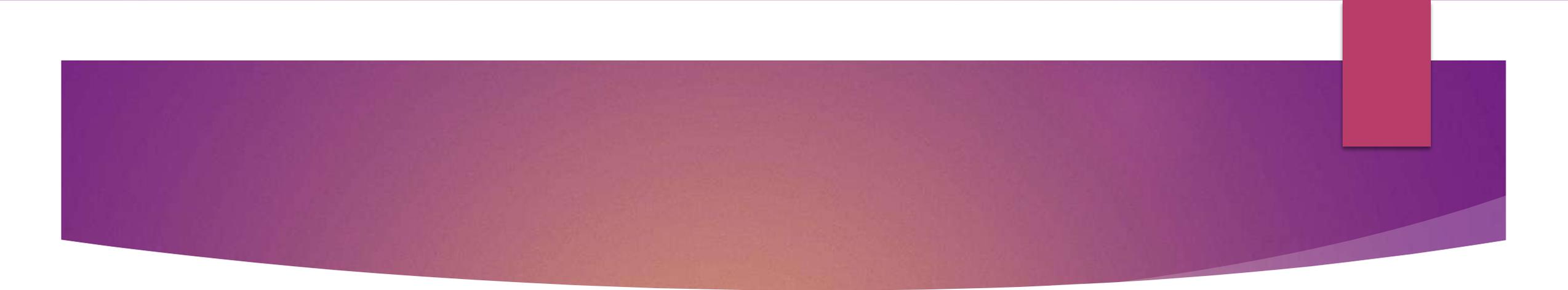
**Pipelining** is one of them

# Pipeline

## What is Pipeline?

- ▶ To complete operation of an instruction some sub-operations are to be performed.
- ▶ Generally sub-operations are-
  - Instruction fetch
  - Instruction decode
  - Operand fetch
  - Execution etc.



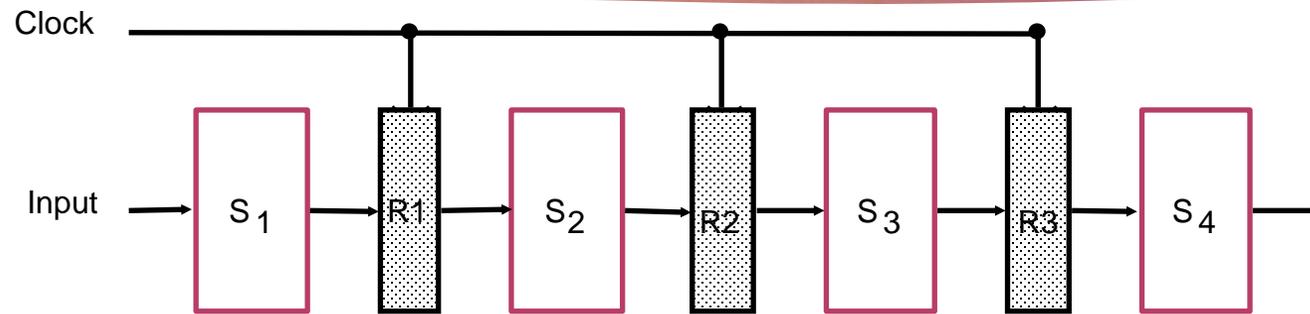
- 
- ▶ Whether the performance of the system enhanced if each sub-operation is performed in a dedicated segment?

