



GATE PEDIA



**COMPUTER SCIENCE &
INFORMATION TECHNOLOGY**

Published By:



ISBN: 978-93-94342-51-4

Mobile App: Physics Wallah (Available on Play Store)



Website: www.pw.live

Email: support@pw.live

Rights

All rights will be reserved by Publisher. No part of this book may be used or reproduced in any manner whatsoever without the written permission from author or publisher.

In the interest of student's community:

Circulation of soft copy of Book(s) in PDF or other equivalent format(s) through any social media channels, emails, etc. or any other channels through mobiles, laptops or desktop is a criminal offence. Anybody circulating, downloading, storing, soft copy of the book on his device(s) is in breach of Copyright Act. Further Photocopying of this book or any of its material is also illegal. Do not download or forward in case you come across any such soft copy material.

Disclaimer

A team of PW experts and faculties with an understanding of the subject has worked hard for the books.

While the author and publisher have used their best efforts in preparing these books. The content has been checked for accuracy. As the book is intended for educational purposes, the author shall not be responsible for any errors contained in the book.

The publication is designed to provide accurate and authoritative information with regard to the subject matter covered.

This book and the individual contribution contained in it are protected under copyright by the publisher.

(This Module shall only be Used for Educational Purpose.)

INDEX

- | | | |
|-----|--|--------------|
| 1. | Engineering Mathematics (EM) | 1.1 – 1.40 |
| 2. | Discrete Mathematics | 2.1 – 2.40 |
| 3. | Digital Logic | 3.1 – 3.75 |
| 4. | Computer Organization & Architecture | 4.1 – 4.56 |
| 5. | Programming & Data Structures | 5.1 – 5.41 |
| 6. | Algorithms | 6.1 – 6.34 |
| 7. | Theory of Computation | 7.1 – 7.39 |
| 8. | Compiler Design | 8.1 – 8.23 |
| 9. | Operating System | 9.1 – 9.44 |
| 10. | Database Management Systems | 10.1 – 10.46 |
| 11. | Computer Networks | 11.1 – 11.37 |
| 12. | General Aptitude | 12.1 – 12.25 |

Engineering Mathematics



Engineering Mathematics

INDEX

- | | | |
|----|----------------------------------|-------------|
| 1. | Basic Calculus | 1.1 – 1.16 |
| 2. | Linear Algebra | 1.17 – 1.28 |
| 3. | Probability and Statistics | 1.29 – 1.40 |



GATE Exam 2025?



SHRESHTH BATCH

ME | CE | EE | EC | CS & IT

- Live Classes & Recorded Videos
- DPPs & Solutions
- Doubt Solving
- Batches are available in *Hindi*

Weekday & Weekend
Batches Available



JOIN NOW!



Physics W

1

BASIC CALCULUS

1.1. Introduction

1.1.1 Limits, Continuity and Differentiability

(a) As x tends to a ($x \rightarrow a$) $\Rightarrow x$ is moving towards a

A value l is said to be limit of a function $f(x)$ at $x \rightarrow a$ if $f(x) \rightarrow l$ as $x \rightarrow a$.

It is mathematically defined as

$$\lim_{x \rightarrow a} f(x) = l = \lim_{x \rightarrow a^-} f(x) = \lim_{x \rightarrow a^+} f(x)$$

That is, Limit exist at any point, if LHL = RHL

A function $f(x)$ is said to be continuous at $x = a$, if

$$\lim_{x \rightarrow a} f(x) = l = f(a) = f(x)|_{x=a}$$

That is, for a function to be continuous at any point, RHL = LHL = Value of function at point $x = a$.

- Note:**
- For $\lim_{x \rightarrow a} f(x)$ to exist, the function need not be continuous at $x = a$.
 - But for $f(x)$ to be continuous at $x = a$, $\lim_{x \rightarrow a} f(x)$ should exist.

- Continuity from Left : $\lim_{x \rightarrow a^-} f(x) = f(a)$

- Continuity from Right : If $\lim_{x \rightarrow a^+} f(x) = f(a)$

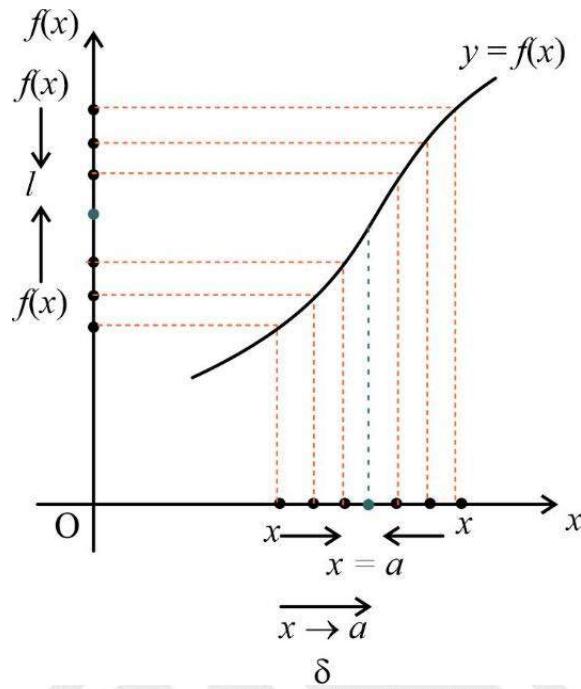
Continuity in an Open Interval

A function ' f ' is said to be continuous in open interval (a, b) , if it is continuous at each point of open interval.

Continuity in a Closed Interval

Let ' f ' be a function defined on the closed interval (a, b) then ' f ' is said to be continuous on the closed interval $[a, b]$, if it is :

1. Continuous from the right at a and
2. Continuous from the left at b and
3. Continuous on the open interval (a, b) .


Fig. 1.1

(b) Concept of differentiability

A continuous function $f(x)$ is said to be differentiable at $x = a$, if $\lim_{x \rightarrow a} \frac{f(x) - f(a)}{x - a}$ exists, that is, RHL and LHL exist at a point under consideration in $f'(x)$.

$$f'(x)|_{x=a} = f'(a) = \lim_{x \rightarrow a} \frac{f(x) - f(a)}{x - a}$$

$f'(a) = \tan \theta$, where θ is the angle made by the tangent to the curve at $x=a$ with x – axis.

(c) Some Standard Derivatives

$$(i) \quad \frac{d}{dx}(x^n) = n \cdot x^{n-1}$$

$$(ii) \quad \frac{d}{dx}(\sin x) = \cos x$$

$$(iii) \quad \frac{d}{dx}(\cos x) = -\sin x$$

$$(iv) \quad \frac{d}{dx}(\tan x) = \sec^2 x$$

$$(v) \quad \frac{d}{dx}(\cot x) = -\operatorname{cosec}^2 x$$

$$(vi) \quad \frac{d}{dx}(\sec x) = \sec x \cdot \tan x$$

$$(vii) \quad \frac{d}{dx}(\operatorname{cosec} x) = -\operatorname{cosec} x \cot x$$

$$(viii) \quad \frac{d}{dx}(\sin^{-1} x) = \frac{1}{\sqrt{1-x^2}}; -1 < x < 1$$

$$(ix) \quad \frac{d}{dx}(\cos^{-1} x) = \frac{-1}{\sqrt{1-x^2}}, -1 < x < 1$$

$$(x) \quad \frac{d}{dx}(\tan^{-1} x) = \frac{1}{1+x^2}$$

- (xi) $\frac{d}{dx}(\cot^{-1} x) = \frac{-1}{1+x^2}$
- (xii) $\frac{d}{dx}(\sec^{-1} x) = \frac{1}{|x|\sqrt{x^2-1}}$
- (xiii) $\frac{d}{dx}(\operatorname{cosec}^{-1} x) = \frac{-1}{|x|\sqrt{x^2-1}}; |x| > 1$
- (xiv) $\frac{d}{dx}(\log_a x) = \frac{1}{x \log_e a}$
- (xv) $\frac{d}{dx}(\log_e x) = \frac{1}{x}$
- (xvi) $\frac{d}{dx}(a^x) = a^x \cdot \log_e a$
- (xvii) $\frac{d}{dx}(e^x) = e^x$
- (xviii) $\frac{d}{dx}(|x|) = \frac{|x|}{x}, (x \neq 0)$
- (xix) $\frac{d}{dx}(x^x) = x^x(1 + \log_e x)$
- (xx) $\frac{d}{dx}(\sinh x) = \cosh x$

(d) Product rule of differentiation

- (i) $\frac{d}{dx}(f(x) \cdot g(x)) = f(x) \cdot g'(x) + f'(x) \cdot g(x)$
- (ii) $d(uvw) = uvw' + uv'w + u'vw$

(e) Quotient rule of differentiation

$$\frac{d}{dx}\left(\frac{f(x)}{g(x)}\right) = \frac{g(x)f'(x) - f(x)g'(x)}{(g(x))^2}, (g(x) \neq 0)$$

(f) Logarithmic differentiation:

Taking log might help in differentiation of a function. For example if $y = v^u$ then we can take log both side and

differentiable to get $\frac{dy}{dx}$

(g) Differentiation in parametric form :

If we write x and y in term of find variable 't' that is $x = f(t)$, $y = \omega(t)$, then $\frac{dy}{dx} = \frac{dy/dt}{dx/dt}$

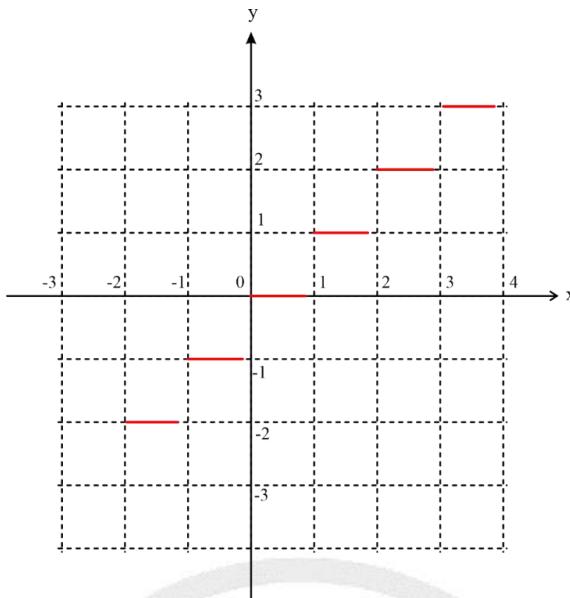
(h) Greatest Integer function / step function / integer part function

$$f(x) = [x] = n, \forall n \leq x < n + 1 \text{ where, } n \in \mathbb{Z}$$

$$\lim_{x \rightarrow a}[x] = \nexists \text{ if } a \text{ is an integer} \quad (\therefore \nexists = \text{do not exist})$$

$$\text{L.H.L.} = \lim_{x \rightarrow a^-}[x] = a - 1$$

$$\text{R.H.L.} = \lim_{x \rightarrow a^+}[x] = a$$


Fig.1.2. Greatest Integer

(i) Properties of Limits

$$(i) \lim_{x \rightarrow a} (f(x) \pm g(x)) = \lim_{x \rightarrow a} f(x) \pm \lim_{x \rightarrow a} g(x)$$

$$(ii) \lim_{x \rightarrow a} (f(x) \cdot g(x)) = \lim_{x \rightarrow a} f(x) \cdot \lim_{x \rightarrow a} g(x)$$

$$(iii) \lim_{x \rightarrow a} \frac{f(x)}{g(x)} = \frac{\lim_{x \rightarrow a} f(x)}{\lim_{x \rightarrow a} g(x)}, (\lim_{x \rightarrow a} g(x) \neq 0)$$

(iv) If $\lim_{x \rightarrow a} f(x)$ exists and $\lim_{x \rightarrow a} g(x) = \emptyset$, then $\lim_{x \rightarrow a} f(x) \cdot g(x)$ may exist

Example: Let $f(x) = \sin x$, $g(x) = \frac{1}{x}$, $\lim_{x \rightarrow 0} f(x) = 0$, $\lim_{x \rightarrow 0} \frac{1}{x} = \emptyset$

$$\text{But } \lim_{x \rightarrow 0} \sin x \cdot \frac{1}{x} = 1$$

(v) Indeterminate form III ($0^0, 1^\infty, \infty^0$)

$$\text{If } y = \lim_{x \rightarrow a} [f(x)]^{\phi(x)}$$

$$\text{Then, } \log y = \lim_{x \rightarrow a} \phi(x) \log [f(x)]$$

Thus $0^0, 1^\infty, \infty^0$ will convert into $\infty \times 0$ from which can be solved easily.

$$(vi) \text{ If } \lim_{x \rightarrow a} \frac{f(x)}{g(x)} = \frac{0}{0} \text{ (or) } \frac{\infty}{\infty}, \text{ then } \lim_{x \rightarrow a} \frac{f(x)}{g(x)} = \lim_{x \rightarrow a} \frac{f'(x)}{g'(x)} \neq \left(\frac{0}{0}\right)$$

$$\text{If } \lim_{x \rightarrow a} \frac{f'(x)}{g'(x)} = \frac{0}{0} \text{ (or) } \frac{\infty}{\infty}, \text{ then } \lim_{x \rightarrow a} \frac{f'(x)}{g'(x)} = \lim_{x \rightarrow a} \frac{f''(x)}{g''(x)} \text{ and so on}$$

$$(vii) \text{ If } \lim_{x \rightarrow a} (f(x) \cdot g(x)) = 0 \times \infty \Rightarrow \lim_{x \rightarrow a} \frac{f(x)}{\left(\frac{1}{g(x)}\right)} = \frac{0}{0} \text{ (Apply L-Hospital Rule again)}$$

(j) Some Standard Limits

- (i) $\lim_{x \rightarrow 0} \frac{\sin x}{x} = 1$
- (ii) $\lim_{x \rightarrow 0} \frac{\tan x}{x} = 1$
- (iii) $\lim_{x \rightarrow 0} \frac{1 - \cos ax}{x^2} = \frac{a^2}{2}$
- (iv) $\lim_{x \rightarrow \infty} \frac{\sin x}{x} = 0$
- (v) $\lim_{x \rightarrow \infty} \frac{\cos x}{x} = 0$
- (vi) $\lim_{x \rightarrow 0} (1 + ax)^{b/x} = e^{ab}$
- (vii) $\lim_{x \rightarrow \infty} \left(1 + \frac{a}{x}\right)^{bx} = e^{ab}$
- (viii) $\lim_{x \rightarrow 0} \left(\frac{a^x + b^x}{2}\right)^{1/x} = \sqrt{ab}$
- (ix) $\lim_{x \rightarrow 0} \left(\frac{1^x + 2^x + 3^x + \dots + n^x}{n}\right)^{1/x} = \sqrt[n]{n!}$
- (x) $\lim_{x \rightarrow 0} \frac{a^x - 1}{x} = \log_e a ; \lim_{x \rightarrow 0} \frac{e^x - 1}{x} = 1$
- (xi) $\lim_{x \rightarrow 0} x \cdot \sin\left(\frac{1}{x}\right) = 0$

1.2 Mean Value Theorems

1.2.1 Lagrange's Mean Value Theorem (LMVT)

If $f(x)$ is continuous in $[a, b]$ and it is differentiable in (a, b) then \exists at least one point 'c' such that $c \in (a, b)$ and

$$f'(c) = \frac{f(b) - f(a)}{b - a}$$

Here $f'(c)$ slope of tangent to $f(x)$ at $x = c$.

Tangent at $x = c$ is parallel to the line connecting the points A and B

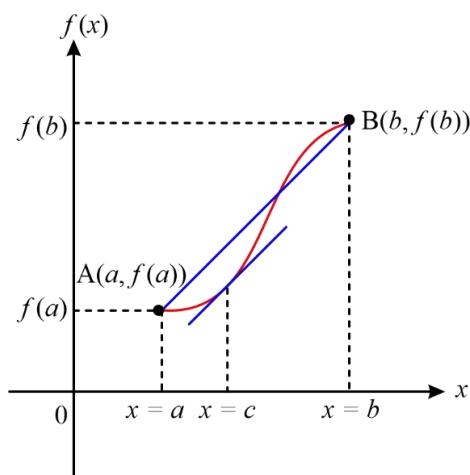


Fig.1.3. LMVT

1.2.2 Rolle's Mean Value Theorem

If $f(x)$ is continuous in $[a, b]$ and differentiable in (a, b) and $f(a) = f(b)$ then \exists at least one-point $c \in (a, b)$ such that $f'(c) = 0$.

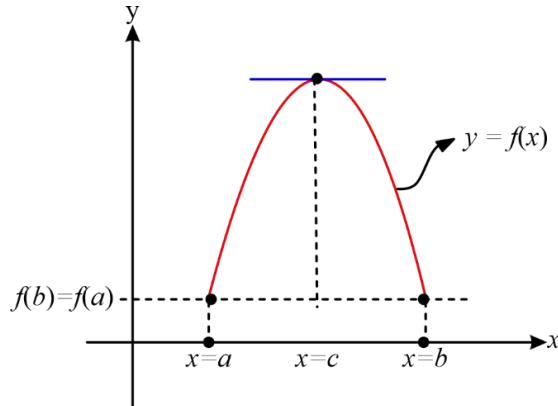


Fig. 1.4. Rolle's mean value

1.2.3 Cauchy's Mean Value Theorem

If $f(x)$ and $g(x)$ are continuous in $[a, b]$ and differentiable in (a, b) then \exists at least one value of 'c' such that $c \in (a, b)$ and $\frac{g'(c)}{f'(c)} = \frac{g(b)-g(a)}{f(b)-f(a)}$

1.3 Increasing and Decreasing Functions

1.3.1 Increasing Functions

A function $f(x)$ is said to be increasing, if $f(x_1) < f(x_2) \forall x_1 < x_2$

Or

A function $f(x)$ is said to be increasing, if $f(x)$ increases as x increases.

For a function $f(x)$ to be increasing at the point $x=a$, $f'(a) > 0$.

Example:

$e^x, \log_e x \rightarrow$ Monotonically increasing functions

$\sin x$ in $(0, \pi/2)$ \rightarrow non-monotonic functions

1.3.2 Decreasing Functions

A function $f(x)$ is said to be a decreasing function, if $f(x_1) > f(x_2) \forall x_1 < x_2$

A function $f(x)$ is said to be decreasing function, if $f(x)$ decreases as x increases.

Example: $e^{-x} \rightarrow$ Monotonically decreasing function, $\sin x$ in $(\frac{\pi}{2}, \pi)$

1.4. Concept of Maxima and Minima

Let $f(x)$ be a differentiable function, then to find the maximum (or) minimum of $f(x)$.

- (1) Find $f'(x)$ and equate to zero.

- (2) Solve the resulting equation for x . Let its roots be a_1, a_2, \dots then $f(x)$ is stationary at $x = a_1, a_2, \dots$. Thus $x = a_1, a_2, \dots$ are the only points at which $f(x)$ can be maximum or a minimum.
- (3) Find $f''(x)$ and substitute in it by terms $x = a_1, a_2, \dots$. wherever $f''(x)$ is negative, we have a maximum and wherever $f''(x)$ is positive, we have a minimum.
- (4) If $f''(a_1) = 0$, find $f'''(x)$ put $x = a_1$ in it. If $f'''(a_1) \neq 0$, there is neither a maximum nor a minimum at $x = a_1$. If $f'''(a_1) = 0$, find $f^{iv}(x)$ and put $x = a_1$ in it. If $f^{iv}(a_1)$ is negative, we have maximum at $x = a_1$, if it is positive there is a minimum at $x = a_1$. If $f^{iv}(a_1)$ is zero, we must find $f^v(x)$, and so on. Repeat the above process for each root of the equation $f'(x) = 0$.

Example: $x = 0$ is a critical point of $f(x) = x^3$

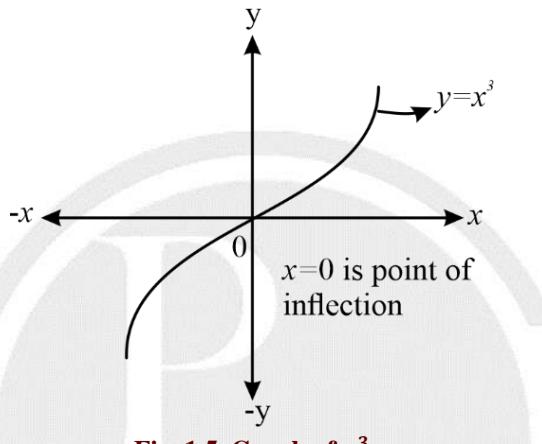


Fig. 1.5. Graph of x^3

$$f(x) = x^3$$

$$\Rightarrow f'(x) = 3x^2 = 0 \Rightarrow x = 0$$

$$f''(x) = 6x \Rightarrow f''(0) = 6(0) = 0$$

- **Global maxima and minima :**

We first find local maxima and minima and then calculate the value of ' f ' at boundary points of interval given e.g. (a, b) we find $f(a)$ and $f(b)$ and compare it with the values of local maxima and minima. The absolute maxima and minima can be decided then.

1.5. Taylor Series

If $f(x)$ is continuously differentiable ($f'(x), f''(x), f'''(x), \dots$ exists) then the Taylor series expansion of $f(x)$ about the point $x = a$ is given by

$$f(x) = f(a) + \frac{f'(a)}{1!}(x - a) + \frac{f''(a)}{2!}(x - a)^2 + \frac{f'''(a)}{3!}(x - a)^3 + \dots \infty$$

If $a = 0$, then $f(x) = f(0) + \frac{f'(0)}{1!}x + \frac{f''(0)}{2!}x^2 + \frac{f'''(0)}{3!}x^3 + \dots \infty$ (Remember that Mc-Lauren Series is same as Taylor Series if $a = 0$)

The coefficient of $(x - a)^n$ in the Taylor series expansion of $f(x)$ is $\frac{f^n(a)}{n!}$.

The general expansion of Taylor series is given by $f(x + h) = f(x) + h \cdot \frac{f'(x)}{1!} + h^2 \cdot \frac{f''(x)}{2!} + h^3 \cdot \frac{f'''(x)}{3!} + \dots \infty$

- Finding the expansion of e^x about $x = 0$

$$f(x) = e^x \Rightarrow f(0) = e^0 = 1$$

$$f'(x) = e^x \Rightarrow f'(0) = e^0 = 1; f''(0) = f'''(0) = f''''(0) = \dots = 1$$

$$f(x) = e^x = 1 + (x - 0) \frac{1}{1!} + (x - 0)^2 \cdot \frac{1}{2!} + (x - 0)^3 \cdot \frac{1}{3!} + \dots$$

$$\Rightarrow e^x = 1 + \frac{x}{1!} + \frac{x^2}{2!} + \frac{x^3}{3!} + \dots$$

1.6 Integral Calculus

If $F(x)$ is anti-derivative of $f(x)$. That is, continuous and differentiable in (a, b) , then we write $\int_{x=a}^{x=b} f(x) dx = F(b) - F(a)$. Here $f(x)$ is integrand

If $f(x) > 0 \forall a \leq x \leq b$, then $\int_a^b f(x) dx$ represents the shaded area in the given figure.

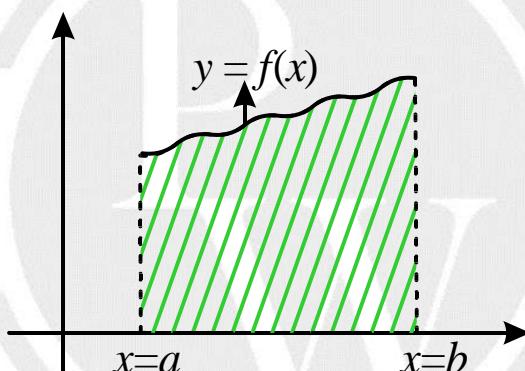


Fig.1. 6. Integration of continuous function

1.6.1 Mean Value Theorem of Integration

If $f(x)$ is continuous in $[a, b]$ and differentiable in (a, b) then ' \exists ' atleast one-point $c \in (a, b)$ such that

$$f(c) = \frac{\int_a^b f(x) dx}{(b-a)}$$

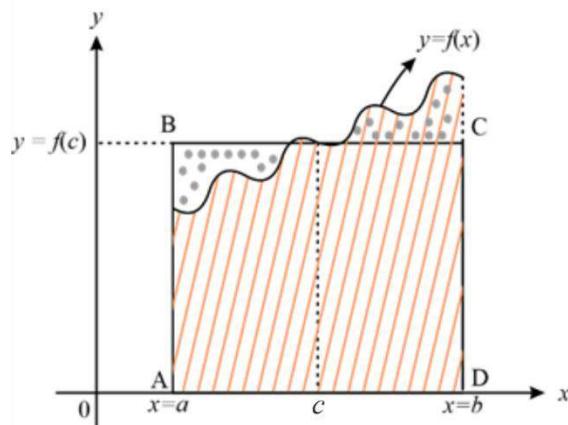


Fig. 1.7. Mean value of integration

1.7. Newton-Leibnitz Rule

If $f(x)$ is continuously differentiable and $\phi(x)$, $\psi(x)$ are two functions for which the 1st derivative exists, then

$$\frac{d}{dx} \left(\int_{\phi(x)}^{\psi(x)} f(x) dx \right) = f(\psi(x)) \cdot \psi'(x) - f(\phi(x)) \cdot \phi'(x)$$

Example: $\frac{d}{dx} \left(\int_x^{x^2} \sin x dx \right) = \sin(x^2) \cdot 2x - \sin x \cdot 1 = 2x \sin(x^2) - \sin x$

1.8. Some Standard Integrals

$$1. \int x^n dx = \frac{x^{n+1}}{n+1} + C, (n \neq -1)$$

$$2. \int \frac{1}{x} dx = \log_e |x| + C$$

$$3. \int \sin x dx = -\cos x + C$$

$$4. \int \cos x dx = \sin x + C$$

$$5. \int \frac{f'(x)}{f(x)} dx = \log_e |f(x)| + C$$

$$6. \int \tan x dx = - \int -\frac{\sin x}{\cos x} dx = -\log_e |\cos x| + C$$

$$\Rightarrow \int \tan x dx = \log_e |\sec x| + C$$

$$7. \int \cot x dx = \int \frac{\cos x}{\sin x} dx = \log_e |\sin x| + C = -\log_e |\cosec x| + C$$

$$8. \int \sec x dx = \int \frac{\sec x (\sec x + \tan x)}{(\sec x + \tan x)} dx = \log_e |\sec x + \tan x| + C$$

$$9. \int \cosec x dx = \log_e |\cosec x - \cot x| + C$$

$$10. \int a^x dx = \frac{a^x}{\log_e a} + C$$

$$11. \int \frac{1}{x \log_e a} dx = \log_a x + C$$

$$12. \int x^x (1 + \log_e x) dx = x^x + C$$

$$13. \int f(x) \cdot f'(x) dx = \frac{1}{2} (f(x))^2 + C$$

$$14. \int \frac{f'(x)}{\sqrt{f(x)}} dx = 2 \cdot \sqrt{f(x)} + C$$

15. If $f(x)$, $g(x)$ are two functions. that are differentiable, then

$$\int f(x) g(x) dx = f(x) \cdot \int g(x) dx - \int [f'(x) g(x)] dx + C$$

Before integrating the product, the functions $f(x)$ and $g(x)$ are to be arranged according to the ILATE Principle.

Here, ILATE stands for INVERSE LOGARITHMIC ALGEBRAIC TRIGONOMETRIC EXPONENTIAL.

1.9 Properties of Definite Integrals

1. If $f(x)$ is differentiable in interval (a, b) , then $\int_a^b f(x) dx = - \int_b^a f(x) dx$
2. If \exists a point $c \in (a, b)$ such that $f(x)$ is not differentiable, then

$$\int_a^b f(x) dx = \int_a^c f(x) dx + \int_c^b f(x) dx$$
3. If $f(x)$ is continuously differentiable function,

$$\int_{-a}^a f(x) dx = 2 \times \int_0^a f(x) dx; \text{ if } f(-x) = f(x), (\text{"f(x) is even function"})$$

$$= 0; \text{ if } f(-x) = -f(x), (\text{"f(x) is odd function"})$$
4. $\int_0^{2a} f(x) dx = 2 \times \int_0^a f(x) dx, \text{ if } f(2a - x) = f(x)$
5. $\int_a^b f(x) dx = \int_a^b f(a + b - x) dx$
6. $\int_a^b \frac{f(x)}{f(x) + f(a+b-x)} dx = \left(\frac{b-a}{2}\right)$

Example:

- (i) $\int_0^{\pi/2} \frac{\sin x}{\sin x + \cos x} dx = \frac{\pi}{4}$
- (ii) $\int_0^{\pi/2} \frac{1}{1 + \sqrt{\tan x}} dx = \int_0^{\pi/2} \frac{1}{1 + \left(\frac{\sqrt{\sin x}}{\sqrt{\cos x}}\right)} dx = \int_0^{\pi/2} \frac{\sqrt{\cos x}}{\sqrt{\cos x} + \sqrt{\sin x}} dx = \frac{\pi}{4}$
- (iii) $\int_2^3 \frac{\sqrt{x}}{\sqrt{x} + \sqrt{5-x}} dx = \left(\frac{3-2}{2}\right) = \frac{1}{2}$
- (iv) $\int_0^{\pi/2} \frac{\sqrt{\tan x}}{\sqrt{\tan x} + \sqrt{\cot x}} dx = \frac{\pi}{4}$
7. $\int_0^{\pi/2} \sin^m x dx = \int_0^{\pi/2} \cos^m x dx = \frac{(m-1)(m-3)(m-5)\dots(1)}{m(m-2)(m-4)} \times \dots \left(\frac{1}{2}\right) \text{ (or) } \frac{2}{3} \times K$

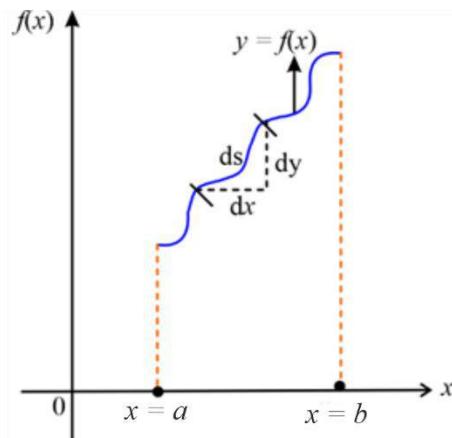
Where $K = \pi/2$ if m is even

= 1 if m is odd.

8. $\int_0^{\pi} \frac{dx}{a^2 \cos^2 x + b^2 \sin^2 x} = \frac{\pi}{ab}$
9. $\int_0^{\pi/2} \frac{dx}{a^2 \cos^2 x + b^2 \sin^2 x} = \frac{\pi}{2ab}$

1.10 Length of a Curve

- (a) The length of the arc of the curve $y = f(x)$ between the points where $x = a$ and $x = b$ is $s = \int_a^b \sqrt{1 + \left(\frac{dy}{dx}\right)^2} dx$


Fig.1.8. Length of the curve

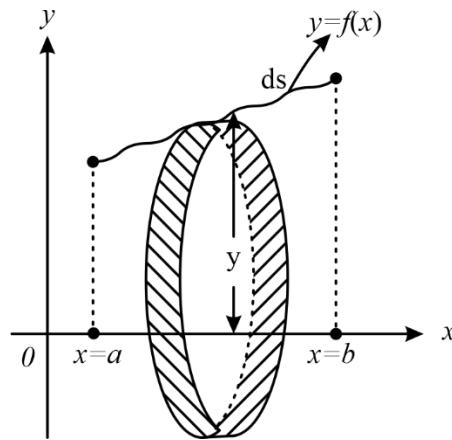
- (b) The length of the arc of the curve $x = f(y)$ between the points where $y = a$ and $y = b$, is $s = \int_a^b \sqrt{1 + \left(\frac{dx}{dy}\right)^2} dy$
- (c) The length of the arc of the curve $x = f(t), y = f(t)$ between the points where $t = a$ and $t = b$, is $s = \int_a^b \sqrt{\left(\frac{dx}{dt}\right)^2 + \left(\frac{dy}{dt}\right)^2} dt$
- (d) The length of the arc of the curve $r = f(\theta)$, between the points where $\theta = \alpha$ and $\theta = \beta$, is $s = \int_{\alpha}^{\beta} \sqrt{r^2 + \left(\frac{dr}{d\theta}\right)^2} d\theta$

1.11 Surface Area of Solid generated by revolving a curve about a fixed axis

Elemental Surface Area

$$dA = 2\pi y \times ds = 2\pi y ds$$

$$\Rightarrow \text{Total surface area } A = \int_{x=a}^{x=b} 2\pi y \sqrt{1 + \left(\frac{dy}{dx}\right)^2} dx$$


Fig.1.9. Surface area

1.12 Volume of the solid

- A. The volume of the solid obtained by revolving the curve $y = f(x)$ between the lines $x = a$ and $x = b$ is given by
 $\Rightarrow V \approx \int_{x=a}^{x=b} \pi y^2 dx$

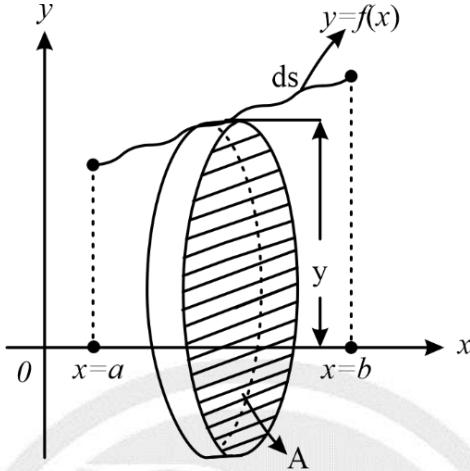


Fig. 1.10. Volume of the solid

- B. **Revolution about the y-axis.** Interchanging x and y in the above formula, we see that the volume of the solid generated by the revolution, about y -axis, of the area, bounded by the curve $x = f(y)$, the y -axis and the abscissa $y = a, y = b$ is $\int_a^b \pi x^2 dy$.

1.13 Gamma Function

The integral $\int_0^\infty e^{-x} \cdot x^{n-1} dx, (n > 0)$ is called Gamma function of n . It is denoted by $\Gamma n = \int_0^\infty e^{-x} x^{n-1} dx$.

Note :
$$\int_0^{\pi/2} \sin^m x \cos^n x dx = \frac{\Gamma\left(\frac{m+1}{2}\right)\Gamma\left(\frac{n+1}{2}\right)}{2\Gamma\left[\frac{m+n+2}{2}\right]}$$

Where $\Gamma(x)$ is called the gamma function.

1.13.1 Properties of Gamma Function

- | | |
|-----------------------------------|--|
| (i) $\Gamma n = (n - 1)!$ | (ii) $\Gamma(n + 1) = (n)!$ |
| (iii) $\Gamma(n + 1) = n\Gamma n$ | (iv) $\Gamma\left(\frac{1}{2}\right) = \sqrt{\pi}$ |

1.14 Beta Function

The function $\beta(m, n) = \int_0^1 x^{m-1} \cdot (1-x)^{n-1} dx (m, n > 0)$ is called β function of m and n .

1.14.1 Properties of β function

- (i) $\beta(m, n) = \frac{\Gamma m \cdot \Gamma n}{\Gamma(m+n)}$
- (ii) $\beta(m, n) = \beta(n, m)$
- (iii) $\beta(m, n) = \int_0^{\infty} \frac{x^{m-1}}{(1+x)^{m+n}} dx$
 $\beta(n, m) = \int_0^{\infty} \frac{x^{n-1}}{(1+x)^{m+n}} dx$
- (iv) $\sin^p \theta \cdot \cos^q \theta dx = \frac{1}{2} \beta \left(\frac{p+1}{2}, \frac{q+1}{2} \right), (p, q > -1)$

1.15 Area between the curves

If the function $f(x) > g(x)$ for all values of x between $x=a$ and $x=b$ then

$$A = \int_a^b f(x)dx - \int_a^b g(x)dx \Rightarrow A = \int_a^b (f(x) - g(x)) dx$$

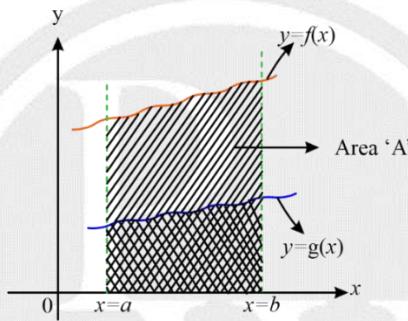


Fig. 1.11. Area under curve

Note : Area bounded by curve $r = f(\theta)$ between $\theta = \alpha$ and β is $\frac{1}{2} \int_{\alpha}^{\beta} r^2 d\theta$

1.16 Multi Variable Calculus

(a) Continuity of a function

A function $f(x, y)$ is said to be continuous at (a, b) , if $\lim_{\substack{x \rightarrow a \\ y \rightarrow b}} f(x, y) = f(a, b)$

(b) Differentiation of a two-variable function

If $f(x, y)$ is a continuous function, then the derivative of $f(x, y)$ with respect to x treating y as constant is given by

$$p = \frac{\partial f}{\partial x} = \lim_{h \rightarrow 0} \frac{f(x+h, y) - f(x, y)}{h}$$

The derivative of $f(x, y)$ with respect to y treating x as constant is given by

$$q = \frac{\partial f}{\partial y} = \lim_{k \rightarrow 0} \frac{f(x, y+k) - f(x, y)}{k}$$

(c) Homogenous Function

A function $f(x, y)$ is said to be homogenous function of degree ' n ' if $f(kx, ky) = k^n \cdot f(x, y)$.

Example: $f(x, y) = x^3 - 3x^2y + 3xy^2 + y^3$

$$\begin{aligned} \Rightarrow f(kx, ky) &= (kx)^3 - 3(kx)^2(ky) + 3(kx)(ky)^2 + (ky)^3 \\ &= k^3(x^3 - 3x^2y + 3xy^2 + y^3) \\ &= k^3 \cdot f(x, y) \Rightarrow f(x, y) \text{ is a homogenous function of degree '3'.} \end{aligned}$$

(d) Euler's Theorem

If $f(x, y)$ is a homogeneous function of degree ' n ' then

$$(i) x \cdot \frac{\partial f}{\partial x} + y \cdot \frac{\partial f}{\partial y} = nf$$

$$(ii) x^2 \cdot \frac{\partial^2 f}{\partial x^2} + 2xy \cdot \frac{\partial^2 f}{\partial x \partial y} + y^2 \cdot \frac{\partial^2 f}{\partial y^2} = n(n-1)f$$

If $f(x, y) = g(x, y) + h(x, y) + \phi(x, y)$ where $g(x, y)$, $h(x, y)$ and $\phi(x, y)$ are homogenous functions of degrees m , n and p respectively, then

$$x \cdot \frac{\partial f}{\partial x} + y \cdot \frac{\partial f}{\partial y} = m \cdot g(x, y) + n \cdot h(x, y) + p \cdot \phi(x, y)$$

$$x^2 \cdot \frac{\partial^2 f}{\partial x^2} + 2xy \cdot \frac{\partial^2 f}{\partial x \partial y} + y^2 \cdot \frac{\partial^2 f}{\partial y^2} = m(m-1) \cdot g(x, y) + n(n-1) \cdot h(x, y) + p(p-1) \cdot \phi(x, y)$$

(e) Total derivative:

$$(i) \text{ If } u = f(x, y) \text{ and if } x = \omega(t), y = v(t) \text{ then } \frac{du}{dt} = \frac{\partial u}{\partial x} \cdot \frac{dx}{dt} + \frac{\partial u}{\partial y} \cdot \frac{dy}{dt}$$

$$(ii) \text{ If } u \text{ be a function of } x \text{ and } y, \text{ where } y \text{ is a function of } x, \text{ then } \frac{du}{dx} = \frac{\partial u}{\partial x} + \frac{\partial u}{\partial y} \cdot \frac{dy}{dx}$$

$$(iii) \text{ If } u = f(x, y) \text{ and } x = f_1(t_1, t_2) \text{ and } y = f_2(t_1, t_2), \text{ then}$$

$$\frac{\partial u}{\partial t_1} = \frac{\partial u}{\partial x} \cdot \frac{\partial x}{\partial t_1} + \frac{\partial u}{\partial y} \cdot \frac{\partial y}{\partial t_1} \text{ and } \frac{\partial u}{\partial t_2} = \frac{\partial u}{\partial x} \cdot \frac{\partial x}{\partial t_2} + \frac{\partial u}{\partial y} \cdot \frac{\partial y}{\partial t_2}$$

$$(iv) \text{ If } x \text{ and } y \text{ are connected by an equation of the form } f(x, y) = 0, \text{ then } \frac{dy}{dx} = -\frac{\partial f / \partial x}{\partial f / \partial y}$$

(f) Concept of Maxima and Minima in Two Variables

If $f(x, y)$ is a two-variable differentiable function, then to find the maxima (or) minima.

Step-1: Calculate $p = \frac{\partial f}{\partial x}$ and $q = \frac{\partial f}{\partial y}$ and equate $p = 0, q = 0$

Let (x_0, y_0) be a stationary point.

Step-2: Calculate r, s, t where $r = \frac{\partial^2 f}{\partial x^2} \Big|_{(x_0, y_0)}$; $s = \frac{\partial^2 f}{\partial x \partial y} \Big|_{(x_0, y_0)}$; $t = \frac{\partial^2 f}{\partial y^2} \Big|_{(x_0, y_0)}$

Case (i): If $rt - s^2 > 0$ and $r > 0$, then the function $f(x, y)$ has minimum at (x_0, y_0) and the minimum value is $f(x_0, y_0)$.

Case (ii): If $rt - s^2 > 0$ and $r < 0$, then the function $f(x, y)$ has maximum at (x_0, y_0) and the maximum value is

$f(x_0, y_0)$.

Case (iii): If $rt - s^2 < 0$; then we cannot comment on the existence of maxima and minima.

Such stationary points where $rt - s^2 = 0$ are called **saddle points**.

(g) Concept of Constraint Maxima and Minima (Method of Lagrange's unidentified multipliers).

If $f(x, y, z)$ is a continuous and differentiable function, such that the variables x, y and z are related/constrained by the equation $\phi(x, y, z) = C$ then to calculate the extreme value of $f(x, y, z)$ using Lagrange's Method of unidentified multipliers.

Step-1: Form the function $F(x, y, z) = f(x, y, z) + \lambda\{\phi(x, y, z) - C\}$, where λ is a multiplier.

Step-2: Calculate $\frac{\partial F}{\partial x}, \frac{\partial F}{\partial y}$ and $\frac{\partial F}{\partial z}$ and equate them to zero

Step-3: Equate the values of λ from the above 3 equations and obtain the relation between the variables x, y and z .

Step-4: Substitute the relation between x, y and z in $\phi(x, y, z) = C$ and get the values of x, y, z . Let they be (x_0, y_0, z_0) .

Step-5: Calculate $f(x_0, y_0, z_0)$

The value $f(x_0, y_0, z_0)$ is the extreme value of $f(x, y, z)$.

(h) Multiple Integrals

If $f(x, y)$ is continuous and differentiable at every point within a region ' R ' bounded by some curves is given by

$$I = \iint_R f(x, y) dx dy$$

If there is a double integral,

$$I = \int_{x=a}^{x=b} \int_{y=\phi(x)}^{y=\psi(x)} f(x, y) dy dx \quad [\text{Let } \psi(x) > \phi(x)]$$

Then $I = \text{area of the region bounded by the curves, } y = \phi(x); y = \psi(x), x = a \text{ and } x = b \text{ if } f(x, y) = 1$

Value of x – co-ordinate of centroid of the region bounded by $y = \phi(x); y = \psi(x); x = a, x = b$ if $f(x, y) = x$

(i) Change of Orders of Integration

$$I = \int_{x=a}^{x=b} \int_{y=\phi(x)}^{y=\psi(x)} f(x, y) dy dx \rightarrow I = \int_{y=c}^{y=d} \int_{x=g(y)}^{x=h(y)} f(x, y) dx dy$$

In change of order of Integration, the order of the Integrating variables changes and the limits as well.

Note : When limits are constants, the order of integration does not matter,

$$\int_{y=c}^{y=d} \int_{x=a}^{x=b} f(x, y) dx dy = \int_{x=a}^{x=b} \int_{y=c}^{y=d} f(x, y) dy dx$$

1.17 Jacobian of the Transformation

(i) The Jacobian of the transformation, $x = f_1(u, v), y = f_2(u, v)$ is defined as,

$$J = \frac{\partial(x, y)}{\partial(u, v)} = \begin{vmatrix} x_u & x_v \\ y_u & y_v \end{vmatrix}$$

(ii) The Jacobian of the transformation,

$x = f_1(u, v, w), y = f_2(u, v, w), z = f_3(u, v, w)$ is defined as

$$J = \frac{\partial(x, y, z)}{\partial(u, v, w)} = \begin{vmatrix} x_u & x_v & x_w \\ y_u & y_v & y_w \\ z_u & z_v & z_w \end{vmatrix}$$

1.18 Change of Variables Formula

$$(i) \iint_R f(x, y) dx dy = \iint_R f(f_1(u, v), f_2(u, v)) |J| du dv$$

$$(ii) \iiint_R f(x, y, z) dxdydz = \iiint_R f(f_1(u, v, w), f_2(u, v, w), f_3(u, v, w)) |J| du dv dw$$

1.19 Change of Variables

(i) Cartesian to polar co-ordinates :

$$x = r \cos \theta$$

$$y = r \sin \theta$$

$$J = r$$

$$dx dy = r dr d\theta$$

(ii) Cartesian to cylindrical polar co-ordinate :

$$x = r \cos \theta$$

$$y = r \sin \theta$$

$$z = z$$

$$J = r$$

$$dxdydz = r dr d\theta dz$$

(iii) Cartesian to spherical polar co-ordinates :

$$x = \rho \sin \phi \cos \theta$$

$$y = \rho \sin \phi \sin \theta$$

$$z = \rho \cos \phi$$

$$J = \rho^2 \sin \phi$$

$$dxdydz = \rho^2 \sin \phi d\rho d\phi d\theta$$



2

LINEAR ALGEBRA

2.1 Matrix

An array of elements in horizontal lines (Rows) and Vertical Lines (Columns) is called a Matrix.

Example: $A = \begin{bmatrix} i & n & d & i & a \\ j & a & p & a & n \end{bmatrix}$

2.1.1 Size of Matrix

If a matrix has 'm' rows and 'n' columns, then we say that the size of the matrix is $m \times n$ (read as m by n)

$$A = \begin{bmatrix} a_{11} & a_{12} & a_{13} & \dots & a_{1n} \\ a_{21} & a_{22} & a_{23} & \dots & a_{2n} \\ \cdot & \cdot & \cdot & \ddots & \cdot \\ \cdot & \cdot & \cdot & \ddots & \cdot \\ a_{m1} & a_{m2} & a_{m3} & \dots & a_{mn} \end{bmatrix}; A = [a_{ij}]_{m \times n} \text{ such that } 1 \leq i \leq m, 1 \leq j \leq n \text{ and } a_{ij} = f(i, j)$$

2.1.2 Addition of Matrices

- (i) Two matrices $A = [a_{ij}]_{m \times n}$ & $B = [b_{ij}]_{p \times q}$ can be added only if $m = p$ & $n = q$.
- (ii) Matrix Addition is commutative ($A + B = B + A$)
- (iii) Matrix Addition is Associative. $A + (B + C) = (A + B) + C$
- (iv) Existence of additive identity : If O be $m \times n$ matrix each of whose elements are zero. Then, $A + O = A = O + A$ for every $m \times n$ matrix A.
- (v) Existence of additive inverse : Let $A = [a_{ij}]_{m \times n}$ then the negative of matrix A is defined as matrix $[-a_{ij}]_{m \times n}$ and is denoted by $-A$.
 \Rightarrow Matrix $-A$ is additive inverse of A. Because $(-A) + A = O = A + (-A)$. Here O is null matrix of order $m \times n$.
- (vi) Cancellation laws holds good in case of addition of matrices, which is $X = -A$.
 $\Rightarrow A + X = B + X \Rightarrow A = B$
 $\Rightarrow X + A = X + B \Rightarrow A = B$
- (vii) The equation $A + X = 0$ has a unique solution in the set of all $m \times n$ matrices.

2.1.3 Multiplication of Matrices

The multiplication of two matrices $A = [a_{ij}]_{m \times n}$ and $B = [b_{ij}]_{p \times q}$ ($\Rightarrow AB_{m \times q}$) is feasible only if $n = P$.

$$A_{m \times n} \cdot B_{p \times q} = C$$

$$A = \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix}_{3 \times 3} \quad B = \begin{bmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \\ b_{31} & b_{32} \end{bmatrix}_{3 \times 2} \quad A_{3 \times 3} \times B_{3 \times 2}$$

$$\Rightarrow \begin{bmatrix} a_{11} \cdot b_{11} + a_{12} \cdot b_{21} + a_{13} \cdot b_{31} & a_{11}b_{12} + a_{12}b_{22} + a_{13}b_{32} \\ a_{21} \cdot b_{11} + a_{22} \cdot b_{21} + a_{23} \cdot b_{31} & a_{21}b_{12} + a_{22}b_{22} + a_{23}b_{32} \\ a_{31} \cdot b_{11} + a_{32} \cdot b_{21} + a_{33} \cdot b_{31} & a_{31}b_{12} + a_{32}b_{22} + a_{33}b_{32} \end{bmatrix}_{3 \times 2}$$

2.1.4 Properties of Multiplication of Matrices

- (i) Matrix Multiplication Need not be commutative.
- (ii) Matrix Multiplication is Associative $(A(BC)) = ((AB)C)$
- (iii) Matrix Multiplication is distributive $A(B + C) = (AB + AC)$
- (iv) The product of two Matrices $A_{m \times n}, B_{n \times q}$ (i.e. $AB_{m \times q}$) can be a zero matrix even if $A \neq O \& B \neq O$.

Example: $A = \begin{bmatrix} 3 & 0 \\ 0 & 0 \end{bmatrix}; B = \begin{bmatrix} 0 & 0 \\ 0 & 4 \end{bmatrix} \Rightarrow AB = \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix}$

- For the multiplication of two matrices $A_{m \times n} \& B_{n \times q}$
 - (i) The No. of Multiplications required = $m n q$
 - (ii) The number of Additions required = $m (n - 1) q$

2.2 Types of Matrices

- (1) **Upper triangular Matrix:** A matrix $A = [a_{ij}]$; $1 \leq i, j \leq n$ is said to be an upper triangular matrix if

$$a_{ij} = 0 \quad \forall i > j$$

- (2) **Lower Triangular Matrix:** A matrix $A = [a_{ij}]_{n \times n}$; $1 \leq i, j \leq n$ is said to be a lower Triangular Matrix

$$\text{if } a_{ij} = 0 \quad \forall i < j$$

- (3) **Diagonal Matrix:** A matrix $A = [a_{ij}]$, $\forall 1 \leq i, j \leq n$ is said to be a diagonal matrix if $a_{ij} = 0 \quad \forall i \neq j$

Example: $A = \begin{bmatrix} d_1 & 0 & 0 \\ 0 & d_2 & 0 \\ 0 & 0 & d_3 \end{bmatrix}$. The diagonal Matrix is also denoted as $A = \text{diag } [d_1, d_2, d_3]$

- (4) **Scalar Matrix:** A Matrix ' A ' = $[a_{ij}]$; $1 \leq i, j \leq n$ is said to be a scalar Matrix if $a_{ij} = \begin{cases} K; i = j \\ 0; 1 \neq j \end{cases}$

If $K = 1$, then $A \rightarrow$ Identity Matrix,

If $K = 0$, then $A \rightarrow$ Null Matrix.

- (5) **Idempotent Matrix:**

A Matrix ' $A_{n \times n}$ ' is said to be an idempotent matrix if $A^2 = A$.

Example: $A = \begin{bmatrix} 4 & -1 \\ 12 & -3 \end{bmatrix}$

$$\Rightarrow A \cdot A = \begin{bmatrix} 4 & -1 \\ 12 & -3 \end{bmatrix} \begin{bmatrix} 4 & -1 \\ 12 & -3 \end{bmatrix} = \begin{bmatrix} 4 & -1 \\ 12 & -3 \end{bmatrix} = A$$

(6) **Nilpotent Matrix:** A matrix A is said to be nilpotent of class x or index if $A^x = 0$ and $A^{x-1} \neq 0$ i.e. x is the smallest index which makes $A^x = 0$.

Example: The matrix $A = \begin{bmatrix} 1 & 1 & 3 \\ 5 & 2 & 6 \\ -2 & -1 & -3 \end{bmatrix}$ is nilpotent class 3, since $A \neq 0$ and $A^2 \neq 0$, but $A^3 = 0$.

(7) **Orthogonal Matrix:** A matrix A is said to be orthogonal if $A \cdot A^T = I$

Example: $\begin{bmatrix} \cos \theta & -\sin \theta \\ \sin \theta & \cos \theta \end{bmatrix}$

(8) **Involutory Matrix:** A matrix A is said to be involutory if $A^2 = I$

Example: $\begin{bmatrix} 2 & 3 \\ -1 & -2 \end{bmatrix}$

2.3 Transpose of a Matrix

For a given matrix $= [a_{ij}]$; $1 \leq i \leq m, 1 \leq j \leq n$, we can say that 'B' where $B = [b_{ij}]$, $i \leq i \leq n, i \leq j \leq m$ is the transpose of the Matrix 'A' if $a_{ij} = b_{ji}$

2.3.1 Properties of Transpose of a Matrix

- (i) $(A^T)^T = A$
- (ii) $(AB)^T = B^T \cdot A^T$
- (iii) $(KA)^T = KA^T$ where 'K' is a scalar.

2.4 Conjugate of a matrix

The matrix obtained by replacing each element of matrix by its complex conjugate.

2.4.1 Properties of conjugate matrix

- (a) $\bar{\bar{A}} = A$ (b) $\bar{(A+B)} = \bar{A} + \bar{B}$
- (c) $\bar{(KA)} = \bar{K}\bar{A}$ (d) $\bar{(AB)} = \bar{A}\bar{B}$
- (e) $\bar{A} = A$ if A is real matrix
- $\bar{A} = -A$ if A is purely imaginary matrix

2.5 Transposed Conjugate of a Matrix

The transpose of conjugate of a matrix is called transposed conjugate. It is represented by A^θ .

- (a) $(A^\theta)^\theta = A$
- (b) $(A+B)^\theta = A^\theta + B^\theta$
- (c) $(KA)^\theta = \bar{K}A^\theta$ (K : Complex number)
- (d) $(AB)^\theta = B^\theta A^\theta$

2.6 Trace of a Matrix

Trace is simply sum of all diagonal elements of a matrix.

2.6.1 Properties of Trace of a matrix

Let A and B be two square matrices of order n and λ is scalar then

1. $Tr(\lambda A) = \lambda Tr(A)$
2. $Tr(A + B) = Tr(A) + Tr(B)$
3. $Tr(AB) = Tr(BA)$ [If both AB and BA are defined]

2.7 Type of Real Matrix

- (a) Symmetric matrix : $(A^T = A)$
- (b) Skew symmetric matrix : $(A^T = -A)$
- (c) Orthogonal matrix : $(A^T = A^{-1}, AA^T = I)$

Note : (a) If A and B are symmetric, then $(A + B)$ and $(A - B)$ are also symmetric.

(b) For any matrix AA^T is always symmetric.

(c) For any matrix, $\left(\frac{A+A^T}{2}\right)$ is symmetric and $\left(\frac{A-A^T}{2}\right)$ is skew symmetric.

(d) For orthogonal matrices, $|A| = \pm 1$

(e) We can write any matrix A as a sum of symmetric and skew symmetric matrix $A = \frac{A+A^T}{2} + \frac{A-A^T}{2}$

2.8 Type of complex matrix

- (a) Hermitian matrix : $(A^\theta = A)$
- (b) Skew-Hermitian matrix: $A^\theta = -A$
- (c) Unitary matrix : $(A^\theta = A^{-1}, AA^\theta = I)$

Note : (a) $\frac{A+A^\theta}{2}$ is Hermitian and $\frac{A-A^\theta}{2}$ is skew Hermitian matrix.

