

---

## 1.1 LEARNING OBJECTIVES

---

After going through this unit, you will be able to :

- define combinational circuit
- define sequential circuit
- describe working principle of half-adder and full-adder
- describe the working principle of half-subtractor and full-subtractor
- describe the working principle of multiplexer and demultiplexure
- describe the working principle of encoder and decoder
- describe the working principle of flip-flop
- describe the working principle of register
- describe the working principle of counter

---

## 1.2 INTRODUCTION

---

You have already been acquainted with different combinational and sequential circuits in *Unit-4* of *BCA(F1)01*. Before going ahead, you may read that unit and start learning this unit with ease.

Various combinational and sequential circuits are discussed here to build your understanding on different digital components that are used in digital applications. A sound knowledge on these components will make you feel confident to understand the working principle of many digital devices in general and the computer system in particular.

---

## 1.3 COMBINATIONAL CIRCUITS

---

A circuit is called a combinational circuit when its output is solely determined by its present input.

Inputs can take values either 0 or 1 and outputs are also available as either 0 or 1. Since the output is related to the inputs by a Boolean expression, therefore a truth table is always associated with all combinational circuits. Conversely, a Boolean expression can be obtained for a combinational circuit from the truth table.



## 1.4 HALF-ADDER

Half-adder is a circuit that can add two binary bits. Its outputs are SUM and CARRY. The following truth table shows various combinations of inputs and their corresponding outputs of a half-adder. X and Y denote inputs and C and S denote CARRY and SUM.

X	Y	CARRY(C)	SUM (S)
0	0	0	0
0	1	0	$1(\bar{X}Y)$
1	0	0	$1(X\bar{Y})$
1	1	$1(XY)$	0

**Truth Table for a Half-Adder**

The minterms for SUM and CARRY are shown in the bracket. The Sum-Of-Product (SOP) equation for SUM is :

$$S = \bar{X}Y + X\bar{Y} = X \oplus Y \dots\dots\dots (1)$$

Similarly, the SOP equation for the CARRY is :

$$C = XY \dots\dots\dots (2)$$

Combining the logic circuits for equation (1) & (2) we get the circuit for Half-Adder as :

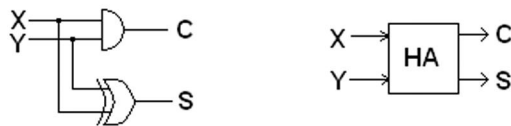


Fig 1.1 : Half-Adder Circuit and Symbol

## 1.5 FULL-ADDER

Full-Adder is a logic circuit to add three binary bits. Its outputs are SUM and CARRY. In the following truth table X, Y, Z are inputs and C and S are CARRY and SUM.

X	Y	Z	CARRY(C)	SUM (S)
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
			1(XYZ)	1(XYZ)

**Truth Table for a Full-Adder**

The minterms are written in the brackets for each 1 output in the truth table. From these the SOP equation for SUM can be written as :

$$\begin{aligned}
 S &= \bar{X}\bar{Y}Z + \bar{X}Y\bar{Z} + X\bar{Y}\bar{Z} + XYZ \\
 &= \bar{X}\bar{Y}Z + Y\bar{Z} + X\bar{Y}\bar{Z} + YZ \\
 &= \bar{X}\bar{S} + XS \dots\dots\dots (3)
 \end{aligned}$$

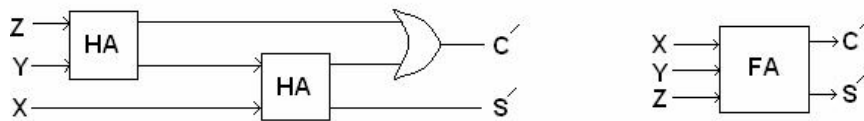
( Exclusive OR and equivalence functions are complement to each other). Here S is SUM of Half-Adder.

Again, SOP equation for Full-Adder CARRY is :

$$\begin{aligned}
 C &= \bar{X}\bar{Y}Z + X\bar{Y}\bar{Z} + XY\bar{Z} + XYZ \\
 &= \bar{X}\bar{Y}Z + XYZ + X\bar{Y}\bar{Z} + XY\bar{Z} \\
 &= \bar{X}\bar{Y}Z + X\bar{Y}\bar{Z} + YZ \\
 &= YZ + XS \\
 &= C + XS\dots\dots\dots (4)
 \end{aligned}$$

Here also C means CARRY of half-adder and S means SUM of half-adder.

Now using two half-adder circuits and one OR gate we can implement equation ( 3) and ( 4) to obtain a full-adder circuit as follows:



**Fig 1.2 : Full-Adder Circuit and its Symbol**

### 1.3.3 HALF-SUBTRACTOR

A half-subtractor subtracts one bit from another bit. It has two outputs viz DIFFERENCE (D) and BORROW (B).

X	Y	BORROW (B)	DIFFERENCE (D)
0	0	0	0
0	1	1	1
1	0	0	1
1	1	0	0

**Truth Table for Half-Subtractor**

The minterms are written within parenthesis for output 1 in each column. The SOP equations are :

$$D = \bar{X}Y + X\bar{Y}$$

$$= X \oplus Y \dots \dots \dots (5)$$

$$B = \bar{X}Y \dots \dots \dots (6)$$



**Fig 1.3 : The half-subtractor circuit and the symbol**

### 1.3.4 FULL-SUBTRACTOR

A full-subtractor circuit can find Difference and Borrow

arising on the subtraction operation involving three binary bits.

X	Y	Z	BORROW (B')	DIFFERENCE (D')
0	0	0	0	0
0	0	1	$\bar{d}YZ$	$\bar{d}YZ$
0	1	0	$\bar{e}YZ$	$\bar{e}YZ$
0	1	1	$dYZ$	0
1	0	0	0	$dYZ$
1	0	1	0	0
1	1	0	0	0
1	1	1	1(XYZ)	1(XYZ)

**Truth Table for Full-Subtractor**

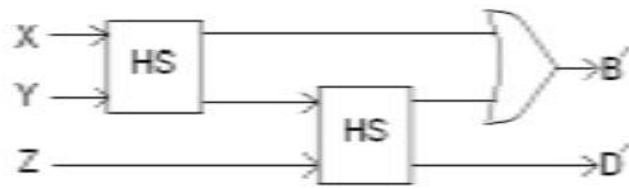
The SOP equation for the DIFFERENCE IS :

$$\begin{aligned}
 D' &= \bar{X}\bar{Y}Z + \bar{X}Y\bar{Z} + X\bar{Y}\bar{Z} + XYZ \\
 &= \bar{X}Y\bar{Z} + X\bar{Y}\bar{Z} + XYZ + \bar{X}\bar{Y}Z \\
 &= \bar{d}YZ + \bar{e}YZ + dYZ + \bar{d}YZ \\
 &= D\bar{Z} + \bar{D}Z \dots\dots\dots (7)
 \end{aligned}$$

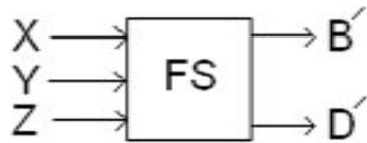
And SOP equation for BORROW is :

$$\begin{aligned}
 B' &= \bar{X}\bar{Y}Z + \bar{X}Y\bar{Z} + \bar{X}YZ + XYZ \\
 &= \bar{X}\bar{Y}Z + XYZ + \bar{X}Y\bar{Z} + \bar{X}YZ \\
 &= \bar{d}YZ + \bar{e}YZ + \bar{d}YZ + \bar{e}YZ \\
 &= D\bar{Z} + \bar{D}Z \dots\dots\dots (7)
 \end{aligned}$$

In equation (7) and (8) , D stands for DIFFERENCE output of half-subtractor. Now, from the equations (7) and (8) we can construct a full-subtractor using two half-subtractor and an OR gate.



**Fig. 1.4 : Full-Subtractor circuit**



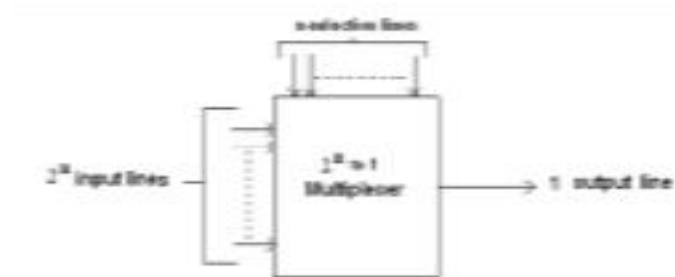
### CHECK YOUR PROGRESS

Choose the correct answer :

1. A half-adder can add
  - (a) Two binary numbers of 4 bits each
  - (b) Two binary bits
  - (c) Add half of a binary
  - (d) None of these number
2. A full-adder is a logic circuit that has two outputs namely:
  - (a) product & Sum
  - (b) sum & borrow
3. A half-subtractor can perform
  - (a) subtraction of two binary bits
  - (b) product of two binary bits
  - (c) complement of half binary bits
  - (d) none of these
4. A full-subtractor has the ability to do
  - (a) subtraction of two binary numbers
  - (b) subtraction of three binary bits
  - (c) product of three binary bits

### 1.3.5 MULTIPLEXER

Multiplexer is a circuit which has many inputs and only one output. Multiplexer can select any one of its many inputs by using selection lines and steer the selected input to the output. Generalized block diagram of a multiplexer is shown in Fig. 1.5.



**Fig. 1.5 : Block Diagram of Multiplexer**

It has  $2^n$  inputs,  $n$  – numbers of selection lines and only one output. A multiplexer is also called a many – to – one data selector.

#### **8 - to - 1 MULTIPLEXER :**

Fig 1.6 shows a 8-to-1 multiplexer, where there are 8 inputs, 3 selection lines and 1 output. The eight inputs are labeled as  $I_0, I_1, I_2, I_3, I_4, I_5, I_6, I_7$  and the selection lines are as A, B, C. Which input is steered to the output depends on the value of ABC. As for example, if

$$ABC = 000$$

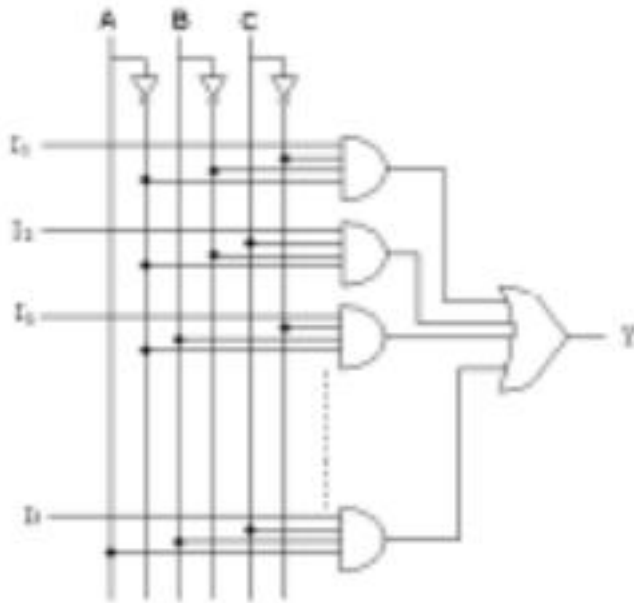
then the upper AND gate is enabled and all other AND gates are disabled. As a result, the  $I_0$  alone is inputsteered to the output. Similarly,

if

$$ABC = 110$$

Then the AND gate connected to the data line  $I_6$  is enabled while all the other AND gates are disabled. Therefore, the input  $I_6$  appears at the output. Hence when  $ABC = 110$ , the output is  $Y = I_6$





**Fig. 1.6 : 8-to-1 Multiplexer**

#### **16 - to -1 MULTIPLEXER :**

Fig 1.7 shows a 16 – to – 1 multiplexer. In this circuit, there are 16 data input lines, 4 selection lines and 1 output. The input lines are denoted by  $I_0, I_1, I_2, I_3, \dots, I_{15}$  and the selection lines are by ABCD. The output is denoted by Y.

Out of 16 input lines, only one is transmitted to the output depending upon the value of ABCD. If

$$ABCD = 0000$$

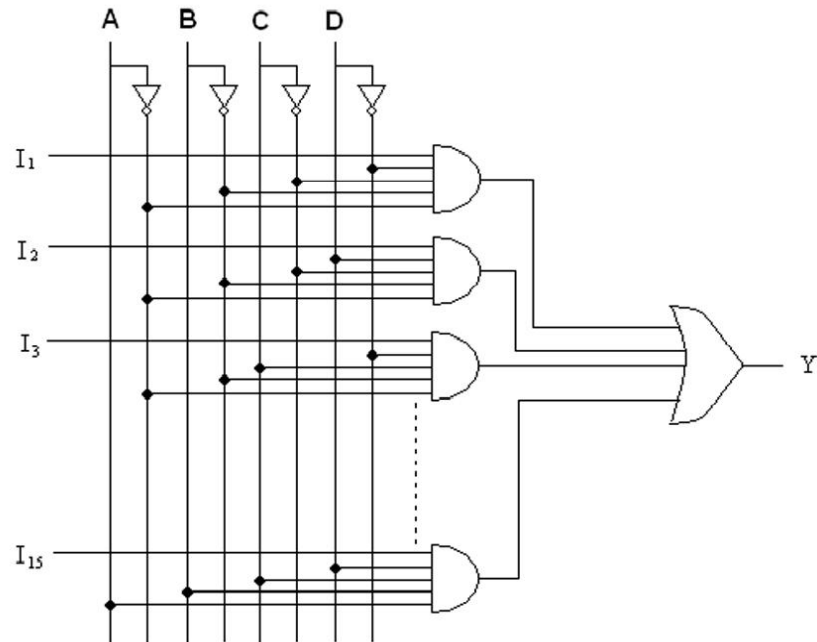
then  $I_0$  is steered to the output since the upper AND gate is enabled alone and all others are disabled. Similarly, if

$$ABCD = 0010$$

then  $I_2$  appears at the output. If

$$ABCD = 1111$$

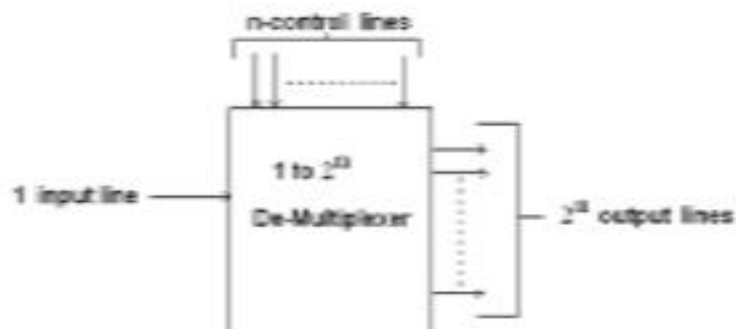
the the last AND gate is one enabled and therefore,  $I_{15}$  appears at the output.



**Fig. 1.7 16-to-1 Multiplexer**

### 1.3.6 DE-MULTIPLEXER

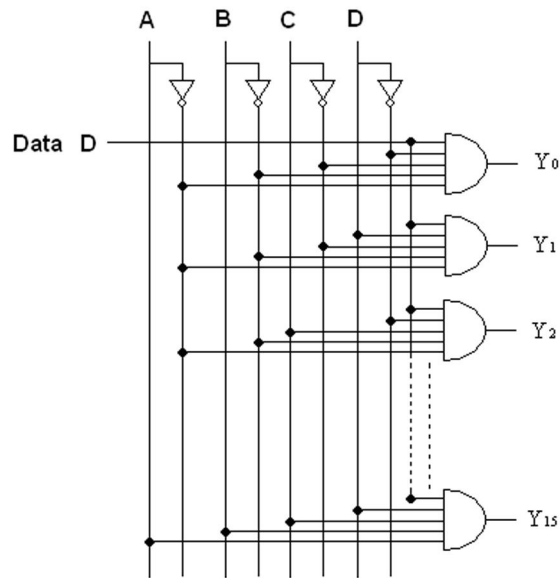
It is opposite to the multiplexer. De-multiplexer has 1 input and many outputs. With the application of appropriate control signal, the common input data can be steered to one of the output lines. Fig 1.8 shows a generalized block diagram of a de-multiplexer.



**Fig. 1.8 : Block diagram of a De-Multiplexer**

To have  $2^n$  output lines, there must be  $n$  control lines in a de-multiplexer.

**1-to-16 DE- MULTIPLEXER** : In Fig 1.9 we have shown a 1-to- 16 de-multiplexer.



**Fig 1.9 : 1-to-16 De-Multiplexer**

Here, the data input line is denoted by  $D$ . This input line is connected to all the AND gates through which output appears. Depending upon the control signal, only one AND gate becomes enabled and the data input  $D$  appears through that AND gate. So, when  $ABCD = 0000$ , the upper AND gate is enabled and data input  $D$  appears at  $Y_0$  as output.

When  $ABCD = 1111$ , the bottom AND gate becomes enabled and  $D$  appears at  $Y_{15}$  as output. For other combination,  $D$  appears at other output terminal.



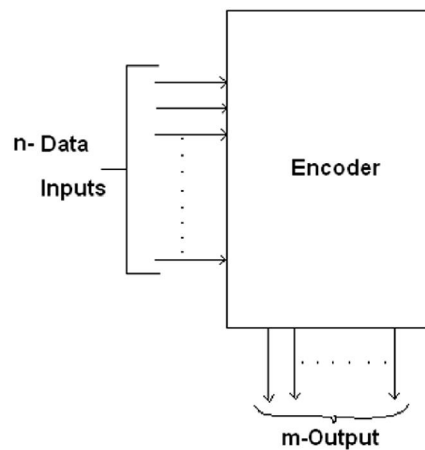
## CHECK YOUR PROGRESS

Give the correct answer :

5. Multiplexer means
  - (a) multiple to many
  - (b) one to many
  - (c) many to one
  - (d) one to one
6. A 16-to-1 multiplexer has
  - (a) 1 selection lines
  - (b) 2 selection lines
  - (c) 3 selection lines
  - (d) 4 selection lines
7. De-multiplexer means
  - (a) deduct multiple bits
  - (b) one-to-many
  - (c) multiple-to-multiple
  - (d) one-to-one

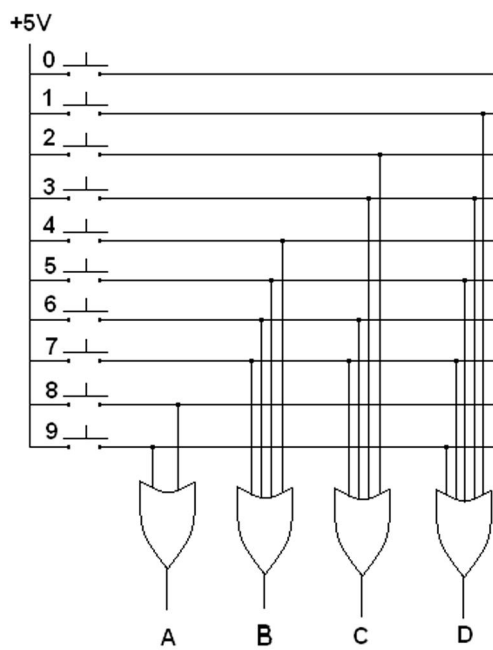
### 1.3.7 ENCODER

An encoder converts a digital signal into a coded signal. A generalized view of an encoder is shown in Fig 1.10.



**Fig. 1.10 : Block diagram of an Encoder**

In the figure, we can see that there are  $n$  input lines and  $m$  numbers of output lines where  $n \leq 2^m$ . Out of  $n$  inputs, only one input line is active at a time. Encoder generates a coded output which is unique for each of the active input.



**Fig. 1.11 : Decimal-to-BCD Encoder**

A decimal to BCD encoder is shown in Fig 1.11. This circuit generates BCD output when any one of the push button switches is pressed. As for example, if button 6 is pressed, the B and C OR gates have high inputs and the corresponding output becomes

$$ABCD = 0110$$

If button 8 is pressed, the OR gate A receives a high input and therefore the output becomes

$$ABCD = 1000$$

### 1.3.8 DECODER

A decoder is a digital circuit which has  $n$ -input lines and  $2^n$  output lines. A decoder and a de-multiplexer has similarity. In a de-multiplexer, there is a single input line connected to every output 'AND' gate whereas in a decoder that input line is absent.

Let us draw a 3 – to – 8 decoder as shown in Fig 1.12. Its truth table is shown below :

Inputs			Outputs							
A	B	C	$Y_0$	$Y_1$	$Y_2$	$Y_3$	$Y_4$	$Y_5$	$Y_6$	$Y_7$
0	0	0	1	0	0	0	0	0	0	0
0	0	1	0	1	0	0	0	0	0	0
0	1	0	0	0	1	0	0	0	0	0
0	1	1	0	0	0	1	0	0	0	0
1	0	0	0	0	0	0	1	0	0	0
1	0	1	0	0	0	0	0	1	0	0
1	1	0	0	0	0	0	0	0	1	0
1	1	1	0	0	0	0	0	0	0	1

Truth table for a 3-to-8 decoder

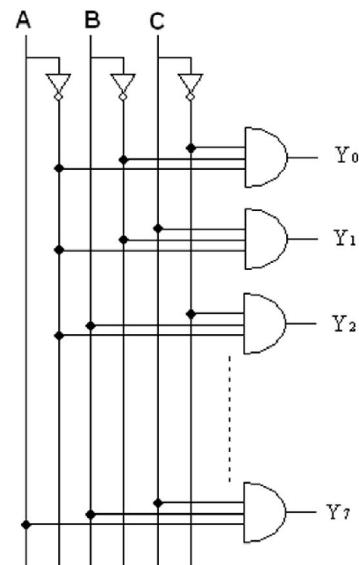


Fig. 1.12 : 3-to-8 decoder

### 1.3.9 MAGNITUDE COMPARATOR

A magnitude comparator circuit can compare two binary numbers to determine which one is greater than the other or their equality. Such a magnitude comparator has three output lines for  $A > B$ ,  $A = B$ ,  $A < B$  where  $A$  &  $B$  are two  $n$ -bits binary numbers. Every bit of one number is compared with the corresponding bit of the other number by ExOR gate.

A 4-bit magnitude comparator, SN 7485 is available in chip form the block diagram of which is shown in Fig 1.13. It

compares two 4-bit binary numbers  $A_3 A_2 A_1 A_0$  and  $B_3 B_2 B_1 B_0$ .

Three output terminals are available for  $A < B$ ,  $A = B$  and  $A > B$

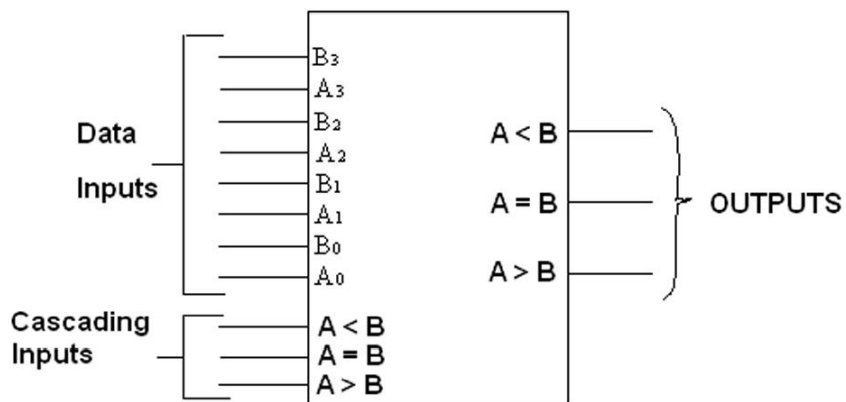


Fig. 1.13 : Two 4-bit words magnitude comparator SN 7485

Comparing Inputs				Cascading Inputs			Outputs		
$A_3B_3$	$A_2B_2$	$A_1B_1$	$A_0B_0$	$A > B$	$A < B$	$A = B$	$A > B$	$A < B$	$A = B$
$A_3 > B_3$	X	X	X	X	X	X	H	L	L
$A_3 < B_3$	X	X	X	X	X	X	L	H	L
$A_3 = B_3$	$A_2 > B_2$	X	X	X	X	X	H	L	L
$A_3 = B_3$	$A_2 < B_2$	X	X	X	X	X	L	H	L
$A_3 = B_3$	$A_2 = B_2$	$A_1 > B_1$	X	X	X	X	H	L	L
$A_3 = B_3$	$A_2 = B_2$	$A_1 < B_1$	X	X	X	X	L	H	L
$A_3 = B_3$	$A_2 = B_2$	$A_1 = B_1$	$A_0 > B_0$	X	X	X	H	L	L
$A_3 = B_3$	$A_2 = B_2$	$A_1 = B_1$	$A_0 < B_0$	X	X	X	L	H	L
$A_3 = B_3$	$A_2 = B_2$	$A_1 = B_1$	$A_0 = B_0$	H	L	L	H	L	L
$A_3 = B_3$	$A_2 = B_2$	$A_1 = B_1$	$A_0 = B_0$	L	H	L	L	H	L
$A_3 = B_3$	$A_2 = B_2$	$A_1 = B_1$	$A_0 = B_0$	L	L	H	L	L	H

To compare any number having more than 4-bits, two or more such chip can be cascaded. The  $A > B$ ,  $A < B$  and  $A = B$  outputs of a stage that handles less significant bits are connected to the corresponding cascading inputs of the next stage that handles the more significant bits.



### CHECK YOUR PROGRESS

Give the correct answer :

8. An encoder
  - (a) converts a digital input to another form of digital output.
  - (b) converts analog input to digital output
  - (c) selects one out of many inputs.
  - (d) none of these.



9. Decoder has  $n$  inputs line and
- (a)  $n$  output lines.
  - (b)  $2^n$  output lines.
  - (c)  $n^2$  output lines.
  - (d) no output lines.
10. Magnitude comparator
- (a) compares two multi bit binary number.
  - (b) magnify any digital signal.
  - (c) compress binary numbers.
  - (d) check error in a binary number.

---

## 1.4 SEQUENTIAL CIRCUITS

---

If the output of a circuit depends on its present inputs and immediate past output, then the circuit is called sequential circuit. To build a sequential circuit, we need memory circuits and combinational circuits. Flip-Flop is used as memory circuit the application of which we would see in counter, register etc.

---

## 1.5 FLIP- FLOPS

---

A digital circuit that can produce two states of output, either high or low, is called a multivibrator. Three types of multivibrators are- monostable, bi-stable and a stable.

A Flip-Flop is a bi-stable multivibrator and therefore it has two stable states of output-either high or low. Depending on its outputs and previous inputs, its new output is either high (or 1) or low (or 0). Once the output is fixed, the inputs can be removed and then also the already fixed output will be retained by the flip-flop. Hence a flip-flop can be used as basic circuit of memory for storing one bit of data. To store multiple bits we can use multiple numbers of flip-flop. Flip-flops are also used to build counter, register etc.

There are many types of flip-flops. These are:

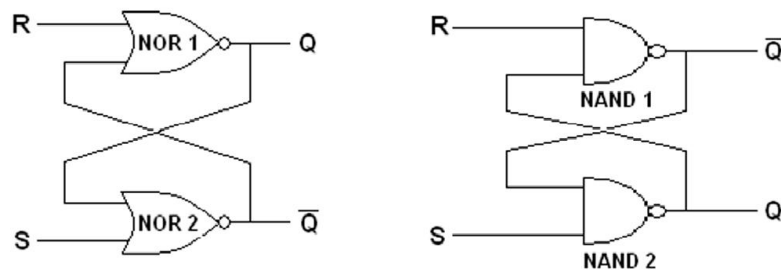
- RS Flip-flop
- D Flip-Flop
- JK Flip-Flop
- MS Flip-Flop

---

### 1.5.1 RS FLIP-FLOP

---

RS flip-flop is also called Set-Reset flip-flop. An RS flip-flop can be built using a pair of NOR gates or NAND gates. Here we have shown two RS flip-flop circuits using NOR and NAND gate. To study its working principle, we can use any one of them. In the following section, let us consider the RS flip-flop using NOR gate.



**Fig. 1.14 : Basic circuit of RS flip-flop using NOR and NAND gate.**

RS flip-flop has two inputs: Set (S) and Reset(R). It has two outputs, Q and  $\bar{Q}$ . It should be noted  $\bar{Q}$  is always the complement of Q. Various combinations of inputs and their corresponding outputs are listed in the truth table below :

R	S	Q	Action
0	0	Last value	No change
0	1	1	Set
1	0	0	Reset
1	1	?	Forbidden

**Truth table of RS flip-flop**

The first input condition in the table is R=0, S=0. Since a

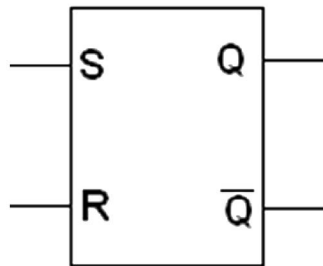
0 input has no effect on its output, the Flip-Flop retains its previous state. Hence Q remains unchanged.

The second input condition  $R=0, S=1$  forces the output of NOR gate2 low. This low output will reach NOR gate1 and when both inputs of NOR gate1 is low, its output Q will be high. Thus, a

1 at the S input will SET the flip-flop and Q will be equal to 1.

The third input condition  $R=1, S=0$  will force the output of NOR gate1 to low. This low will reach NOR gate2 and forces its output to high. Hence, when  $R=1, S=0$ , then  $Q=0, \bar{Q}=1$ . Thus, the flip-flop is RESET.

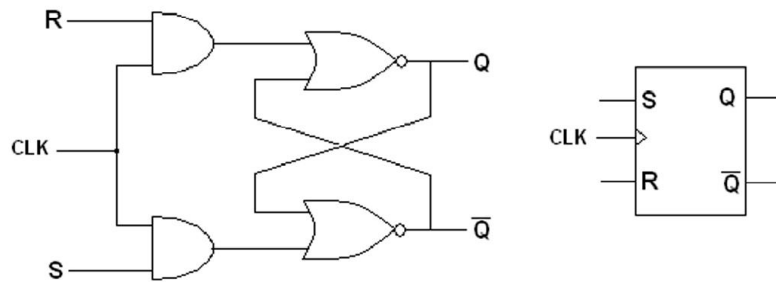
The last input condition in the table  $R=1, S=1$  is forbidden since it forces both the NOR gates to the low state, means both  $Q=0, \bar{Q}=0$  at the same time, which violates the basic definition of flip-flop that requires Q to be the complement of  $\bar{Q}$ . Hence, this input condition is forbidden and its output is unpredictable.



**Fig. 1.15 : Symbol of RS Flip-Flop**

### **CLOCK INPUT:**

For synchronization of operation of multiple flip-flop, an additional signal is added to all types of flip-flop which is called clock signal, generally abbreviated as CLK. Addition of CLK signal ensures that, whatever may be the input to the flip-flop, it effects the output only when CLK signal is given. Fig. 1.16 shows a clocked RS flip-flop.

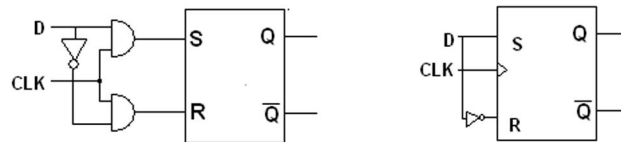


**Fig. 1.16 : Clocked RS flip-flop**

**Symbol of Clock RS flip-flop**

### 1.5.2 D FLIP-FLOP

D flip-flop is a modification of RS flip-flop. In RS flip-flop when both the inputs are high i.e.,  $R=1$ ,  $S=1$ , the output becomes unpredictable and this input combination is termed as forbidden. To avoid this situation, the RS flip-flop is modified so that both the inputs can not be same at a time. The modified flip-flop is called D flip-flop. Fig 1.17 shows a clocked D flip-flop.



**Fig. 1.17 : (a) Clocked D flip-flop Symbol of D flip-flop**

In D flip-flop both inputs of RS flip-flop are combined together to make it one by a NOT gate so that inputs can not be same at a time. Hence in D flip-flop there is only one input. The truth table is:

CLK	D	Q
0	X	Last state
0	0	0
1	1	1

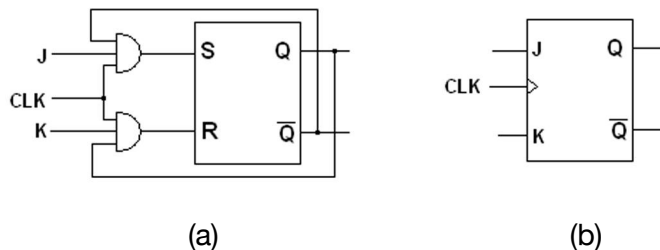
In a clocked D flip-flop the value of D cannot reach the output Q when the clock pulse is low. During a low clock,

both AND

gates are disabled. Therefore, D can change value without affecting the value of Q. On the other hand, when the clock is high, both AND gates are enabled. In this situation, Q is forced to be equal to the value of D. In another way we can say that in the D flip-flop above, Q follows the value of D while the clock is high. This kind of D flip-flop is often called a D latch.

### 1.5.3 JK FLIP-FLOP

In RS flip-flop, the input  $R=S=1$  is called forbidden as it causes an unpredictable output. In JK flip-flop this condition is used by changing the RS flip-flop in some way. In JK flip-flop both input can be high simultaneously and the corresponding toggle output makes the JK flip-flop a good choice to build counter- a circuit that counts the number of +ve or -ve clock edges. Fig 1.18 shows one way to build a JK flip-flop.



**Fig. 1.18 : (a) JK Flip-Flop (b) Symbol of JK Flip-Flop**

CLK	J	K	Q
X	0	0	Last state
↑	0	1	0
↑	1	0	1
↑	1	1	Toggle

**Truth table for JK flip-flop**

The inputs J and K are called control inputs because their combinations decide what the output of JK flip-flop will be when a +ve clock pulse arrives. When J and K are both low,

both the

AND gates are disabled. Therefore, the CLK pulse has no effect. The first input combination of the truth table shows this and under this case the output Q retains its last state.

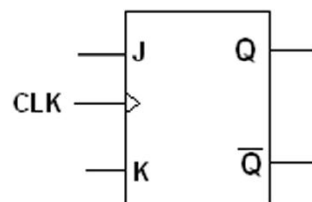
When J is low, K is high, the upper AND gate is disabled while the lower AND gate is enabled. Hence the flip-flop cannot be set; instead it is reset, i.e.  $Q=0$ . This is shown by the second entry in the truth table.

When J is high, K is low the upper AND gate is enabled while the lower one is disabled. So the flip-flop is set there by making  $Q = 1$ .

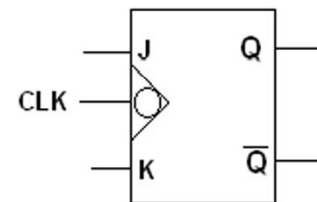
When J and K are both high, then the flip-flop is set or reset depending on the previous value of Q. If Q is high previously, the lower AND gate sends a RESET trigger to the flip-flop on the next clock pulse. Then Q becomes equal to 0. On the other hand, if Q is low previously, the upper AND gate sends a SET trigger on the flip-flop making  $Q=1$ .

So, when  $J = K = 1$ , Q changes its value from 0 to 1 or 1 to 0 on the positive clock pulse. This changing of Q to  $\bar{Q}$  is called toggle. Toggle means to change to the opposite state.

Any flip-flop may be driven by +ve as well as -ve clock. As such JK flip-flop can also be driven by positive clock as well as negative clock. Fig 1.19 shows symbol of positive clocked and negative clocked JK flip-flop.



**Fig. 1.19 : Positive clocked JK Flip-Flop**

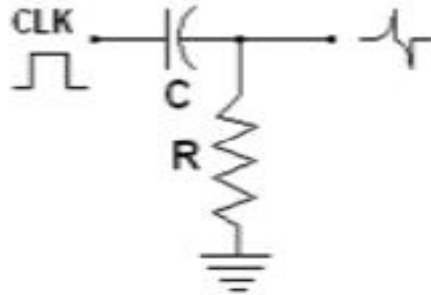


**Negative clocked JK Flip-Flop**



### RC Differentiator Circuit:

The clock pulse applied to a flip-flop is a square wave signal. Clock pulse is used to achieve synchronization of flip-flop operation. To make the synchronization more precise, the square wave pulse is further modified to make it a narrow spike by using a RC differentiator circuit as shown in fig 1.20



**Fig. 1.20 : RC Differentiator Circuit**

The upper tip of the differentiated pulse is called positive edge and the lower tip is called negative edge. When a flip-flop is triggered by this type of narrow spike, it is called edge triggered flip-flop. If the flip-flop is driven by +ve edge, it is called +ve edge triggered flip-flop. If it is driven by negative edge, it is called negative edge triggered flip-flop.