That gives the concept of **Pipeline**

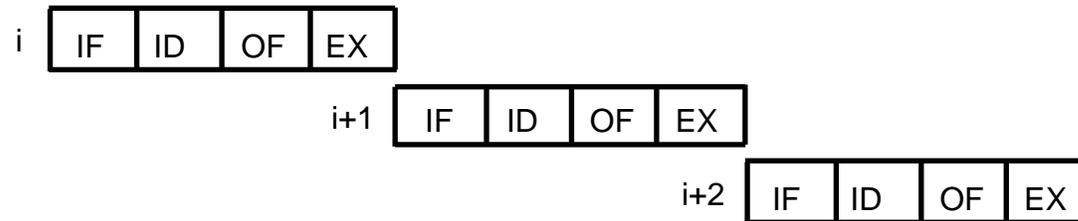
- ▶ Pipelining is used by all modern microprocessors to enhance the performance by overlapped execution of instructions.
- ▶ Analogous to fluid flow in pipelines and assembly line in factories.
- ▶ It is a technique of decomposing a sequential process into sub-operations with each sub-operation being in a dedicated segment.
- ▶ Each segment performs partial processing and is transferred to the next segment in the pipeline.
- ▶ The final result is obtained after the instructions have passed through all segments.

- ▶ If the stages of a pipeline are not balanced and one stage is slower than another, the entire throughput of the pipeline is affected.
- ▶ The overlapping of computation is made possible by associating register with each segment in the pipeline.
- ▶ Registers provide isolation between each segment so that each can operate on distinct data simultaneously.
- ▶ In a non pipelined computer, these steps must be completed before the next instruction can be issued.
- ▶ Pipelining doesn't help latency of single instruction, it helps throughput of entire workload.
- ▶ Time taken by each stage is  $t_p$  and that is to be decided first.