(b) We can write any matrix as a sum of Hermitian and skew Hermitian matrix $A = \frac{A+A^\theta}{2} + \frac{A-A^\theta}{2}$

2.9 Determinant

The summation of the product of elements of a row(or) column of a matrix with their corresponding Co-factors.

$$A \cdot adj(A) = |A| \cdot I$$

Determinant can be calculated only if matrix is a square matrix.

Suppose, we need to calculate a 3×3 determinant,

$$\Delta = \sum_{j=1}^3 a_{1j} cof(a_{1j}) = \sum_{j=1}^3 a_{2j} cof(a_{2j}) = \sum_{j=1}^3 a_{3j} cof(a_{3j})$$

We can calculate determinant along any row or column of the matrix.

2.9.1 Properties of Determinants

- (i) If 'A' is a Square Matrix of size ' $n \times n$ ' and 'k' is a Scalar then
 $|K \cdot A_{n \times n}| = K^n \cdot |A_{n \times n}|$
- (ii) $|adj(A)| = |A|^{(n-1)}$
- (iii) $|adj(adj(A))| = (|A|)^{(n-1)^2}$
- (iv) $|AB| = |A| \cdot |B|$
- (v) $|(AB)^T| = |B^T| \cdot |A^T|$
- (vi) If two rows (or) two columns of a determinant are interchanged, then the determinant changes its sign.
- (vii) The determinant of an upper triangular Matrix/a lower triangular Matrix/a diagonal Matrix is the product of the principal diagonal elements of the Matrix.
- (viii) The determinant of Every Skew-Symmetric Matrix of odd order ($A_{n \times n}$) (' n ' is odd) is zero.
- (ix) The determinant of an orthogonal Matrix $A_{n \times n}$ is ± 1
- (x) The determinant of an Idempotent Matrix is either 0 (or) 1.
- (xi) The determinant of an Involuntary Matrix is ± 1
- (xii) The determinant of a Nilpotent Matrix is always zero.
- (xiii) If the product of two Non-zero Matrices $A_{n \times n} \neq 0; B_{n \times n} \neq 0$ is a zero Matrix ($(AB)_{n \times n} = 0$), then both $|A| = 0$ & $|B| = 0$.
- (xiv) If two rows (or) two columns of a Matrix are either equal or Proportional, then the determinant of the Matrix is equal to zero.
- (xv) The number of terms in the general expansion of an ' $n \times n$ ' determinant is $n!$
- (xvi) Value of the determinant is invariant under row and column interchange i.e., $|A^T| = |A|$
- (xvii) If any row or column is completely zero, then $|A| = 0$.
- (xviii) If any single row or column of the matrix is multiplied by k then the determinant of new matrix $= K|A|$
- (xix) In a determinant the sum of the product of the element of any row or column with its cofactor gives a determinant of the matrix.
- (xx) In determinant the sum of the product of the element of any row or column with a cofactor of another row or column will give zero.
- (xxi) $|AB| = |A| \times |B|$
- (xxii) Elementary operations don't effect the determinant that is $A \xrightarrow{R_i=R_i+KR_j} B$ then $|A| = |B|$
 $A \xrightarrow{C_i=C_i+KC_j} B$ then $|A| = |B|$

2.10 Minors, Cofactor and Adjoint of a Matrix

Minor of an element is equal to the determinant of the remaining elements of the matrix, after excluding the row and column containing the particular element. The cofactor of an element can be calculated from the minor of the element. The cofactor of an element is equal to the product of the minor of the element, and -1 to the power of position values of row and column of the element.

$$\text{Cofactor of an Element} = (-1)^{i+j} \times \text{Minor of an Element}$$

Here i and j are the positional values of the row and column of the element.

Example :

$$\text{If } \Delta = \begin{vmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{vmatrix}$$

$$\text{Minor of element } a_{21}: M_{21} = \begin{vmatrix} a_{12} & a_{13} \\ a_{32} & a_{33} \end{vmatrix}$$

$$\text{Co-factor of an element, } a_{ij} = (-1)^{i+j} M_{ij}$$

- To design co-factor matrix, we replace each element by its co-factor.
- Adjoint of a matrix = transpose of cofactor matrix
- $A^{-1} = \frac{\text{Adj}(A)}{|A|}$

2.11 Inverse of a matrix

Inverse of a matrix only exists for square matrices.

$$(A^{-1}) = \frac{\text{Adj}(A)}{|A|} \text{ and } |A| \neq 0$$

Properties:

- $AA^{-1} = A^{-1}A = I$
- $(AB)^{-1} = B^{-1}A^{-1}$
- $(ABC)^{-1} = C^{-1}B^{-1}A^{-1}$
- $(A^T)^{-1} = (A^{-1})^T$
- The inverse of 2×2 matrix should be remembered,

$$\begin{bmatrix} a & b \\ c & d \end{bmatrix}^{-1} = \frac{1}{(ad-bc)} \begin{bmatrix} d & -b \\ -c & a \end{bmatrix}$$

- Interchange the diagonal elements and put negative sign on the rest.
- Divide by determinant.

2.12 Rank of a Matrix

- The rank of the matrix refers to the number of linearly independent rows or columns in the matrix. $\rho(A)$ is used to denote the rank of matrix A.
- A matrix is said to be of rank zero when all of its elements become zero.
- The rank of the matrix is the dimension of the vector space obtained by its columns.
- The rank of a matrix cannot exceed more than the number of its rows or columns. The rank of the null matrix is zero.
- The nullity of a matrix is defined as the number of vectors present in the null space of a given matrix. In other words, it can be defined as the dimension of the null space of matrix A called the nullity of A. Rank + Nullity is the number of all columns in matrix A.

A real Number 'r' is said to be the rank of a matrix ' $A_{m \times n}$ ' if

- (1) There is at least one square sub-matrix of A of order r whose determinant is not equal to zero.
- (2) If the matrix A contains any square sub-matrix of order $(r + 1)$ and above, then the determinant of such a matrix should be zero.

It is mathematically denoted by $\rho(A) = r$

2.12.1 Properties of Rank of a Matrix

- $\rho(A_{m \times n}) \leq (m, n)$
- $\rho(AB) \leq \min\{\rho(A), \rho(B)\}$
- Rank of transpose of matrix is equal to rank of matrix
- Elementary operations do-not affect the rank the matrix
- $\rho(A + B) \leq \{\rho(A) + \rho(B)\}$

2.12.2 Row Echelon Form

A Matrix $A_{m \times n}$ is said to be in row-echelon form if

- Number of zeroes before the 1st Non-zero element in any row is less than the number of such zeroes in its succeeding row.
- Zero rows (if any) should lie at the bottom of the Matrix.

$\rho(A_{m \times n})$ = Number of non-zero rows in the Row-Echelon form of A.

2.13 System of Equations

The given system of equations

$$a_{11}x_1 + a_{12}x_{12} + a_{13}x_3 = b_1$$

$$a_{21}x_1 + a_{22}x_2 + a_{23}x_3 = b_2$$

$$a_{31}x_1 + a_{32}x_2 + a_{33}x_3 = b_3$$

can be written in Matrix form as

$$\begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \\ b_3 \end{bmatrix}$$

$$\begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \\ b_3 \end{bmatrix}$$

\downarrow
Coefficient

\downarrow
Variable

\downarrow
Constants

Matrix

Matrix

Matrix

The system $Ax = B$ is said to be a homogeneous system if $B = 0$.

The system of $Ax = B$ is said to be a non-homogeneous system if $B \neq 0$.

2.13.1 Consistency of a non-homogeneous system of Equations

For the above system of non – homogeneous equations, $Ax = B$; Augmented Matrix = $[A/B] = \begin{bmatrix} a_{11} & a_{12} & a_{13} & b_1 \\ a_{21} & a_{22} & a_{23} & b_2 \\ a_{31} & a_{32} & a_{33} & b_3 \end{bmatrix}$

- (i) If $\rho(A) = \rho(A/B)$ = Number of unknowns, then the system $Ax = B$ has a unique solution.
- (ii) If $\rho(A) = \rho(A/B) <$ Number of unknowns, then the system has infinitely many solutions.
- (iii) If $\rho(A) \neq \rho(A/B)$, then the system has no solution.

Number of linearly independent solutions for a system of 'n' equations given by $Ax = B$ is $n - \rho(A)$

2.13.2 Consistency of Homogeneous System of Equations

$$a_{11}x + a_{12}y + a_{13}z = 0$$

$$a_{21}x + a_{22}y + a_{23}z = 0$$

$$a_{31}x + a_{32}y + a_{33}z = 0$$

$$Ax = 0 \Rightarrow \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix} \begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix} [A/B] = \begin{bmatrix} a_{11} & a_{12} & a_{13} & 0 \\ a_{21} & a_{22} & a_{23} & 0 \\ a_{31} & a_{32} & a_{33} & 0 \end{bmatrix}_{3 \times 4}$$

If $\rho(A) = \rho(A/B) = n$ (i.e $|A| \neq 0$); the system has a unique solution.

(Trivial solution; $x = 0, y = 0, z = 0$)

If $\rho(A) = \rho(A/B) < n$ ($|A| = 0$); the system has infinitely many solutions (Non-trivial solution exists for the system).

2.14 Linear Combination of Vectors

If $x_1, x_2, x_3, \dots, x_n$ are 'n' rows vectors, then the combination $k_1x_1 + k_2x_2 + k_3x_3 + \dots + k_nx_n$ is called a linear combination of x_1, x_2, \dots, x_n ($k_1, k_2, k_3, \dots, k_n$ are scalars)

- (1) The linear combination $k_1x_1 + k_2x_2 + k_3x_3 + \dots + k_nx_n$ is said to be linearly dependent if $k_1x_1 + k_2x_2 + k_3x_3 + \dots + k_nx_n = 0$ when $k_1, k_2, k_3, \dots, k_n$ (NOT All zeroes).

If $x_1 = [a_1 \ b_1 \ c_1]$; $x_2 = [a_2 \ b_2 \ c_2]$; $x_3 = [a_3 \ b_3 \ c_3]$, then the vectors x_1, x_2, x_3 are said to be linearly dependent if $\begin{vmatrix} a_1 & b_1 & c_1 \\ a_2 & b_2 & c_2 \\ a_3 & b_3 & c_3 \end{vmatrix} = 0$.

- (2) The combination $k_1x_1 + k_2x_2 + \dots + k_nx_n$ is said to be linearly independent if $k_1x_1 + k_2x_2 + \dots + k_nx_n = 0$ when $k_1 = k_2 = k_3 = \dots = k_n = 0$

2.14.1 Eigen Values and Eigen Vectors

For any square Matrix $A_{n \times n}$, the equation $|A - \lambda I| = 0$ where ' λ ' is a scalar is called the characteristic equation.

The roots of the characteristic equation of a Matrix are called Eigen Values.

2.14.2 Properties of Eigen Values

- (i) If $\lambda_1, \lambda_2, \lambda_3, \dots, \lambda_n$ are 'n' Eigen Values of $A_{n \times n}$, then

- (a) Sum of Eigen Values of 'A' = $\lambda_1 + \lambda_2 + \lambda_3 + \dots + \lambda_n = \sum_{i=1}^n \lambda_i = \text{trace}(A)$ = Sum of Principal diagonal elements
- (b) Product of all the Eigen Values of 'A' = $\lambda_1 \cdot \lambda_2 \cdot \lambda_3 \cdot \dots \cdot \lambda_n = \prod_{i=1}^n \lambda_i = |A|$
- (c) Eigen Values of A^m are $\lambda_1^m, \lambda_2^m, \lambda_3^m, \dots, \lambda_n^m$
- (d) Eigen Values of $\text{adj}(A)$ are $\frac{|A|}{\lambda_1}, \frac{|A|}{\lambda_2}, \frac{|A|}{\lambda_3}, \dots, \frac{|A|}{\lambda_n}$
- (e) Eigen Values of A & A^T are the same.
- (f) Eigen Values of $k_1A + k_2I$ (Where k_1 and k_2 are scalar) are

$$k_1\lambda_1 + k_2, k_1\lambda_2 + k_2, k_1\lambda_3 + k_2, k_1\lambda_4 + k_2, \dots, k_1\lambda_n + k_2$$

- (ii) '0' is always an Eigen Value of an odd-order Skew-Symmetric Matrix.

- (iii) Eigen Values of a Real Symmetric Matrix are always real.

- (iv) Eigen Values of the Skew-Symmetric Matrix are either zero (or) purely Imaginary.

- (v) The Eigen values of an Orthogonal Matrix are of unit modulus.

- (vi) If the sum of all the elements in a row (or Column) is constant ($= k$) for all the rows (or columns) in the matrix respectively, then 'k' is an Eigen Value of the Matrix.

Example: If $A = \begin{bmatrix} a_1 & b_1 & c_1 \\ a_2 & b_2 & c_2 \\ a_3 & b_3 & c_3 \end{bmatrix}$ and if $a_1 + b_1 + c_1 = a_2 + b_2 + c_2 = a_3 + b_3 + c_3 = k$,

then 'k' is an Eigen Value of 'A'.

- (vii) The Eigen Values of an upper triangular Matrix, a lower triangular Matrix, a diagonal Matrix are the Principal diagonal elements of the Matrix.

2.15 Eigen Vector

A non-zero column vector $X_{n \times 1}$ is said to be an Eigen Vector of $A_{n \times n}$ corresponding to the Eigen Value ' λ ', if $AX = \lambda X (X \neq 0)$.

2.15.1 Properties of Eigen Vectors

- (i) Eigen Vectors of A & A^T are not the same.
- (ii) Eigen Vectors of A & A^M are same.
- (iii) The Eigen Vectors of a Real Symmetric Matrix are always orthogonal.
- (iv) The number of linearly independent Eigen Vectors of ' $A_{n \times n}$ ' is equal to the number of distinct Eigen Values of ' $A_{n \times n}$ '.

2.15.2 Cayley Hamilton Theorem

Every Matrix satisfies its characteristic equation.

This means that, if $c_0\lambda^n + c_1\lambda^{n-1} + \dots + c_{n-1}\lambda + c_n = 0$ is the characteristic equation of a square matrix A of order n , then

$$c_0 A^n + c_1 A^{n-1} + \dots + c_{n-1} A + c_n I = 0 \quad \dots (i)$$

Note: When λ is replaced by A in the characteristic equation, the constant term c_n should be replaced by $c_n I$ to get the result of the Cayley-Hamilton theorem, where I is the unit matrix of order n .

Also, 0 in the R.H.S. of (i) is a null matrix of order n .

2.16. Subspace (Basis of Dimensions)

2.16.1 Vector

An ordered n -tuple of numbers is called an n -vector.

2.16.2 Linearly Independent and Dependent Vector

Let X_1 and X_2 be the non-zero vectors:

- $\{x_1, x_2, \dots, x_k\}$ are linearly independent if $r_1x_1 + r_2x_2 + \dots + r_k x_k = 0$ only for $r_1 = r_2 = \dots = r_k = 0$.
- The vectors x_1, r_2, \dots, x_k are linearly dependent if they are not linearly independent; that is, if there exist scalars r_1, r_2, \dots, r_k which are not all zero such that

$$r_1x_1 + r_2x_2 + \dots + r_k x_k = 0$$

Note: Let X_1, X_2, \dots, X_n be ' n ' vector of matrix A .

- If rank (A) = number of vectors then vector X_1, X_2, \dots, X_n are linearly independent.
- If rank (A) \neq number of vectors then vector X_1, X_2, \dots, X_n are linearly dependent.

2.16.3 Vector Space \mathbb{R}^n

If n is a positive integer, then an ordered n -tuple is a sequence of n real numbers $(\alpha_1, \alpha_2, \dots, \alpha_n)$. The set of all ordered n -tuples is called n -space and is denoted by \mathbb{R}^n .

2.16.4 Subspaces of an N -vector space V_n

A non-empty set S , of vectors of $V_n(F)$, is called a subspace of $V_n(F)$, if

- ξ_1, ξ_2 are any two members of S , then $\xi_1 + \xi_2$ is also a member of S ; and
- ξ is a member of S , and k is a scalar then $k\xi$ is also a member of S .

Briefly, we may say that a set S of vectors $V_n(F)$ is a subspace of $V_n(F)$ if it is closed w.r.t. the compositions of “addition” and “multiplication with scalars”.

Every subspace of V_n contains the zero vector; being the product of any vector with the scalar zero.

2.16.5 Construction of Subspaces

- **A subspace Spanned by a Set of Vectors:** A subspace that arises as a set of all linear combinations of any given set of vectors is said to be spanned by the given set of vectors.
- **Basis of a subspace:** A set of vectors is said to be a basis of a subspace, if
 - The subspace is spanned by the set, and
 - The set is linearly independent.

Note: If we have N vectors and they are independent then they span N -dimension space. But if they are dependent then they span only a subspace of N -dimension space.

2.16.6 Orthogonality of Vectors

- Two vectors are orthogonal if each is non-zero and $X_1^T X_2 = 0$
- If n vectors X_1, X_2, \dots, X_n each of n dimensions is orthogonal then they are surely linearly independent and form the basis for n -dimension space.
- The set of vectors is orthonormal if they are orthogonal and have unit magnitude.

2.17 Similar Matrices

- Two matrices A and B are similar if there exist a non-singular matrix P such that $B = P^{-1}AP$
- Similar matrices have the same eigenvalues
- If A is similar to B then B is also similar to A
- If A is similar to B and B is similar to C then A is similar to C .

2.18 Diagonalization of a matrix

Finding a matrix D which is a diagonal matrix and which is similar to A is called diagonalization i.e., we wish to find a non-singular matrix M such that $A = M^{-1}DM$ where D is a diagonal matrix.

2.18.1 Condition for a Matrix to be Diagonalizable

1. A necessary and sufficient condition for a matrix $A_{n \times n}$ to be diagonalizable is that the matrix must have n linearly independent eigen vectors.
2. A sufficient (but not necessary) condition for a matrix $A_{n \times n}$ to be diagonalizable is that the matrix must have n linearly independent eigen values.

This is because if a matrix has n linearly independent eigen values then it surely has n linearly independent eigen vectors (although the converse of this is not true).



3

PROBABILITY AND STATISTICS

3.1 Random Experiment

The experiment in which the outcome is uncertain is called a Random Experiment (RE).

Example: Flipping a coin, rolling a pair of dice, Picking a ball from a bag.

3.1.1 Sample Space

The set contains all the possible outcomes of a random experiment. It is denoted by 'S'.

If RE is flipping a coin, $S = \{\text{Head, Tail}\}$

If RE is rolling a dice, $S = \{1,2,3,4,5,6\}$

3.2 Event

Any subset of sample space 'S' is called an Event.

Example: If RE is flipping a coin, then the occurring of a Head is an Event.

If RE is rolling a dice, then getting an odd number is an Event.

3.2.1 Probability of an Event

If 'A' is any event with in the sample space 'S' of a Random experiment, then the probability of event 'A' is given by

$$P(A) = \frac{\text{No. of outcomes favouring event 'A' to happen}}{\text{Total number of elements in 'S'}} = \frac{n(A)}{n(S)}$$

Probability of getting an Even Number when a dice is rolled.

$$P(\text{Even Number}) = \frac{3}{6} = 0.5$$

$$S = \{1,2,3,4,5,6\},$$

$$A = \{2,4,6\}$$

Note: Probability can also be expressed as odds if favour and odds against an event:

- **Odds is favour of an event:**

Odds in **favour** of an event = Number of successes : Number of failures = $m : (n - m)$.

- **Odds against an event:**

Odds against an event = Number of failures : Number of successes = $(n - m) : m$.

3.2.2 Axioms Probability

- (i) If 'A' is any event within the sample space 'S' of a RE, then $0 \leq P(A) \leq 1$

$$\frac{0}{n(S)} \leq \left[\frac{n(A)}{n(S)} \right] \leq \frac{n(S)}{n(S)}$$

\downarrow

$0 \leq P(A) \leq 1$

- (ii) $P(S) = 1$

When a RE is conducted the experiment yields a possible outcome.

3.2.3 Types of Events

(i) Mutually Exclusive Events:

If A, B are two events within a sample space 'S', then A & B are said to be mutually exclusive if $A \cap B = \emptyset$.

Example: If 'A' is the event of getting a prime number when a dice is rolled and 'B' is the event of getting a composite number when a dice is rolled then

$$S = \{1, 2, 3, 4, 5, 6\}, A = \{2, 3, 5\}, B = \{4, 6\} \Rightarrow A \cap B = \emptyset \Rightarrow P(A \cap B) = 0$$

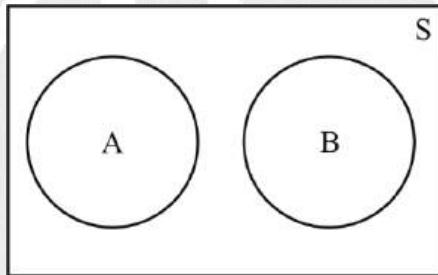


Fig. 5.1. Mutually exclusive event

(ii) Mutually Exhaustive Events:

If 'A', and 'B' are two events within a sample space 'S', then 'A' & 'B' are said to be mutually exhaustive if $A \cup B = S$

Example: If 'A' is the event of getting an odd number when a dice is rolled and 'B' is the event of getting an Even Number, then

$$A \cup B = S$$

$$S = \{1, 2, 3, 4, 5, 6\}$$

$$A = \{1, 3, 5\}, B = \{2, 4, 6\}$$

$$\Rightarrow A \cup B = S$$

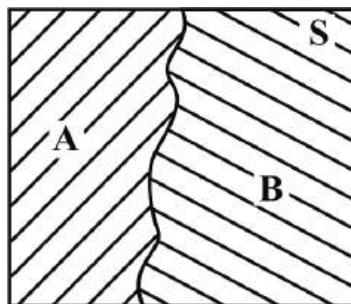
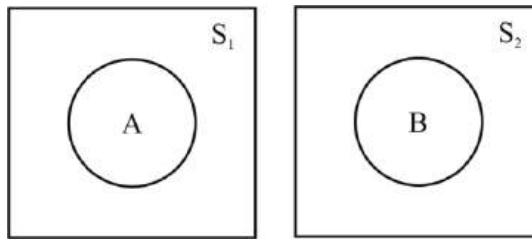


Fig. 5.2. Mutually exhaustive event

(iii) Independent Events:

Two events 'A' & 'B' within the sample space 'S' (or) within two different sample spaces ' S_1 ' & ' S_2 ' are said to be independent if $P(A \cap B) = P(A) \cdot P(B)$.


Fig. 5.3. Independent Event
(iv) Impossible Event (ϕ):

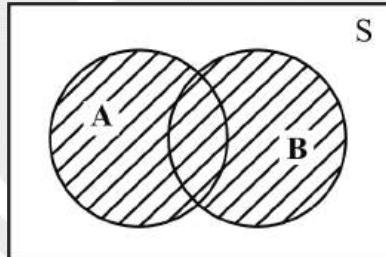
The event with zero probability is called an Impossible Event $P(\phi) = 0$.

3.3 Addition Theorem of Probability

If A, and B are two events with a sample space 'S' of a Random Experiment, then

$$P(A \cup B) = P(A) + P(B) - P(A \cap B)$$

$$\frac{n(A \cup B)}{n(S)} = \frac{n(A)}{n(S)} + \frac{n(B)}{n(S)} - \frac{n(A \cap B)}{n(S)}$$


Fig. 5.4. Addition theorem

$$\Rightarrow P(A \cup B) = P(A) + P(B) - P(A \cap B)$$

When A, and B are mutually exclusive events, $A \cap B = \phi$.

$$\Rightarrow P(A \cap B) = 0$$

$$P(A \cup B) = P(A) + P(B)$$

- If $E_1, E_2, E_3, \dots, E_n$ are mutually exclusive events ($E_i \cap E_j = \phi$), then $P(E_1 \cup E_2 \cup E_3 \cup \dots \cup E_n) = \sum_{i=1}^n P(E_i)$
 $= P(E_1) + P(E_2) + P(E_3) + \dots + P(E_n)$

3.3.1 De Morgan's Law

- $(A \cup B)^C = A^C \cap B^C$
- $(A \cap B)^C = A^C \cup B^C$

3.3.2 Union and Intersection properties

For any two events A and B:

$$(a) P(A \cup B) = P(A) + P(B) - P(A \cap B)$$

$$(b) P(A^c \cap B^c) = 1 - P(A \cup B)$$

For any three events A, B and C:

$$(a) P(A \cup B \cup C) = P(A) + P(B) + P(C) - P(A \cap B) - P(B \cap C) - P(C \cap A) + P(A \cap B \cap C)$$

$$(b) P(A^c \cap B^c \cap C^c) = 1 - P(A \cup B \cup C)$$

3.3.3 Conditional Probability:

The probability of the happening of event 'A' when it is known that event 'B' has already occurred is given by $P(A/B)$

$$P(A/B) = \frac{P(A \cap B)}{P(B)} = \frac{n(A \cap B)}{n(B)}$$

3.3.4 Joint Probability:

- $f(x, y)$ is the joint probability of two RV'S x, y .
- If the two RV are Independent then
 $f(x, y) = f(x) \cdot f(y)$
- $P(a \leq x \leq b, c \leq y \leq d) = \int_a^b \int_c^d f(x, y) dy dx$
- $f(x) = \int_{-\infty}^{\infty} f(x, y) dy$
- $f(y) = \int_{-\infty}^{\infty} f(x, y) dx$

3.3.5 Multiplication Theorem of Probability:

If A, and B are two events within a sample space 'S', then $P(A/B) \cdot P(B) = P(B/A) \cdot P(A)$

$$P(A/B) = \frac{P(A \cap B)}{P(B)} \Rightarrow P(A \cap B) = P(A/B) \cdot P(B) \rightarrow (1)$$

$$P(B/A) = \frac{P(B \cap A)}{P(A)} \Rightarrow P(B \cap A) = P(B/A) \cdot P(A) \rightarrow (2)$$

From (1) & (2)

$$P(A/B) \cdot P(B) = P(B/A) \cdot P(A)$$

3.3.6 Total Theorem of Probability:

If $E_1, E_2, E_3, \dots, E_n$ are 'n' mutually exclusive ($E_i \cap E_j = \emptyset; \forall i \neq j$) and collectively exhaustive event ($E_1 \cup E_2 \cup E_3 \cup \dots \cup E_n = S$) and 'A' is any event with in the sample space 'S', then

$$P(A) = P(E_1) \cdot P(A/E_1) + P(E_2) \cdot P(A/E_2) + \dots + P(E_n) \cdot P(A/E_n)$$

$$P(A) = \sum_{i=1}^n P(E_i) \cdot P(A/E_i)$$

3.3.7 Baye's Theorem

If $E_1, E_2, E_3, \dots, E_n$ are mutually exclusive ($E_i \cap E_j = \emptyset \forall i \neq j$) and collectively exhaustive event ($E_1 \cup E_2 \cup E_3 \cup \dots \cup E_n = S$) and 'A' is any event with in the sample space 'S', then

$$P(E_i/A) = \frac{P(E_i) \cdot P(A/E_i)}{\sum_{i=1}^n P(E_i) \cdot P(A/E_i)}$$

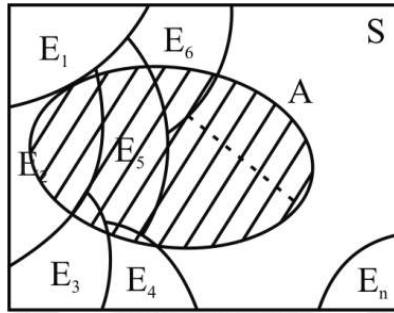


Fig. 5.5. Baye's theorem

3.3.8 Use of permutation and combination

What is combination?

A combination of 'n' objects taken 'r' at a time (r-combination of 'n' objects is an unordered selection of 'r' of the objects).

Number of ways of combining of 'r' object out of 'n' objects without repetition

$${}^n C_r = \frac{n!}{(n-r)!r!}$$

What is permutation?

A combination of 'n' objects taken 'r' at a time (r-combination of 'n' objects is an ordered selection of 'r' of the objects).

Number of ways of selection of r object out of n objects without repetition

$${}^n P_r = \frac{n!}{(n-r)!}$$

Result:

$$(i) \quad {}^n C_r = {}^n C_{n-r}$$

$$(ii) \quad {}^n C_0 + {}^n C_1 + {}^n C_2 + \dots + {}^n C_n = 2^n$$

$$(iii) \quad {}^n C_0 + {}^n C_2 + {}^n C_4 + \dots = 2^{n-1}$$

$$(iv) \quad {}^n C_1 + {}^n C_3 + {}^n C_5 + \dots = 2^{n-1}$$

$$(v) \quad 0. {}^n C_0 + 1. {}^n C_1 + 2. {}^n C_2 + \dots + n. {}^n C_n = n. 2^{n-1}$$

Permutations with Repetition

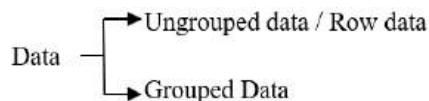
The number of permutations of n objects, where p objects are of one kind, q objects are of another kind and the rest, if any, are of a different kind is $\frac{n!}{p!q!}$.

Combination with Repetition

Number of combinations of ' n ' distinct things taking ' r ' at a time when each thing may be repeated any number of times is given by ${}^{n-1+r}C_r$.

3.4 STATISTICS

Statistics → Collection and Analysis of Data



3.4.1 Analysis of Ungrouped Data

If $x_1, x_2, x_3, \dots, x_n$ are ' n ' observations, then

- (1) The range of the data = $R = \max\{x_1, x_2, \dots, x_n\} - \min\{x_1, x_2, x_3, \dots, x_n\}$
- (2) Arithmetic mean : Mean of the data is equal to sum of observations divided by the total number of observations.

$$\bar{x}(\text{or})\mu = \frac{x_1 + x_2 + \dots + x_n}{n} = \boxed{\frac{\sum_{i=1}^n x_i}{n} = \bar{x} = \mu}$$

- The mean of 1st ' n ' natural numbers = $\frac{\left(\frac{n(n+1)}{2}\right)}{n} = \frac{n+1}{2}$
- The mean of 1st ' n ' odd numbers = $\frac{n^2}{n} = n$
- The mean of 1st ' n ' even numbers = $n + 1$

3.4.2 Median

The middle most observation of the data $(x_1, x_2, x_3, \dots, x_n)$ When the data is arranged in either ascending or descending order.

If $x_1, x_2, x_3, x_4, \dots, x_n$ are ' n ' observations that are arranged in ascending/descending order then

- (i) Median of the Data = $\left(\frac{n+1}{2}\right)^{th}$ observation, if ' n ' is odd.
- (ii) Median of the Data = Mean of $\left(\frac{n}{2}\right)^{th}$ & $\left(\frac{n}{2} + 1\right)^{th}$ observations, if ' n ' is even.

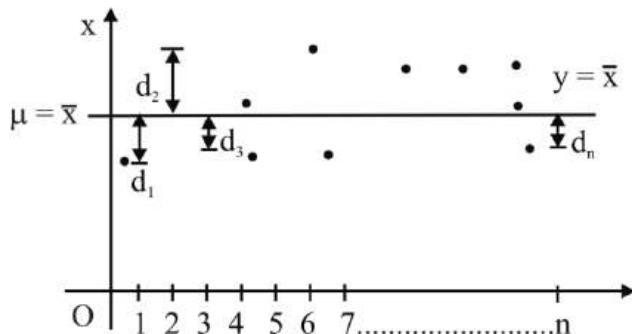
3.4.3 Mode

The observation with highest frequency is called mode.

Any Data with two Modes is called → Bimodal Data

If $x_1, x_2, x_3, \dots, x_n$ are ' n ' data points, $\bar{x} = \mu = \frac{x_1+x_2+\dots+x_n}{n}$

Mean Deviation of the observation $(x_i) = d_i = x_i - \bar{x}$


Fig. 5.6. Discrete data

$$\begin{aligned}
 \text{Sum of derivations of all the observations} &= \sum d_i = (x_1 - \bar{x}) + (x_2 - \bar{x}) + \dots + (x_n - \bar{x}) \\
 &= \sum d_i = (x_1 + x_2 + \dots + x_n) - n\bar{x} \\
 &\boxed{\sum d_i = 0}
 \end{aligned}$$

The sum of mean deviations of all the observations is equal to zero.

3.4.4 Absolute Mean Deviation

If $x_1, x_2, x_3, \dots, x_n$ are 'n' data points with Mean = \bar{x} , then the absolute mean deviation of x_i about \bar{x} is given by $|d_i| = |x - \bar{x}|$. The sum of absolute mean derivations of given data is not zero.

$$(\sum |d_i| \neq 0) \Rightarrow (|x_1 - \bar{x}| + |x_2 - \bar{x}| + \dots + |x_n - \bar{x}| \neq 0)$$

3.4.5 Standard Deviation

If $x_1, x_2, x_3, \dots, x_n$ ('n' is very large), then the standard deviation of the data is given by

$$\text{Population Standard Deviation } \sigma = \sqrt{\frac{1}{n} \sum (x_i - \bar{x})^2}, \quad n \rightarrow \text{size of population}$$

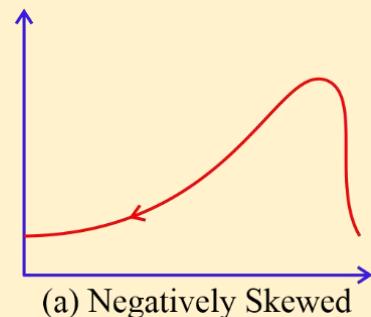
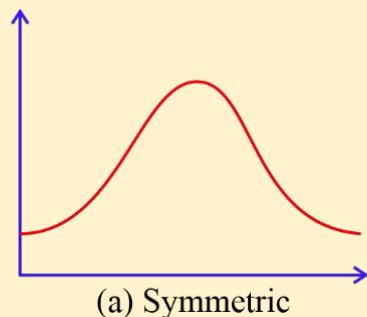
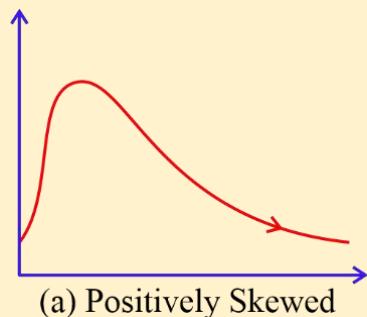
$$\text{Sample Standard derivation: } \sigma = \sqrt{\frac{1}{(n-1)} \sum (x_i - \bar{x})^2}, \quad n \rightarrow \text{size of sample}$$

Generally ($n > 29 \rightarrow \text{population}$) ($n < 29 \rightarrow \text{sample}$)

Note: Measures of skewness (The degree of asymmetry)

A lack of symmetry is skewness.

- For symmetric distribution mean (M) = Median (M_d) = Mode (M_e)
- For negatively skewed distribution mean (M) < Median (M_d) < Mode (M_e)
- For positively skewed distribution Mean (M) > Median (M_d) > Mode (M_e).



3.5 Random Variables

The variable that connects the outcome of a Random Experiment to a real number.

Example: 'x' is the value of the number that a dice shows when it is rolled.

Discrete RV → The RV whose value is obtained by counting, defined by PMF

Random Variable

Continuous RV → The RV whose value is obtained by Measuring, defined by PDF

- If a data consists of ' f_1 ' data points with value ' x_1 ', ' f_2 ' data points with value ' x_2 ', ..., ' f_n ' data point with value ' x_n ', then
 - Expectation of 'x' = $E(x) = \sum_{i=1}^n x_i P(x = x_i)$
 - Variance of 'x' = $\sigma^2 = E(x^2) - (E(x))^2$ and σ is the standard deviation.

3.5.1 Probability Mass Function (PMF)

The PMF $p(x)$ of a discrete random variable X taking values x_1, x_2, \dots, x_n is defined such that,

$$(i) p(x_i) \geq 0$$

$$(ii) \sum_{i=1}^n p(x_i) = 1$$

$$(iii) p(x_i) = p(X = x_i)$$

3.5.2 Probability Density Function (PDF)

The pdf $f(x)$ of a continuous random variable X is defined such that,

$$(i) f(x) \geq 0$$

$$(ii) \int_{-\infty}^{\infty} f(x) dx = 1$$

$$(iii) P(a < X < b) = \int_a^b f(x) dx$$

3.5.3 Expected Value

- Expected value of a random variable X, $E[X]$, is defined as, $E[X] = \begin{cases} \sum xp(x); & X \text{ is discrete rv} \\ \int_{-\infty}^{\infty} xf(x) dx; & X \text{ is continuous rv} \end{cases}$
- Expected value of X^2 is,

$$E[X^2] = \begin{cases} \sum x^2 p(x); & X \text{ is discrete rv} \\ \int_{-\infty}^{\infty} x^2 f(x) dx; & X \text{ is continuous rv} \end{cases}$$

Note: $E[X^n]$ is called nth moment.

3.5.4 Mean of Random Variable 'X'

Mean = $\mu = E[X]$

3.5.5 Variance of a Random Variable 'X'

$$\text{Var}(X) = E[(X - \mu)^2]$$

$$\text{Or, } \text{Var}(X) = E[X^2] - \mu^2$$

3.5.6 Properties of Expectation

- (i) $E[c] = c$, c is a constant.
- (ii) $E[ax] = aE[X]$
- (iii) $E(aX + b) = aE(X) + b$
- (iv) If X and Y are random variable $E[X \pm Y] = E(x) \pm E(Y)$.
- (v) If X and Y are random variables $E(X, Y) = E(X) \cdot E(Y / X)$.
- (vi) If X and Y independent random variables $E(X, Y) = E(X) \cdot E(Y)$.

3.5.7 Properties of Variance

- (i) $\text{Var}[C] = 0$, C is constant.
- (ii) $\text{Var}(aX) = a^2 V(X)$ where X is random variable and ' a ' constant.

$\text{Var}(-X) = (-1)^2 \text{Var}(X) = \text{Var}(X)$ Variance is always positive.

- (iii) $\text{Var}(ax + b) = a^2 \text{Var}(X) + 0$
- (iv) If X and Y are independent random variables.

$$\text{Var}(X + Y) = \text{Var}(X) + \text{Var}(Y)$$

$$\text{Var}(X - Y) = \text{Var}(X) + \text{Var}(Y)$$

- (v) $\text{Var}(ax + by) = a^2 v(x) + b^2 v(y) + 2ab \text{Cov}(x, y)$
- (vi) $\text{Cov}(x, y) = E(x, y) - E(x) E(y)$
- (vii) For independent random variables $\text{Cov}(x, y) = 0$

3.5.8 Continuous RV

The value of the Random Variable is obtained by Measuring.

3.6 Probability Distribution Function (PDF)

A continuous & differentiable function $P(x)$ is said to be a probability distribution/density function of a continuous random variable 'x' if $P(a \leq x \leq b) = \int_a^b P(x)dx$

3.6.1 Mean (or) Expectation

If $P(x)$ is a probability distribution/density function of a continuous Random Variable 'x' then the Mean of 'x' = $E(x) = \int_{-\infty}^{\infty} x \cdot P(x)dx$

3.6.2 Median

The value of 'x' for which the total probability is exactly divided into two equal halves is called Median.

3.6.3 Mode

The value of 'x' at which $P(x)$ is maximum is called mode.

3.6.4 Variance

$$= \sigma^2 = E(x^2) - (E(x))^2$$

$$\Rightarrow \sigma^2 = \int_{-\infty}^{\infty} x^2 \cdot P(x)dx - \left(\int_{-\infty}^{\infty} x \cdot P(x)dx \right)^2$$

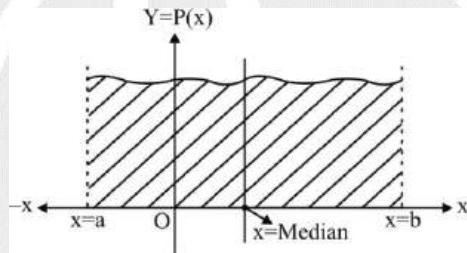


Fig. 5.7. Continuous random variables

3.7 Continuous RV distributions

(1) Gaussian/Normal Distribution:

If 'x' is a continuous Random variable with mean ' μ ' and standard deviation ' σ ', then the probability distribution/density function of normally distributed variable 'x' is given by

$$P(x) = \frac{1}{\sigma\sqrt{2\pi}} \cdot e^{-\frac{(x-\mu)^2}{2\sigma^2}}$$

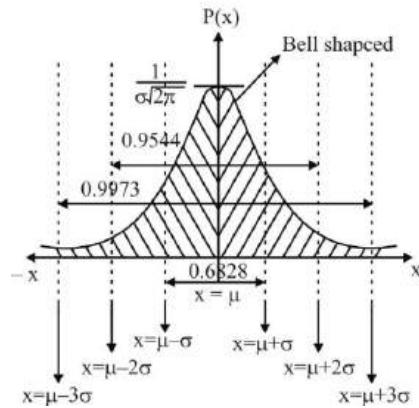


Fig. 5.8. Normal distribution

$$\begin{aligned} \text{Mean} &= \text{Median} = \text{Mode} = \mu \\ P(\mu - \sigma \leq x \leq \mu + \sigma) &= 0.6828 \\ P(\mu - 2\sigma \leq x \leq \mu + 2\sigma) &= 0.9544 \\ P(\mu - 3\sigma \leq x \leq \mu + 3\sigma) &= 0.9973 \\ P(x) &= \frac{1}{\sigma \cdot \sqrt{2\pi}} \cdot e^{\frac{-(x-\mu)^2}{2\sigma^2}} \end{aligned}$$

(2) Standard Normal Distribution:

$$\text{Assuming } z = \frac{x-\mu}{\sigma}; \mu = 0; \sigma = 1, P(z) = \frac{1}{\sqrt{2\pi}} \cdot e^{\frac{-z^2}{2}}$$

$$P(-1 \leq z \leq 1) = 0.6828$$

$$P(-2 \leq z \leq 2) = 0.9544$$

$$P(-3 \leq z \leq 3) = 0.9973$$

Note:

1. The normal distribution curve is bell shaped curve
2. The points of inflection of the normal distribution curve are at $x = \mu + \sigma$ and $x = \mu - \sigma$.
3. The cumulative function graph is of 'S' Shape.
4. For a given normal distribution, Mean = median = Mode

(3) Uniform Distribution:

If 'x' is a uniformly distributed random variable such that $a \leq x \leq b$ then the Pdf is given by

$$P(x) = \frac{1}{(b-a)}$$

$$\text{Mean} = \int_a^b x \cdot P(x) dx = \int_a^b x \cdot \frac{1}{b-a} dx = \frac{1}{(b-a)} \int_a^b x \cdot dx$$

$$\left(\frac{b+a}{2} \right) = \text{Mean}$$

$$\Rightarrow \text{Variance} = \sigma^2 = \frac{(b-a)^2}{12}$$

$$\text{Std. deviation} = \sigma = \frac{(b-a)}{\sqrt{12}}$$

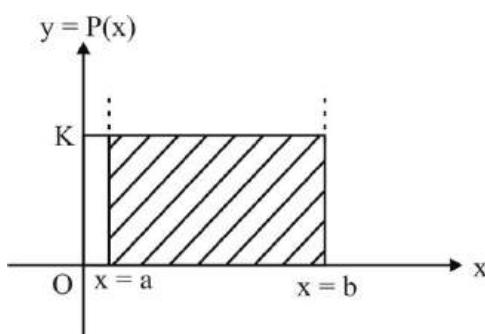


Fig. 5.9. Uniform Distribution

3.7.1 Properties of Mean and Variance:

$$E(ax + by) = a \cdot E(x) + b \cdot E(y)$$

$$V(ax + by) = a^2 \cdot V(x) + b^2 \cdot V(y) - 2abCOV(x, y)$$

where $COV(x, y) = E(xy) - E(x) \cdot E(y)$

If x, y are independent random variables, then $E(xy) = E(x) \cdot E(y) \Rightarrow COV(x, y) = 0$

(1) Exponential Distribution:

If 'x' is a continuous random variable with mean as $\frac{1}{\lambda}$ then the exponential distribution of 'x' is given by the function

$$f(x) = \begin{cases} \lambda \cdot e^{-\lambda x} & ; x \geq 0 \\ 0 & : \text{otherwise} \end{cases}$$

$$\text{Mean} = \frac{1}{\lambda}$$

$$\sigma^2 = \frac{1}{\lambda^2}$$

$\text{Mean} = \text{Standard Deviation} = \frac{1}{\lambda}$

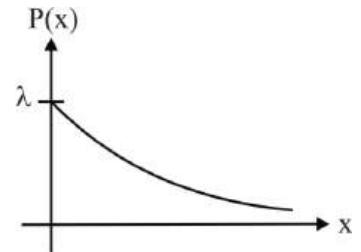


Fig. 5.10. Exponential distribution

3.8 Discrete Random Variable Distributions

If a Random experiment has **only two Possible outcomes**, (one is Success & other is failure) and the Probability of Success doesn't depend on time, then the probability of occurring of exactly 'r'-successes' in 'n-trials' is given by

$$P(X = r) = {}^n C_r \cdot P^r \cdot q^{n-r}$$

Where, P → Probability of Success,

q → Probability of Failure

$$p + q = 1$$

$$\text{Mean} = np, \text{Variance} = npq = \sigma^2, \text{standard deviation} = \sigma = \sqrt{npq}$$

3.8.1 Poisson Distribution

If a random experiment has only two possible outcomes, and the average number of successes in a given time 't' is λ , then the probability that exactly 'r' successes occur within the same time 't' given by

$$P(x = r) = \frac{e^{-\lambda} \cdot \lambda^r}{r!}$$

$$\text{Mean} = \lambda.$$

$$\text{Mean} = \text{Variance} = \lambda$$

$$\Rightarrow \sigma = \sqrt{\lambda}$$



GATE Exam 2024?



PARAKRAM BATCH

ME | CE | EE | EC | CS & IT

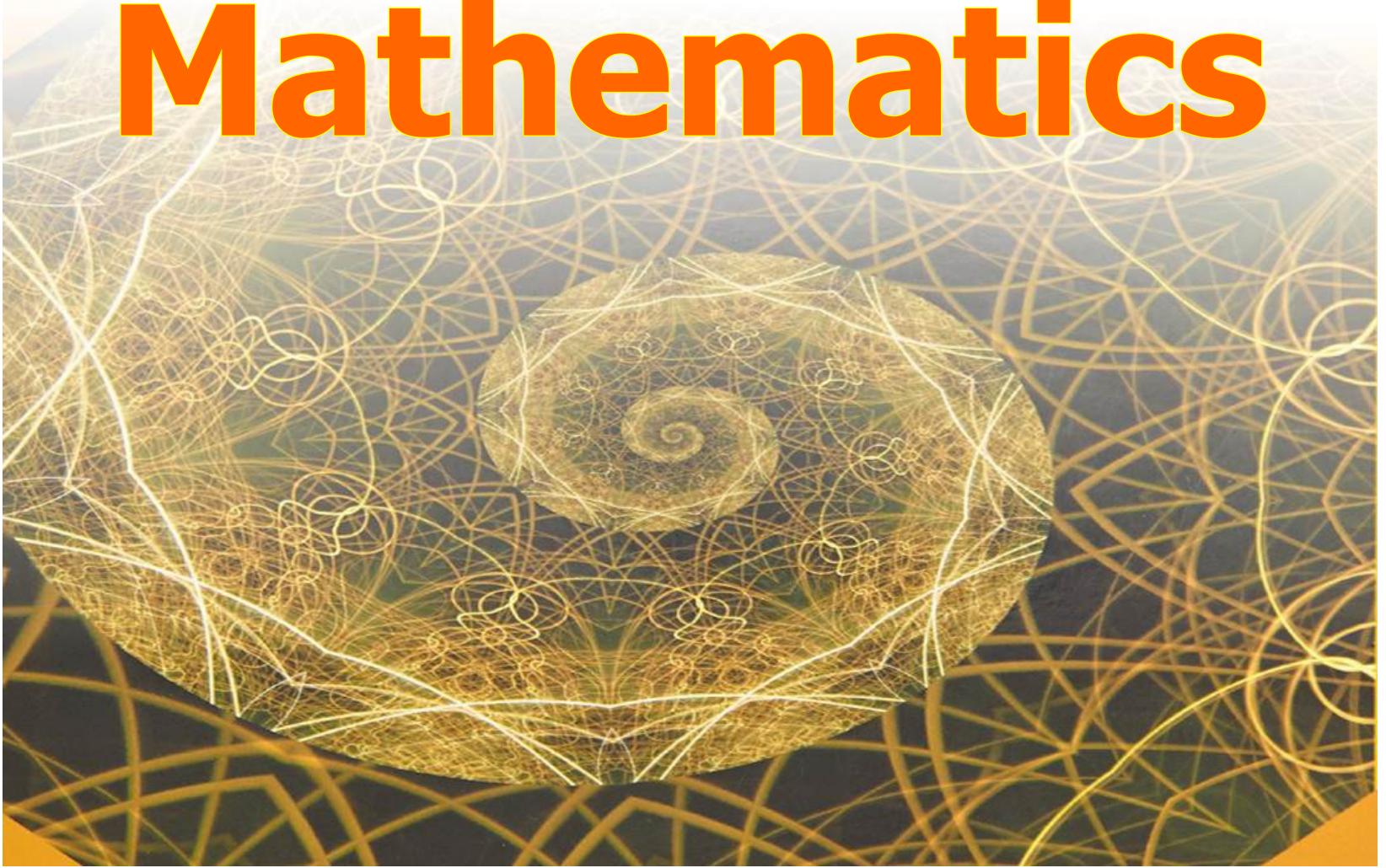
- Live Classes & Recorded Videos
- DPPs & Solutions
- Doubt Solving
- Batches are available in **Hindi**

Weekday & Weekend
Batches Available



JOIN NOW!

Discrete Mathematics



Discrete Mathematics

INDEX

1. Graph Theory 2.1 – 2.10
2. Logic Handbook 2.11 – 2.17
3. Set Theory 2.18 – 2.35
4. Combinatorics 2.36 – 2.40



1

GRAPH THEORY

1.1 Basics of Graphs:

The number of vertices of odd degree in a graph is always even.

Every graph has an even number of odd vertices.

- Based on Parallel edges & self-loops graphs are classified into 3 types

	Parallel graph	Self-loops
Simple graph	✗	✗
Multi graph	✓	✗
Pseudograph	✓	✓

(We mainly discuss simple graph in our syllabus)

1.1.1 Theorem1:

Maximum degree of a vertex, in a simple graph with n vertices, is $\leq n - 1$.

Note:

Hand Shaking Lemma: Sum of all degrees = $2 \times$ sum of all edges.

1.1.2 Theorem2 :

Maximum no. of edges, in a simple graph with n vertices, is $\leq n_{c_2} = \frac{n(n-1)}{2}$

Note: No of different graphs possible with n distinct vertices is $= 2^{\frac{n(n-1)}{2}}$

No. of different graphs possible with n distinct vertices and ' e ' edges is $\left[\frac{n(n-1)}{2} \right]_{C_e}$

1.1.3 Degree sequence:

If the degrees of a graph are written in increasing order or decreasing order, we call it a degree sequence

- Not all degree sequence forms simple graph.
- The degree sequence which forms simple graph is called graphical

1.1.4 Theorem 3:

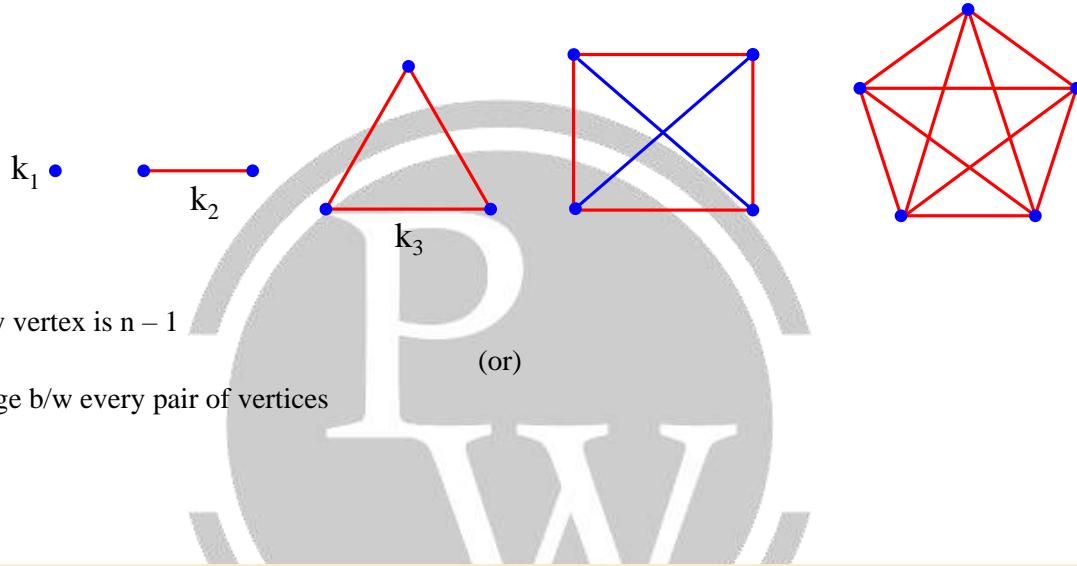
In a simple graph at least two vertices have same degree ($n \geq 2$)

Example: {5, 4, 3, 2, 1} is not graphical

1.1.5 Theorem 4:

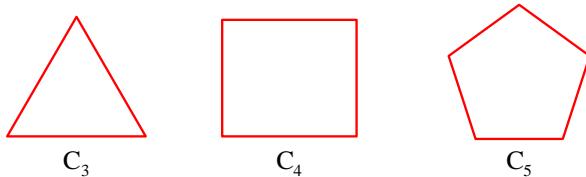
Max degrees in a given graph G is denoted as $\Delta(G)$ & Min degree is denoted as $\delta(G)$

$$\delta(G) \leq \frac{2e}{n} \leq \Delta(G) \leq n - 1$$

1.1.6 Complete Graph (k_n) ($n \geq 1$):**1.1.7 Regular Graph:**

A graph in which degree of all vertices is same is called a regular graph.

$$n.\delta(G) = 2e = n.\Delta(G)$$

1.1.8 Cycle Graph (C_n) ($n \geq 3$):

If given degree sequence is all 2's, then it's not guaranteed that it's a cycle graph

$$G: [2, 2, 2, 2, 2, 2]$$

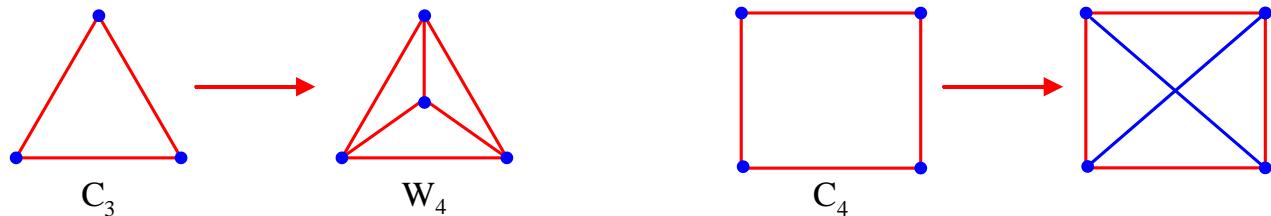


Degree of all vertices is 2 in a cycle graph

- Every C_n is a regular graph
- Number of Edges = n = Number of Vertex

1.1.9 Wheel Graph (W_n) ($n \geq 4$):

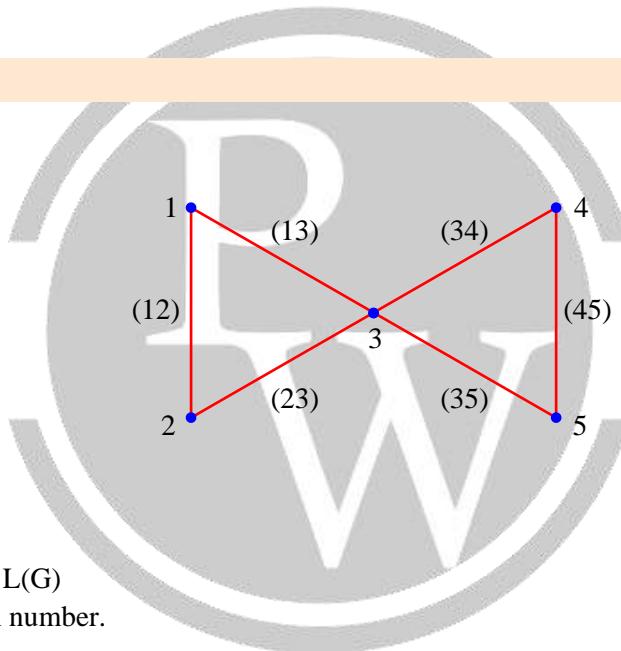
A wheel graph W_n is obtained by adding a vertex (hub) to C_{n-1} (Cycle Graph) such that this vertex is adjacent to all the other vertices.