### Racing

In a flip-flop if the output toggles more than once during a clock pulse then it is called **racing**. All flip-flop has a propagation delay which means the output changes its state after a certain time period from applying the input and the clock pulse. So, when a flip-flop is edge triggered, then due to propagation delay the output cannot affect the input again, because by that time the edge of clock pulse has already passed away. If the propagation

delay of a flip-flop is 20 ns and the width of the spike is less than 20 ns, then the returning  $\overline{Q}$  and  $Q$  arrive too late to cause

false triggering.

### 1.5.4 MS FLIP-FLOP

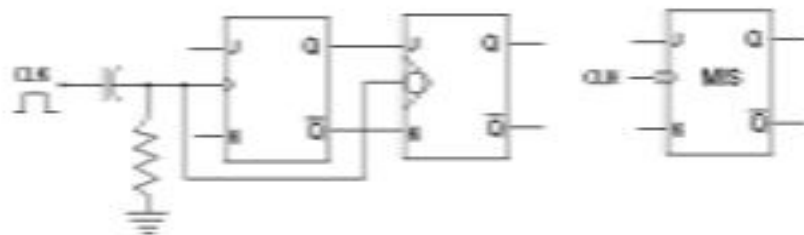
MS flip-flop is another way to avoid racing. Fig 1.21 shows one way to build MS flip-flop using two JK flip-flop, one of which is positive edge triggered and the other is negative edge triggered. The first JK flip-flop is master and the latter is slave. The master responds to its J and K inputs at the positive edge.  $J=1$ ,  $K=0$ , the master sets on the positive clock edge. The high Q output of the master drives the J input of the slave. So, at the negative clock edge, the slave also sets, copying the action of the master.

When  $J=0$ ,  $K=1$ , master resets at +ve clock edge and the slave resets of the -ve clock edge.

When  $J = K = 1$  master toggles at +ve clock edge, and the slave toggles at the -ve clock edge.

Hence, whatever master does, the slave copies it.

MS flip-flop is a very popular flip-flop in industry due to its inherent resistance to racing. Hence, to build counter it is extensively used.



**Fig.1.21:**

**(a) Edge triggered JK MS  
flip-flop**

**(b) Symbol of JK MS  
flip-flop**



## CHECK YOUR PROGRESS 4

11. A flip-flop is basically a
  - (a) mono-stable multi-vibrator
  - (b) a stable multi-vibrator
  - (c) bi-stable multi-vibrator
  - (d) none of these.
12. JK flip-flop has the specialty in
  - (a) fast response time
  - (b) toggle property
  - (c) spike shaped clock input
  - (d) preset input
13. In MS flip-flop the master changes state
  - (a) after the slave
  - (b) with the slave at the same time
  - (c) before the slave
  - (d) never

---

## 1.6 COUNTER

A counter is one of the most useful sequential circuits in a digital system. A counter driven by a clock can be used to count the number of clock cycles. Since the clock pulse has a definite time period, the counter can be used to measure time, the time period or frequency.

There are basically two types of counter: **Synchronous counter** and **asynchronous counter**.

Counters are constructed by using flip-flops and other logic gates. If the flip-flops are connected serially then the output of one flip-flop is applied as input to the next flip-flop. Therefore, this type of counter has a cumulative settings time due to propagation delay. Counters of this type are called serial or asynchronous counter. These counters have speed limitation.

Speed can be increased by using parallel or synchronous counter.

Here, flip-flops are triggered by a clock at a time and thus setting time is equal to the propagation delay of a single flip-flop. But this type of synchronous counters require more hardware and hence they are costly.

Combination of serial or parallel counter is also done to get an optimum solution of speed and hardware/cost. If each clock pulse advances the contents of the counter by one, it is called **up counter**. If the content of the counter goes down at each clock pulse, it is called **down counter**.

Before operation, some time it is required to reset all the flip-flops to zero. It is called "Clear". Some time, it is required to set the flip-flops. It is called **preset**. To do these, two extra inputs are there in every flip-flop called CLR and PR.

---

### 1.6.1 Asynchronous Counter

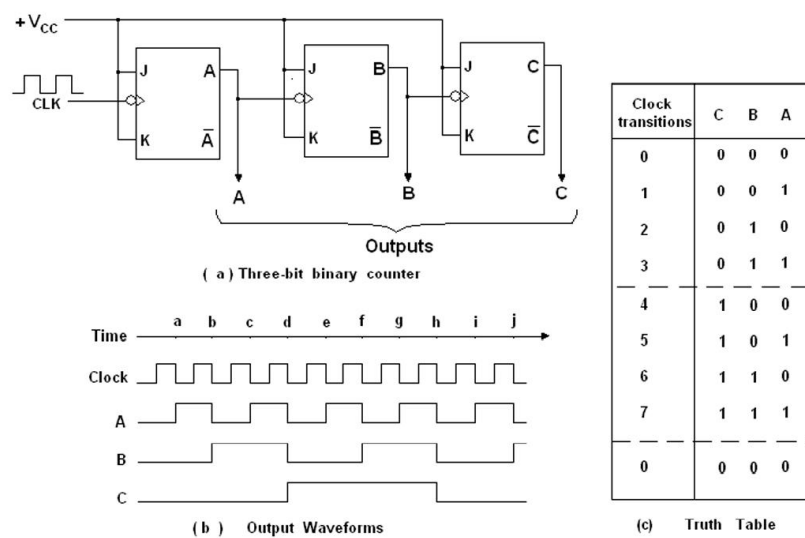
---

When the output of a flip-flop is used as the clock input for the next flip-flop it is called asynchronous counter.

Asynchronous counters are also called ripple counter because flip-flop transitions ripple through from one flip-flop to the next in a sequence until all flip-flops reach a new state.

A binary ripple counter can be constructed by using clocked JK flip-flop. Fig 1.22(a) shows three MS JK flip-flops connected in series. The clock drives flip-flop A. The output of A drives B and the output of B drives C. J and K inputs of all the flip-flops are connected to positive to make them equal to 1. Under this condition each flip-flop will change state (toggle) with a negative transition at its clock point.

In the counter shown in Fig 1.22 (a), the flip-flop A changes its state at the negative edges of the clock pulses. Its output is applied to the B flip-flop as its clock input.



**Fig. 1.22 : Three Bit Binary Counter**

The output of B flip-flop toggles at the negative edges of the output of A flip-flop. Similarly, the output of B flip-flop is used as clock input of the C flip-flop and therefore C toggles at the negative edges of the output of B flip-flop. We can see that triggering pulses move through the flip-flops like a ripple in water.

The wave form of the ripple counter is shown in fig 1.22(b). It shows the action of the counter as the clock runs. To understand the wave form let us assume that the counter is cleared before the operation. The A output is assumed the least significant bit (LSB) and C is the most-significant-bit (MSB). Hence, at the very beginning the contents of the counter is CBA=000.

Flip-flop A changes its state to 1 after the negative pulse transition. Thus, at point a on the time line, A goes high. At point b it goes low, at C it goes back to high and so on.

Now, output of A acts as clock input of B. So, each time the output of A goes low, flip-flop B will toggle. Thus, at point b on the time line, B goes high at point b and it goes low at point d, and toggles back high again at point f and so on.

Since B acts as the clock input for C, each time the output of B goes low, the C flip-flop toggles. Thus C goes high at point

don the time line, it goes back to low again at point h.



We can see, at the output, wave form of A has half the frequency than the clock input wave. B has half the frequency than that of A and C has half the frequency than that of B.

We can further see that since the counter has 3 flip-flops cascaded together, it progresses through 000--- 001---010--- 011---100---101---110---111 as its CBA output. After CBA= 111, it

starts the cycle again from CBA=000. One cycle from 000---111 takes 8 clock pulses, as it is evident from the wave form as well as from the truth table.

---

### 1.6.2 Synchronous Counter

---

An asynchronous counter or ripple counter has limitation in its operating frequency. Each flip-flop has a delay time which is additive in asynchronous counter.

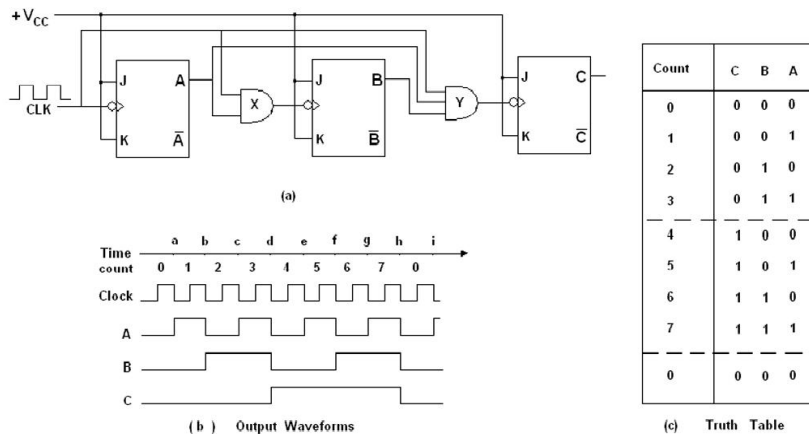
In synchronous counter the delay of asynchronous counter is overcome by the use of simultaneous applications of clock pulse to all the flip-flops. Hence, in synchronous counter, the common clock pulse triggers all the flip-flops simultaneously and therefore the individual delay of flip-flop does not add together. This feature increases the speed of synchronous counter. The clock pulse applied can be counted by the output of the counter.

To build a synchronous counter, flip-flops and some additional logic gates are required. Fig 1.23 shows a three stage synchronous or parallel binary counter along with its output wave forms and truth table. Here the J and K inputs of each flip-flop is kept high and therefore the flip-flops toggle at the negative clock transition at its clock input. From figure we can see that the output of A is ANDed with CLK to drive the 2nd flip-flop and the outputs of A, B are ANDed with CLK to drive the third flip-flop. This logic configuration is often referred to as “steering logic” since the clock pulses are steered to each individual flip-flop.

In the figure, the clock pulse is directly applied to the first flip-flop. Its J and K are both high, so the first flip-flop toggles

state at the negative transition of the input clock pulses. This can be seen at points a,b,c,d,e,f,g,h,i on the time line.

The AND gate X is enabled when A is high, and it allows a clock pulse to reach the 2nd flip-flop. So the 2nd flip-flop toggles with every other negative clock transition at points b, d, f and h on the time line.



**Fig. 1.23 : Parallel Binary Counter**

The AND gate Y is enabled only when both A and B are high and it transmits the clock pulses to the clock input of the 3rd flip-flop. The 3rd flip-flop toggles state with every fourth negative clock transition at d and h on the time line.

The wave form and the truth table shows that the synchronous counter progresses upward in a natural binary sequence from 000 to 111. The total count from 000 to 111 is 8 and hence this



### CHECK YOUR PROGRESS

#### 14. State *True or False*

- Counters are non sequential digital
- Asynchronous counters are fast in operation than synchronous counters.
- The natural progression of a counter is called MODE.
- Counters can be used to build digital clock.

counter can also be called MOD-8 counter, in count up mode.

---

## 1.7 REGISTER

---

A number of flip-flops connected to store binary number is called a register. The number to be stored is entered or shifted into the register and also taken out or shifted out as per necessity. Hence, registers are also known as shift register.

Registers are used to store data temporarily. Registers can be used to perform some important arithmetic operations like complementation, multiplication, division etc. It can be connected to form counters, to convert serial data to parallel and parallel to serial data.

Types of registers: According to shifting of binary number also called shift registers, different types of registers are:

- Serial In—Serial Out (SISO)
- Serial In –Parallel Out (SIPO)
- Parallel In –Serial Out (PISO)
- Parallel In –Parallel Out (PIPO)

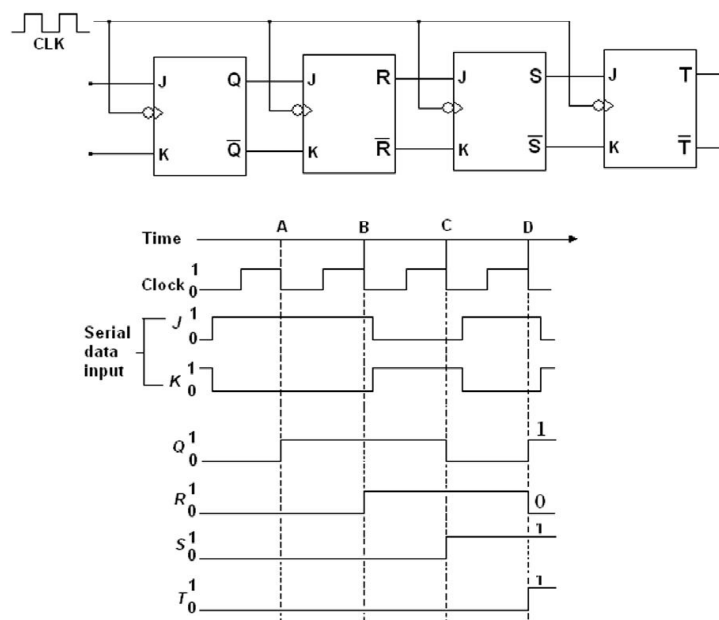
---

### 1.7.1 Serial In - Serial Out (SIPO)

---

Fig 1.24 shows a typical 4 bit SISO register using flip-flops. Here the content of the register is named as QRST. Let us consider that all flip-flops are initially reset. Hence, at the beginning QRST=0000. Let us consider a binary number 1011 which we want to store in the SISO register.

At time A: A 1 is applied at the D input at the first flip-flop. At the negative edge of the CLK pulse, this 1 is shifted into Q. The O of Q is shifted into R, O of R is shifted into S and O of S is shifted into T. The output of flip-flop just after time A are QRST=1000.



**Fig. 1.24 : 4 Bit Serial In-Serial Out Shift Register**

At time B: Another 1 is applied in the data input of the first flip-flop. So at the negative CLK edge, this 1 is shifted to Q. The 1 of Q is shifted in R, 0 of R shifted in S, 0 of S is shifted into T.

So, at the end of time B the output of all the flip-flops are QRST=1100.

At time C: A 0 (zero) is applied in the D input of the 1st flip-flop. At the negative CLK edge, this 0 shifts to Q. The 1 of Q shifts into R, 1 of R shifts into S, 0 of R shifts into S, 0 of S shifts into T. Hence the output becomes QRST=0110.

At time D: 1 is applied to D input of the first flip-flop. So this 1 shifts into Q at the negative transmission of CLK. The previous 0 of Q shifts into R, the 1 of R shifts into S, the 1 of S shifts into T. Hence at the end of time D, the registers contain QRST=1011.

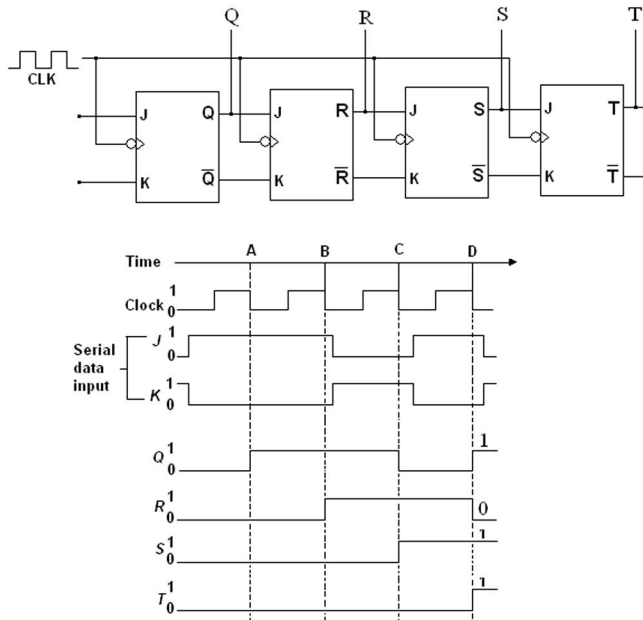
In the above steps, using 4 CLK pulses, we have shifted a 4-bit binary number 1011 in the register in a serial fashion.

To take out this binary number serially, we need another 4 CLK

pulses and 4 O inputs into D pin of the first flip-flop. The binary

number leave the register serially through the T pin of the last flip-flop.

### 1.7.2 Serial In - Parallel Out (SIPO)



**Fig. 1.25 : Bit Serial In - Parallel Out Shift Register**

In this type of shifts register, data is entered serially into the register and once data entry completed it can be taken out parallelly. To take the data parallelly, it is simply required to have the output of each flip-flop to an output pin. All other constructional features are same as Serial In—Serial Out (SISO) register.

The shifting of data into SIPO is same as SISO registers. In the SIPO of Fig 1.25 , a binary number, say 1011 would be shifted just like the manner as described in the previous section. It would take 4 CLK pulses to complete the shifting. As soon as shifting is completed, the stored binary number becomes available in the output pins QRST. SIPO register is useful to convert serial data into parallel data.

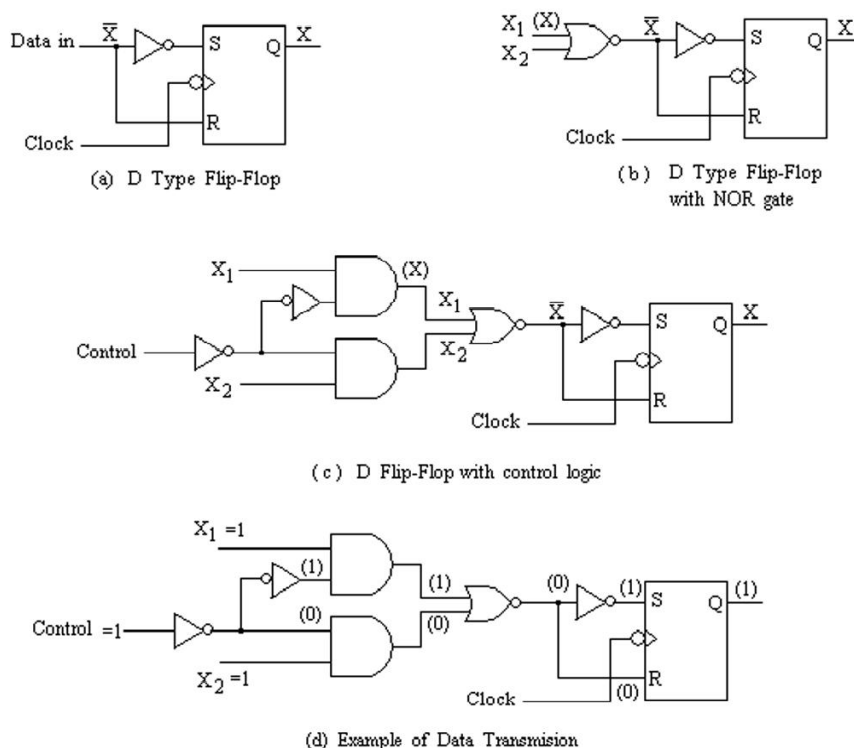


### 1.7.3 Parallel In-Serial Out (PISO)

PISO registers take data parallelly and shifts data serially. Commercially available TTL IC for PISO is 54/74166. To understand the functional block diagram of 54/74166 we should first understand the following Fig 1.26(a) is a clocked RS flip-flop, which is converted to a D flip-flop by a NOT gate. The

output of the flip-flop is 1 if Data IN ( $\bar{X}$ ) is 0. Next add a NOR gate as in Fig 1.26(b). Here, if  $X_2$  is at ground level,  $X_1$  will be inverted by the NOR gate. As for example, if  $X_1 = 1$ , then output

of the NOR gate will be  $X = 0$ , thereby a 1 will be clocked into the flip-flop. This NOR gate allows entering data from two sources, either from  $X_1$  or  $X_2$ . To shift  $X_1$  into the flip-flop,  $X_2$  is kept at ground level and to shift  $X_2$  into the flip-flop,  $X_1$  is kept at ground level.

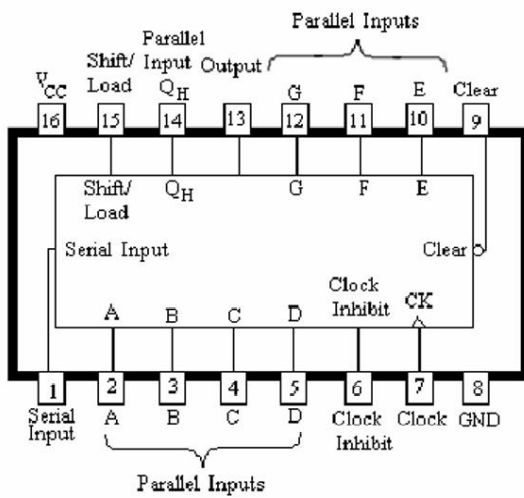


**Fig. 1.26 : Building Block of Parallel In-Serial Out Register**

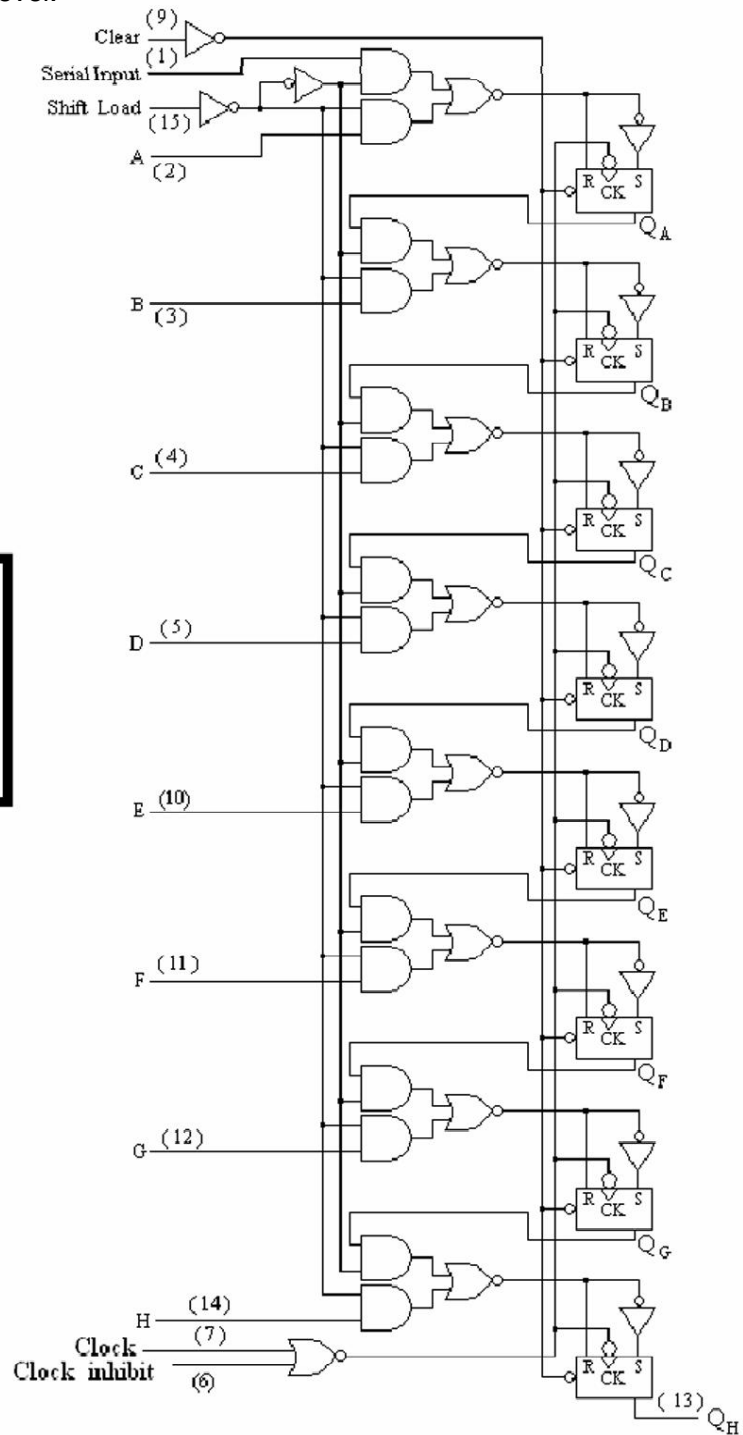
Now in Fig 1.26 (c) two AND gates and two NOT gates are

added. These will allow the selection of data  $X_1$  or data  $X_2$ . If the control line is high, the upper AND gate is enabled and the lower AND gate is disabled. Thus, the data  $X_1$  will enter at the upper leg of the NOR gate and at the same time the lower leg of the

NOR gate is kept at ground. Opposite to this, if the control is low, the upper AND gate is disabled and the lower AND gate is enabled. So  $X_2$  will appear at the lower leg of the NOR gate and during this time the upper leg of the NOR gate is kept at ground level.



(a) Pin Out Diagram of Serial In-Parallel Out Register



(b) Logic Diagram of Serial In-Parallel Out Register

Fig 1.27 Circuit of 54/74166

If we study Fig 1.27 of PISO we see that circuit of Fig 1.26( c ) is repeated 8 times to form the 54/74/66 shift register. These 8 circuits are connected in such a style that it allows two operations: (1) The parallel data entry and (2) shifting of data serially through the flip-flop  $Q_A$  from  $Q_B$  toward

If Fig 1.27 the  $X_2$  input of Fig 1.26( c ) is taken out from each flip-flop to form 8 inputs named as ABCDEFGH to enter 8 bit data parallelly to the register. The control is named here as SHIFT/LOAD which is kept low to load 8 bit data into the flip-flops with a single clock pulse parallelly. If the SHIFT/LOAD is kept high it will enable the upper AND gate for each flip-flop. If any input is given to this upper AND gate then a clock pulse will shift a data bit from one flip-flop to the next flip-flop. That means data will be shifted serially.

---

#### 1.7.4 Parallel In - Parallel Out Register (PIPO)

---

The register of Fig 1.27 can be converted to PIPO register simply by adding an output line from each flip-flop.

The 54/74198 is an 8 bits such PIPO and 54/7459A is a 4 bit PIPO register. Here the basic circuit is same as Fig 1.26(C). The parallel data outputs are simply taken out from the Q sides of each flip-flop. In Fig 1.28 the internal structure of 54/ 7459A is shown.

When the MODE CONTROL line is high, the data bits ABCD will be loaded into the register parallelly at the negative clock pulse. At the same time the output is available  $Q_A Q_B Q_C Q_D$ . When the MODE CONTROL is low, then the left AND gate of the NOR gate is enabled. Under this situation, data can be entered to the register serially through SERIAL INPUT. In each negative transition, a data bit shifted serially from  $Q_A$  to  $Q_B$ , from  $Q_B$  to  $Q_C$  and so on. This operation is called right-shift operation.

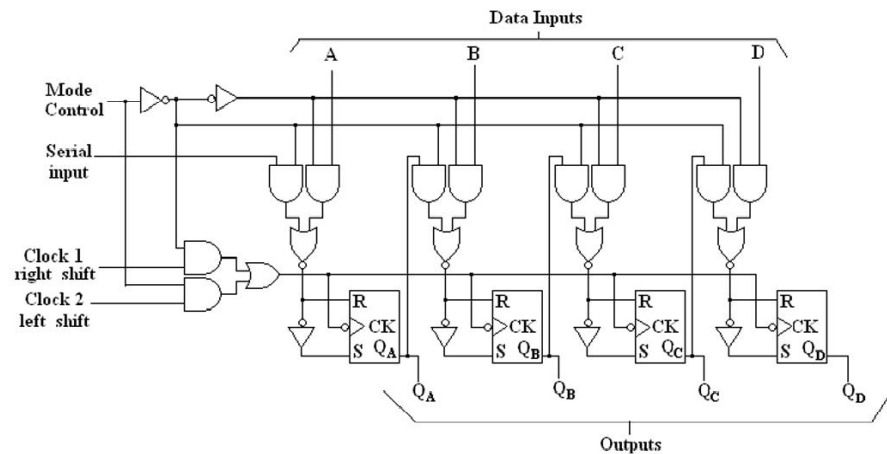
With a little modification of the connection, the same circuit can be used for shift-left operation. To operate in shift-left mode, the

input data is to be entered through D input pin. It is also necessary

to connect  $Q_D$  to C,  $Q_C$  to B,  $Q_B$  to A as shown in Fig 1.28. MODE CONTROL line is high to enter data through the D input pin and each stored data bits of flip-flops will be shifted to left flip-flop on each negative clock transition. This is serial data and left shift operation.

To clock inputs—clock1 and clock2 is used here to perform shift right and shift left operation..

Hence 54/7495A can be used as Parallel In – Parallel Out shift register as well as shift right and shift left register.



**Fig. 1.28 : Parallel In - Parallel Out Shift Register**



### CHECK YOUR PROGRESS 6

15. Shift registers are
  - (a) basically a synchronous circuit
  - (b) an asynchronous circuit
  - (c) permanent memory
  - (d) none of these
16. In SIPO
  - (a) data enters parallelly and leaves serially
  - (b) data enters serially and leaves serially
  - (c) data enters serially and leaves parallelly
  - (d) data enters parallelly and leaves parallelly

## 1.8 LET US SUM UP

---

- Digital circuits are of two categories - combinational and sequential
- A combinational circuit is some combinations of logic gates as per specific relationship between inputs and outputs.
- Adder and subtractor circuits can perform binary addition and subtraction.
- A multiplexer is a combinational circuit which selects one of many inputs.
- Demultiplexer is opposite to a multiplexer.
- An encoder generates a binary code for input variables.
- A decoder decodes an information receives from n input lines and transmits the decoded information to maximum outputs.
- A sequential circuit's output depends on past output and present inputs.
- A flip-flop is basically a single cell of memory which can store either 1 or 0.
- Sequential circuits use flip-flop as their building block.
- There are many types of flip-flop viz RS, D, JK, MS flip-flop.
- A counter is a sequential circuit that can count square waves give as clock input. There are two types of counters- asynchronous and synchronous counter.
- Shift registers are also sequential circuit which are used to store binary bits. They are of four different types - Serial In- Serial Out, Serial In- Parallel Out, Parallel In- Parallel Out and Parallel In-Serial Out register.



## 1.9 ANSWERS TO CHECK YOUR PROGRESS

---

1. (b)    2. (c)    3. (a)    4. (b)
5. (c)    6. (d)    7. (b)
8. (a)    9. (b)    10. (b)
11. (c)    12. (b)    13. (c)
14. (a) False    (b) False    (c) True    (d) True
15. (a)    16 (c)



## 1.10 FURTHER READINGS

---

1. Mano, M. M., Digital Logic and Computer Design, PHI.
2. Mano, M. M., Computer System Architecture, PHI.
3. Malvino, Albert Paul & Leach, Donald P., Digital Principles and Applications, Mcgraw-Hill International.
4. Lee, Samuel C, Digital Circuits and Logic Design, PHI.
5. Talukdar, Dr. Pranhari, Digital Techniques, N. L. Publications.



## 1.11 MODEL QUESTIONS

---

1. Distinguish between combinational circuit and sequential circuit.
2. With truth table and logic diagram explain the working of a full-adder circuit.
3. With truth table and logic diagram explain the working of a Full-subtractor circuit.



4. What do you mean by multiplexer ? With diagram explain the working of a 8-to-1 multiplexer.
5. Explain the principle of an encoder. Draw a decimal-to-BCD encoder.
6. What are the differences between asynchronous and synchronous counter ? Draw a MOD-8 counter and explain its working principle.
7. Draw a logic diagram with output wave form of a 4-bit Serial In- Parallel Out shift register for an input of 1101. Explain its operation.
8. Why is square wave clock pulse converted to a narrow spike to be used for flip-flops ? Draw a RC differentiator circuit to convert a square wave into a narrow spike.
9. What is called racing ? To get rid of racing what techniques are used ?
10. What do you mean by magnitude comparator? Draw a block diagram and the function table of the magnitude comparator SN 7485.

\*\*\*\*\*



## UNIT STRUCTURE

- 2.1 Learning Objectives
- 2.2 Introduction
- 2.3 Input-Output Devices
- 2.4 Input-Output Interface
- 2.5 Different I/O techniques
  - 2.5.1 Programmed I/O
  - 2.5.2 Interrupt-Driven I/O
  - 2.5.3 Direct Memory Access (DMA)
  - 2.5.4 I/O Processors
- 2.6 Let Us Sum Up
- 2.7 Further Readings
- 2.8 Answers To Check Your Progress
- 2.9 Model Questions

---

### 2.1 LEARNING OBJECTIVES

---

After going through this unit, you will be able to :

- learn about the input and output devices
- learn how data transfer takes place between computer and peripherals
- know about the interrupts
- describe input-output processors

---

### 2.2 INTRODUCTION

---

In the previous unit, we have learnt about the various digital components. In this unit, we will discuss how the data is fed to the computer and the way they are being processed, that is, how the data is being transferred to and from the external devices to the CPU and the memory.

---

### 2.3 INPUT- OUTPUT DEVICES

---

The computer communicates with the users with the help of input and output devices. Input devices enter data and instructions into the

computer for processing and the output devices display the result obtained for the users to see. The input-output devices attached to the computer are also known as peripherals. There are three types of peripherals: input, output and input-output peripherals. Most common peripherals are :

- Keyboard,
- Monitor,
- Printers,
- The auxiliary storage devices such as magnetic disks and tapes.

A brief description of their functions are given below :

#### **Keyboard and Monitor :**

Keyboard allows the user to enter alphanumeric information. On pressing a key, a binary coded character, typically 7 or 8 bits in length is sent to the computer. The most commonly used code is a 7-bit code referred to as ASCII ( American Standard Code For Information Interchange). Each character in this code is represented by a unique 7-bit binary code; thus a total of 128 different characters can be represented as shown in table 2.1

**Table 2.1** The American Standard Code for Information  
Interchange(ASCII)

	0 0 0	0 0 1	0 1 0	0 1 1	1 0 0	1 0 1	1 1 0	1 1 1
0000	NUL	DLE	SP	0	@	P		p
0001	SOH	DC <sub>1</sub>	!	1	A	Q	a	q
0010	STX	DC <sub>2</sub>	"	2	B	R	b	r
0011	ETX	DC <sub>3</sub>	#	3	C	S	c	s
0100	EOT	DC <sub>4</sub>	\$	4	D	T	d	t
0101	ENQ	NAK	%	5	E	U	e	u
0110	ACK	SYN	&	6	F	V	f	v
0111	BEL	ETB	'	7	G	W	g	w
1000	BS	CAN	(	8	H	X	h	x
1001	HT	EM	)	9	I	Y	i	y
1010	LF	SUB	*	:	J	Z	j	z
1011	VT	ESC	+	;	K	[	k	{
1100	FF	FS	,	<	L	\	l	
1101	CR	GS	-	=	M	]	m	}
1110	SO	RS	.	>	N	^	n	~
1111	SI	US	/	?	O	_	o	DEL

The **Control Character** are :

NUL - Null

SOH - Start of Heading

STX - Start of Text

ETX - End of Text

EOT - End of Transmission

ENQ - Enquiry

ACK -

AcknowledgeBEL -

Bell

BS - Backspace

HT - Horizontal Tab

LF - Line feed

VT - Vertical Tab

FF - Form feed

CR - Carriage Return

SO - Shift Out

SI - Shift In

DLE - Data Line Escape

DC1, DC2, DC3, DC4 - Device

ControlsNAK - Negative

Acknowledge

SYN - Synchronous Idle

ETB - End of Transmission Block

CAN - Cancel

EM - End of

MediumSUB- Substitute

ESC - Escape

FS - File Separator

GS - Group Separator

RS - Record

SeparatorUS - Unit

Separator DEL -

Delete

Video monitors may be of different types but most commonly used one is the CRT (Cathode Ray Tube) monitors. Another one that becomes popular now-a-days is LCD (Liquid Crystal Display) monitor.

The monitor displays a cursor on the screen which marks the position on the screen where the next character is to be inserted.