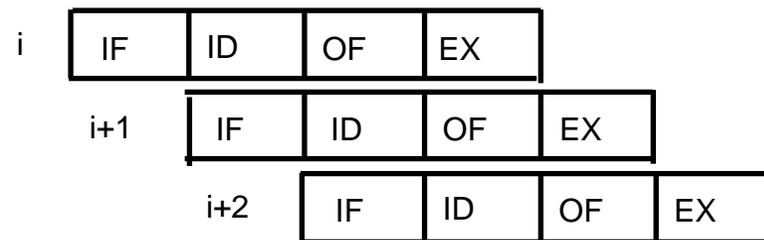
# General Structure of Pipeline



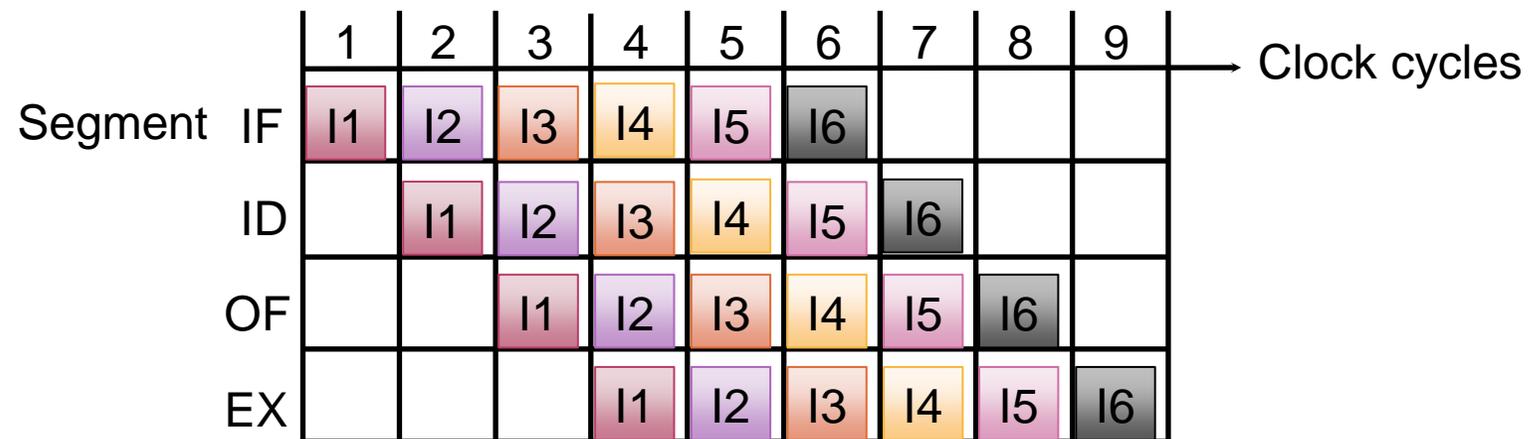
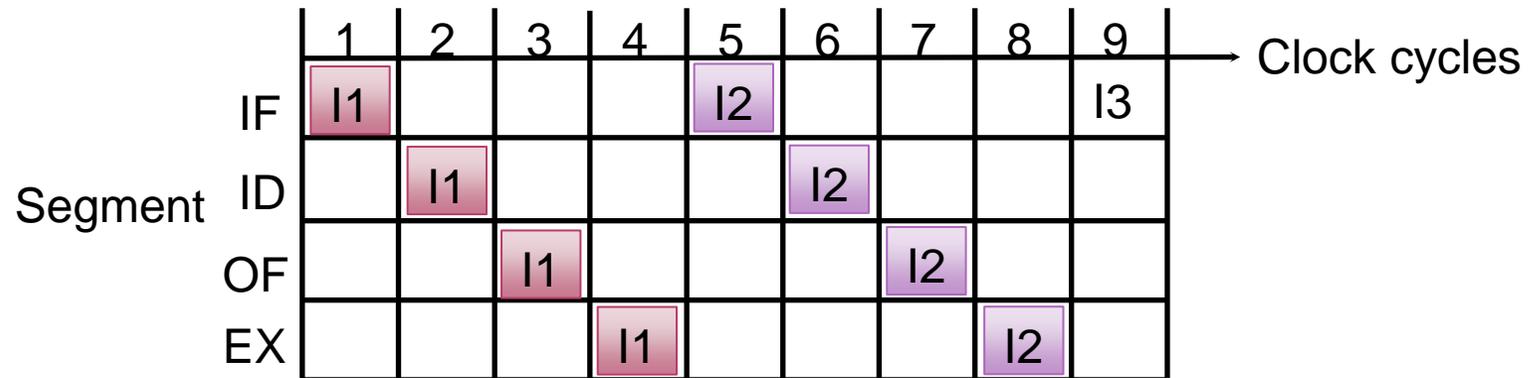
Non-Pipeline



Pipeline



# Space-Time Diagram



# Non-pipeline vs. Pipeline

n: Number of tasks to be performed

Conventional Machine (Non-Pipelined)

$t_n$ : Clock cycle

$t_1$ : Time required to complete the n instructions

$$t_1 = n * t_n$$

Pipelined Machine (k stages)

$t_p$ : Clock cycle (time to complete each sub operation)

$t_k$ : Time required to complete the n instructions

$$t_k = (k + n - 1) * t_p$$

Speedup ( $S_k$ )

$$S_k = n * t_n / (k + n - 1) * t_p$$

$$\lim_{n \rightarrow \infty} S_k = \frac{t_n}{t_p} = k, \text{ if } t_n = k * t_p$$

$S_k = k$ , this is the maximum theoretical Speed up that a Pipeline can achieve.

Efficiency ( $E_k$ )

$$E_k = \frac{S_k}{k} = \frac{n \cdot t_n}{k(k+n-1) \cdot t_p} = \frac{n}{k+(n-1)}$$

If  $n \gg k$  then  $E_k = 1$

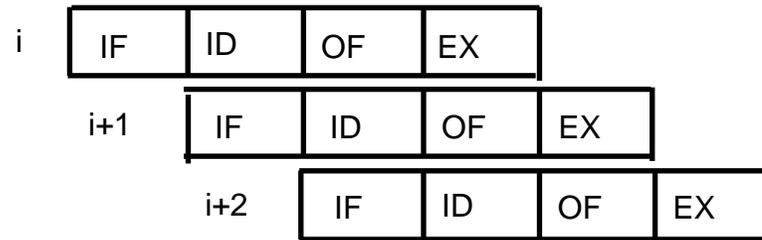
Throughput ( $H_k$ )