Number of edges = $2(n - 1)$

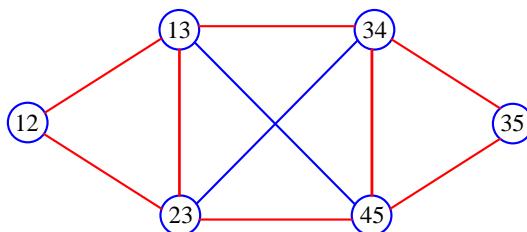
1.1.10 Line Graph ($L(G)$):

Consider below graph, G

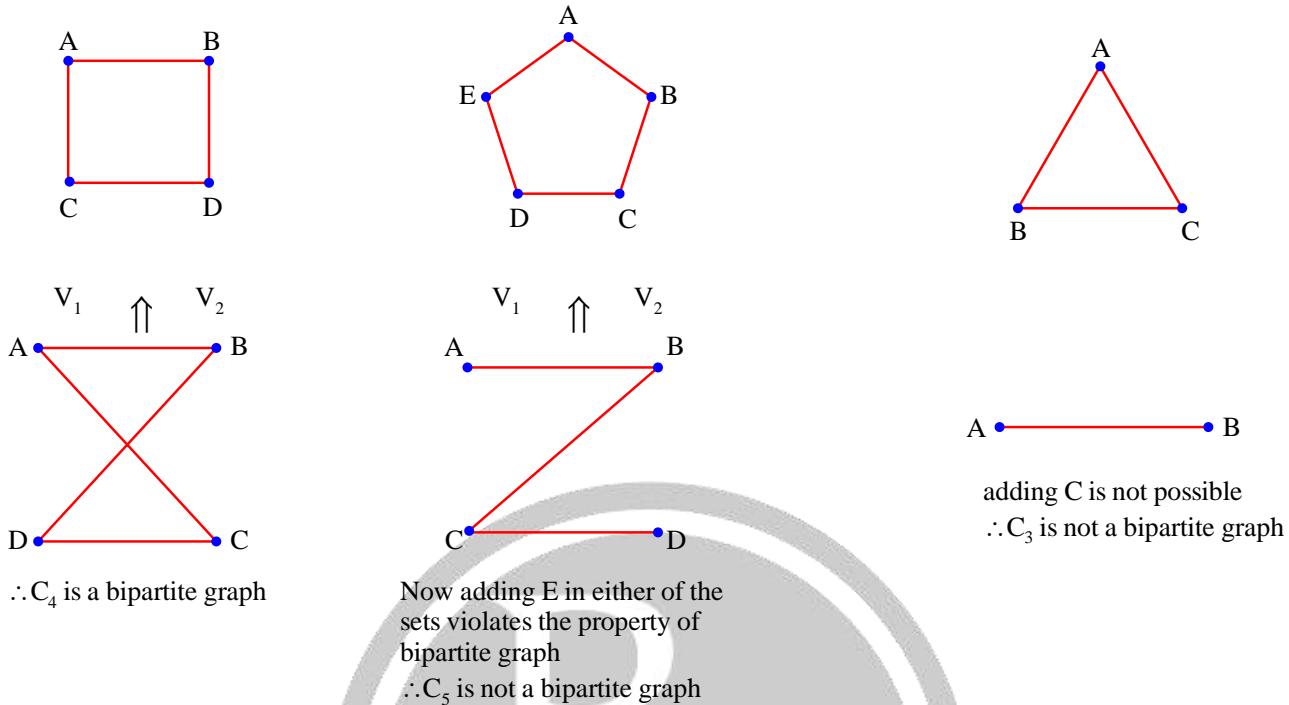


Step to construct $L(G)$

- (i) define edges in G
- (ii) label these edges as vertices in $L(G)$
- (iii) Connect vertices with common number.



Line graph of every cycle graph is also a cycle.

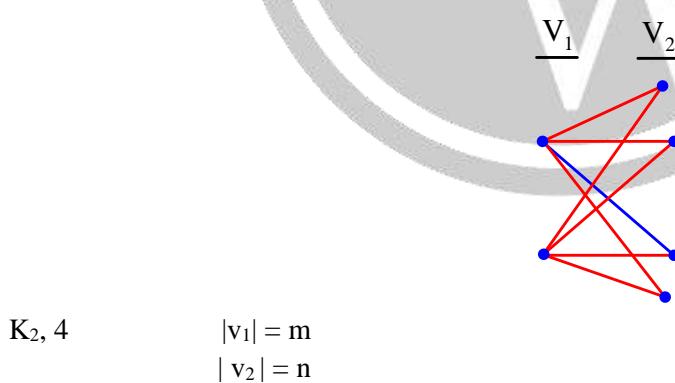
1.1.11 Bipartite ($L(G)$):


Note: Bi – partite graphs do not contain odd length cycle.

1.1.12 Complete bipartite graph ($K_{m,n}$) :

Each vertex in set V_1 is adjacent to all vertices in set V_2 .

Example:



$K_{2, 4}$

$$\begin{aligned} |V_1| &= m \\ |V_2| &= n \end{aligned}$$

In $K_{m,n}$ number of vertices = $m + n$

number of edges = mn

$$\Delta(K_{m,n}) = \max(m, n), \quad \delta(K_{m,n}) = \min(m, n)$$

1.1.13 Theorem 5 :

- Maximum no. of edges possible in bipartite graph of n vertices is $\leq \left\lfloor \frac{n^2}{4} \right\rfloor$

1.1.14 Star graph ($k_{1,n-1}$) :

It is complete bipartite graph with one vertex in one set and rest of the vertices in other set.

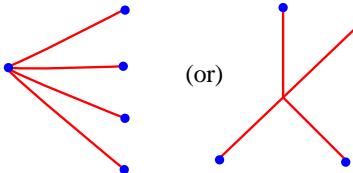
(or)

Star graph ($k_{1, n-1}$) is complete bipartite graph possible with n vertices and minimum no. of edges.

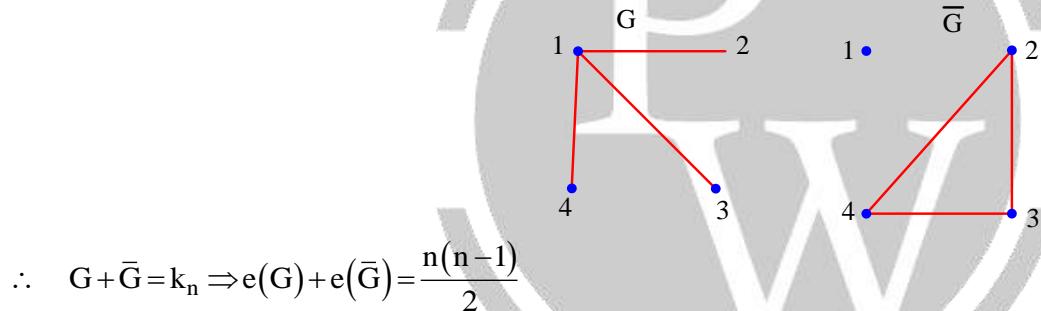
Eg: Star graph of 5 vertices, $k_{1, 4}$ is

total no of edges in $k_{1, n-1} = n - 1$

$$\Delta(k_{1, n-1}) = n - 1, \delta(k_{1, n-1}) = 1$$

**1.1.15 Complement graph (\bar{G}) :**

For a graph G , complement of $G(\bar{G})$ is the graph which contains all the vertices present in G and does not contain the edges present in G .



If the degree of vertex v is x in graph G , then the degree of vertex v in \bar{G} is $[(n-1)-x]$

If d_1, d_2, \dots, d_n is degree sequence for G then

$(n-1-d_1), (n-1-d_2), \dots, (n-1-d_n)$ is degree sequence of \bar{G}

Example:

consider a graph of degree sequence $\{5, 2, 2, 2, 2, 1\}$.

What is the degree sequence of complement graph?

Total vertices, $n = 6$

$$k_6 \rightarrow 5, 5, 5, 5, 5, 5$$

$$G \rightarrow 5, 2, 2, 2, 2, 1$$

$$\bar{G} \rightarrow \{0, 3, 3, 3, 3, 4\}$$

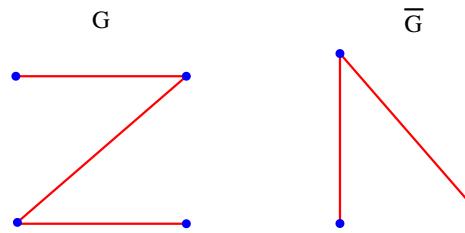
Isomorphic Graphs:

Let $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$ be two undirected graphs. A function $f: V_1 \rightarrow V_2$ is called a graph isomorphism if (a) f is one-to-one and onto, and (b) for all $a, b \in V_1$. $\{a, b\} \in E_1$ if and only if $\{f(a), f(b)\} \in E_2$. When such a function exists, G_1 and G_2 are called isomorphic graphs.

(ii) Self-complement: $(G \equiv \bar{G})$

It is a graph which is isomorphic to its own complement.

i.e., $G = \bar{G}$



It is clear that above two graphs are self-complement to each other.

w.r.t

$$e(G) + e(\bar{G}) = \frac{n(n-1)}{2}$$

Let e be no. of edges in G

$$e = \frac{n(n-1)}{2}$$

- $e = \frac{n(n-1)}{4}$ i.e., no. of edges in a self-complement graph

Complement of star graph $K_{1, n-1}$ gives one isolated vertex and a complete graph K_{n-1}
 n must be congruent to 0 or 1 mod 4.

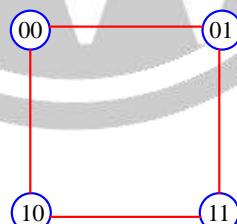
Hypercube (Q_n):

Q_1

2' vertices – 0, 1

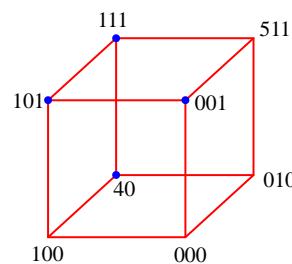
Q_2

4' vertices – 00, 01, 10, 11



Every cycle in hypercube B of even length (think why). So, every hypercube is bipartite graph

Q_3 : 8 vertices



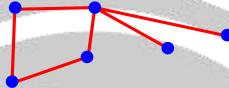
$$\text{Number of edges} = \frac{n \times 2^n}{2} = n \times 2^{n-1}$$

1.2 Connectivity

Name	Repeated Vertex (Vertices)	Repeated Edge(s)	Open	Closed
Walk (open)	Yes	Yes	Yes	
Walk (closed)	Yes	Yes		Yes
Trail	Yes	No	Yes	
Circuit	Yes	No		Yes
Path	No	No	Yes	
Cycle	No	No		Yes

1.2.1 Connected graph:

For every two pair of vertices, there must exist a path between them.

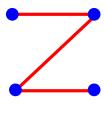


1.2.2 Disconnected graph:

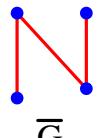
If we can find at least one pair of vertices, such that there is no path available b/w these 2 verities, then the graph is said to be disconnected graph.

Disconnected graph

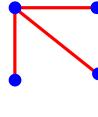
- If G is connected then \bar{G} may be connected or disconnected



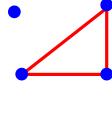
Connected



Connected



Connected

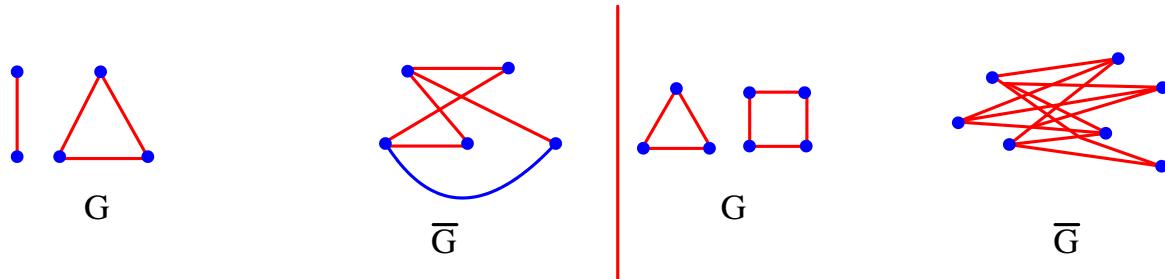


\bar{G} disconnected

1.2.3 Theorem 6:

If G is disconnected, then \bar{G} is connected.

Example:

**1.2.4 Range of edges for a connected graph ($k=1$):**

(k is connected components)

Tree

- Minimum no of edges required to get a possibility to make graph connected with n vertices is $n-1$.
- $$(n-1)_{\text{Tree}} \leq e \leq \frac{n(n-1)}{2}$$
- The connected graph with $n-1$ edges is doesn't have a cycle.
 - This graph is known as minimally connected graph.
 - In this kind of graph, there will be a unique path b/w any two pair of vertices.
 - This kind of graph is called a tree (a connected graph with no cycles)

1.2.5 Range of edges for a disconnected graph:

- Edges range b/w

$$n-k \leq e \leq \frac{(n-k)(n-k+1)}{2}$$

Proof:

Here lets say we have k components with n_1, n_2, \dots, n_k components

$$\therefore n_1 + n_2 + \dots + n_k : k$$

For min no. of edge, each component must be minimally connected.

\therefore min no of edges is

$$\begin{aligned} N_1 - 1 + n_2 - 1 + \dots + n_k - 1 \\ = (n_1 + n_2 + \dots + n_k) - k = n - k \end{aligned}$$

Note:

1. Let G be a graph of order n . If

$$\deg u + \deg v \geq n - 1$$

nonadjacent vertices u and v of G , then G is connected and $\text{diam}(G) \leq 2$.

2. If G is a graph of order n with $\delta(G) \geq (n-1)/2$, then G is connected.

3. A directed graph is strong connected if there is a path from a to b and from b to a whenever a and b are vertices in the graph.

4. A directed graph is weakly connected if there is a path between every two vertices in the underlying undirected graph.

5. $k(G) \leq \lambda(G) \leq \min_{v \in V} \deg(v)$.

6. Simple graph G with n vertices is connected if it has more than $(n - 1)(n - 2)/2$ edges.
7. Let $G = (V, E)$ be a loop-free graph with $n (\geq 2)$ vertices. If $\deg(v) \geq (n - 1)/2$ for all $v \in V$, then G has a Hamilton path.
8. If $G = (V, E)$ is a loop-free undirected graph with $|V| = n \geq 3$, and if $|E| \geq \binom{n-1}{2} + 2$, then G has a Hamilton cycle.
9. If $G = (V, E)$ is a loop-free undirected graph with $|V| = n \geq 3$, and $\deg(v) \geq n/2$ for all $v \in V$, then G has a Hamilton cycle.
- (10) If G_1, G_2 are (loop-free) undirected graphs, G_1, G_2 are isomorphic if and only if \bar{G}_1, \bar{G}_2 are isomorphic.
- (11) If G is an undirected graph or multigraph with no isolated vertices, then we can construct an Euler trail in G if and only if G is connected and has exactly two vertices of odd degree.
- (12) $k(G) \leq \lambda(G) \leq \delta(G) \leq \Delta(G) \leq n - 1$

1.3 Planarity

A graph (or multigraph) G is called planar if G can be drawn in the plane with its edges intersecting only at vertices of G. Such a drawing of G is called an embedding of G in the plane.

Kuratowski's Theorem. A graph is nonplanar if and only if it contains a subgraph that is homomorphic to either K_5 or $K_{3,3}$.

Let $G = (V, E)$ be a connected planar graph or multigraph with $|V| = v$ and $|E| = e$. Let r be the number of regions in the plane determined by a planar embedding (or, depiction) of G; one of these regions has infinite area and is called the infinite region. Then $v - e + r = 2$.

Let $G = (V, E)$ be a loop-free connected planar graph with $|V| = v$, $|E| = e > 2$, and r regions. Then $3r \leq 2e$ and $e \leq 3v - 6$.

1.3.1 Coloring:

Note: Every MIS will always be MDS. But reverse need not to be true

Domination number \leq Independence number.

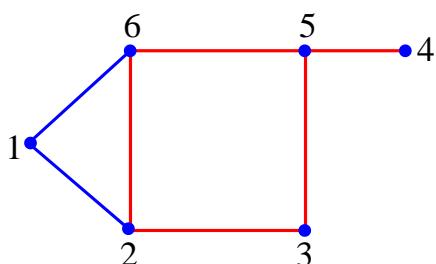
$$\alpha(G) = \leq \beta(G)$$

1.3.2 Theorem 7 :

Sum of size of minimum vertex cover and size of maximum independent set is equal to number of vertices

1.4 Covering:

It is set of edges such that all vertices should incident on at least one edge.



{16, 12, 65, 53, 54}
{16, 54, 23}, {16, 12, 53, 54}
{12, 16, 65, 23, 53, 62, 54}

Set of all edge is also a covering set

- It is also known as edge covering set.

1.4.1 Minimal Covering set:

It is a covering set from which we can't remove new elements(edge).

{16, 12, 53, 54} . {16, 54, 23} are MCS

1.4.2 Covering Number (C(G)):

It is no of edges present smallest covering set.

For above graph $C(G) = 3$

1.5 Perfect Matching:

A matching is said to be perfect matching if every vertex in the graph is matched

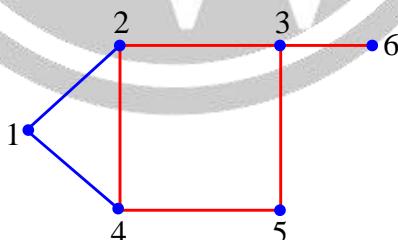
(or)

Induced degree of all the vertices is 1.

1.5.1 Induced degree:

The degree of a vertex in a matching is called induced degree

Example:



{12, 45, 36} is perfect matching

Note: Every perfect matching is maximal, but reverse need not to be true.

If perfect matching exists, then no. of vertices will always be even but reverse need not to be true.

A graph may contain more than one perfect matching.

Total no. of perfect matching possible for a complete graph with $2n$ vertices is $\frac{(2n)!}{2^n \cdot n!}$



2

LOGIC HANDBOOK

2.1 Introduction

p	$\neg p$
T	F
F	T

p	q	$p \wedge q$
T	T	T
T	F	F
F	T	F
F	F	F

p	q	$p \vee q$
T	T	T
T	F	T
F	T	T
F	F	F

Let p and q be propositions. The conditional statement $p \rightarrow q$ is the proposition “if p , then q .”

The conditional statement $p \rightarrow q$ is false when p is true and q is false, and true otherwise. In the conditional statement $p \rightarrow q$, p is called the hypothesis (or *antecedent* or *premise*) and q is called the conclusion (or *consequence*).

- | | |
|---|--|
| “if p , then q ” | “ p implies q ” |
| “if p , q ” | “ p only if q ” |
| “ p is sufficient for q ” | “a sufficient condition for q is p ” |
| “ q if p ” | “ q whenever p ” |
| “ q when p ” | “ q is necessary for p ” |
| “a necessary condition for p is q ” | “ q follows from p ” |
| “ q unless $\neg p$ ” | “ q provided that p ” |

p	q	$p \rightarrow q$
T	T	T
T	F	F
F	T	T
F	F	T

p	q	$p \leftrightarrow q$
T	T	T
T	F	F
F	T	F
F	F	T

Let p and q be propositions. The biconditional statement $p \leftrightarrow q$ is the proposition “ p if and only if q ”. The biconditional statement $p \leftrightarrow q$ is true when p and q have the same truth values, and is false otherwise. Biconditional statements are also called bi-implications.

<i>Operator</i>	<i>Precedence</i>
\neg	1
\wedge	2
\vee	3
\rightarrow	4
\leftrightarrow	5

A compound proposition that is always true, no matter what the truth values of the propositional variables that occur in it, is called a tautology. A compound proposition that is always false is called a contradiction. A compound proposition that is neither a tautology nor a contradiction is called contingency.

- Every contingency is satisfiable, but reverse need not to be true 1.
- Every tautology is satisfiable, but reverse need not to be true.

For any primitive statements p, q, r , any tautology ‘T’ and any contradiction ‘F’.

- (1) $\neg\neg p \Leftrightarrow p$ Law of Double Negation
- (2) $\neg(p \vee q) \Leftrightarrow \neg p \wedge \neg q$ DeMorgan’s Laws
- $\neg(p \wedge q) \Leftrightarrow \neg p \vee \neg q$
- (3) $p \vee q \Leftrightarrow q \vee p$ Commutative Laws
- $p \wedge q \Leftrightarrow q \wedge p$
- (4) $p \vee (q \vee r) \Leftrightarrow (p \vee q) \vee r$ Associative Laws
- $p \wedge (q \wedge r) \Leftrightarrow (p \wedge q) \wedge r$
- (5) $p \vee (q \wedge r) \Leftrightarrow (p \vee q) \wedge (p \vee r)$ Distributive Laws
- $p \wedge (q \vee r) \Leftrightarrow (p \wedge q) \vee (p \wedge r)$
- (6) $p \vee p \Leftrightarrow p$ Idempotent Laws

- $p \wedge p \Leftrightarrow p$
- (7) $p \vee F \Leftrightarrow p$ Identity Laws
- $p \vee T \Leftrightarrow p$
- (8) $p \vee \neg p \Leftrightarrow T$ Inverse Laws
- $p \wedge \neg p \Leftrightarrow F$
- (9) $p \vee T \Leftrightarrow T$ Domination Laws
- $p \wedge F \Leftrightarrow F$
- (10) $p \vee (p \wedge q) \Leftrightarrow p$ Absorption Laws
- $p \vee (p \wedge q) \Leftrightarrow p$

$p \rightarrow q \equiv \neg p \vee q$
$p \rightarrow q \equiv \neg q \rightarrow \neg p$
$p \vee q \equiv \neg p \rightarrow \neg q$
$p \wedge q \equiv \neg (p \rightarrow \neg q)$
$\neg (p \rightarrow q) \equiv p \wedge \neg q$
$(p \rightarrow q) \wedge (p \rightarrow r) \equiv p \rightarrow (q \wedge r)$
$(p \rightarrow r) \wedge (q \rightarrow r) \equiv (p \vee q) \rightarrow r$
$(p \rightarrow q) \vee (p \rightarrow r) \equiv p \rightarrow (q \vee r)$
$(p \rightarrow r) \vee (q \rightarrow r) \equiv (p \wedge q) \rightarrow r$
$p \leftrightarrow q \equiv (p \rightarrow q) \wedge (q \rightarrow p)$
$p \leftrightarrow q \equiv \neg p \leftrightarrow \neg q$
$p \leftrightarrow q \equiv (p \wedge q) \vee (\neg p \wedge \neg q)$
$\neg (p \leftrightarrow q) \equiv p \leftrightarrow \neg q$

Rule of Inference	Related Logical Implications	Name of Rule
1) $\frac{p}{p \wedge (p \rightarrow q)} \rightarrow q$	$[p \wedge (p \rightarrow q)] \rightarrow q$	Rule of Detachment (Modus Ponens)
2) $\frac{p \rightarrow q \quad q \rightarrow r}{p \rightarrow r}$	$[(p \rightarrow q) \wedge (q \rightarrow r)] \rightarrow (p \rightarrow r)$	Law of the syllogism
3) $\frac{p \rightarrow q \quad \neg q}{\neg p}$	$[(p \rightarrow q) \wedge \neg q] \rightarrow \neg p$	Modus Tollens

4) $\frac{p}{\begin{array}{c} q \\ \therefore p \wedge q \end{array}}$ 5) $\frac{p \vee q}{\begin{array}{c} \neg p \\ \therefore q \end{array}}$ 6) $\frac{\neg p \rightarrow F}{\begin{array}{c} F \\ \therefore p \end{array}}$ 7) $\frac{p \wedge q}{\begin{array}{c} p \\ \therefore p \end{array}}$ 8) $\frac{p}{\begin{array}{c} p \vee q \\ \therefore p \vee q \end{array}}$ 9) $\frac{p \rightarrow (q \rightarrow r)}{\begin{array}{c} p \wedge q \\ \therefore r \end{array}}$ 10) $\frac{p \rightarrow q}{\begin{array}{c} q \rightarrow r \\ \therefore (p \vee q) \rightarrow r \end{array}}$ 11) $\frac{p \vee r}{\begin{array}{c} p \rightarrow q \\ r \rightarrow s \\ \therefore q \vee s \end{array}}$ 12) $\frac{p \rightarrow q}{\begin{array}{c} r \rightarrow s \\ \neg q \vee \neg s \\ \therefore \neg p \vee \neg r \end{array}}$	$[(p \vee q) \wedge \neg p] \rightarrow q$ $(\neg p \rightarrow F_0) \rightarrow p$ $(p \wedge q) \rightarrow p$ $p \rightarrow p \vee q$ $[(p \wedge q) \wedge [p \rightarrow (q \rightarrow r)]] \rightarrow r$ $[(p \rightarrow r) \wedge (q \rightarrow r)] \rightarrow [(p \vee q) \rightarrow r]$ $[(p \rightarrow q) \wedge (r \rightarrow s) \wedge (p \vee r)] \rightarrow (q \vee s)$ $[(p \rightarrow q) \wedge (r \rightarrow s) \wedge (\neg q \vee \neg s)] \rightarrow (\neg p \vee \neg r)$	Rules of Conjunction Rule of Disjunctive Syllogism Rule of Contradiction Rule of Conjunctive simplification Rule of Disjunctive Amplification Rule of Conditional Proof Rule for Proof by Cases Rule of the Constructive Dilemma Rule of the Destructive Dilemma
---	--	--

$\exists x p(x)$	For some (at least one) a in the universe, $p(a)$ is true.	For every a in the universe, $p(a)$ is false.
$\forall x p(x)$	For every replacement a from the universe, $p(a)$ is true.	There is at least one replacement a from the universe for which $p(a)$ is false.
$\exists x \neg p(x)$	For at least one choice a in the universe, $p(a)$ is false, so Its negation $\neg p(a)$ is true.	For every replacement a in the universe, $p(a)$ is true.

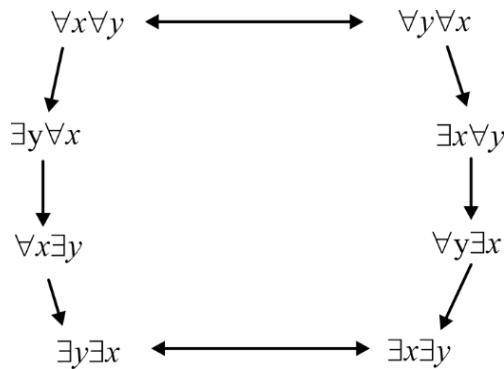
$\forall x \neg p(x)$	For every replacement a from The universe, $p(a)$ is false and its negation $\neg p(a)$ is true	There is at least one replacement a from the universe for which $\neg p(a)$ is false and $p(a)$ is true.
-----------------------	---	--

$$\begin{aligned}\neg [\forall x p(x)] &\Leftrightarrow \exists x \neg p(x) \\ \neg [\exists x p(x)] &\Leftrightarrow \forall x \neg p(x) \\ \neg [\forall x \neg p(x)] &\Leftrightarrow \exists x \neg\neg p(x) \Leftrightarrow \exists x p(x) \\ \neg [\exists x \neg p(x)] &\Leftrightarrow \forall x \neg\neg p(x) \Leftrightarrow \forall x p(x)\end{aligned}$$

$\exists x [P(x) \vee Q(x)] \equiv \exists x P(x) \vee \exists x Q(x)$
$\forall x [P(x) \vee Q(x)] \equiv \forall x P(x) \vee \forall x Q(x)$
$\exists x [P(x) \wedge Q(x)] \rightarrow \exists x P(x) \wedge \exists x Q(x)$
$\forall x [P(x) \wedge \forall Q(x)] \rightarrow \forall x [P(x) \vee Q(x)]$
$\forall x [P(x) \rightarrow Q(x)] \rightarrow \forall x P(x) \rightarrow \forall x Q(x)$
$\forall x [P(x) \leftrightarrow Q(x)] \rightarrow \forall P(x) \leftrightarrow \forall x Q(x)$
$(\forall x) P(x) \wedge (\forall x) Q(x) \Leftrightarrow (\forall x) [P(x) \wedge Q(x)]$
$(\forall x) P(x) \wedge (\forall x) Q(x) \Leftrightarrow (\forall x) (\forall y) [P(x) \vee Q(y)]$
$(\exists x) P(x) \vee (\exists x) Q(x) \Leftrightarrow (\exists x) (\exists y) [P(x) \wedge Q(y)]$
$(\exists x) P(x) \vee (\exists x) Q(x) \Leftrightarrow (\exists x) [P(x) \vee Q(y)]$
$(\forall x) P(x) \wedge (\exists x) Q(x) \Leftrightarrow (\forall x) (\exists y) [P(x) \wedge Q(y)]$
$(\forall x) P(x) \vee (\exists x) Q(x) \Leftrightarrow (\forall x) (\exists y) [P(x) \vee Q(y)]$
$A \vee (\forall x) P(x) \Leftrightarrow (\forall x) [A \vee P(x)]$
$A \vee (\exists x) P(x) \Leftrightarrow (\exists x) [A \vee P(x)]$
$A \wedge (\forall x) P(x) \Leftrightarrow (\forall x) [A \wedge P(x)]$
$A \wedge (\exists x) P(x) \Leftrightarrow (\exists x) [A \wedge P(x)]$

Statement	When True?	When False?
$\forall x \forall y P(x, y)$	$P(x, y)$ is true for every pair x, y	There is a pair x, y for which $P(x, y)$ is false.
$\forall y \forall x P(x, y)$	For every x there is a y for which $P(x, y)$ is true	There is an x such that $P(x, y)$ is false for every y .
$\exists x \forall y P(x, y)$	There is an x for which $P(x, y)$ is true for every y .	For every x there is a y for which $P(x, y)$ is false.

$\exists x \exists y P(x, y)$	The is a pair x, y for which $P(x, y)$ is true	$P(x, y)$ is false for every pair x, y
-------------------------------	--	--



Rule of Inference	Name
$\frac{\forall x P(x)}{\therefore P(c)}$	Universal instantiation
$\frac{P(c) \text{ for an arbitrary } c}{\therefore \forall x P(x)}$	Universal generalization
$\frac{\exists x P(x)}{\therefore \text{for some element } c}$	Existential instantiation
$\frac{P(c) \text{ for some element } c}{\therefore \exists x P(x)}$	Existential generalization

Number of non-equivalent propositional function possible with 'n' propositional variable			
2^{2^n}			
Nested Quantifier			
$\forall x \forall y$	$\forall x \exists y$	$\exists y \forall x$	$\exists x \exists y$
English statements		Logical Expressions	

All graphs are connected	$\forall x[G(x) \rightarrow C(x)]$
Not all graphs are connected	$\neg\forall x[G(x) \rightarrow C(x)]$
All graphs are not connected \equiv No graphs are connected	$\forall x[G(x) \rightarrow \neg C(x)] \equiv \forall x[G(x) \rightarrow \neg\neg C(x)]$

◻◻◻



3

SET THEORY

3.1 Introduction

Collection of unordered, distinct and well defined objects.

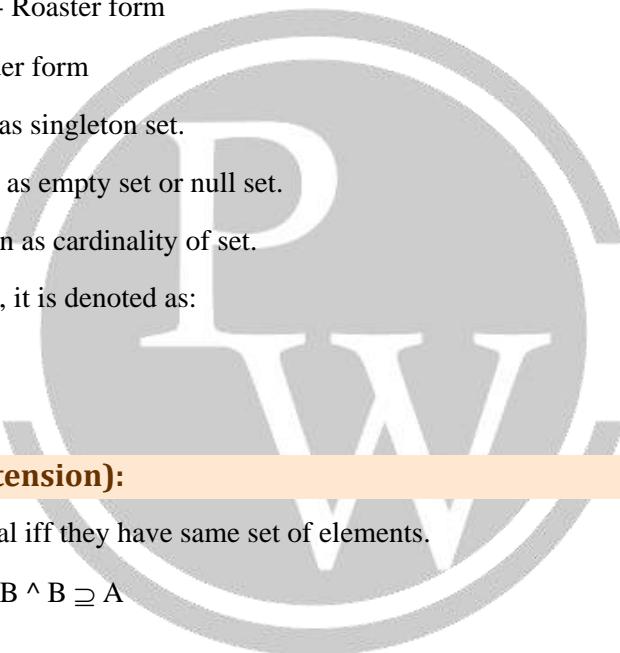
- $\{1, 2\} = \{2, 1\} = \{1, 2, 1\}$ (order & repetition doesn't matter)

Set representation: A {1, 2, 3} - Roaster form

A = {z|z ∈ N $x \leq 3$ } – Set builder form

- Set with one element is known as singleton set.
- Set with zero element is known as empty set or null set.
- No of elements in a set is known as cardinality of set.
- If a set A contains an element a, it is denoted as:

a ∈ A.



3.1.1 Equal sets (Axiom of extension):

Two sets A and B are said to be equal iff they have same set of elements.

$$A = B \Leftrightarrow \forall x (x \in A \Leftrightarrow x \in B) \Leftrightarrow A \subseteq B \wedge B \subseteq A$$

3.1.2 Subset:

- A is said to be subset of B iff every element in A is also then in B. Denoted as $A \subseteq B$

$$\forall x (x \in A \Rightarrow x \in B)$$

3.1.3 Proper subset:

- 'A' is said to be a proper subset of B if every element in A. exist in B and $A \neq B$. Denoted as $A \subset B$

$$\text{i.e., } A \subset B \Leftrightarrow \forall x (x \in A \rightarrow x \in B) \wedge \exists x. (x \in B \wedge x \notin A)$$

or

$$A \subset B \Leftrightarrow \forall x (x \in A \rightarrow x \in B) \wedge (|A| \neq |B|)$$

3.1.4 Powerset:

Powerset of a set A is set of all subsets of A. Denoted as $P(A)$

- if $|A| = n$
 $|P(A)| = 2^n$
- For any set A
 $\emptyset \subseteq A$
 If $A = \emptyset$, then
 $\emptyset \subset A$

3.1.5 Operations on set:

(i) **Union:**

$$A \cup B = \{x | x \in A \vee x \in B\}$$

(ii) **Intersection:**

$$A \cap B = \{x | x \in A \wedge x \in B\}$$

(iii) **Complement:**

If A is a set,

Complement of A, $\bar{A} = A^c = \{x | x \in U \wedge x \notin A\}$

i.e., $\bar{A} = U - A = U \cap \bar{A}$

(iv) **Difference:**

$$A - B = A - B = \{x | x \in A \wedge x \notin B\} = A \cap \bar{B}$$

$A - \bar{B}$ is also called complement of B w.r.t. to A.

“complement of B in A”

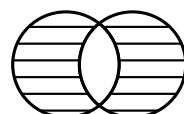


(v) **Symmetric Difference:**

$$A \Delta B = \{x | x \in A \vee x \in B, \text{ but not both}\}$$

Since it corresponds to XOR operator, it is also denoted as $A \oplus B$

$$A \Delta B = (A - B) \cup (B - A) = (A \cup B) - (A \cap B)$$



- Two sets are said to be disjoint if their intersection \emptyset .

3.1.6 Laws involving set operations:

- $A \cup A = A$
 $A \cap A = A$ } Idempotent law
- $A \cap \emptyset = \emptyset$
 $A \cup U = U$ } Domination law
- $A \cap U = A$
 $A \cup \emptyset = A$ } Identity law
- $A \cup (B \cup C) = (A \cup B) \cup C$
 $A \cup (B \cap C) = (A \cap B) \cup C$
 $A \oplus (B \oplus C) = (A \oplus B) \oplus C$ } Associative law
- \therefore XOR operation is associative
- $A \cup (B \cup C) = (A \cup B) \cup (A \cup C)$
 $A \cap (B \cap C) = (A \cap B) \cap (A \cap C)$ } Distributive law
- $A \cup (A \cap B) = A$
 $A \cap (A \cup B) = A$ } absorption law
- $\overline{A \cup B} = \bar{A} \cap \bar{B}$
 $\overline{A \cap B} = \bar{A} \cup \bar{B}$ } Demorgan's law

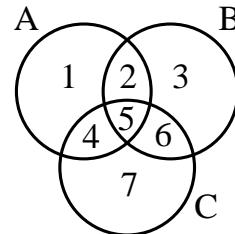
Note:

$A \oplus B = A \oplus C \Rightarrow B = C$
 $A \oplus C = B \oplus C \Rightarrow A = C$ } The proof can be obtained by relating this to XOR operation in logic.

3.1.7 Representation of sets using Venn diagrams:

Rather than direct calculations we sometimes represent sets with Venn diagram and see them as regions.

Let A, B, C be 3 sets



The region 1 is represented by $A \cap \bar{B} \cap \bar{C}$

The region 2 is represented by $A \cap B \cap \bar{C}$

Similarly, we have 8 regions including region outside of A, B, C

The principle of duality:

Let s denote a theorem dealing with the equality at two expressions which involve only operation \cup and \cap . The dual of s(d) (i.e., expression obtained by interchanging \cup and \cap) is also a theorem.

Finding dual of $A \subseteq B$

$A \subseteq B$ can be written as

$$A \cup B = B$$

$$\text{Dual is } A \cap B = B$$

$$\text{i.e., } B \subseteq A$$

\therefore Dual of $A \subseteq B$ is $B \subseteq A$

Duality principle should be applied only for general cases but not for particular cases

3.1.8 Cartesian Product:

$$A \times B = \{(a, b) | a \in A \wedge b \in B\}$$

- Cartesian product is associative

However

$$(A \times B) \times C \quad \neq \quad A \times (B \times C)$$

↓ ↓

it has ordered pairs type $((a, b), c)$ $(a, (b, c))$

$$A \times B \neq B \times A$$

If $A \times B = B \times A$ then $(A = B \vee (A = \emptyset) \vee (B = \emptyset))$

- $A \times (B \cap C) = (A \times B) \cap (A \times C)$

$$A \times (B \cup C) = (A \times B) \cup (A \times C)$$

$$\bar{A} \times \bar{B} \neq \overline{A \times B}$$

$$\bar{A} \times \bar{B} \leq \overline{A \times B}$$

3.1.9 Multisets:

Multiset is an unordered collection of elements where one element can occur more than once.

Eg: $\{m_1.a_1, m_2.a_2, \dots, m_n.a_n\}$

Where m_i is called multiplicity of set

3.1.10 Operations on multisets:

- Union: Maximum of multiplicities is considered.
- Intersection: Minimum of multiplicities is considered
- Difference: Difference of multiplicities is considered.
If the difference is -ve, it is considered 0.
- Sum: Sum of multiplicities is considered.
It is denoted as $P + Q$

3.2 Relations:

A binary relation from set A to set B is any subset of $A \times B$.

If $|A| = M$ and $|B| = n$, then $|A \times B| = mn$

∴ number of relations possible from A to B = 2^{mn}

If relation R from A to A we say it relation on A.

∴ Number of relations possible on a set A = 2^{n^2} , $|A| = n$

Domain of relation : $\{x \mid (x, y) \in R\}$

Range of relation : $\{y \mid (x, y) \in R\}$

- $(x, y) \in R$ is written as 'xRy' or 'k relates y'

3.2.1 Inverse of relations:

Inverse of a relation, $R^{-1} = \{(y, x) \mid (x, y) \in R\}$

3.2.2 Diagonal relation:

Diagonal relation on set A is

$$\Delta_A = \{(a, a) \mid \forall a \in A\}$$

3.2.3 Reflexive relations:

Relation R defined on set A is reflexive

$$\Leftrightarrow aRa \quad \forall a \in A$$

i.e., if R is a reflexive relation then $\Delta_A \leq R$

3.2.4 Important type of relations:

Type of relation	Condition	No. of relatives possible	Union	Intersection
Reflective	$aRa, \forall a \in A$	2^{n^2-n}	✓	✓
Irreflexive	$\forall a \in A (a \not R a)$	2^{n^2-n}	✓	✓
Symmetric	$\forall x, y \in A (xR_y \Rightarrow yR_x)$	$\frac{n^2-n}{2}$	✓	✓
Antisymmetric	$\forall x, \forall y (xR_y \wedge yR_x \Rightarrow x = y)$	$\frac{n^2-n}{2}$	✗	✓
Asymmetric	$\forall x, \forall y (xR_y \Rightarrow y \not R_x)$	$\frac{n^2-n}{2}$	✗	✓
Transitive	$\forall x, \forall y (\forall z (xR_y \wedge yR_z \Rightarrow xR_z))$	—	✗	✓

- Note that all the above relations are defined on a single set.
- Also to prove any relation is not of certain type we need to P.T the logical formula for that relation is false.
- Every Asymmetric relation is antisymmetric relation.

3.2.5 Composition of relations:

If R is a relation from A to B, S is a relation from B to C, the composition of R & S is given by

SoR from A to C.

$$S.R = \{(a, c) | (a, b) \in R \text{ and } (b, c) \in S\}$$

If R is relation on A, then composition is denoted as R^2, R^3 .

Note:

If R is any relation on set A, then

$$R_o\Delta_A = R \text{ and } \Delta_A oR = R$$

3.2.6 Closure:

Clause of a relation R under given property is smallest possible relation that contains R and Satisfy the property.

(i) Reflexive Closure (R^*):

Reflexive closure of a relation R,

$$R^\# = R \cup \Delta_A$$

(ii) **Symmetric closure:**

$$R^+ = RUR^{-1}$$

(iii) **Transitive closure:**

- Finding transitive closure has no formula, but we have a procedure as shown below.

Steps: Represent relation R with a directed graph such that whenever aRb draw a directed edge from a to b.

Steps: Now from each vertex, find all reachable vertices and for each reachable vertex b from a add the ordered pair (a, b) to the closure.

- The standard procedure for finding transitive closes is

$$R\# = R \cup R^2 \cup R^3 \cup \dots \cup R^{n-1} \cup R^n$$

'n' is a positive integer such that $R^{n-1} = R^n$

- when a relation is represent as a graph,

If $(a, b) \in R^n$, we can say that there exists an n-length path from a to b.

- when asked to find more than one closure, the order to be followed is reflexive, symmetric, transitive.

Following any other order may give redundancy.

- The closure of a relation, if exists, under a property is intersection of the relations with that property containing R.

- If R is a transitive relation,

- $R^n \leq R \forall n \geq 1$

- also R^n is transitive relation.

3.2.7 Partition:

A partition of set s is dividing s in disjoint subsets such that

$$A_1 \cup A_2 \cup \dots \cup A_n = s$$

$$A_i \cap A_j = \emptyset \quad \forall i, j \leq n$$

3.2.8 Refinement:

Partition P_2 is called refinement of partition P_1 .

if every partitioned subset of P_2 is subset to some partitioned subset of P_1 .

3.2.9 Equivalence Relation:

A relation which is reflexive

Symmetric

Transitive

is called an equivalence relation.

- Equivalence relation creates partition.
- Each set of the partition is called an equivalence class.
- Every element of same equivalence class are related to each other.
- No two element of different classes of related to each other

- Equivalence class is represented by
 - [a] R

Where a is any element of the equivalence classes

- If R is an equivalence relation, the below two sentences means the same

- (i) aRb
- (ii) $[a]_R = [b]_R$

aRb & $[a]_R \neq [b]_R$ also mean same.

If R is an equivalence relation.

aRb is read as “a is equivalent to b”.

- If R_1, R_2 are two equivalence relations

$R_1 \cup R_2$ need not to be an equivalence relation

$R_1 \cup R_2$ is an equivalence relation.

3.2.10 Finding number of equivalence relations:

- No of equivalence relations on a set with n elements is given by

$$\text{Bell number } B_n = \sum_{k=1}^n s(n, k)$$

Where sterling 2nd kind number

$$s(m, n) = \frac{1}{n!} \sum_{i=0}^n (-1)^i n c_i (n-i)^m$$

The below is a shortcut to find the Bell number

$$B_0 \rightarrow (1)$$

$$B_1 \rightarrow (1) \quad 2$$

$$B_2 \rightarrow (2) \quad 3 \quad 5$$

$$B_3 \rightarrow (5) \quad 7 \quad 10 \quad 15$$

$$B_4 \rightarrow (15) \quad 20 \quad 27 \quad 37 \quad 52$$

Find no. of equivalence relations is nothing but find no. of way we can partition the set.

3.3 Partial Ordering Relations:

A relation R on a set A is called a partial order if R is reflexive, antisymmetric and transitive.

3.3.1 Poset:

A set ‘A’ together with a partial order R is called a poset

It is denoted as $[A:R]$. In a poset aRb is denoted as $a \leq b$. $a < b$ means $a \leq b$ and $a \neq b$.

3.3.2 To set:

A poset $[A; R]$ is called toset if every pair of elements of set A are comparable.

Toset is also known as linearly ordered set (or) chain.

If $[A, R]$ is a poset, then

$[A ; R^{-1}]$ is called dual of the poset.

i.e., these diagram of $[A : R^{-1}]$ can be obtained by turning the hasse diagram of $[A, R]$ upside down.

3.3.3 Hasse Diagram (Poset Diagram):

- It is graphical representation of a poset.

It is constructed as below:

- Create vertex corresponding to every element of set A .
- All loops & Edges implied from transitivity are not shown.

- Let $[A, \leq]$ be a poset

We say yes, covers $x \in s$, if $x < y$ such that there doesn't exist any $z \in s$, $x < z < y$

- Thus hasse diagram shows edges b/w two elements x & y only if x covers y (or) y covers x . The set of such pairs $x < y$ is called covering relation of (s, \leq) .
- Thus applying reflexive transitive closure on covering relation of a poset gives the poset.
- Hasse Diagram of a toset is a chain.

3.3.4 Maximum Element:

In a poset an element is called maximal if it is not related to any other element.

3.3.5 Greatest element (or) Maximum element:

In a poset an element is called greatest if every element of the set relates to that element.

3.3.6 Minimal element:

In a poset an element is called minimal, if no other element is related to it.

3.3.7 Least element (or) Minimum Element:

In a poset least element is the one which relates to every other element of the set.

- Every finite, nonempty lattice has at least one minimal and one maximal element.
- Greatest or least element if exists is unique.
- Greatest and least elements may or may not exist if they exist they are the only maximal and only minimal elements respectively.

3.3.8 Upper Bound:

If $[A : R]$ is a poset and $B \leq A$

Upper band of $B = \{x | \forall b \in B, x \leq b \text{ and } x \in A\}$

3.3.9 Least Upper Bound (lub or join or Supremum):

In a poset $[A : R]$ lub of two elemis $a, b \in A$ is least element of upper bound of $\{a, b\}$. It is denoted as $a \vee b$.

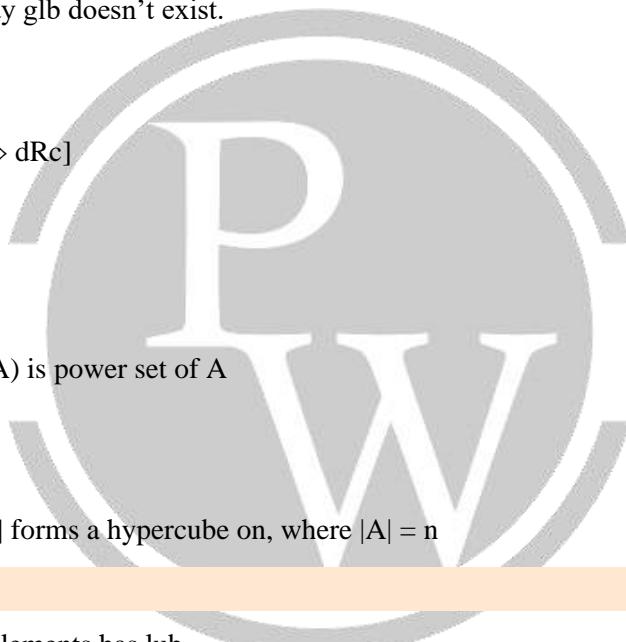
i.e., $a \vee b \leq x \forall x \in$ upper band of $[a, b]$

- If no least element exists in the upper bound, we say lub doesn't exist.
- In other word,
 - If $a \vee b = c$ then
 $aRc \& bRc$ and
If $aRd \wedge bRd \Rightarrow cRd$

3.3.10 Greatest lower Bound (glb or meet or infimum)

In a poset $[A, R]$ glb of a, b is greatest element of lower bound of $\{a, b\}$. It is denoted as $a \wedge b$

- If no such element exists we say glb doesn't exist.
- Other way to define glb is
 - If $a \wedge b = c$, then
 $[eRa \wedge cRb] \wedge [(dRa \wedge dRb) \Rightarrow dRc]$
- For poset $[D_n ; 1]$
 $Lub(a, b) = LCM(a, b)$
 $Glb(a, b) = GCD(a, b)$
- For poset $[P(A) ; \leq]$, where $P(A)$ is power set of A
 $lub(x, y) = x \cup y$
 $lub(x, y) = x \cap y$
- The hasse diagram of $[P(A); \leq]$ forms a hypercube on, where $|A| = n$



3.3.11 Join Semi Lattice:

It is a poset in which every pair of elements has lub.

3.3.12 Meet Semi Lattice:

It is a poset in which every pair of elements has glb.

3.4 Lattice

A lattice is a poset in which every pair of elements has both glb & lub.

i.e., Lattice is both join semi lattice & meet semi lattice.

A lattice L is denoted as (L, \wedge, \vee)

- It is not needed that every lattice has greatest and least element.
- Every finite lattice has least and greatest elements.

3.4.1 Properties of Lattice:

- $a \wedge a = a$ } Idempotent
- $a \vee a = a$
- $a \vee b = a \vee a$ } Commutative
- $a \wedge v = a \wedge a$
- $a \vee (b \vee c) = (a \vee b) \vee c$ } Associative
- $a \wedge (b \wedge c) = (a \wedge b) \wedge c$
- $a \vee (a \wedge b) = a$ } Absorption law
- $a \wedge (a \vee b) = a$
- $a \leq b \Rightarrow a \vee c \leq b \vee c$
- $a \leq b \Rightarrow a \wedge c \leq b \wedge c$

If $a \leq b$ and $c \leq d$ then

$$a \vee c \leq b \vee d$$

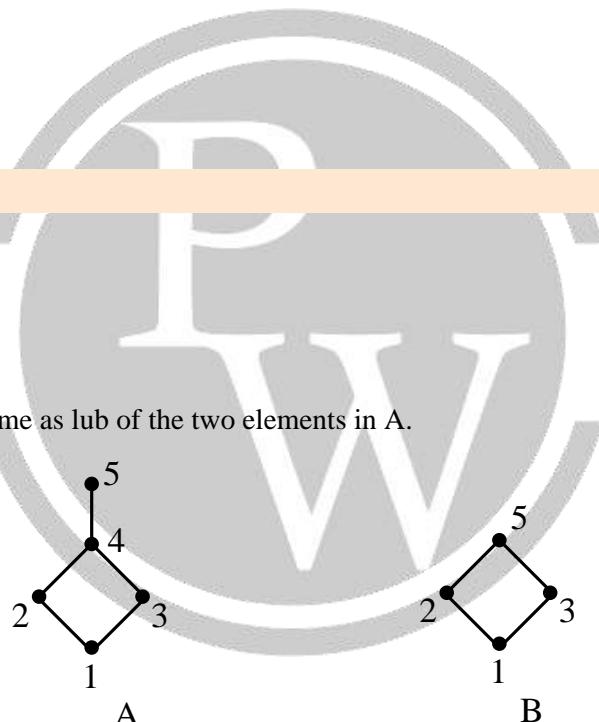
$$a \wedge c \leq b \wedge d$$

3.4.2 Sublattice:

If A is a lattice B is called

Sublattice of A iff

- B itself is a lattice.
- lub of any two element of B is same as lub of the two elements in A.



In above figure B is subset of A and B is a lattice still B is not a sublattice of A.

Because $\begin{bmatrix} \text{in } A \text{ glb } (2,3) = 4 \\ \text{in } B \text{ glb } (2,3) = 5 \end{bmatrix}$ not equal \therefore not sublattice

3.4.3 Types of Lattices:

(1) Bounded Lattice:

- A lattice with greatest element and least element is called bounded lattice.
- Every finite lattice is bounded lattice.

(2) Complemented Lattice:

- Complemented lattice is a bounded in which every element has atleast one complement. B is said to be complement of a iff.
- $\text{glb } (a, b) = \text{greatest element}$ and $\text{lub } (a, b) = \text{least element}$.

(3) Distributive Lattice:

- A lattice is said to be distributive if following distributive laws hold.
- $a \vee (b \wedge c) = (a \vee b) \wedge (a \vee c)$.
- $a \wedge (b \vee c) = (a \wedge b) \vee (a \wedge c)$.

Note: If L is a bounded distributive lattice then complement of an element if exists, is unique. The reverse need not to be true.

Hence if we find more than one complement for an element, we can conclude that the lattice is not distributive

(4) Boolean Algebra:

- A lattice which is both distributive and complemented is known as Boolean algebra
- It is called so because it satisfies all the properties of Boolean algebra. Thus when given lattice is a Boolean algebra we can apply all the rules that we apply in logic.
- In Boolean algebra every element has exactly one complement.
- $[D_n : 1]$ is a distributive lattice for any n.
- It is because distributive properties hold for lcm & gcd.
- $[D_n : 1]$ is a Boolean lattice if n is a square free number.
- $[P(s); \leq]$ is a Boolean lattice.
- Every Boolean lattice with 2^n elements, $\forall n \geq 0$.
Every Boolean lattice contains 2^n elements is isomorphic to the lattice $(P(s); \leq)$ where s is a set with n elements.
Also this Boolean lattice is a hypercube Q_n .
- Sublattice of a complemented lattice need not to be a complemented lattice.
- Sublattice of a bounded lattice is a bounded lattice.

3.5 Function (or) Mapping (or) Transformation

A function F from set A to set B is an assignment of exactly one element of B to each element of A. It is denoted as: $F : A \rightarrow B$

Here A is called Domain B is called codomain

Function is a special type of relation.

- Consider $f(a) = b$
b is called image of a
a is called preimage of b.