### **Printers :**

Printers produce a hard copy of the output data. Based on the technology used printers may be either impact or non-impact. Impact printers operate by using a hammer to strike a character against an inked ribbon; the impact then causes an image of the character to be printed. Types of impact printers are dot-matrix, daisy wheel, line printers. Non-impact printers do not touch the paper while printing. Non-impact printers like Laser printers use a rotating photographic drum to imprint the character images and then transfer the pattern onto the paper. Inkjet, deskjet and thermal printers also fall under this category.

### **Magnetic Tapes and Disks :**

The magnetic tape drives and disk drives are used to read and write data to and from the magnetic tapes and the disks. The surfaces of these output devices are coated with magnetic material so as to store data in them. These surfaces can be magnetized and the presence of a magnetic field represents a '1' bit and the absence of a magnetic



field represents a '0' bit. The magnetic disk is a circular metal disk that is coated on both sides to store data. The magnetic tape stores data

in a sequential manner. To process a data in a tape ,it must begin searching at the beginning and check each record until the desired record is found.

In case of magnetic disks, the data can be stored and accessed directly and so it is fast. Floppy disks, hard disks are examples of magnetic disks.



### CHECK YOUR PROGRESS

1. Fill in the blanks :

- (a) The Keyboard is a \_\_\_\_\_ device.
- (b) Daisy Wheel is a \_\_\_\_\_ printer and the Deskjet isa \_\_\_\_\_ printer.
- (c) Phosphor on being struck by \_\_\_\_\_ emits light on the monitor screen.
- (d) Magnetic tape stores data on its \_\_\_\_\_ sides.

---

## 2.4 I/O INTERFACE

---

The I/O Interface is responsible for exchange of data between the peripherals and internal storage of the computer. Instead of directly connecting the peripherals to the system bus, an I/O Interface is used in between because,

- 1. The mode of operation of the different peripherals is different from the operation of the CPU and memory.
- 2. The data transfer rate of the peripherals is much slower compared to that of CPU and memory.
- 3. Peripherals use different data formats than the word format in the CPU and memory.

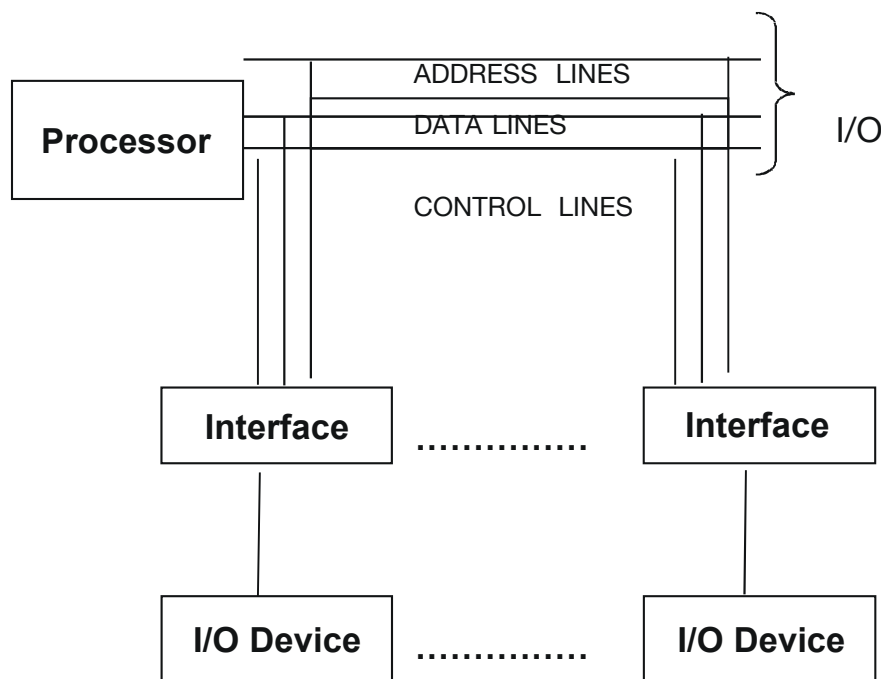
The interface units are hardware components between the CPU and the peripherals that supervise and synchronize all the I/O transfers.

Two main types of interfaces are CPU Interface that corresponds to the system bus and input-output interface that is tailored to the nature

and operation of the peripheral devices. The major functions of the I/O Interface units are :

1. Data Conversion; conversion between digital and analog signals.
2. Synchronization; synchronizes speeds of the CPU and other peripherals.
3. Device Selection; selection of the I/O device by the CPU in a queue manner.

Input-Output devices are attached to the processor with the help of the I/O Interface as shown in the figure below:



**Fig. 2.1 : Processor and the I/O Interface**

The I/O interface circuit also has an address decoder, control circuits and data and status registers. Whenever the CPU has to communicate with a particular device, it places its address in the address lines; the address decoder in the address lines then helps in recognizing the address of its connected I/O device. The I/O interface can also generate addresses associated with the device that it controls. The control circuits accept the detailed control information from the CPU.

The I/O interface also holds the responsibility of data buffering. The

data registers in it

holds the data being transferred to and from the interface. This is necessary as the data transfer rate into and out of the CPU or memory is quite high as compared to the peripheral devices. The status registers provides the current status information relevant to the input-output device operation. The data and status registers are connected to the data bus.

The input-output interface can detect errors and then after detection reports them to the CPU.



### CHECK YOUR PROGRESS

2. Write True or False :

- (a) The I/O Interface is a hardware component in the computer.
- (b) The Control lines of the System Bus carries the addresses of the peripherals.
- (c) Main memory can transfer data at high speed compared to the peripherals.
- (d) The Status register in the I/O Interface holds the data that is being transferred.

---

## 2.5 DIFFERENT I/O TECHNIQUES

---

There are different I/O techniques in the computer. Some may involve the CPU for transferring data between the computer and the input-output devices and others may directly transfer data from and to the memory units. The three possible techniques for I/O operations are :

- Programmed I/O,
- Interrupt - Driven I/O,
- Direct Memory Transfer (DMA)

---

### 2.5.1 Programmed I/O

---

In the Programmed I/O, data transfer takes place between the peripherals and the processor during program execution.

When the CPU encounters an I/O instruction, while executing a computer program, it immediately sends the address of the particular I/O device in the address lines of the I/O bus and also at the same time it issues a command to the particular I/O interface in the control lines of the I/O bus.

There are four types of commands that an interface may receive:

- o Control command
- o Status command
- o Read command
- o Write command

A *Control command* is used to activate the peripheral and tell it what to do. This command depends on the type of the particular peripheral.

A *Status command* is used to test the various status of the peripheral and the I/O interface.

A *Read command* causes the interface to get the data from the peripheral and place it in its data registers. The interface then places the data in the data lines of the bus for the processor when it requests for it.

A *Write command* causes the interface to transfer the data from the bus into its data registers. When it completes transferring data to its data registers, it then transmits that data to the particular I/O device.

In Programmed I/O, the CPU must constantly check the status of the I/O interface. That is, on issuing a command to the I/O interface, the processor must wait until the I/O operation is complete. This is a time consuming process as it keeps the CPU needlessly busy.

---

### 2.5.2 Interrupt- Driven I/O

---

In Programmed I/O the processor has to constantly monitor the status of the interface to check when it is ready for data transfer. This causes wastage of the CPU time. An alternative to this method is the Interrupt-Driven method, where after issuing a command to the interface, the CPU can switch to execution of other programs in the computer. When the interface is ready for the transfer, it will immediately send an interrupt signal to the CPU. The CPU will then do the data transfer and resume back to what it was doing before it received the interrupt signal. One of the bus control lines, called an Interrupt Request Line, is dedicated for this purpose.

In this method, the advantage is that the CPU need not spend time waiting for an I/O operation to be completed, thereby increasing its efficiency.

The routine that is executed when there is an interrupt request is called Interrupt Service Routine (ISR). When the CPU gets an interrupt signal, while it is executing an instruction of a program, say instruction  $I$ , it first finishes execution of that instruction. It then stores the current contents of the Program Counter (PC), which now points to the next instruction  $i+1$ , and stores it in a memory stack. This is its return address after execution of the ISR. The control then branches to the service routine that processes the required I/O transfer.

#### **Determining the I/O device requesting Interrupt**

When the CPU receives an interrupt signal over the interrupt request line, it must determine which I/O interface has sent it. There are different techniques for determining this. They are:

1. Software Poll
2. Vectored Interrupt
3. Bus arbitration

### **1. Software Poll**

When the CPU detects an interrupt, it branches to an interrupt- handler. This interrupt-handler poll each I/O interface to determine which interface caused the interrupt. The interface which signaled the interrupt responds positively to it. This is time-consuming.

### **2. Vectored Interrupt**

This is a more efficient technique which uses a daisy chain. It provides a hardware poll. The interrupts are signaled through a common Interrupt request line. The interrupt acknowledge line is daisy-chained through the I/O interface. On receiving an interrupt, CPU sends an acknowledge signal through it. The interface which sent the interrupt responds by placing its vector on the data lines.

### **3. Bus Arbitration**

At a time only one interface can send an interrupt signal. When the CPU detects the interrupt signal, it acknowledges through the interrupt acknowledge line. The interface signaling the interrupt then places its vector on the data lines.

## **Handling Multiple Interrupts**

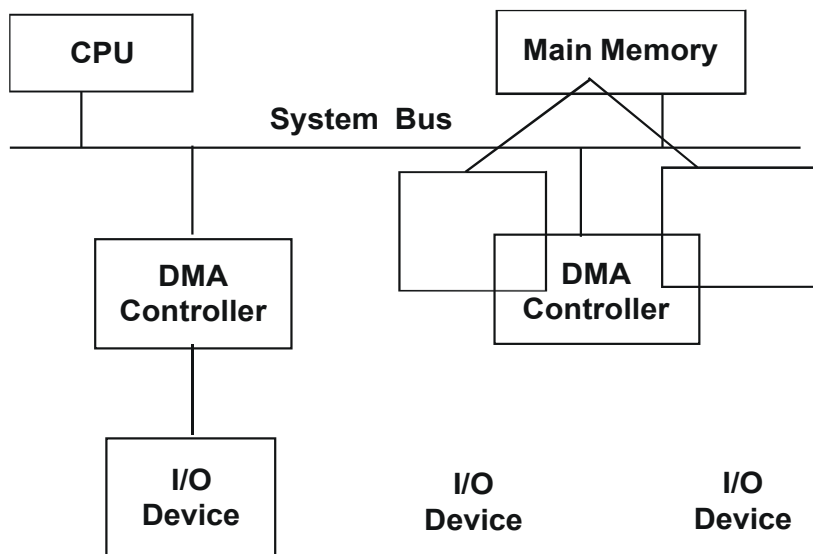
There may be multiple interrupts simultaneously. In those cases the CPU has to decide which one to service first. Priorities are thus assigned to the I/O interfaces to determine the order in which the interrupts are serviced. The devices having higher speeds such as the disks are given higher priorities, while the slower devices such as keyboards are given low priority. Also higher priority is given to the interrupts which if delayed may cause serious consequences. So, when two device interfaces sends interrupt requests to the processor at the same time, the device having higher priority will be serviced first.



### 2.5.3 Direct Memory Access

When large blocks of data are to be transferred between memory and high-speed I/O devices, a more efficient approach is used. This alternative technique is the Direct Memory Access (DMA) technique. It eliminates the continuous intervention of the CPU in transferring data.

DMA transfers are performed by a DMA controller, which is a control unit in the I/O interface. This unit carries out the functions of the processor when accessing the main memory during the data transfers. In DMA transfers, the controller takes control of the memory buses from the CPU and does the transfer between the peripheral and the memory.



**Fig. 2.2 : DMA Controllers in a Computer System**

When the CPU gets an instruction that involves huge amount of data transfer while executing a computer program, it issues a command to the DMA Controller sending the address of the I/O device, the number of words to be transferred and whether a read/write operation is to be performed. Thus the DMA, CPU just initializes the data transfer and become free for other works. i.e.,

the data transfer operation is completed by the DMA controller without the continuous involvement of the CPU.

The transfer may be in two ways / modes:

**Burst Transfer** : By Burst Transfer a block of words is transferred in a continuous burst. This mode is needed for fast devices where data cannot be stopped until the entire block is transferred. Then the controller informs the CPU by sending an interrupt signal.

**Cycle Stealing** : The controller transfers one word at a time and return the bus control to the CPU. The CPU then temporarily suspends its operation for one memory cycle to allow the DMA transfer to steal its memory cycle.

The DMA controller uses a register for storing the starting address of the word to be transferred, a register for storing the word count of the transfer and another register contains the status and the control flags. The status and the control registers are shown below

IRQ	IE		R/W	Done
-----	----	--	-----	------

IRQ : Interrupt Request

IE : Interrupt Enabled

R/W : Read or Write operation

The IRQ bit is set to 1 when the controller has requested an interrupt. When the data transfer is completed, it sets the IE to

1 thereby to raise an interrupt. When it is ready for the next command, after completion, the Done bit is set to 1. When the R/W bit is 1, the controller performs a Read operation, otherwise a Write operation is performed.

Whenever a word is transferred, the DMA Controller increments its address register to point to the next word to be transferred

and decrements its word count register. When finally the word count becomes zero, the controller stops any further transfer



### CHECK YOUR PROGRESS

3. Write True or False :

- (a) There are only 2 techniques for the I/O operations in the computer.
- (b) Read command in Programmed I/O causes the interface to get the data from I/O devices and place it in its data registers.
- (c) Interrupt Driven I/O wastes the CPU time needlessly.
- (d) Interrupt Service Routine is an interrupt signal.
- (e) In Direct Memory Access, the data transfer takes place directly between the peripheral and the memory.
- (f) The DMA Controller carries out the functions of the processor when accessing the main memory during the data transfers.

and informs the CPU by raising an interrupt.

#### 2.5.4 I/O PROCESSOR

An Input – Output processor (IOP) is a processor in addition to the CPU. It directly communicates with the input and output devices. The IOP need not interfere with the tasks of the CPU as they themselves can fetch and execute the instructions. Other than fetching and executing I/O instructions the Input Output processor can also do processing tasks like arithmetic, logic and branching.

Data from the devices are first collected into the IOP and then transferred to the memory directly by stealing one memory cycle

from the CPU. Similarly data is transferred from memory to IOP

first and the gathered data is sent to the respective I/O devices.

---

## 2.6 LET US SUM UP

---

- Data transfer between the peripherals and the computer takes place through an Input-Output Interface.
- Each input/output device is connected to the system bus with the help of the I/O interface.
- There are three different I/O techniques :
  - 1) Programmed I/O
  - 2) Interrupt-Driven I/O
  - 3) Direct Memory Access (DMA)
- In Programmed I/O, the processor must constantly check the status of the interface until the I/O task is completed, which is a waste of the CPU time.
- In Interrupt-driven I/O, the processor is free to perform other tasks while the I/O operation is going on. On its completion, the interface sends an interrupt signal to the CPU.
- In DMA, data is transferred directly between the memory and the external devices without intervening the CPU.



## 2.7 FURTHER READINGS

---

1. M. Morris Mano, Computer System Architecture, PEARSON Prentice Hall.
2. Hamacher, Vranesic, Zaky Computer Organization, Mc Graw Hill.



## 2.8 ANSWERS TO CHECK YOUR PROGRESS

---

1. a ) Input                      b ) Impact, Non-impact                      c ) electrons
2. a ) True                      b ) False                      c ) True                      d ) False
3. a ) False                      b ) True                      c ) False                      d ) False  
    e) True                      f) True



## 2.9 MODEL QUESTIONS

---

1. Briefly describe the work of the I/O Interface in data transferring.
2. Describe the functions of the I/O Interface units in a computer.
3. What is the difference between Programmed I/O and Interrupt Driven I/O? What are their advantages and disadvantages?
4. What is the task of the DMA Controller when there is a request for memory transfer?
5. What are the methods for determining which I/O device has requested an interrupt?
6. Why are priorities being assigned to the external devices?

\*\*\*\*\*

# UNIT-3 MEMORY UNIT

## UNIT STRUCTURE

- 3.1 Learning Objectives
- 3.2 Introduction
- 3.3 Memory Hierarchy
- 3.4 Main Memory
- 3.5 Semiconductor RAM
  - 3.5.1 Static and Dynamic RAM
  - 3.5.2 Internal Organization of Memory Chips
- 3.6 ROM
  - 3.6.1 Types of ROM
- 3.7 Locality of Reference
- 3.8 Cache Memory
  - 3.8.1 Cache Operation - an overview
- 3.9 Mapping Functions
  - 3.9.1 Direct Mapping
  - 3.9.2 Associative Mapping
  - 3.9.3 Set-Associative Mapping
- 3.10 Replacement Algorithm
- 3.11 Virtual Memory
  - 3.11.1 Paging
- 3.12 Magnetic Disk
  - 3.12.1 Data Organization in Magnetic Disk
  - 3.12.2 Disk Access Time
- 3.13 RAID
- 3.14 Optical Memory
  - 3.14.1 CD-ROM
  - 3.14.2 DVD Disks
- 3.15 Magnetic Tape
- 3.16 Let Us Sum Up
- 3.17 Further Readings
- 3.18 Answers To Check Your Progress
- 3.19 Model Questions



---

## 3.1 LEARNING OBJECTIVES

---

After going through this unit, you will be able to :

- learn about main memory
- describe the role of RAM, ROM and their types
- learn the concept of locality of reference
- learn about cache and virtual memory
- describe different types of mapping techniques
- describe the mechanism of paging
- illustrate the organization of data in magnetic disks as well as magnetic tapes
- learn about RAID technology
- describe the technology of optical memory

---

## 3.2 INTRODUCTION

---

In the previous unit, we have learnt about the peripheral devices associated with a computer system and various techniques with the help of which data transfers between the main memory and the input-output devices takes place.

In this unit we shall discuss about various types of memory associated with a computer system including main memory, cache and virtual memory and various technology associated with these memory units. Finally, we conclude the unit discussing the concept of secondary memory along with their types.

---

## 3.3 MEMORY HIERARCHY

---

The computer stores the programs and the data in its memory unit. The CPU fetches the instructions out of the memory unit to execute and process them.

Memory can be primary (or main) memory and secondary (or auxiliary) memory. Main memory stores programs and data currently executed by the CPU of a computer. Auxiliary memory provides backup storage of information. Data as well as instructions are transferred from the secondary memory to the main memory whenever it is needed by the CPU.

The capacity of the memory is typically expressed in terms of bytes or words (1 byte = 8 bits). Word lengths are commonly 8 bits, 16 bits and 32 bits. The size of a word is generally the number of bits that are transferred at a time between the main memory and the CPU.

Memory has different locations, which are called its addresses, to store the data. There are different methods for accessing those address locations such as **sequential access**, direct access and **random access**.

- In **Sequential access** method, the records or the data are accessed in a linear fashion, from its current location in the memory to the desired location moving through each and every record in the memory unit. For example, in case of the magnetic tapes this method is used.
- In **Direct Access**, each record has different addresses based on the physical location of the memory and the shared Read/Write head moves directly to the desired record. This method is used in magnetic disks.
- In **Random access** each location can be randomly selected and accessed directly. Main memory can be randomly accessed.

Memory has two basic operations: *Read* and *Write* operations. In Read operation, the processor reads data from a particular memory location and transmits that data to the requesting device via bus. On the other hand, a memory Write operation causes the memory to accept data from a bus and to write that particular information in a memory location.

Regarding the speed of the memory, there are two useful measures of the speed of memory units : *Memory Access Time* and *Memory Cycle Time*.

Memory Access Time is the time between the initiation of a memory operation and the completion of that operation. Memory Cycle Time is the minimum time delay that is required between the initiation of two successive memory operations, for example between two successive memory read operations.

The computer system has a memory hierarchy consisting of the storage devices in it. A typical memory hierarchy is illustrated in

figure below :

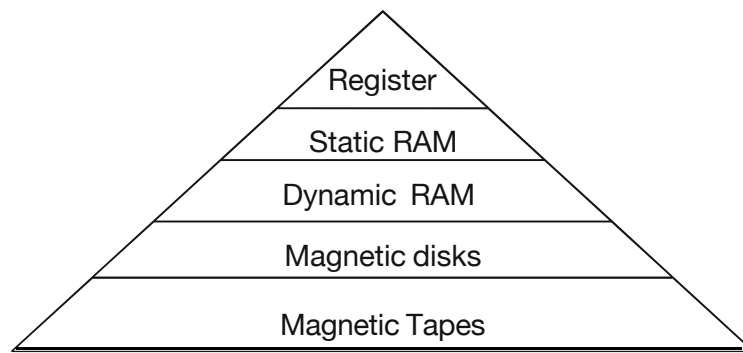


Fig. 3.1 : Memory Hierarchy

There are three key characteristics of the memory. They are *cost*, *capacity* and *access time*. On moving down the memory hierarchy, it is found that the cost of the storage devices decreases but their storage capacity as well as the memory access time increases. In other words, the smaller memories are more expensive and much faster. These are supplemented by the larger, cheaper and slower storage devices.

Thus from the above figure it can be seen that the registers are at the top of the hierarchy and so provides the fastest, the smallest and the most expensive type of memory device for the CPU to access data. Registers are actually small amount of storage available on the CPU and their contents can be accessed more quickly than any other available storage. They may be 8-bit registers or 32-bit registers according to the number of bits that they can hold in them.

Magnetic disks and Magnetic Tapes are the secondary storage mediums whose data holding capacities are much larger than the Processor Registers and the semiconductor memories which cannot hold all the data to be stored in the computer system. The magnetic tapes are more suited for the off-line storage of the large amounts of the computer data. The data are kept as records which are again separated by gaps.

---

### 3.4 MAIN MEMORY

---

The main memory is the central storage unit of the computer system. Main memory refers to the physical memory which is internal to a

computer. The word “Memory” when used usually refers to the Main Memory of the computer system. The computer can process only

those data which are inside the main memory. For execution, the programs and the data must be first brought into the main memory from the storage device where they are stored. Computer memory has a crucial role in the performance, reliability and the stability of the system. It is also an important factor in the software support of the system. More number of software can be used with more memory than with lesser ones. There are two types of main memory :

- **RAM** (Random Access Memory) and
- **ROM** (Read Only Memory).

**RAM** : In RAM, it is possible to both read and write data from and to the memory in a fixed amount of time independent of the memory location or address. RAM is also a volatile memory which means it stores the data in it as long as power is switched on. Once the power goes off, all the data stored in it is also lost. Therefore, a RAM cell must be provided a constant power supply.

**ROM** : ROM is a non-volatile semiconductor memory; that is, it doesn't lose its contents even when the power is switched off. ROM is not re-writable once it has been written or manufactured. ROM is used for programs such as bootstrap program that starts a computer and load its operating system.

---

## 3.5 SEMICONDUCTOR RAM

---

The basic building block of the semiconductor memories is the RAM chip. RAM is actually made up of a number of RAM chips. They contain a number of memory cells, which are electronic circuits having two stable states : 0 and 1. The binary information are stored in the form of arrays having rows and columns in the memories. With the advent and advances of VLSI (Very Large Scale Integration) circuits, thousands of memory cells can be placed in one chip. As a result, the cost of the semiconductor memories has dropped dramatically.

---

### 3.5.1 Static and Dynamic RAM

---

There are two main types of semiconductor RAM Memories : Static RAM (SRAM) and Dynamic RAM (DRAM) and also their

variations.

Static RAM consists of internal flip-flops to store the binary information. Each flip-flop has four to six transistors in them. SRAM can hold its data as long as the power is supplied to the circuit. Also SRAM cells can keep the data intact without any external refresh circuitry. This fact makes SRAM simple and contrasted to Dynamic RAM, which needs to be refreshed many times per second in order to hold its data contents.

The figure below shows the implementation of a SRAM cell. A latch is formed by cross-connecting two inverters. The latch is then connected to two bit lines by the transistors T<sub>1</sub> and T<sub>2</sub>. T<sub>1</sub> and T<sub>2</sub> are controlled by a word line. They are in the off-state when the word line is at the ground level.

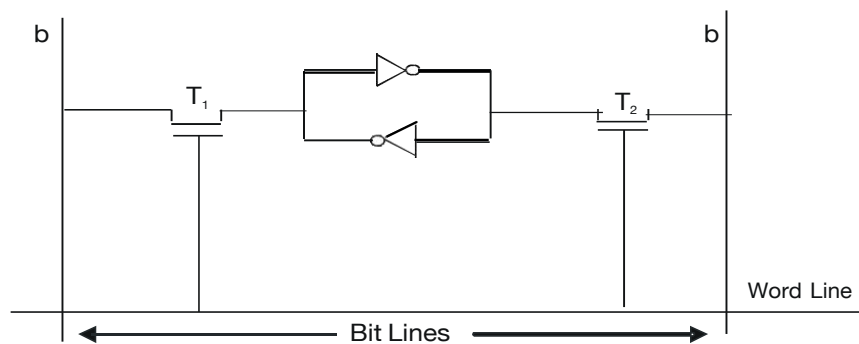


Fig. 3.2 : Implementation of SRAM cell

**Static RAM** is so called as they can retain their state as long as the power is applied. As SRAM never has to be refreshed, it is very fast. However, because it requires several transistors, it takes up more space on a RAM chip than the Dynamic RAM cells. Thus, there is less memory per chip which makes it lot more expensive. The size of SRAM is so larger comparatively than DRAM. In other words performance-wise SRAM is superior to the DRAM. But because of the size and cost of the SRAMs, DRAM is used for the system memory or main memory instead and SRAM are used for Cache memory as cache memory needs to be more faster and small. D-type and RS-type flip-flops are generally used for SRAM.

**Dynamic RAM** are so named because their cells do not retain

their states indefinitely. DRAM stores the information in the form of a charge on capacitors. The capacitor holds a charge if the bit is a “1” and holds no charge if the bit is a “0”. Unlike SRAM, DRAM uses only one transistor to read the contents of the capacitor. The capacitors are very tiny and can hold a charge only for a short period of time, after which it starts fading away. Therefore a refresh circuitry is required in case of DRAM cells to read the contents of the cell and to refresh them with a fresh charge before the contents are lost. This refreshing of the cells are done hundreds of times in every second irrespective of whether the computer is using the DRAM memory at that time or not. So the DRAM is slower than the SRAM just because of the refresh circuitry overhead.

DRAMs are used for the computer’s main memory system as they are cheaper and take up much less space than the SRAM. Even though there is the overhead of the refresh circuitry, it is but possible to use a large amount of inexpensive main memory. The figure given below is a DRAM cell consisting of a capacitor C and a transistor T.

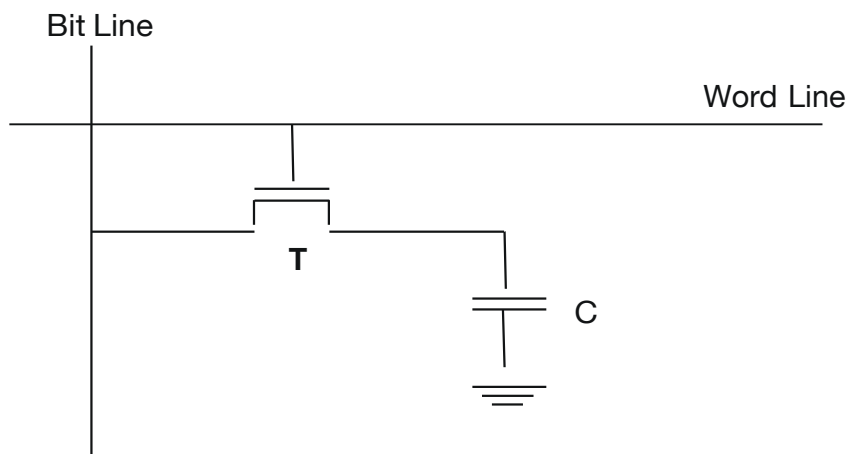
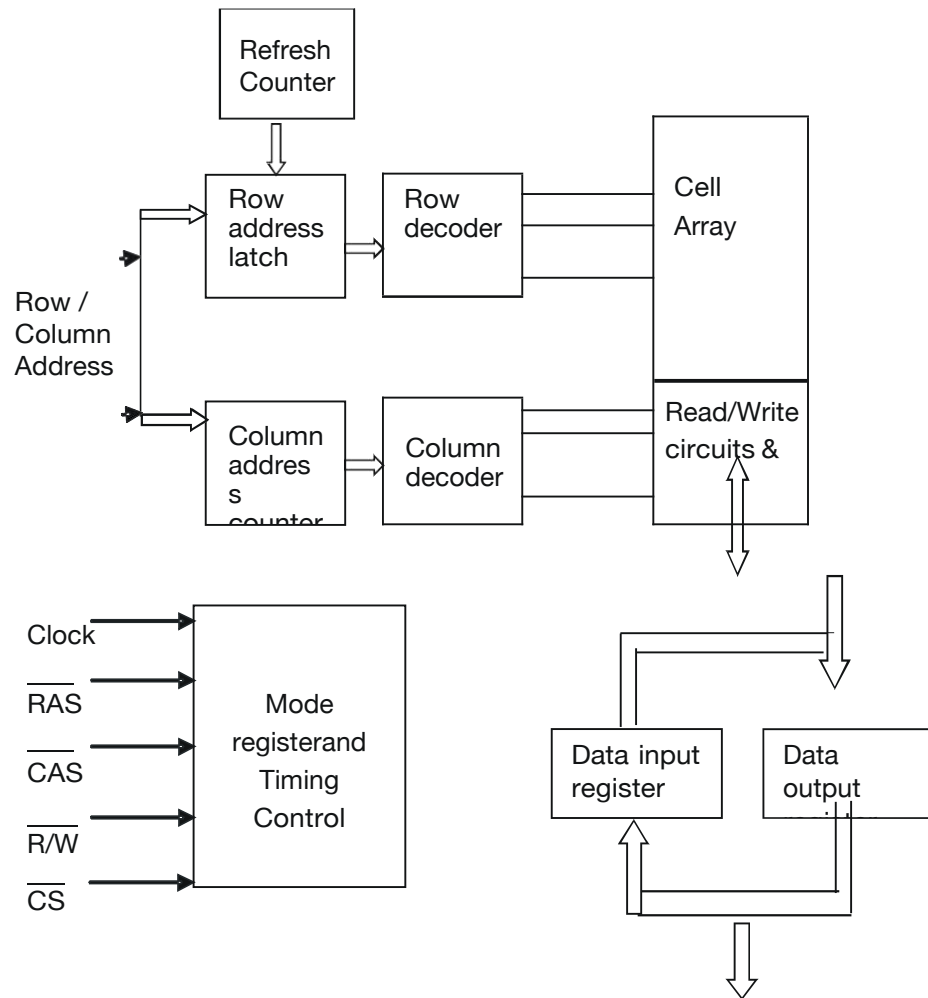


Fig. 3.3 : A DRAM cell

To store the binary information in this cell, the transistor T is first turned on and voltage is applied to the bit line. This causes the capacitor to get some amount of charge. After the transistor is turned off, the capacitor begins to discharge. Therefore, the cell contents can be got correctly only if it is read before the capacitor’s charge falls below some threshold value. During a

Read operation, the transistor in a DRAM cell is first turned on. To check whether the amount of charge on the capacitor is above the threshold value, there is a sense amplifier connected to the bit line. If so, the amplifier pulls the bit line to full voltage to represent logical 1. This voltage then recharges the capacitor to its full voltage.



3.4: Synchronous DRAM Data

On the other hand, if it was detected that the amount of charge on the capacitor is below the threshold value, then the bit line is pulled down to the ground level so that the capacitor now has no charge at all, that is it will represent logical 0. Thus, reading the cell contents automatically refreshes its cell contents. The time taken in doing all this is very short and is expressed in nanoseconds.

DRAM can again be Synchronous DRAM and Asynchronous DRAM. The DRAM discussed above is Asynchronous DRAM,



that is, the memory is not synchronized to the system clock. The memory signals are not at all coordinated with the system clock.

The Synchronous DRAM or SDRAM is synchronized with the system clock; that is, it is synchronized with the clock speed of the microprocessor. All signals are according to the clock so the timings are controlled and tight. The figure below shows the structure of SDRAM.

The speed of the DRAM has become more critical as the electronic systems that utilize these devices are now operating at increasing speeds. Thus SDRAM will soon be replacing the conventional DRAM as it is designed to work with higher operating bus speeds in the computer systems.

---

### 3.5.2 Internal Organization of Memory Chips

---

Internally, the memory in the computer system is organized in

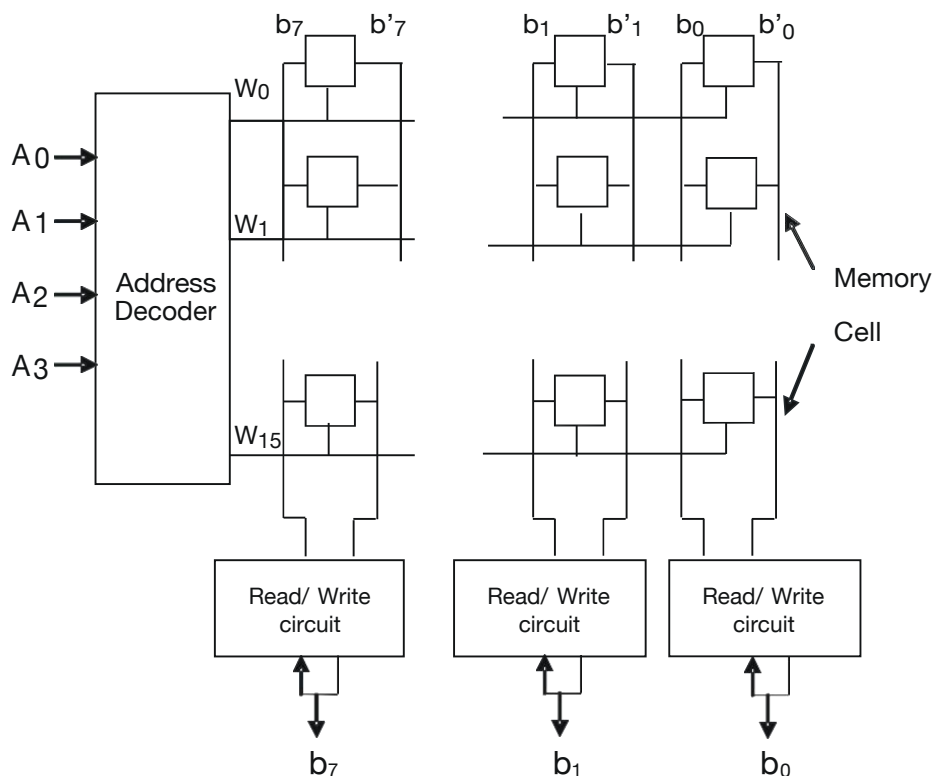


Fig.3.5: Organization of cells in a memory chip

the form of array of rows and columns. Each cell in the array can hold one bit of information. Each row in the array forms

a memory word. The cells in the column are all connected to a Read/Write circuit. The figure below presents a possible organization of the memory cells.

Let us consider a simple example of the organization of a 64-bit memory. If the memory is organized into 16 groups or words, then it can be arranged as 16 x 4 memory, that is, it contains 16 memory words and each of the word is 4 bits long. There are also other ways of organizing the same memory, like for example, it can be arranged as 64 x 1, 32 x 2 or 8 x 8.



## CHECK YOUR PROGRESS

1. Compare the characteristics of SRAM and DRAM.
2. Fill in the blanks :
  - (a) Data and \_\_\_\_\_ are transferred from the secondary memory to the \_\_\_\_\_ whenever it is needed by the CPU.
  - (b) The records or the data are accessed in a linear fashion, from its current location in the memory in \_\_\_\_\_ access.
  - (c) Memory \_\_\_\_\_ time is the time between the initiation of a memory operation and the completion of that operation.
  - (d) Memory \_\_\_\_\_ time is the minimum time delay that is required between the initiation of two successive memory operations.
  - (e) Memory access time is \_\_\_\_\_ in Magnetic disks than in magnetic tapes.
  - (f) Registers are small storage inside the \_\_\_\_\_.
  - (g) RAM can access data in a fixed amount of time \_\_\_\_\_ of the memory location or address.
  - (h) RAM is a \_\_\_\_\_ memory and ROM is a \_\_\_\_\_ memory.
  - (i) Registers are measured by the number of \_\_\_\_\_ that they can hold.
  - (j) The magnetic tapes are more suited for the \_\_\_\_\_ of

the large amounts of the computer data.