The number of instructions completed per unit time

$$H_k = \frac{n}{[k+(n-1)]t_p} = \frac{E_k}{t_p} \quad \text{If } E_k = 1 \text{ then } H_k = \frac{1}{t_p}$$

Pipelined system is better than non-pipelined system

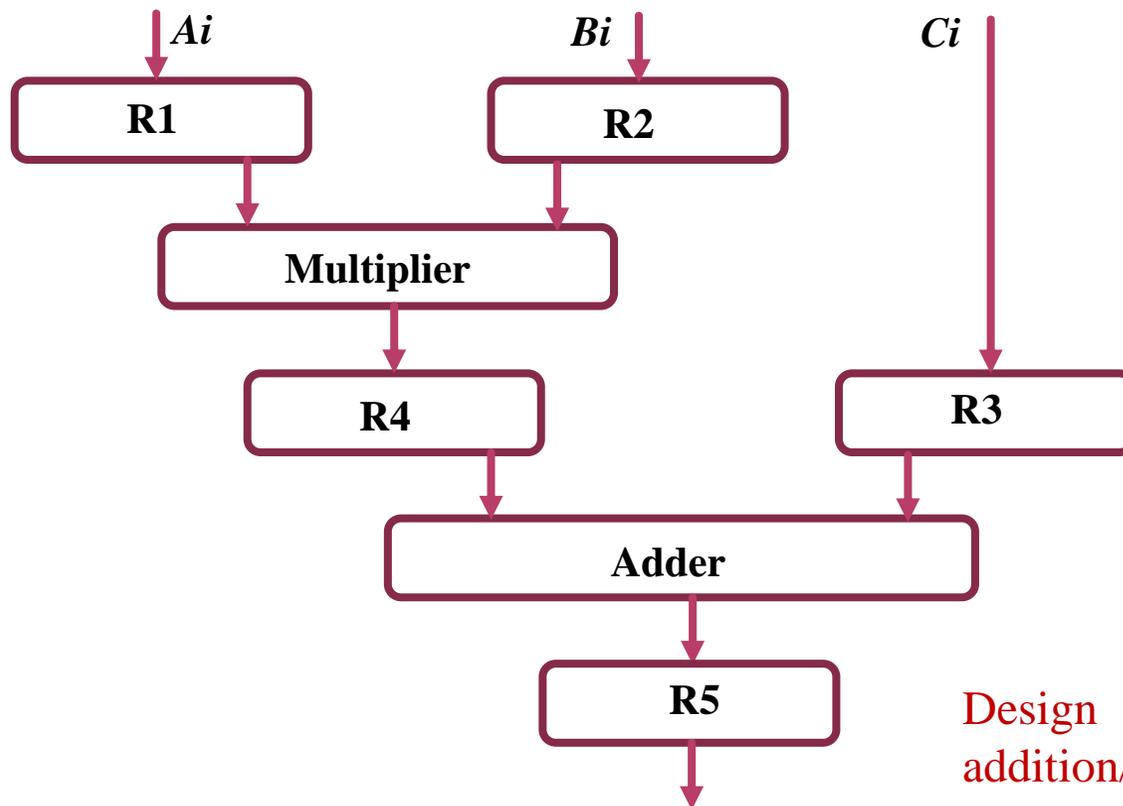
- ▶ Generally there are two areas of computer design where the pipeline mostly applicable.
  - Instruction pipeline
  - Arithmetic pipeline
- ▶ The pipeline what are discussed is **Instruction pipeline**.



- ▶ The **Arithmetic pipeline** divides an arithmetic operation into sub-operations for execution in the pipeline segments.
- ▶ Pipelined arithmetic units are usually found in very **high speed special purpose** computers.
- ▶ Generally they are used to solve scientific, complex problems etc.
- ▶ An array multiplier is an example of pipelined multiplier.