- Range: It is set of images of all the elements of A.
Range \neq Co-domain (Range need not to be equal to co-domain)
- If f_1, f_2 are two functions
 $f_1(x) + f_2(x)$ is denoted as $(f_1 + f_2)(x)$
 $f_1(x) \cdot f_2(x)$ is denoted as $(f_1 \cdot f_2)(x)$
- $\frac{1}{f(x)}$ is denoted as $f^{-1}(x)$, (f^{-1} is not equal to inverse function)
- If A and B are two sets with
 $|A| = n$ $|B| = m$ then
number of functions possible from A to B = m^n

3.6 Types of functions:

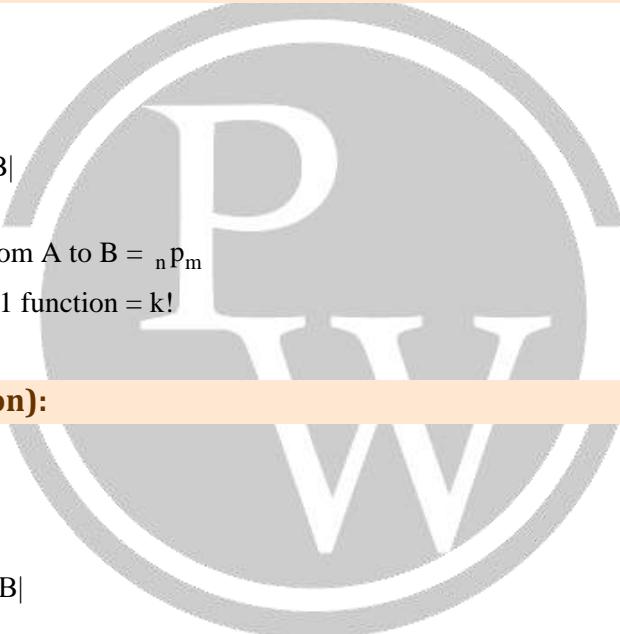
3.6.1 One - One function (Injection):

$f : A \rightarrow B$ is one to one

iff

$$\forall a \forall b (f(a) = f(b) \Rightarrow a = b)$$

- If $f : A \rightarrow B$ is 1-1 then $|A| \leq |B|$
- If $|A| = m$ and $|B| = n$
no of 1 - 1 function possible from A to B = ${}_n p_m$
- If $m = n = k$, then number of 1:1 function = $k!$



3.6.2 Onto function (Surjection):

$f : A \rightarrow B$ is onto iff

$$\forall b \in B \ \exists a \in A \text{ such that } f(a) = b$$

- If $f : A \rightarrow B$ is onto then $|A| \geq |B|$
- If $|A| = m$ $|B| = n$ then

$$\therefore \text{number of onto functions} = \sum_{i=0}^n (-1)^i n c_i (n-i)^m$$

- If $m = n = k$, then number of 1:1 function = $k!$

3.6.3 One-to-One Correspondence (or) Bijection:

A function $f : A \rightarrow B$ is Bijection iff

f is both one-one and onto

- If $f : A \rightarrow B$ is bijection, then $|A| = |B|$
- If $f : A \rightarrow B$ is 1 - 1 and $|A| = |B|$
then f is a bijection
- If $f : A \rightarrow B$ is onto and $|A| = |B|$
then f is a bijection

- If $|A| = |B| = n$ then no of bijections possible from A to B are $n!$
- If $f : A \rightarrow A$ is a function and A is a finite set then
A is 1-1 \Leftrightarrow A is onto

3.6.4 Inverse of a function:

$f : A \rightarrow B$ is invertible if its inverse relation f^{-1} is a function from B to A.

$f : A \rightarrow B$ is invertible $\Leftrightarrow f$ is a bijection

- Inverse doesn't exist if a function is not bijection. However, we can find inverse image of subset of codomain. If $f : A \rightarrow B$ is a function and $S \subseteq B$

Inverse image of S = $\{a \in A | f(a) \subseteq S\}$

- If f is a function from A to B and let S, T be subsets of A.

$$f(S \cup T) = f(S) \cup f(T)$$

$$f(S \cap T) = f(S) \cap f(T)$$

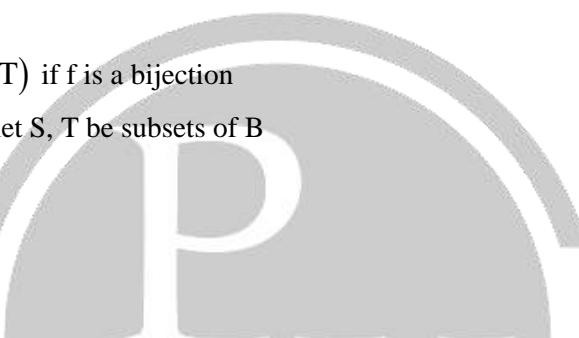
However, $f(S \cap T) = f(S) \cap f(T)$ if f is a bijection

- If f is a function from A to B and let S, T be subsets of B

$$f^{-1}(S \cup T) = f^{-1}(S) \cup f^{-1}(T)$$

$$f^{-1}(S \cap T) = f^{-1}(S) \cap f^{-1}(T)$$

$$f^{-1}(\bar{S}) = \overline{f^{-1}(S)}$$



3.6.5 Identity function:

A mapping $I_A : A \rightarrow A$ is called an identity function if

$$I_A = \{(x, x) | x \in A\}$$

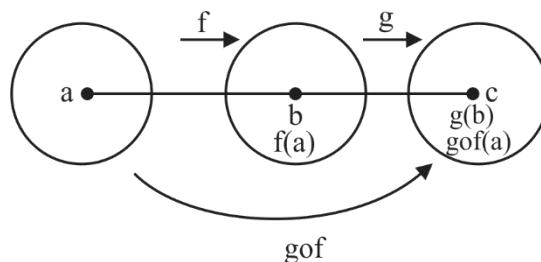
3.6.7 Constant function:

A function $f : A \rightarrow B$ is said to be a constant function if

$$f(x) = c \quad \forall x \in A$$

3.6.8 Composition of functions:

If $f : A \rightarrow B$ and $g : B \rightarrow C$ are two functions then $gof : A \rightarrow C$ is called a composition function of f & g.

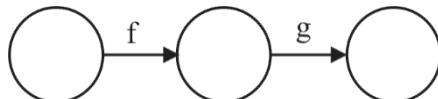


$$gof(x) = g(f(x))$$

- In $gof : A \rightarrow C$
A is domain of gof
C is codomain of gof

Range of $g \circ f$ is image of (range of f under g)

- If $f : A \rightarrow B$ and $g : B \rightarrow C$ are two functions
 $g \circ f$ is defined for every case



But $f \circ g$ is defined iff

range of $g \leq$ domain of f

$f \circ g$ maps from B to B i.e., $f \circ g : B \rightarrow B$

$f \circ g \neq g \circ f$ (i.e., composition is not commutative)

$(f \circ g) \circ h = f \circ (g \circ h)$ (i.e., Associative)

$(f \circ g)^{-1} = g^{-1} \circ f^{-1}$ (Think why)

Let $f : A \rightarrow B$ be an invertible function then

$f^{-1} : B \rightarrow A$ is a inverse of f

$f^{-1} \circ f : A \rightarrow A$ is an identity function I_A

$f \circ f^{-1} : B \rightarrow B$ is an identity function I_B

Note:

- f is 1-1 & g is 1-1 $\Rightarrow g \circ f$ is 1-1
- f is onto & g is onto $\Rightarrow g \circ f$ is onto
- f is bijection & g is bijection $\Rightarrow g \circ f$ is bijection
- If $g \circ f$ is onto then g is onto
- If $g \circ f$ is 1-1 then f is 1-1
- If $g \circ f$ is a bijection then f is onto $\Leftrightarrow g$ is 1-1

3.6.9 Partial Functions:

$f : A \rightarrow B$ is called partial function if ' f ' is defined only for some $a \in A$.

The subset of A on which f is defined is called domain definition of f .

3.7 Groups:

3.7.1 Binary Operation:

The binary operator $*$ is said to be a binary operation on a non-empty set A if the set is closed under the operation.
i.e., $(a * b) \in A \quad \forall a, b \in A$

3.7.2 Binary Structure (or) Algebraic Structure:

A nonempty set A is called binary structure with respect to a binary operator $*$, if $*$ is binary operation on A . it is denoted as $(A, *)$

3.7.3 Semi Group:

$(A, *)$ is semigroup iff

(i) is closed operation

(ii) is an associative property

3.7.4 Monoid:

(A, *) is called monoid iff:

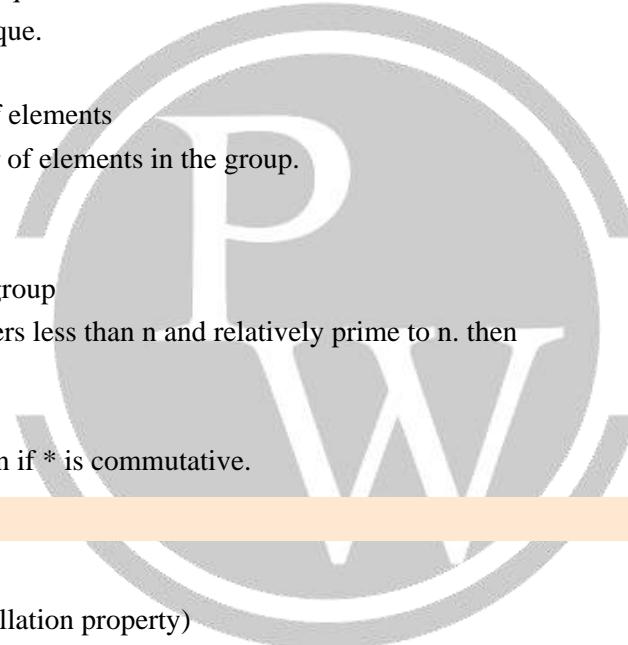
- (i) is closed operation
- (ii) is an associative property
- (iii) Identity element exists

3.7.5 Group:

(A, *) is called monoid iff:

- (i) is closed operation
- (ii) is an associative property
- (iii) Identity element exists in A
- (iv) Every element of A has inverse

- Identity element if exists is unique.
- Inverse element if exist is unique.
- Finite Group:
A group with finite number of elements
- Order of a group: It is number of elements in the group.
- In a group of 2 element
 $a^{-1} = a, \forall a \in G$
- $(\{0,1,2,3, \dots, m-1\}, \oplus_m)$ is a group
- Let S_n be set of positive integers less than n and relatively prime to n. then (S_n, \otimes_n) is a group.
- Abelian Group:
A group $(G, *)$ is called abelian if * is commutative.



3.7.6 Properties of Groups

- Let $(G, *)$ be a group
 - if $ab = ac \Rightarrow b = c$ (Left cancellation property)
 - $ba = ca \Rightarrow b = c$ (Right cancellation property)
 - due to these property every row and column in Caley table has exactly one element
- If G is a group and $a, b \in G$
then $(ab)^{-1} = b^{-1}a^{-1}$
- Group G is abelian $\Leftrightarrow (ab)^{-1} = b^{-1}a^{-1} \forall a, b \in G$
- $\forall a \in G, a^0 = e$

3.7.7 Homomorphism

- If $(G, *)$ and (G', \oplus) are two groups then a function $f : G \rightarrow G'$ is called a homomorphism
if $f(a * b) = f(a) \oplus f(b)$
- If $f : G \rightarrow G'$ is a bijection, then we call the homomorphism isomorphism. It is denoted as $G \cong G'$
- If e_1, e_2 are identity elements of G and G' respectively then $f(e_1) = e_2$
- If $a \in G$, and $f : G \rightarrow G'$ is a homomorphism $f(a^{-1}) = (f(a))^{-1}$

3.7.8 Order of an element:

The smallest positive integer n such that $a^n = e$ is called of order of an element a in the group.

- The order of an element is divisor of order of the group.
- $\text{Order}(a) = \text{order}(a^{-1}) \quad \forall a \in G$
- $a^{-n} = b^n$ if $a^{-1} = b$ and n is any positive integer.

3.7.9 Subgroup:

A non-empty subset H of a group G is called subgroup of G if $(H, *)$ is also a group.

- For every group G , $\{e\}$ and G are called trivial subgroups of G .
- Every subgroup contains identity element of its parent group.
- If H is a subgroup of G then $\text{order}(H)$ divides $\text{order}(G)$ (Lagrange's theorem). The converse of Lagrange's theorem holds only for abelian groups.
- If H and k are two subgroups of same group, then $H \cap k$ is also, a subgroup
- $H \cup k$ need not to be a subgroup.
- If $H \subseteq G$, then H is called subgroup of G iff:
 - (i) $(a * b) \in H \quad \forall a, b \in H$
 - (ii) $a^{-1} \in H \quad \forall a, b \in H$
- If $H \subseteq G$, and if H is finite \rightarrow (applies only when H is finite) and nonempty, then H is called subgroup iff: $(a * b) \in H \quad \forall a, b \in H$
- If G is a subgroup of composite order, then G necessarily has non-trivial subgroup.

3.7.10 Cyclic Groups:

A group G is called cyclic, if $\exists a \in G$ such that every element can be written as an integral power of a such an element 'a' is called generator.

- The order of generator is order of the group.
- If 'a' is a generator then a^{-1} is also a generator.
- All subgroups of a cyclic group are cyclic.
- If G is a cyclic group of order n , then no of generator of G is given by $\phi(n)$ (Euler's phi function)
- If 'a' is a generator of G , and let m be an integer such that $1 < m \leq n$ then

$$\text{order}(a^m) = \frac{n}{\text{gcd}(n, m)}$$

- If G is a cyclic group with generator a and let d be divisor of $|G|$.
For every d there exists exactly one subgroup of order d . this subgroup is generated by $a^{n/d}$ where $n = |G|$
 \therefore No of subgroup of a cyclic group = no. of divisors of $|G|$ each subgroup is generated by $a^{n/d}$. (d is divisor of $|G|$)

3.7.11 Cyclic subgroup of a group:

If $(G, *)$ is any group and let $a \in G$.

The smallest subgroup of G containing a is $\langle a \rangle$ where

$$\langle a \rangle = \{a^n / \forall n \in \mathbb{Z}\}$$

Also $\langle a \rangle$ is a cyclic subgroup.

order of the subgroup $\langle a \rangle$ = order of element 'a' in G .

- If H is any subgroup of G and if H contains 'a' then
 $\langle a \rangle \subseteq H$
- Thus if G is a group
 $\langle a \rangle$ is a cyclic subgroup of G, $\forall a \in G$.
Such subgroups are called generating sets

Note:

- Every group of prime order is cyclic in which every element (except e) is a generator element.
- Every cyclic group is abelian.
- A group is cyclic \Leftrightarrow it can't be expressed as union of two proper subgroups.
- If $(G, *)$ and (H, \oplus) are two groups
 $(G \times H, \bullet)$ is a group where \bullet is defined as

$$(g_1, h_1) \bullet (g_2, h_2) = (g_1 * g_2, h_1 \oplus h_2)$$

This group is called direct product of G and H.

- Every group of order less than or equal to 5 is abelian.
- There is only one unique group for each of the order 1,2,3.



4

COMBINATORICS

4.1 Introduction

Let $m \in \mathbb{N}$. For a procedure of m successive distinct and independent steps with n_1 outcomes possible for the first step, n_2 outcomes possible for the second step, ..., and n_m outcomes possible for the m th step, the total number of possible outcomes is

$$n_1 \cdot n_2 \cdot \dots \cdot n_m$$

For a collection of m disjoint sets with n_1 elements in the first, n_2 elements in the second, . . . , and n_m elements in the m th, the number of ways to choose one element from the collection is

$$(x+y)^n = \sum_{k=0}^n \binom{n}{n-k} x^k y^{n-k}.$$

$$\sum_{i=0}^n C(n, i) (-1)^i = 0$$

For n even,

$$\sum_{i=0}^{n/2} C(n, 2i) = \sum_{i=0}^{n/2-1} C(n, 2i+1)$$

For n odd,

$$\sum_{i=0}^{\lfloor n/2 \rfloor} C(n, 2i) = \sum_{i=0}^{\lfloor n/2 \rfloor + 1} C(n, 2i+1)$$

$$\sum_{i=1}^n iC(n, i) = n2^{n-1}.$$

For positive integers n, t , the coefficient of $x_1^{n_1} x_2^{n_2} x_3^{n_3} \dots x_t^{n_t}$ in the expansion of $(x_1 + x_2 + x_3 + \dots + x_t)^n$ is

$$\frac{n!}{n_1! n_2! n_3! \dots n_t!},$$

where each n_i is an integer with $0 \leq n_i \leq n$, for all $1 \leq i \leq t$, and $n_1 + n_2 + n_3 + \dots + n_t = n$.

When we wish to select, with repetition, r of n distinct objects, we are considering all arrangements of r x's and $n - 1$ |'s and that their number is

$$\frac{(n+r-1)!}{r!(n-1)!} = \binom{n+r-1}{r}.$$

Consequently, the number of combinations of n objects taken r at a time, with repetition, is $C(n+r-1, r)$.

4.1.1 we recognize the equivalence of the following:

- (a) The number of integer solutions of the equation

$$x_1 + x_2 + \dots + x_n = r, \quad x_i \geq 0, \quad 1 \leq i \leq n.$$

- (b) The number of selections, with repetition, of size r from a collection of size n .

- (c) The number of ways r identical objects can be distributed among n distinct containers.

$$b_n = \binom{2n}{n} - \binom{2n}{n-1} = \frac{1}{n+1} \binom{2n}{n}, \quad n \geq 1, b_0 = 1.$$

The numbers b_0, b_1, b_2, \dots are called the Catalan numbers, after the Belgian mathematician

Order Is Relevant	Repetitions Are Allowed	Type of Result	Formula
Yes	No	Permutation	$P(n, r) = n!/(n-r)!, \quad 0 \leq r \leq n$
Yes	Yes	Arrangement	$N^r, \quad n, r \geq 0$
No	No	Combination	$C(n, r) = n!/[r!(n-r)!] = \binom{n}{r}, \quad 0 \leq r \leq n$
No	Yes	Combination with repetition	$\binom{n+r-1}{r}, \quad n, r \geq 0$

If there are n objects with n_1 indistinguishable objects of a first type, n_2 indistinguishable objects of a second type, . . . and n_r indistinguishable objects of an r th type, where $n_1 + n_2 + \dots + n_r = n$, then there are $\frac{n!}{n_1!n_2!...n_r!}$ (linear) arrangements of the given n objects.

Let A and B be subsets of a finite universal set U . Then

- (a) $|A \cup B| = |A| + |B| - |A \cap B|$
- (b) $|A \cap B| \leq \min\{|A|, |B|\}$, the minimum of $|A|$ and $|B|$
- (c) $|A \setminus B| = |A| - |A \cap B| \geq |A| - |B|$
- (d) $|A^c| = |U| - |A|$
- (e) $|A \oplus B| = |A \cup B| - |A \cap B| = |A| + |B| - 2|A \cap B| = |A/B| + |B/A|$
- (f) $|A \times B| = |A| \cdot |B|$

Given a finite number of finite sets, A_1, A_2, \dots, A_n , the number of elements in the union $A_1 \cup A_2 \cup \dots \cup A_n$ is

$$|A_1 \cup A_2 \cup \dots \cup A_n| = \sum_i |A_i| - \sum_{i < j} |A_i \cap A_j| + \sum_{i < j < k} |A_i \cap A_j \cap A_k| - \dots + (-1)^{n+1} |A_1 \cap A_2 \cap \dots \cap A_n|,$$

where the first sum is over all i , the second sum is over all pairs i, j with $i < j$, the third sum is over all triples i, j, k with $i < j < k$, and so forth.

The number of derangements of $n \geq 1$ ordered symbol is

$$D_n = n! \left(1 - \frac{1}{1!} + \frac{1}{2!} - \frac{1}{3!} + \dots + (-1)^n \frac{1}{n!} \right).$$

Let a_0, a_1, a_2, \dots be a sequence of real numbers. The function

$$f(x) = a_0 + a_1 x + a_2 x^2 + \dots = \sum_{i=0}^{\infty} a_i x^i$$

is called the generating function for the given sequence.

4.1.2 Extended Binomial coefficient

$$\begin{aligned} \binom{-n}{r} &= \frac{(-n)(-n-1)(-n-2)\dots(-n-r+1)}{r!} \\ &= \frac{(-1)^r (n)(n+1)(n+2)\dots(n+r-1)}{r!} \\ &= \frac{(-1)^n (n+r-1)!}{(n-1)! r!} = (-1)^r \binom{n+r-1}{r}. \end{aligned}$$

For all $m, n \in \mathbb{Z}^+, a \in \mathbb{R}$,

$$(1) \quad (1+x)^n = \binom{n}{0} + \binom{n}{1}x + \binom{n}{2}x^2 + \dots + \binom{n}{n}x^n$$

$$(2) \quad (1+ax)^n = \binom{n}{0} + \binom{n}{1}ax + \binom{n}{2}a^2x^2 + \dots + \binom{n}{n}a^n x^n$$

$$(3) \quad (1+x^m)^n = \binom{n}{0} + \binom{n}{1}x^m + \binom{n}{2}x^{2m} + \dots + \binom{n}{n}x^{nm}$$

$$(4) \quad (1-x^{n+1})/(1-x) = 1 + x + x^2 + \dots + x^n$$

$$(5) \quad 1/(1-x) = 1 + x + x^2 + x^3 + \dots = \sum_{i=0}^{\infty} x^i$$

$$(6) \quad 1/(1-ax) = 1 + (ax) + (ax)^2 + (ax)^3 + \dots$$

$$= \sum_{i=0}^{\infty} (ax)^i = \sum_{i=0}^{\infty} a^i x^i$$

$$= 1 + ax + a^2 x^2 + a^3 x^3 + \dots$$

$$(7) \quad 1/(1+x)^n = \binom{-n}{0} + \binom{-n}{1}x + \binom{-n}{2}x^2 + \dots$$

$$= \sum_{i=0}^{\infty} \binom{-n}{i} x^i$$

$$= 1 + (-1) \binom{n+1-1}{1} x + (-1)^2 \binom{n+2-1}{2} x^2 + \dots$$

$$= \sum_{i=0}^{\infty} (-1)^i \binom{n+i-1}{i} x^i$$

$$(8) \quad 1/(1-x)^n = \binom{-n}{0} + \binom{-n}{1}(-x) + \binom{-n}{2}(-x)^2 + \dots$$

$$= \sum_{i=0}^{\infty} \binom{-n}{i} (-x)^i$$

$$= 1 + (-1) \binom{n+1-1}{1} (-x) + (-1)^2 \binom{n+2-1}{2} (-x)^2 + \dots$$

$$= \sum_{i=0}^{\infty} \binom{n+i-1}{i} x^i$$

If $f(x) = \sum_{i=0}^{\infty} a_i x^i$, $g(x) = \sum_{i=0}^{\infty} b_i x^i$, and $h(x) = f(x)g(x)$, then

$h(x) = \sum_{i=0}^{\infty} c_i x^i$, where for all $k \geq 0$,

$$c_k = a_0 b_x + a_1 b_{x-1} + \dots + a_{k-1} b_1 + a_k b_0 = \sum_{j=0}^k a_j b_{k-j}.$$

Objects Are Distinct	Containers Are Distinct	Some Container(s) May Be Empty	Number of Distributions
No	Yes	Yes	$\binom{n+m-1}{m}$ (1) $p(m)$, for $n = m$ (2) $p(m, 1) + p(m, 2) + \dots + p(m, n)$, for $n < m$
No	No	Yes	$\binom{n+(m-n)-1}{(m-n)} = \binom{m-1}{m-n} = \binom{m-1}{n-1}$
No	Yes	No	
No	No	No	$p(m, n)$

Consider the nonhomogeneous first-order relation

$$a_n + C_1 a_{n-1} = k r^n,$$

where k is a constant and $n \in \mathbb{Z}^+$. If r^n is not a solution of the associated homogeneous relation

$$a_n + C_1 a_{n-1} = 0,$$

then $a_n^{(p)} = Ar^n$, where A is a constant. When r^n is a solution of the associated homogeneous relation, then $a_n^{(p)} = Bnr^n$, for B a constant.

Now consider the case of the nonhomogeneous second-order relation

$$a_n + C_1a_{n-1} + C_2a_{n-2} = kr^n,$$

Where k is a constant. Here we find that

- (a) $a_n^{(p)} = Ar^n$, for A a constant, if r^n is not a solution of the associated homogeneous relation;
- (b) $a_n^{(p)} = Bnr^n$, where B is a constant, if $a_n^{(h)} = c_1r^n + c_2r_n^m$, where $r_1 \neq r$; and
- (c) $a_n^{(p)} = Cn^2r^n$, for C a constant, when $a_n^{(h)} = (c_1 + c_2n)r^n$.

Given a linear nonhomogeneous recurrence relation (with constant coefficients) of the form $C_0a_n + C_1a_{n-1} + C_2a_{n-2} + \dots + C_k a_{n-k} = f(n)$, where $C_0 \neq 0$ and $C_k \neq 0$, let $a_n^{(h)}$ denote the homogeneous part of the solution a_n .

	$a_n^{(p)}$
c , a constant	A , a constant
n	$A_1n + A_0$
n^2	$A_2n^2 + A_1n + A_0$
n^t , $t \in \mathbb{Z}^+$	$A_t n^t + A_{t-1}n^{t-1} + \dots + A_1n + A_0$
r^n , $r \in \mathbb{R}$	Ar^n
$\sin \theta n$	$A \sin \theta n + B \cos \theta n$
$\cos \theta n$	$A \sin \theta n + B \cos \theta n$
$n^t r^n$	$r^n (A_t n^t + A_{t-1}n^{t-1} + \dots + A_1n + A_0)$
$r^n \sin \theta n$	$Ar^n \sin \theta n + Br^n \cos \theta n$
$r^n \cos \theta n$	$Ar^n \sin \theta n + Br^n \cos \theta n$



GATE Exam 2024?



PARAKRAM BATCH

ME | CE | EE | EC | CS & IT

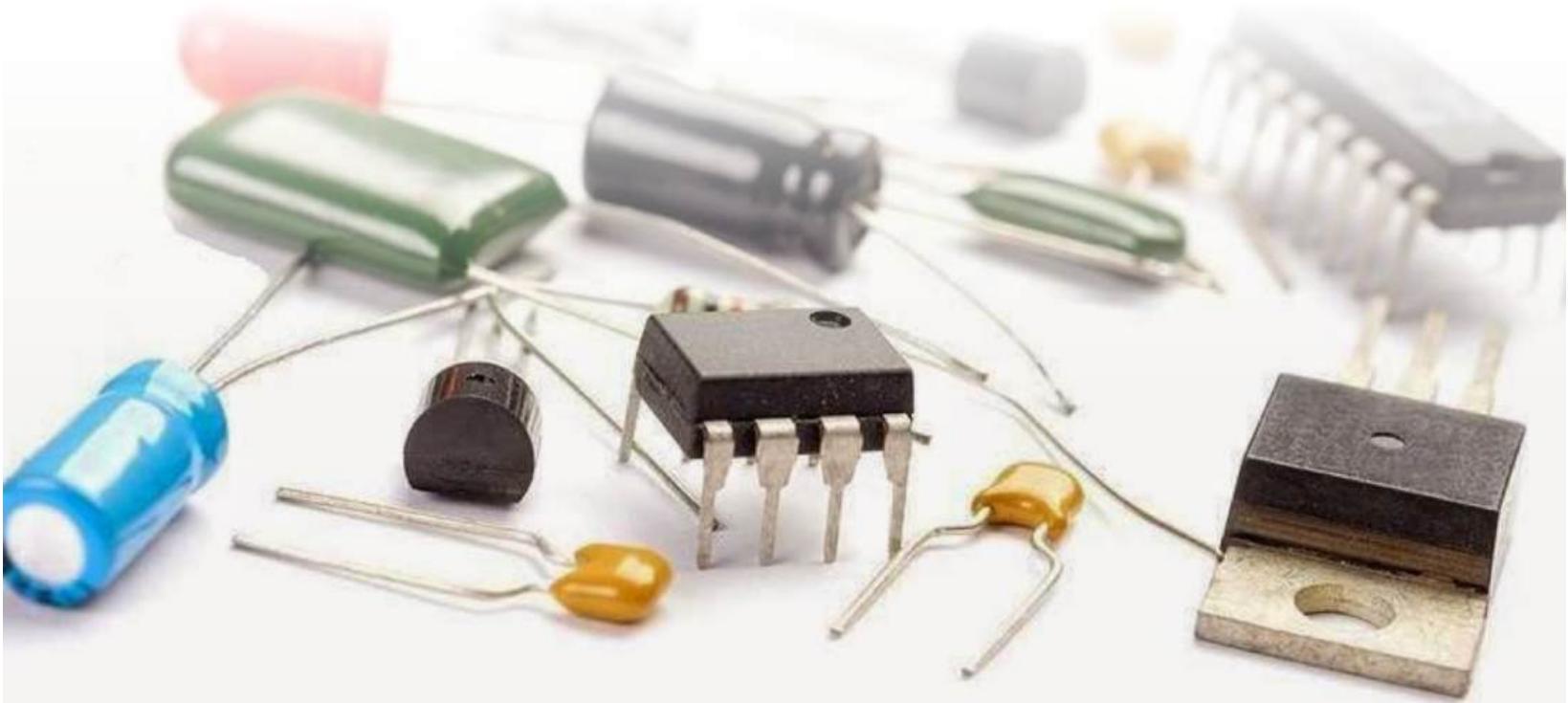
- Live Classes & Recorded Videos
- DPPs & Solutions
- Doubt Solving
- Batches are available in *English*

Weekday & Weekend
Batches Available



JOIN NOW!

Digital Logic



Digital Logic

INDEX

1. Logic Gate 3.1 – 3.12
2. Minimization of Boolean Function 3.13 – 3.22
3. Combinational Circuits 3.23 – 3.42
4. Sequential Logic Circuits 3.43 – 3.67
5. Number System 3.68 – 3.75



1

LOGIC GATE

1.1. Logic Operations

In Boolean algebra, all the algebraic functions performed is logical. The AND, OR and NOT are the basic operations that are performed in Boolean algebra. There are some derived operations such as NAND, NOR, EX-OR, EX-NOR that are also performed in Boolean algebra.

1.1. NOT Operation

Symbol:



Fig. 1.1.

$$A \xrightarrow{\text{NOT}} \bar{A} \text{ or } A' \text{ (Complementation law)}$$

$$\text{and } \bar{\bar{A}} = A \Rightarrow \text{Double complementation law}$$

Truth table for NOT operation

Input A	Output $Y = \bar{A}$
0	1
1	0

A NOT gate can be represented using switch whose circuit representation is shown in figure below.

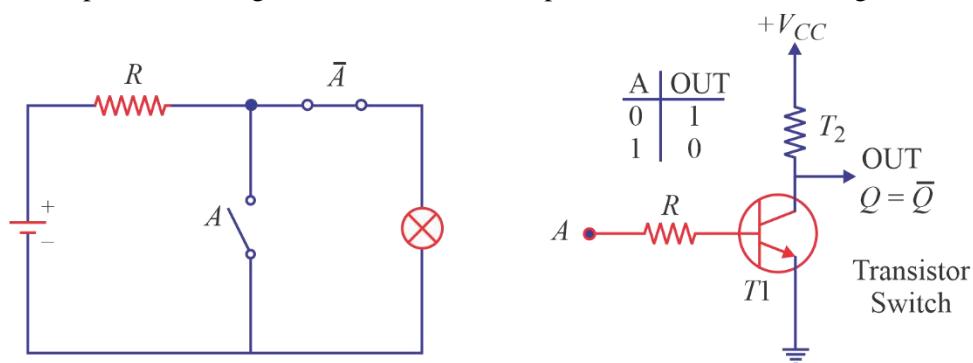
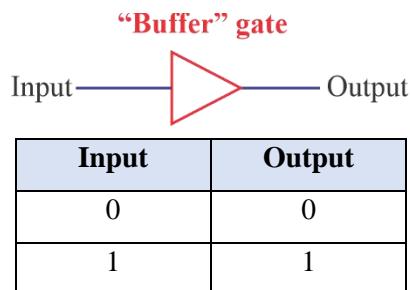


Fig. 1.2.

A buffer is a basic logic gate that passes its input, unchanged, to its output. Its behaviour is the opposite of a NOT gate.

The main purpose of a buffer is to regenerate the input, usually using a strong high and a strong low. Buffers are also used to increase the propagation delay of circuits by driving the large capacitive loads.



1.1.2. AND Operation

Symbol:



$$A \cdot A = A, A \cdot 0 = 0, A \cdot 1 = A, A \bar{A} = 0$$

Truth table for AND operation:

Input		Output
A	B	Y = AB
0	0	0
0	1	0
1	0	0
1	1	1

1.1.3. OR Operation

Symbol:



$$A + A = A, A + 0 = A, A + 1 = 1, A + \bar{A} = 1$$

Truth table for OR operation:

Input		Output
A	B	Y = A+B
0	0	0
0	1	1
1	0	1
1	1	1

Example: Reduce the combinational logic circuit shown figure such that the desired output can be obtained using only one gate.

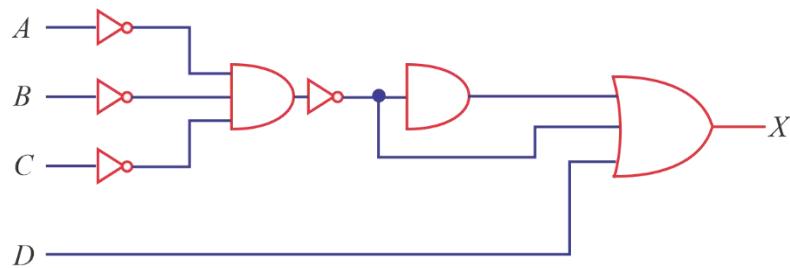


Fig. 1.3.

Solution:

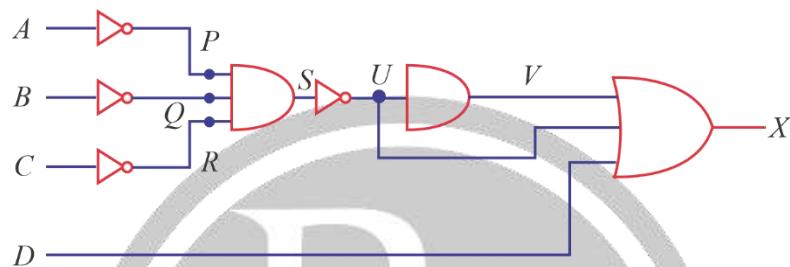


Fig. 1.4.

$$P = \bar{A}, Q = \bar{B}, R = \bar{C}$$

$$S = \bar{A} \cdot \bar{B}$$

$$V = U = \overline{\bar{A}\bar{B}\bar{C}}$$

$$X = U + V + D$$

$$= \overline{\bar{A}\bar{B}\bar{C}} + \overline{\bar{A}\bar{B}\bar{C}} + D$$

$$= A + B + C + D$$



Fig. 1.5.

Enable or Disable Input:

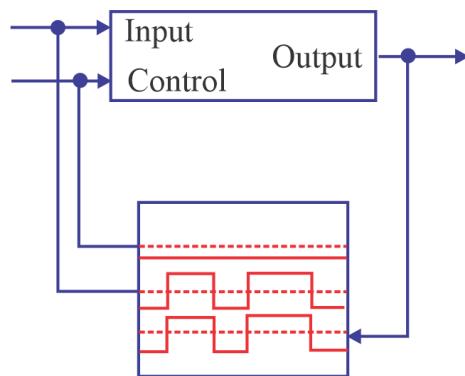


Fig. 1.6.

Enable:

- Allow a signal to pass when the control signal is HIGH.
- Prevent a signal from passing when the control signal is LOW.

Disable:

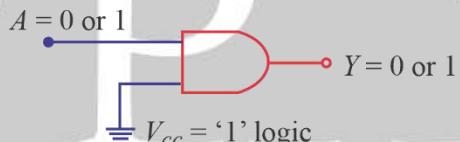
- Prevent a signal from passing when the control signal is HIGH.
- Allow a signal to pass when the control signal is LOW.
- Enable and Disable Functions:
- AND and OR gates can both be used to enable or disable a transmitted waveform.

For a two input AND gate:**For a two input OR gate:**

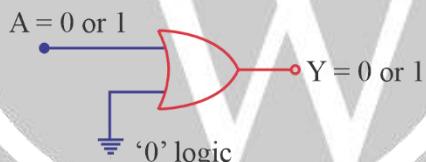
- Control '0' disable



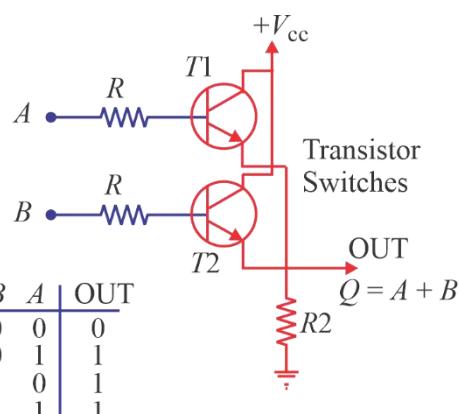
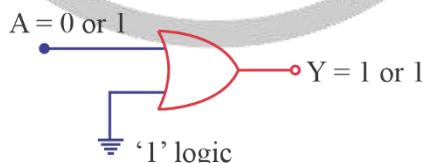
- Control '1' enable (Buffer)



- Control '0' enable (Buffer)



- Control '1' Always enable



1.1.4. Switch Diagram for AND/OR gate

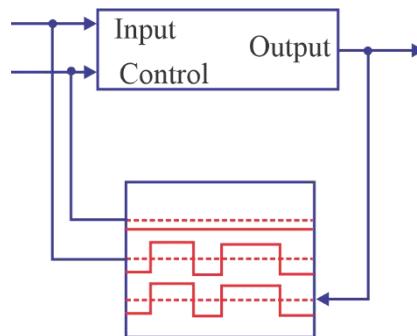


Fig. 1.7.

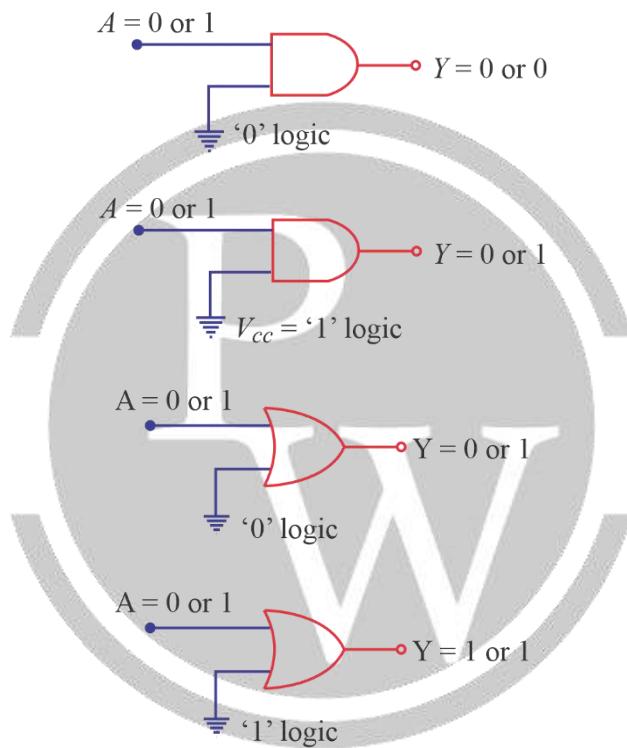


Fig. 1.8.

1.1.5. Basic law applications in AND/OR gate.

(a) Commutative Law:

The commutative law allows change in position of AND or OR variables. There are two commutative laws.

$$A + B = B + A$$

$$A \times B = B \times A$$

(b) Associative Law:

$$(A + B) + C = A + (B + C)$$

$$(A \times B) \times C = A \times (B \times C)$$

1.1.6. Circuit Diagram for AND/OR gate.

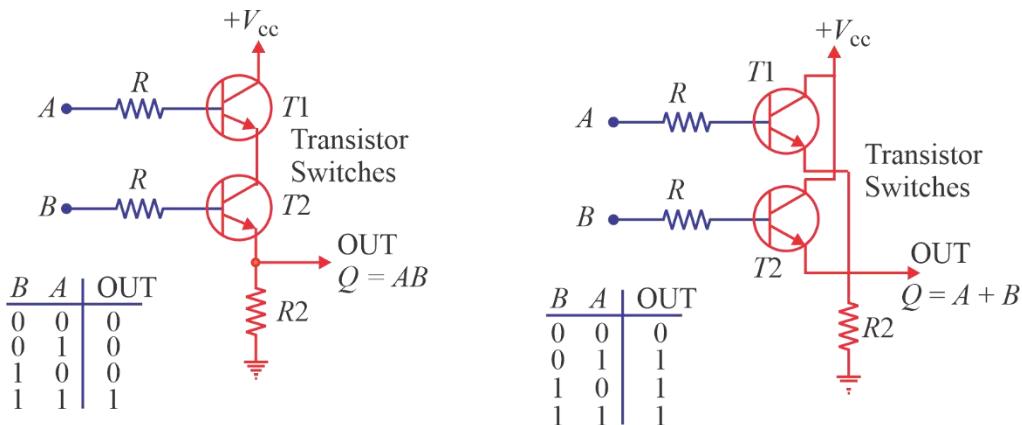


Fig. 1.9.

1.1.7. Switch Diagram for AND/OR Gate

The circuit shown below shows the switch representation of AND gate which is basically the series connection of switches A and B.

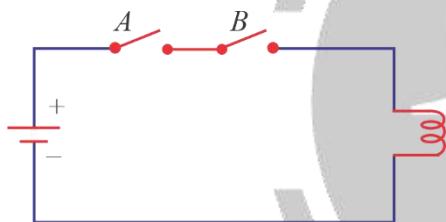


Fig. 1.10.

The circuit shown below shows the switch representation of OR gate which is basically the parallel connection of switches A and B.

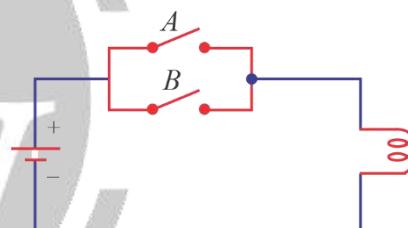


Fig. 1.11.

1.1.8. Venn Diagram:

NOT			\bar{A}	A	Output	
AND			$A \cdot B$	A	B	Output
OR			$A + B$	A	B	Output

1.2. Logic Gates

Logic gates are the fundamental building blocks of digital systems.

Types of logic gates: There are three basic logic gates, namely

- OR gate
- AND gate
- NOT gate

And other logic gates that are derived from these basic gates are:

- NAND gate
- NOR gate
- Exclusive OR gate
- Exclusive NOR gate

1.2.1. NAND gate:

The term NAND gate equivalent to AND gate followed by a NOT gate, implies NOT-AND

Symbol:



Fig. 1.12.

Truth table of 2-input NAND gate.

Input		Output
A	B	$Y = \overline{AB}$
0	0	1
0	1	1
1	0	1
1	1	0

Switching and Circuit Diagram for NAND gate.

B	A	OUT
0	0	1
0	1	1
1	0	1
1	1	0

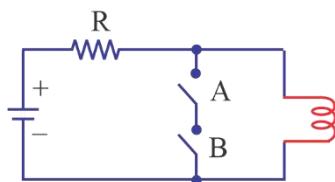


Fig. 1.13.

NAND gate acts as Universal Gate

Logic Gates using only NAND Gates

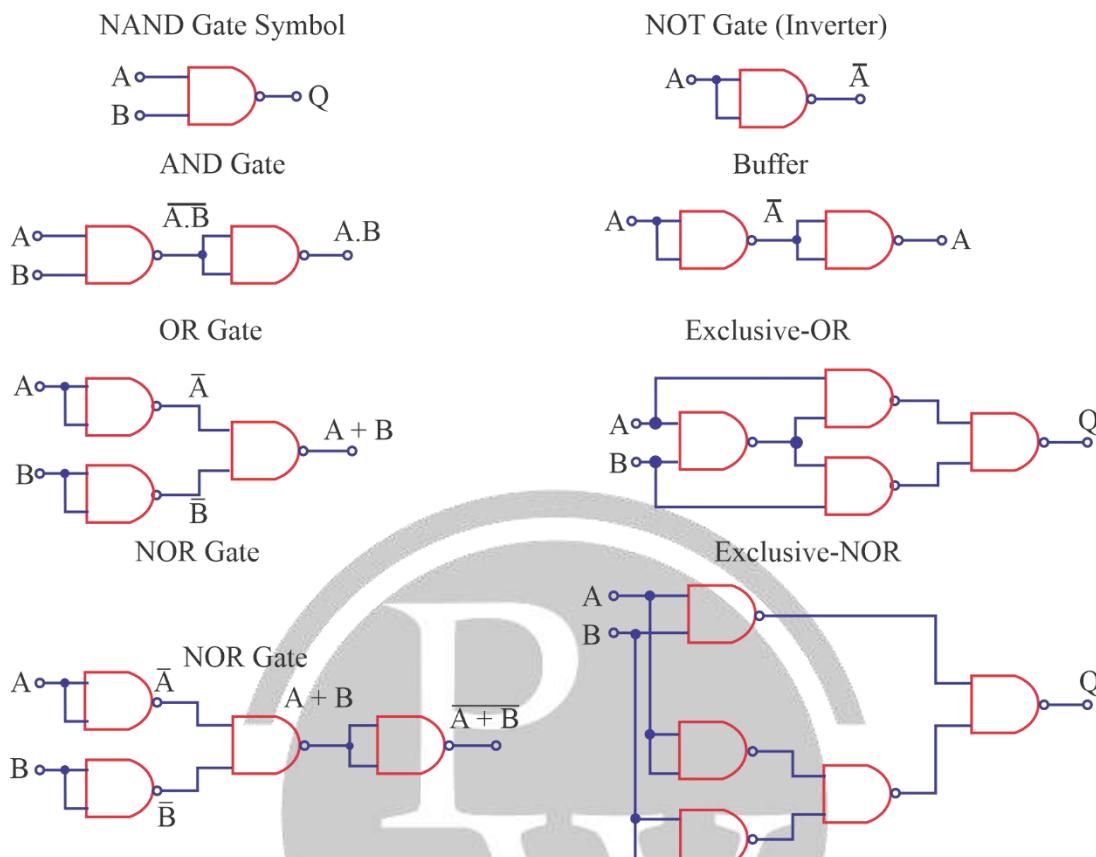


Fig. 1.14.

All the logic gate functions can be created using only NAND gates. Therefore, it is also known as a Universal logic gate.

1.2.2. NOR Gate

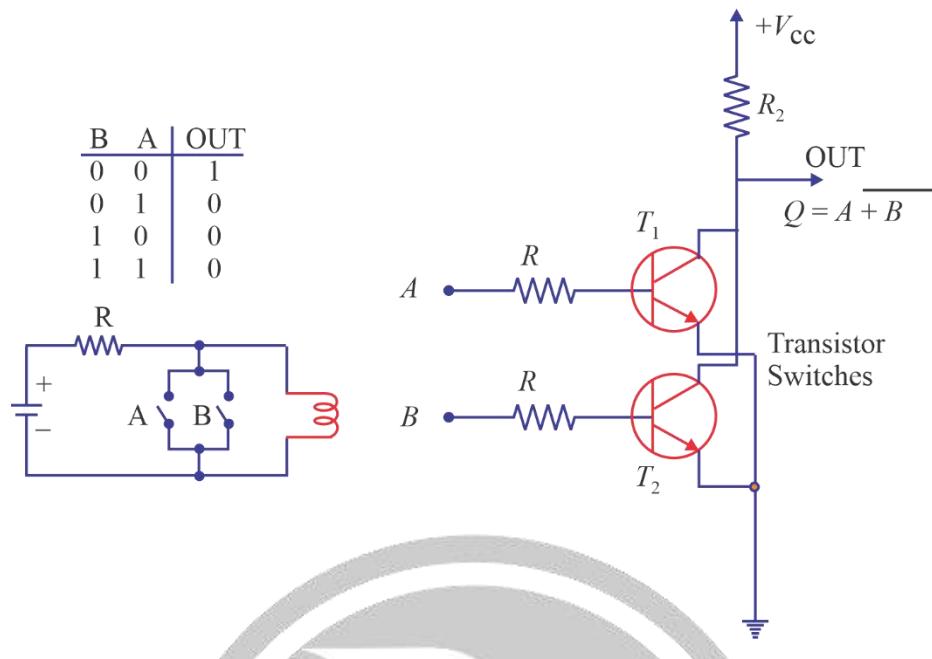
A NOR gate is equivalent to OR gate followed by a NOT gate.

Symbol:

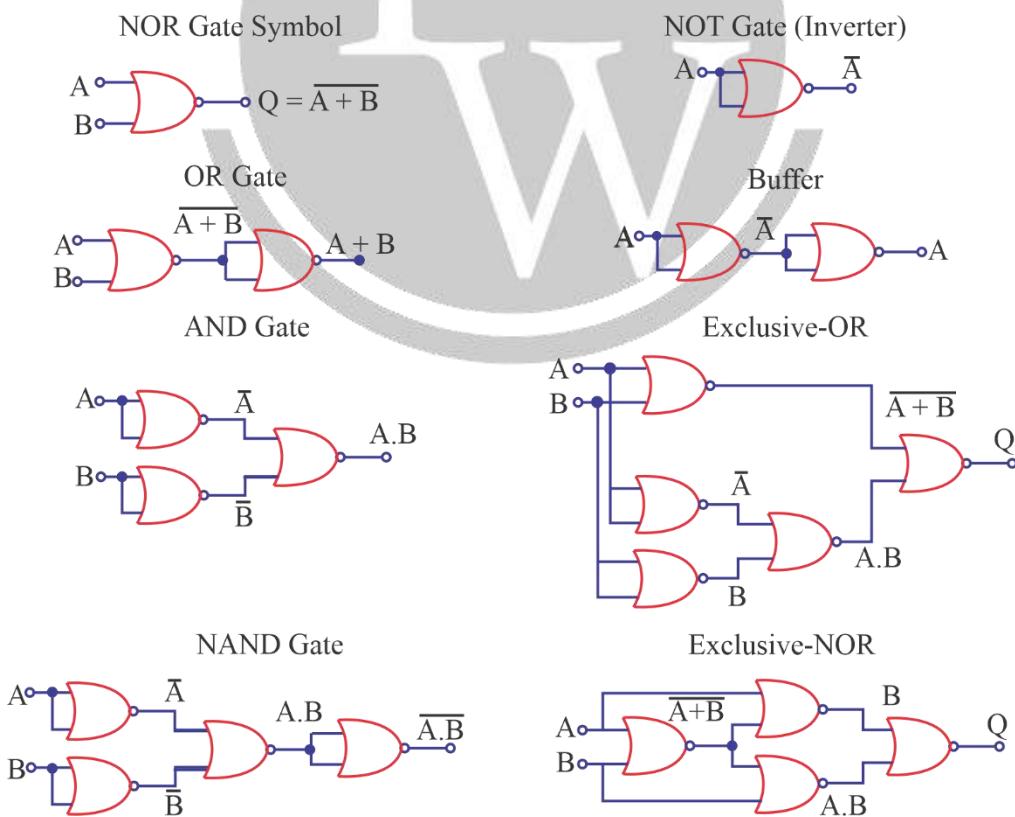


Truth Table for 2-input NOR gate

Input		Output
A	B	$Y = \overline{A+B}$
0	0	1
0	1	0
1	0	0
1	1	0

Switching and Circuit Diagram for NOR gate

Fig. 1.15.

NOR gate acts as Universal Gate.

Logic Gates using only NOR Gates

Fig. 1.16.

All the logic gate functions can be created using only NOR gates. Therefore, it is also known as a Universal logic gate.

1.2.3. XOR Gate

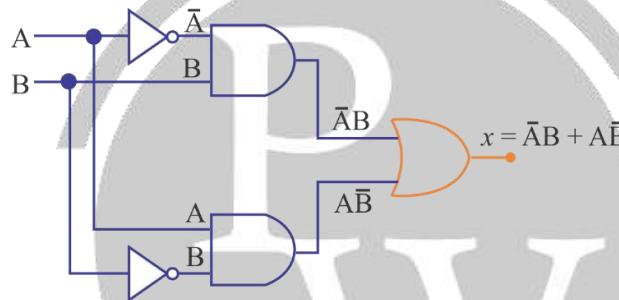
Symbol of two input XOR gate



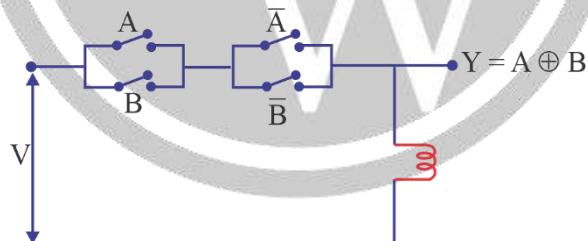
Truth table for 2-input XOR gate

Input		Output
A	B	$Y = A \oplus B$
0	0	0
0	1	1
1	0	1
1	1	0

XOR gate using AND, OR and NOT gate



Switching diagram of XOR gate



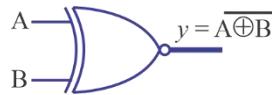
Truth Table:

A	B	Y
0	0	0
0	1	1
1	0	1
1	1	0

$$\begin{aligned}
 Y &= (A + B)(\bar{A} + \bar{B}) \\
 &= \bar{A}B + A\bar{B} \\
 &= A \oplus B
 \end{aligned}$$

1.2.4. X-NOR gate:

Symbol for two input X-NOR gate



Truth table for 2-input X-NOR gate

Input		Output
A	B	$Y = A \odot B$
0	0	1
0	1	0
1	0	0
1	1	1

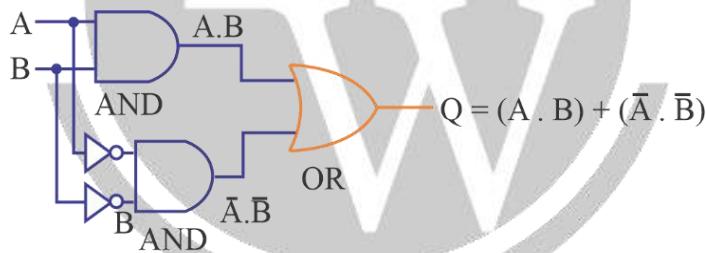
Boolean expression for EX-NOR gate is $Y = A \oplus B$

Apply De-Morgan's theorem:

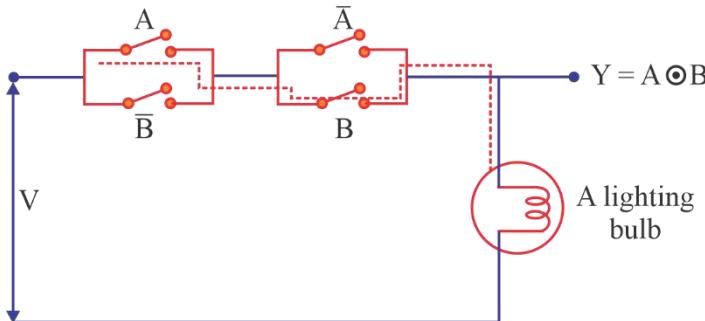
$$\overline{A \oplus B} = \overline{\overline{AB} + \overline{A}\overline{B}} = \overline{AB} \cdot \overline{A}\overline{B} = (A + \overline{B})(\overline{A} + B) = AB + \overline{A}\overline{B}$$

The output of a two input EX-NOR gate is logic '1' when the inputs are same and a logic '0' when they are different.

X-NOR gate using AND OR and NOT gate



Switching Diagram of X-NOR gate



$$\begin{aligned}
 Y &= (A + \overline{B})(\overline{A} + \overline{B}) \\
 &= AB + \overline{A}\overline{B} \\
 &= A \oplus B
 \end{aligned}$$

Truth Table:

A	B	Y
0	0	1
0	1	0
1	0	0
1	1	1

1.3. Alternate Logic Gate Representation

Logic	Normal symbol	Alternate symbol
NOT	$A \rightarrow \bar{A}$	$A \rightarrow \bar{A}$
AND	$A \cdot B = AB$	$A \cdot B = \bar{\bar{A}} + \bar{\bar{B}} = AB$
OR	$A + B = A+B$	$A + B = \bar{\bar{A}} \cdot \bar{\bar{B}} = A+B$
NAND	$\bar{A} \cdot \bar{B} = \overline{AB}$	$\bar{A} + \bar{B} = \overline{A \cdot B}$
NOR	$\bar{A} + \bar{B} = \overline{A+B}$	$\bar{A} \cdot \bar{B} = \overline{A+B}$

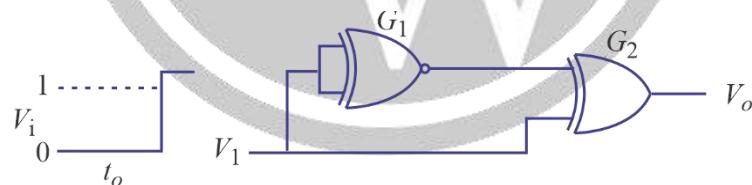
Example: In the following circuit, find the output Z?

Solution: From the given circuit, we can observe that input to last XNOR is same, so, the XNOR output is given by (let input is X)

$$Z = X \cdot X + \bar{X} \cdot \bar{X} = X + \bar{X} = 1$$

i.e. the output will be high [logic 1] irrespective of the inputs A and B.

Example: The gate G_1 and G_2 in figure shown below have propagation delays of 10ns and 20ns respectively.