## 3.6 ROM

ROM (Read Only Memory) is another type of main memory that can only be read. Each memory cell in ROM is hardware preprogrammed during the IC (Integrated Circuit) fabrication process. That is the code or the data in ROM is programmed in it at the time of its manufacture. The data stored in ROM is not lost even if the power is switched off. For this reason, it is called a non-volatile storage. It is used to store programs that are permanently stored and are not subject to change. The system BIOS program is stored in ROM so that the computer can use it to boot the system when the computer is switched on.

The figure below shows a possible configuration for a ROM memory cell.

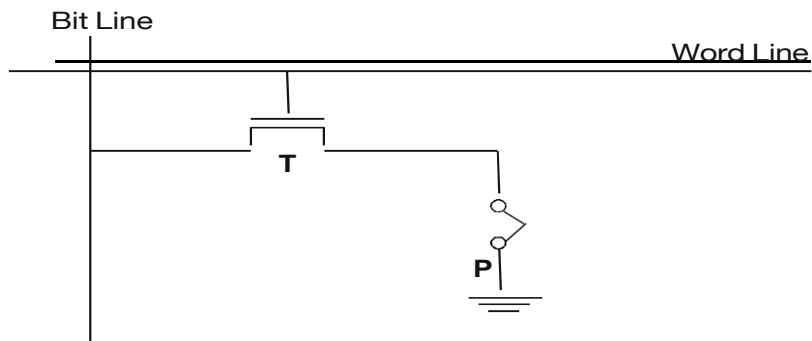


Fig. 3.6 : A ROM memory cell

If the point at P is connected to ground, then a logic value 0 is stored in the cell and if it is not connected then a logic value 1 is stored in the cell. To read the cell contents, the word line is activated. A sense circuit connected at the end of the bit line generates the output value.

ROM can also be randomly accessed as RAM. Only unlike RAM it cannot be written.

---

### 3.6.1 Types of ROM

---

There are five types of ROM :

1. ROM (Read Only Memory)
2. PROM (Programmable Read Only Memory)
3. EPROM (Erasable Programmable Read Only Memory)

4. EEPROM (Electrically Erasable Programmable Read Only Memory)
5. Flash EEPROM Memory

### **1. ROM (Read Only Memory)**

The contents of ROM are permanent and is programmed at the factory. It is designed to perform a specific function and cannot be changed. It is reliable.

### **2. PROM (Programmable Read Only Memory)**

As the name indicates, this type of ROM chips allows the user to code data into it. They were created as making ROM chips from scratch. It is time-consuming and also expensive in creating small numbers of them. PROM chips are like ROM chips but the difference is that PROM chips are created by inserting a fuse at the point P (in the above diagram). Before programming, all the memory cells in PROM contains 0. The user then can insert 1's wherever needed by burning out the fuses at those locations by sending high current pulses. However, PROMs can be programmed only once. They are more fragile than ROMs.

### **3. EPROM (Erasable Programmable Read Only Memory)**

An EPROM is another type of ROM chip that allows data to be erased and reprogrammed. They can be rewritten many times. It is also similar to a ROM chip. However, an EPROM cell has two transistors. One of the transistors is known as the floating gate and the other is known as the control gate. In an EPROM cell, the connection to ground is always made at point P. The erasing of the contents is done by exposing the chip to ultraviolet light. EPROM chips are mounted in packages that has a small glass window through which the UV light is sent into it.

### **4. EEPROM (Electrically Erasable Programmable Read Only Memory)**

The drawbacks of EPROMs are that they must be physically removed to be rewritten and also the entire chip has to be completely erased to just change a particular portion of it. EEPROM was introduced to remove these drawbacks of

EPROM. EEPROM chips can be both programmed and the contents can be erased electrically. They are versatile but slow as they are changed one byte at a time.

## 5. Flash EEPROM Memory

As EEPROM is slow to be used in products that have to make quick changes to the data on the chip, so the Flash EEPROM devices were developed. Although the flash memory devices are similar to EEPROM, there are also differences between them. In EEPROM, a single cell contents can be read and written while in flash memory single cell contents can be read but it is possible to write an entire block of cells. Also, before writing, the previous contents of the block are erased.

The advantage is that they work faster and the power consumption is low.



### CHECK YOUR PROGRESS

#### 3. Write True or False :

- (a) SRAM and DRAM are the two types of semiconductor memories.
- (b) SRAM stores the data in flip-flops and DRAM stores the data in capacitors.
- (c) Main memory is actually the Static RAM.
- (d) DRAM is more expensive as compared to SRAM.
- (e) The capacitors have their own tendency to leak their charge.

#### 4. Fill in the blanks :

- (a) A refresh circuitry is required in case of\_\_\_\_\_.
- (b) SRAM are used for \_\_\_\_\_ memory and DRAM is used for\_\_\_\_\_ memory
- (c) DRAM uses one transistor to read the contents of the \_\_\_\_\_.
- (d) The type of RAM that can hold its data without external refresh for as long as power is supplied to the circuit is called \_\_\_\_\_.
- (e) SDRAM is synchronized to the\_\_\_\_\_.
- (f) \_\_\_\_\_ is rapidly becoming the new memory

---

### 3.7 LOCALITY OF REFERENCE

---

During program execution, memory access time is of utmost importance. It has been observed that data and instructions which are executed repeatedly are located near to each other. Many instructions in localized areas of the program are executed repeatedly during some time period and the other remaining instructions of the program are accessed relatively infrequently. This characteristic of the program is referred to as the “**locality of reference**”. There are two types of locality of references. These are : *temporal locality* and *spatial locality*.

- **Temporal locality of reference** means that a recently executed instruction is likely to be executed again very soon. This is so because when a program loop is executed the same set of instructions are referenced and fetched repeatedly. For example, loop indices, single data element etc.
- **Spatial locality of reference** means that data and instructions which are close to each other are likely to be executed soon. This is so because, in a program, the instructions are stored in consecutive memory locations. For example, sequential code, array processing, code within a loop etc.

---

### 3.8 CACHE MEMORY

---

To make use of the locality of reference principle, a small high-speed memory can be used to hold just the active portions of code or data. This memory is termed as **cache memory**. The word *cache* is pronounced as *cash*. It stores data and instructions which are to be immediately executed. Two types of caching are commonly found in computers. These are : *memory caching* and *disk caching*.

- **Memory Caching**

In case of memory caching, the cache is made up of high-speed static RAM (SRAM). Static RAM is made up of transistors that do not need to be constantly refreshed. It is much faster than the dynamic RAM; access time is about 10 nanoseconds(ns). The main memory is usually made up of dynamic RAM(DRAM) chip. It is directly addressed by the CPU and its access time is about 50 ns. Data and instructions stored in cache memory are transferred to the CPU many times faster as compared to main memory. By using an intelligent algorithm, a cache contains the data that is accessed most often between a slower peripheral device and the faster processor. SRAM chips are much more expensive as compared to DRAM chips. If the whole main memory is made using SRAM chips, then the need of cache memory will be eliminated.

Some memory caches are built into the architecture of microprocessors. These are called **internal cache**. For example, the *Intel 80486* microprocessor contains a 8K memory cache and the *Pentium* has a 16K cache. Such internal caches are often called **Level 1(L1)** caches. Cache outside the microprocessor i.e., on the motherboard is called **external cache or Level 2(L2)** cache. External caches are found in almost all modern personal computers. These caches are placed between the CPU and the DRAM. Like L1 caches, L2 caches are composed of SRAM but they are much larger.

- **Disk Caching**

Disk caching works under the same principle as memory caching, but instead of using high-speed static RAM, it uses dynamic RAM. The most recently accessed data from disk is stored in the main memory which is made up of DRAM. When a program needs to access data from the disk, it first checks the disk cache to see if the data is there. Disk caching can improve the performance of applications, because accessing a byte of data in DRAM can be





---

### 3.8.1 Cache Operation - an overview

---

When the CPU sends an address of instruction code or data, the *cache controller* examines whether the content of the specified address is present in the cache memory. If the requested data or instruction code is found in the cache, the cache controller enables the cache memory to send the specified data/instruction to the CPU. This is known as a '**hit**'. If it is not found in the cache memory, then it is said that a '**miss**' has occurred and the cache controller enables the controller of the main memory to send the specified code or data from the main memory. The performance of cache memory is measured in terms of **hit ratio**. It is the ratio of number of hits divided by the total number of requests made. By adding number of hits and number of misses, the total request is calculated.

It is not necessary for the processor to know about the existence of cache. Processor simply issues READ and WRITE requests using addresses that refer to locations in the memory. Both the main memory and the cache are divided into equal-size units called *blocks*. The term *block* is usually used to refer to a set of contiguous address locations of some size. In a **READ** operation, the main memory is not involved. When a READ request is received from the processor, the contents of a block of memory words containing the location specified are transferred into the cache, one word at a time. Subsequently, when the program references any of the locations in this block, the desired contents are read directly from the cache. Usually, the cache memory can store a reasonable number of blocks at any given time. The correspondence between the mainmemory blocks and those in the cache is specified by a *map- ping function*. When the cache is full, it becomes necessary to implement a *replacement algorithm*. The replacement algorithm decides which block should be moved out of the cache to make room for the new block. Normally, the block that will be replaced is the one that will not be accessed or

needed again for the longest time. Cache provides a timer to control this situation.

When the memory operation is a **WRITE**, there are two ways to proceed: *write-through method* and *write-back method*.

- **Write-through method:** In this method, the cache location and the main memory location are updated simultaneously.
- **Write-back method:** This method is to update only the cache location and to mark it as updated with an associated *flag bit* also known as *dirty bit*. Here, the update of main memory occurs only when the block containing this marked word is to be removed from the cache to make room for a new block.



## CHECK YOUR PROGRESS

5. Choose the appropriate option:

- (i) The cache is made up of high-speed \_\_\_\_\_ in case of memory caching.
  - (a) EPROM
  - (b) SRAM
  - (c) ROM
  - (d) none of these
- (ii) Cache memory is used in computer system to
  - (a) Replace static RAM
  - (b) Replace ROM
  - (c) Ensure first booting
  - (d) Speed-up memory access
- (iii) Compared with primary storage, secondary storage is
  - (a) fast and inexpensive
  - (b) fast and expensive
  - (c) slow and inexpensive
  - (d) slow and expensive

6. Fill in the blanks:

- (i) \_\_\_\_\_ occurs when the requested word is not present in the cache memory.
- (ii) \_\_\_\_\_ states that data and instruction which are close to each other are likely to be accessed in near future.
- (iii) Level 2 cache are also termed as \_\_\_\_\_ cache.
- (iv) The performance of cache memory is measured in

---

## 3.9 MAPPING FUNCTIONS

---

To search for a word in the cache memory, cache is mapped to the main memory. There are three different ways that this mapping can generally be done. These are:

- *Direct Mapping*
- *Associative Mapping*
- *Set-Associative Mapping*

**Mapping functions** are used as a way to decide which main memory block occupies which line of cache. As there are less lines (or block) of cache than main memory blocks, an algorithm is needed to decide this. Let us take an example, a system with a cache of 2048 (2K) words and 64K (65536) words of the main memory. Each block of the cache memory is of size 16 words. Thus, there will be 128 such blocks (i.e.,  $16 \times 128 = 2048$ ). Let the main memory is addressable by 16 bit address (i.e.,  $2^{16} = 65536 = 64 \times 1024$ ).

---

### 3.9.1 Direct Mapping

---

The simplest way for determining the cache location for placement of a main memory block is the *direct mapping* technique. Here, the **block  $i$**  of the main memory maps onto **block  $i \text{ modulo } 128$**  of the cache memory. For example,

Block 0 of main memory =  $0 \% 128 = 0$ , i.e., Block 0 of cache memory

Block 128 of main memory =  $128 \% 128 = 0$ , Block 0

Block 256 of main memory =  $256 \% 128 = 0$ , i.e., Block 0

Block 1 of main memory =  $1 \% 128 = 1$ , i.e., Block 1

Block 129 of main memory =  $129 \% 128 = 1$ , i.e., Block 1

Block 257 of main memory =  $257 \% 128 = 1$ , i.e., Block 1

Thus, whenever one of the main memory *blocks 0, 128, 256,...* is loaded in the cache, it is stored in cache *block*

*0. Block 1, 129, 257, ...* are stored in cache *block 1*, and so on. Since more than one main memory block is mapped onto a particular cache block, contention may arise for that position even when the cache is not full. Currently resident block in

the

cache is overwritten by new block. The detailed operation of the direct mapping technique is as follows:

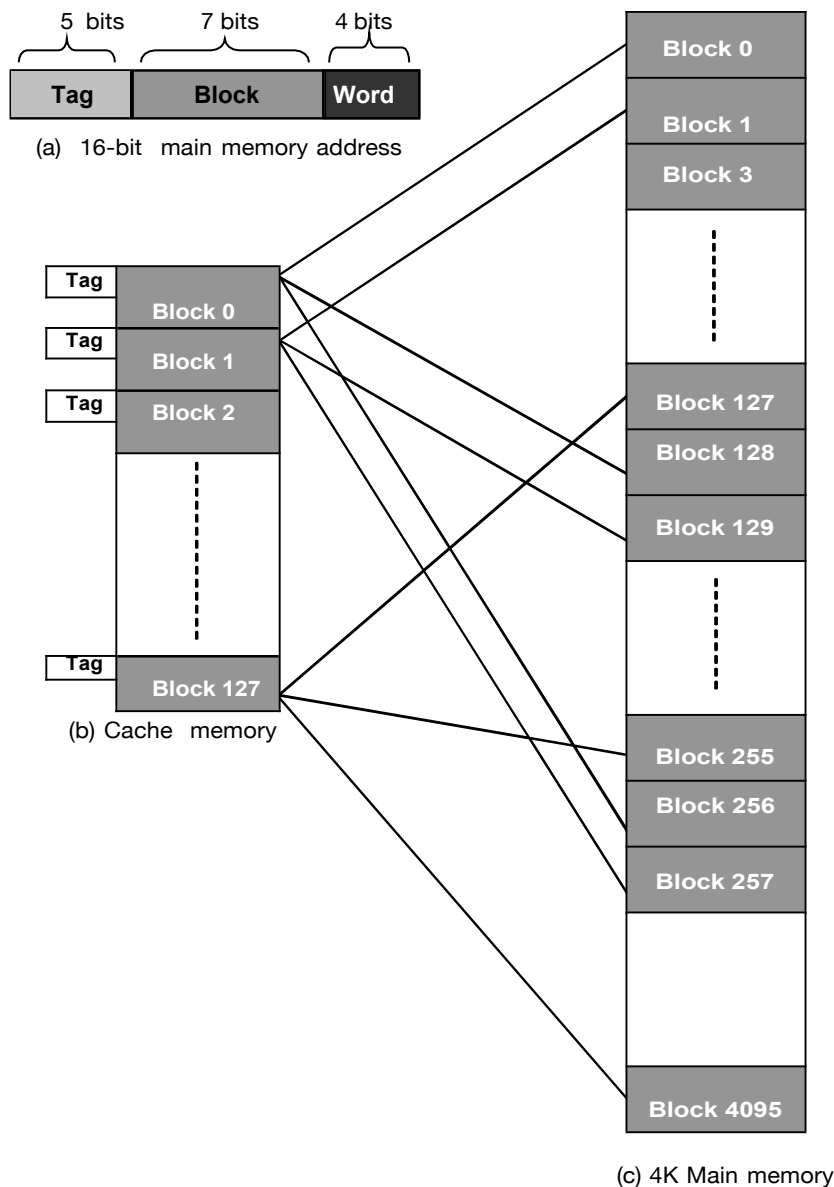


Fig.3.7 : Direct Mapping Technique

In the figure 3.7(c), the 64K words main memory is viewed as 4K (4096) blocks of 16 words each. In direct mapping technique, the 16 bit address sent by CPU is interpreted a tag, a block and a word field as shown in figure 3.7(a). The low-order 4 bits are used to select one of 16 words in a block. When a new block enters the cache, the 7 bits block field determines the cache position in which this new block can be stored. The high-order 5 bits of the memory address of the block are stored in 5 tag bits associated

with



its location in the cache. These bits identify which of the 32 blocks that are mapped into this cache block field of each address generated by the processor points to a particular block location in the cache. The tag field of that block is compared to the tag field of the address. If they match, then the desired word specified by the low-order 4 bits of the address is in that block of the cache. If there is no match, then the block containing the required word must first be read from the main memory and loaded into the cache.

The advantage of direct mapping is that it is simple and inexpensive. The main disadvantage of this mapping is that there is a fixed cache location for any given block in main memory. If a program accesses two blocks that map to the same cache line repeatedly, then cache misses are very high.

**Cache Line**  
(or **Cache Block**)  
It is the smallest unit of data that can be transferred between main memory and cache. The size of a cache line is generally 4 or 8 bytes, typically ranging from 16 to 256 bytes.

### 3.9.2 Associative Mapping

This type of mapping overcomes the disadvantage of direct mapping by permitting each main memory block to be loaded

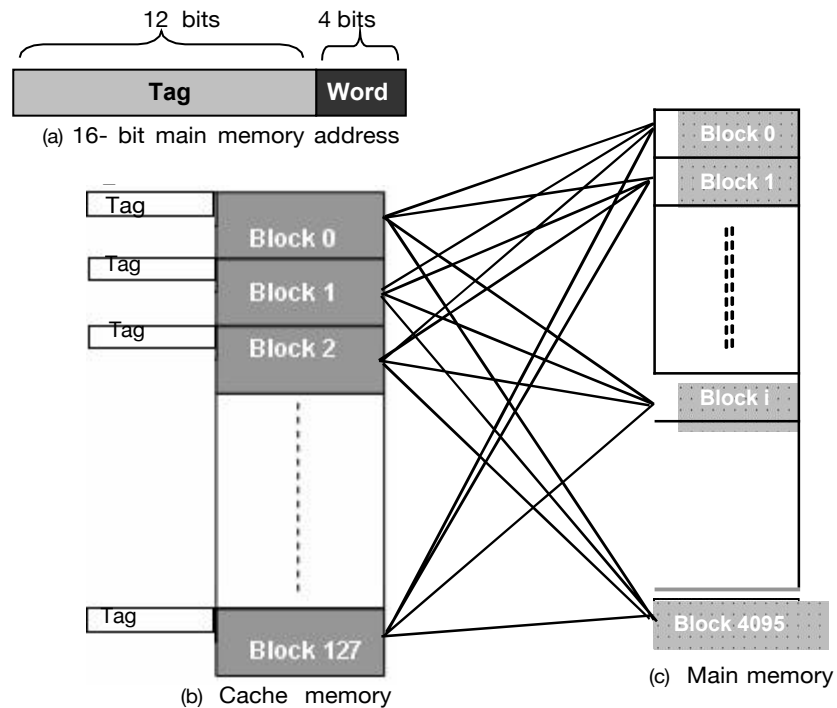



Fig.3.8 : Associative Mapping Technique



ne  
e block) :  
smallest unit of  
that can be  
d between the  
memory and the  
e cache line is  
fixed in size,  
anging from 16  
tes.

into any line of cache. To do so, the cache controller interprets a memory address as a *tag* and a *word* field. The tag uniquely identifies a block in main memory. For a 16 bit memory address, 12 bits are used as tag field and 4 bits are used as word field as shown in figure 3.8(a).

With this mapping, the space in the cache can be used more efficiently. The primary disadvantage of this method is that to find out whether a particular block is in cache, all cache lines would have to be examined. Using this method, replacement algorithms are required to maximize its potential.

---

### 3.9.3 Set-Associative Mapping

---

*Set-associative* mapping combines the best of *direct* and *associative* cache mapping techniques. In this mapping, cache memory blocks are grouped into **sets**. It allows a block of the main memory to reside in any block of a specific set. Thus, the contention problem which usually arise in direct mapping can be avoided by having a few choices for placement of block. In the figure 3.9, set-associative mapping technique is shown. Here, the cache is divided into sets where each set contains 2 cache blocks. Thus, there will be 64 sets in the cache of 2048(2K) words. Each memory address is assigned a set, and can be cached in any one of those 2 locations within the set that it is assigned to. In other words, within each set the cache is associative, and thus the name set associative mapping.

The 6 bits **set** field of the address determines which set of the cache might contain the desired block. The **tag** field of the address must then be associatively compared to the tags of the two blocks of the set to check if the desired block is present. For the main memory and cache sizes shown in figure 3.9, four blocks per set can be accommodated by a *5 bits set* field, eight blocks per set by a *4 bits set* field, and soon. If there is 128 block per set, then it requires no set bits and it becomes a fully associative technique with 12 tag bits. The other extreme condition of one block per set is the direct-mapping method. A cache that has **N** blocks per set is referred

to as **N-way set-associative cache**. Figure 3.9(b) is a 2-way set-associative cache.

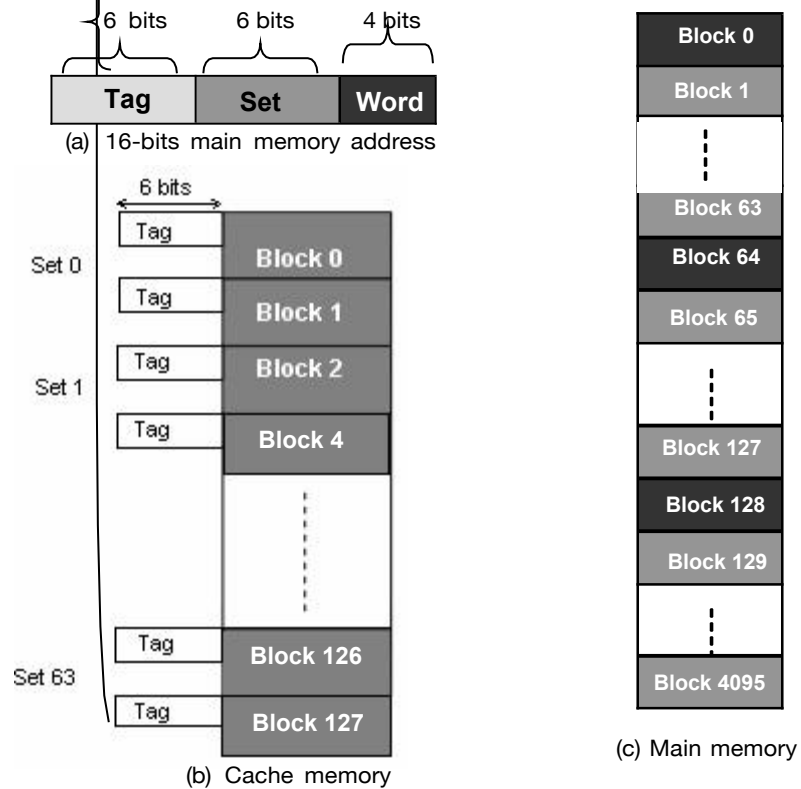


Fig.3.9 : Set-Associative Mapping Technique



## CHECK YOUR PROGRESS

7. Fill in the blanks :

- (i) Main memory block can reside in any cache position incase of \_\_\_\_\_ mapping.
- (ii) In Set-associative mapping blocks of the cache are grouped into \_\_\_\_\_.
- (iii) In \_\_\_\_\_ mapping, main memory address consists of tag, block and word fields.
- (iv) A cache that has 4 blocks per set is referred to as \_\_\_\_\_.

---

### 3.10 REPLACEMENT ALGORITHMS

---

For direct mapping where there is only one possible line for a block of memory, no replacement algorithm is required. For associative and set associative mapping, however, an algorithm is needed. At this point, we will describe the one most common replacement algorithm, called *LRU algorithm*. This algorithm is easy to understand and provides a good background to understand the more advanced replacement algorithms. Several other replacement algorithms are also used in practice such as: *first in first out* replacement algorithm, *random replacement* algorithm etc.

**Least Recently Used (LRU)** : Due to locality of reference, programs usually stay in localized areas for reasonable period of time. So there is a high probability that the blocks that have been referenced recently will be referenced again soon. Therefore, it overwrites block in cache memory that has been there the longest with no reference to it. That block is known as the least recently used block and the technique is known as ***least recently used (LRU)*** replacement algorithm. In this method, a counter is associated with each page in the main memory and it is incremented by 1 at fixed intervals of time. When a page is referenced, its counter is set to 0. Thus, counter gives the age of a page. When a page needs to be removed, the page with the highest counter is removed.

---

### 3.11 VIRTUAL MEMORY

---

Before coming to the point *virtual memory*, let us review some basics: *physical memory(RAM)* versus *secondary memory(disk space)*.

Computers these days typically have somewhere between 128 megabytes (128 million bytes) and 4 gigabytes (4 billion bytes) of main memory (RAM). What is important is that when we turn the computer off or if it crashes - anything stored in RAM is gone. That is why when we are editing a document it is a better to save to disk often. When we talk about *disks*, we are talking about the hard



address of main memory. In such a case, an address generated by the CPU is called **virtual address** or *logical address*. A set of such addresses is called the *address space*. Virtual addresses might be the same as physical addresses. **Physical address** refers to the address of a location of the main(physical) memory. A set of physical addresses is termed as *memory space*. If virtual and physical addresses are different, then virtual addresses must be mapped into physical addresses and this mapping is done by Memory Management Unit (MMU).

---

### 3.11.1 Paging

---

Paging is a method for achieving virtual memory. It is the most common memory management technique. Here, the virtual address space and memory space are broken up into several equal sized groups. To facilitate copying the virtual memory into the main memory, the operating system divides *virtual address space* into fixed size **pages**. Physical address space(memory space) is also broken up into fixed size **page frames**. *Page* in virtual address space fits into *frame* in physical memory. Each page is stored on a secondary storage (hard disk) until it is needed. When the page is needed, the operating system copies it from disk to main memory(RAM), translating the virtual addresses into physical addresses. This process of copying the virtual pages from disk to main memory is known as **paging**.

To illustrate these, let us consider a computer with 14 bit address field in its instruction and 4096(4K) words of memory(RAM). A program on this computer can address  $2^{14}$   
 $= 16384 = 16 \text{ K}$  words of memory. Thus the virtual address space is from 0 to 16383. If the address space and the memory space is divided into groups having  $1\text{K}=1024$  words each, then 16K virtual address space will consist of 16 pages

[figure]



3.10(a)] and 4K memory space will consist of 4 page frames [figure 3.10(b)].

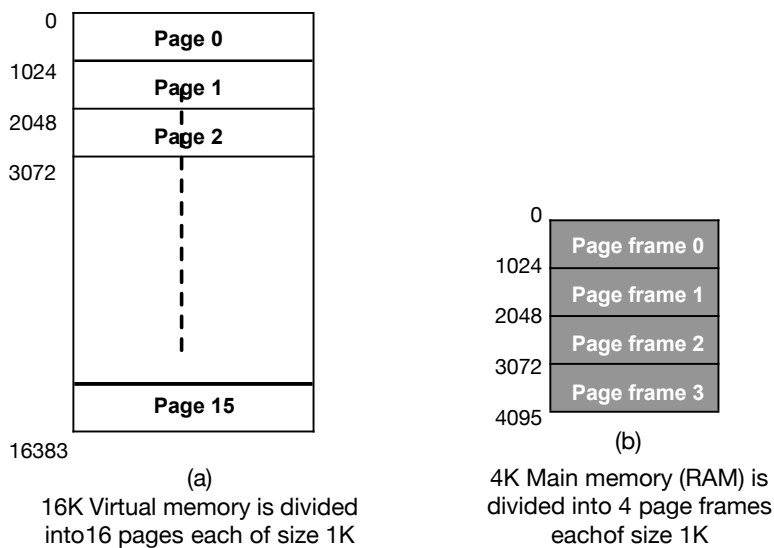


Fig. 3.10: A possible way to divide up 16K address space and 4K memory space

Out of these 16 pages, only 4 pages can accommodate in the main memory at a time. The physical address must be specified with 12 bits (as  $2^{12} = 4096 = 4K$ ). At any instant of time 4096 words of memory can be directly accessible but they need not correspond to address 0 to 4095. For the execution of the program, the 14 bit virtual address needs to be mapped into 12 bit physical address.

When the program refers to memory in fetching data or instruction or to store data, it would first generate a 14 bit address corresponding to a virtual address between 0 and 16383. This 14 bit addresses are interpreted as follows (figure 3.11) :

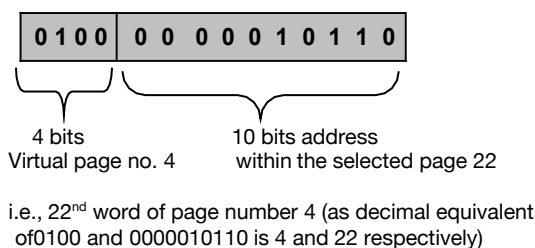


Fig.3.11

Here, 4 bits are shown as virtual page number and 12 bits as address within the selected page. The decimal equivalent of 14 bit address 01000000010110 is 4118, which is interpreted as address 22 of page

4. Having discovered that virtual page number 4 is needed, the operating system must find out where virtual page 4 is located. The page 4 may reside in any of the four page frame in the main memory or somewhere in the secondary memory. To find out where it is, the operating system searches it in the *page table* which is in the main memory. A **page table** is a table stored in main memory or in some fast memory which has one entry for each of the virtual pages. The *page table* contains the list of all the 16 pages of virtual memory and the page frame number where the page is stored in the main memory. A **presence bit** is also there to indicate whether the page is present in the main memory or not. If the page is present in the main memory then the presence bit will be 1 otherwise it will be 0. For example, if the pages 2, 4, 5, 8 are present in the main memory, the content of memory page table will be as shown in the figure 3.12.

Let us assume that the virtual page 4 is in main memory. The first 4 bits of a virtual address will specify the *page number* where the word is stored. Similarly, the first 2 bits of a physical address will specify the page frame number of the memory where the word is stored. The 4 bit page number is taken from the virtual address and it is compared with the memory page table entry. If the presence bit against this page number is 1, then the block number (2 bits) is taken and is written in the place of page number in the address (figure 3.12). Thus, a 14 bit virtual address is mapped into a 12 bit physical address. The block number is searched in the main memory and then the word located at that address is fetched.

When a reference is made to an address on a page not present in main memory, it is called a **page fault**. After a page fault has occurred it is necessary for the operating system to read in the required page from the secondary memory, enter its new physical location in the page table and then repeat the instruction that cause

page fault.

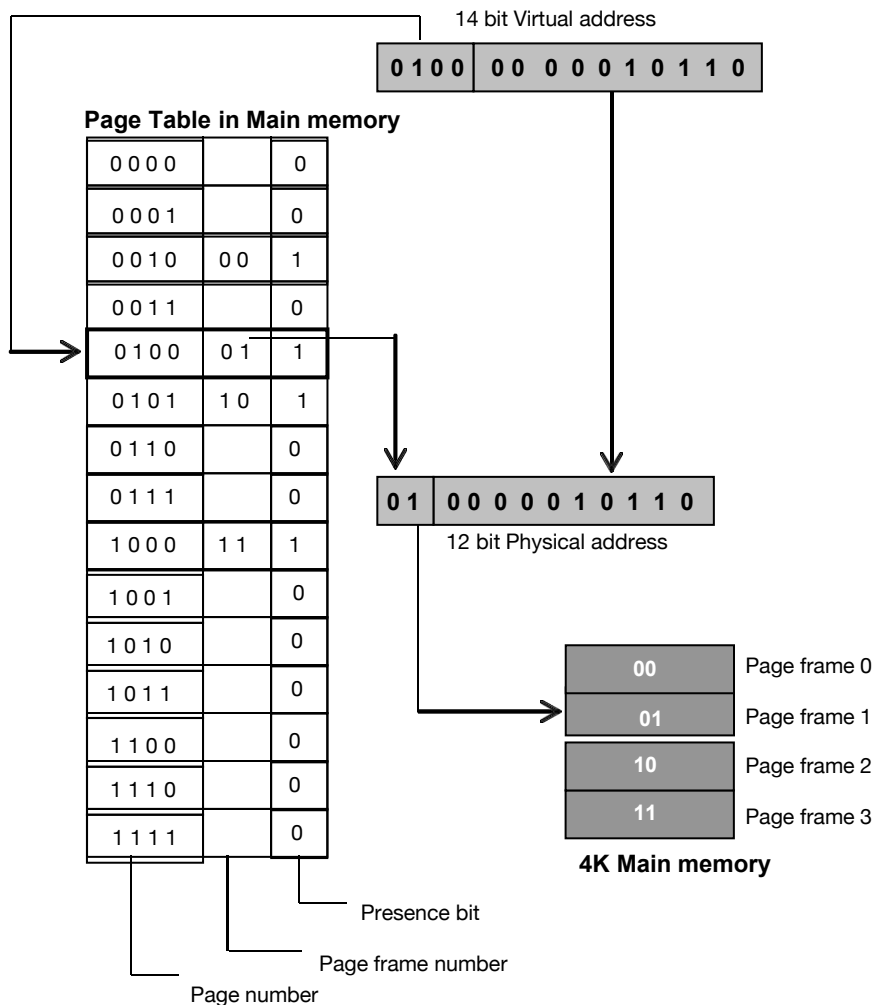


Fig.3.12:Contents of *memory page table* when pages 2, 4, 5 and 8 are present in the main memory and formulation of main memory address from virtual address



## CHECK YOUR PROGRESS

8. Fill in the blanks :

- A page replacement policy is not necessary in \_\_\_\_\_ mapped cache.
- Paging is a method of achieving\_\_\_\_\_.
- Virtual address is also known as\_\_\_\_\_ address.
- In paging, memory space is broken up into equal size \_ and address space is broken up into equal size\_\_\_\_\_.
- Set of physical address is known as\_\_\_\_\_.
- Set of virtual address is known as\_\_\_\_\_.

- (vii) \_\_\_\_\_ in virtual address space fits into \_\_\_\_\_ in physical memory

(viii) Paging is the most common \_\_\_\_\_ technique.

---

## 3.12 MAGNETIC DISKS

---

Magnetic disks are the widely used popular secondary storage medium. The magnetic disks are used for storing large amounts of data and programs. A magnetic disk is a circular plate which is constructed of metal or plastic coated with a magnetizable material on which electronic data are stored. Data can be stored on both sides of the disk. Several disks can be stacked on top of the other on a spindle. These disks are actually rotating platters with a mechanical arm that moves a read/write head between the inner and outer edges of the disks surface. A read/write head is available on each of the disk surface. A magnetic disk works on the principle of magnetic charge.

The disks rotate at very high speeds. During a read/write operation, only the disk rotates and the head is always stationary.

---

### 3.12.1 Data Organization in Magnetic Disk

---

Data on the magnetic disks is organized in a concentric set of circles, called tracks. The bits of data are stored as magnetic spots in these tracks. A gap exists between each track to prevent errors due to interference of magnetic fields. The number of bits stored in each track is equal. Therefore, the data density is more in the inner tracks than the outer ones. The more the number of tracks, the more is the storage capacity of the disk.

Tracks are again divided into sections called sectors. A small gap also exists between two sectors so as to distinguish between them. Each sector usually contains 512 bytes of data.

The set of corresponding tracks on all surfaces of a stack of disks forms a **logical cylinder**. The disk platters rotate together at a very high speed.

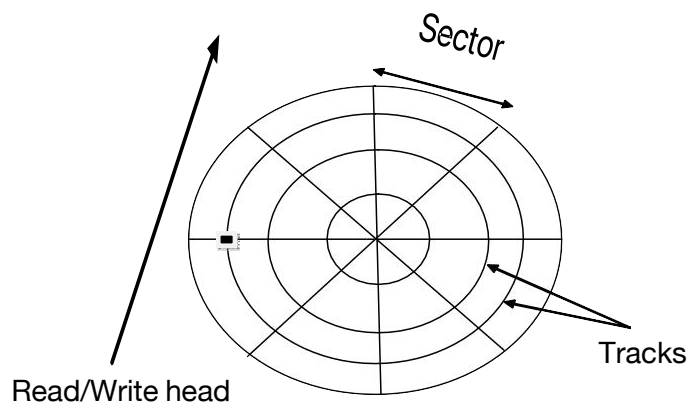


Fig. 3.13: Organization of data on Magnetic Disk

A write head is used to record the data bits as magnetic spots on the tracks and these recorded bits are detected by a change in the magnetic field produced by a recorded magnetic spot, on the disk surface, as it passes through a read head. The data on the disk surfaces are accessed by specifying the surface number, track number and sector number.