# Example of Arithmetic Pipeline

- ▶ Suppose we want the operation  $A_i \times B_i + C_i$  for  $i = 1, 2, 3, \dots, n$



Design arithmetic pipeline for floating point addition/subtraction and multiplication/division.

# Comments about Pipeline

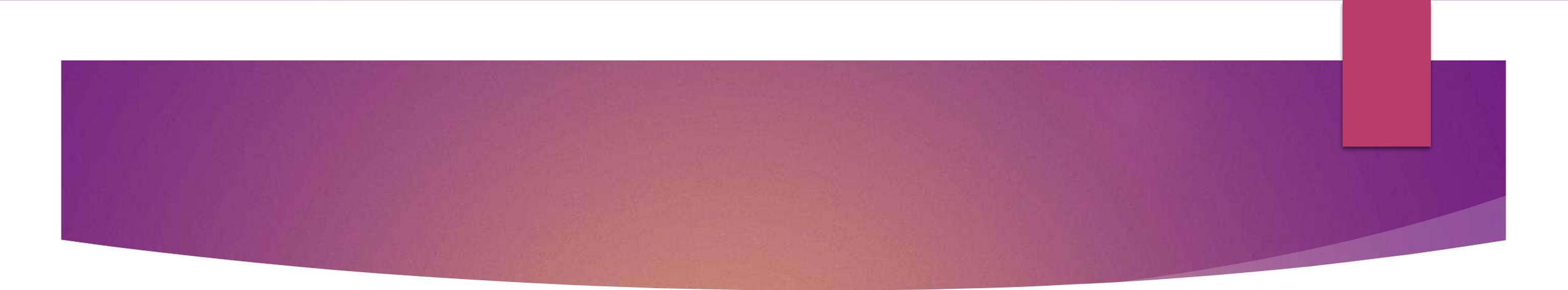
- ❑ The good news
  - ❑ Multiple instructions are being processed at same time
  - ❑ Best case speedup is equal to number of stages of the pipeline
- ❑ The bad news
  - ❑ Instructions interfere with each other and unable to proceed at their designated  $t_p$  - **Hazard**
    - Example: different instructions may need the same piece of hardware (e.g., memory) in same clock cycle – **Structural Hazard**
    - Example: instruction may require a result produced by an earlier instruction that is not yet complete – **Data Hazard**
    - Example: branch and other instructions that changes the PC makes the fetch of the exact instruction to be delayed or some of the wrong instructions entered to pipeline – **Control Hazard**

# CPU Architecture

- ▶ An important aspect of CPU architecture is the design of the instruction set for the processor.
- ▶ The instruction set chosen for a particular computer determines the way that machine language programs are constructed.
- ▶ As digital hardware became cheaper with the advancement of integrated circuits, computer instructions tended to increase both in number and complexity.
- ▶ In one approach, a computer has large number of instructions and many of which are capable of performing complex tasks.
- ▶ A computer that follows this approach is referred to as a Complex Instruction Set Computer (CISC).

# CISC

- ▶ The instruction set of a CISC system is made efficient by incorporating a large number of powerful instructions.
- ▶ The number of instructions are more and of variable length.
- ▶ More number of addressing modes.
- ▶ Single instruction for each statement.
- ▶ The goal is to reduce the size of the compiled program and to improve the performance.
- ▶ Algorithms can be expressed in more concise form.
- ▶ Because of the limited size of the compiled program, the requirement for main memory is also small.

- 
- ▶ The lesser the number of instructions in a compiled program, the lesser is the time spend by the CPU to fetch instructions.
  - ▶ This decreases the execution time considerably.
  - ▶ However, a highly efficient compiler is required to use the powerful instructions more frequently.
  - ▶ Hence, the system software (compiler) becomes huge in order to generate a small object code.
  - ▶ High level statements are implemented in hardware using complex decoding circuits. Thus, more emphasis is given to the hardware.
  - ▶ Example: Motorola 68000, DEC VAX, IBM 370, Intel Pentium etc.