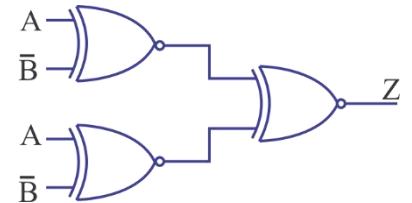
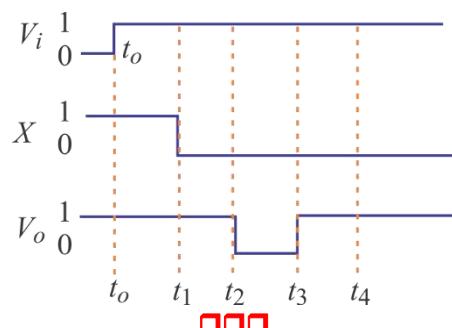


If input V_i makes an abrupt change from logic 0 to 1 at $t = t_0$, then find the output waveform V_o ?

Here, $t_1 = t_0 + 10$ ns, $t_2 = t_1 + 10$ ns, $t_3 = t_2 + 10$ ns.

Solution: Let the output of $G_1 = X$

The output waveform will be as shown in figure below.



2

MINIMIZATION OF BOOLEAN FUNCTION

2.1. Boolean Algebra

Boolean algebra is a system of mathematical logic. It is an algebraic system consisting on the set of elements (0, 1) two binary operators called OR, AND and one unary operator NOT. It is the basic mathematical tools in the analysis and the synthesis of switching circuits. It is a way to express logic functions algebraically.

Note: Any functional relation in Boolean algebra can be provided by the method of perfect induction perfect inductions the method of proof, where by a function relation is verified for every possible combination of values that the value may assume.

Axioms of Boolean Algebra:

Axioms of Boolean algebra are a set of logical expressions that we accept without proof and upon which we can build a set of useful theorem.

	AND operator	OR operator	NOT operators
Axioms 1:	$0 \cdot 0 = 0$	$0 + 0 = 0$	$\bar{1} = 0$
Axioms 2:	$0 \cdot 1 = 0$	$0 + 1 = 1$	$\bar{0} = 1$
Axioms 3:	$1 \cdot 0 = 0$	$1 + 0 = 1$	
Axioms 4:	$1 \cdot 1 = 1$	$1 + 1 = 1$	

Logic operations: In Boolean Algebra all the algebraic function performed is logical. These actually represents logic operations. The AND, OR and NOT are the basic operations that are performed in Boolean Algebra. In addition to these operations, there are some derived operations such as NAND, NOR, EX-OR and EX-NOR that are also performed in Boolean Algebra.

1.1.1. NOT Operation

The NOT operation in Boolean algebra is similar to inversion in ordinary algebra

$$1 : \bar{0} = 1$$

$$2 : \bar{1} = 0$$

$$3 : \text{if } A = 0 \text{ then } \bar{A} = 1$$

$$4 : \bar{\bar{A}} = A \text{ (Double inversion)}$$

1.1.2. AND Operation

It is a logical operation that are performed by AND gate. The AND operation in Boolean Algebra is similar to multiplication in ordinary algebra.

$$1 : A \cdot 0 = 0 \text{ (Null Law)}$$

- 2: $A \cdot 1 = A$ (Identity law)
- 3: $A \cdot A = A$
- 4: $A \cdot \bar{A} = 0$

1.1.3. OR Operation

It is the logical operation that are performed by OR gate. The OR operation in Boolean Algebra is similar to addition in ordinary algebra.

- 1: $A + 0 = A$ (Null law)
- 2: $A + 1 = 1$ (Identity law)
- 3: $A + A = A$
- 4: $A + \bar{A} = 1$

1.1.4. NAND Operation:

The NAND operation in Boolean Algebra is performed by AND operation followed by NOT operation i.e., the negation of AND operation is performed by NAND gate.

1.1.5. NOR Operation:

The NOR operation in Boolean Algebra is performed by OR operation followed by NOT operation i.e., the negation of OR operation is performed by NOR gate.

2.2. Laws of Boolean Algebra

2.2.1. Commutative Law

1. $A + B = B + A$
 $A + B + C = B + C + A = C + A + B = B + A + C$
2. $AB = BA$
 $A \cdot BC = B \cdot CA = C \cdot AB = B \cdot AC$

Violation: Inhibition (1) for Example x/y (x but not y) is not commutative law it means $x/y \neq y/x$

2.2.2. Associative law:

This law arrows grouping of variables

1. $(A + B) + C = A + (B + C)$
 $A + (B + C + D) = (A + B + C) + D$
 $= (A + B) + (C + D)$
2. $(A \cdot B)C = A \cdot (B \cdot C)$
 $A(BCD) = (ABC) \cdot D$
 $A(BCD) = AB \cdot CD$

Note:- NAND and NOR gates are not Associative

2.2.3. Distributive Law

- 1: $A(B + C) = AB + AC$
 $A + BC = (A + B)(A + C)$

2.2.4. Redundant Literal Rule

1. $A + \bar{A}B = A + B$
2. $A(\bar{A} + B) = AB$

2.2.5. Idempotent Law

1. $A \cdot A = A$
2. $A + A = A$

2.2.6. Absorption Law

1. $A + AB = A$
2. $A(A + B) = A$

2.2.7. Involutionary Law

The law that for any variable A.

$$\bar{\bar{A}} = (A')' = A$$

2.2.8. Consensus theorem:

There are two consensus theorems

$$AB + \bar{A}C + BC = AB + \bar{A}C$$

$$(A + B)(\bar{A} + C)(B + C) = (A + B)(\bar{A} + C)$$

2.3. De-Morgan's theorem:

De-Morgan's theorem represents two of the most important rules of Boolean algebra.

$$\text{I. } \overline{A \cdot B} = \bar{A} + \bar{B}$$

$$\text{II. } \overline{A + B} = \bar{A} \cdot \bar{B}$$

The above two laws can be extended for 'n' variables as,

$$\overline{A_1 \cdot A_2 \cdot A_3 \dots + A_n} = \bar{A}_1 + \bar{A}_2 + \dots + \bar{A}_n \text{ and } \overline{A_1 + A_2 + \dots + A_n} = \bar{A}_1 \cdot \bar{A}_2 \dots \bar{A}_n$$

2.3.1 Duality theorem:

Duality Theorem states that,

- (a) Change each OR sign by an AND sign and vice versa.
- (b) Compliment any '0' or '1' appearing in expression
- (c) Keep literals as it is.

Note: With n variables, maximum possible distinct logic function = 2^{2^n}

Example : If a function is given as $f = AB + \bar{A}\bar{B}$ then find its complement.

Solution : Given $f = (AB + \bar{A}\bar{B})$

$$\begin{aligned} \text{Complement of } \bar{f} &= \overline{AB + \bar{A}\bar{B}} = \overline{AB} \cdot \overline{\bar{A}\bar{B}} \\ &= (\bar{A} + \bar{B})(A + B) \\ &= A\bar{A} + A\bar{B} + B\bar{A} + B\bar{B} = A\bar{B} + \bar{A}B \end{aligned}$$

Example : Show that

$$AB + B\bar{C} + AC = AC + B\bar{C}$$

Solution : LHS = $AB + B\bar{C} + AC$

$$\begin{aligned} &= AB(C + \bar{C}) + B\bar{C}(A + \bar{A}) + A(B + \bar{B})C \\ &= ABC + AB\bar{C} + AB\bar{C} + \bar{A}B\bar{C} + ABC + A\bar{B}C \\ &= ABC + AB\bar{C} + \bar{A}B\bar{C} + A\bar{B}C \\ &= AC(B + \bar{B}) + B\bar{C}(A + \bar{A}) = AC + B\bar{C} \\ &= \text{RHS} \end{aligned}$$

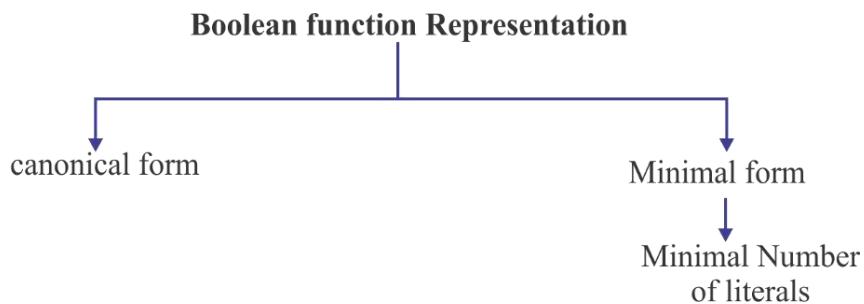
2.4. Minimization of Boolean function:

Every Boolean function expression must be reduced to as simple form as possible before realization because every logic operation in the expression represents a corresponding elements of hardware. Realization of digital circuit with minimal expression has several advantages as:

1. The number of logic gates will reduced.
2. The speed of operation will increase
3. power dissipation will decrease
4. The FAN IN may reduced
5. The complexing of the circuit reduces

The simple method of minimization of Boolean function using certain Algebraic rules which results in the reduction of number of term and/or number of arithmetic operations the various theorem and rules that are already discussed are very useful for the simplification of Boolean expression.

A function of n Boolean variables denoted by $f(A_1, A_2, \dots, A_n)$ is another variable of Algebra and takes one of the two possible values either 0 or 1. The various way of representing a given function are discussed below.



All the terms contain all the variable either in complementary or in uncomplimentary form
The literal means the Binary variable either in complementary or in uncomplimentary form.

2.4.1. Minimization of Boolean function using k-map

- **Using K-map:** The Boolean function can be simplified Algebraically but being not a symmetric method, we can never be sure that whether the minimal expression obtained is the real minimal or not.
- **Karnaugh Map (k-map):** A k-map is a graphical representation of Boolean expression, A two variable k-map will have four cell or squares 3-variable k-map will have 8-cells, n-variable k-map will have 2^n cells.

Note: Adjacent cells differ by 1 bit to maintain adjacently property gray code sequence is used in k-maps (Any two adjacent cells will differ by only one bit)

Min terms & Max terms:

1. n-binary variable have 2^n possible combinations.
2. Min term is a product term, it contains all the variables either complementary or un complementary form for that combination the function output must be '1'.
- Max term is a sum term, it contains all the variables either complementary or uncomplimentary form for that combination the function output must be '0'.

For two variable

x	y	Min terms	Max terms
0	0	$m_0 = \bar{x} \bar{y}$	$M_0 = x + y$
0	1	$m_1 = \bar{x} y$	$M_1 = x + \bar{y}$
1	0	$m_2 = x \bar{y}$	$M_2 = \bar{x} + y$
1	1	$m_3 = x y$	$M_3 = \bar{x} + \bar{y}$

- In min terms we assigns '1' to each uncomplemented variables and '0' to each complemented variable.
- In Max terms we assign '0' to each uncomplemented variable and '1' to each complemented variables.

2.5. Representation of Boolean Functions

Any Boolean expression can be expressed in two forms

- Sum of Product form (SOP)
- Product of Sum form (POS)

2.5.1. SOP Form

The SOP expression usually takes the forms of two or more variables OR together.

$$Y = \bar{A}\bar{B}\bar{C} + A\bar{B}\bar{C} + A\bar{B}C$$

$$Y = A\bar{B} + B\bar{C}$$

SOP forms are used to write logical expression for the output becoming logic '1'.

Example:

Input (3-variables)			Output (Y)
A	B	C	Y
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

∴ Notation of SOP expression is:

$$f(A, B, C) = \Sigma m(3, 5, 6, 7)$$

$$Y = m_3 + m_5 + m_6 + m_7$$

$$\text{Also, } Y = \bar{A}\bar{B}C + A\bar{B}\bar{C} + A\bar{B}C + ABC$$

2.5.2. POS Form

The POS expression usually takes the form of two or more OR variables within parentheses, ANDed with two or more such terms.

Example: $Y = (A + \bar{B} + C)(B\bar{C} + D)$

Each individual term in standard POS form is called maxterm.

POS forms are used to write logical expression for output becoming logic '0'.

we get $f(A, B, C) = \pi M(0, 1, 2, 4)$

$$Y = M_0.M_1.M_2.M_4$$

$$Y = (A + B + C)(A + B + \bar{C})(A + \bar{B} + C)(\bar{A} + B + C)$$

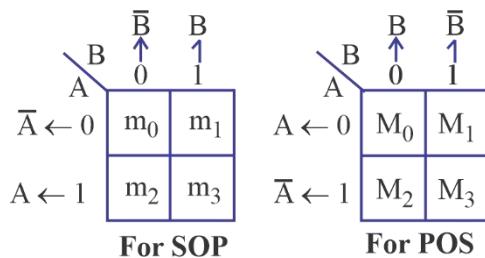
∴ We can also conclude from Table 2 and from above equations:

if $Y = \Sigma m(3, 5, 6, 7)$ or $Y = \pi M(0, 1, 2, 4)$

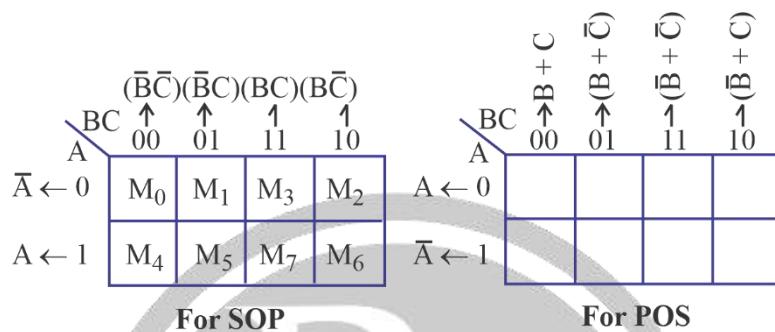
2.6. Karnaugh Map (K-MAP)

The K-map is a graphical method which provides a systematic method for simplifying the Boolean expressions. In n variable K-map, there are 2^n cells.

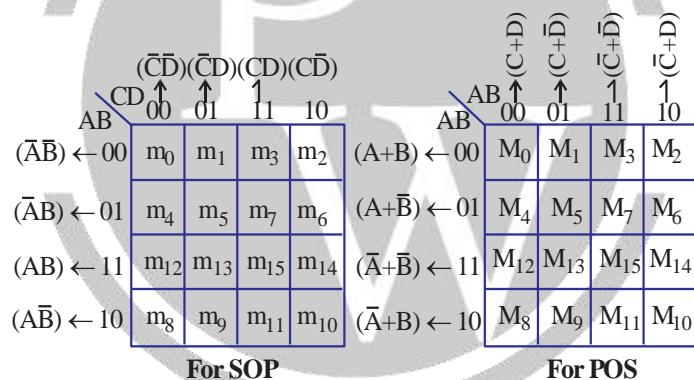
2.6.1. Two variable K-map



2.6.2. Three variable K-map



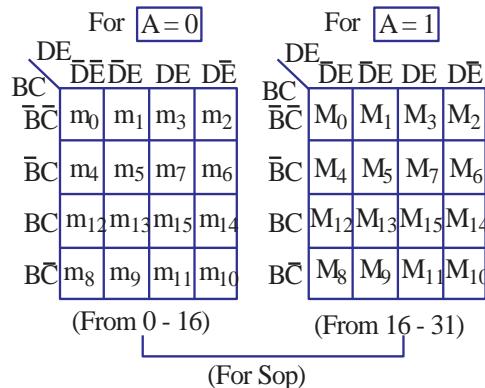
2.6.3. Four Variable K-map



2.6.4. Five variable K-map

- 32 cells
- 32 Minterms (Maxterms)

Here, we have $f(A, B, C, D, E)$



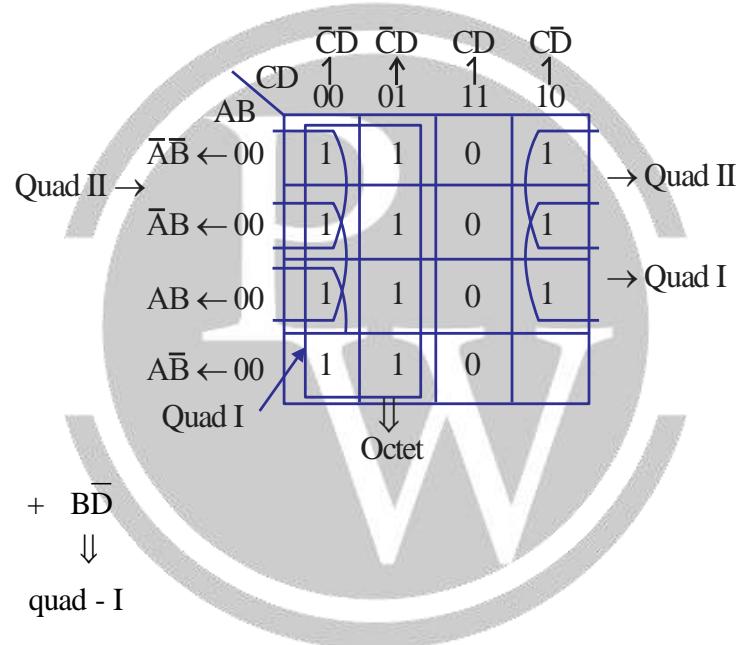
2.7. Simplification Rules

1. Construct the K-map and place 1's in those cells corresponding to the 1's in the truth table. Place the 0's in the other cells.
2. Examine the map for adjacent 1's and loop those 1's which are not adjacent to any other 1's. These are called isolated 1's.
3. Next, look for those 1's which are adjacent to only one other 1. Loop any pair containing such a 1.
4. Loop any octet even if it contains some 1's that have already been looped.
5. Loop any quad that contains one or more 1's which have not already been looped, making sure to use the minimum number of loops.
6. Form the OR sum of all the terms generated by each loop.

Example: Simply a four variable logic function using K-map

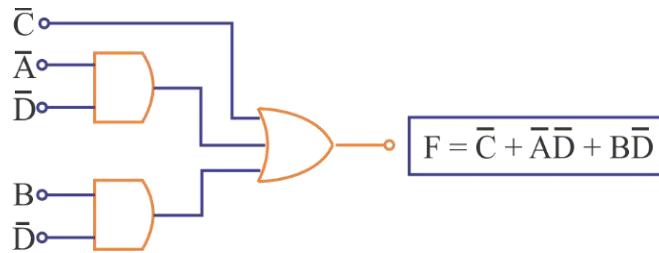
$f(A, B, C, D) = \Sigma m(0, 1, 2, 4, 5, 6, 8, 9, 12, 13, 14)$ also implement the simplified expression with AND-OR logic.

Solution:



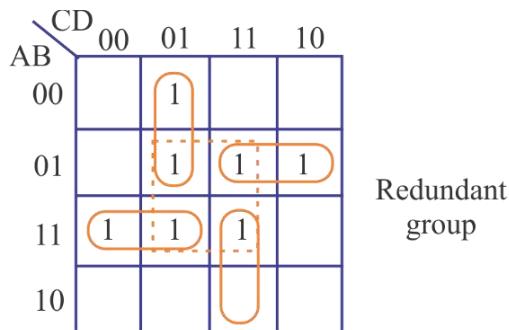
$$\begin{aligned} f &= \bar{C} + \bar{A}\bar{D} + BD \\ \therefore &\quad \downarrow \quad \downarrow \quad \downarrow \\ \text{octet} &\quad \text{quad - II} \quad \text{quad - I} \end{aligned}$$

⇒ Gate implementation:



2.8. Redundant Group

If all the 1's in a group are already involved in some other groups, then that group is caused as a redundant group. A redundant group has to be eliminated, because it increases the no of gates required.



2.9. Don't Care Condition

The combinations for which the values of the expression are not specified are called don't care conditions.

Example: Simplify the given equation in part (i) and (ii)

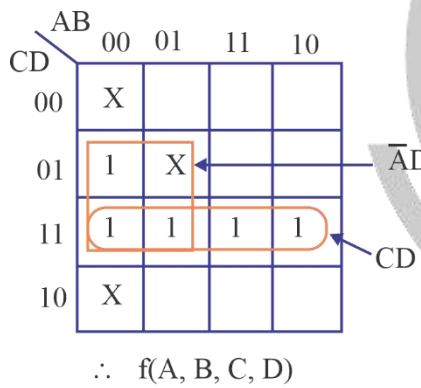
(i) In terms of SOP. and don't care conditions

$$f(A, B, C, D) = \Sigma m(1, 3, 7, 11, 15) + d(0, 2, 5)$$

(ii) In terms of POS and don't care conditions.

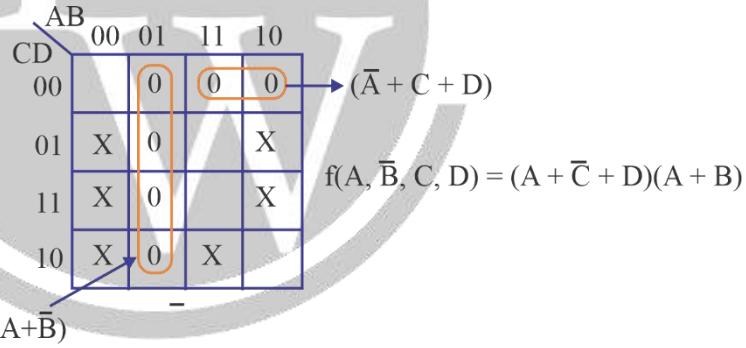
$$f(A, B, C, D) = \pi M(4, 5, 6, 7, 8, 12) + d(1, 2, 3, 9, 11, 14)$$

Solution:



$$\therefore f(A, B, C, D)$$

(i)



(ii)

2.10. Implicants, Prime Implicants and Essential Prime Implicants

2.10.1. Implicants

Implicants is a product term on the given function for that combination the function output must be 1.

2.10.2. Prime Implicant (PI)

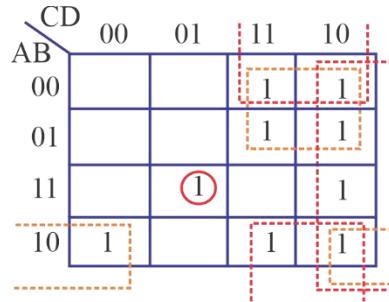
Prime implicant is a smallest possible product term of the given function,

2.10.3. Essential Prime Implicants (EPI)

EPI is a prime implicant it must cover at least one minterms, which is not covered by other PI.

Example: Reduce the expression using mapping $F = \Sigma m(2, 3, 6, 7, 8, 10, 11, 13, 14)$

Solution :

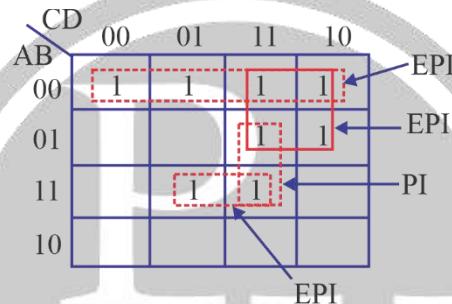


$$F = AB\bar{C}D + A\bar{B}\bar{D} + \bar{A}C + \bar{B}C + C\bar{D}$$

Example : Reduce the following expression using k-map and identify PI's and EPI

$$F = \Sigma m(0, 1, 2, 3, 6, 7, 13, 15)$$

Solution :



$$EPI = \bar{A}\bar{B}, \bar{A}C, ABD$$

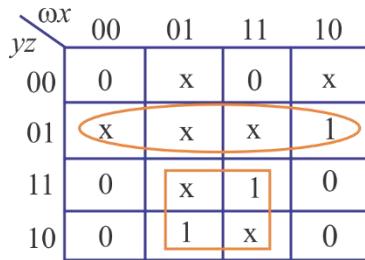
$$PI = BCD$$

$$\text{Minimal } F = \bar{A}\bar{B} + \bar{A}C + ABD$$

Example: Given the following Karnaugh map, which one of the following represents the minimal sum of products of the map.

- A. $xy + \bar{y}z$
- B. $\omega\bar{x}\bar{y} + xy + xz$
- C. $\bar{\omega}x + \bar{y}z + xy$
- D. $xy + y$

Solution :



$$= xy + \bar{y}z$$



	00	01	11	10
00	0	x	0	x
01	x	x	x	1
11	0	x	1	0
10	0	1	x	0

3

COMBINATIONAL CIRCUITS

3.1. Combinational Circuits

The combinational circuit has ‘ n ’ input variables and ‘ m ’ output variables. Since, the number of input variables is n , there are 2^n possible combinations of bits at the input. Each output can be expressed in terms of input variables by a Boolean expression.

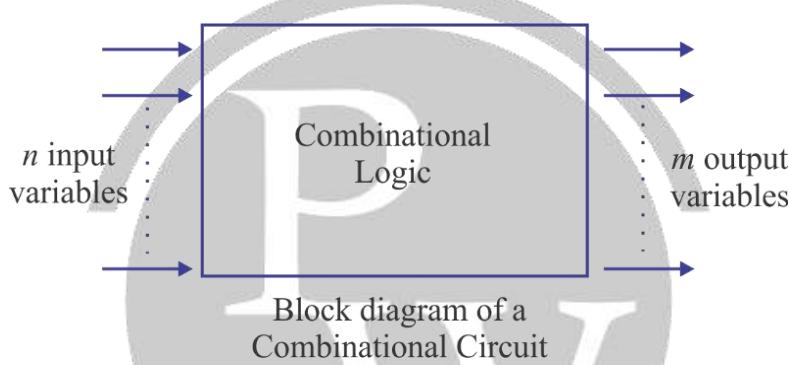


Fig. 3.1. Block diagram of a Combinational Circuit

3.2. Adders

The most basic arithmetic operation is the addition of two binary digits. A combinational circuit that performs the addition of two 1-bit numbers is called as half adder, and the logic circuit that adds three 1-bit numbers is called as full adder.

3.2.1. Half Adder

The logic circuit that performs the addition of two 1-bit numbers is called as half adder. It is the basic building block for addition of two single bit numbers. This circuit has two outputs namely carry (C) and sum (S).

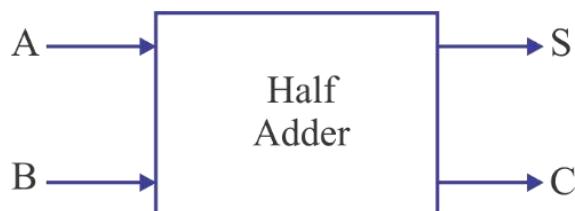


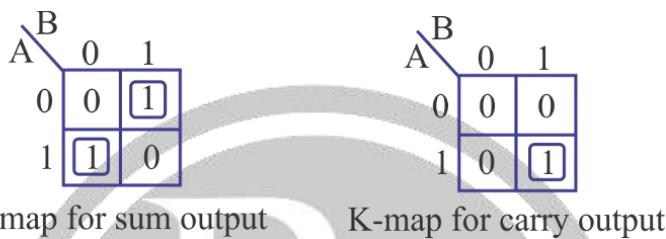
Fig. 3.2. Block Diagram of a 2-bit Half Adder

The truth table of half adder: where A and B are the inputs and sum and carry

Table 1: Truth Table of Half Adder

Inputs		Outputs	
A	B	Sum (S)	Carry (C)
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

K-map simplification for Carry and Sum: Boolean expressions for the sum (S) and carry (C) outputs from K – maps:



$$\text{Sum, } S = \bar{A}\bar{B} + A\bar{B} = (A \oplus B)$$

$$\text{Carry, } C = AB$$

Logic Diagram:



Fig. 3.3. Logic Diagram of Half Adder

3.2.2. Full Adder

A full adder circuit is an arithmetic circuit block that can be used to add three bits to produce a sum and a carry output. Let us consider A and B as two 1-bit inputs & C_{in} is a carry generated from the previous order bit additions. Let S (sum) and C_{out} (carry) are the outputs of the full adder.

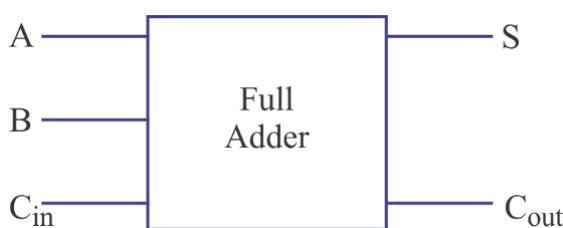


Fig. 3.4. Block Diagram of a Full Adder

The Truth Table for Full Adder is given as:

Table: Truth Table for Full Adder

Inputs			Outputs	
A	B	C _{in}	Sum (S)	Carry C _{out}
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

K – map Simplification for Carry and Sum:

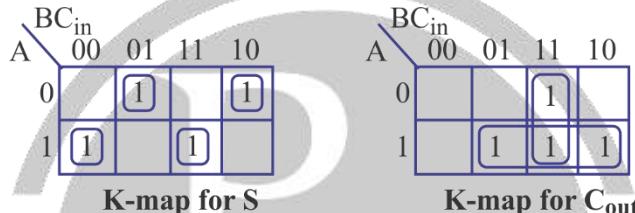


Fig. 3.5.

Simplified Boolean expressions for the sum (S) and carry (C_{out}) output from K-maps is

$$\begin{aligned} \text{Sum, } S &= \bar{A}\bar{B}C_{\text{in}} + \bar{A}B\bar{C}_{\text{in}} + ABC_{\text{in}} + A\bar{B}\bar{C}_{\text{in}} = C_{\text{in}}(\bar{A}\bar{B} + AB) + \bar{C}_{\text{in}}(\bar{A}B + A\bar{B}) \\ &= C_{\text{in}}(A \odot B) + C_{\text{in}}(A \oplus B) \\ &= C_{\text{in}}(\overline{A \oplus B}) + C_{\text{in}}(A \oplus B) \end{aligned}$$

$$\text{Sum, } S = C_{\text{in}} \oplus A \oplus B$$

$$\text{Carry, } C_{\text{out}} = AB + AC_{\text{in}} + BC_{\text{in}}$$

Logic Diagram:

We can realize logic diagram of a full adder using gates as shown in below figure:

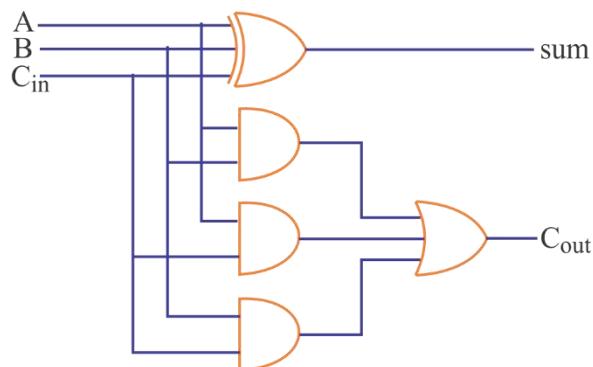


Fig. 3.6. Logic Diagram of Full Adder

Example: A full adder is implemented using two input OR gate and two half adders. Half adder is implemented using two input XOR and two input AND gate. The propagation delays of XOR gate, AND gate and OR gate respectively are 2ns, 1.5ns. and 1ns. The propagation delay of full adder is ns.

Solution:

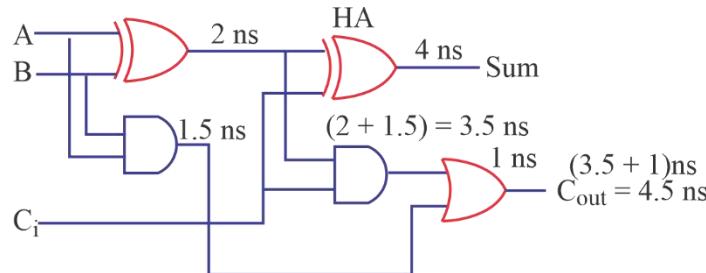


Fig. 3.7.

3.3. Subtractors

3.3.1. Half subtractor

A half subtractor is a combinational logic circuit, which performs the subtraction of two 1-bit numbers. It subtracts one binary digit from another to produce a DIFFERENCE output and a BORROW output.

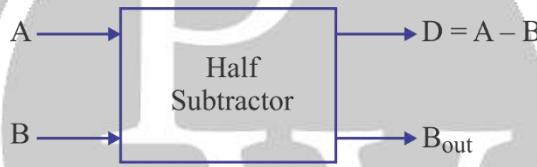


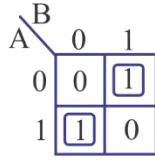
Fig. 3.8. Block diagram of a half subtractor

The truth table of half – subtractor, where A, B are the inputs, and difference (D) and borrow (B_{out}) are the outputs.

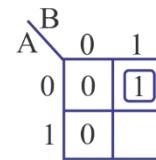
Table: Truth table of Half – Subtractor

Inputs		Outputs	
A	B	D	B _{out}
0	0	0	0
0	1	1	1
1	0	1	0
1	1	0	0

K – map Simplification for Difference and Borrow:



K-map for difference output



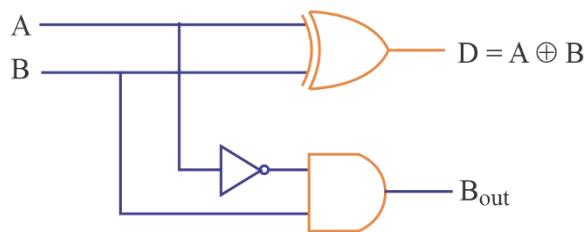
K-map for borrow output

Difference,

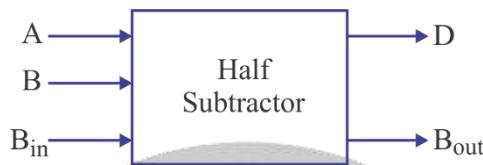
$$D = \bar{A}B + A\bar{B} = A \oplus B$$

Borrow,

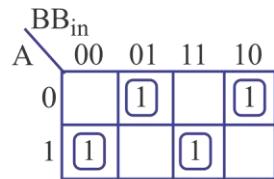
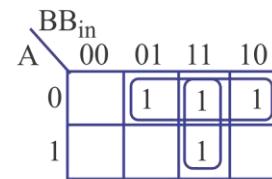
$$B_{out} = \bar{A}B$$

Logic Diagram:

Fig. 3.9. Logic Diagram of a Half subtractor
3.3.2. Full Subtractor

A full subtractor performs subtraction operation on two bits, a minuend and a subtrahend.


Fig. 3.10. Block Diagram of a Full subtractor
Truth table of Full subtractor:
Table: Truth table of Full subtractor

Inputs			Outputs	
A	B	B _{in}	D	B _{out}
0	0	0	0	0
0	0	1	1	1
0	1	0	1	1
0	1	1	0	1
1	0	0	1	0
1	0	1	0	0
1	1	0	0	0
1	1	1	1	1

K – map simplification for Difference and Borrow:

K-map for difference output

K-map for borrow output

Difference,

$$\begin{aligned}
 D &= \bar{A}\bar{B}B_{in} + \bar{A}B\bar{B}_{in} + A\bar{B}\bar{B}_{in} + AB{B}_{in} \\
 &= B_{in}(AB + \bar{A}\bar{B}) + \bar{B}_{in}(A\bar{B} + \bar{A}B) \\
 &= B_{in}(\overline{A\bar{B}}) + B_{in}(A\bar{B})
 \end{aligned}$$

$$D = A \oplus B \oplus B_{in}$$

Borrow,

$$B_{in} = \bar{A}B + \bar{A}B_{in} + BB_{in}$$

Logic Diagram

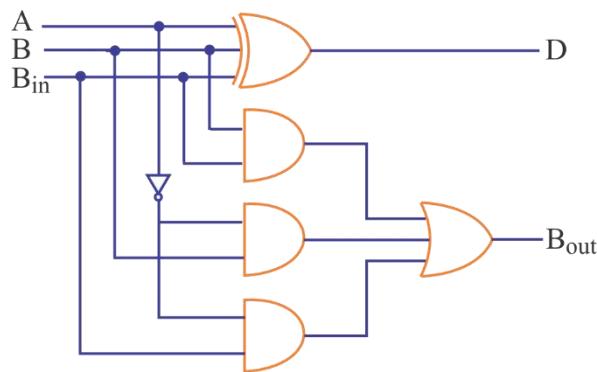


Fig. 3.11. Logic Diagram of a Full Subtractor

3.4. Binary Parallel Adder

An n-bit parallel adder can be constructed using n number of full adders are connected in parallel and hence; it is also known as parallel adder such that the previous carry or carry input for adder 0 is set to zero. The carry output of each adder is connected to the carry input of the next higher order adder. Hence, it is also known as carry propagate adder.

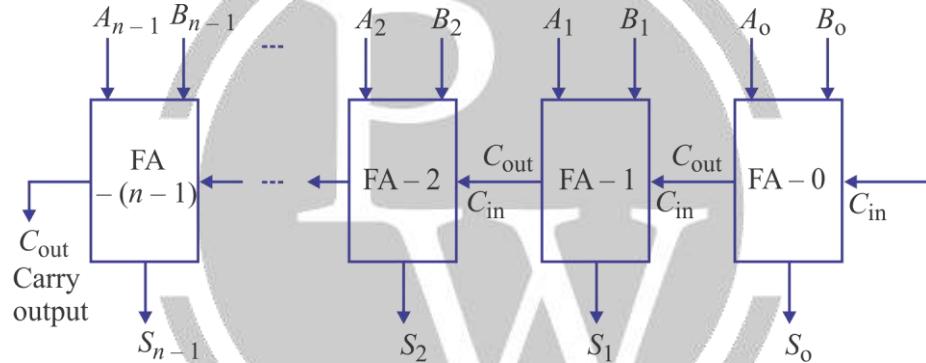


Fig. 3.12. n-bit Binary Adder

3.4.1. Propagation Delay in Parallel Adder:

Parallel adders suffer from propagation delay problem because higher bit additions depend on the carry generated from lower bit addition. In effect, carry bits must propagate or ripple through all stages before the most significant sum bit is valid. Thus, the total sum (the parallel output) is not valid until after the cumulative delay of all the adder.

3.5. Carry Look Ahead Adder

The look ahead carry adder speeds up the operation by eliminating this ripple carry delay. It examines all the input bits simultaneously and also generates the carry in bits for all the stages simultaneously. The method of speeding up the addition process is based on the two additional functions of the full adder called the carry generate and carry propagate functions.

3.5.1. Carry Generation

Carry is generated only if both the input bits are 1, that is, if both the bits A and B are 1's, a carry has to be generated in this stage regardless of whether the input carry C_{in} is a 0 or a 1. Let G as the carry generation function,

$$G = A \cdot B$$

Consider the present bit as the nth, then

$$G_n = A_n \cdot B_n$$

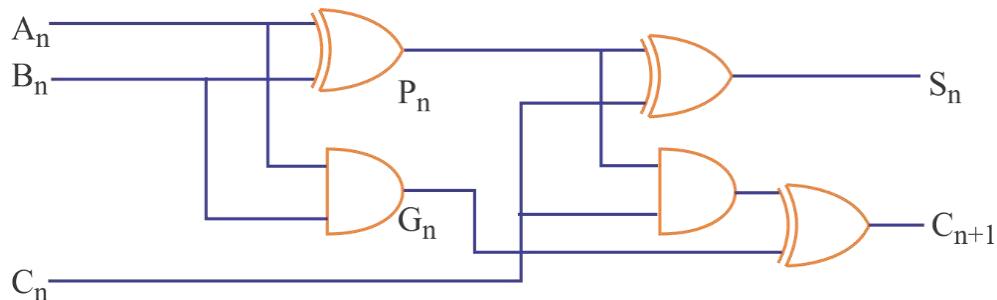


Fig. 3.13. Carry Look – ahead Generator Circuit

3.5.2. Carry Propagation

A carry is propagated if any one of the two input bits A or B is 1. If both A and B are 0, a carry will never be propagated. On the other hand, if both A and B are 1, then will not propagate the carry but will generate the carry. Let P as the carry – propagation function, then

$$P_n = A_n \oplus B_n$$

3.5.3. Look ahead Expressions

Let n^{th} bit adder, the sum (S) and the carry out (C) for the n^{th} bit may be expressed in terms of the carry generation function (G) and the carry propagation function (P) as

$$S_n = P_n \oplus C_n$$

$$C_{n+1} = G_n + P_n \cdot C_n$$

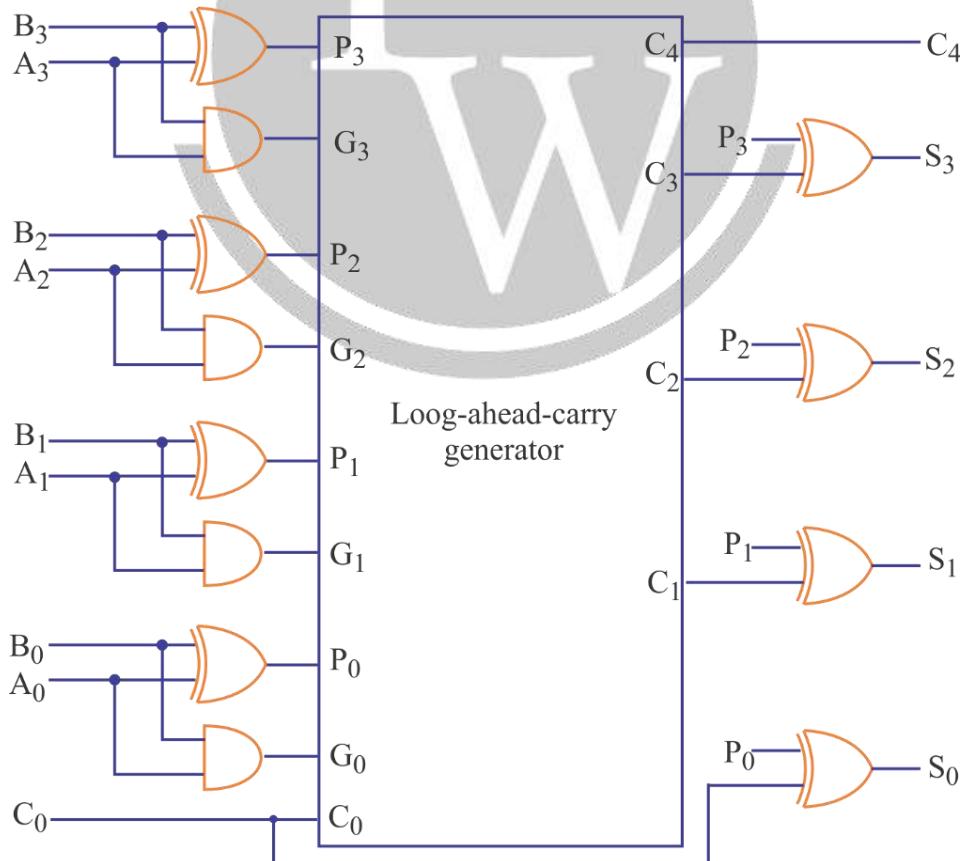


Fig. 3.14. 4-bit Full Adder with a look Ahead Carry Generator

Example: A full adder can be realized using half adder? Explain it in detail.

Solution: A full adder realization using half adder is given by:

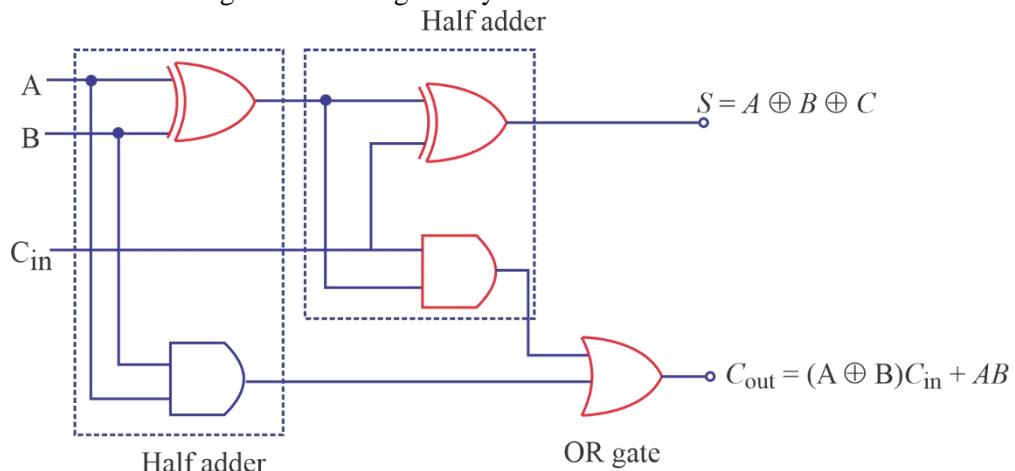


Fig. 3.15.

3.6. Comparator

The comparator is a combinational logic circuit. It compares the magnitude of two n-bit numbers and provides the relative result as the output. Let A and B are the two n-bit inputs. The comparator has three outputs namely $A > B$, $A = B$ and $A < B$. Depending upon the result of comparison, one of these outputs will go high.

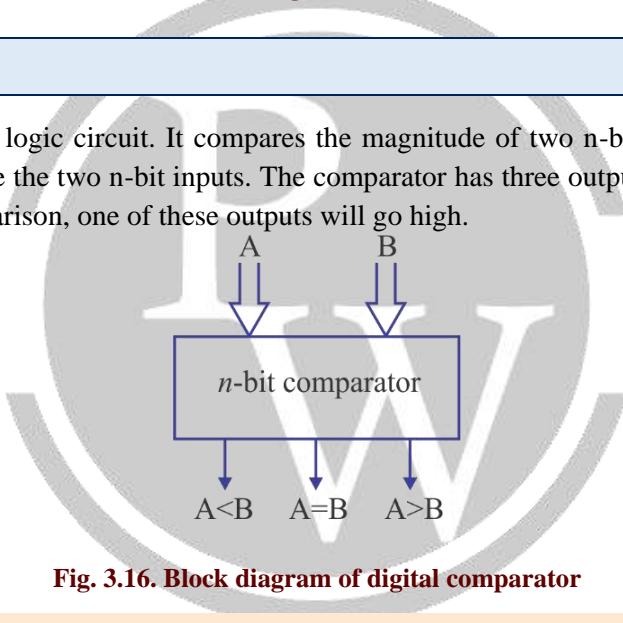


Fig. 3.16. Block diagram of digital comparator

3.6.1. 1-bit Magnitude Comparator

The 1-bit comparator is a combinational logic circuit with two inputs A and B and three outputs namely $A < B$, $A = B$ and $A > B$.

Table : Truth Table of a 1-bit Comparator

Inputs		Outputs		
A	B	X ($A < B$)	Y ($A = B$)	Z ($A > B$)
0	0	0	1	0
0	1	1	0	0
1	0	0	0	1
1	1	0	1	0

Design of 1-bit Magnitude Comparator: We can write the expressions for the three outputs as under:

For $(A < B)$, $X = \bar{A}_0 B_0$

For $(A = B)$, $Y = \bar{A}_0 \bar{B}_0 + A_0 B_0 = \overline{\bar{A}_0 \oplus B_0}$

For $(A > B)$, $Z = A_0 \bar{B}_0$

Logic Diagram of 1-bit Comparator:

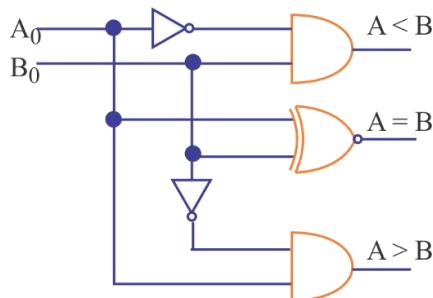


Fig. 3.17. Logic Diagram of 1-bit Comparator

Example: The circuit shown in given figure is:

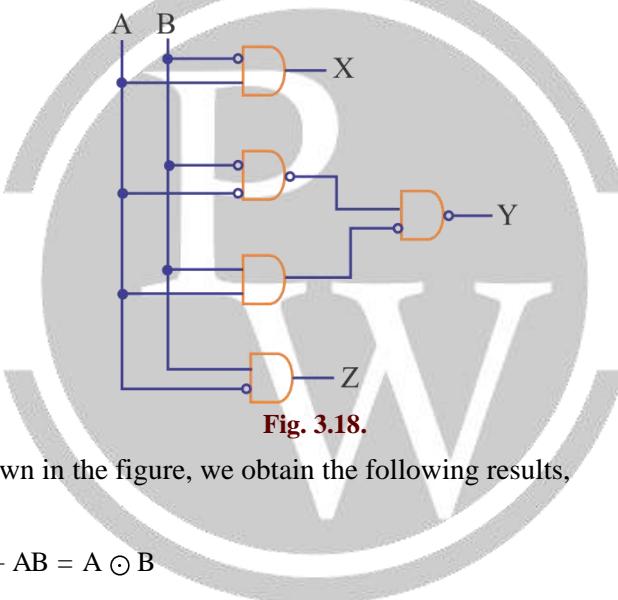


Fig. 3.18.

Solution: From the logic circuit shown in the figure, we obtain the following results,

$$X = A\bar{B}$$

$$Y = \overline{(\bar{A}\bar{B})(AB)} = \bar{A}\bar{B} + AB = A \odot B$$

$$Z = \bar{A}\bar{B}$$

So, we obtain the truth table for the above function as shown below.

From truth table, we deduce the following results.

If $A > B$, then $x = 1$

If $A = B$, then $y = 1$

If $A < B$, then $z = 1$

Therefore, it is a comparator circuit.

A	B	X	Y	Z
0	0	0	1	0
0	1	0	0	1
1	0	1	0	0
1	1	0	1	0

3.7. Multiplexer

A multiplexer, abbreviated as MUX, is a digital switch which selects one of the many inputs to a single output. A number of control lines determine which input data is to be routed to the output. If there are n select lines, then the number of maximum possible input lines is 2^n and the multiplexer is referred to as a 2^n -to-1 multiplexer or $2^n \times 1$ multiplexer.

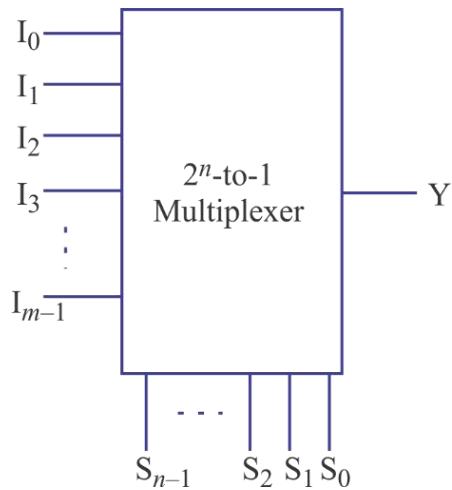


Fig. 3.19. Block diagram of a 2^n to 1 multiplexer

3.7.1. 2×1 MUX

A 2 to 1 multiplexer has 2 inputs. Since $2 = 2^1$, this multiplexer will have one control (select) line. It has two data inputs I₀ and I₁, one select input S, and one output.

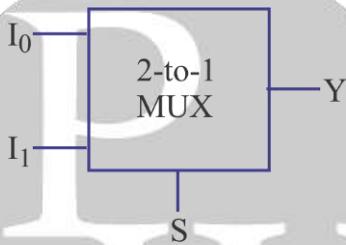


Fig. 3.20. Schematic block diagram of 2:1 Multiplexer

The truth table of this MUX is given below,

Select Line (S)	Output Y
0	I ₀
1	I ₁

Thus, the SOP expression for the output Y is,

$$Y = I_0 \bar{S}_0 + I_1 S_0$$

Realization of a 2:1 MUX using Logic Gates:

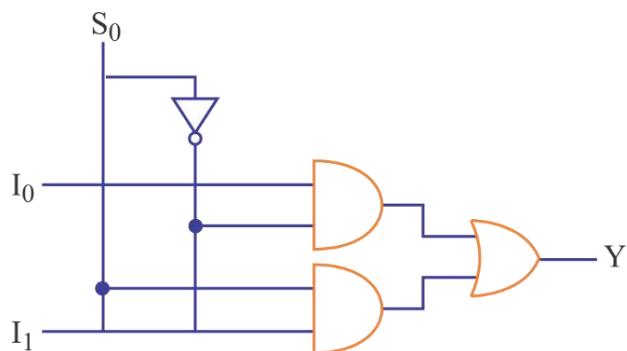


Fig. 3.21. Logic Diagram of a 2×1 Multiplexer

2.7.2. 4×1 MUX

A 4-to-1 multiplexer has 4 inputs and two select lines, where I_0 to I_3 are the four inputs to the multiplexer, and S_0 and S_1 are the select lines.

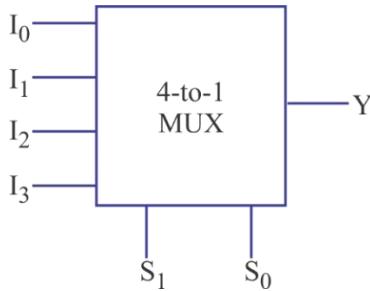


Fig. 3.22. Schematic block diagram of 4×1 MUX

Truth Table of a 4-to-1 Multiplexer

Select Inputs		Output
S_1	S_0	Y
0	0	I_0
0	1	I_1
1	0	I_2
1	1	I_3

Output Y for a 4-input multiplexer is

$$Y = I_0 \bar{S}_1 \bar{S}_0 + I_1 \bar{S}_1 S_0 + I_2 S_1 \bar{S}_0 + I_3 S_1 S_0$$

3.8. Implementation of Higher Order Mux Using Lower Order MUX

The methods for implementing higher order MUX using lower order MUX are

- Step 1:** If 2^n is the number of input lines in the available lower order multiplexer and 2^N is the number of input lines in the desired multiplexer, then the number of lower order multiplexers required to construct the desired multiplexer circuit would be $2^N - n$.
- Step 2:** From the knowledge of the number of selection inputs of the available multiplexer and that of the desired multiplexer, connect the less significant bits of the selection inputs of the desired multiplexer to the selection inputs of the available multiplexer.
- Step 3:** The most significant bits of the selection inputs of the desired multiplexer circuit are used to enable or disable the individual multiplexers so that their outputs when OR produce the final output.

Example: In realization of $32 : 1$ MUX using $2 : 1$ MUX, the required number of $2 : 1$ MUX is ?

Solution: In realization of $2^n : 1$ MUX using $2 : 1$ MUX, the required number of $2 : 1$ MUX is $2^n - 1$, since, we have to realize $32 : 1$ MUX, so we have

$$n = 5$$

Hence, the required number of $2 : 1$ MUX is

$$2^n - 1 = 2^5 - 1 = 31$$

3.9. Applications of Multiplexers

1. It is used as a data selector to select one out of many data inputs.
2. They are used in designing the combinational circuits.
3. They are used in digital-to-analog and analog-to-digital converters.
4. They can be used for simplification of logic design.
5. Multiplexers are also used in data acquisition systems.

3.10. Demultiplexer

The demultiplexer is a combinational logic circuit that performs the reverse operation of a multiplexer. The demultiplexer has one input line and m output lines. Again $m = 2^n$, so it requires n select lines. A demultiplexer with one input and m output is called a 1-to- m demultiplexer.

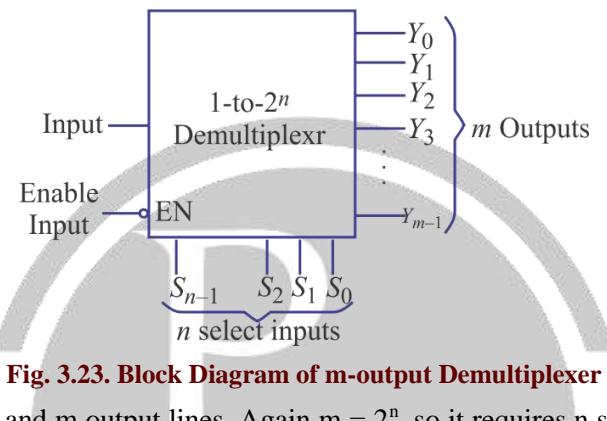


Fig. 3.23. Block Diagram of m -output Demultiplexer

The demultiplexer has one input line and m output lines. Again $m = 2^n$, so it requires n select lines. A demultiplexer with one input and m outputs is called a 1-to- m demultiplexer.

3.10.1. 1×2 Demultiplexer

A 1 to 2 demultiplexer has one input and two outputs. Since $2 = 2 \times 1$, it requires only one control (select) line.