Some magnetic disk uses a single read/ write head for each disk surface while others uses separate read/ write heads for each track on the disk surface. Accordingly, the read/write head may be movable or fixed. If the magnetic disk uses a single head for each disk surface then the read/write head must be able to be positioned above any track in the surface. Therefore, the head has to be a movable head. In such case, the head is mounted on an arm and the arm can be extended or retracted to position the head on any track.

In a fixed-head disk, there is one read/write head per track in the surface. All the heads are mounted on a rigid arm that extends across all tracks.

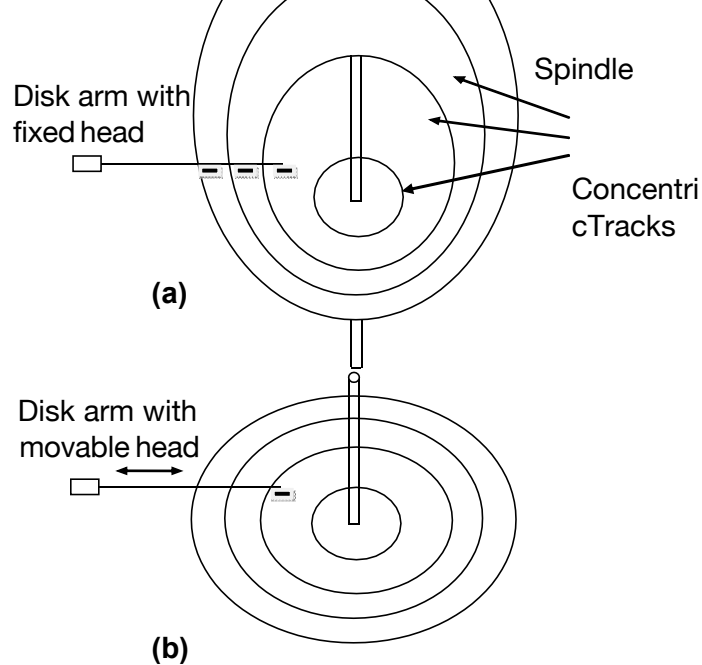


Fig. 3.14 (a) Fixed Read/Write Head, (b) Movable Read/Write Head

The read and write operations start at the sector boundaries. The bits of data are recorded serially on each track. To read or write data, the head must be first positioned to that particular track.

The heads should always be at a very small distance from the moving disk surfaces so that the high bit densities and also due to this more reliable read/write operations can be performed.

Nowadays, Winchester technology is used where both the disks and the read/write heads are placed in sealed enclosures. This technology has two advantages. First, in such disk units, the heads operate closer to the tracks as there are no dust particles, as in disk units which are not sealed. As such data can be more densely stored along the tracks and the tracks can also be closer to one another. These Winchester disk units have a larger capacity for storing data.

Secondly, in these disk units, the data integrity is more as they are not exposed to contaminating elements.

**The disk system consists of three key parts.** First is the stack of disk platters, usually referred to as the **disk**.

Second is the electromechanical mechanism that rotates the disk and moves the read/write heads, which is called the **disk drive**.

Third is the **disk controller**, which controls the disk system operation. It also provides an interface between the disk drive and the bus that connects it to the computer system.

The disks which are attached to the computer units and cannot be removed by the occasional users are called the hard disk. Those which can be inserted and removed from the system easily by the users are called floppy disks.

---

### 3.12.2 Disk Access Time

---

To perform a read or a write operation, the read/write head is first positioned on the desired track and sector. In fixed-head systems the head over the desired track is selected electronically. In a movable head system, the head is positioned on that particular track. In such systems, the time that is taken to move the head to the proper track is called the **seek time**. It depends on the initial position of the head relative to the track specified in the address.

In either case, after the read/write head is positioned over the track, the system waits until the appropriate sector passes under the read/write head. This delay in time is called the **rotational delay or latency time**.

The sum of these two delays, that is the seek time and the latency time is called the **disk access time**.

The storage capacity of a disk is a multiple of the number of



recording surfaces, number of tracks per surface, number of sector per track, and the number of bytes per sector. That is,  
 Storage capacity of a disk system = Number of recording surfaces  
 x Number of tracks per surfaces  
 x Number of sector per track  
 x Number of bytes per sector

The following Fig. shows a typical disk pack. Always remember that- **the upper surface of the top plate, and the lower surface of the bottom plate are not used for information storing.**

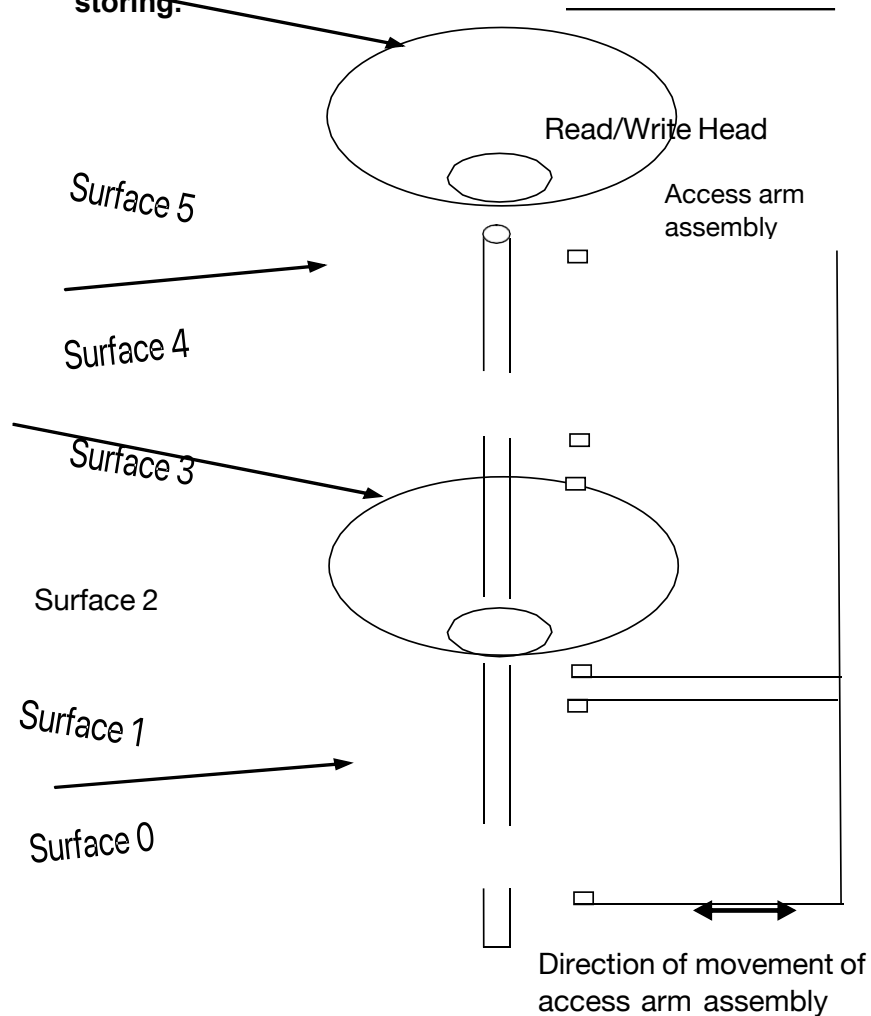


Fig. 3.15: Disk pack



## CHECK YOUR PROGRESS

9. What are tracks, sectors and cylinders?
10. Fill in the blanks :
- The density of data is \_\_\_\_\_ in the inner tracks and \_\_\_\_\_ in the outer tracks.
  - The inter-track gap prevents \_\_\_\_\_.
  - Data is recorded and accessed in magnetic disks with the help of a \_\_\_\_\_ mechanism.
  - Read/write heads may be \_\_\_\_\_ or \_\_\_\_\_.
  - Data is stored as \_\_\_\_\_ in the magnetic disks.
  - In case of Winchester disks, the disks and the heads are \_\_\_\_\_.
  - The three vital parts of a disk system are \_\_\_\_\_, \_\_\_\_\_ and \_\_\_\_\_.
  - \_\_\_\_\_ is the time required to position the head to the desired track.
  - The seek time and the rotational delay together is called the \_\_\_\_\_ time.

### 3.13 RAID

The rate in the increase in the speeds of the processor and the main memory is much more than that in the improvement of the secondary storage devices. Of course there has been an increase in the storage capacities of these devices.

However, it was been recognized that high performance could be achieved at a reasonable cost by using multiple low-cost devices to operate in parallel. This lead to the use of arrays of disks that operate independently and in parallel.

In 1988, researchers at the University of California-Berkeley proposed a multiple disk storage system, called **RAID (Redundant Array of Inexpensive Disks)**. There were six different configurations, which

were known as the RAID levels - **RAID 0**, **RAID 1**, **RAID 2**, **RAID 3**, **RAID 4** and **RAID 5**. All of these six levels share three common characteristics. They are -

1. RAID is a set of physical disk drives which operates as one logical drive.
2. The data is distributed across the physical drives of an array.
3. In case of a disk failure, the data can be recovered as identical data is stored on more than one disk in the set of the disks.

In RAID 0, data striping is used. This means a single file is stored in several separate disks by breaking the file into a number of smaller pieces. Whenever a file is read, all the disks deliver their stored portion of the file in parallel. So the total transfer time of the file is equal to the transfer time that would be required in case of a single disk system divided by the number of disks used in the array.

The disk is divided into strips, which may be physical blocks or some sectors.

In the RAID 1 scheme identical copies of the same data are stored on two disks. Data striping is also used here but the strip is mapped to both the physical disks. The two disks are mirrors of each other. If a disk failure occurs, then all the operations on its data can be done on its mirror disk. The disadvantage of RAID 1 scheme is the cost involved with it to improve the reliability.

RAID 2, RAID 3, RAID 4 and RAID 5 in all these schemes some parity calculations are done so as to achieve reliability and recover the errors. The data are not fully duplicated RAID 2 requires fewer disks but is still costly. RAID 5 distributes the parity strips across all disks.

RAID offers excellent performance and are generally used in high performance computers or in systems where higher degree of data reliability is required.



## CHECK YOUR PROGRESS

11. Find true or false :

- a. RAID combines two or more physical hard disks into a single logical unit using special hardware or software.
- b. The full form of RAID is Redundant Array of Inexpensive Disks.
- c. The levels of the RAID are hierarchy of one another.
- d. All the disks in RAID behave as one logical disk drive.
- e. In case of a disk failure, the data cannot be recovered in RAID.
- f. RAID1 level uses parity calculations for error-recovery.

12. What are the three key concepts in RAID?

---

## 3.14 OPTICAL MEMORY

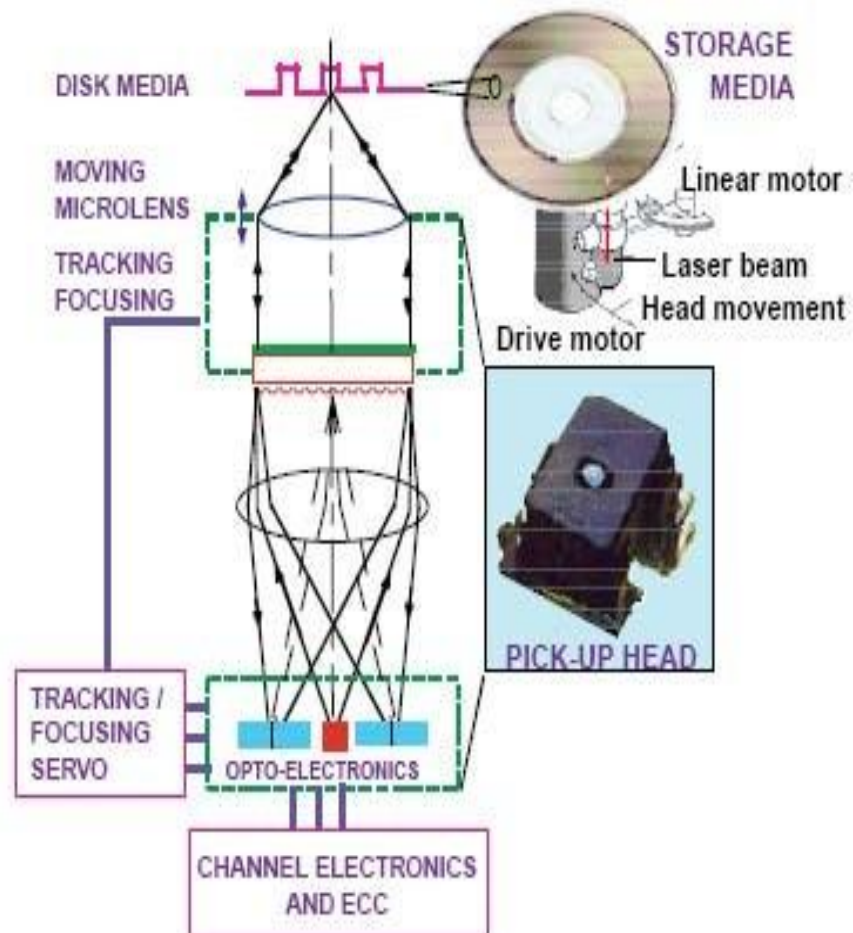
---

In optical memory, the data is stored on an optical medium like CD-ROM. The stored data can then be read with the help of a laser beam. Large amounts of data can be stored in optical memory at a very reasonable cost. The audio CD (Compact Disk) were the first application of such technology.

In the mid 1980's Sony and Philips companies developed the first generation of CDs. The CDs are non-volatile and they could not be erased.

Optical storage systems consist of a drive unit and a storage medium in a rotating disk form. In general the disks are pre-formatted using grooves and lands (tracks) to enable the positioning of an optical pick-up and recording head to access the information on the

subsequently detected by a detector in the optical head. The disk media and the pick-up head are rotated and positioned through drive motors and servo systems controlling the position of the head with respect to data tracks on the disk. Additional peripheral electronics are used for control and data acquisition and encoding/decoding.



Such a system is illustrated in Fig. 3.16.

Fig. 3.16: Key components of an optical disk system

In the past few years a variety of optical disk systems have been introduced:

- CD (Compact Disk)
- CD-ROM (Compact Disk Read Only Memory )
- WORM (Write Once Read Many)
- Erasable Optical Disk, etc.

CD-ROM disk is a shiny, silver color metal disk usually of 5 1/4-inch (12cm) diameter. It is made of polycarbonate plastic and thin layer of pure aluminum is applied to make the surface reflective. For some good quality disks, gold layer is used. A thin layer of lacquer protects it. The surface of an optical disk consists of *pits* and *land*. The information is read from pits and lands, like 1s and 0s. It is changed into binary so computers can read it. An optical reader reads the patterns of pits that stands for bytes. One CD can hold 650MB of data or 300,000 pages of text. Most CDs are read only, which means you cannot save data to the disk. This device is usually not used as a primary storage device for data.

An optical disk is mounted on an optical disk drive for reading/writing of information on it. An optical disk drive contains all the mechanical, electrical and electronic components for holding an optical disk and for reading/writing of information on it. That is, it contains the tray on which the disk is kept, read/write laser beams assembly, and the motor to rotate the disk. Access time for optical disks are in the range of 100 to 300 milliseconds.

Advantages of CD-ROM are -

- a) Storage capacity is much more in optical disks.
- b) Multiple copies of the disk along with its contents can be made inexpensively.
- c) They are transportable from one computer to another very easily.

Disadvantages are -

- a) They are read-only and contents cannot be changed or updated.

- b) The access time in optical disks is more than in case of magnetic disks.

---

### 3.14.2 DVD Disks

---

The success of CD technology and the continuing quest for greater storage capability has led to the development of DVD. **DVD**, also known as **Digital Versatile Disk** or **Digital Video Disk**, is an optical disk storage media format, and was invented and developed by Sony, and Philips in 1995. Its main uses are video and data storage. DVDs are of the same dimensions as compact disks (CDs), but store more than six times as much data.

Variations of the term *DVD* often indicate the way data is stored on the disks : DVD-ROM (read only memory) has data that can only be read and not written; DVD-R and DVD+R (recordable) can record data only once, and then function as a DVD-ROM; DVD-RW (re-writable), DVD+RW, and DVD-RAM (random access memory) can all record and erase data multiple times. The wavelength used by standard DVD lasers is 650 nm thus, the light has a red color.

A DVD disk consists of two substrates (0.6mm thick) bonded together. Each side can contain two layers called 'Layer 0' and 'Layer 1' (the outermost layer). The physical format of a DVD determines the capacity of the DVD disk. DVD capacity is determined by pit size, track pit spacing and the number of layers the disk contains. In the following some list of disks (current) and their capacities are given :

Physical Format	Capacity	Layers	Side(s)
DVD -5	4.7GB	1	1
DVD -9	8.54GB	2	1
DVD -10	9.4 GB	1	2
DVD -14	13.24GB	2	2
DVD -18	17.08GB	2	2
DVD -R	4.7GB	1	1
DVD -R	9.4GB	1	2
DVD -RW	4.7GB	1	1
DVD -RW	9.4GB	1	2

---

### 3.15 MAGNETIC TAPE

---

The magnetic tape is mostly used for off-line storage of large amount of data. They are the cheapest and the slowest methods for data storage.

The magnetic tape is a strip of plastic coated with a magnetic film. The tape is very small and generally is of 0.5 or 0.25 inch wide. The data recording on the magnetic tape is same as in the case of magnetic disks. There are a number of parallel tracks on the tape. Seven or nine bits corresponding to one character are recorded simultaneously with a parity bit.

Earlier the tapes had nine tracks each but the newer tape systems use 18 or 36 tracks, corresponding to a word or a double word. A separate read/write head is mounted one in each track so that data can be recorded and read in a sequential manner.

Data are organized in the form of records and these records are separated by gaps referred to as inter-record gaps.



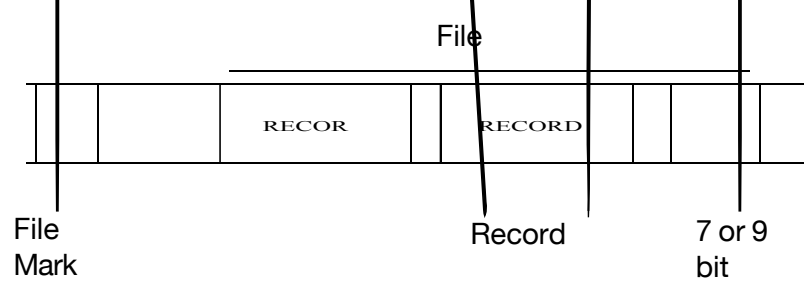


Fig. 3.17: Data Organization on a Magnetic Tape

The data on a magnetic tape is recorded and accessed in a sequential manner. Whenever the tape head reaches a record gap, the tape motion can be stopped. Each record has an identification pattern both at the beginning and at the end. The starting bit pattern gives the record number and when the tape head reaches the bit pattern at the record end, it comes to know that there is a gap after it.

A file is a collection of some related records. The starting of a file is always marked by a file mark as shown in the figure above. The gap after the file mark can be used as a header or identifier for the file. There are gaps after each record to distinguish between them. In addition to the read and write commands, there are a number of other control commands executed by the tape drive, which includes the following operations :

- a. Rewind tape
- b. Erase tape
- c. Forward space one record
- d. Backspace one record
- e. Forward space one file
- f. Backspace one file

The end of the tape is marked by EOT (End of Tape). The records in a tape may be of fixed or variable length.



## CHECK YOUR PROGRESS

13. Fill in the blanks

- (a) The width of a magnetic tape is generally \_\_\_\_\_ or \_\_\_\_\_ inch.
- (b) To access the data on a tape there is a \_\_\_\_\_ in each track.
- (c) \_\_\_\_\_ gaps are in between two records to distinguish one from another.
- (d) Magnetic tape is a \_\_\_\_\_ access device storage.
- (e) Tape motion stops when it reaches a \_\_\_\_\_.
- (f) Data on a tape are organized in the form of \_\_\_\_\_.
- (g) A tape is written from the \_\_\_\_\_ to the end.

14. Find True or False

- i) Data storage in optical memory is very costly.
- ii) CDs are non-erasable.
- iii) WORM is an optical disk product.
- iv) Recorded data can be read with the help of a laser beam.

---

### 3.16 LET US SUM UP

---

- The memory hierarchy in the computer system consists of the storage devices employed in the system with the faster but smaller devices at the top and the slow but high capacity and larger memory devices at the bottom.
- Main memory holds the programs and data currently executed by the processor.
- Main memory is of two types: Random Access Memory (RAM) and Read Only Memory (ROM).
- RAM is a volatile memory; that is, the data and programs in RAM are lost when the power is switched off whereas ROM is non-volatile memory; so it doesn't lose its contents when power

is turned off.

- RAM may be Static RAM (SRAM) and Dynamic RAM (DRAM). SRAM is simple and has more speed than DRAM. But it is larger and costlier than DRAM.
- Main memory is actually DRAM. SRAM is used in Cache memory.
- ROM is a non-volatile memory that can only be read and not written. ROM is used in certain functions in the computer because of its data permanency and data security as the data cannot be modified easily. There are 5 types of ROM such as ROM, PROM, EPROM, EEPROM and Flash EEPROM.
- The *cache* is a small amount of high-speed memory. Compared to the size of main memory, cache is relatively small. It operates at or near the speed of the processor. Cache memory contains *copies* of sections of the main the memory and is very expensive compared to the main memory.
- Caches exploit the property of locality which states that applications tend to re-use data which they have recently used. This is seen in two forms: *temporal locality* and *spatial locality*.
- Mapping is a technique for determining where main memory blocks can be placed in the cache. There are three different mapping techniques: *direct mapping*, *associative mapping* and *set-associative mapping*.
- In direct mapping, one block from main memory maps into only one possible line of cache memory. As there are more blocks of main memory than there are lines of cache, many blocks in main memory can map to the same line in cache memory.
- Associative cache mapping is the most complex, but it is most flexible with regards to where data-can reside. A newly read block of main memory can be placed anywhere in an

associative cache. If the cache is full, a replacement algorithm is used to determine which block in the cache gets replaced by the new

data.

- Set-associative mapping is designed to utilize the strengths of the previous two mappings. As with a direct mapped cache, blocks of main memory data will still map into a specific set, but they can now be in any N-cache block frames within each set. Here, The cache is divided into a number of sets containing an equal number of lines. Each block in the main memory maps into one set in cache memory similar to that of direct mapping. Within the set, the cache acts as associative mapping where a block can occupy any line within that set. Replacement algorithms may be used within the set.
- *Virtual Memory* is simply the operating system using some amount of disk space as if it were real memory.
- *Paging* is a method of achieving virtual memory. Virtual space is broken up into equal size *pages* and memory space is broken up into equal size *page frames*.
- The magnetic disk is divided into tracks, sectors and cylinders where the data are recorded.
- There may be a single read/write head for each disk surface or separate heads for each track in the surfaces. Accordingly, heads may be movable or fixed.
- Magnetic disk systems consists of three vital parts- the disk, disk drive and the disk controller.
- RAID combines two or more physical disk units and behave as a single disk unit.
- The three important concepts in RAID are - mirroring, striping and parity.
- Optical memory stores data on an optical medium and the data is read using laser beams and photodetector.



### 3.17 FURTHER READINGS

1. “*Computer Organization and Architecture*”, William Stallings, PHI
2. “*Computer Organization*”, V. Carl Hamacher, Zvonko G. Vranesic, Safwat G. Zaky, McGraw-Hill International Edition
2. “*Digital Logic and Computer Design*”, M. M. Mano, PHI
3. “*Computer System Architecture*”, M. M. Mano, PHI
4. “*Digital Techniques*”, Dr. P.H.Talukdar, N.L. Publications



### 3.18 ANSWERS TO CHECK YOUR PROGRESS

1. The characteristics of SRAM and DRAM are as follows ::
  - Simplicity :: SRAM are more simple as no refresh circuitry are needed to keep their data intact.
  - Speed :: SRAM is faster than DRAM
  - Cost :: SRAM is costlier than DRAM
  - Size :: SRAM is larger compared to DRAM.
2. a) instructions, main memory b) sequential c) Access d) Cycle  
e) less f) processor  
g) independent h). volatile, non-volatile  
i) bits j) off-line storage
3. a) True b) True c) False  
d) False. e) True f) False
4. a) Dynamic RAM b) Cache, Main  
c) capacitor d) Static RAM  
e) System Clock f) SDRAM
5. (i) (b) SRAM (ii) (d) Speed-up memory access  
(iii) (b) slow and inexpensive
6. (i) Miss (ii) Spatial locality (iii) external (iv) hit ratio

7. (i) direct (ii) sets (iii) direct (iv) 4-way set-associative cache  
8. (i) direct (ii) virtual memory (iii) logical (iv) page frames, pages  
(v) memory space (vi) address space (vii) page, page frame,  
(viii) memory management

9. Tracks are the concentric set of rings along which the data is stored in magnetic disks.

The tracks are divided into sections called sectors.

A set of corresponding track on all the surfaces of a disk pack forms a logical cylinder.

10. a) more, less, b) interference of magnetic fields,  
c) read/write head, d) fixed, movable, e) magnetic spots,  
f) sealed, g) disk, disk drive, disk controller, h) seek time,  
i) disk access time
11. a. True, b. True, c. False, d. True, e. False, f. False
12. The three key concepts in RAID are -  
(i) mirroring – writing identical data to more than one disk  
(ii) striping – splitting of data across more than one disk.  
(iii) error-correction – uses parity for error detection and recovery.
13. a) 0.5, 0.25, b) read/write head, c) inter-record, d) sequential,  
e) record gap, f) records, g) beginning
14. i) False, ii) True, iii) True, iv) True



### 3.19 MODEL QUESTIONS

---

1. What is memory hierarchy? Discuss.
2. Briefly describe Main Memory.
3. Differentiate between Static RAM and Dynamic RAM.
4. Discuss synchronous DRAM.
5. Explain ROM and its different types.
6. What is the function of cache memory? Explain the term *cache hit* and *cache miss*.

7. Discuss physical and virtual memory? What is logical and physical address?
8. What is memory and disk caching? What do mean by *level 1* and *level 2* cache?
9. What do you mean by paging? What is page table and where is it located?
10. Write short notes on any one of the following:
  - (a) Paging
  - (b) Direct Mapping
  - (c) Associative mapping
  - (d) Set-Associative mapping
11. What do you mean by locality of reference? What are its types?
12. What is Winchester disks? Briefly discuss its advantages.
13. Differentiate between fixed-head systems and movable head-systems.
14. Explain optical memory with example.
15. Describe in brief the magnetic tape systems.
16. Differentiate between CD-ROM and DVD.
17. List the types of optical storage.
18. List the major types of magnetic storage.
19. Write short notes on the following:
  - a) Hard Disk
  - b) RAID
  - c) Optical Memory
  - d) CD-ROM
  - e) DVD
  - f) Magnetic Tapes

\*\*\*\*\*



## UNIT - 4: CPU ORGANIZATION

### UNIT STRUCTURE

- 4.1 Learning Objectives
- 4.2 Introduction
- 4.3 CPU Building Blocks
  - 4.3.1 Arithmetic and Logic Unit
  - 4.3.2 Control Unit
  - 4.3.3 CPU Registers
- 4.4 BUS and its Characteristics
  - 4.4.1 System BUS
  - 4.4.2 BUS Design Elements
- 4.5 Instruction Format
- 4.6 Addressing Modes
- 4.7 Interrupts: Concept and Types
- 4.8 Instruction Execution Cycle with Interrupt
- 4.9 Hardwired and Micro Programmed Control
  - 4.9.1 Hardwired Control
  - 4.9.2 Micro Programmed Control
  - 4.9.3 Hardwired vs. Micro Programmed
- 4.10 Introduction to RISC and CISC
- 4.11 Let Us Sum Up
- 4.12 Answers to Check Your Progress
- 4.13 Further Readings
- 4.14 Model Questions

---

### 4.1 LEARNING OBJECTIVES

---

After going through this unit, you will be able to:

- know the components of Central Processing Unit
- learn about the Bus and its design issues
- describe Instruction Representation and Instruction Functionalities
- define Interrupt and Instruction Cycles
- elaborate Control Unit design

---

## 4.2 INTRODUCTION

---

The computer is composed of four functional units - CPU, Memory, Input and Output and out of these; the CPU behaves like the heart of the computer. Without CPU, the computer is a pen without ink. Hence, since the beginning of the development of computing devices, several researches over the CPU development have been going on, for enhancing the performance of the CPU. Here, in this unit, it has been discussed about the building blocks of the CPU in details.

Obviously, Buses are also playing an important role as a communication medium between the different modules of the computer and so, in this chapter it has been discussed about the Bus and its design elements as well.

A computer system works on basis of the instructions stored in the memory and hence there is a need to know about the layout of the instruction and working modes of the instructions and those are discussed in this unit.

In this unit, the intermediate steps that are needed to execute a computer instruction are discussed thoroughly and as well as about the interrupts and about the situation of the instruction cycle, if interrupt occurs.

Finally, a special focus has been given to the Control Unit, a sub division of CPU by looking the hardware to software design issues of the Control Unit. RISC and CISC architecture also have come to focus in this unit.

---

## 4.3 CPU BUILDING BLOCKS

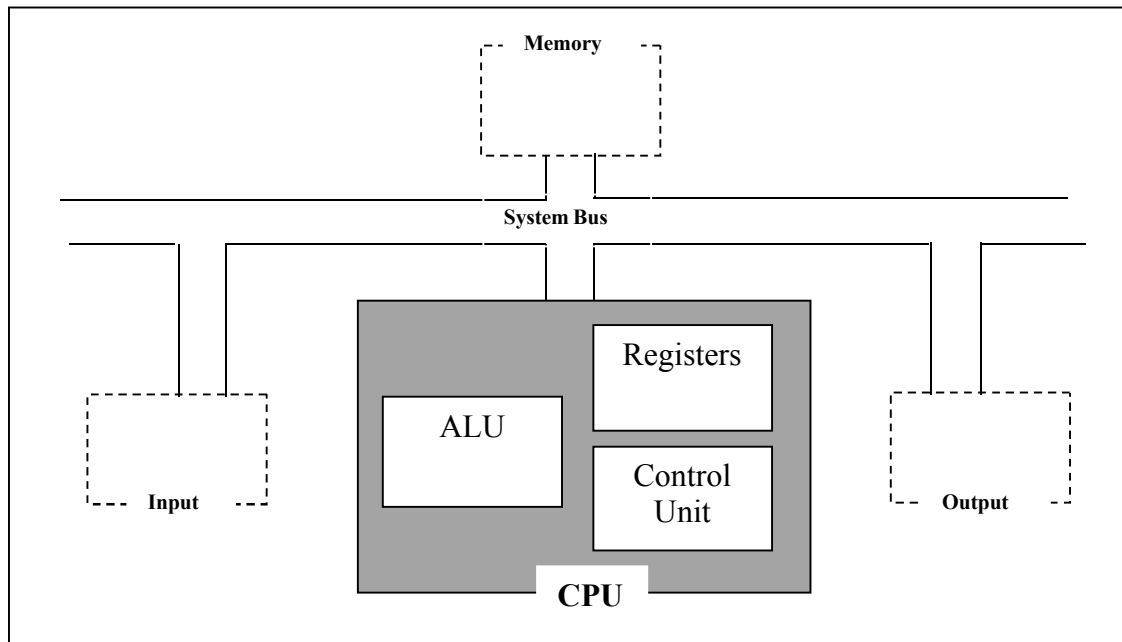
---

CPU stands for Central Processing Unit, which is the principal part of a computer. Once some tasks will be submitted to the computer through the input devices, then the CPU is the responsible for performing the operations over the given tasks and for giving the result to the outside world through the output devices. So, the part of the computer that executes program instructions is known as the Central Processing Unit (CPU) or simply Processor. Now at this point, a question arises that there should have something within the CPU for executing the program instructions as well as something to provide the way to carry out the instructions from the rest of system or simply something for controlling the sequence of the operations. In the next paragraph we will be able to get the answer or to know about the components that makes a CPU.

Basically, the major building blocks of the CPU are-

- Arithmetic and Logic Unit (ALU)
- Control Unit

In addition to the above mentioned building blocks, the Registers within the CPU are also treated as the major components of the CPU. In the **figure-4.1**, basic organization of a computer system is presented by focusing a high level abstract view of the building blocks of the CPU, along with indicating its connection to the rest of the system via the system bus.



**Figure- 4.1:** A basic organization of Computer System

### 4.3.1 ARITHMETIC AND LOGIC UNIT

John Von Neumann proposed the Arithmetic Logic Unit in 1945 when he was working on EDVAC. Arithmetic and Logic unit, simply ALU is a digital circuit for performing the arithmetic and logical operations, such as-

- Addition
- Subtraction
- Logical AND
- Logical OR
- Logical Exclusive OR
- Complement
- Increment
- Decrement
- Left Shift, Left Rotate, etc

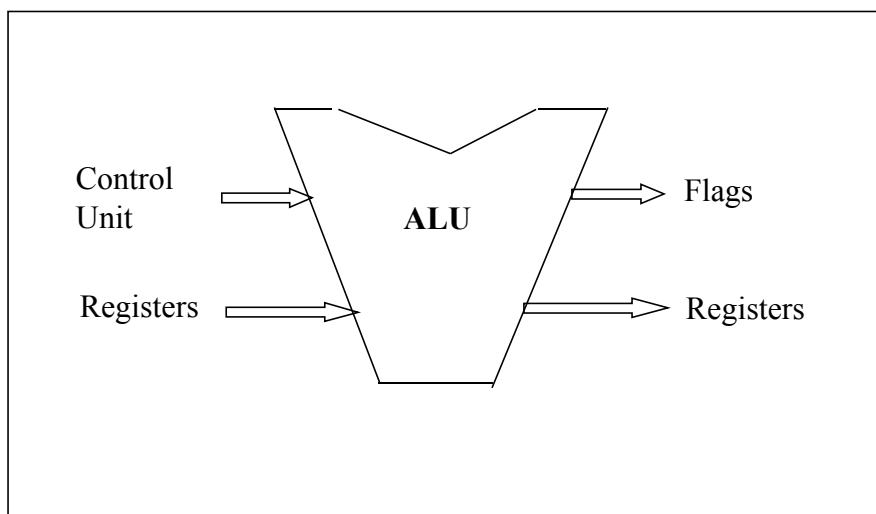
To have a clear understanding about ALU, we can consider a situation that one number is located in the memory location and the CPU needs to decrement the number by one. Then the

*Computer Organization and Architecture*

number

needs to bring to the CPU and needs to store in a high speed tiny storage element, called register and then the actual operation will be done by the ALU. Then the modified number may be stored back into the memory or retained in the register for the immediate use. That means, data are presented to the ALU in registers and the results of an operation are stored in registers. Executing one operation by the ALU, ALU may also set the flags as the result of an operation. For example, if in above mentioned decrement operation done by the ALU, the result becomes zero, then ALU will set the ZERO flag.

In the **figure-4.2**, it has been shown that how ALU is interconnected with the rest of the CPU.



**Figure- 4.2:** ALU inputs and Outputs

---

### 4.3.2 CONTROL UNIT

---

The control unit, the name itself reflects its functionality that it is a control center for sending control signal to other units to carry out their job and receiving the status of the other unit. The operations that will be performed by the ALU must be coordinated in some way and this is one of the functionalities of the control unit. It generates timing and control signals which are necessary for triggering the operations to be executed in ALU. It directs the movement of electronic signal between memory and ALU. It also directs the control signals between CPU and input/output devices for mutual understanding for communication purpose. After all, control unit can be assumed as the circuitry that controls the flow of information through the processor, coordinates the activities of the other units.