# Advantages

- ▶ Presence of more varieties of instructions, variable length instructions, and more addressing modes eases the task of the compiler writer.
- ▶ Improved execution efficiency, because complex sequence of operations can be implemented in microcode.
- ▶ Provides support for even more complex and sophisticated high-level language.
- ▶ Programs are shorter and faster.
- ▶ Fewer instructions means fewer instruction bytes to be fetched.
- ▶ In a paging environment, smaller programs occupy fewer pages, reducing number of page faults.

# Disadvantages

- ▶ The control unit design (mainly instruction decoding) becomes complex since the instruction set is large with heavily encoded instructions.
- ▶ There is a lot of hardware circuitry due to complexity of the CPU. This increases the hardware cost of the system as well as power consumption.
- ▶ Due to increased circuits, the propagation delays are more, and the CPU cycle time is large and hence, the effective clock speed is reduced.
- ▶ The heavy hardware prone to frequent failures.
- ▶ Troubleshooting and detecting a fault is a big task due to presence of a large number of circuits.
- ▶ These complex hardware circuits compete for space with the CPU registers leading to reduction in the number of general-purpose registers.
- ▶ Reduction in the number of general-purpose registers leads to more memory-to-memory operations which increases the average clock cycles required to complete an instruction execution.
- ▶ Not suitable for pipeline implementation due to presence of variable length instructions as balancing pipeline stages becomes difficult.

# RISC

- ▶ The second approach follows a different type of architecture.
- ▶ It is observed that usually 80% of instructions are used less frequently only for 20% of the time.
- ▶ The 20% of instructions are used more frequently for 80% of the time.
- ▶ Keeping this point RISC architecture has been developed.
- ▶ Only simple instructions.
- ▶ Small instruction set.
- ▶ Instruction length remains same for all the instructions.
- ▶ Single clock cycle instruction execution.
- ▶ Large number of registers for storing the operands.
- ▶ Emphasis on register-to-register operation.

- ▶ The only references to memory are limited to load and store of an operand.
- ▶ Reduced addressing modes. This makes decoding instructions easy.
- ▶ Almost all instructions use register addressing mode except load and store.
- ▶ Additional addressing modes like implied or immediate addressing mode may be included.
- ▶ Decoding circuits can be integrated as part of the CPU chip. This enables instruction decoding faster.
- ▶ Suitable for pipeline processing. Emphasis on optimizing the instruction pipeline.
- ▶ No operation that combine LOAD/STORE with arithmetic.
- ▶ No more than one memory-addressed operand per instruction.
- ▶ Hardwired rather than microprogrammed control unit.
- ▶ Example: Motorola 88000, Intel i860, Power PC

# Advantages

- ▶ Presence of simple instructions makes it low cycles per instruction.
- ▶ Reduced instructions lead to less hardware circuits.
- ▶ More space available for general purpose registers.
- ▶ Instruction execution is faster.
- ▶ More operands can be stored in registers
- ▶ Less memory references will be generated making instruction execution faster.
- ▶ Simple instruction format simplify the control unit.
- ▶ Suitable for pipeline implementation and easier to balance the pipeline stages.

# Disadvantages

- ▶ Simple instructions make code size larger.
- ▶ More instruction bytes to be fetched.
- ▶ More memory space required to store program.
- ▶ In a paging environment, larger programs occupy more pages, increasing number of page faults.

# RISC Vs. CISC

Parameter	RISC	CISC
Instruction Types	Simple	Complex
No of Instructions	Reduced(30-40)	Extended(100-200)
Duration of an Instruction	One Cycle	More Cycles (4-120)
Instruction Format	Fixed	Variable
Instruction execution	In Parallel(Pipeline)	Sequential
Addressing Modes	Simple	Complex
Instruction Accessing the memory	Two: Load and Store	Almost all from set
Register set	Multiple	Unique
Complexity	In compiler	In CPU