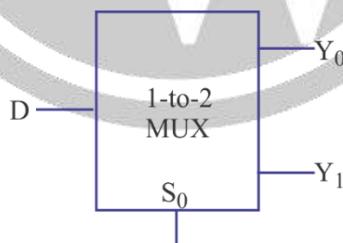


Fig. 3.24. Logic Diagram of 1×2 De-MUX

Truth table of a 1-to-2 demultiplexer

Table: Truth table of a 1-to-2 demultiplexer

Input	Select input		Output	
	S	S ₀	Y ₁	Y ₀
D	0		0	D
D	1		D	0

Thus, the Boolean expressions for the outputs can be written as

$$Y_0 = D\bar{S}_0 \quad \& \quad Y_1 = DS_0$$

Realization of a 1×2 Demultiplexer using Logic Gates:

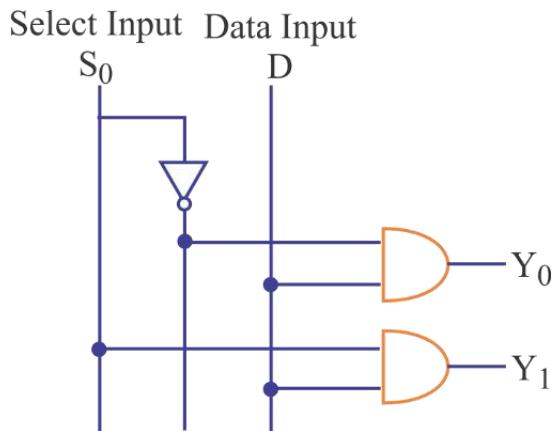


Fig. 3.25. Logic Diagram of 1×2 Demultiplexer

3.10.2. Applications of Demultiplexers

Demultiplexers are used in

1. Data transmission
2. Implementation of Boolean Functions
3. Combinational logic circuit design
4. Generate enable signals (enable one out of many). The application of enable signals in microprocessor systems are:
 - (a) Selecting different banks of memory
 - (b) Selecting different input/output devices for data transfer
 - (c) Enabling different functional units
 - (d) Enabling different rows of memory chips depending on address

3.11. Comparison Between Multiplexer and Demultiplexer

Table : Comparison between Multiplexer and Demultiplexer

S.No.	Parameter of comparison	Multiplexer	Demultiplexer
1.	Type of logic circuit	Combinational	Combinational
2.	Number of data inputs	m	1
3.	Number of select inputs	n	N
4.	Number of data output	1	M
5.	Relation between input/output lines and select lines	$m = 2^n$	$M = 2^N$
6.	Operation principle	Many to 1 or as data selector	1 to many or data distributor

3.12. Decoder

A decoder is a combinational circuit that converts an n -bit binary input data into 2^n output lines, such that each output line will be activated for only one of the possible combinations of inputs. Decoders are usually represented as n -to- 2^n line decoders, where n is the number of input lines and 2^n is the number of maximum possible output lines.

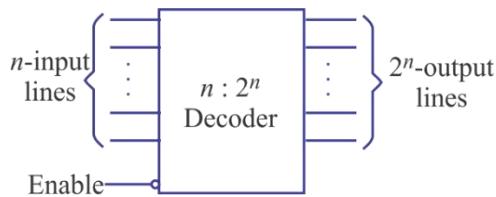


Fig. 3.26. Block Diagram of n-to-2ⁿ Decoder

If there are some unused or ‘don’t care’ combinations in the n-bit code, then there will be less than 2ⁿ output lines. In general, if n and m are respectively the numbers of input and output lines, then m ≤ 2ⁿ.

3.12.1. 2 to 4 Line Decoder

Consider a 2 to 4-line decoder, where A and B are two inputs whereas Y_0 through Y_3 are the four outputs.

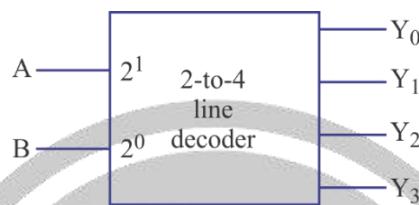


Fig. 3.27. Block Diagram of a 2 to 4 Line Decoder

Truth Table of a 2 to 4 Line Decoder

Table : Truth Table of a 2 to 4 Line Decoder

Inputs		Outputs			
A	B	Y_0	Y_1	Y_2	Y_3
0	0	1	0	0	0
0	1	0	1	0	0
1	0	0	0	1	0
1	1	0	0	0	1

The Boolean expressions for the four outputs is given as:

$$Y_0 = \bar{A} \bar{B} \text{ and } Y_1 = \bar{A} B$$

$$Y_2 = A \bar{B} \text{ and } Y_3 = AB$$

Realization of a 2 to 4 Line Decoder using Logic Gates

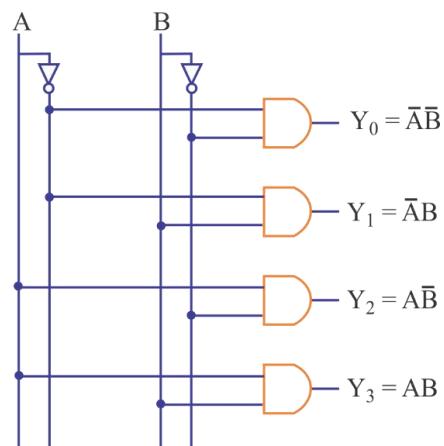


Fig. 3.28. Logic Diagram of a 2 to 4 Line Decoder

3.12.2. Applications of Decoder

Some of important applications of decoder are as follows:

1. When the decoder inputs come from a counter which is being continually pulsed, the decoder outputs will be activated sequentially. Hence, they can be used as timing or sequencing signals to turn devices on or off at specific times.
2. Decoder are use in memory system of a computer where they respond to the address code generated by the microprocessor to activate a particular memory location.
3. They are also used in computers for selection of external devices that include printers, modems, scanners, internal disk drives, keyboard, video monitor etc.

3.13. Encoders

An encoder is a combinational logic circuit that performs the inverse operation of a decoder. An encoder has 2^n (or fewer) input lines and n output lines.

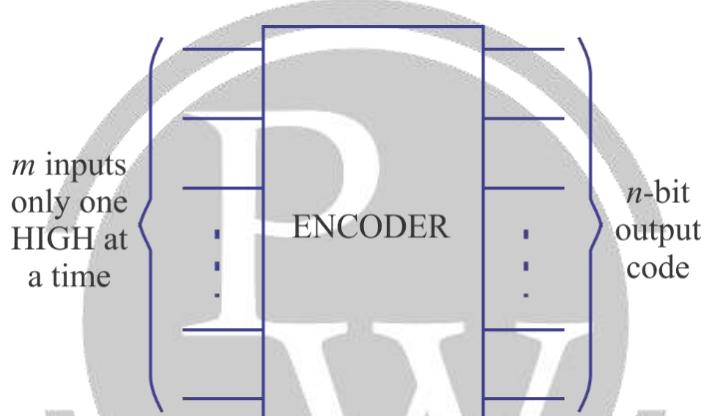


Fig. 3.29. Block Diagram of Encoder

3.13.1. Octal to Binary Encoder

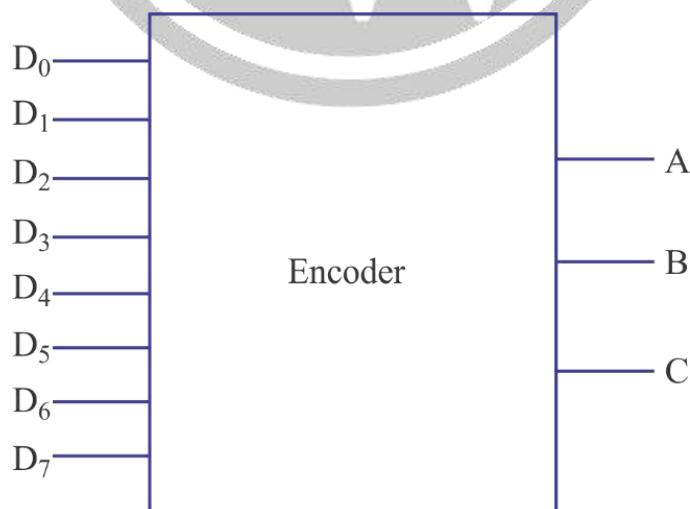


Fig. 3.30. Octal to Binary Encoder

Truth Table of an Octal to Binary Encoder:**Table: Truth Table of an Octal to Binary Encoder**

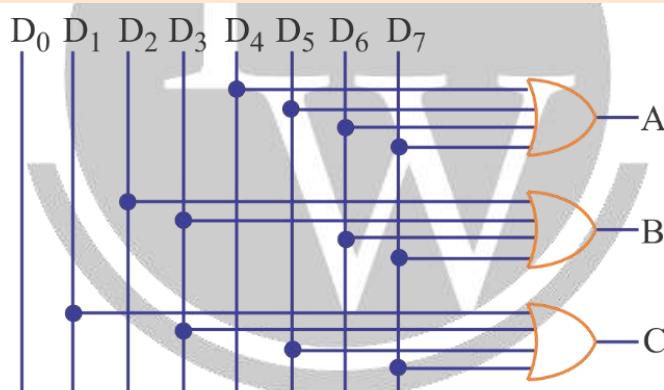
Inputs								Outputs		
D ₀	D ₁	D ₂	D ₃	D ₄	D ₅	D ₆	D ₇	A	B	C
1	0	0	0	0	0	0	0	0	0	0
0	1	0	0	0	0	0	0	0	0	1
0	0	1	0	0	0	0	0	0	1	0
0	0	0	1	0	0	0	0	0	1	1
0	0	0	0	1	0	0	0	1	0	0
0	0	0	0	0	1	0	0	1	0	1
0	0	0	0	0	0	1	0	1	1	0
0	0	0	0	0	0	0	1	1	1	1

The logical expressions for the outputs as follows:

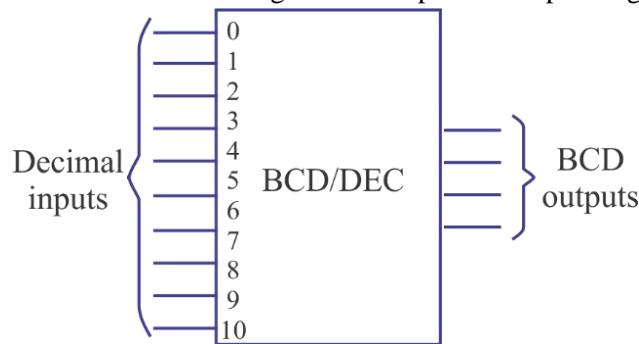
$$A = D_4 + D_5 + D_6 + D_7$$

$$B = D_2 + D_3 + D_6 + D_7$$

$$C = D_1 + D_3 + D_5 + D_7$$

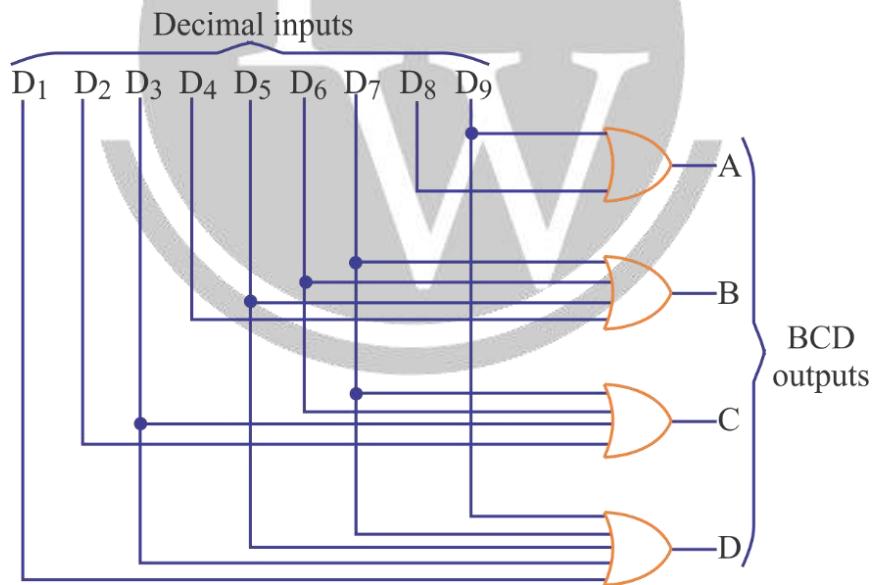
3.13.2. Octal to Binary Encoder**Fig. 3.31. Logic Diagram of Octal-to-Binary Encoder****3.13.3. Decimal to BCD Encoder**

This type of encoder has 10 inputs one for each decimal digit and 4 outputs corresponding to the BCD code.

**Fig. 3.32. Block Diagram of a Decimal-to-BCD Encoder**

Truth Table of a Decimal to Binary Encoder:
Table : Truth Table of a Decimal to Binary Encoder

Input										Output			
0	1	2	3	4	5	6	7	8	9	A	B	C	D
D ₀	D ₁	D ₂	D ₃	D ₄	D ₅	D ₆	D ₇	D ₈	D ₉	A	B	C	D
1	0	0	0	0	0	0	0	0	0	0	0	0	0
0	1	0	0	0	0	0	0	0	0	0	0	0	1
0	0	1	0	0	0	0	0	0	0	0	0	1	0
0	0	0	1	0	0	0	0	0	0	0	0	1	1
0	0	0	0	1	0	0	0	0	0	0	0	1	0
0	0	0	0	0	1	0	0	0	0	0	0	1	0
0	0	0	0	0	0	1	0	0	0	0	0	1	0
0	0	0	0	0	0	0	1	0	0	0	0	1	1
0	0	0	0	0	0	0	0	1	0	0	0	1	0
0	0	0	0	0	0	0	0	0	1	0	0	0	0
0	0	0	0	0	0	0	0	0	0	1	0	0	1


Fig. 3.33. Logic Diagram of Decimal-to-BCD encoder

The outputs of a decimal-to-BCD encoder:

$$A = D_8 + D_9$$

$$B = D_4 + D_5 + D_6 + D_7$$

$$C = D_2 + D_3 + D_6 + D_7$$

$$D = D_1 + D_3 + D_5 + D_7 + D_9$$

3.14. Priority Encoder

3.14.1. Truth Table of a Four Input Priority Encoder: (Taking LSB as priority)

Table: Truth Table of a Four Input Priority Encoder

Inputs				Outputs	
D ₀	D ₁	D ₂	D ₃	A	B
0	0	0	0	X	X
1	0	0	0	0	0
X	1	0	0	0	1
X	X	1	0	1	0
X	X	X	1	1	1

According to the truth table, the higher the subscript number, the higher the priority of the input.

The X's are don't care conditions indicating that the binary values they represent may be equal to 0 or 1.

3.15. Code Converters

A code converter is a combinational logic circuit which accepts the input information in one binary code, converts it and produces an output into another binary code.

3.15.1. The truth table for 4-bit Binary and its Equivalent BCD

Table: Truth table for 4-bit Binary and its Equivalent BCD

Decimal	Binary Input				BCD Output				
	A	B	C	D	B ₄	B ₃	B ₂	B ₁	B ₀
0	0	0	0	0	0	0	0	0	0
1	0	0	0	1	0	0	0	0	1
2	0	0	1	0	0	0	0	1	0
3	0	0	1	1	0	0	0	1	1
4	0	1	0	0	0	0	1	0	0
5	0	1	0	1	0	0	1	0	1
6	0	1	1	0	0	0	1	1	0
7	0	1	1	1	0	0	1	1	1
8	1	0	0	0	0	1	0	0	0
9	1	0	0	1	0	1	0	0	1
10	1	0	1	0	1	0	0	0	0
11	1	0	1	1	1	0	0	0	1
12	1	1	0	0	1	0	0	1	0
13	1	1	0	1	1	0	0	1	1
14	1	1	1	0	1	0	1	0	0
15	1	1	1	1	1	0	1	0	1

The minimized expression of outputs are as follows:

$$B_4 = AB + AC$$

$$B_1 = \bar{A}C + ABC$$

$$B_2 = \bar{A}\bar{B} + BC$$

$$B_3 = A\bar{B}\bar{C}$$

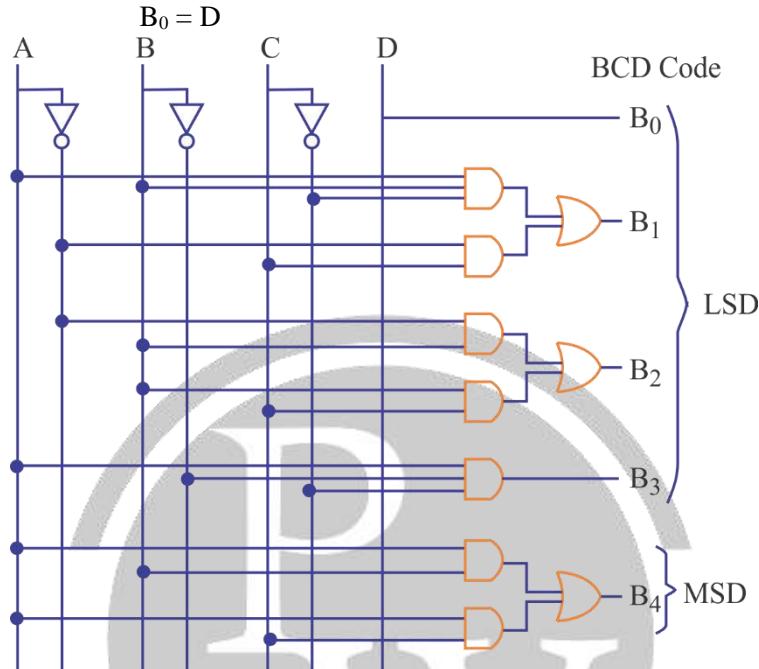


Fig. 3.34. Logic Diagram of a Binary-to-BCD Code Converter

3.16. Parity Generator

Parity generators are circuits that accept an (n-1) bit data stream and generate an extra bit that is transmitted with the bit stream. This extra bit is referred to as the parity bit. The parity added in binary message is such that the total number of 1's in the message can be either odd or even according to the type of parity used.

3.16.1. Even Parity Generator

The even parity generator is a combinational logic circuit that generates the parity bit such that the number of 1's in the message becomes even. The parity bit is '1' if there are odd number of 1's in the data stream and the parity bit is '0' if there are even number of 1's in the data stream.

Truth table for 4-bit data with Even Parity:

Table: Truth table for 4-bit data with Even Parity

4-bit data				Even Parity
A	B	C	D	P
0	0	0	0	0
0	0	0	1	1
0	0	1	0	1
0	0	1	1	0

0	1	0	0	1
0	1	0	1	0
0	1	1	0	0
0	1	1	1	1
1	0	0	0	1
1	0	0	1	0
1	0	1	0	0
1	0	1	1	1
1	1	0	0	0
1	1	0	1	1
1	1	1	0	1
1	1	1	1	0

The minimized expression for even parity generator is

$$P = A \oplus B \oplus C \oplus D$$

The logic diagram for the even parity generator is given as

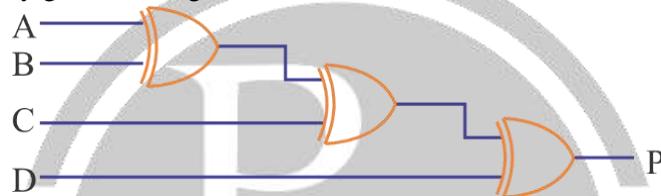


Fig. 3.35. Logic diagram of even parity generator

3.16.2. Odd Parity Generator

The odd parity generator is a combinational logic that generates the parity bit such that the number of 1's in the message becomes odd. The parity bit is '0' for odd number of 1's and '1' for even number of 1's in the bit stream.

Example: A parity generation circuit required to generate an odd parity bit may use _____?

Solution: Odd parity generation circuit consists of combination of EX-OR and EX-NOR gates, whereas even priority generator consists only EX-OR gates.

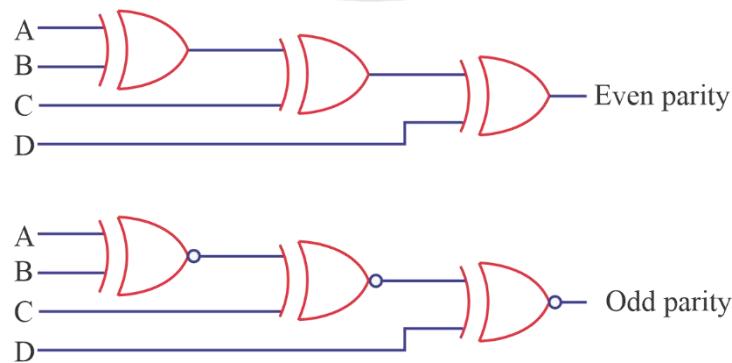


Fig. 3.36

∴

It is combination of EX-OR and EX-NOR gates.



4

SEQUENTIAL LOGIC CIRCUITS

4.1. Sequential Logic Circuits

In sequential logic circuits, the output is a function of the present inputs as well as the inputs and outputs. Sequential circuit include memory elements to store past data. The flip-flop is a basic element of sequential logic circuits.

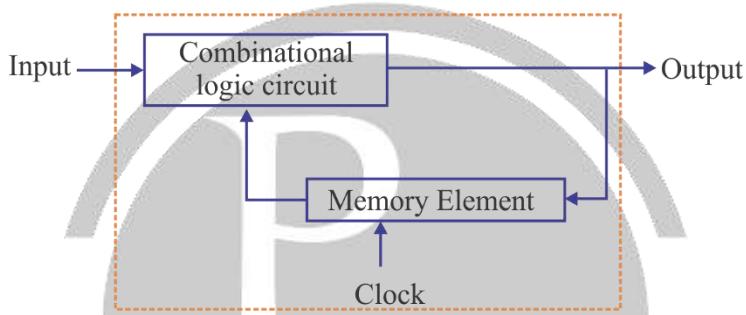


Fig. 4.1. General Block diagram of Sequential Logic Circuit

There are two types of sequential circuits:

4.1.1. Synchronous Circuits

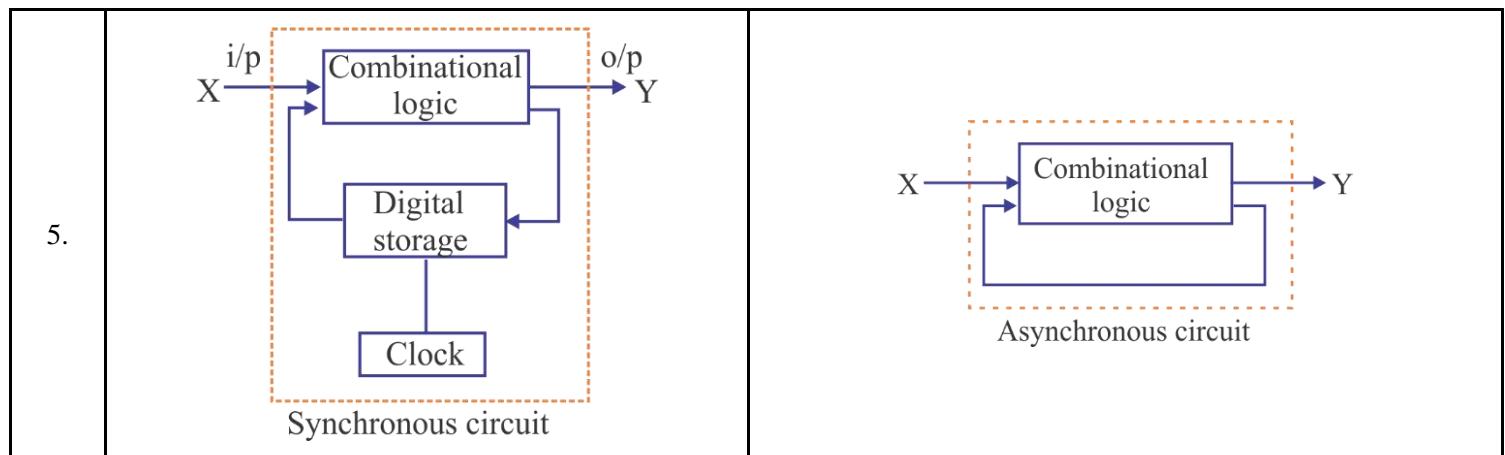
The sequential circuits which are controlled by a clock are called synchronous sequential circuits. These circuits get activated only when clock signal is present.

4.1.2. Asynchronous Circuits

The sequential circuits which are not controlled by a clock are called asynchronous sequential circuits, i.e. the sequential circuits in which events can take place any time the inputs are applied are called asynchronous sequential circuits.

4.2. Difference Between Synchronous and Asynchronous Sequential Circuits

S.No.	Synchronous Sequential Circuits	Asynchronous Sequential Circuits
1.	In synchronous circuits, the change in input signals can affect memory elements upon activation of clock signal.	In asynchronous circuits, change in input signals can affect memory elements at any instant of time.
2.	In synchronous circuits, memory elements are clocked FF's.	In asynchronous circuits, memory elements are either unlocked FF's or time delay elements.
3.	The maximum operating speed of the clock depends on time delays involved.	Since the clock is not present, asynchronous circuits can operate faster than synchronous circuits.
4.	They are easier to design.	More difficult to design.



4.3. Latches

Flip-flop is an electronic circuit or device which is used to store a data in binary form. Actually, flip-flop is a one-bit memory device and it can store either 1 or 0. Flip-flops is a sequential device that changes its output only when a clocking signal is changing. On the other hand, latch is a sequential device that checks all its inputs continuously and changes its outputs accordingly at any time independent of a clock signal. It refers to non-clocked flip-flops, because these flip-flops, because these flip-flops ‘latch on’ to a 1 or a 0 immediately upon receiving the input pulse.

4.3.1. General Block Diagram of a Latch or Flip-flop

Figure shown below is the general type of symbol used for a latch. In case of a flip-flop, a clock signal must be shown at input side. It has many inputs and two outputs, labelled Q and \bar{Q} . The Q output is the normal output of the latch and \bar{Q} is the inverted output.

Note: A flip-flop is said to be in HIGH state or logic 1 state or SET state when $Q = 1$, and in LOW state or logic 0 state or RESET state or CLEAR state when $Q = 0$.

4.3.2. Difference between Latches and Flip-flops

S.No.	Latch	Flip-flop
1.	A latch is an electronic sequential logic circuit used to store information in an asynchronous arrangement.	A flip-flop is an electronic sequential logic circuit used to store information in a synchronous arrangement. It has two stable states and maintains its states for an indefinite period until a clock pulse is applied.
2.	One latch can store one-bit information, but output state changes only in response to data input.	One flip-flop can store one-bit data, but output state changes with clock pulse only.
3.	Latch is an asynchronous device and it has no clock input.	Flip-flop has clock input and its output is synchronised with clock pulse.
4.	Latch holds a bit value and it remains constant until new inputs force it to change.	Flip-flop holds a bit value and it remains constant until a clock pulse is received.
5.	Latches are level-sensitive, and the output tracks the input when the level is high. Therefore, as long as the level is logic level 1, the output can change if the input changes.	Flip-flops are edge sensitive. They can store the input only when there is either a rising or falling edge of the clock.

4.4. Latch

A latch is a type of bistable logic device or multivibrator that is most often used in applications that require data storage. The main characteristics of latch is that the output is not dependent solely on the present state of the input but also on the proceeding output state.

Latches are sometimes used for multiplexing data onto a bus. For example, data being input to a computer from a external source have to share the data bus with data from other sources. When the data bus becomes unavailable to external source, the existing data must be temporarily stored, and hence the latches are placed between the external source and data bus.

4.4.1. SR Latch

For the SR latch (S stands for set and R for reset). The logic circuit for SR latch is shown in figure below:

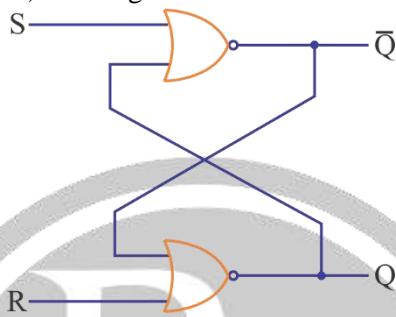


Fig. 4.1. Logic circuit of SR latch.

The state table for the SR latch is:

S	R	Q	Q^+	\bar{Q}^+
0	0	0	0	1
0	0	1	1	0
0	1	0	0	1
0	1	1	0	1
1	0	0	1	0
1	0	1	1	0
1	1	0	0	1
1	1	1	0	0

The symbol for SR Latch is:

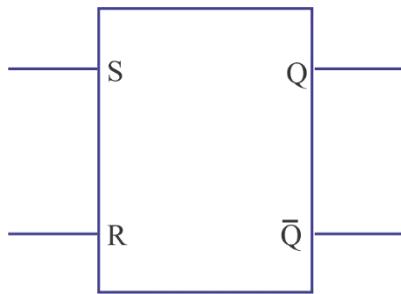


Fig. 4.2.

Obtaining the characteristic equations of the NOR gate based latch are; we get

$$Q^+ = \bar{R} \times S + \bar{R} \times Q = \bar{R} \times (S + Q) \text{ and } \bar{Q}^+ = \bar{S} \times R + \bar{S} \times \bar{Q} = \bar{S} \times (R + \bar{Q})$$

Note: It must be noted that the complementing Q^+ does not yield \bar{Q}^+ .

Hence, the truth table for SR latch is

S	Q	Q^+	\bar{Q}^+	
0	0	Q	\bar{Q}	⇒ No change
1	1	0	1	⇒ Reset Q^+ to 0
1	0	1	0	⇒ Set Q^+ to 1
1	1	0	0	⇒ Forbidden state

However, the forbidden state ($S = R = 1$) is considered a don't care state.

Consider a Timing diagram for SR latch

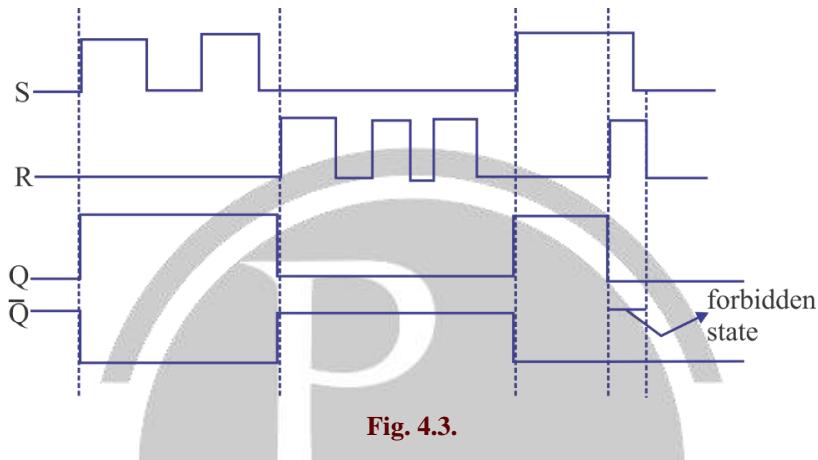


Fig. 4.3.

4.4.2. $\bar{S}\bar{R}$ Latch:

An $\bar{S}\bar{R}$ latch can be implemented using NAND gates, as shown in figure below

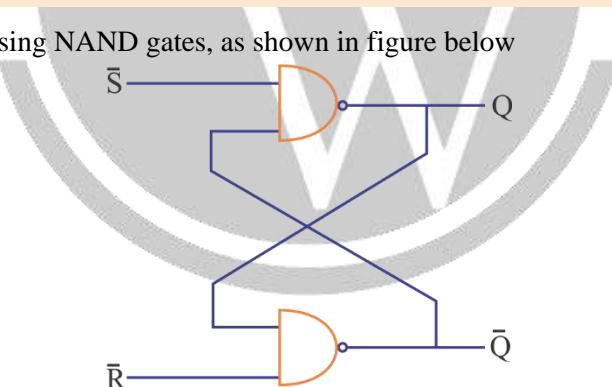


Fig. 4.4. Logic circuit for $\bar{S}\bar{R}$ Latch

The $\bar{S}\bar{R}$ latch is said to be set-dominant 1,

The symbol for $\bar{S}\bar{R}$ latch is shown below:

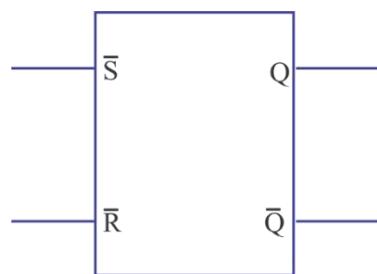


Fig. 4.5.

The truth table for $\bar{S}\bar{R}$ latch is given as:

\bar{S}	\bar{R}	Q^+	\bar{Q}^+	
1	1	Q	\bar{Q}	⇒ No change
1	0	0	1	⇒ Reset Q^+ to 0
0	1	1	0	⇒ Set Q^+ to 1
1	1	1	1	⇒ Forbidden state

Application of $\bar{S}\bar{R}$ latch: The application of $\bar{S}\bar{R}$ latch is in switch bouncing i.e. contact bounces of a push-button switch during its opening or closing can be eliminated by using $\bar{S}\bar{R}$ latch.

4.4.3. Gated SR Latch on Enable SR Latch or clocked SR Latch

A gated or level-sensitive SR latch uses a control signal C that can be used as a clock signal or can be used as enable input. The logic circuit diagram, symbol and truth is given as

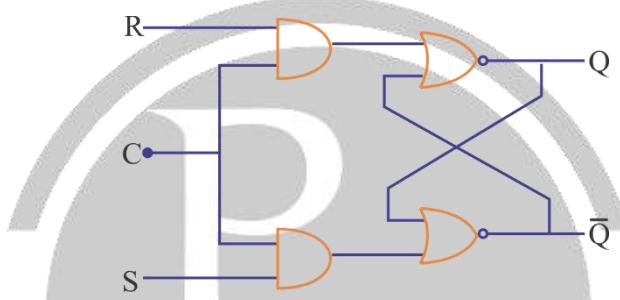


Fig. 4.6. Logic circuit of clocked SR latch.

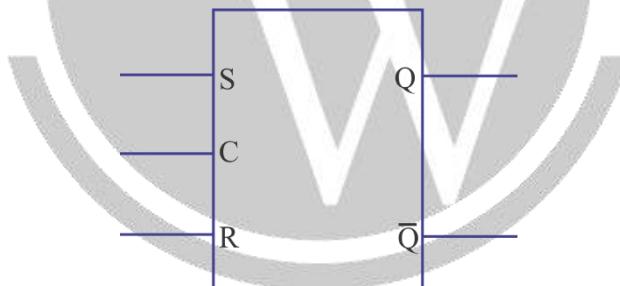


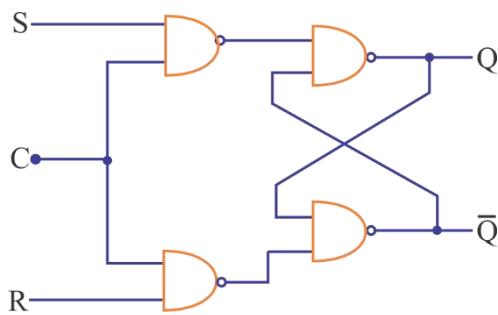
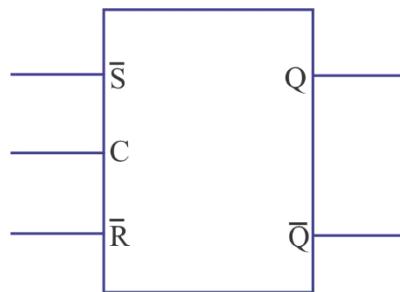
Fig. 4.7. Symbol for clocked SR latch

C	S	R	Q^+	\bar{Q}^+	
0	✗	✗	Q	\bar{Q}	{ No change state
1	0	0	Q	\bar{Q}	⇒ Reset
1	0	1	0	1	⇒ Set
1	1	0	1	0	⇒ Forbidden state
1	1	1	0	0	

4.4.4. Gated $\bar{S}\bar{R}$ Latch or enable $\bar{S}\bar{R}$ Latch or clocked $\bar{S}\bar{R}$ Latch

Gated $\bar{S}\bar{R}$ latch is implemented using two NAND gates and an $\bar{S}\bar{R}$ latch.

The logic circuit diagram, symbol and truth is given as


Fig. 4.8. Logic diagram of clocked $\bar{S}\bar{R}$ latch.

Fig. 4.9.

The truth table of the gated SR latch based on a $\bar{S}\bar{R}$ latch:

C	J	K	Q	\bar{Q}^+
O	X	X	Q	\bar{Q}
1	0	0	Q	\bar{Q}
1	0	1	0	1
1	1	0	1	0
1	1	1	1	1

The characteristic equation for SR flip-flop is given as

$$Q^+ = Q_{n+1} = S + \bar{R}Q_n = S + \bar{R}Q$$

4.5. Flip-Flops

Flip-flops are synchronous bistable devices also known as bistable multivibrator. Its output change its state only at a verified point (i.e. leading or trailing edge) on the triggering input called the clock (CLK), i.e. changes in the output occur in synchronization with the clock.

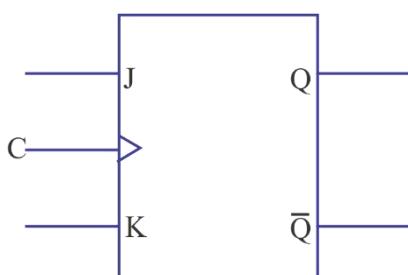
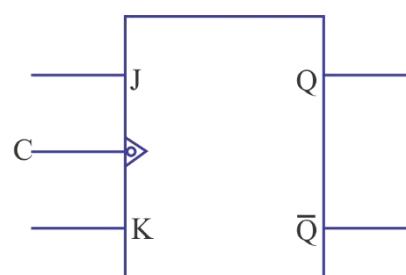
Flip-flops are edge-triggered or edge-sensitive whereas gated latches are level-sensitive.

4.5.1. Edge-triggered flip-flop

An edge-triggered flip-flop changes its state either at positive edge (rising edge) or at negative edge (falling edge) of the clock pulse.

There are two type of edge-triggered flip-flops. The key to identify an edge-triggered flip-flop is by its logic symbol by small triangle inside the block at the clock input C. This triangle is called the dynamic input indicator.

Positive edge triggered has no bubble at input C whereas negative edge triggered has bubble at input C.


Fig. 4.10. Positive edge triggered flip-flop.

Fig. 4.11. Negative edge-triggered flip-flop.

4.5.2. Basic JK flip-flop

JK Flip-flop (J as a set input and K as a reset input) is the most versatile of the basic flip-flops.

The logic circuit of the gated JK flip-flop is shown in figure below:

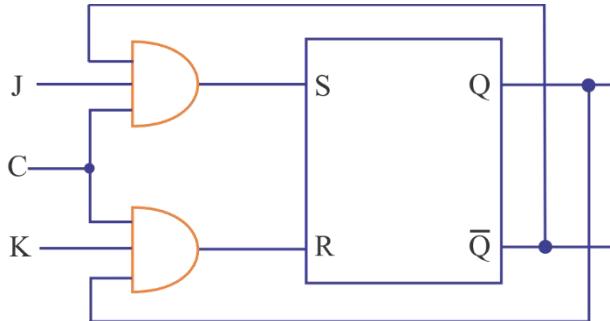


Fig. 4.12. Logic circuit diagram of clocked JK flip-flop.

The state table for the JK flip-flop is given as

C	J	K	Q	Q ⁺
O	X	X	X	Q
1	0	0	0	0
1	0	0	1	1
1	0	1	0	0
1	0	1	1	0
1	1	0	0	1
1	1	0	1	1
1	1	1	0	1

Hence, the truth table becomes,

C	S	R	Q ⁺	\bar{Q}^+
0	X	X	Q	\bar{Q}
1	0	0	Q	\bar{Q}
1	0	1	0	1
1	1	0	1	0
1	1	1	0	0

} No change state

=> Reset

=> Set

=> Forbidden state

Note: The forbidden state, inherent to SR flip-flop is eliminated by adding two feedback loops such that the output becomes 1 only if Q = 0 and reset to only if Q = 1.

It should also be noted that when the inputs (J & K) are set to 1 and clock signal change to 1, then the feedback value of Q & \bar{Q} forced the flip-flop to toggle its value.

(i.e. to switch its state to its logical complement) hence, to ensure this operation in smooth fashion, the pulse width of the clock must be smaller than the propagation delay of the flip-flop.

The characteristic equation of the JK flip-flop

$$Q_{n+1} = J\bar{Q}_n + \bar{K}Q_n \text{ or } Q^+ = J\bar{Q} + \bar{K}Q$$

4.5.3. T-flip-flop

A JK flip-flop can be transformed into a T- flip-flop (T stands for Toggle). When T flip-flop is activated, its output changes its state at every time a pulse is applied to the input T.

The logic circuit of the gated T flip-flop is.

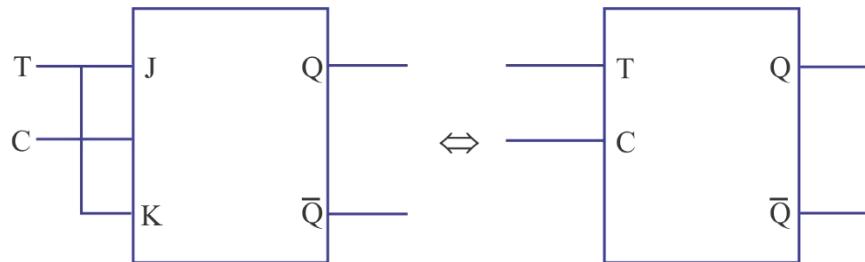


Fig. 4.13.

The state or characteristic table for T flip-flop is

C	T	Q	Q^+
0	X	X	Q
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	0

As $J = K = T$, we obtain the characteristic equation as

$$Q^+ = T \times \bar{Q} \times C + (\bar{T} + \bar{C}) \times Q$$

If $C = 1$, the characteristic the equation is reduced to

$$Q^+ = T \oplus Q$$

$$\text{If } C = 0, Q^+ = Q$$

Hence, the truth table of the T-flip flop is given as

C	T	Q^+	\bar{Q}^+
0	X	Q	\bar{Q}
1	0	Q	\bar{Q}
1	1	\bar{Q}	Q

} \Rightarrow No change state
 } \Rightarrow Toggle

Fig. 4.14.

4.5.4. D Flip-Flop

D-flip-flop can be obtained by use of only two combinations of S-R or J-K flip-flop. It has only one input i.e. D-input or data input.

The logic symbol for D- flip-flop is given as

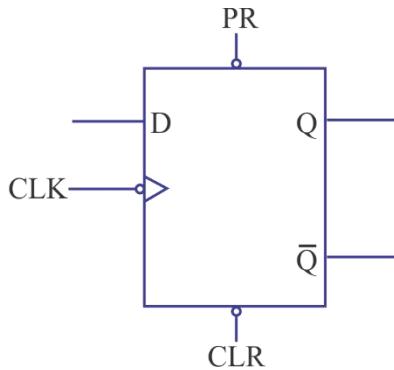


Fig. 4.15.

The truth table for D-flip-flop is

Input	Output
D	Q_{n+1}
0	0
1	1

The characteristic equation of D-flip-flop is:

$$Q_{n+1} = D$$

4.5.5. Excitation table of Flip-flops

The truth table of a flip-flop is sometimes referred as characteristic table as it specifies the operational characteristics of the flip-flop there may occurs some situations in which the present state and the next state of the circuit is desired and known. Then the designing of input conditions to as to fulfil the requirements of the circuit, there is a table called excitation table.

It is very important and useful design aid for sequential circuit.

The excitation table for flip-flops:

Present state	Next state	SR Flip-flop		JK Flip-flop		T Flip-flop	D Flip-flop
		S	R	J	K	T	D
0	0	0	×	0	×	0	0
0	1	1	0	1	×	1	1
1	0	0	1	×	1	1	0
1	1	×	0	×	0	0	1

4.6. Operating Characteristics of Flip-Flops

4.6.1. Propagation Delay Time:

Propagation delay time is the time interval required after an input signal has been applied for the resulting output change to occur.

There are four categories of propagation delay times which are as follows:

A. Propagation delay t_{PLH} , it is measured from the triggering edge of the clock pulse to Low-to-High transition of the output.

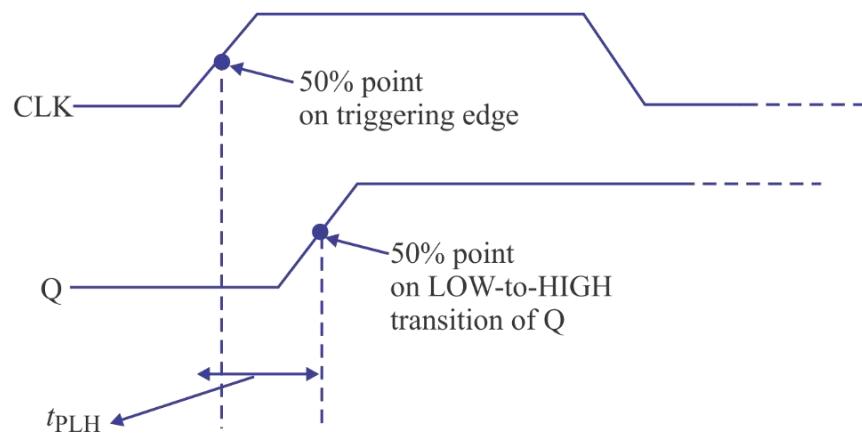


Fig. 4.16.

B. Propagation delay t_{FLH} , it is measured from the triggering edge of the clock pulse to HIGH-to-LOW transition of the output.

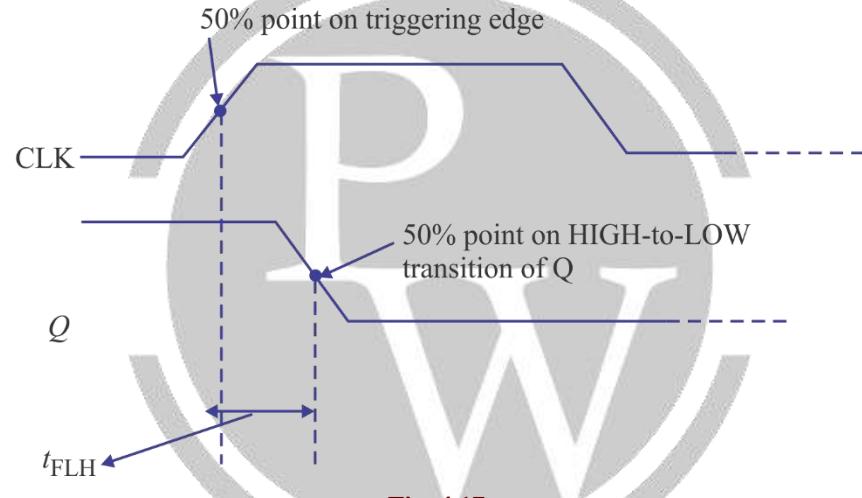


Fig. 4.17.

C. Propagation delay t_{PHL} , it is measured from the leading edge of the PRESET input to LOW-to-HIGH transition of the output.

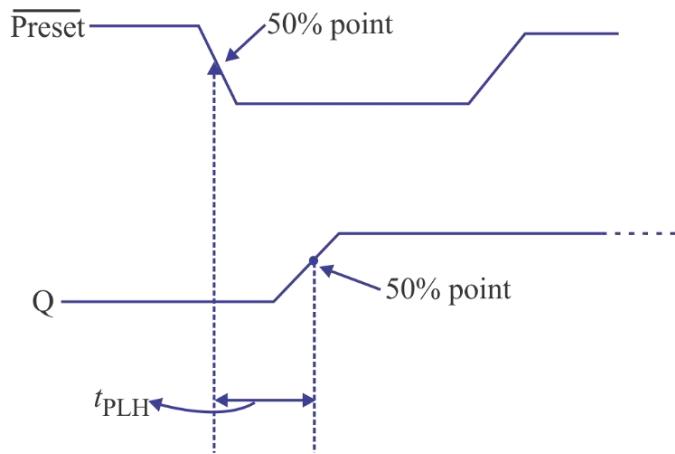


Fig. 4.18.

D. Propagation delay t_{PHL} , it is measured from the leading edge of the clear input to the HIGH-to-LOW transition of the output.

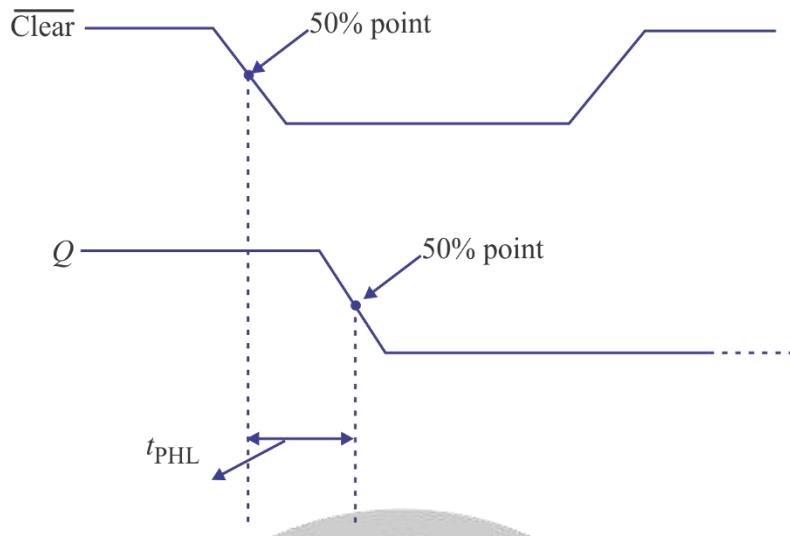


Fig. 4.19.

4.6.2. Set-up time (t_s)

It is the minimum time interval required for the logic levels (0 or 1) to be maintained constantly on the inputs (J, K or D) prior to the triggering edge of the clock pulse in order for the levels to be reliably clocked into the flip-flop.

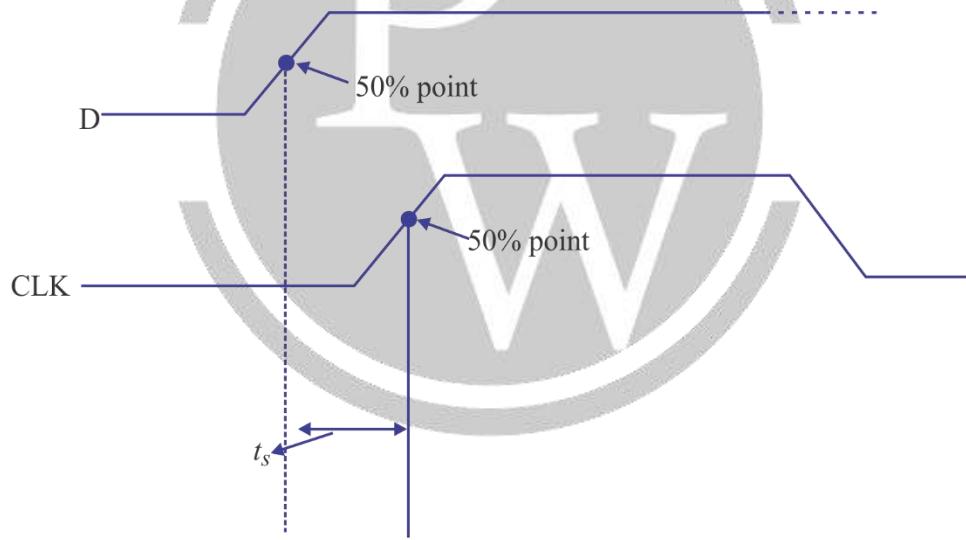


Fig. 4.20.

4.6.3. Hold time (t_h)

It is the time for which the data must remain stable after the triggering edge of the clock.

4.6.4. Clock-pulse width

The minimum time duration for which the clock pulse must remain HIGH and LOW which are designed by manufacturers. Failure to clock pulse width results in unreliable triggering.

4.6.5. Maximum clock frequency

The maximum clock frequency (f_{max}) is the highest rate at which flip-flop can be reliably operated.

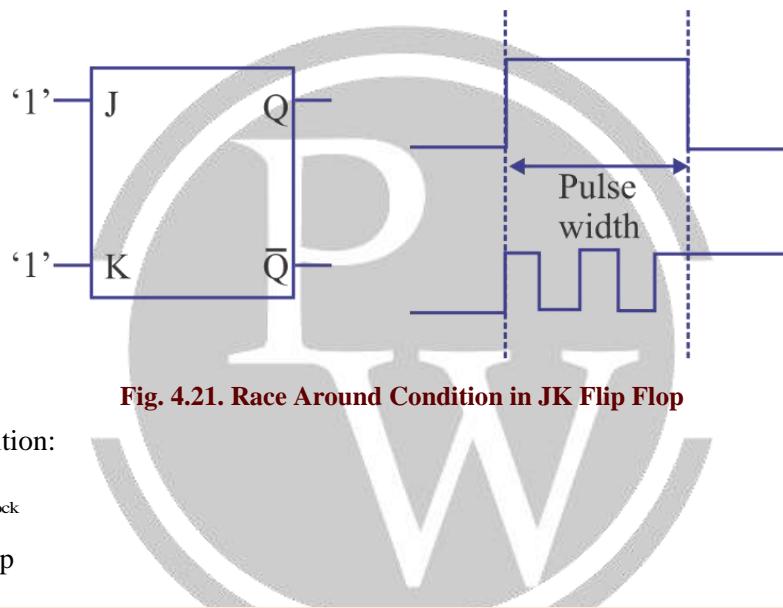
4.7. Applications of Flip-Flops

Some of the common applications of flip-flops are as follows:

1. Switch bouncing.
2. Registers.
3. Counters.
4. Memory elements.

4.8. Race Around Condition

JK flip flop suffers from the problem of race around condition. When $J = 1$ & $K = 1$, is applied to the JK flip flop and JK flip flop is level triggered then output of the JK flip flop toggles so many times during the pulse width of the clock and output of the flip flop settled either at 1 or 0 depending upon the pulse width of the clock and propagation delay of the flip flop is called race around condition.



To avoid Race Around condition:

- $T_{\text{pulse-width}} < T_{\text{pd}} < T_{\text{clock}}$
- Master Slave flip flop

4.8.1. Master Slave Flip flop:

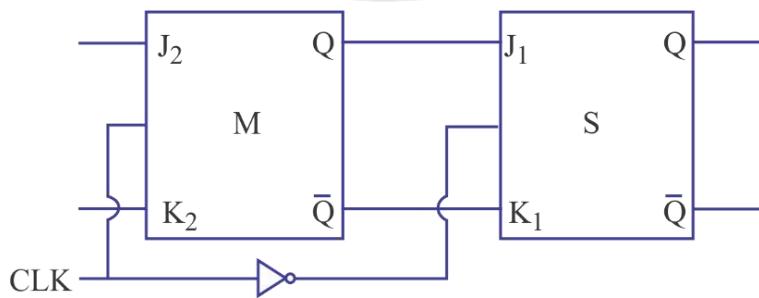


Fig. 4.22. Logic Diagram for Master Slave JK Flip Flop

- (a) In master slave flip flop, inverted clock is given to the slave.
- (b) Master slave flip flop is used to store single bit because output is taken only from slave flip flop.
- (c) Here, master flip flop is level triggered while slave is negative edged triggered.

Note: JK flip flop is also known as Universal flip flop.

4.9. Designing of One Flip Flop by Other Flip Flop

The steps for designing of one flip flop or new flip flop using existing or same existing flip flop.

Step 1: Write the characteristic table for the designed flip flop.

Step 2: Write the excitation table for the available flip flop.

Step 3: Write the logical expression.

Step 4: Minimize the logical expression.

Step 5: Circuit Implementation.

4.10. Shift Registers

An array of flip-flops is required to store binary information, and the number of flip-flops required is equal to the number of bits used to store is referred as registers.

Examples of registers are general purpose registers flags, etc.

Now, the information or data can be stored or entered in serial form (one-bit at a time) or in parallel form (all the bits simultaneously) and can be retrieved like this manner too. The data will be entered or retrieved in serial form is known as temporal code and which is in parallel form is called special code.

Hence, registers can be classified into four categories depending upon the data being entered or retrieved.

4.10.1. SISO (serial-in, serial-out) Shift Register:

In serial-in, serial-out shift register, data input is in serial form and common clock pulse is applied to each of the flip-flop. After each clock pulse, data moves by one position. The output can be obtained in serial form, as shown in figure below:

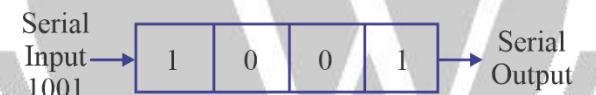


Fig. 4.23. SISO Shift Register

- It is the slowest shift register among all the shift registers.
- To store n-bits in a n-bit SISO register, then the minimum “n” clock pulses are required.
- To retrieve n-bits from a n-bit SISO register, then the minimum “(n-1)” clock pulses are required.