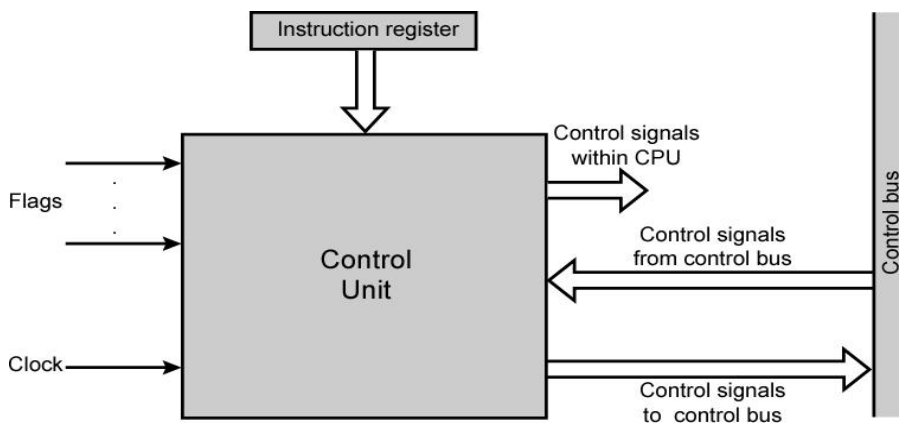
As we know that the execution of program consists of operations and these operations consist of a sequence of micro-operations which fall into one of the following categories:

- Data transfer from register to register
- Data transfer from a register to an external interface.
- Data transfer from external interface to register.
- Perform an arithmetic and logical operation, using registers for input and output.

Now, it is easy to summarize that why actually control unit is required. The basic needs of control unit are as follows-

- **Sequencing:** The control unit directs the processor to execute the micro-operations in an ordered way, based on the program being executed.
- **Execution:** The control unit makes execute micro-operations by triggering control signal.

At this point it is clear why control unit is needed, but it is not known to us that how the control unit works by issuing the control signal. Let us have a look at the **figure-4.3**.



**Figure- 4.3:** Block diagram of Control Unit

In the above **figure 4.3**, the inputs are Clock, Flags, Instruction Register, Control signals from control bus and the outputs are Control signals within the CPU and Control signals to control bus.

- **Clock:** The Control unit performs one micro-operation to be executed in each clock pulse, which is known as processor cycle time or clock cycle time.
- **Instruction Register:** Instruction register holds the current instruction to be executed. Looking at the operation code of the instruction that is currently available in the instruction register, it is determined that which micro-operations to be executed during execute cycle.

- **Flags:** Flags are set or reset by the ALU operation based on the result of the operation, i.e. status of the flags give the status of the processor and the outcome of the previous ALU operations. Therefore, based on the status of the flags, it is determined what to do in the next.
- **Control signals from control Bus:** These are the signals for mutual understanding that comes from other units and peripherals, such as interrupt signal, acknowledgment signal etc.
- **Control signals within the processor:** These signals cause the data to move from one register to another and for triggering the appropriate ALU operation.
- **Control signals to control bus:** Signals that sent to the Memory or to the other I/O module.

Now, let us consider that how to fetch the instruction from memory. For that Memory Address Register (holds the address of the memory location to be accessed currently) must be filled by the content of the Program Counter (holds the address of the next instruction to be executed) and that is done by activating a control signal from the Control Unit that opens the gates between the bits of the PC and MAR. Next, a control signal will be issued for allowing the content of MAR onto the address bus, and then a memory read control signal will be issued in the control bus to the memory from the control unit. Now, next control signal is issued by the control unit to open the gates to move the content of the data bus in MBR (Memory Buffer Register). Next one control signal will be issued to add 1 to the content of PC and store the result back to the PC again. Finally, the control unit issues a control signal to open the gates between MBR and IR such that the content of MBR can be moved to the IR and hence fetching is over. So, from the above mentioned fetching example, it is noticed that why actually control unit is needed.

---

### 4.3.3 REGISTERS

---

Registers are assumed to be the building blocks of the CPU, as they are providing themselves as a temporary storage at the time of the execution of the instructions.

The registers available in the CPU, can be categorized in two broad sections-

**User Visible Registers-** These registers make the main memory references to be minimized, as these registers are used as temporary storage for storing data, address and flags status.



Therefore a machine level language programmer can use these registers. Again, the User Visible Register can be sub-categorized into the following-

- **General Purpose Register**- The use or functionality of General Purpose registers, totally depend on the programmer. It is up to the programmer, for what purpose he wants to use this type of register. That means, any General Purpose Register can contain the operand for any opcode.
- **Data Register**- Data Registers are a bit strict that it can hold only data, but cannot be employed in the calculation of an operand address.
- **Address Register**- Address registers are like General Purpose Registers, but they may be devoted to a particular addressing mode. For example,  
  
    **Segment pointers** (a segment pointer register hold the address of the base of the segment, when a machine supports segmented addressing), **Index register** (Used for indexed addressing), **Stack pointer** (dedicated to hold the address of the top element of the stack in user visible stack addressing)
- **Conditional Codes Register**- This holds the status of the flags affected by the execution of the operation.

**Control and Status Registers**- These are the special purpose registers used by the control unit to control the operations of the CPU and by the operating system privileged program.

- **Program Counter**- Program Counter is providing the way to control of the flow of execution of the instruction, as it holds the address of the next instruction to be executed or address of the instruction to be fetch.
- **Instruction Register**- This is the register to hold the instruction that is currently being fetched. When the CPU fetch the instruction from the memory, then the fetched instruction will be first stored in MBR and from MBR, the instruction will be copied to the Instruction register. Once the instruction will be available in the Instruction register, the ALU can execute the instruction from the IR.
- **Memory Address Register**- It holds the address of the memory location that is to be accessed. Once CPU try to access one memory location, the content of Program Counter will be copied to the MAR.

- **Memory Buffer Register-** It holds the data that is to be written into the memory or CPU read recently from the memory.
- **Program Status Word-** It holds the status information of the CPU. Status information means the conditional codes along with some other information about interrupt, processor execution mode etc. Basically common fields of PSW are as follows-
  - **Sign Flag:** It is set to 1, if the result of the last arithmetic operation is negative; else it is 0.
  - **Carry Flag:** It is set to 1, if the execution of the last arithmetic operation produces a carry, otherwise it is 0. The carry flag is set or reset in case of addition as well as subtraction. In case of addition, if the operation results carry and in case of subtraction, if the borrow occurs in the operation, then carry flag will be set.
  - **Zero Flag:** It is set to 1, if the result of the last arithmetic or logical operation performed is zero; otherwise set to 0.
  - **Equal Flag:** It is set to 1, if the last logical compare result is equality; else set to 0.
  - **Parity Flag:** It is set to 1, if the arithmetic or logical operation produces a result that contains even number of 1's; else it is set to 0.
  - **Interrupt enable/ disable:** Once there will be interrupt request in the processor and the processor gives the service to the interrupt, then the interrupt enable/ disable bit will be set to 1. After completion of the interrupt service routine, the bit will be set to 0.
  - **Supervisor:** Indicates where the CPU is executing in supervisor mode or user mode. If it is in supervisor mode, the bit is set to 1; else it is set to 0.



## CHECK YOUR PROGRESS

1. Fill in the blanks.

- (i) CPU stands for\_\_\_\_\_.
- (ii) Major building blocks of CPU are\_\_\_\_\_, \_\_\_\_\_and\_\_\_\_\_also treated as major building block.
- (iii) ALU stands for\_\_\_\_\_.
- (iv) The basic needs of control unit are \_\_\_\_\_and\_\_\_\_\_.
- (v) The Control unit performs one micro-operation to be executed in each clock pulse, which is known as \_\_\_\_\_.
- (vi) \_\_\_\_\_ holds the address of the next instruction to be executed.
- (vii) \_\_\_\_\_register hold the address of the base of the segment, when a machine supports segmented addressing.

---

## 4.4 SYSTEM BUS CHARACTERISTICS

---

As we know that computer is a collection of different modules. The most basic modules of a computer are CPU, Memory and Input/ Output module. Therefore, there should have a way within the computer to communicate with the each other modules or we can say that there must be some paths to connect the modules. The whole collection of the connecting paths between the modules is known as **Interconnection Structure**.

The following types of transfers are necessary to support by an **Interconnection Structure**-

- From Memory to CPU: The CPU should be able to read the instructions or data from the memory.
- From CPU to Memory: The CPU must be able to write into the memory.

- From I/O to CPU: The CPU must be able to read from an I/O device through I/O module.
- From CPU to I/O: The CPU has to be able to send data to the I/O devices, by writing into the I/O module.
- I/O to or from Memory: Without involving the CPU, the Memory and I/O should be able to exchange the information directly in between them.

Now, one question may arise that what is a Bus? Well, Bus is a communication pathway connecting two or more modules. The information transmission in Bus is broadcast in nature and a Bus consists of multiple communication lines.

---

#### 4.4.1 SYSTEM BUS

---

The Bus which is connecting the major three components of a computer (CPU, Memory and I/O), is called System Bus. The System Bus generally consists of 50 to 100 separate lines. The System Bus is divided into three categories based on their functionalities as follows-

- **Address Lines:** These lines are used to specify the source/ destination of the data transfer. Address lines width determine the maximum possible memory capacity of the system.
- **Data Lines:** These are the lines to carry the data from one module to another module of the system. Data lines width determine the overall system performance.
- **Control Lines:** The Control lines are used to control the access to and use of data and address lines. Some of the control lines are- Memory Write, Memory Read, I/O Write, I/O Read, Bus request, Bus grant, Interrupt request, Interrupt ACK, Clock, etc.

---

#### 4.4.2 BUS DESIGN ELEMENTS

---

Some of the basic design elements that can differentiate buses areas follows-

- **Type:** Bus can be divided into two types- Dedicated and Multiplexed. A Dedicated Bus line is permanently assigned either to one function or to a physical subset of computer components.

But, the address and data information can be transmitted over the same lines by introducing an Address Valid control signal for time multiplexing purpose. This method is called Multiplexed Bus.

- **Method of Arbitration:** The device that is allowed to transfer the data at a given time is called the Bus Master and the process of selecting the next Bus Master is called Arbitration. There are two types of Arbitration- Centralized and Distributed Arbitration.

In Centralized Arbitration, one single hardware device is responsible for selecting the Bus Masters at a given time. In Distributed Arbitration, there is no central device responsible for arbitration, but each module contains access control logic and modules act together to share the Bus.

- **Timing:** Timing refers to the way in which events are coordinated on the Bus. Two types of Timing exist- Synchronous Timing and Asynchronous Timing.

In Synchronous Timing, the occurrence of event on the Bus is determined by a clock. Again, in Asynchronous Timing, the occurrence of one event on a Bus, follows and depends on the occurrence of a previous event.

- **Bus Width:** Bus width means the number of lines available in the Bus. So, if the number of data line is more, then more number of bits can be transferred from one module to another one and which leads to the better system performance.

Again, if the number of address lines is more, then more number of memory locations can be referenced and which leads to better system capacity.

---

## 4.5 INSTRUCTION FORMAT

---

An Instruction is composed of Operation Code (OPCODE) and Operand. The first part of the instruction specifies the task to be performed called OPCODE and the second part of the instruction is data to be operated on, and it is called Operand. Now, operand of the instruction can be in various forms such that 8-bit or 16-bit data, 8-bit or 16-bit address, register or memory address etc. So, instruction format may be different in different instructions (An instruction format is the layout of the bits of an instruction, in terms of its constituent fields). Therefore, generally an instruction is of the following form-



Generally, based on the address field, four common instruction formats are available and they are-

- **Zero-Address Instructions**



For example,  
 PUSH A    i.e., insert the content of A into the stack

- **One-Address Instructions**



For example,  
 ADD B    i.e. Accumulator  $\leftarrow$  Accumulator + B

- **Two-Address Instructions**



For example,  
 ADD A, B    i.e.     $A \leftarrow A+B$

- **Three-Address Instructions**



For example,  
 ADD A, B, C    i.e.     $A \leftarrow B+C$

Some of the design points for the Instruction Format-

**Instruction Length:** Question is that what should be the length of the instruction format? Obviously, it depends on the memory size, memory organization, bus structure, processor complexity and processor speed. For making more richness and flexible, assembly language programmer wants more opcodes, more operands, and more addressing modes, because more opcodes and more operands provide a easy way to write program and more addressing modes give better flexibility for implementing certain functions. But, problem is with the space, because more space is required for more opcodes, operands and for more addressing modes, and hence we say there is a trade off. Again, one more important point is that whether the instruction length should be equal to the memory transfer length or should be multiple of the transfer length.

**Allocation of Bits:** How to allocate the bits of the instruction for opcode and address field. If more number of bits allocated for the opcode part, then more number of opcodes can have, but reduces the number of bits available for addressing. Hence, here is also trade off in allocation of bits.

Some of the factors for determining number of the bits for the addressing part of the instructions are-

- Number of addressing mode.
- Number of operands.
- Number of register in the register set.
- Address range for referencing memory.

---

## 4.6 ADDRESSING MODES

---

Each instruction of a computer specifies an operation on certain data. There are various ways of specifying address of the data to be operated on. These different ways of specifying data are called the addressing modes. The most common addressing modes are:

- **Immediate addressing mode:** This is the simplest form of addressing. Here, the operand is given in the instruction itself. This mode is used to define a constant or set initial value of variables. The advantage of this mode is that no memory reference other than instruction fetch is required to obtain operand. The disadvantage is that the size of the number is limited to the size of the address field, which is small compared to word length in most instruction sets.
- **Direct addressing mode:** In direct addressing mode,

effective address of the operand is given in the



address field of the instruction. It requires one memory reference to read the operand from the given location and provides only a limited address space. Length of the address field is usually less than the word length.

- **Indirect addressing mode:** In Indirect addressing mode, the address field of the instruction refers to the address of a word in memory, which in turn contains the full length address of the operand. The advantage of this mode is that for the word length of  $N$ , an address space of  $2^N$  can be addressed. The disadvantage is that instruction execution requires three or more memory references is required to fetch the operand.
- **Register addressing mode:** Register addressing mode is similar to direct addressing. The only difference is that the address field of the instruction refers to a register rather than a memory location. So, only 3 or 4 bits are used as address field to reference 8 to 16 general purpose registers. The advantages of register addressing are small address field is needed in the instruction and no time-consuming memory references are need. The disadvantage of register addressing is that the address space is very limited.
- **Register indirect addressing mode:** This mode is similar to indirect addressing. The address field of the instruction refers to a register. The register contains the effective address of the operand. This mode uses one memory reference to obtain the operand. The address space is limited to the width of the registers available to store the effective address. The advantage of this mode is large address space and disadvantage is one extra memory reference is needed.
- **Displacement addressing mode:** The displacement addressing modes is combination of the direct addressing and register indirect addressing. In displacement addressing, the instruction have two address fields, one will have the memory location address and another will have displacement value to be added to get the effective address. There are 3 types of addressing mode in displacement addressing mode. They are :

**Relative addressing-** Here next instruction address is added to the address field to produce the effective address of the operand. Thus the effective address is a displacement relative to the address of the instruction. Here, address field is assumed as 2's complement number.

**Base register addressing-** Here the referenced register contains a main memory address and address field contains a displacement from that address.

**Indexing addressing-** Here, the address field references a main memory address, and the reference register contains a positive displacement from that address.

The advantage of this mode is flexibility in addressing and disadvantage is complexity.

- **Stack addressing mode:** Stack is a linear array of locations referred to as last-in first out queue. The stack is a reserved block of location, appended or deleted only at the top of the stack. Stack pointer is a register which stores the address of top of stack location. This mode of addressing is also known as implicit addressing. Here the effective address is the top of the stack, so no memory reference which is the advantage. Its disadvantage is limited applicability.



## CHECK YOUR PROGRESS

### 2. Fill in the blanks.

- (i) \_\_\_\_\_ is communication path way between connecting two or more modes.
- (ii) The bus connecting CPU, Memory and I/O is known as \_\_\_\_\_.
- (iii) System bus is composed of \_\_\_\_\_, \_\_\_\_\_ and \_\_\_\_\_.
- (iv) Two types of method of arbitration (bus design elements) are \_\_\_\_\_ and \_\_\_\_\_ arbitration.
- (v) An Instruction is composed of \_\_\_\_\_ and \_\_\_\_\_.
- (vi) The addressing modes in which data is directly available as an operand, is known as \_\_\_\_\_.

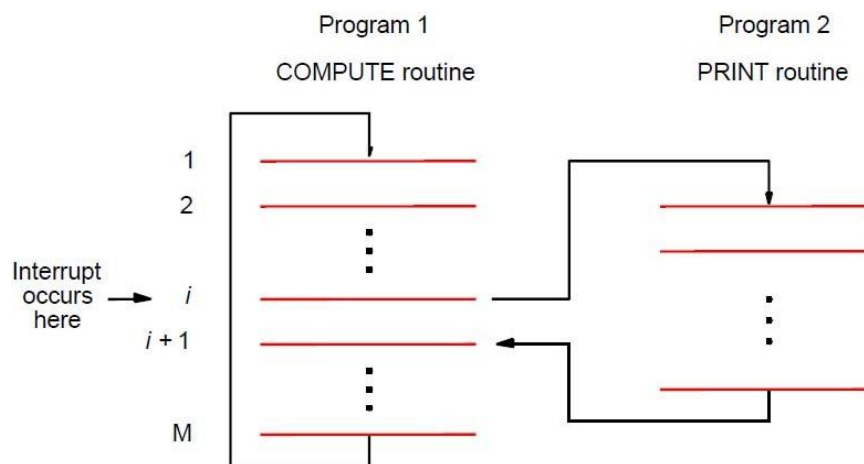
### 3. State whether the following statements are True or False

- (i) The address and data information can be transmitted over the same lines by introducing an Address Valid control signal for time multiplexing purpose. This type is called Multiplexed Bus.
- (ii) In asynchronous Timing, the occurrence of event on the Bus is determined by a clock.
- (iii) More number data lines lead to more system capacity and more number of address lines lead to better system performance.
- (iv) Relative addressing, base register addressing and index addressing is the type of immediate addressing mode.

## 4.7 INTERRUPTS: CONCEPT AND TYPES

The mechanism, by which the execution control of the CPU temporarily gets transferred from one module to another module or it can be said as the CPU temporarily gets jumped from one program routine execution to another one program routine execution, is known as Interrupt. Now, for happening so, there must have to be one input event or signal that will trigger the CPU for getting its control transferred, and that input event is known as Interrupt Request Signal and the signal issued in response to the interrupt request to let inform the source of the interrupt request that interrupt request is granted, is known as Interrupt Acknowledge Signal. Again, the routine executed in response to an interrupt request is known as Interrupt Service Routine. Interrupt Service Routine or ISR is similar to sub-routine of a program, but may not have any relationship with the program.

Now, let us take an example scenario for transferring the control of the CPU execution from one program segment to another one program segment or routine, through interrupt, and let's have a look to observe that how to make it possible.



**Figure: 4.4 Execution of an Interrupt request**

In the above **figure 4.4**, it has been shown that the interrupt request arrives during execution of the  $i^{\text{th}}$  instruction of the program

1 (i.e. COMPUTE routine) and then the CPU suspends the execution of the current program being executed, i.e. COMPUTE Routine and saves the address of the next instruction i.e.  $(i+1)^{\text{th}}$  instruction (or you can say the current content of the program counter) and saves any other relevant information to the processor current activity in the temporary storage. Then, the CPU loads the program counter with the base address/ address of the first instruction of the interrupt service routine, i.e. PRINT Routine and executes the PRINT routine. After finishing the execution of the interrupt service routine, the CPU will load the program counter with address of the  $i+1^{\text{th}}$  instruction of COMPUTE Routine that is already saved in temporary storage and get back for the execution of the COMPUTE routine, starting from the  $i+1^{\text{th}}$

instruction.

**Types of the interrupts:**

Basically, there are two types of the interrupts- Hardware Interrupt and Software Interrupt. Look at the following for some classes of interrupts belonging to either hardware interrupt or software interrupt-

**Program:** Generated by some condition that occurs as a result of an instruction execution, such as arithmetic overflow, division by zero, attempt to execute an illegal machine instruction, or reference outside a user's allowed memory space.

**Timer:** Generated by the timer within the processor. This allows the operating system to perform certain functions on a regular basis.

**I/O:** Generated by an I/O controller, to signal normal completion of an operation or to signal a variety of error conditions.

**Hardware Failure:** Generated by a failure such as power failure or memory parity error.

**Handling Multiple Interrupts:**

At this point, we are very much familiar with what an interrupt is, what an interrupt request signal is, what an interrupt acknowledgement signal is, what an interrupt service routine is, types of interrupt and how to handle an interrupt. So, it is the time to handle multiple interrupt as it is possible to occur at the same time, because it is possible for occurring other interrupts while one interrupt service routine is being processed. So, there should have a module that can handle the interrupt requests from various devices and allow one by one to the processor and is known as Interrupt Controller. Basically, there are two methods for handling multiple interrupts and they are as follows-

- **Disabling the interrupts:** Here, question comes to mind that what a disabled interrupt is. A disabled interrupt means that the CPU can and will ignore that interrupt request signal. If an interrupt occurs during this time, it generally remains pending and will be checked by the CPU after the CPU has enabled the interrupts. So, when a user program is executing and interrupt occurs, interrupts are disabled immediately, and after the completion of the interrupt service routine, interrupts are enabled before resuming the user program, and the processor checks to see if additional interrupts have occurred.

The main disadvantage of this approach is that it does not take into account time-critical needs.

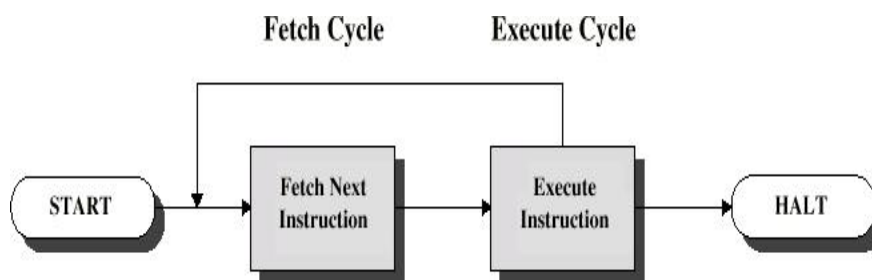
- **Defining priorities to the interrupts:** Here, question comes to mind that what a priority interrupts is. A priority interrupt is an interrupt that establishes a priority over the various sources to determine which condition is to be serviced first when two or more requests arrive simultaneously.

In this method, priorities will be defined for the interrupts and will be allowed a higher priority interrupt to cause a lower priority interrupt service routine to be itself interrupted. When the interrupt service routine for the higher priority interrupt will be completed, then the interrupt service routine of the lower priority will be resumed first and finally control comes to the user program. So, it's a nested kind of nature of the transfer of the control of the CPU.

## 4.8 INSTRUCTION EXECUTION CYCLE AND INTERRUPT

We know that the computer is for executing the instructions of the program stored in the memory and the processing required for executing a single instruction is known as Instruction Cycle and one important thing is that one instruction cycle consists of two sub-cycles or steps-

1. Read the instruction from the memory to Instruction register, called Fetch cycle.
2. Execute the instruction from the instruction register, called Execute Cycle.



**Figure-4.5 An instruction cycle**

In the fetch cycle, CPU loads the instruction in the instruction register from the memory location address which is holding by the program counter and then program counter will be incremented by one to hold the address of the next instruction to be executed.

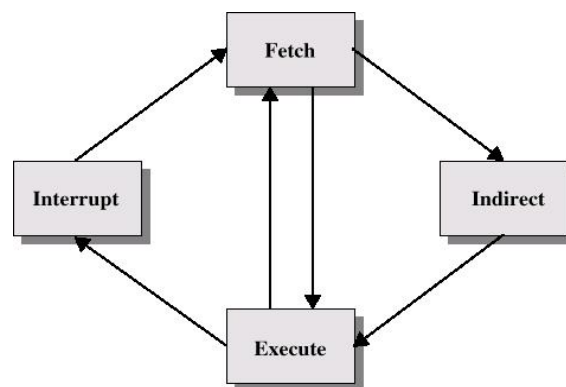
In the execute cycle, interprets the opcode of the instruction and perform the indicated operation.

So, the basic instruction cycle states are as follow-

1. Determine the address of the next instruction.
2. Read the instruction from the memory to instruction register.
3. Interprets the instructions for the operation and about the operand.
4. If operation involves a reference to an operand in memory or in I/O, then calculate the address of the operand and read the operand from memory or I/O.
5. Perform the operation indicated in the instruction.
6. Store the result back into the memory or out through the I/O

So, the important thing is that one more sub-cycle may be introduced within the execute cycle, called indirect cycle. Because, after an instruction is fetch, it is examined to determine if any indirect addressing is involved and if so, the required operand is fetch using indirect addressing. Therefore, if the execution of an instruction involves one or more operands in memory, each of which requires a memory access and it is known as indirect cycle.

Now, what will happen if we consider the occurrence of interrupt in the execute cycle. Suppose, an interrupt occurs when the CPU is executing an instruction, then the CPU will not transferred its control to the interrupt service routine without completing the execution of current instruction. So finally, the instruction cycle can be represented as follows-



**Figure- 4.6 A complete instruction cycle**





## CHECK YOUR PROGRESS

4. Fill the blanks-

- (i) The routine executed in response to an interrupt request is known as\_\_\_\_\_.
- (ii) Methods for handling multiple interrupts are \_\_\_\_\_ and\_\_\_\_\_.
- (iii) The module that can handle the interrupt requests from various devices and allow one by one to the processor and is known as\_\_\_\_\_.
- (iv) The processing required for executing a single instruction is known as\_\_\_\_\_.
- (v) Reading the instruction from the memory to Instruction register, called\_\_\_\_\_, and executing the instruction from the instruction register, called\_\_\_\_\_.

---

### 4.9 HARDWIRED AND MICRO PROGRAMMED CONTROL

---

We know that the execution of a program is nothing but the execution of a sequence of instruction cycles, with one machine instruction per cycle. Again, each instruction cycle is made up of some small cycles and those are fetch cycle, indirect cycle, execute cycle and interrupt cycle, with only fetch and execute cycles always occurring. Now, each of the smaller cycles i.e. fetch cycle, indirect cycle, execute cycle and interrupt cycle, involves a series of steps, each of which involves the processor registers and these steps are called Micro-Operations. Therefore, a micro-operation is an elementary CPU operation, performed during one clock pulse and an instruction consists of a sequence of micro-operations.

Now, we will try to see the instruction cycle as a sequence of Micro-Operations. For that, first we have to take the fetch cycle, a sub-cycle of the instruction cycle. In the fetch cycle, the Micro-Operations are as follows-

$\mu\text{Op1}$ - Move the content of Program Counter to MemoryAddress Register.  
i.e.  $\text{MAR} \leftarrow (\text{PC})$

$\mu\text{Op2}$ - Move the content of memory location specified by Memory Address register to Memory Buffer Register.  
i.e.,  $\text{MBR} \leftarrow \text{Memory}$

$\mu\text{Op3}$ - Increment the Program Counter by the length of instruction.  
i.e.,  $\text{PC} \leftarrow \text{PC} + I$ , where  $I$  is the instruction size.

$\mu\text{Op4}$ - Move the content of the Memory Buffer Register to the Instruction Register.  
i.e.,  $\text{IR} \leftarrow (\text{MBR})$

Therefore, it is found that four Micro-Operations are needed for the fetch sub-cycle. Here,  $\mu\text{Op1}$  will be done in time  $T_1$ ,  $\mu\text{Op2}$  and  $\mu\text{Op3}$  will be done in time  $T_2$  and  $\mu\text{Op4}$  will be done in  $T_3$ . Or,  $\mu\text{Op1}$  will be done in time  $T_1$ ,  $\mu\text{Op2}$  will be done in time  $T_2$  and  $\mu\text{Op3}$  and  $\mu\text{Op4}$  will be done in  $T_3$ .

<p>T1: <math>\text{MAR} \leftarrow (\text{PC})</math></p> <p>T2: <math>\text{MBR} \leftarrow \text{Memory}</math> <math>\text{PC} \leftarrow (\text{PC}) + I</math></p> <p>T3: <math>\text{IR} \leftarrow (\text{MBR})</math></p>	Or,	<p>T1: <math>\text{MAR} \leftarrow (\text{PC})</math></p> <p>T2: <math>\text{MBR} \leftarrow \text{Memory}</math></p> <p>T3: <math>\text{PC} \leftarrow (\text{PC}) + I</math> <math>\text{IR} \leftarrow (\text{MBR})</math></p>
---	-----	---

Next, Micro-Operations for the Indirect Cycle are as follows-

T1:  $\text{MAR} \leftarrow (\text{IR (Address)})$  // Move the address field of instruction to MAR

T2:  $\text{MBR} \leftarrow \text{Memory}$

T3:  $\text{IR (Address)} \leftarrow (\text{MBR (Address)})$  // Address field of IR is updated from the MBR

Next, Micro-Operations for the Interrupt Cycle are as follows-

T1:  $\text{MBR} \leftarrow (\text{PC})$

T2:  $\text{MAR} \leftarrow \text{Save\_address}$

$\text{PC} \leftarrow \text{Routine\_Address}$

T3:    Memory  $\leftarrow$  (MBR)

In the above interrupt Cycle, first the content of PC is moved to the MBR, so that it can be saved, as it will be needed after returning from the interrupt service routine. Then MAR is loaded with the address of the location where PC content is to be saved and PC will be loaded with the base address of the interrupt service routine. Finally, store the content of MBR into the memory as MBR is holding the old value of the PC.

Finally, for gaining the knowledge of Micro-Operations for the execute cycle, let us take an example of an ADD instruction for adding the content of location X with register R1, as follows-

ADD R1, X

So, for the execution of the above instruction, the following Micro-Operations may occurs-

- T1:  $MAR \leftarrow (IR \text{ (Address)})$
- T2:  $MBR \leftarrow \text{Memory}$
- T3:  $R1 \leftarrow (R1) + (MBR)$   
//add the content of R1 with MBR content.

At this point we are familiar with Micro-Operations and the two basic tasks of the control unit as follows-

- **Sequencing:** The control unit causes the processor to step through a series of Micro-Operation in a proper sequence, based on the program being executed.
- **Execution:** The control unit causes each Micro-Operation to be performed.

Now, come to the topic Control Unit Implementation. There are several implementation techniques and most of them fall into the following two categories-

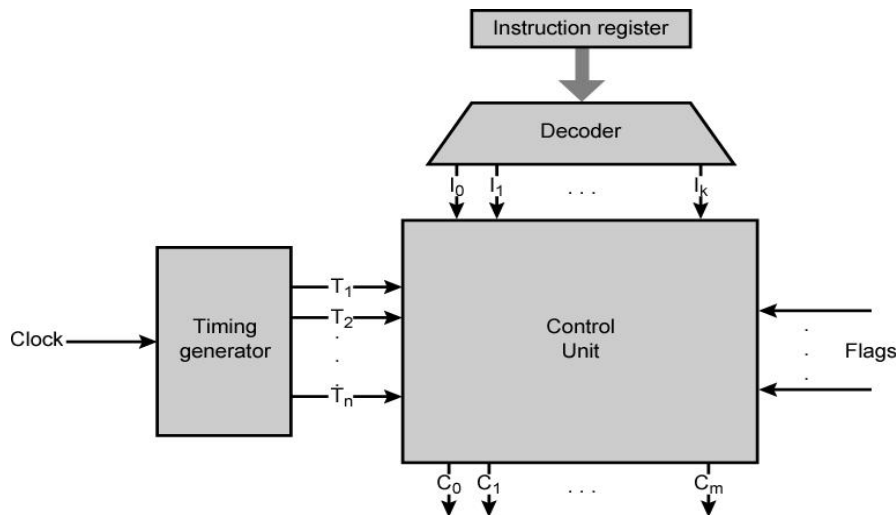
- Hardwired Control
- Micro programmed Control

---

### 4.9.1 HARDWIRED CONTROL

---

In this technique, the control unit is a combinational circuit, where the input logic signals are transformed into a set of output logic signals.

**Figure- 4.7**

Control unit takes instruction register, the clock, flags and control bus signals as input. Actually control unit does the use of opcode and will perform different actions for different instructions. Here, as shown in the **figure-4.7**, the opcode of the instructions available in the Instruction Register will be the input to the decoder and will be decoded by the decoder. So, the decoded output will be the input to the control unit and hence there would be a unique logic input for each opcode.

The clock is also input to the control unit that issues a repetitive sequence of pulses for measuring the duration of micro-operations. The control unit produces outputs which are control signals at different time units within a single instruction cycle that drives various components in the computer. These control signals are the sequence of events to be executed under the control of a system clock.

---

### 4.9.2 MICRO PROGRAMMED CONTROL

---

This technique does not use the interconnection of the basic logic elements to implement a control unit, because there should have a logic for sequencing through micro operations, for executing micro operations, for interpreting opcodes and for making decisions based on the ALU flags, but it is not an easy task. So, micro programmed control technique uses a different way based on micro programming language.

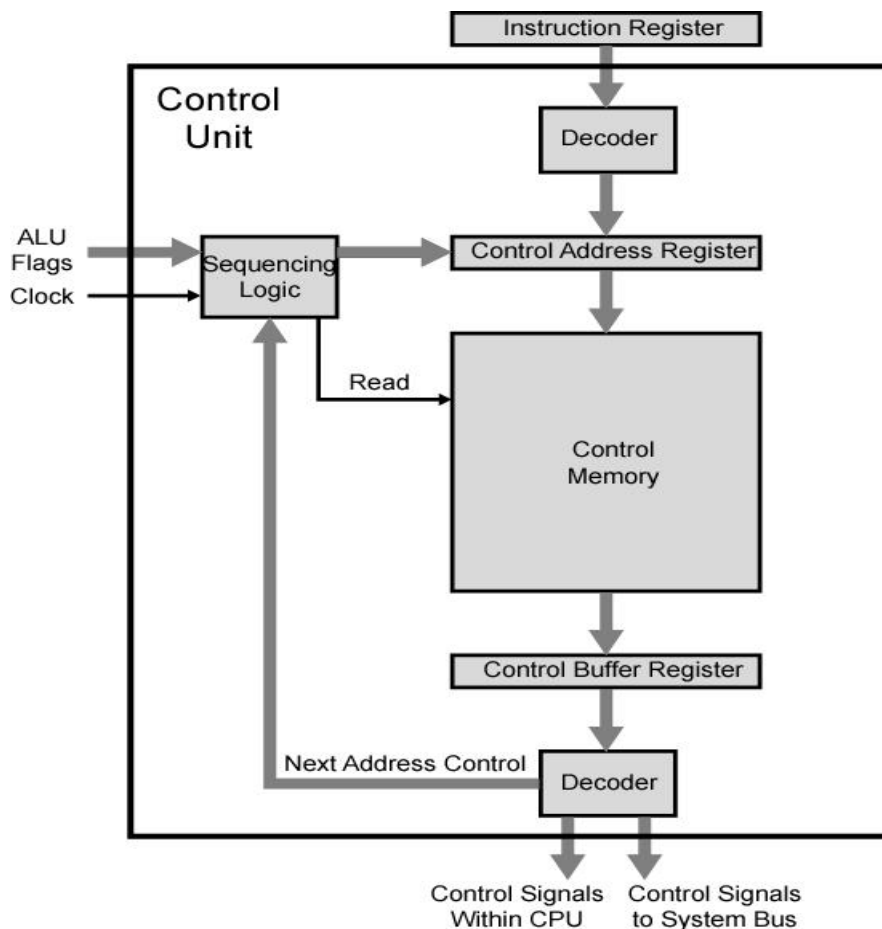
We know that for any micro operation (termed as micro instruction), each control line output from the control unit is either 1 or 0. So, it is possible to construct a control word in which each bit represents one control line. Now, suppose we put together a sequence of control words to represent the sequence of micro operations performed by the control unit. Then, place the control

words in a memory (known as control memory), with each word

having a unique address. Now, add an address field to each control word, indicating the location of the next word to be executed if a certain condition is true. For specifying the condition, add some other bits in each control word, and add one bit for each Internal CPU control signals, one bit for each system bus control signals. The representation of the control word after adding this address field and some other bits as mentioned is known as Horizontal Microinstruction and it will be executed in the following fashion-

1. For executing this microinstruction, turn on all the control lines indicated by a 1 bit; and leave all other control lines off indicated by a 0 bit. The resulting control signals will cause one or more micro-operations to be performed.
2. If the condition indicated by the condition bits is false, then execute the next microinstruction in sequence, else next instruction to be executed is indicated in the address field.

Now, let us examine the Functioning of Micro programmedControl Unit in the following **figure- 4.8**.



**Figure- 4.8**

In the figure, the set of micro instructions is stored in the Control Memory. The Control Address Register is for storing the address of the control memory, generated by the micro program

sequencer,



or for storing the address of the next micro instruction to be read. Then Control Buffer Register is for holding the micro instruction transferred from the control memory after reading it.

Now, these are the steps in the following that happens during one clock pulse, as the functioning of micro programmed control unit-

1. For executing an instruction, the sequencing logic issues a READ control signal to the control memory.
2. The micro instruction or the control word, whose address is specified by the control address register, is then transferred to the control buffer register.
3. The content of the control buffer register generates control signals and next address information for the sequencing logic unit.
4. The sequencing logic unit loads a new address into the control address register based on the next control word address field from the control buffer register and the ALU flags. The new address may be next instruction address or may be a jump to a new routine based on a jump micro instruction or may be jump to a machine instruction routine.

---

### 4.9.3 HARDWIRED vs. MICRO PROGRAMMED

---

Both the techniques are having advantages and disadvantages over each other.

1. In micro programmed scheme, implementing control operation is through micro programmed organization where the control unit is implemented through programming.

But, in hardwired scheme, implementing the control operation is through sequential circuits.

2. In micro programmed, speed of operation is low as it involves memory access

But, in hardwired, speed of operation is high.

3. In micro programmed, changes in the control behavior are easy by modifying the micro instructions in control memory.

But, in hardwired scheme, it is possible if the entire unit gets redesigned.

---

## 4.10 INTRODUCTION TO RISC AND CISC

---

CISC stands for Complex Instruction Set Computer. In Complex Instruction Set Computer, a single instruction can execute several low level operations (such as a load from memory, an arithmetic operation, and a memory store) and/ or capable of multi-step operations or addressing modes within single instructions.

The design constraints that lead to the development of CISC give CISC instructions set some common characteristics:

- A 2-operand format, where instructions have a source and a destination i.e., register to register, registers to memory, and memory to register commands. It also provides multiple addressing modes for memory, including specialized modes for indexing through arrays.
- Variable length instructions where the length often varies according to the addressing modes.
- Instructions which need multiple clock cycles to execute.
- Complex instruction decoding logic, driven by the need of a single instruction to support multiple addressing modes.
- A small number of general purpose registers and several special purpose registers.
- A “condition code” register which is set as a side effect of most instructions. This register reflects whether the result of the last operation is less than, equal to, or greater than zero and records if certain error condition occurs.

Let us have an example of a specific instruction called MULT. When executed this instruction, it loads the two values (or operand) into separate registers and multiplies the operand in the execution unit, and then stores the product in the appropriate register. Thus, the entire task of multiplying two numbers completed with one instruction: **MULT 2:3 5:2**

So, MULT is known as complex instruction. It operates directly on the computer's memory banks and does require the programmer to explicitly call any loading or storing functions. It closely resembled a command in a higher level language. For instance, if we let “a” represent the value of 2:3 and “b” represents the value of 5:2, then this command is identical to the C statement “a = a \* b”.

The primary advantages of this approach is are-

- Compiler has to do very little work to translate a high-level language statement into assembly.

- As so many low level instructions are embedded into a single instruction, this approach needs little memory for storing the instructions.

RISC stands for Reduced Instruction Set Computer. RISC architecture based computer only use simple instructions that can be executed within one clock cycle. Thus the “MULT” command described above could be divided into three separate command-

1. LOAD- means moves data from memory bank to a register.
2. PROD- means finds the product of the two operands located within the registers.
3. STORE- means moves data from a register to the memory bank.

In order to make the exact series of the steps mentioned above, the programmer needs to code four lines of assembly code as follows-

LOAD A, 2:3

LOAD B, 5:2

PROD A, B

STORE 2:3, A

The following are the differences between CISC and RISC Architecture:

1. CISC architecture emphasis on hardware, but RISC architecture emphasis on software.
2. CISC architecture includes multi-clock complex instructions, but RISC architecture uses single clock reduced instruction only.
3. CISC architecture uses variable length instructions, but RISC architecture uses fixed length instructions.
4. CISC architecture uses many addressing modes, but RISC architecture few addressing modes.
5. In CISC architecture, complexity is in compiler, but in RISC architecture, complexity in micro-code.
6. In CISC architecture, many instructions can access memory, but in RISC architecture only LOAD/ STORE instructions can access memory.

As we know that following performance equation is commonly used for expressing a computer's performance ability-

Time per program execution is-

$$(\text{Time/ Cycle}) \times (\text{Cycles/ Instruction}) \times (\text{Instructions/ Program})$$

The CISC approach attempts to minimize the number of instructions per program, sacrificing the number of cycles per instructions. But RISC approach minimizes number of instructions per program, by sacrificing the number of cycles per instructions.



### CHECK YOUR PROGRESS - 4

5. Fill in the blanks.

- (i) A \_\_\_\_\_ is an elementary CPU operation, performed during one clock pulse
- (ii) In hardwired scheme, implementing the control operation is through \_\_\_\_\_.
- (iii) RISC stands for \_\_\_\_\_ and CISC stands for \_\_\_\_\_.
- (iv) RISC approach minimizes the \_\_\_\_\_, by sacrificing the \_\_\_\_\_.

6. State whether the following statements are True or False

- (i)  $\text{MAR} \leftarrow (\text{PC})$ , means move the content of memory location specified by Memory Address register to Memory Buffer Register.
- (ii)  $\text{MAR} \leftarrow (\text{IR (Address)})$ , means move the address field of instruction to memory address register.
- (iii) In micro programmed implementation of control unit, speed of operation is high, but in hardwired implementation, speed of operation is low.
- (iv) CISC architecture emphasis on software, but RISC architecture emphasis on hardware.
- (v) RISC approach minimizes the number of number of instructions per program, by sacrificing the number of cycles per instructions.

---

## 4.11 LET US SUM UP

---

- CPU stands for Central Processing Unit. The building blocks of the CPU are **ALU**, **Control Unit** and **Registers**.
- **ALU** is a digital circuit for performing the arithmetic and logical operations. Control Unit basically performs **sequencing** and **execution**.
- Registers are divided into two categories- **User Visible Registers** and **Control and Status Registers**.
- The Bus which is connecting the major three components of a computer (CPU, Memory and I/O), is called **System Bus**
- An **instruction format** is the layout of the bits of an instruction, in terms of its constituent fields.
- The mechanism, by which the execution control of the CPU temporarily gets jumped from one program routine execution to another one program routine execution, is known as **Interrupt**.
- The processing required for executing a single instruction is known as **Instruction Cycle**.
- Two types of implementation for the control unit available- **Hardwired** and **Micro programmed**.
- **RISC** stands for Reduced Instruction Set Computer and **CISC** stands for Complex Instruction Set Computer.



## 4.12 ANSWERS TO CHECK YOUR PROGRESS

---

1. (i) Central Processing Unit  
(ii) ALU, Control Unit, Registers  
(iii) Arithmetic Logic Unit  
(iv) Sequencing, Execution  
(v) clock cycle time.  
(vi) Program Counter  
(vii) Segment Pointer register
2. (i) Bus  
(ii) System Bus  
(iii) address lines, data lines, control lines  
(iv) centralized, distributed  
(v) opcode, operand  
(vi) Intermediate mode

3. (i) true      (ii) false      (iii) false      (iv) false

4.
  - (i) interrupt service routine
  - (ii) disabling interrupts, defining priorities to the interrupts
  - (iii) interrupt controller
  - (iv) instruction cycle
  - (v) fetch cycle, execute cycle
5.
  - (i) micro operation
  - (ii) sequential circuit
  - (iii) Reduced Instruction Set Computer, Complex InstructionSet Computer
  - (iv) number of instructions per program, number of cycles perinstructions
6. (i) false      (ii) true      (iii) false      (iv) false      (v) true



### 4.13 FURTHER READINGS

---

- William Stallings: Computer Organization & Architecture, Designing for Performance. Pearson, Prentice Hall.
- C. Hamacher, Z. Vranesic, S. Zaky: ComputerOrganization. Mc Graw Hill.
- Sujatha. K & Jasmine Mary. S: Computer Organization and Architecture, lecture notes.
- B. Ram: Microprocessors and Micro Computers. Fifth edition, Dhanpat Rai Publication.



### 4.14 PROBABLE QUESTIONS

---

1. What are the building blocks of Central Processing Unit? Briefly discuss the functions of each unit.
2. Discuss about the registers available in the CPU.
3. What is Bus? What is System Bus? What are the design elements of Bus and explain each design element of Bus.

4. Discuss about different forms of the Instruction format.



5. What do you mean by Addressing Mode? Discuss different types of addressing modes including the advantages and disadvantages of each mode.
6. Write briefly about interrupt? How to handle multiple interrupts.
7. Discuss the Instruction Execution Cycle without interrupt and with interrupt.
8. Discuss about the Hardwired implementation of the Control Unit.
9. Discuss about the Micro Programmed Controlled implementation of the Control unit.
10. Write the differences between Hardwired and Micro Programmed implementation of Control Unit.
11. Write the differences between the RISC architecture and CISC architecture.
12. What is the Windows 7 Start menu? Explain its parts.
13. How do we restart, log-off and lock a computer.
14. How can we customize the start menu?
15. Describe the different parts of the Window Explorer.

\*\*\*\*\*



# UNIT- 5 MULTI-PROCESSOR ORGANIZATION

## UNIT STRUCTURE

- 5.1 Learning Objective
- 5.2 Introduction
- 5.3 Classification of Parallel Computation
  - 5.3.1 The Single-Instruction-Single-Data (SISD)
  - 5.3.2 The Single-Instruction-Multiple-Data (SIMD)
  - 5.3.3 Multiple-Instruction-Single-Data (MISD)
  - 5.3.4 Multiple-Instruction-Multiple-Data (MIMD)
- 5.4 Single-Instruction-Multiple-Data (SIMD)
- 5.5 Multiple-Instruction Multiple-Data (MIMD)
  - 5.5.1 Shared Memory Organization
  - 5.5.2 Message Passing Organization
- 5.6 Analysis and Performance of Multiprocessor Architecture
- 5.7 Future Direction of Multiprocessor Architecture
- 5.8 Let Us Sum Up
- 5.9 Answer to Check Your Progress
- 5.10 Further Reading
- 5.11 Model Questions

---

## 5.1 LEARNING OBJECTIVES

---

After going through this unit, you will able to

- know the multiprocessor motivation
- describe the classification of parallel computation
- explain the multiprocessor organizations
- know the shared memory multiprocessors
- illustrate the message-passing multiprocessors
- explain the performance of multiprocessor architecture
- know the future direction of multiprocessor architecture

---

## 5.2 INTRODUCTION

---

Today computers are not used for solving only scientific and military applications but are used in all other areas such as banking, share markets, Universities, reservation such as airline reservations and train reservations. Although kind of data to be processed in each application is different but there is one common factor among the data processed in all these applications. The common factor is data to be processed is huge and in most of the applications there is a symmetry among the data in each applications. For example airline reservation has to process the data that has information about passengers, flight and flight schedules. Thus instead of using high end server like mainframe to process these related large volume of data, we can use multiple programs running on different computers which are interconnected to obtain the same result. This and many more detailed observations listed below give insight into benefit of executing the programs in parallel.

- Many scientific applications such as Modeling of weather patterns, astrophysics, chemical reactions, ocean currents, etc take too long to run on a single processor machine. Many of these are parallel applications which largely consist of loops which operate on independent data. Such applications can make efficient use of a multiprocessor machine with each loop iteration running on a different processor and operating on independent data.
- Many multi-user environments require more compute power than available from a single processor machine. Example of this category of applications are Airline reservation system, department store chain inventory system, file server for a large department, webserver for a major corporation, etc. These consist of largely parallel transactions which operate on independent data. In this unit we will basically concentrate on the multiprocessor organization.

---

## 5.3 CLASSIFICATION OF PARALLEL COMPUTATION

---

---

Michael J. Flynn created one of the earliest classification systems for parallel (and sequential) computers and programs, now known

as Flynn's taxonomy. Flynn classified programs and computers by whether they were operating using a single set or multiple sets of instructions, whether or not those instructions were using a single or multiple sets of data. In this system classifications are based upon the number of concurrent instructions and data streams present in the computer architecture. These classification are

- The single-instruction-single-data (SISD)
- The single-instruction-multiple-data (SIMD)
- Multiple-instruction-single-data (MISD)
- Multiple-instruction-multiple-data (MIMD)

Except SISD, other three category of applications such as SIMD, MISD, MIMD are candidate applications for multiprocessor architecture. A typical multiprocessor architecture performing the parallel processing is shown in the following Fig-5.1.

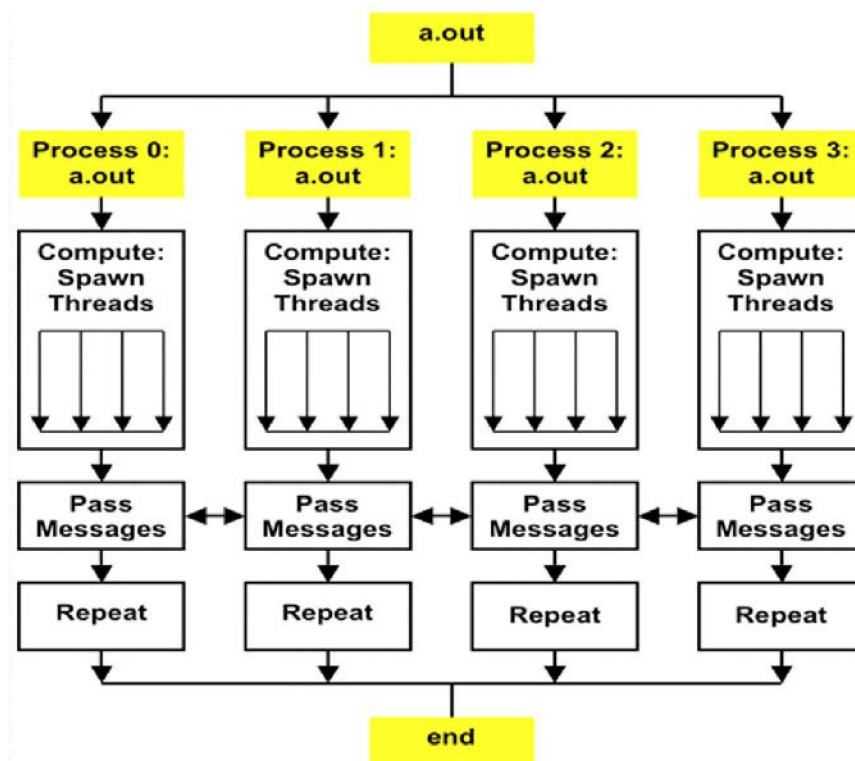


Fig. 5.1 Block diagram for multiprocessor architecture  
Instead of executing whole program on a single machine.

Execution of a program is partitioned into different subprograms

and each of

these subprograms are executed on different machines. During time of execution, programs running on different machines may want to communicate with each other. Finally result from each processor is combined to obtain the desired final result.

Now, we consider in detail the each category of architecture of Flynn's classification.

---

### 5.3.1 The Single-Instruction-Single-Data (SISD)

---

SISD architecture is equivalent to a machine executing program entirely in sequential manner. The program of type calculating a cuberoot of a given number. A single program computes the cuberoot on a given number. A Uniprocessor architecture is best suited for this kind of programs. This is the classic "Von Neumann" architecture dating from 1945, where a control unit coordinates the traditional "Fetch - Execute" cycle, obtaining a single instruction from memory, feeding it to the processor for execution on the data specified within the instruction, then storing the result back into memory. Although modern computers especially desktop do not adhere completely to this arrangement, they are still classed as sequential rather than parallel processors. Modern computers may use separate data and instruction buses and the use pipelining at a low level to increase the speed of execution but still considered executing the program in sequential fashion. Following block diagram Fig-4.2 shows main functional components of the SISD architecture.

---

### 5.3.2 The Single-Instruction-Multiple-Data (SIMD)

---

SIMD architecture is equivalent to a machine doing the same operation repeatedly over a large data set. This is commonly done in signal processing applications. If we extend the previous program to compute cuberoot for a set of given numbers then that program is classified into SIMD. The architecture is also referred to as an "Array processor", comprising a single control unit and several processing elements which all execute the same instruction at the same time on different data. Machines using this architecture are rather specialized compare to "Von Neumann".



These are designed to be

for greater performance in their use when one wants to employ their full parallelism. SIMD perform excellently on digital signal and image processing and on certain types of Monte Carlo simulations. Though limited to a small number of applications such as image processing and the solution of 2D and 3D field problems, the speed up factor is significant in that it is almost directly proportional to the number of processing elements. Fig-5.3 above gives the block diagram for SIMD architecture.

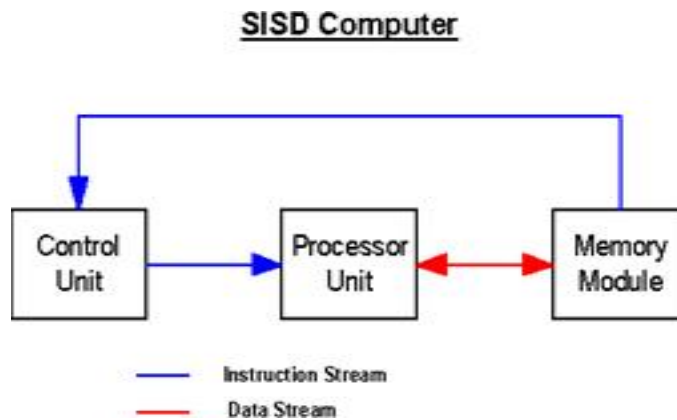


Fig. 5.2 Block diagram of SISD architecture

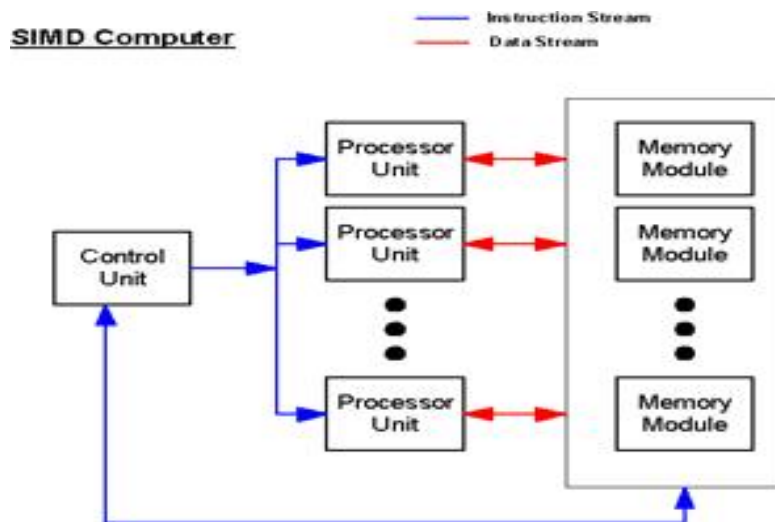


Fig. 5.3 Block diagram of SIMD architecture

### **5.3.3 Multiple-Instruction-Single-Data (MISD)**

---

Conceptually there are a number of processing elements operating different instructions on the same data. Though this type of processor can be conceptually illustrated, such that each processor output becomes the input to the next in a form of pipe, there are very few actual applications currently using this architecture. It has been argued that Systolic Array processing may fit this criteria. Another application such as stream based processing may fit into this class. Fig-5.4 gives the block diagram for MISD architecture.

### **5.3.4 Multiple-Instruction-Multiple-Data (MIMD)**

---

MIMD programs are by far the most common type of parallel programs. If we extend the program of SIMD to compute cube root for different set of numbers, wherein each set contains different data types. One set may contain integer data type and other set may contain floating-point numbers. In this architecture each processor has its own set of instructions which are executed under the control of a Control Unit associated within that processor. Usually each processor will have its own local memory upon which the instructions operate. Thus synchronization can only be achieved by the provision of explicit interprocessor mechanisms, which provide access to an area of shared memory. Fig-5.5 gives the block diagram for MIMD architecture.

Among four classification two classifications SIMD and MIMD are two architectures which are well suited for parallel processing. We discuss in detail these two classifications and finally we discuss related problem that may arise because of parallel processing and away of overcoming these problems.

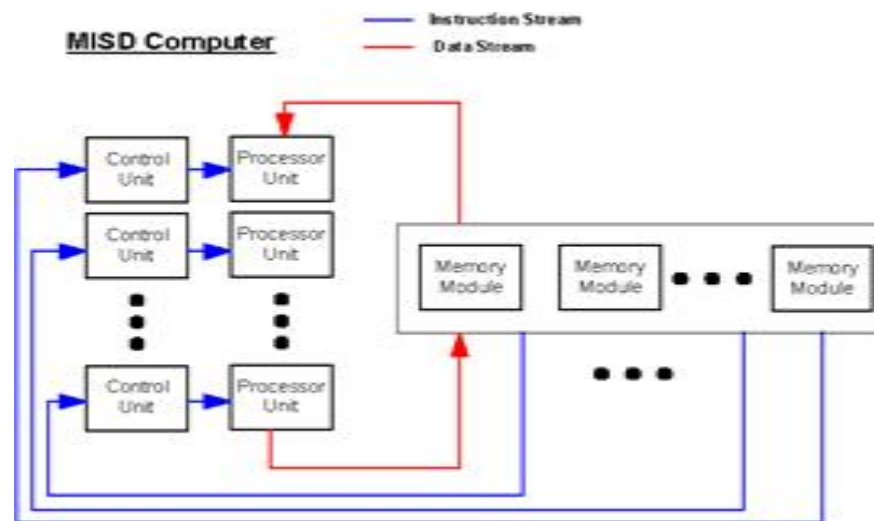


Fig. 5.4 Block diagram of MISD architecture

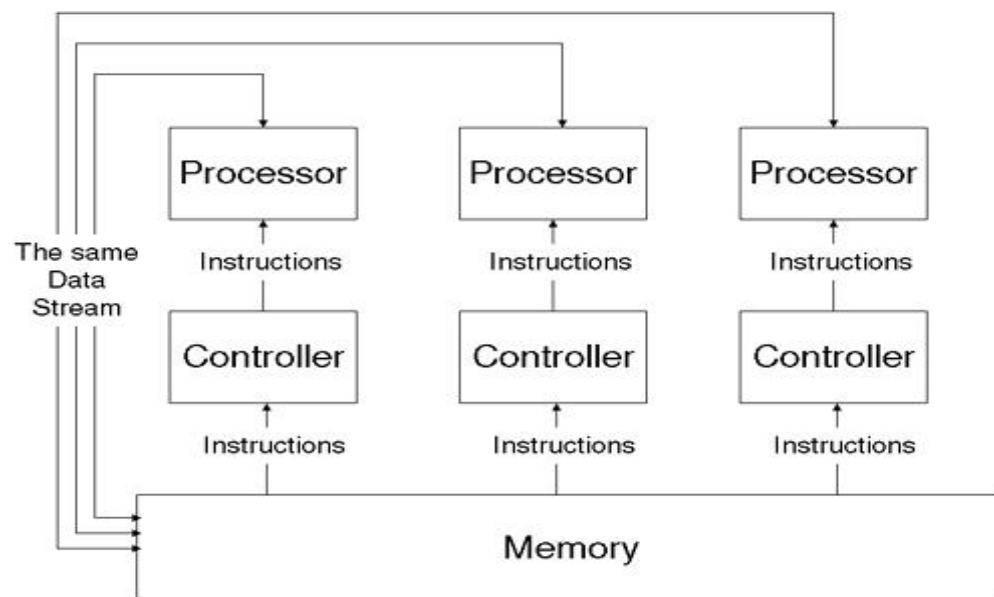


Fig. 5.5 Block diagram of MIMD architecture

## 5.4 SINGLE-INSTRUCTION-MULTIPLE-DATA (SIMD)

In this section, we will explain SIMD class of architecture. It is important to know that SIMD are mostly designed to exploit the inherent parallelism encountered in matrix (array) operations. Array operations are most required operations in applications such as image processing. There are two main SIMD configurations in parallel processing. These two schemes are shown in Fig-5.6.

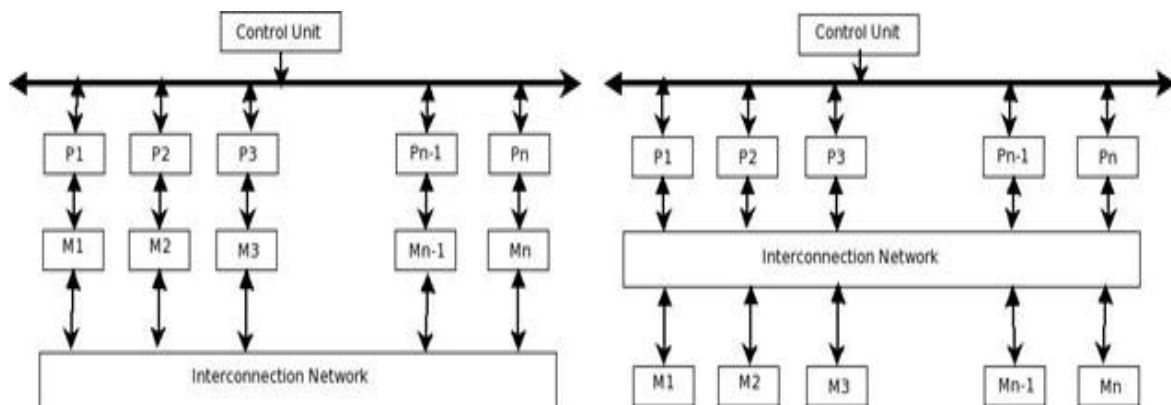


Fig. 5.6 Two SIMD schemes



\* The ILLIAC IV was one of the most infamous super computers ever built

\*\* The Burroughs Scientific Processor (BSP), a high-performance computer system combined parallelism and pipelining, performing memory-to-memory operations

In the first scheme, each processor is independent. Independent means each has its own local memory. When processors want to communicate with each other they done through the interconnection network. If the interconnection network does not provide direct con-nection between a given pair of processors, then this pair can ex- change data via an intermediate processor. Super computer ILLIAC- IV\* used this kind of inter connection scheme. The interconnection network in the ILLIAC-IV allowed each processor to communicate directly with four neighboring processors in an 8 X 8 matrix pattern such that the  $i^{\text{th}}$  processor can communicate directly with the  $(i-1)^{\text{th}}$ ,  $(i+1)^{\text{th}}$ ,  $(i-8)^{\text{th}}$ , and  $(i+8)^{\text{th}}$  processors. This similar to movement of the soldier in the chess. He can move one step above, below or top, two but not move in diagonal.

In the second SIMD scheme, processors and memory modules communicate with each other via the interconnection network.

Two

processors can transfer data between each other via intermediate one or more memory modules or through one or more intermediate processors. We illustrate this with an example. Suppose processor  $i$  is connected to memory modules  $(i-1)$ ,  $i$ , and  $(i+1)$ . In this case, processor 1 can communicate with processor 5 via memory modules 2, 3, and 4 as intermediaries. We explain why memory modules 2, 3, 4 are used. First processors 1 and 3 are connected to memory module

2. Next processor 3 and 4 are connected to memory module 3. Finally processor 4 and five are connected to memory module 4. Now if we trace movement of data, data was first places into memory module 2 by processor 1. Processor 3 transfer from memory module 2 to memory module 3. Then processor 4 translate from memory module 3 to memory module 4. Finally memory module 4 is read from processor 5. The BSP\*\* (Burroughs' Scientific Processor) used the second SIMD scheme. In order to illustrate the effectiveness of SIMD in handling array operations, consider the operations of adding the corresponding elements of two one dimensional arrays A and B and storing the results in a third one-dimensional array C. Assume also that each of the three arrays has N elements.

Assume SIMD scheme 1 is used. The N additions required can be done in one step if the elements of the three arrays are distributed such that M0 contains the elements A (0), B (0), and C (0), M1 contains the elements A (1), B (1), and C (1), . . . , and MN-1 contains the elements A (N - 1), B (N - 1), and C (N - 1). In this case, all processors will execute simultaneously an add instruction of the form  $C = A + B$ . After executing this single step by all processors, the elements of the resultant array C will be stored across the memory modules such that M0 will store C (0), M1 will store C(1), . . . , and MN-1 will store C (N - 1). It is customary to formally represent an SIMD machine in terms of five-tuples (N, C, I, M, F). The meaning of each argument is given below.

1. N is the number of processing elements ( $N = 2^k$ ,  $k \geq 1$ ).
2. C is the set of control instructions used by the control unit, for example, *do, for, step*.
3. I is the set of instructions executed by active processing units.

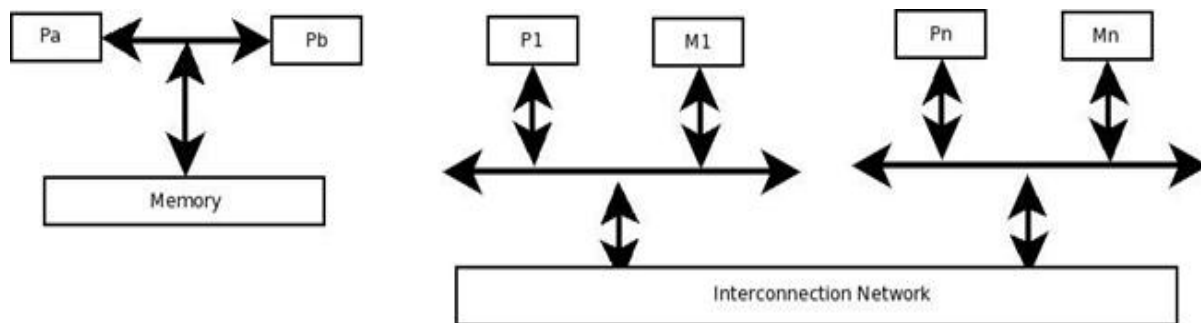
4.  $M$  is the subset of processing elements that are enabled.
5.  $F$  is the set of interconnection functions that determine the



communication links among processing elements.

## 5.5 MULTIPLE-INSTRUCTION MULTIPLE-DATA (MIMD)

Multiple-instruction multiple-data streams (MIMD) parallel architectures has multiple processors and multiple memory modules connected together through interconnection network. We can divide MIMD architecture into two broad categories one shared memory based and another message passing. Fig-7 illustrates the general architecture of these two categories. In shared memory there is central shared memory and processors exchange information through this central shared memory. Processors communicate through a bus and cache memory controller. As multiple servers are accessing the shared memory, shared memory need to be an expensive multiported memories supporting simultaneous reads and writes. Some example architectures that use Shared memory for information exchange



a) MIMD Using shared memory

b) MIMD Using Message Passing.

are Sun Microsystems multiprocessor servers, and Silicon Graphics Inc. multiprocessor servers.

Fig. 5.7 MIMD variants a) Shared memory b) Message Passing

In case of message passing system exchange of information is done by exchange of messages through interconnection network. In message passing system (also referred to as distributed memory) each node combines the local memory and processor. Thus there is no global memory. Because of this it is necessary to move data from one local memory to another by means of message passing. Send/Receive pair of commands are used to

achieve the passing message from one node to another node.  
Usually message passing

system calls are available at application level. Thus message passing mechanism is implemented at the application layer. Programmers wanting to achieve parallel processing using message passing method need to know message-passing paradigm, which involves data copying and dealing with consistency issues. Commercial examples of message passing architectures c. 1990 were the nCUBE, iPSC/ 2, and various Transputer-based systems. Concept of Internet is similar to multiprocessor systems using message passing. In Internet a node is either Internet servers or clients and information is exchanged using messages.

Next question we ask is which among these is better. Distributing memory is one way of efficiently increase the number of processors managed by a parallel and distributed system. Instead if we use centralized memory then by increasing number of processors results in greater conflicts. Thus to scale to larger and larger systems (as measured by the number of processors) the systems had to use distributed memory techniques. These two forces created a conflict: programming in the shared memory model was easier, and designing systems in the message passing model provided scalability.

---

### 5.5.1 Shared Memory Organization

---

In shared memory model all processors communicate with each other by reading and writing locations in a shared memory. The shared memory is made equally accessible by all processors. Apart from centralized shared memory each processor may have registers, buffers, caches, and local memory as additional memory resources. Since centralized resource is shared simultaneously by number of processors many basic issues in the design of shared memory systems have to be taken into consideration. These include access control, synchronization, protection, and security.

#### **Access Control.**

Access control determines which process can access which of possible resources. This list is often maintained in separate table. For every access request issued by the

processors to the shared

memory, access control module make the required check against the contents of the access control table to determine whether to give or not to give(deny) access permission. If there are access attempts to resources, then until the desired access is completed, all disallowed access attempts and illegal processes are blocked. Requests from sharing processes may change the contents of the access control table during execution. Access control mechanism along with synchronization rules guide the sharing of shared memory among multiple processors.

### **Synchronization.**

Synchronization controls and coordinates access of shared resources by multiple processors. Appropriate synchronization ensures that the information flows properly and ensures system functionality. Many synchronization primitives such as semaphores are used along the shared memory access.

### **Protection**

Protection is a system feature that prevents processes from making arbitrary access to resources belonging to other processes. One must not confuse between access control and protection. Access control is something to determine what are you allowed to do. This has to do with the policy making regarding who are all can use which are all resources. Whereas in protection question asked is who are you. Protection is often done by authorization. There are many authorization mechanisms ranging from simpler password based to more powerful cryptographic methods may be used for authorization.

### **Sharing**

Sharing and protection are incompatible; sharing allows access, whereas protection restricts it. Sharing often leads to resource conflicts and synchronization problem. One extreme of sharing is not to share the resources. Another extreme is to share the resources completely. Shared memory when used along with the cache results in greater data inconsistency leading to cache coherence problem. We explain cache coherence problem briefly here. For example when two processors cache the data then

when one processor write-Back the modified data back into memory, the

updated data is not visible to other process. Cache coherence is way of ensuring data consistency in the case of write-backs.

There are many shared memory variants for the multiprocessor architectures. The simplest shared memory system consists of one memory module that can be accessed from two processors. In this requests arrive at the memory module through its two ports. An arbitration unit within the memory module passes requests to a memory controller. If the memory module is not busy and a single request arrives, then the arbitration unit passes that request to the memory controller and the request is granted. During this time module is placed in the busy state . If a new request arrives while the memory is busy , then requesting processor put the request in the queue until the memory becomes free or it may repeat its request sometime later.

Depending on the interconnection network, a shared memory systems are classified as: **Uniform Memory Access (UMA)**, **NonUniform Memory Access (NUMA)**, and **Cache-Only MemoryArchitecture (COMA)**. In the UMA system, a central shared memory is accessible by all processors through an interconnection network

.Therefore, all processors have equal access time to any memory location. The interconnection network used in the UMA can be a single bus, multiple buses, a crossbar, or a multiport memory. Instead of centralized shared memory, in the NUMA system, each processor has part of the shared memory attached. But the memory has a single address space. This can be visualized as single memory is divided into different pages and these pages are distributed among processors. Therefore, any processor could access any memory location directly using its real address. However, the access time to modules depends on the distance of processor with respect to memory module it try accessing . This results in a nonuniform memory access time. COMA is similar to the NUMA wherein each processor has part of the shared memory. However, in this case the shared memory consists of cache memory. In COMA system data is migrated to the processor requesting it. After data has been migrated to the cache, then the

data in cache is used by processor.