4.10.2. SIPO (serial-in, parallel-out) Shift Register

In serial-in, parallel-out shift register, data is applied at the input of register in serial form and the output can be obtained in parallel form after the completely shifting of data in register. Figure below shows the serial input data, and then parallel output.

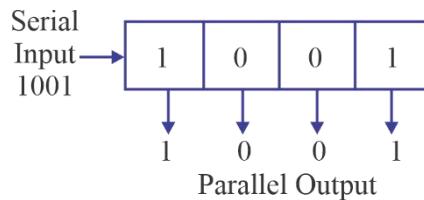


Fig. 4.24. SIPO Shift Register

- To store n-bits in a n-bit SIPO register, the minimum “n” clock pulses are required.
- To retrieve n-bits from a n-bit SIPO register, there is no pulse required.

4.10.3. PISO (parallel-in, serial out) Shift Register

In parallel-in, serial-out shift register, data is loaded into shift register in parallel form and the data output obtained will be serial form as shown in figure below:

- To store n-bit in a n-bit PISO register, a single clock pulse is required.
- To retrieve n-bit from n-bit PISO register, the minimum “(n-1)” clock pulses are required.

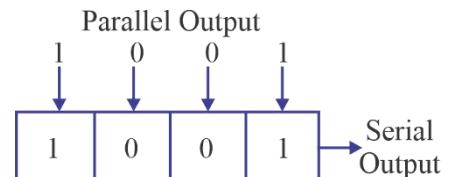


Fig. 4.25. PISO shift register

4.10.4. PIPO (parallel in, parallel out) Shift Register

In parallel-in, parallel-out shift register, data is loaded in parallel form and the data output obtained will be in parallel, as shown in figure below:

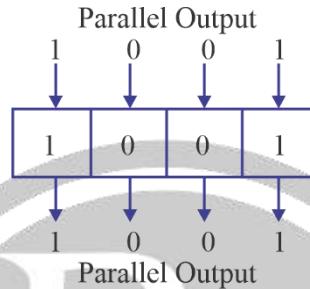


Fig. 4.26. PIPO Shift Register

- To store n-bit in n-bit PIPO register, only a single clock pulse is required.
- To retrieve n-bits from n-bit PIPO register, no clock pulse is required.

Serial Input: The data in the serial form is applied at the serial input after clearing the flip-flops using CLR.

The waveform of serial input shift register is shown below:

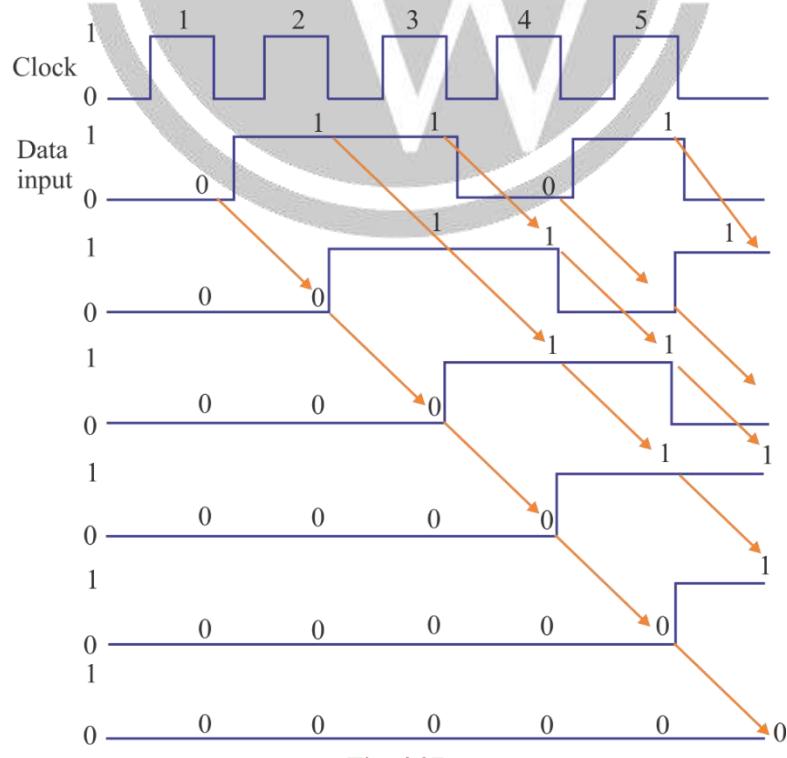


Fig. 4.27.

Parallel Input: Data can be entered in the parallel form making use of the pre-set inputs. Then after clearing the flip-flops, if the data lines are connected to the parallel lines and '1' is applied to the PRESET input.

4.10.5. Universal Shift Register

If the flip-flop outputs of a shift register are accessible, then information entered serially by shifting can be taken out in parallel from the outputs of the flip-flops. If a parallel

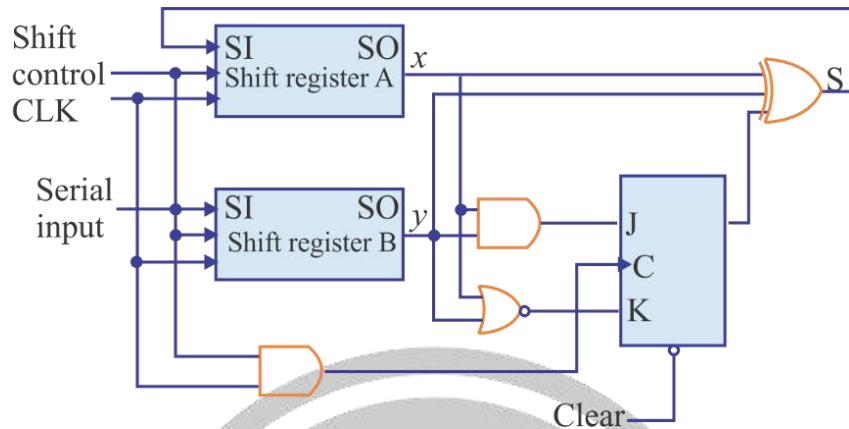


Fig. 4.28. Second form of serial adder

load capability is added to a shift register, then data entered in parallel can be taken out in serial fashion by shifting the data stored in the register. Some shift registers provide the necessary input and output terminals for parallel transfer. They may also have both shift-right and shift-left capabilities.

The most general shift register has the following capabilities:

1. A clear control to clear the register to 0.
2. A clock input to synchronize the operations.
3. A shift-right control to enable the shift-right operation and the serial input and output lines associated with the shift right.
4. A shift-left control to enable the shift-left operation and the serial input and output lines associated with the shift left.
5. A parallel-load control to enable a parallel transfer and the n input lines associated with the parallel transfer.
6. "n" parallel output lines.
7. A control state that leaves the information in the register unchanged in response to the clock.

Other shift registers may have only some of the preceding functions, with at least one. shift operation. A register capable of shifting in one direction only is a unidirectional shift register. One that can shift in both directions is a bidirectional shift register. If the register can shift in both directions and has parallel-load capabilities, it is referred to as a universal shift register. The block diagram symbol and the circuit diagram of a four-bit universal shift register is shown in figure below:

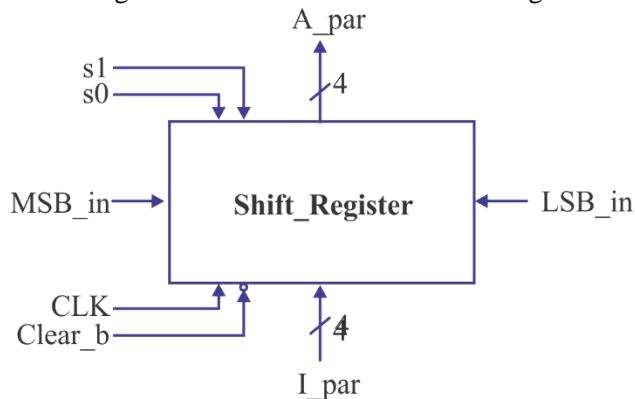


Fig. 4.29. 4-bit Universal Shift Register

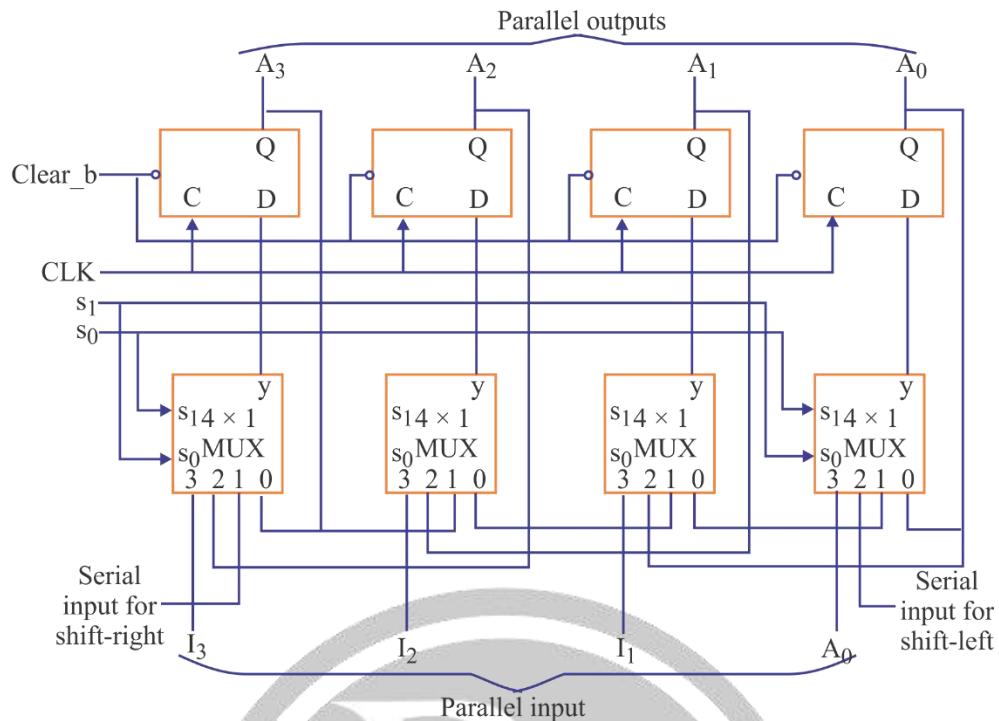


Fig. 4.30. Logic Diagram of 4-bit Universal shift register

The function for the Universal Shift Register is as follows:

Mode Control		Register Operation
S ₁	S ₀	
0	0	No change
0	1	Shift right
1	0	Shift left
1	1	Parallel load

Shift registers are often used to interface digital systems situated remotely from each other. For example, suppose it is necessary to transmit an n-bit quantity between two points. If the distance is far, it will be expensive to use n lines to transmit its bits in parallel. It is more economical to use a single line and transmit the information serially, one bit at a time. The transmitter accepts the n-bit data in parallel into a shift register and then transmits the data serially along the common line. The receiver accepts the data serially into a shift register. When all n bits are received, they can be taken from the outputs of the register in parallel. Thus, the transmitter performs a parallel-to-serial conversion of data and the receiver does a serial-to-parallel conversion.

4.10.6. Applications of Shift Registers

- (a) **Delay line:** A shift register can be used to introduce a delay (Δt) in signals

$$\Delta t = N \times \frac{1}{f_c}$$

Where N is number of stages & f_c is the clock frequency.

- (b) Serial-to-parallel converter
(c) Parallel-to-serial converter

- (d) Ring counter
- (e) Twisted ring counter
- (f) Sequence counter

4.11. Asynchronous Counter Or Ripple Counter

A circuit which is used for counting the numbers or pulses is known as counter. Counter is referred to as modulo-N (or divide by N), where the word modulo indicates the number of states in the counter.

4.11.1. 3- Bit Binary Counter

Consider a 3-bit binary counter which has total '8' number of states which require three flip-flops and Q_2 , Q_1 and Q_0 are the outputs of those flip-flops.

The circuit diagram or logic circuit diagram for 3-bit binary counter,

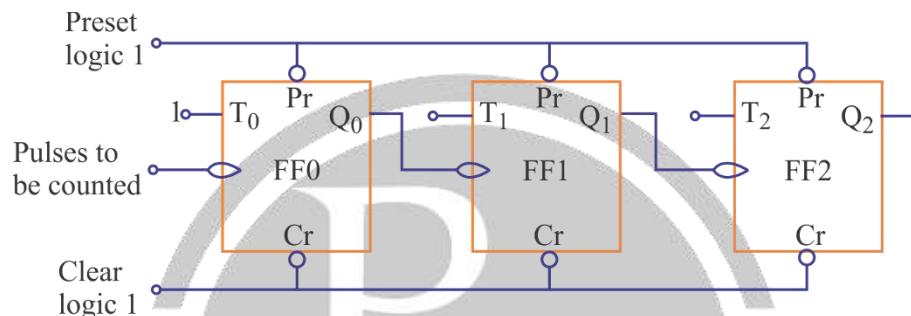


Fig. 4.31. A 3-bit Binary Counter

The truth table for 3-bit binary counter is given as:

Counter state	Count		
	Q_2	Q_1	Q_0
0	0	0	0
1	0	0	1
2	0	1	0
3	0	1	1
4	1	0	0
5	1	0	1
6	1	1	0
7	1	1	1

Output waveforms of the above counter is:

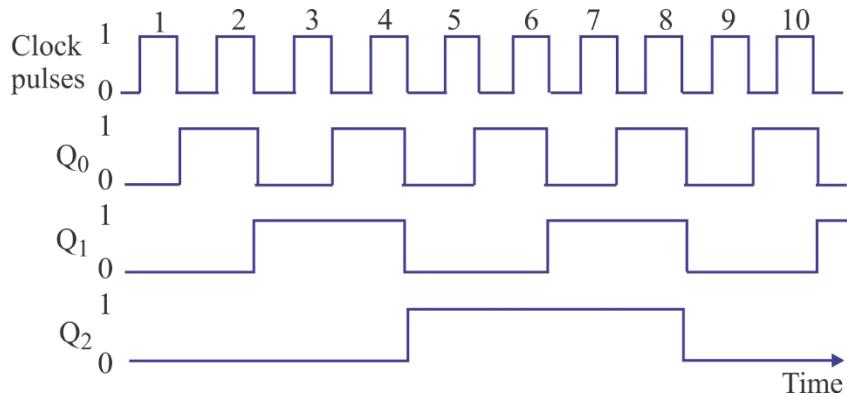


Fig. 4.32.

The frequency 'f' of clock pulses for reliable operation of the counter is given as

Where, N = number of flip-flops

t_{pd} = propagation delay of one flip-flop.

T_s = strobe pulse width.

If during the operation of counter, if some pulses are falsely operated for short duration, known as spikes, which change the state of the flip-flop. It may happen when the propagation delay of each flip-flop may vary and may happen that, all the flip-flops may not change their states or may be only one flip-flop changes its state during the pulse time.

This problem of spikes can be eliminated by using a strobe pulse with the help of strobe pulse, the state will change only when flip-flops of the counter are in steady state.

Example: In a 4-stage ripple counter, the propagation delay of a flip-flop is 50n sec. If the pulse width of the strobe is 30n sec. Find the maximum frequency at which the counter operates reliably.

Solution: The maximum frequency is

$$f_{\max} = \frac{1}{nt_{pd} + t_s}$$

n = number of flip-flops or stage = 4

t_{pd} = propagation delay of each flip-flop = 50 nsec.

t_s = Strobe pulse width = 30 nsec

$$f_{\max} = \frac{1}{(4 \times 50 + 30) \times 10^{-9}} = \frac{1000}{(200 + 30)} \text{ MHz} = \frac{1000}{230} \text{ MHz}$$

4.11.2. Modulo-6 Asynchronous Down Counter

Down counter is the counter which counts the values of pulses in descending order. Consider a Stable-8 counter ($2^3 = 8$), which uses three flip-flops.

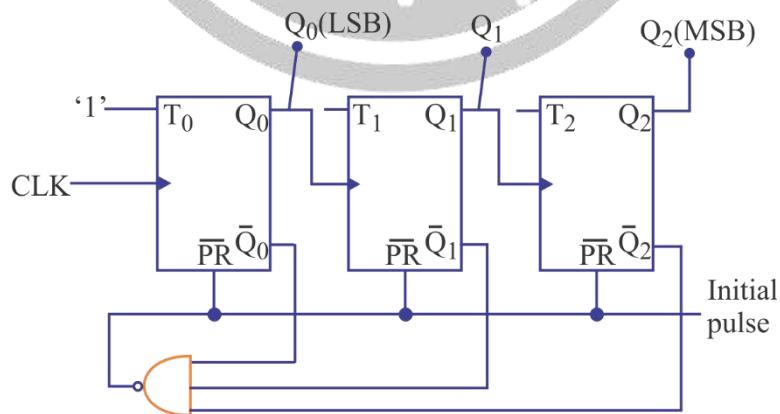


Fig. 4.33.

1. Only sequential counter can be designed. Random counter cannot be designed.
2. Glitch (undesirable state) would appear in case of asynchronous counter.
3. Speed of asynchronous counter is not fast.

4.11.5. BCD Ripple Counter

A decimal counter follows a sequence of 10 states and returns to 0 after the count of 9. Such a counter must have at least four flip-flops to represent each decimal digit, since a decimal digit is represented by a binary code with at least four bits. The sequence of states in a decimal counter is dictated by the binary code used to represent a decimal digit. If the BCD code is used, the sequence of states is as shown in the state diagram. A decimal counter is similar to a binary counter, except that the state after 1001 (the code for decimal digit 9) is 0000 (the code for decimal digit 0).

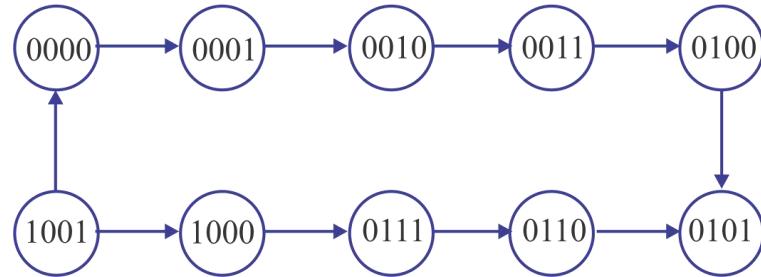


Fig. 4.34. State diagram of a decimal BCD counter.

The logic diagram of a BCD ripple counter using JK flip-flops is shown in figure below. The four outputs are designated by the letter symbol Q, with a numeric subscript equal to the binary weight of the corresponding bit in the BCD code. Note that the output of Q₁ is applied to the C inputs of both Q₂ and Q₄ and the output of Q₂ is applied to the C and output of Q₂ and Q₃ applied to J through a two input AND gate.

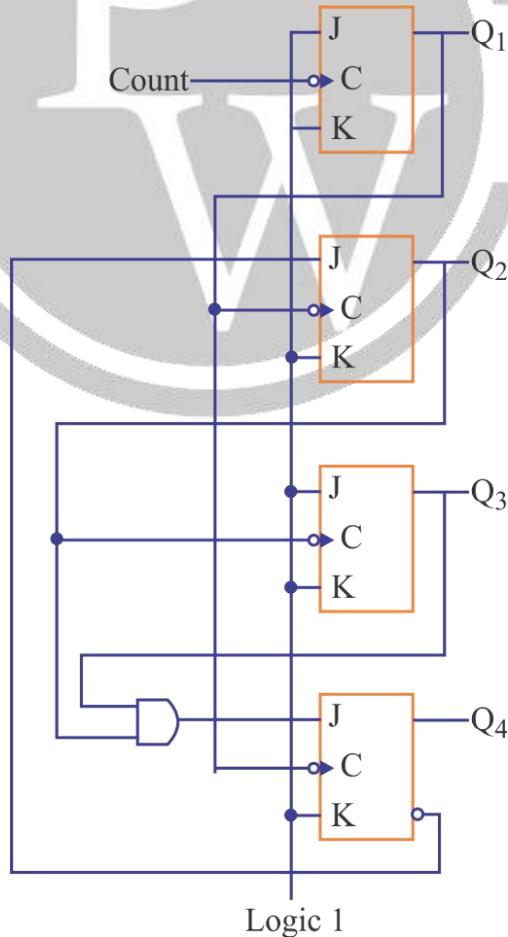


Fig. 4.35. BCD ripple counter

4.12. Synchronous Counter

The ripple counters have the advantage of simplicity (only FLIP-FLOP's are required) but their speed is low because of ripple action. The maximum time is required when the output changes from 111....1 to 00....0 and this limits the frequency of operation of ripple counters.

The speed of operation improves significantly if all the FLIP-FLOP's are clocked simultaneously. The resulting circuit is known as a synchronous counter. Synchronous counters can be designed for any count sequence (need not be straight binary).

The output Q_0 of the least-significant FLIP-FLOP changes for every clock pulse. This can be achieved by using a T-type FLIP-FLOP with $T_0 = 1$. The output Q_0 changes whenever Q_0 changes from 1 to 0. Therefore, if Q_0 is connected to T input (T_1) of the next FLIP-FLOP, Q_1 will change from 1 to 0 (or 0 to 1) when $Q_0 = 1$ ($T_1 = 1$) and will remain unaffected when $Q_0 = T_1 = 0$. Similarly, Q_2 changes whenever Q_1 and Q_0 are both "1". This can be achieved by making the T-input (T_2) of the most-significant FLIP-FLOP equal to $Q_1.Q_0$.

In addition to FF's, synchronous counters require some gates also. JK FLIP-FLOP's are the most commonly used FLIP-FLOP's for the design of synchronous counters. In this, each FLIP-FLOP has two control inputs (J and K) and circuit is required to be designed for each control input. Many programmable logic devices (PLDs) used for the design of digital systems utilise D FLIP-FLOP's for their memory elements, therefore, counter design using D FLIP-FLOP's will be useful for programming inside a PLD. It has only one control input which makes its design simpler than the design using J-K FLIP-FLOP's.

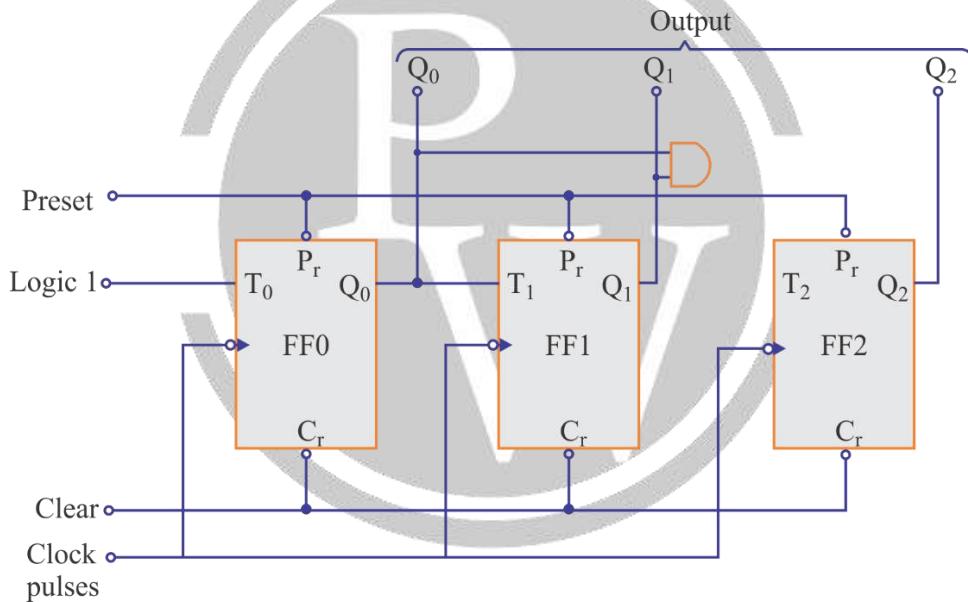


Fig. 4.36. A 3-bit Synchronous Counter

4.12.1. Synchronous Counter Design

Synchronous counters for any given count sequence and modulus can be designed in the following way:

1. Find the number of FLIP-FLOPs required.
2. Write the count sequence in the tabular form.
3. Determine the FLIP-FLOP inputs which must be present for the desired next state from the present state using the excitation table of the FLIP-FLOP's.
4. Prepare K-map for each FLIP-FLOP input in terms of FLIP-FLOP outputs as the input variables.
5. Simplify the K-maps and obtain the minimized expressions.
6. Connect the circuit using FLIP-FLOP's and other gates corresponding to the minimized expressions.

Example: Design a 3-bit synchronous counter using JK Flip-Flops.

Solution: The number of FLIP-FLOP required is 3. Let the FLIP-FLOPs be FF0, FF1, FF2 and their inputs and outputs are given below:

FLIP-FLOP	Inputs	Outputs
FF0	J ₀ , K ₀	Q ₀
FF1	J ₁ , K ₁	Q ₁
FF2	J ₂ , K ₂	Q ₂

The count sequence and the required inputs of FLIP-FLOPs is shown below.

Counter state			FLIP-FLOP INPUTS								
			FF0		FF1		FF2				
Q ₂	Q ₁	Q ₀	J ₀	K ₀	J ₁	K ₁	J ₂	K ₂			
0	0	0	1	X	0	X	0	X			
0	0	1	X	1	1	X	0	X			
0	1	0	1	X	X	0	0	X			
0	1	1	X	1	x	1	1	X			
1	0	0	1	X	0	X	X	0			
1	0	1	X	1	1	X	X	0			
1	1	0	1	X	X	0	X	0			
1	1	1	x	1	X	1	x	1			
0	0	0									

Q ₂ Q ₁		00	01	11	10
Q ₀		0	1	1	1
		1	x	x	x

$$J_0 = 1$$

Q ₂ Q ₁		00	01	11	10
Q ₀		0	x	x	0
		1	x	x	1

$$J_1 = Q_0$$

Q ₂ Q ₁		00	01	11	10
Q ₀		0	x	x	x
		1	1	1	1

$$K_0 = 1$$

Q ₂ Q ₁		00	01	11	10
Q ₀		0	x	0	x
		1	x	1	x

$$K_1 = Q_0$$

	$Q_2 Q_1$	00	01	11	10
Q_0	0	0	0	x	x
	1	0	1	x	x

$$J_2 = Q_0 Q_1 \\ (e)$$

	$Q_2 Q_1$	00	01	11	10
Q_0	0	x	x	0	0
	1	x	x	1	0

$$K_2 = Q_0 Q_1 \\ (f)$$

Fig. 4.37. K-Maps of 3-bit Synchronous Counter

Example:

Design a natural binary sequence mod-8 synchronous counter using D FLIP—FLOPS.

Solution:

The number of FLIP-FLOPS required is 3. Let the FLIP—FLOPS be FF0, FF1 and FF2 with inputs D_0 , D_1 and D_2 , respectively. Their outputs are Q_0 , Q_1 , and Q_2 respectively

Counter State			FLIP-FLOP inputs		
Q_2	Q_1	Q_0	D_0	D_1	D_2
0	0	0	1	0	0
0	0	1	0	1	0
0	1	0	1	1	0
0	1	1	0	0	1
1	0	0	1	0	1
1	0	1	0	1	1
1	1	0	1	1	1
1	1	1	0	0	0

	$Q_3 Q_2$	00	01	11	10
$Q_1 Q_0$	00	1	1	x	1
	01	x	x	x	x
	11	x	x	x	x
	10	1	1	x	x

$$J_0$$

	$Q_3 Q_2$	00	01	11	10
$Q_1 Q_0$	00	x	x	x	x
	01	1	1	x	1
	11	1	1	x	x
	10	x	x	x	x

$$K_0$$

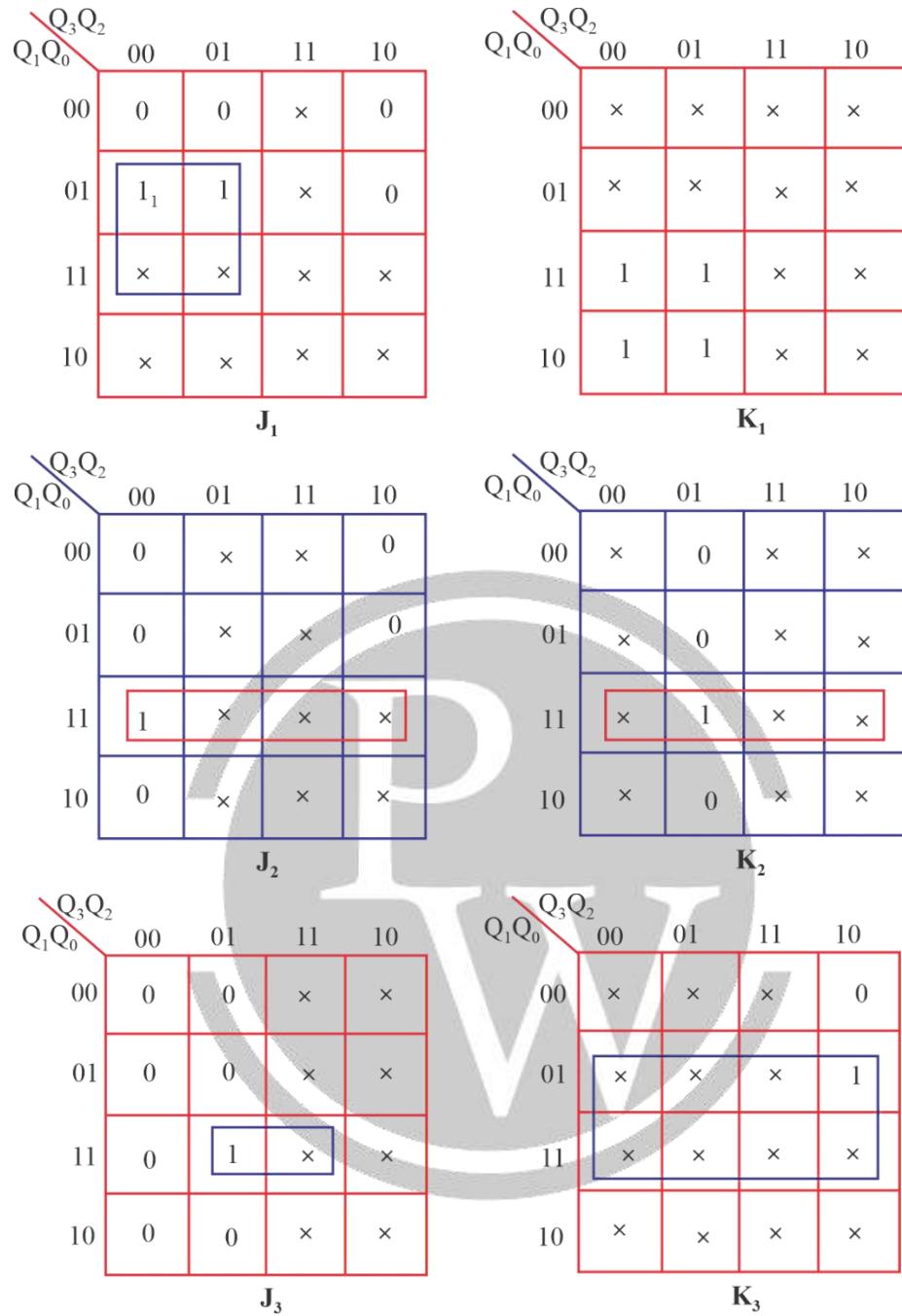


Fig. 4.38. K-Maps for 8-bit Synchronous counter

The K-maps for D_0 , D_1 and D_2 are given as,

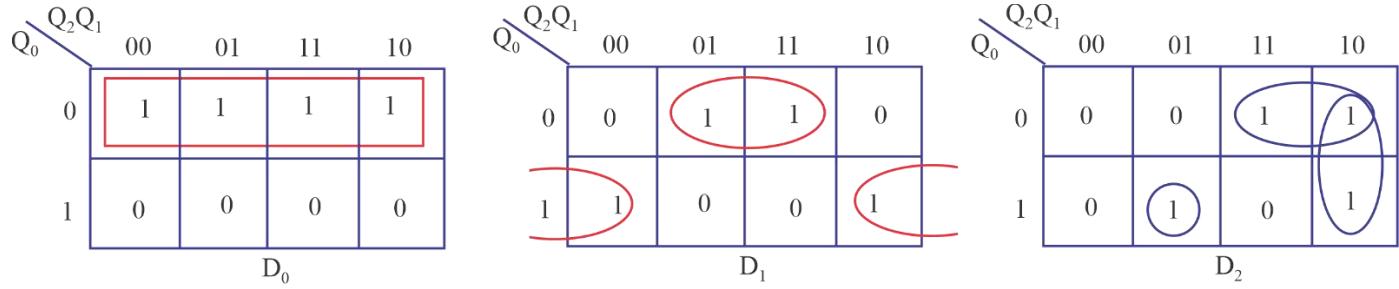


Fig. 4.39.

The minimised expressions for D0, D1 and D2 are:

$$D_0 = \bar{Q}_0$$

$$D_1 = Q_1\bar{Q}_0 + \bar{Q}_1Q_0$$

$$\begin{aligned} D_2 &= Q_2\bar{Q}_0 + Q_2\bar{Q}_1 + Q_2Q_1Q_0 = Q_2(\bar{Q}_0 + \bar{Q}_1) + \bar{Q}_2Q_1Q_0 = Q_2(\bar{Q}_0 \cdot \bar{Q}_1) + \bar{Q}_2(Q_1Q_0) \\ &= Q_2 \oplus Q_1 \cdot Q_0 \end{aligned}$$

The complete circuit of the synchronous counter using positive edge triggered D FLIP-FLOPs is shown in figure below as

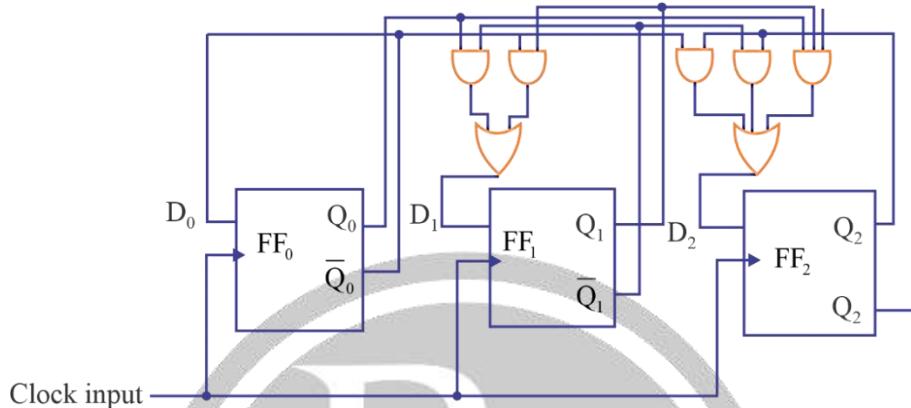


Fig. 4.40. 8-bit Synchronous Counter Circuit

4.12.2 Synchronous Sequential Circuit Models

A general block diagram of clocked sequential circuit is also known as finite state machine (FSM). Depending upon the external outputs, there are two types of models of sequential circuits.

Mealy Model

In Mealy model, the next state of the function depends on present state as well as present inputs.

Moore Model

In Moore model, the next state depends on the present state. The block diagram of a Moore model is given as

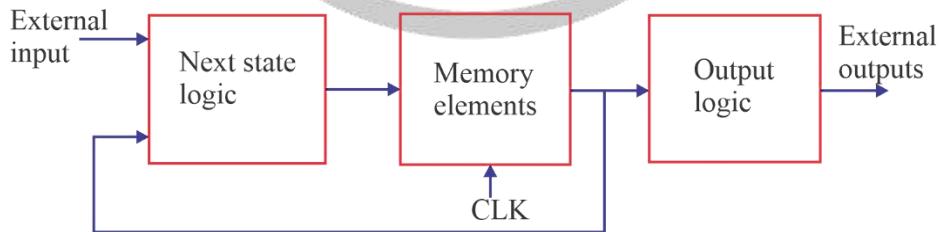


Fig. 4.41.

The systematic procedure for designing of clocked sequential circuit is based on the concept of ‘state’. Hence the sequence of inputs, present & next states and output is represented by a state table or state diagram & if the procedure follows in the form of flow chart, it is known as algorithms state machine (ASM).

4.12.3. State Diagram

It is a directed graph, consisting of vertices (or nodes) and directed arcs between the nodes. Every state of the circuit is represented by a node in the graph. A node is represented by a circle with the name of the state written inside the circle. The directed arcs represent the state transitions.

With the circuit in may one state, at the occurrence of a clock pulse, there will be a state transition to the next state and there will be an output, corresponding to the requirement of the circuit. This state transition is represented by a directed line and we use each (/) for representing present state and the next state.

Example:

Draw the state diagram of D-flip-flop.

Solution:

The D flip-flop has only input (D) & two output states ($Q = 0$ & $Q = 1$).

Using the state table or characteristic table of the D-flip flop. The state diagram is given as

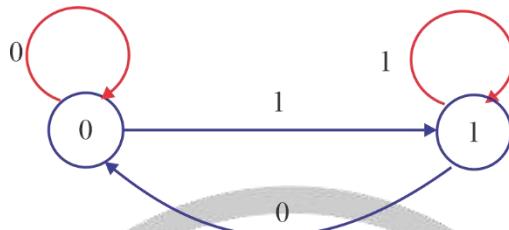


Fig. 4.42.

Example:

Draw the state diagram of a JK flip-flop.

Solution:

A JK flip-flop has inputs (J & K) and one clock input (CLK) and the two output states ($Q = 0$ & $Q = 1$).

Using the state table or characteristic table of the JK flip-flop, the state diagram is given as

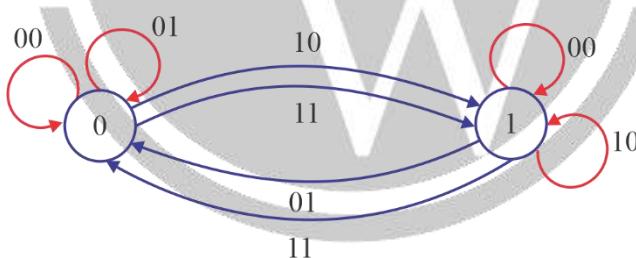


Fig. 4.43.



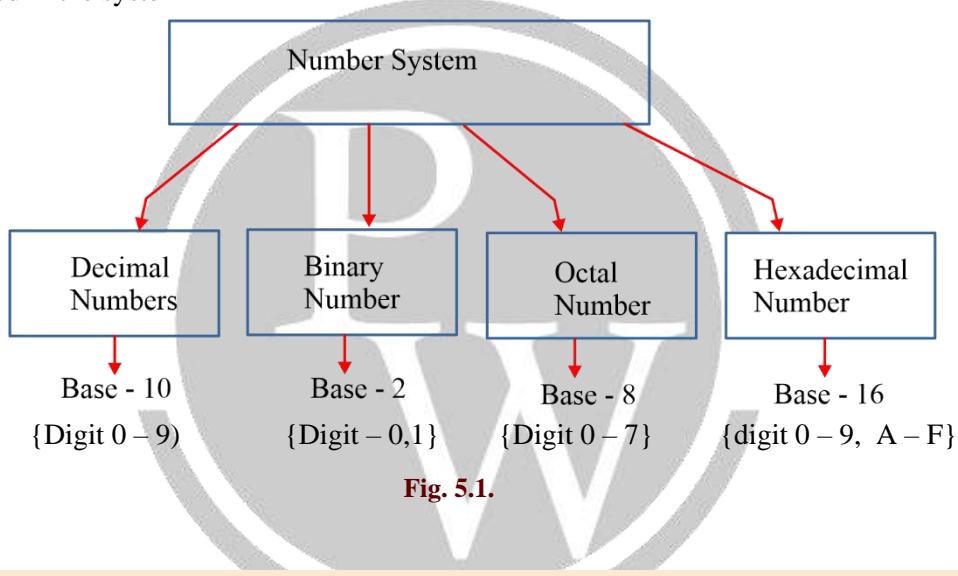
5

NUMBER SYSTEM

5.1. NUMBER SYSTEM

5.1.1. Base (Radix)

Total number of digit used in the system



5.1.2. Decimal Number System

$$\dots \quad 10^4 \quad 10^3 \quad 10^2 \quad 10^1 \quad 10^0 \quad 10^{-1} \quad 10^{-2} \quad 10^{-3} \dots$$
$$\dots \quad a_4 \quad a_3 \quad a_2 \quad a_1 \quad a_0 \quad a_{-1} \quad a_{-2} \quad a_{-3} \dots$$

$a_i \rightarrow$ Coefficient of decimal number system

$10^i \rightarrow$ Weight of decimal number system

Example: - $(501.23)_{10}$

$$\begin{array}{ccccc} 10^2 & 10^1 & 10^0 & 10^{-1} & 10^{-2} \\ 5 & 0 & 1 & 2 & 3 \end{array}$$

Base	Digit
2	0, 1
3	0, 1, 2
4	0, 1, 2, 3
5	0, 1, 2, 3, 4
6	0, 1, 2, 3, 4, 5

7	0, 1, 2, 3, 4, 5, 6
8	0, 1, 2, 3, 4, 5, 6, 7
9	0, 1, 2, 3, 4, 5, 6, 7, 8
10	0, 1, 2, 3, 4, 5, 6, 7, 8, 9
11	0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A
12	0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B
13	0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C
14	0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D
15	0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E
16	0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F

5.1.3 Binary Number System (Base (Radix) = 2)

...	2^4	2^3	2^2	2^1	2^0	2^{-1}	2^{-2}	$2^{-3} \dots$
...	a_4	a_3	a_2	a_1	a_0	a_{-1}	a_{-2}	$a_{-3} \dots$

$2^i \rightarrow$ Weight of Binary number system

$a_i \rightarrow$ Coefficient of Binary number system {0, 1}

Example:- $(101.11)_2$

2^2	2^1	2^0	2^{-1}	2^{-2}
1	0	1	1	1

5.1.4. Octal Number System (Base (Radix) = 8)

...	8^3	8^2	8^1	8^0	8^{-1}	8^{-2}	$8^{-3} \dots$
...	a_3	a_2	a_1	a_0	a_{-1}	a_{-2}	a_{-3}, \dots

$8^i \rightarrow$ Weight of Octal number system

$a_i \rightarrow$ Coefficient of Octal number system {0 - 7}

Example:- $(728.64)_8$

8^2	8^1	8^0	8^{-1}	8^{-2}
7	2	8	6	4

5.1.5 Hexadecimal Number System (Base (Radix) = 16):

...	16^3	16^2	16^1	16^0	16^{-1}	16^{-2}	$16^{-3} \dots$
...	a_3	a_2	a_1	a_0	a_{-1}	a_{-2}	a_{-3}, \dots

$16^i \rightarrow$ Weight of Hexadecimal number system

$a_i \rightarrow$ Coefficient of Hexadecimal number system {0 – 9, A–F}

Example: $(A2C.F)_{16}$

16^2	16^1	16^0	16^{-1}
A	2	C	F

5.1.6. In base conversion 2 key points are there:

- (A) Any base to Decimal conversion
- (B) Decimal to any other base conversion

(A) Any base to Decimal conversion:

$$(a_3 \ a_2 \ a_1 \ a_0 \cdot a_{-1} \ a_{-2})_r = (\)_{10}$$

$$(a_3 \times r^3 + a_2 \times r^2 + a_1 \times r^1 + a_0 \times r^0 + a_{-1} \times r^{-1} + a_{-2} \times r^{-2})_{10}$$

Case (1) : Binary to Decimal conversion

Ex. $(1011.11)_2 = (\)_{10}$

$$\Rightarrow [(1 \times 2^3) + (0 \times 2^2) + (1 \times 2^1) + (1 \times 2^0) + (1 \times 2^{-1}) + (1 \times 2^{-2})]_{10}$$

$$\Rightarrow [8 + 0 + 2 + 1 + 0.5 + 0.25]_{10}$$

$$\Rightarrow (11.75)_{10}$$

Case (2) : Octal to Decimal conversion

Ex. $(721.4)_8 = (\)_{10}$

$$\Rightarrow [(7 \times 8^2) + (2 \times 8^1) + (1 \times 8^0) + (4 \times 8^{-1})]_{10}$$

$$\Rightarrow [448 + 16 + 1 + 0.5]_{10}$$

$$\Rightarrow (465.5)_{10}$$

Case (3) : Hexadecimal to Decimal conversion

Ex. $(A2B.C)_{16} = (\)_{10}$

$$\Rightarrow [(A \times 16^2) + (2 \times 16^1) + (B \times 16^0) + (C \times 16^{-1})]_{10}$$

$$\Rightarrow [(10 \times 256) + (2 \times 16) + (11 \times 1) + (12 \times 16^{-1})]_{10}$$

$$\Rightarrow [2560 + 32 + 11 + 0.75]_{10}$$

$$\Rightarrow (2603.75)_{10}$$

Case (4) : Base 5 to Decimal conversion

Ex. $(432.22)_5 = (\)_{10}$

$$\Rightarrow [(4 \times 5^2) + (3 \times 5^1) + (2 \times 5^0) + (2 \times 5^{-1}) + (2 \times 5^{-2})]_{10}$$

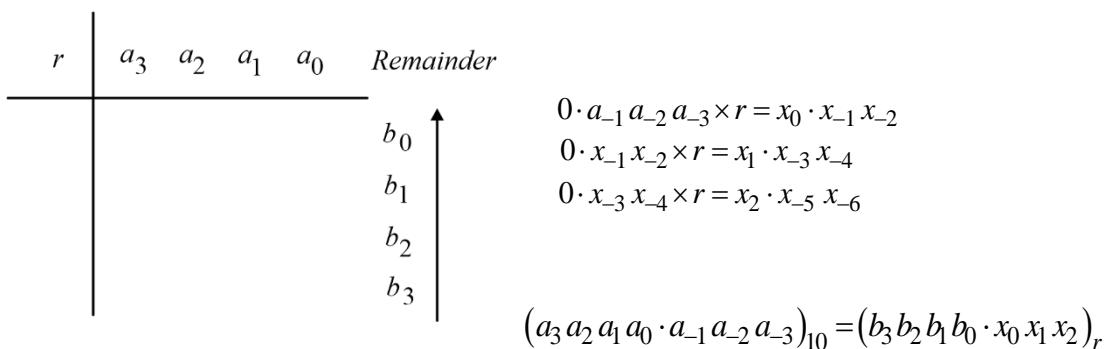
$$\Rightarrow [100 + 15 + 2 + 0.4 + 0.08]_{10}$$

$$\Rightarrow (117.48)_{10}$$

(B) Decimal to any other Base conversion

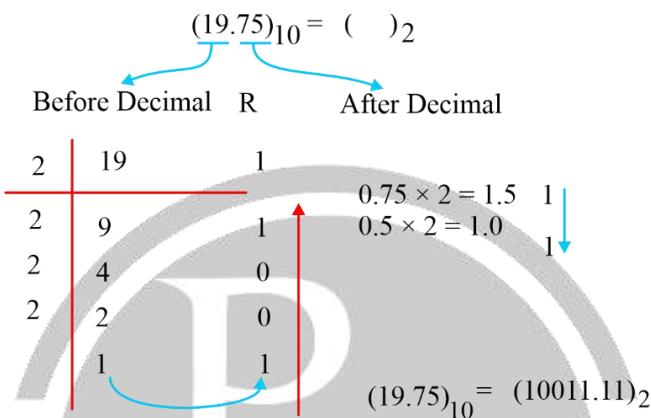
$$(a_3 \ a_2 \ a_1 \ a_0 \cdot a_{-1} \ a_{-2} \ a_{-3})_{10} = (\)_r$$

Before Decimal After Decimal



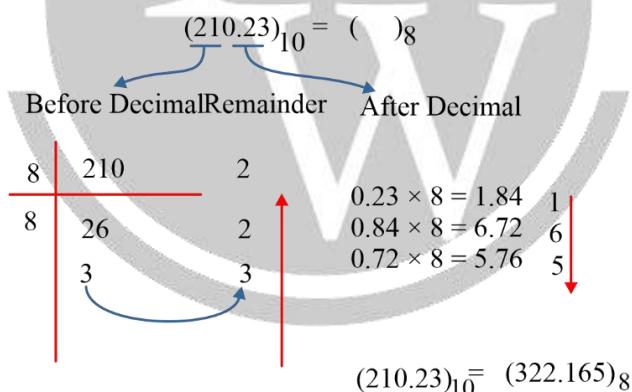
Case (1) : Decimal to Binary Base conversion.

Ex.



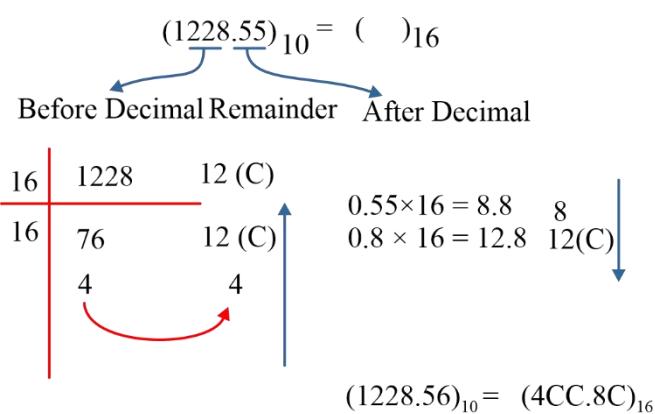
Case (2) : Decimal to Octal Base conversion.

Ex.



Case (3) : Decimal to Hexadecimal Base conversion.

Ex.



5.2. Some Special Case

Case (1): Binary to Octal base conversion

Example: $(10110111)_2 = (\quad)_8$

Octal \rightarrow means base 8

$$8 = 2^3$$

Every three digits of binary represent one digit of octal

010 110 111

2 6 7

Hence $(10110111)_2 = (267)_8$

Case (2): Binary to Hexadecimal base conversion

Example: $(10110111)_2 = (\quad)_{16}$

Hexadecimal \rightarrow means base 16

$$16 = 2^4$$

Every four digits of binary represent one digit of Hexadecimal.

0101 1011

5 11(B)

Hence $(10110111)_2 = (5B)_{16}$

5.1.2. BCD (Binary Coded Decimal)

- In this each digit of the decimal number is represented by its four-bit binary equivalent. It is also called natural BCD or 8421 code. It is weighted code.
- Excess – 3 Code:** This is an non weighted binary code used for decimal digits. Its code assignment is obtained from the corresponding value of BCD after the addition of 3.
- BCO (Binary Coded Octal):** In this each digit of the Octal number is represented by its three-bit binary equivalent.
- BCH (Binary Coded Hexadecimal):** In this each digit of the hexadecimal number is represented by its four bit binary equivalent.

Decimal Digits	BCD 8421	Excess – 3	Octal digits	BCO	Hexadecimal Digits	BCH
0	0000	0011	0	000	0	0000
1	0001	0100	1	001	1	0001
2	0010	0101	2	010	2	0010
3	0011	0110	3	011	3	0011
4	0100	0111	4	100	4	0100
5	0101	1000	5	101	5	0101
6	0110	1001	6	110	6	0110
7	0111	1010	7	111	7	0111
8	1000	1011			8	1000

9	1001	1100			9	1001
					A	1010
					B	1011
					C	1100
					D	1101
					E	1110
					F	1111

Don't care values or unused states in BCD code are 1010, 1011, 1100, 1101, 1110, 1111.

Don't care values or unused states in excess – 3 code are 0000, 0001, 0010, 1101, 1110, 1111.

The binary equivalent of a given decimal number is not equivalent to its BCD value.

Example: $25_{10} = 11001_2$.

The BCD equivalent of decimal number $25 = 00100101$ from the above example the BCD value of a given decimal number is not equivalent to its straight binary value.

The BCO (Binary Coded Octal) value of a given Octal number is exactly equal to its straight binary value.

Example: $25_8 = 21_{10} = 010101_2$

The BCO Value of 25_8 is 010101 .

From the above example, the BCO value of a given Octal number is same as binary equivalent of the same number.

The BCH (Binary Coded Hexadecimal) value of a given hexadecimal number is exactly equal to its straight binary.

Example: $25_{16} = 37_{10} = 100101_2$

The BCH value of hexadecimal number $25_{16} = 00100101$.

From this example the above statement is true.

	Binary	Octal	Decimal	Hexadecimal
Complement	$r=2$	$r=8$	$r=10$	$r=16$
$(r-1)$'s	1's	7's	9's	15's
r 's Complement	2's	8's	10's	16's

Example: Add the two Binary numbers 101101_2 .

Augned 101101

addend 100111

1111

Sum 1010100

Example: Subtract the Binary number 100111_2 from 101101_2 .

Minuend : 101101

Subtracted: 100111

Difference: 000110

Example: Multiple the Binary number 1011_2 from 101_2 .

$$\begin{array}{r}
 \text{Multiplicand: } 1011 \\
 \text{Multiplier: } X101 \\
 \hline
 & 1011 \\
 & 0000 \\
 & 1011 \\
 + & \\
 \hline
 \text{Product: } 110111
 \end{array}$$

While storing the signed binary numbers in the internal registers of a digital computer} most significant bit position is always reserved for sign bit and the remaining bits are used for magnitude.

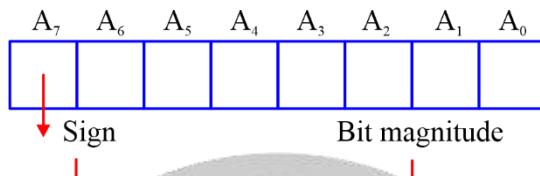


Fig. 5.2.

When the binary number is positive, the sign is represented by '0'. When the number is negative, the sign is represented by '1'.

5.2.2. Fixed-Point Representation and Floating-Point Representation;

The representation of the decimal point (ordinary point) in a register is complicated by the fact that it is characterized by a position between two flip-flops in the register.

There are two ways of specifying the position of the decimal point in a register.

- (1) Fixed Point and
- (2) Floating Point.

The fixed point method assumes that the decimal point (or binary point) is always fixed in one position. The two positions most widely used are (1) a decimal point in the extreme left of the register to make the stored number a fraction, and (2) a decimal point in the extreme right of the register to make the stored number an integer.



The floating-point representation uses a second register to store a number that designates the position of the decimal point in the first register.

Positive numbers are stored in the registers of digital computer in sign magnitude form only.

Negative number can be represented in one of three possible ways.

1. Signed – magnitude representation.
2. Signed – 1's complement representation.
3. Signed – 2's complement representation.

Example: +9

-9

- Signed – magnitude 0 0001001 (a) 1 000 1001 signed – magnitude
 (b) 1 111 0110 signed – 1's complement
 (c) 1 111 0111 signed – 2's complement

The 2's complement of a given binary number can be formed by leaving all least significant zeros and the first non-zero digit unchanged, and then replacing 1's by 0's and 0's by 1's in all other higher significant digits.

Example: The 2's complement of 10011000_2 is 01101000 .

Subtraction using 2's complement: Represent the negative number in signed 2's complement form, add the two numbers, including their sign bit, and discard any carry out of the most significant bit.

Since negative numbers are represented in 2's compliment form, negative results also obtained in signed 2's compliment form.

Example: 1's complement:

+ 6 0000110	− 6 1111001	+ 6 0000110	− 6 1111001
+ 9 0001001	+ 9 0001001	− 9 1110110	− 9 1110110
<hr/>	<hr/>	<hr/>	<hr/>
+ 15 0001111	+3 (i) 0000010	− 3 1111100	− 15 (1) 1101111
	Carry + 1		Carry + 1
	<hr/>		<hr/>
	+ 3 0000011		1110000
	carry		carry

The advantage of signed 2's complement representation over the signed 1's compliment form (and the signed – magnitude form) is that it contains only one type of zero.

The general form of floating – point number is mr^e . Where M = Mantissa, r = base, e = exponent.

Example: $+0.3574 \times 10^5$.

The mantissa can be a fixed point fraction or fixed point integer.

Normalization: Getting non-zero digit in the most significant digit position of the mantissa is called Normalization.