---

### 5.5.2 Message Passing Organization

---

Multiprocessor architecture with message passing systems is made up of a set of processors each having their own local memory. Communications in message passing systems are performed via send and receive operations. A node in such a system consists of a processor and its local memory. Nodes are typically able to store messages in buffers (temporary memory locations where messages wait until they can be sent or received), and perform send/ receive operations at the same time as processing. Simultaneous message processing and message processing are handled by the underlying operating system. Each node in this system is interconnected in a many ways ranging from architecture-specific interconnection structures to geographically dispersed networks. As stated earlier, message passing approach is scalable. By scalable, it means that the number of processors can be increased without significant decrease in efficiency of operation. Two important design factors must be considered in designing interconnection networks for message passing systems. These are the link bandwidth and the network latency. The link bandwidth is defined as the number of bits that can be transmitted per unit time (bits/ s). The network latency is defined as the time to complete a message transfer.

An message routing protocol called Wormhole routing is an improvement over traditional store-and-forward routing. In Wormhole routing in order to reduce the size of the required buffers and to decrease the message latency, a packet is divided into smaller units that are called flits (flow control bits). Each flit moves in a pipeline fashion with the header flit moves first followed by remaining flits. When the header flit is blocked due to network congestion, the remaining flits are blocked as well. This not only reduces the unnecessary traffic but also decreases time for transmission of messages.

#### Interconnection Networks

There are many ways of classifying interconnection networks. Classification is often based on mode of operation, control system, switching techniques used and topology used.

According to the mode of operation, they are classified as synchronous versus asynchronous . In synchronous mode of operation, a single global clock is used by all components in the system. Asynchronous mode of operation, on the other hand, does not require a global clock. Handshaking signals are used instead in order to coordinate the operation of asynchronous systems.

According to the control strategy, Interconnection networks can be classified as centralized versus decentralized. In centralized control systems, a single central control unit is used to oversee and control the operation of the components of the system. In decentralized control, the control function is distributed among different components in the system.

Interconnection networks can be classified according to the switching mechanism as circuit versus packet switching networks. In the circuit switching mechanism, a complete path has to be established prior to the start of communication between a source and a destination. The established path will remain in existence during the whole communication period. In a packet switching mechanism, communication between a source and destination takes place via messages that are divided into smaller entities, called packets. On their way to the destination, packets can be sent from one node to another in a store-and-forward manner until they reach their destination. While packet switching tends to use the network resources more efficiently, compared to circuit switching, it suffers from variable packet delays.

Based on topology used for interconnection, interconnection networks are classified as single bus, Crossbar networks, Multistage networks and Hypercube Networks.

### **Single bus**

In this interconnection system a common communication path is shared by all functional units. As each unit try to same same path mechanisms such as bus arbitration are needed when multiple units try accessing the shared bus simultaneously. Further performance of shared bus is limited by number of processors connecting to it. Performance of shared bus decrease according to number of processors connected to it.

## Crossbar networks

In crossbar networks memory units are connected to processor through separate paths as shown in the figure below wherein each processor is connected each memory module through a cross point switch. This enables all processors to send memory requests independently and asynchronously. This kind of interconnection scales well but increases the hardware complexity.

Each crossbar switch in a crossbar network can be set open or closed, providing a point-to-point path between processor and memory module. On each row of a crossbar mesh multiple switches can be connected simultaneously but only one switch in a column is opened at any point of time. This provides a processor to interact with multiple memory modules simultaneously but one memory module can communicate with only one processor at any point of time.

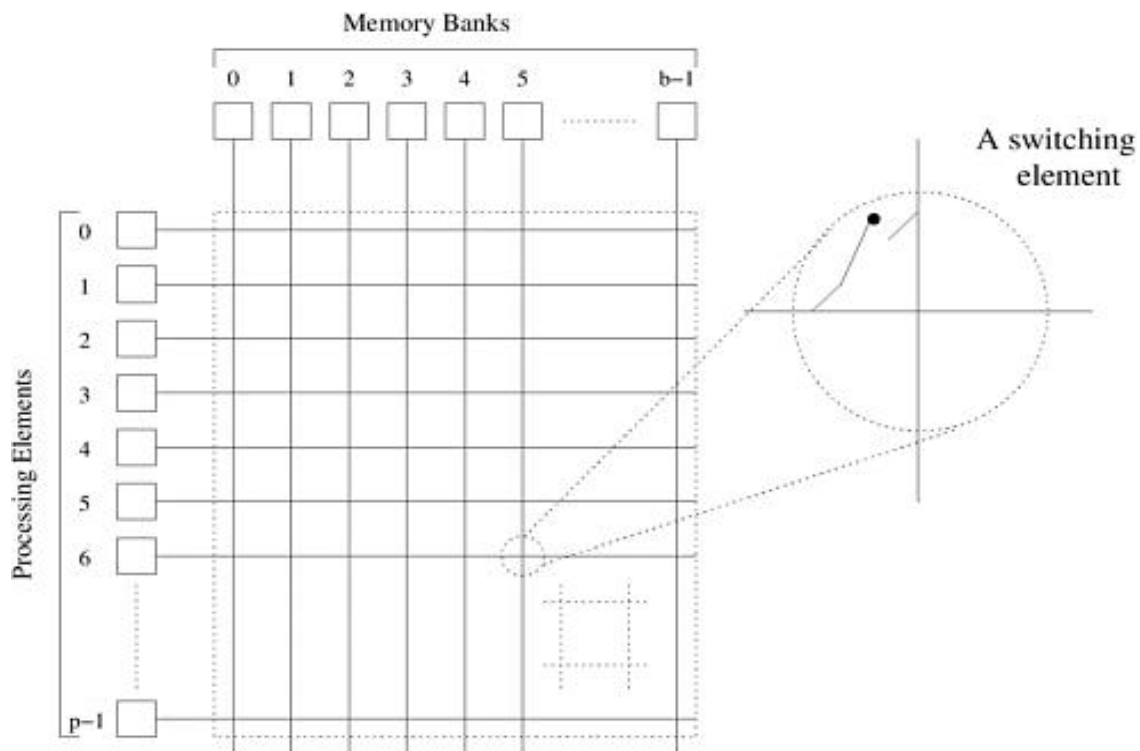


Fig. 5.8 Multiprocessor connected through crossbar switches

### Multistage networks

Multistage Interconnection Networks(MINs) are extension of

crossbar networks connecting processors and memory modules through switching elements. The switching elements themselves are usually connected to each other in stages, hence the name. The main difference between MINs and crossbar networks is number of stages used. In crossbar networks a complete mesh connects processors to memory. This can be thought as single stage. In MINs multiple such stages are used for interconnections. This provides reliability and efficiency by deriving the paths through permutation among different stages. Banyans, Clos, Batcher sorters are examples of this interconnections. Following figure shows one such multistage network.

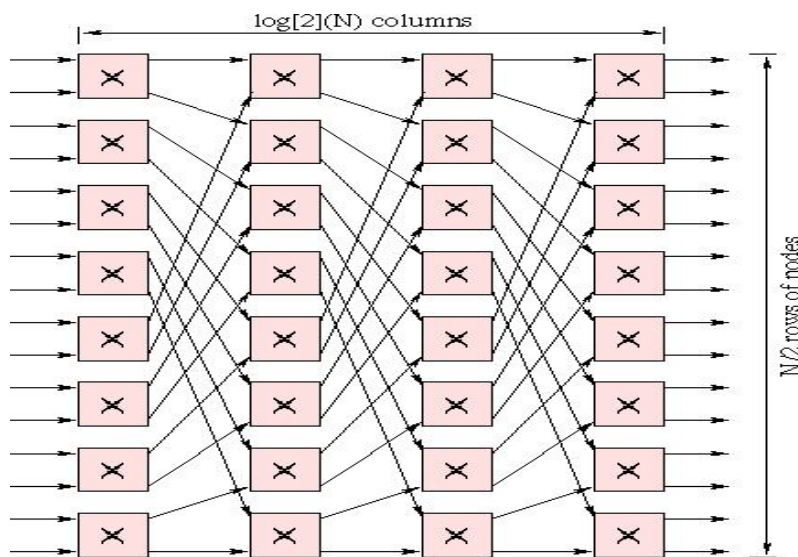


Fig. 5.9 Multiprocessor connected through multistage network

### Hypercube Networks

Hypercube networks are derived from cubic structures. Hypercubes can be thought of as a hierarchical cubic design wherein a higher level is designed by connecting many lower level cubic designs. Usually, coordinates of the cubes are connected to provide multistage connectivity between processors and memory modules. Often, hypercubes are considered to be more reliable interconnections as they provide multiple paths through their different dimensions.

## 5.6 ANALYSIS AND PERFORMANCE OF MULTIPROCESSOR ARCHITECTURE

In this section, we provide some basic ideas about the performance issues in multiprocessors. A fundamental question that is usually asked is how much faster a given problem can be solved using multiprocessors as compared to a single processor? This question can be formulated into the speed-up factor defined below.

$$S(n) = \frac{\text{Execution time using a single processor}}{\text{Execution time using } n \text{ processors}}$$

A related question is that how efficiently each of the  $n$  processors is utilized. The question can be formulated into the efficiency defined below.

$$E(n) = \text{Efficiency}$$

$$E(n) = \frac{S(n)}{n} \times 100\%$$

Where  $s(n)$  is the speedup gained by dividing task into  $n$  subtasks.

In executing tasks (programs) using a multiprocessor, it may be assumed that a given task can be divided into  $n$  equal subtasks each of which can be executed by one processor. Therefore, the expected speed-up will be given by the  $S(n)=n$  while the efficiency  $E(n) = 100\%$ . The assumption that a given task can be divided into unequal subtasks, each executed by a processor, is unrealistic. Even if we assume that time taken for a doing task and the time taken to execute the task using multiprocessor, then also we cannot achieve 100 % efficiency. Because time of executing job using multiprocessor not only involves time for execution but also time for dividing the task into subtasks, assigning them to different machines, time taken for multiprocessor communication and time for combining the results.

---

## 5.7 FUTURE DIRECTION OF

---

### MULTIPROCESSOR ARCHITECTURE

---

Initially when the idea of parallel processing was conceived there were limited parallel processing applications such as vector manipulation which are extensively used in computational science and engineering and problems too large to solve on one computer, may use 100s or 1000s of computers to solve these.

Many companies in the 80s/90s “bet” on parallel computing and failed to make profit because computers got faster too quickly. What initially thought of computer as slow computing device as taken a momentum according to Moore’s law and become very fast computing device such that it overshadowed the concept of parallel processing. But we still claim the importance of parallel processing because following two reasons :

There are more applications that can take  
benefit of parallelism  
! Moore’s law does not hold any more

Now-a-days, there are many parallel frameworks that wanted to exploit the benefits of parallel processing. Examples such as MapReduce, Graph traversal, Dynamic programming, Backtracking/B&B, Graphical models, N-Body, (Un) Structured Grid primitively exhibit the nature of parallel execution. These new applications have open up new avenues in the area of parallel processing.

Further, Moore’s law no longer holds good. Following graph gives the number of transistors used in a processor as the years progressed.

In a present situation we cannot follow Moore’s law because although VLSI and nanotechnologies enabled putting large transistors on the chip but clock speed is not possible to increase in that speed. Further overcrowding the transistors on the chip is leading into new problem of heat dissipation. Thus new design



philosophy is designing multicore processors. All the aspects we had discussed with multiprocessor architecture such as processor communication, data

sharing and cache coherence still holds good in multicore architectures. Only difference between these two is in multiprocessor architecture each processor uses bus exclusively whereas in multicore a bus is shared by many cores. Problems such as bus arbitration are newly introduced in multicore architectures.

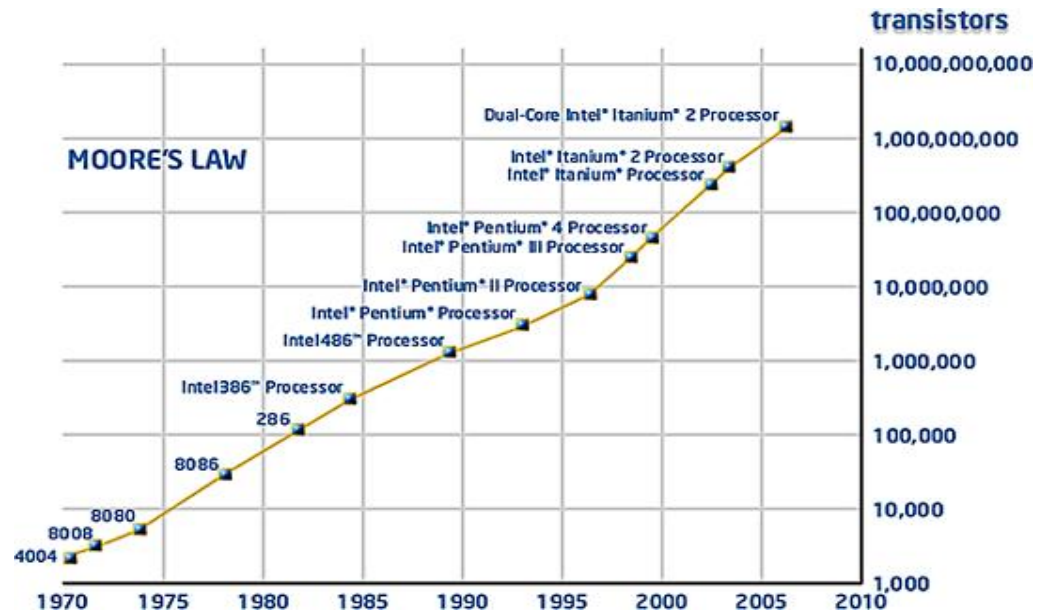


Fig. 5.10 Moore's law showing number of transistors against year



## Check Your Progress

- 1) State whether following statements are True or False.
  - a) Multiprocessor must have more than one processors
  - b) Concepts of Multiprocessors and pipelining are same
  - c) Multiprocessors and Multicore architecture are same
  - d) SISD is a single processor architecture.
  - e) Many concepts of Multiprocessor are applicable in Multicore architecture.
- 2) Flynn's classification while classifying systems and programs into four groups taken in consideration
  - a) Computers operating on single or set of instructions
  - b) Computers operating on single or set of data.State which of the following is true
  - i) Only a
  - ii) Only b
  - iii) Both a & b
  - iv) Neither a nor b

## 5.8 LET US SUM UP

- There are some applications that are inherently parallel in nature and there are some applications when executed in parallel yields greater results in terms of performance.
- Flynn's classified computing based on categorizing instruction set and data sets that a program can have and that computers used to execute those programs.
- Flynn's classification has resulted in four category of computing
  - SISD, SIMD, MISD, MIMD

- Among four classification two classifications SIMD and MIMD are two architectures well suited for parallel processing.
- Two variants of SIMD are based on whether the memory is local or centralized.
- Two variants of MIMD are based on use of Shared memory or message-passing techniques for interprocessor communication.



## 5.9 ANSWER TO CHECK YOUR PROGRESS

- 1)
  - a) True
  - b) False
  - c) False
  - d) True
  - e) True
- 2) iii



## 5.10 FURTHER READING

- 1) Computer organization and design by David A Peterson, John L Hennessy - Morgan Kaufmann Publications.
- 2) Advance Computer Architecture - Kai Hwang – McGRAW-HILL publications.
- 3) Computer Organization and architecture : Designing for performance by William Stalling – Prentice Hall publications.



## 5.11 Model Questions

- 1) Explain two motivations behind the concepts of Multiprocessor architecture.

- 2) State and explain in brief Flynn's classification of Multiprocessor architecture.
- 3) Which kind of applications are most suited for SISD class of Multiprocessor architecture. With a neat diagram explain two variants of SIMD class of Multiprocessor architecture.
- 4) State and explain shared memory variant of MIMD multiprocessor architecture with a neat diagram.
- 5) State and explain message-passing variant of MIMD multiprocessor architecture with a neat diagram.
- 6) State and explain associated problems with the shared memory in shared memory variant of MIMD multiprocessor architecture.
- 7) Give five tuple representation of SIMD scheme.
- 8) Explain Wormhole routing for message-passing technique.
- 9) Give reasons why concepts of Multiprocessor are still relevant today.

\*\*\*\*\*



---

## UNIT STRUCTURE

- 6.1 Learning Objectives
- 6.2 Introduction
- 6.3 Pipeline - The Basic Technique
- 6.4 Types of Pipelines
- 6.5 Instruction Pipelining
- 6.6 Pipeline Performance
- 6.7 Pipeline Hazards
- 6.8 Merits and Demerits of Pipelining
- 6.9 Let Us Sum Up
- 6.10 Answers to Check Your Progress
- 6.11 Further Readings
- 6.12 Possible Questions

---

### 6.1 LEARNING OBJECTIVES

---

After going through this unit, you will be able to

- define and elaborate what is pipelining
- describe instruction pipelining
- illustrate the hazards in pipelining
- know the benefits of pipelining

---

### 6.2 INTRODUCTION

---

So far, you have come across with lots of the interesting topics of computer organization like addressing modes, instruction formats, ALU and CU organizations etc. Whatever you have learnt from the previous units - the main objective in the mind of the researchers is to design a computer system with highest performance by using those techniques. In this unit, we will discuss one of the most important technique called pipelining, which is used in the modern computers to achieve extreme performance.

---

## 6.3 PIPELINE - THE BASIC TECHNIQUE

---

The speed of computation of the modern processors increases day by day to a certain extent. This happens by adopting new techniques for organizing the concurrent activities in a computer system. One of such technique is called pipelining. The idea behind the pipelining is very simple.

In everyday life, people do many tasks in stages. For instance, when we do the laundry, we place a load in the washing machine. When it is done, it is transferred to the dryer and another load is placed in the washing machine. When the first load is dry, we pull it out for folding or ironing, moving the second load to the dryer and start a third load in the washing machine. We proceed with folding or ironing of the first load while the second and third loads are being dried and washed, respectively. We may have never thought of it this way but we do laundry by **pipeline processing**.

*A Pipeline is a series of stages, where some work is done at each stage. The work is not finished until it has passed through all stages.*

Now let us see, how the idea of pipelining can be used in computers. We know that the processor executes a program by fetching and executing instructions, one after the other. In the following figure, shows two hardware unit of a processor, one for fetching the instructions and the other for executing instructions. The intermediate storage buffer B1 stores the instruction fetched by the fetch unit. This buffer is needed to enable the execution unit to execute the instruction while the fetch unit is fetching the next instruction. Thus, with pipelining, the computer architecture allows the next instructions to be fetched while the processor is performing execution of the first instruction, holding them in a buffer close to the processor until each instruction operation can be performed.



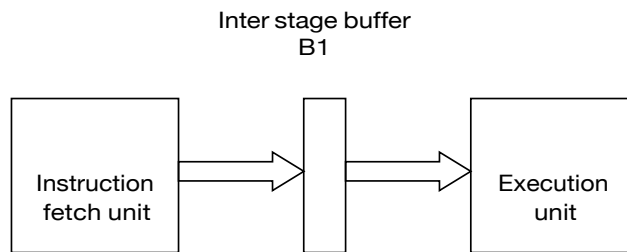


Fig 6.1 Fetch and Execution Unit

The staging of instruction fetching is continuous. The result is an increase in the number of instructions that can be performed during a given time period.

So in computers, a pipeline is the continuous and somewhat overlapped movement of instruction to the processor to perform an instruction.

But what will happen without pipelining ? Without a pipeline, a computer processor gets the first instruction from memory, performs the operation it calls for, and then goes to get the next instruction from memory, and so forth. While fetching (getting) a instruction, the executing part of the processor will remain idle. It must wait until it gets the next instruction. Thus, it results slower in execution because less number of instructions will be executed during a given time slot.

Thus, the Pipelining is used to obtain improvements in processing time that would be unobtainable with existing non-pipelined technology. The IBM 7030 (the Stretch Computer) had attained its over-all performance of 100 times by using the pipelining technique, whereas circuit improvements would only give a factor-of-10 improvement. This goal could only be met with overlapping instructions, i.e. pipelining.

John Hayes provides a definition of a pipeline as it applies to a computer processor.

*"A pipeline processor consists of a sequence of processing circuits, called segments or stages, through which a stream of operands can be passed. Partial processing of the operands takes place in each*

*Computer Organization and Architecture*



#### NOTE

The **IBM 7030**, also known as **Stretch**, was IBM's first transistorized super-computer in 1961. Original priced was \$13.5 million. Even though the 7030 was much slower than expected, it was the fastest computer in the world from 1961 until

segment. A fully processed result is obtained only after an operand set has passed through the entire pipeline."

---

## 6.4 TYPES OF PIPELINES

---

The concept of pipelining is divided into two categories :

- **Instructional pipeline**, where different stages of an instruction fetch and execution are handled in a pipeline.
- **Arithmetic pipeline**, where different stages of an arithmetic operation are handled along the stages of a pipeline.



### NOTE

The processor operates at the speed of an internal clock whose clock rate is determined by the frequency of an oscillator crystal (quartz crystal) that, when subjected to an electrical current, send pulses, called "peaks". The clock speed (also called cycle), corresponds to the number of pulses per second, written in Hertz (Hz). Thus, a 200 MHz computer has a clock that sends 200,000,000 pulses per second. Clock frequency is generally a multiple of the system frequency (FSB, Front- Side Bus),

---

## 6.5 INSTRUCTION PIPELINING

---

An instruction pipeline is a technique used in the design of computers to increase their instruction throughput (the number of instructions that can be executed in a unit of time). The fundamental idea is to split the processing of a computer instruction into a series of independent steps, with storage at the end of each step.

Most of the modern CPUs are designed to operate on a synchronization signal which known as a **clock signal**. In each clock pulse, the fetch step and execution step should be completed. The above situation discussed in the previous section (Fig. 6.1) can be illustrated by introduction of clock pulse as shown in the figure below :

In the first clock cycle, the fetch unit fetches an instruction  $I_1$  (step  $F_1$ ) and stores it in buffer  $B_1$  at the end of the clock cycle. In the second clock cycle, the instruction fetch unit proceeds with the fetch operation for instruction  $I_2$  (step  $F_2$ ). At the same time, the execution unit performs the operation specified by instruction  $I_1$ , which is available to it in buffer  $B_1$  (step  $E_1$ ).

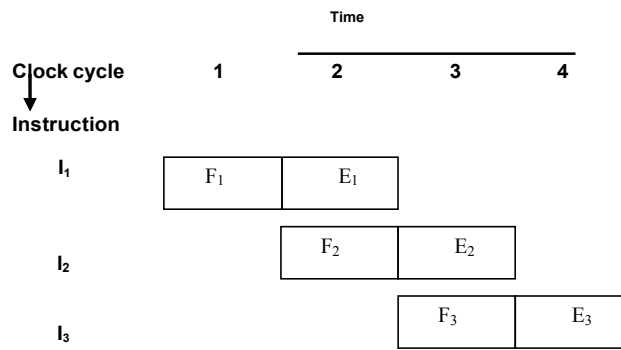


Fig. 6.2 Pipelined Execution

By the end of the second clock cycle, the execution of instruction  $I_1$  is completed and instruction  $I_2$  is available. Instruction  $I_2$  is stored in

$B_1$ , replacing  $I_1$ , which is no longer needed. Step  $E_2$  is performed by the execution unit during the third clock cycle, while instruction  $I_3$  is

being fetched by the fetched unit.

Thus, the above fetch and execute unit constitute a two stage pipeline. Both the fetch and execute units are kept busy all the time after each clock pulse and also new information is loaded into the buffer after each clock pulse.

Actually, the steps of processing an instruction is not limited within two steps, instead four steps are used as shown in Fig. 5.4 to process an instruction. These steps are :

- **Fetch** : read the instruction from the memory
- **Decode** : decode the instruction and fetch the source operands
- **Execute** : perform the operation specified by the instruction
- **Write** : store the result in the destination location

So, it will need four different hardware units as shown in Fig 5.3 for performing each of these steps simultaneously and without interfering with one another (because for two steps we get two hardware unit as shown in the fig. 6.1).

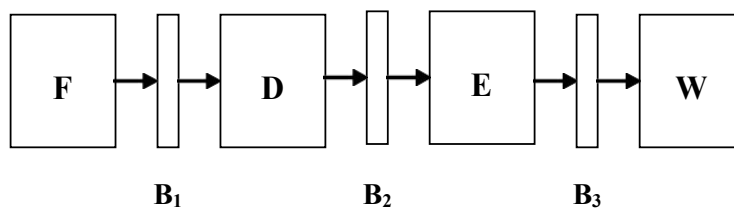


Fig. 8.3 Hardware units need for Instruction Processing

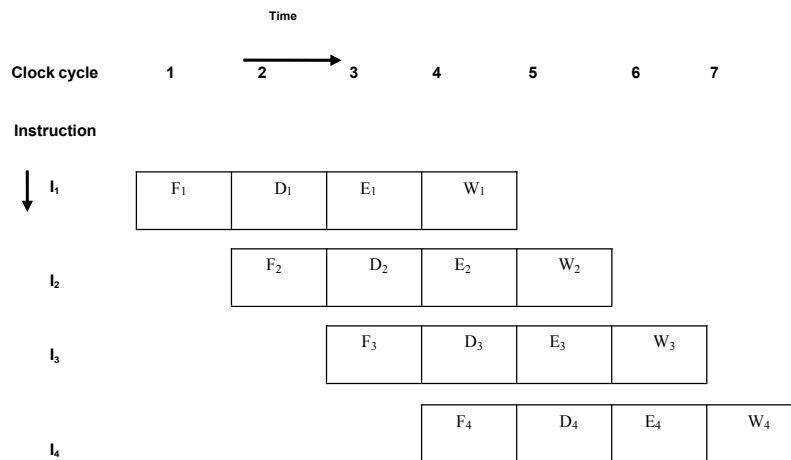


Fig. 6.4 Instruction processing in four steps

Information is passed from one unit to the next unit through a storage buffer. The information in the buffers at the *clock pulse 4* is given in the following table :

#### Buffers

#### Contents

- B<sub>1</sub> Holds instruction I<sub>3</sub> that was fetched in clock pulse 3 and will be decoded by the decoding unit.
- B<sub>2</sub> Holds the information produced by the decoding unit in clock pulse 3. This buffer also holds the information needed for the write step of instruction I<sub>2</sub> (step W<sub>2</sub>). This information must be passed on to stage W in the following clock cycle to enable that stage to perform the required Write operation.
- B<sub>3</sub> Holds the result produced by the execution unit and the destination information for instruction I<sub>1</sub>.

## 6.6 PIPELINE PERFORMANCE

Pipelining increases the CPU instruction throughput, its meaning is that pipelining increases the number of instructions completed per unit of time. But it does not reduce the execution time of an individual instruction. In fact, it usually slightly increases the execution time of each instruction due to overhead in the pipeline control. The increase in instruction throughput means that a program runs faster and has lower total execution time.

The potential increase in performance resulting from pipelining should be proportional to the number of pipeline stages. But in practical, this is not absolutely correct because due to some other factors the performance varies.

A pipeline stage may not be able to complete its processing task during the allotted time slot. The execution stage actually responsible for arithmetic and logic operation and one clock cycle may not be sufficient for it. The following figure shows that in instruction  $I_2$  the Execution stage takes three cycles (4,5,6) to complete. Thus, in cycles 5 and 6, the Write stage must be told to do nothing, because it has no data to work with.

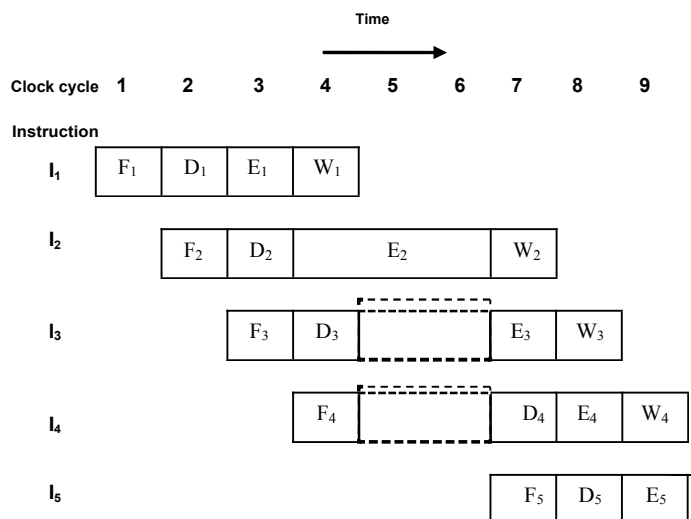


Fig. 6.5 Execution stage taking more than one clock cycle

Meanwhile, the information in buffer  $B_2$  must remain intact until the Execute stage has completed its operation. This means that stage 2 and, in turn, stage 1 are blocked from accepting new instructions because the information in  $B_1$  cannot be overwritten. Thus, steps  $D_4$  and  $F_5$  must be postponed as shown.

Thus, the normal pipeline operation interrupted for two clock cycle, which is called **stalled**. In stalled situation, the normal pipeline operation halt for more than one clock cycle. In the figure, we see that, pipeline functioning resumes normally from the clock cycle 7.

---

## 6.7 PIPELINE HAZARDS

---

**Pipeline hazards** are situations that prevent the next instruction in the instruction stream from executing during its assigned clock cycle. Then, the instruction is said to be stalled. When an instruction is stalled, all instructions later in the pipeline than the stalled instruction are also stalled. Instructions earlier than the stalled one can continue. No new instructions are fetched during the stall. This condition is clearly illustrated in the Fig. 5.5. Thus, the hazards reduces the performance gained by pipelining.

There are three classes of hazards :

- a) Structural hazards
- b) Data hazards
- c) Control hazards

### a) Structural hazards :

Structural hazards arise from resource conflict. This is the situation when two instructions **require the use of a hardware resource** at the same time. This hazard may arise frequently when two instruction refers to the memory locations at the same time. Suppose one instruction try to access the memory as result of Execute or Write stage and another instruction tries to access the memory for fetching instruction. In such a condition, the structural hazards arises. If

instruc-

tion and data reside in the same cache unit, only one instruction can proceed and the other instruction is delayed. In general, structural hazards are avoided by providing sufficient hardware resources on the processor chip.

#### **b) Data hazards :**

Data hazard arises when the output of one instruction is fed to the input of the next instruction. A data hazard is any condition in which either the **source or the destination operands** of an instruction are not available at the time expected in the pipeline. As a result, some operation has to be delayed, and the pipeline stalls. We have already discussed a situation (in Fig. 6.5) where the normal pipeline operations were halted for two clock cycles. That was happened due to the data hazards only.

#### **c) Control hazards :**

Due to the delay in availability of an instruction the pipeline may stall. For example, a cache miss causes the instruction to be fetched from the main memory. Such hazards are often called *control hazards* or *instruction hazards*.

The following Fig. illustrates a cache miss in pipeline operation. Here, the instruction  $I_1$  is fetched from the cache in cycle 1, and its execution proceeds normally. Next, the fetch operation for the instruction  $I_2$ , which is started in cycle 2, results in a cache miss, it means that the instruction  $I_2$  is not available in the cache memory and that instruction will be fetched from the main memory. The instruction fetch unit must now suspend any further fetch requests and wait for  $I_2$  to arrive. At the end of the clock cycle 5, the instruction  $I_2$  is received and loaded into the buffer  $B_1$ . The pipeline resumes its normal operation from the clock cycle 5.



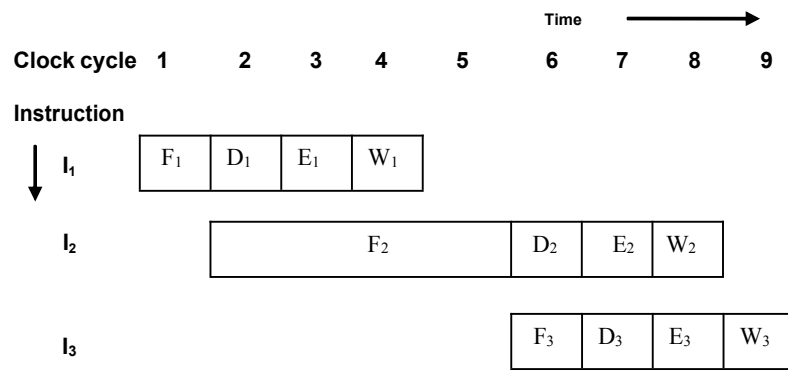


Fig. 6.6 Cache miss in pipeline operation

## 6.8 MERITS AND DEMERITS OF PIPELINING

Pipelining does not help in all cases. There are several possible disadvantages. An instruction pipeline is said to be *fully pipelined* if it can accept a new instruction at every clock cycle. A pipeline that is not fully pipelined has wait cycles that delay the progress of the pipeline.

### **Advantages of Pipelining:**

1. The cycle time of the processor is reduced, thus increasing instruction issue-rate in most cases.
2. Some combinational circuits such as adders or multipliers can be made faster by adding more circuitry. If pipelining is used instead, it can save circuitry vs. a more complex combinational circuit.

### **Disadvantages of Pipelining:**

1. A non-pipelined processor executes only a single instruction at a time. This prevents branch delays (in effect, every branch is delayed) and problems with serial instructions being executed concurrently. Consequently the design is simpler and cheaper to manufacture.
2. The instruction latency in a non-pipelined processor is slightly lower than in a pipelined equivalent. This is due to the fact that extra flip flops must be added to the data path of a pipelined

processor.

3. A non-pipelined processor will have a stable instruction bandwidth. The performance of a pipelined processor is much harder to predict and may vary more widely between different programs.



## LET US KNOW

We have come to know that pipeline operation will be effective if it completes the every stage in each clock cycle. So, each clock cycles should sufficiently long to complete the task of each stage. Thus, the performance of a pipeline will increase if the task performed in each stage require the same amount of time.

Now, consider a fetch operation where the instructions are fetched from the main memory and for that fetched operation one clock cycle may not be sufficient. Because the access time of the main memory may be as much as ten times greater than the time needed to perform basic pipeline stage operations inside the processor. So, if each instruction fetch requires access to the main memory, then the pipelining technique will be a time consuming process and meaning less job.

We have already know that cache memory is the second fastest memory element in the memory hierarchy. When a cache is included on the same chip of the processor, called *processor cache*, access time to the cache is usually the same as the time needed to perform other basic operations inside the processor. Thus, the use of the cache memory eliminates the above difficulty and accelerates the pipeline operations.



## CHECK YOUR PROGRESS

### 1. State True or False

- a) Arithmetic operation is handled in every stage of an instruction pipelining.
- b) Pipeline reduces the execution time of an individual instruction.
- c) Stall is a situation which speedup the normal pipeline operation.
- d) Data hazard arises when two instruction refers to the same memory location.
- e) Branch instruction causes the control hazard.
- f) A cache miss causes structural hazard

---

## 6.9 LET US SUM UP

---

- 1. In computers, a pipeline is the continuous and somewhat overlapped movement of instruction to the processor or in the arithmetic steps taken by the processor to perform an instruction.
- 2. Computer processor pipelining is sometimes divided into an instruction pipeline and an arithmetic pipeline.
- 3. The increase in instruction throughput means that a program runs faster and has lower total execution time.
- 4. A hazard is a potential problem that can happen in a processor. There are typically three types of hazards : data hazards, structural hazards, and control hazards.
- 5. Data hazard arises when the output of one instruction is fed to the input of next instruction.
- 6. A structural hazard occurs when a part of the processor's hardware is needed by two or more instructions at the same time.

7. Control hazards are produced by branch instructions.



## 6.10 ANSWERS TO CHECK YOUR PROGRESS

---

1.     a) False,        b) False,        c) False,        d) False,  
         e) True,        f) False        g) True



## 6.11 FURTHER READINGS

---

1.     M. Morris Mano, Computer System Architecture,  
         PEARSON Prentice Hall
2.     Hamacher, Vranesic, Zaky Computer Organization,  
         Mc Graw Hill
3.     William Stallings, Computer Organization and Architecture,  
         PHI



## 6.12 POSSIBLE QUESTIONS

---

1.     What is the significance of pipelining ? Discuss it.
2.     Differentiate between a pipelined computer and a non-pipelined computer.
3.     What is pipelining ? What are the types of pipelining ?
4.     Assuming four stage (i.e. fetch, decode, execute, write) of processing an instruction discuss the normal pipeline operation for three instructions.
5.     What is a pipeline hazard ? What are the types of hazard ?
6.     What is stall ? Discuss data hazard giving appropriate example.
7.     Discuss control hazard giving appropriate example.
8.     What is pipeline hazard ? Write the merits and demerits of pipelining.

\*\*\*\*\*