- If the floating point number is normalized, more number of significant digits can be stored, as a result accuracy can be improved.
- A zero cannot be normalized because it does not contain a non-zero digit. The hexadecimal code is widely used in digital systems because it is very convenient to enter binary data in a digital system using hexcode.
- The parity of a digital word is used for detecting error in digital transmission. Hollerith code is used for punched card data.
- In weighted codes, each position of the number has specific weight. The decimal value of a weighted code number is the algebraic sum of the weights of those positions in which 1's appears.
- Most frequently used weighted codes are 8421, 2421 code, 5211 code and 8421 code.
- **Reflective Code:** A code is called reflective or self-complimenting, if the code for 9 is the compliment for the code for 0, code for 8 is the compliment from 1 and so on. 2421, 842'1', 5211 are examples for reflected codes.
- **Sequential Code:** A code is called sequential, if each successive code-is one binary number greater than its preceding code.

Example: 8421



GATE Exam 2025?



SHRESHTH BATCH

ME | CE | EE | EC | CS & IT

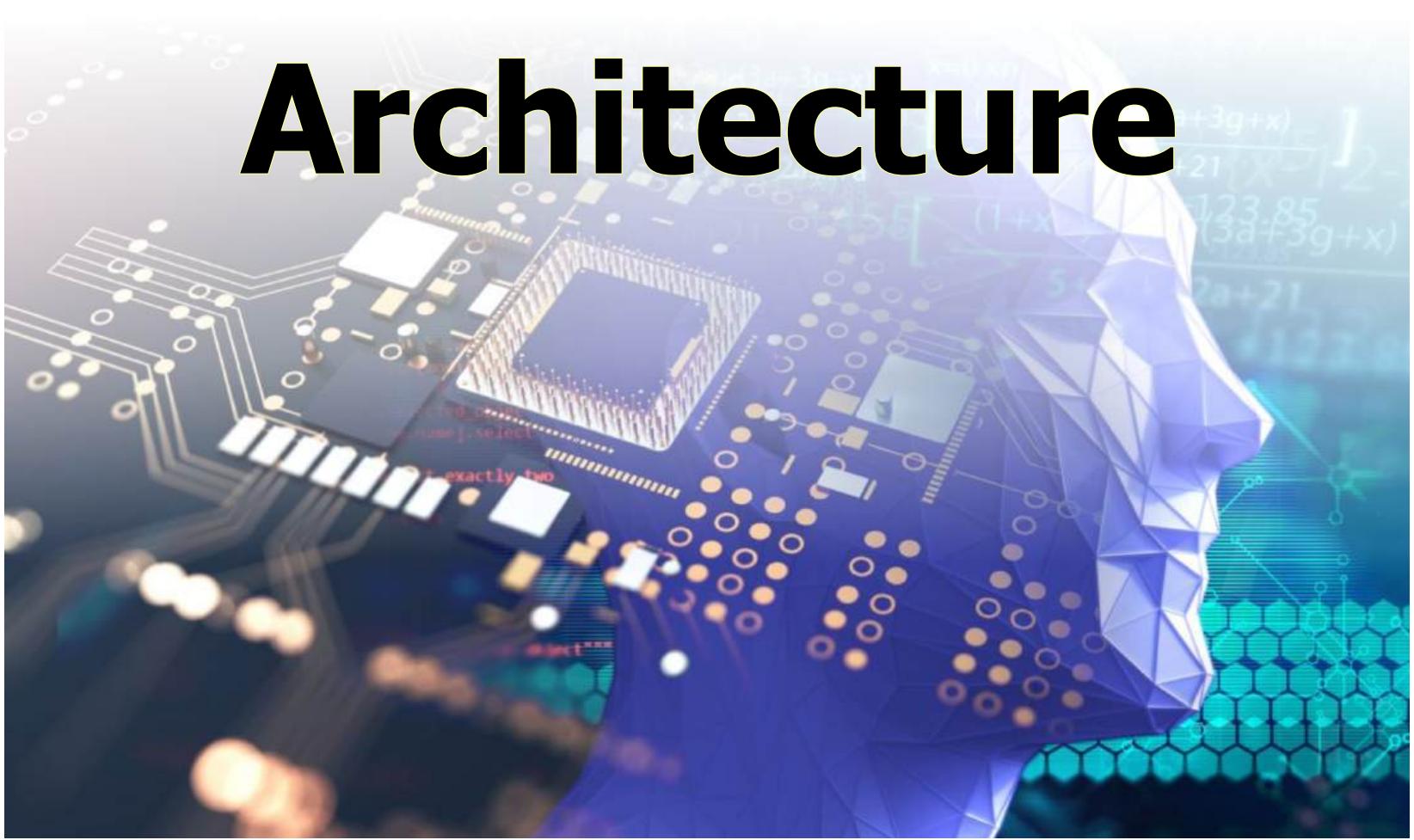
- Live Classes & Recorded Videos
- DPPs & Solutions
- Doubt Solving
- Batches are available in *English*

Weekday & Weekend
Batches Available

JOIN NOW!



Computer Organization & Architecture



Computer Organization & Architecture

INDEX

- | | | |
|-----|--------------------------------|-------------|
| 1. | Machine Instructions | 4.1 – 4.5 |
| 2. | Addressing Modes..... | 4.6 – 4.8 |
| 3. | ALU Data Path | 4.9 – 4.13 |
| 4. | CPU Control Unit Design..... | 4.14 – 4.19 |
| 5. | Memory Interfacing | 4.20 – 4.22 |
| 6. | Input Output Interfacing | 4.23 – 4.28 |
| 7. | Pipelining | 4.29 – 4.43 |
| 8. | Cache Memory | 4.44 – 4.48 |
| 9. | Main Memory | 4.49 – 4.51 |
| 10. | Secondary Storage..... | 4.52 – 4.56 |

1

MACHINE INSTRUCTIONS

1.1 Introduction

Opcode – Specifies the operation code. The number of bits in opcode depends on total operations.

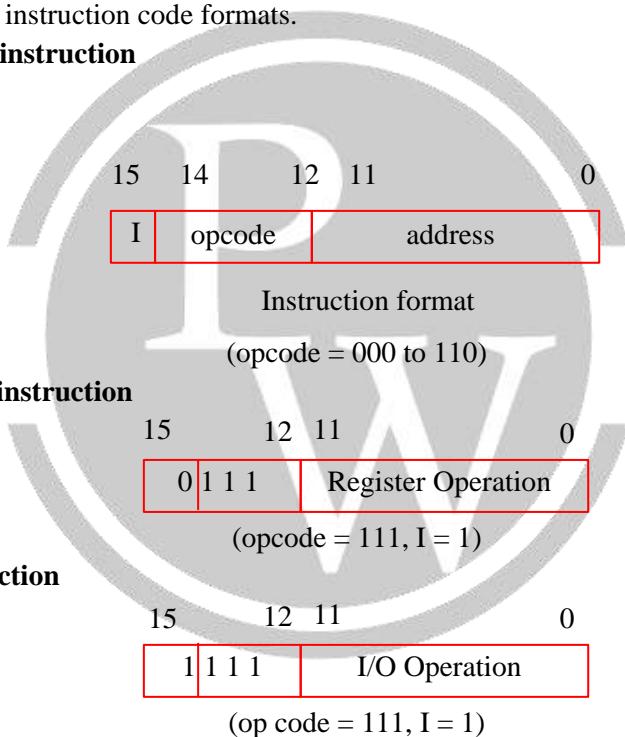
Address – Specifies the operand address.

- The basic computer has three instruction code formats.

(i) **Memory – Reference instruction**

$I = O \Rightarrow$ Direct addressing

$I = 1 \Rightarrow$ Indirect addressing

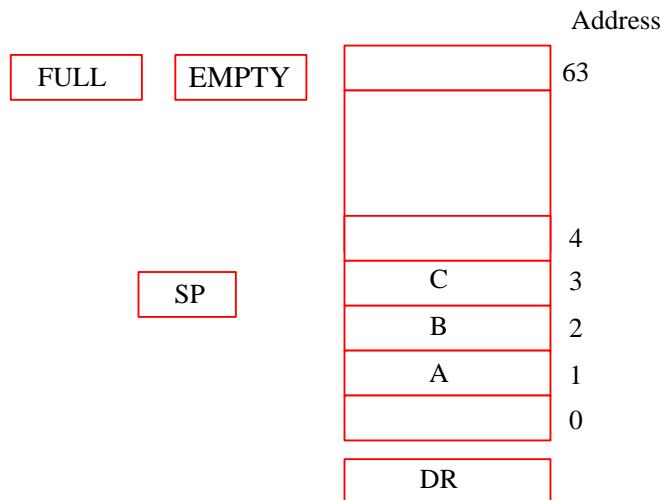


A stack can be placed in a portion of a large memory or it can be organized as a collection of a memory words or registers. i.e., it may be

- (i) Register stack
- (ii) Memory stack

1.1.1. Register Stack

- The following figure shows the organization of a 64-word register stack. The stack pointer register SP contains a binary number whose value is equal to the address of the word that is currently on top of stack.
- Initially SP is cleared, EMPTY is set to 1, FULL is cleared to 0. If the stack is not full (if FULL = 0), a new item is inserted. With a push operation.

**PUSH:**

$SP \leftarrow SP + 1$	[increment stack pointer]
$M\{SP\} \leftarrow DR$	[write item on top of stack]
If ($SP = 0$) then $FULL \leftarrow 1$	[check if stack is full]
$EMTY \leftarrow 0$	[mark the stack not empty]

- A item is deleted from the stack if the stack is not empty (if $EMTY = 0$), called POP operation

$DR \leftarrow M\{SP\}$	[Read item from top of stack]
$SP \leftarrow Sp - 1$	[Decrement stack pointer]
If ($SP = 0$) then $EMTY = 1$	

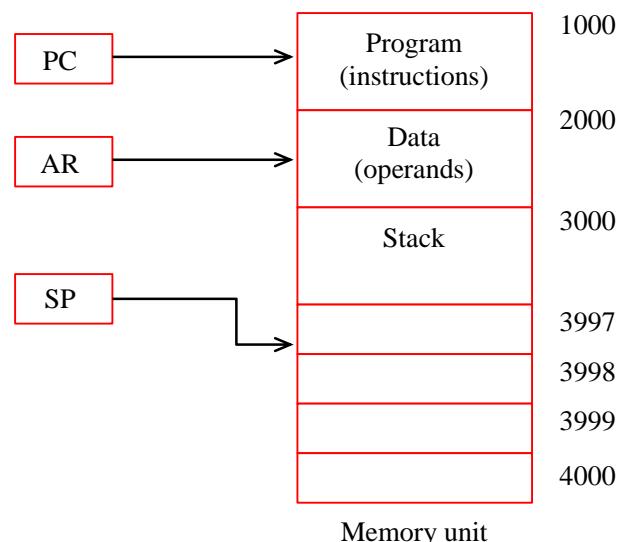
1.1.2. Memory Stack

- A stack can be implemented in a random-access memory. The stack can be implemented by assigning a portion of memory to a stack operation and using a processor register as a stack pointer. i.e.,
- A new item is inserted with the push operation as follows:

PUSH: $SP \leftarrow SP - 1$
 $M[SP] \leftarrow DR$

- A new item is deleted with a POP operation as follows

$DR \leftarrow M[SP]$
 $SP \leftarrow SP + 1$



- A stack pointer is loaded with an initial value. This initial value must be the bottom address of an assigned stack in memory. SP is automatically incremented or decremented with every **PUSH** or **POP** operation.

1.1.3. Reverse Polish Notation

- An expression in post fix form is often called reverse polish notation.

The infix expression **(A + B) * [C*(D + E) + F]** in reverse polish notation as **AB + DE + C * F + ***

- A reverse polish expression can be implemented or evaluated using stack for the expression

$$X \leftarrow (A + B) * (C + D)$$

With values A = 2, B = 3, C = 4, d = 2 is

$$X \leftarrow (2 + 3) * (4 + 2) \leftarrow \text{Infix expression}$$

$$x \leftarrow 23 + 42 + * \leftarrow \text{Reverse polish expression}$$

1.2 Instruction Types

- The basic unit of program is the instruction.

Instructions are classified based on

- (1) Opcode – called functional classification
- (2) Based on number of references.

1.3 Functional Classification

- Data transfer instructions – (LOAD, STOR, MOV, EXCT, IN, OUT, PUSH, POP)
- Arithmetic instructions – (INC, DEC, ADD, SUB, MUL, DIV)
- Logical Instructions & Shift instructions. (CLR, COM, AND, XOR, OR, SHR, SHL, SHRA, SHLA, RO, ROL....)
- Branching instructions. (ABR, JMP, SKP, CALL, RETURN)

1.4 Based on Number of References

Based on number of references, the instructions can be classified as

- 4 – address instructions
- 3- address instructions
- 2 – address instructions
- 1 – address instructions
- 0 – address instructions
- RISC Instructions

(i) 4 – Address Instructions:

Opcode	A ₁	A ₂	A ₃	A ₄
--------	----------------	----------------	----------------	----------------

- A₁, A₂ refers operands
- A₃ refers destination
- A₄ next instruction address.
- Since A₄ is like program counter, the processor which supports 4-address instructions need not use PC.
- The length of instruction is more, hence requires more than one reference.

(ii) 3 – address Instructions

Opcode	A ₁	A ₂	A ₃
--------	----------------	----------------	----------------

- Each address field specify a register or memory operand.
- It results in short program when evaluating arithmetic expressions.
- Requires too many bits to specify three addresses

Example: To evaluate X = (A + B) * (C + D)

Add R₁, A, B R₁ ← M[A] + M[B]

Add R₂, C, D R₂ ← M[C] + M[D]

MUL X, R₁, R₂ M[X] ← R₁ * R₂

For these instructions program counter must be required.

(iii) 2 – address Instructions:

Opcode	A ₁	A ₂
--------	----------------	----------------

- A₁ – first operand and destination
- A₂ – second operand
- Uses MOV instruction for data transfer

Example:

X ← (A + B) * (C + D)

MOV R₁, A

ADD R₁, B

MOV R₂, C

ADD R₂, D

MUL R₁, R₂

MOV X R₁

One of the operand permanently lost.

(iv) 1 – address Instructions

- Uses an implied accumulator (AC) register for all data manipulation.
- Easily decoded and processed instructions.

Example: X ← (A + B) * (C + D)

LOAD	A	AC ← M[A]
ADD	B	AC ← AC + M[B]
STOR	T	M[T] ← AC
LOAD	C	AC ← M[C]
ADD	D	AC ← AC + M[D]
MUL	T	AC ← AC * M[T]
STOR	X	M[X] ← AC

(v) Zero – address Instructions:

- The operands are referenced complexity from stack.
- More complex approach than others.
- Any changes in order of operands effect the result.

Example: X = (A + B) * (C + D)

PUSH	A
PUSH	B
ADD	
PUSH	C
PUSH	D
ADD	
MUL	
POP	X

(vi) RISC Instructions:

- All instructions are executed with in the registers of CPU, without referring to memory (Except LOAD, STOR)
- Uses in RISC processor
- LOAD & STOR are used between data transfer.

Example: X = (A + B) * (C + D)

LOAD	R ₁ , A
LOAD	R ₂ , B
LOAD	R ₃ , C
LOAD	R ₄ , D
ADD	R ₁ , R ₁ , R ₂
ADD	R ₃ , R ₃ , R ₄
MUL	R ₁ , R ₁ , R ₃
STOR	X, R ₁



2

ADDRESSING MODES

2.1 Introduction

The various addressing modes commonly used are:

Implied Mode:

In this mode the operands are specified implicitly in the definition of the instruction.

Example:

- Complement Accumulator (CMA) [All register reference instructions that use an accumulator are implied - mode instructions]
- Zero-address instructions in a stack organized computer. [here the operands are implied to be on top of the stack]

Immediate Mode:

In this mode the operand is specified in the instruction itself.

- It is very faster
- Useful for initializing register to a constant value.

Example:

ADD 10 i.e., AC \leftarrow AC + 10

- The range of value limited by the address field

Register Mode:

In this mode the operands are in registers that reside within the CPU.

- A K-bit field can specify any one of 2^k registers.
- Reduces the length of the address field.

Example:

ADD R1

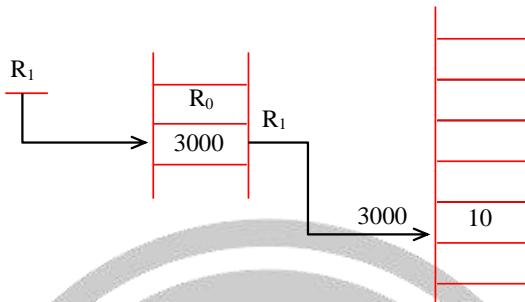
Register Indirect Mode:

In this mode the instruction specifies a register in the CPU whose contents give the address of an operand in memory.

- Before using this mode, the programmer must ensure that the memory address of the operand is placed in the processor register with a previous instruction.
- The advantage is the address field uses fewer bits to select or register.

Example:

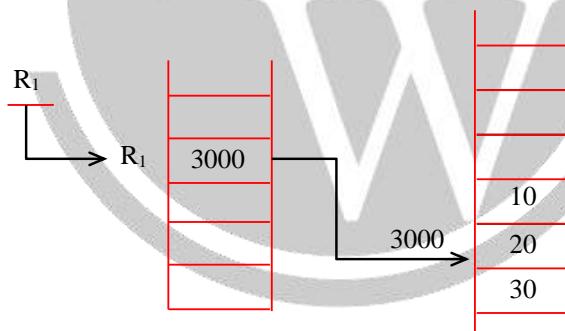
ADD (R₁)

**Autoincrement or Autodecrement Mode:**

It is similar to register indirect mode except that the register is incremented or decremented after its value is used to access memory.

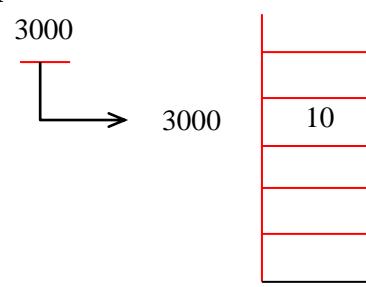
Example:

ADD (R₁)

**Direct Addressing Mode:**

In this mode the effective address is equal to the address part of the instructions.

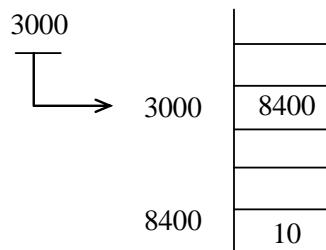
- The operand resides in memory and its address is given directly by the address field of the instruction.
- Used to represent global variables in a program.
- In a branch type instruction, the address field specifies the actual branch address space.
- Allows to access limited address space.
- The effective address is defined to be the memory address obtained from the computation dictated by the given addressing mode.
- The direct addressing mode is also called “Absolute mode”.



Indirect Addressing mode:

In this mode, the address field of the instruction gives the address where the effective address is stored in memory.

- Allows to access larger address space
- Requires 2 memory cycles to read an operand

Example ADD 3000**2.1.1 Displacement addressing modes**

The address field of the instruction added to the content of a specific register in the CPU. i.e,
Effective Address = Address part of instruction + content of Register.

2.1.2 The Various displacement addressing modes are**Relative addressing mode:**

In this mode the content of program counter (PC) is added to the address part of the instruction.

- The address part is a signed number (2's complement) that can be either positive or negative.
- The result produces effective address whose position in memory is relative to the address of the next instruction.

$$EA = \text{Address Part (off set)} + \text{PC value}$$

- Used with branch-type instructions when the branch address is in the area surrounding the instruction word itself.
- The advantage is, address field can be specified with a small number of bits compared to direct address.

Indexed Addressing mode:

In this mode the content of an index register is added to the address part of the instruction.

- Index register contains index value.
- Address part specifies the beginning address of a data array in memory.

$$EA = \text{Address Part (base address of data array)} + \text{Index register value (index value)}$$

- Used for accessing array of data.
- The index register can be special CPU register or any general purpose register.

Base register Addressing mode:

In this mode the content of a base register is added to the address part of the instruction.

- The base register is assumed to hold base address.
- The address field gives the displacement relative to this base address.

$$EA = \text{Address Part (displacement/offset)} + \text{Base register value (Base address)}$$



3

ALU DATA PATH

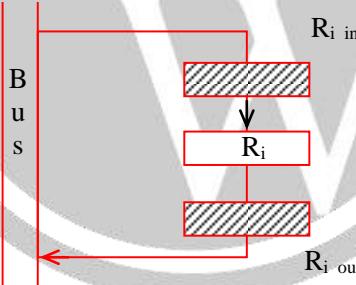
3.1 Introduction

- The sequence of operations involved in processing an instruction constitutes an instruction cycle.
The instruction cycle consists of phases like.
 - (1) Fetch cycle
 - (2) Decode cycle
 - (3) Operand fetch cycle
 - (4) Execute cycle
- To perform these, the processor unit has to perform set of operations called “Micro-Operations”.

3.2 The Basic Operations Performed are

3.2.1 Register transfers:

- In general, the input & output of register R_i are connected to the bus via switches controlled by the signals $R_{i\text{ in}}$ and $R_{i\text{ out}}$



- When $R_{i\text{ in}}$ is set to 1, the data on the bus are loaded into R_i .
- When $R_{i\text{ out}}$ is set to 1, the contents of register R_i are placed on the bus.
- To transfer the contents of R_1 to register R_4 : $R_4 \leftarrow R_1$.
- Enable the output of register R_1 by setting $R_{1\text{ out}}$ to 1. This places the contents of R_1 on the processor bus.
- Enable the input to register R_4 by setting $R_{4\text{ in}}$ to 1. This loads data from the processor bus into register R_4 .

3.2.2 Performing ALU operations

The ALU has two inputs A and B and one output

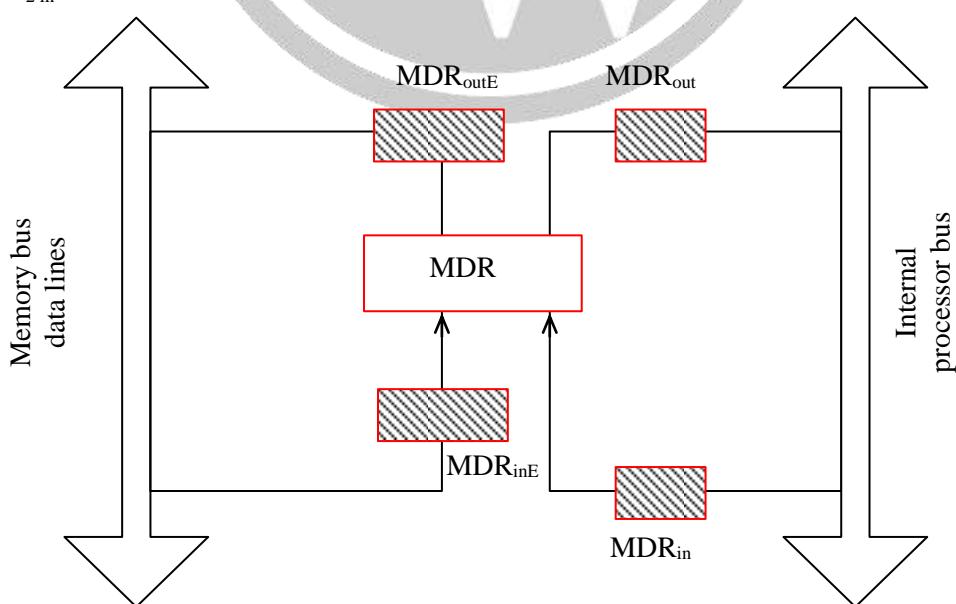
- A gets operand from output of MUX
- B gets operand from bus
- The result is gets stored in Z-register.

The sequence of steps for ALU operation $R_3 \leftarrow R_1$

- (1) The contents of R_1 are loaded in Y.
 - (2) The contents of Y_1R_2 are applied to A & B inputs of ALU, performs ALU operation & stores the result in Z-register.
 - (3) The contents of Z are stored in R_3 .
- The sequence of operations is
 - (1) R_1 out, Y_{in}
 - (2) R_2 out, select Y, Add, Z_{in}
 - (3) Z_{out} , R_3 in
 - The functions performed by ALU depends on the signals applied to its control lines. (Here Add line is set to 1).
 - Only one register output can be connected to the bus during any clock cycle.
 - The no of steps indicates no of clocks.

3.2.3 Fetching a word from memory (read operation)

- To read a memory word, consider MOV (R_1), R_2 . The action needed to execute this instruction are
 - (1) $MAR \leftarrow [R_1]$
 - (2) Start a read operation on the memory bus
 - (3) Wait for the MFC (Memory function to complete) response from memory.
 - (4) Load MDR from the memory bus
 - (5) $R_2 \leftarrow [MDR]$
- Hence the memory read operation sequence of operations is
 - (1) R_1 out, MAR_{in} , Read
 - (2) WMFC (wait for memory operation to complete), MDR_{inE}
 - (3) MDR_{out} , R_2 in



3.2.4 Storing a word in Memory (write operation)

- The sequence of steps for write operation $\text{MOV R}_2,$
 - (1) The desired address is loaded into MAR
 - (2) The data to be written is loaded into MDR
 - (3) Write command is issued.
 - (4) Wait for memory operation to complete.
- The sequence of operations is
 - (1) $\text{R}_1 \text{ out, MAR}_{\text{in}}$
 - (2) $\text{R}_2 \text{ out, MDR}_{\text{in}}, \text{ write}$
 - (3) $\text{MDR}_{\text{outE}}, \text{ WMFC}$

3.2.5 Branch Instructions

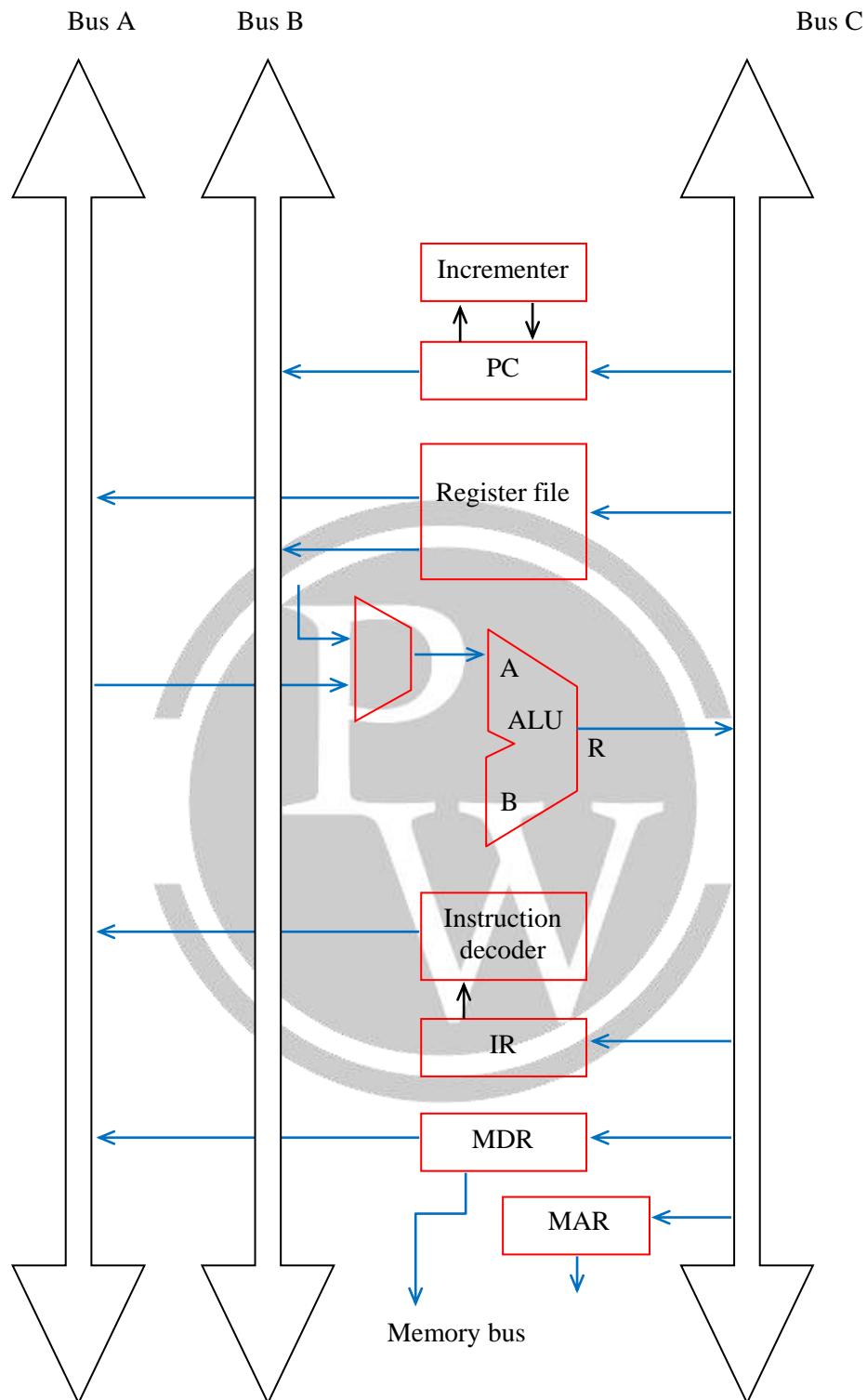
- A branch instruction replaces the content of PC with the branch target address. The address is obtained by adding an offset X, which is given in the branch instruction, to the updated value of PC.
- The control sequence for branching (unconditional) is
 - (1) $\text{PC}_{\text{out}}, \text{PC}_{\text{in}}, \text{Y}_{\text{in}}, \text{WMFC}$
 - (2) $\text{Z}_{\text{out}}, \text{PC}_{\text{in}}, \text{Y}_{\text{in}}, \text{WMFC}$
 - (3) $\text{MDR}_{\text{out}}, \text{IR}_{\text{in}}$
 - (4) Offset – of – $\text{IR}_{\text{out}}, \text{Add}, \text{Z}_{\text{in}}$

3.2.6 Execution of complete Instruction

- Executing an instruction requires ($\text{Add} (\text{R}_3), \text{R}_1$)
 - (1) Fetch the instruction
 - (2) Fetch the first operand (memory location specified by R_3)
 - (3) Perform the addition
 - (4) Load the result into R_1
- The control sequence for the execution of $\text{ADD} (\text{R}_3), \text{R}_1$ in a single-bus organization is
 - (1) $\text{PC}_{\text{out}}, \text{MAR}_{\text{in}}, \text{Read, Select, Add, Z}_{\text{in}}$
 - (2) $\text{Z}_{\text{out}}, \text{PC}_{\text{in}}, \text{Y}_{\text{in}}, \text{WMFC}$
 - (3) $\text{MDR}_{\text{out}}, \text{IR}_{\text{in}}$
 - (4) $\text{R}_3 \text{ out, MAR}_{\text{in}}, \text{ Read}$
 - (5) $\text{R}_1 \text{ out, Y}_{\text{in}}, \text{ WMFC}$
 - (6) $\text{MDR}_{\text{out}}, \text{select Y, Add, Z}_{\text{in}}$
 - (7) $\text{Z}_{\text{out}}, \text{R}_1 \text{ in, End}$

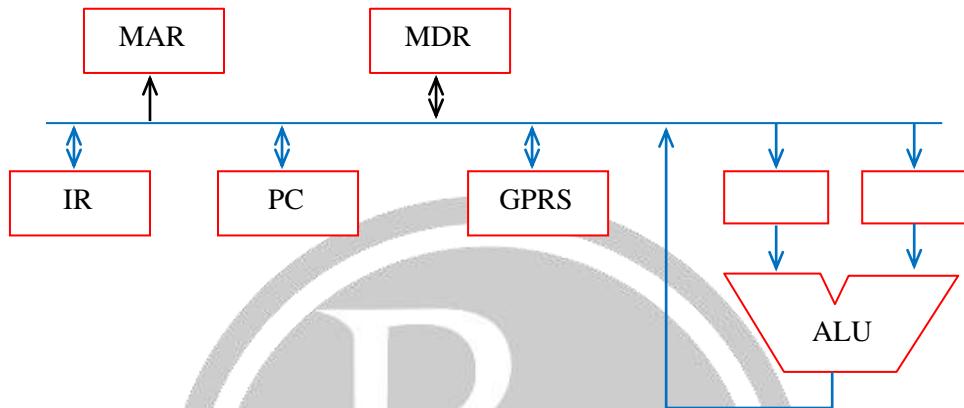
3.3 Multiple - Bus Organization

- With simple single bus structure, the resulting control sequence is quite long because only one data item can be transferred over the bus in a clock cycle. To reduce the number of steps needed, most commercial processors provide multiple internal paths that enable several transfers to take place in parallel.
- The three bus organization of data path is



- All general – purpose registers are combined into a single block called register file. It has two – outputs, allowing the contents of two different registers to be accessed simultaneously and have their contents placed on buses A and B. The data on bus ‘C’ to be loaded into third register during same clock.

- Buses A and B are used to transfer the source operands to the A and B inputs of ALU, the result is transferred to the destination over bus C.
- The control sequence for instruction ADD R₄, R₅, R₆
 - (1) P_{Cout}, R = B, MAR_{in}, Read, Inc PC
 - (2) WMFC
 - (3) MDT_{out} B, R = B, IRin
 - (4) R₄ outA, R₅ outB, select A, Add, R₆ in, end.

Example (1)

The ALU, bus & registers in data path are of identical size. All operations including incrementing of PC and GPRSs are to be carried out in the ALU. 2-clock cycles are needed for memory read operation. (one for loading address in MAR and loading data from memory into the MDR).

4

CPU CONTROL UNIT DESIGN

4.1 Introduction

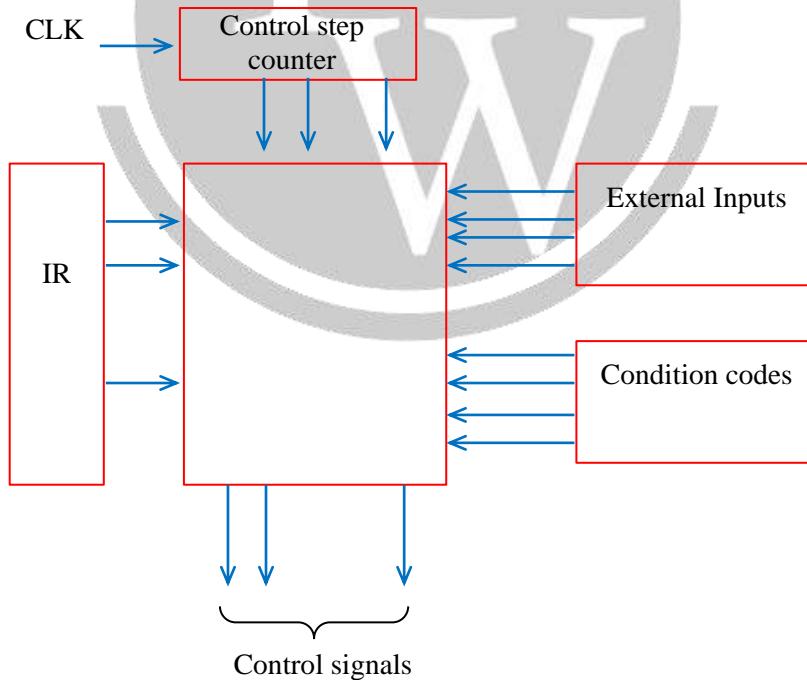
To execute instructions, the processor must have some means of generating the control signals needed in the proper sequence. The two approaches used for this purpose are

- (1) Hardwired control
- (2) Micro programmed control

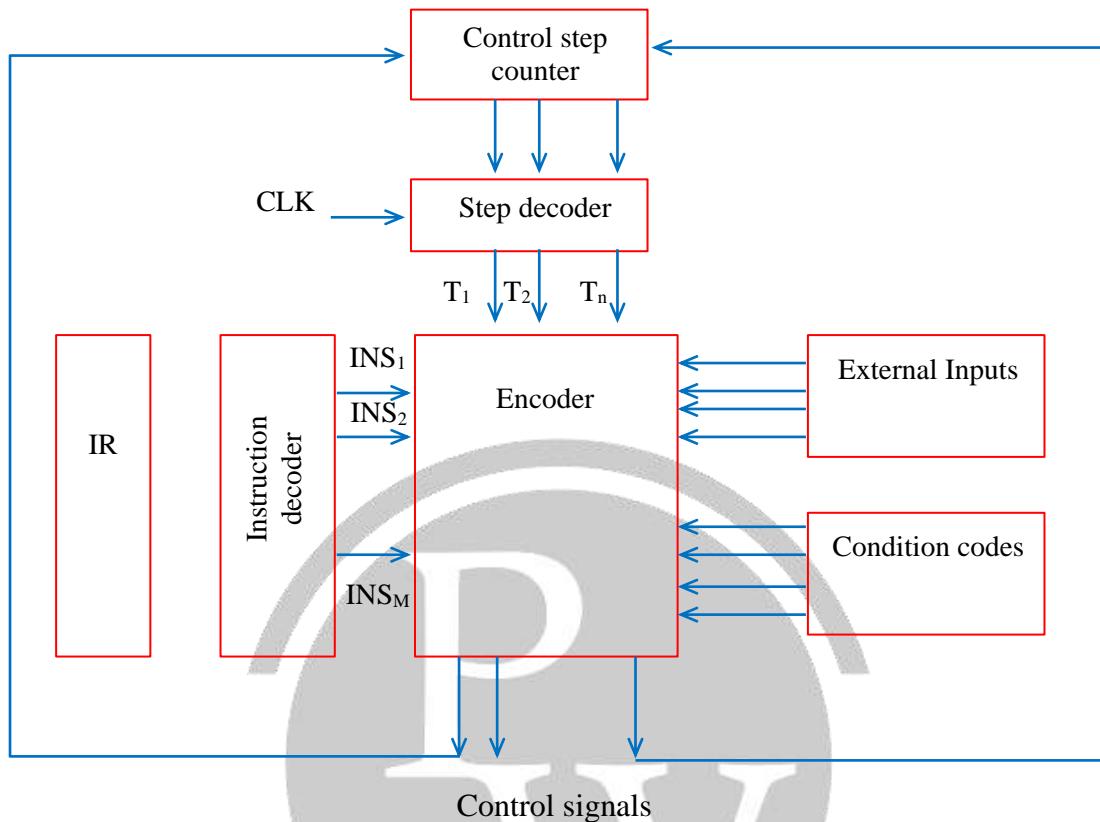
The purpose of control unit is to generate accurate timing & control signals.

4.1 Hardwired Control

- The control unit uses fixed logic circuits to interpret instructions and generate control signals from them. Every control signal is expressed as SOP (sum of products) expression and realized using digital hardware.



- The below figure shows the detailed hardwired control unit design.



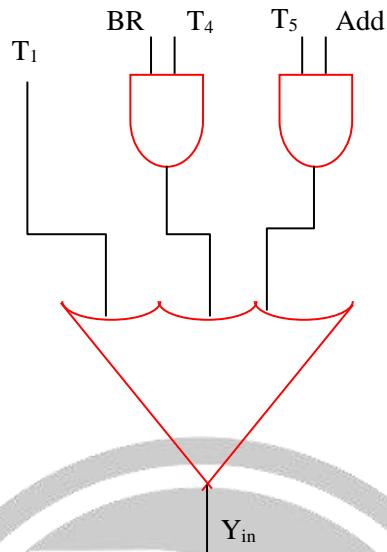
- For executing an instruction completely each step is completed in one clock period. A counter is used to keep track of the control steps.
- The required control signals are determined with the following information.
 - (1) Contents of control step counter.
 - (2) Contents of instruction register
 - (3) Contents of condition code flags
 - (4) External input signals, such as MFC and interrupt requests.
 - The instruction decoder decodes the instruction loaded in IR
 - The step decoder provides a separate signal line for each step or time slot in a control sequence.
 - The encoder gets input from instruction decoder step decoder, external inputs and condition codes, thus uses to generate the individual control signals.
 - After execution of each instruction, the end signal is generated which resets control step counter and makes it ready for generation of control step for next instruction.

For example:

- (1) The encoder circuit implements the following logic function to generate Y_{in} .
$$Y_{in} = T_1 + T_5 \text{ Add} + T_4 \text{ BR} + \dots$$

i.e. Here Y_{in} signal is asserted during time interval T_1 for all instructions, during T_5 for add instruction, during T_4 for branch instruction.

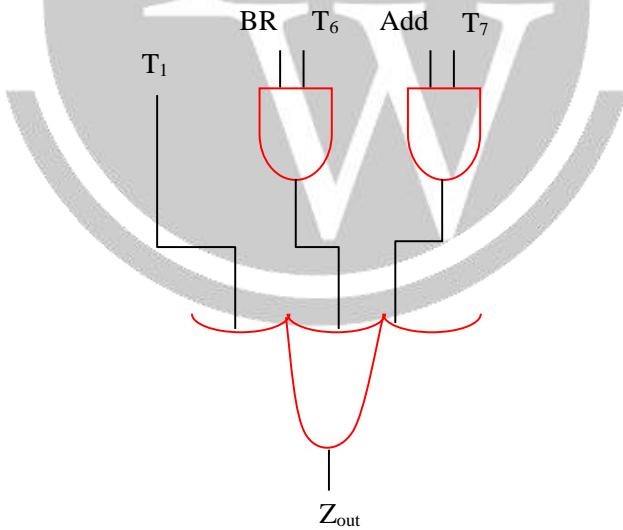
The generation of Y_{in} control signal is



- (2) The logic function to generate Z_{out} signal can be given by

$$Z_{out} = T_2 + T_7 \text{ add} + T_6 \text{ BR} + \dots$$

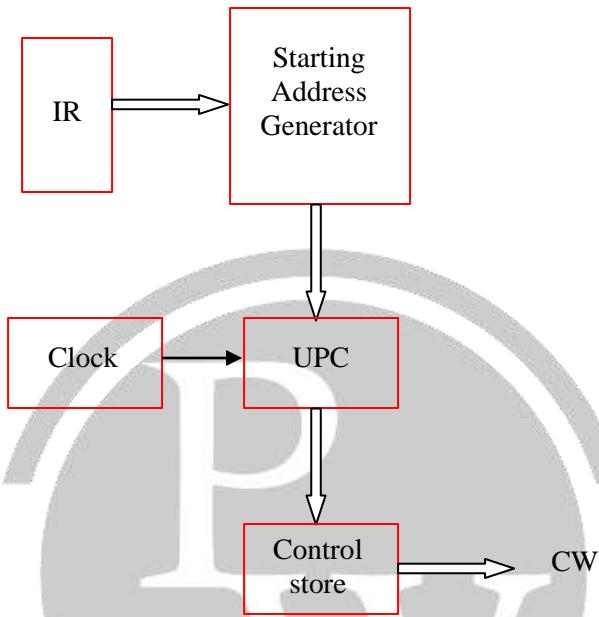
i.e., the Z_{out} signal is asserted during time interval T_2 for all instructions, during T_7 for add instruction, during T_6 for unconditional branch etc.



4.2 Microprogrammed Control

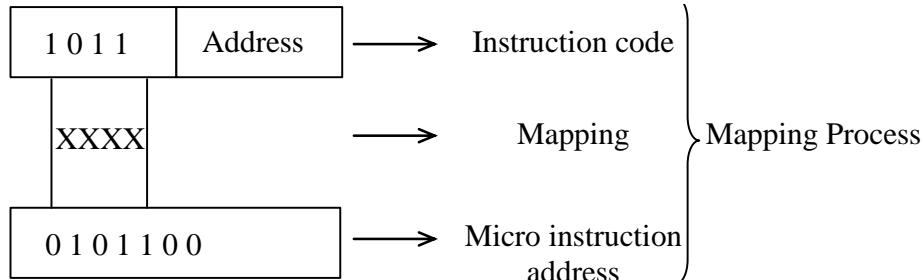
- Every instruction in a processor is implemented by a sequence of one or more sets of micro operations. Each micro operation is associated with a specific set of control lines which when activated, causes that micro operation to take place.
- Using micro programmed control, control signals are generated by a program. Using this the control signal selection and sequencing information is stored in ROM or RAM called control memory (CM).

- The control memory is part of control unit; it contains fixed programs (micro programs) that cannot be altered by occasional user.
- A control word (CW) is a word whose individual bits represent the various control signals.
- A sequence of CWs corresponding to the control sequence of a machine instruction constitutes the micro routine for that instruction, (micro program)
- The individual control words in the microprogram are referred as micro instruction.
- The basic organization of a microprogrammed control unit is

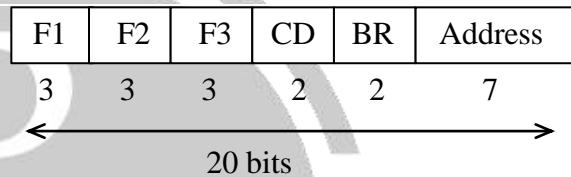


- To read the control words sequentially from the control memory a “micro program counter” (μ PC) is used.
-
- This diagram shows the detailed organization of the microprogrammed control unit. It includes an 'External I/P' block, a 'sequencer' block, a 'UPC/CAR' block, a 'Control memory' block, and a 'CDR' (Control Data Register) block. The 'External I/P' feeds into the 'sequencer'. The 'sequencer' has two outputs: one to the 'Next address Generator' and one to the 'UPC/CAR' block. The 'UPC/CAR' block contains a 'Control Address register (CAR)'. The 'UPC/CAR' also receives a signal from the 'Clock' and provides an address to the 'Control memory'. The 'Control memory' outputs a 'CW' to the 'CDR'. The 'CDR' also has a feedback connection to the 'UPC/CAR' block.
- The sequence of operations are:
 - (1) CAR holds the address of next micro instruction to be read.
 - (2) When address is available in UPC, the sequencer issues the READ command to CM.
 - (3) The word from addressed location is read into CDR/UIR.
 - (4) The UPC is incremented automatically by the clock, causing successive micro instructions to be read from CM.
 - (5) The contents of UIR generates control signals which are delivered to various parts of the processor in correct sequence.
 - The μ PC or CAR can be updated with various options using the address sequencer circuit as:

- (1) When a new instruction is loaded in IR, the μ PC is loaded with the starting address of the micro routine for that instruction (called mapping process).
- (2) When a branch instruction is (micro) encountered and branch condition satisfied, the μ PC is loaded with the branch address.
- (3) When END instruction is encountered, the μ PC is loaded with address of first word.
- (4) In any other situation, the μ PC is incremented every time a new micro instruction is fetched from CM.



- The transformation from the instruction code bits to an address in control memory where the routine is located is called Mapping process.
- The basic microinstruction format is
- The function fields (F1, F2, F3,), condition field (CD) & Branch field (BR) may be optional.



Comparison between Hardwired & Microprogrammed

	Hardwired	Micro programmed
1. Speed	Fast	Slow
2. Control functions	Implemented in Hardwire	Implemented in software
3. Ability to handle complex instructions	Complex	Easier
4. Design process	Complicated	Orderly and systematic
5. Instruction set size	Under 100	Over 100
6. ROM size	NIL	2k to 10k (20–400 bit micro instruction)
7. Applications	RISC processors	CISC, Main frames

4.2.1. The Micro Programmed Control Unit Can be

(1) Horizontal Microprogramming:

- One bit per control signal.
- Maximum parallelism i.e., more than one signal can be simultaneously.
- No extra decoders are required for decoding.
- The length of control word is large, need to access more than once from control memory.

(2) Vertical Microprogramming:

- The control signals are encoded in form for k-bits 2^k signals.
 - Maximum degree of parallelism is 1.
 - The length of micro instruction is small.
 - Response is relatively slower.
 - Requires a decoder additionally.
-
- To increase the degree of parallelism soft vertical microprogramming is used which divides the control signals into mutually exclusive groups. Each group is associated with associated number of control bits. (i.e., combination of both vertical and horizontal microprogramming).



5

MEMORY INTERFACING

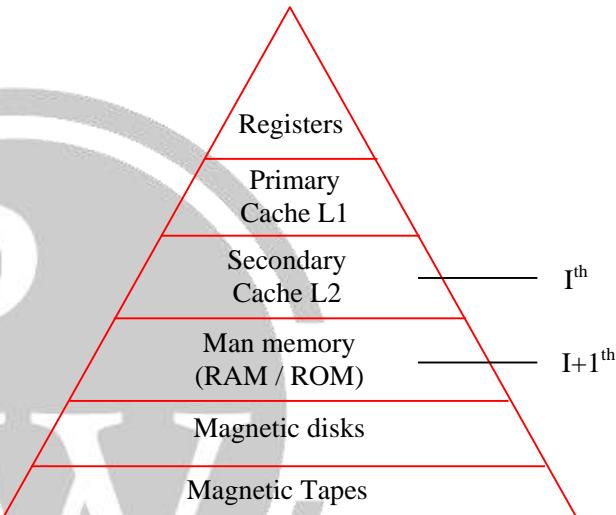
5.1 Memory Hierarchy

- The memory hierarchy system consists of all storage devices.

The purpose of memory hierarchy is to bridge the speed mismatch between the fastest processor to the slowest memory component at reasonable cost.

- In the structure of memory hierarchy I^{th} level memory is physically positioned higher than $(I + 1)^{th}$ level memory.
- Let $T_i, S_i, C_i, & F_i$ are the access time, size, cost per bit and frequency of references to the I^{th} level memory. Therefore

$$\begin{array}{lll} T_i & < & T_{i+1} \\ S_i & < & S_{i+1} \\ C_i & > & C_{i+1} \\ f_i & > & f_{i+1} \end{array}$$



- Since same data presents at different levels. I^{th} level memory is the subset of $I + 1^{th}$ level.
i.e. $I^i \subset I^{i+1}$

5.2 Memory Characteristics

Location:	Memory can be placed in 3 locations like registers, main memory, Auxiliary (or) secondary storage.
Capacity:	Word size, number of words i.e. capacity = number of words * word size.
Unit of transfer:	Maximum number of bits that can be read or written (blocks, bytes...)
Access method:	Random or sequential

Performance: Access time, memory cycle time, transfer rate, physical type.

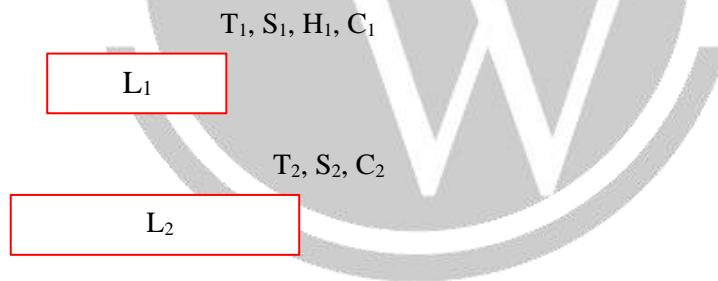
Physical: Volatile / non volatile
erasable / non erasable

	Serial Access		Random Access
(1)	Memory is organized into units of data called records/blocks accessed sequentially	(1)	Each storage location has an address uniquely
(2)	Access time depends on position of storage location	(2)	Access time is independent of storage location
(3)	Slower to Access	(3)	Faster to access
(4)	Cheaper	(4)	Costly relatively
(5)	Nonvolatile memories	(5)	May be relative or non
(6)	Example: Magnetic tapes	(6)	Example: Magnetic disks.

- When processor reads I^{th} level memory, if it is found in that level, his will occur otherwise it will be fault.

5.2.1. In a Two – Level Memory System

Case – I: When fault occurs, in L1 reads from L2 (Proxy Hierarchy)

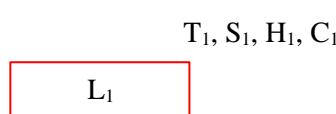


Case – II: When fault occurs in L1, must be brought from L2 to L1 memory. (**strict hierarchy**)

$$T_{\text{avg}} = H_1 * T_1 + (1-H_1) * (T_2 + T_1)$$

5.2.2. In a Three – Level Memory System

Case – I: When fault occurs in one level, then reads from its down level.



T_2, S_2, H_2, C_2

L_2

T_2, S_2, H_2, C_2

L_3

$$T_{avg} = H_1 * T_1 + (1 - H_1) * H_2 * T_2 + (1 - H_1) * (1 - H_2) * T_3$$

Case – II: When fault occurs, must access from L1.

$$T_{avg} = H_1 * T_1 + (1 - H_1) (H_2) (T_2 + T_1) + (1 - H_1) (1 - H_2) (T_3 + T_2 + T_1)$$

* Average cost per bit

$$C_{Avg} = \frac{C_1 S_1 + C_2 S_2}{S_1 + S_2} \quad (\text{Two - level})$$

$$C_{Avg} = \frac{C_1 S_1 + C_2 S_2 + C_3 S_3}{S_1 + S_2 + S_3} \quad (\text{Three - level})$$



6

INPUT OUTPUT INTERFACING

6.1 Input – Output Organization

- The input – output subsystem (I/O) provides an efficient mode of communication b/n system and outside environment. The Commonly used peripheral devices are. Keyboard, monitors, printer & magnetic tape, magnetic disc.....
- The input & output devices that communicates with computer & people usually with alphanumeric character from ASCII – 128-character set, 7 – bits are used represent each character. It contains 26 upper case letters, 26 lower case letters, 10 numerals, 32 special chars such as %, *, \$ In general ASCII chars are stored in a single unit called 1 byte (8-bits). The 8th bit may be used for parity bit for error detection.

6.1.1. Input – Output Interface

- The I/o interface provides a method of transferring information b/n internal storage (main memory) & external I/o devices. The devices/ peripherals connected to a computer need special communication links for interfacing with CPU because.
 - Peripherals are electromechanical & electromagnetic devices, whereas CPU & memory are electronic devices hence the operations are different.
 - The data transfer rate is shown then the transfer rate of CPU.
 - The data codes & formats are different.
 - The operating modes of peripherals is different and must be controlled.
- The four types of I/o command an interface will receive are

6.1.2. Control Command

Issued to activate the peripheral & and to inform it what to do. Depending on mode of operation a control command sequence is issued.

6.1.3. Status Command

Used to test various status conditions in the interface & the peripheral. Eg. Checking status of device, errors detected by interface. The status information is maintained in status register.

6.1.4. Data Output

Causes the interface to respond by transferring data from the bus into one of its registers buffer.

6.1.5. Data Input

With this command, interface receives an item of data from peripheral & places it in its buffer register. Then transfers to data bus of processor.

The I/o read & I/o write are enabled during I/o transfer, memory ready & memory write are enabled during memory transfer. The two configurations possible for communication are.

(a) Isolated I/o:

The CPU has distinct input & output instructions, each instruction is associated with the address of an interface register. When CPU fetches & decodes the opcode, it places address associated with the instruction into the common address lines, at the same time, it enables the I/o read or I/o write control line. An isolated I/o method isolates memory & I/o addresses each has its own address space, hence memory address values are not affected by interface address. Uses one common bus for memory & I/o, with common control lines.

(b) Memory – Mapped I/o:

In this configuration, only one set of read & write signals and do not distinguish b/n memory & I/o addresses and I/o.

6.2 Modes of Data Transfer

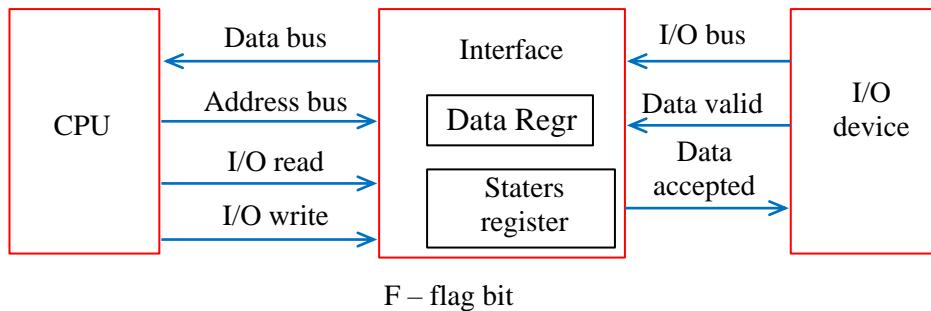
Data transfer b/n the central computer (main memory) and I/o device may be handled in a variety of modes like programmed I/o, Interrupt – Initiated I/o & Direct memory access.

6.2.1. Programmed I/O

- Programmed I/o operations are the result of I/o instruction written in computer program:

Example:

In programmed I/o, the I/o device doesn't have direct access to memory. A transfer from I/o device to memory requires the CPU to execute several instruction.



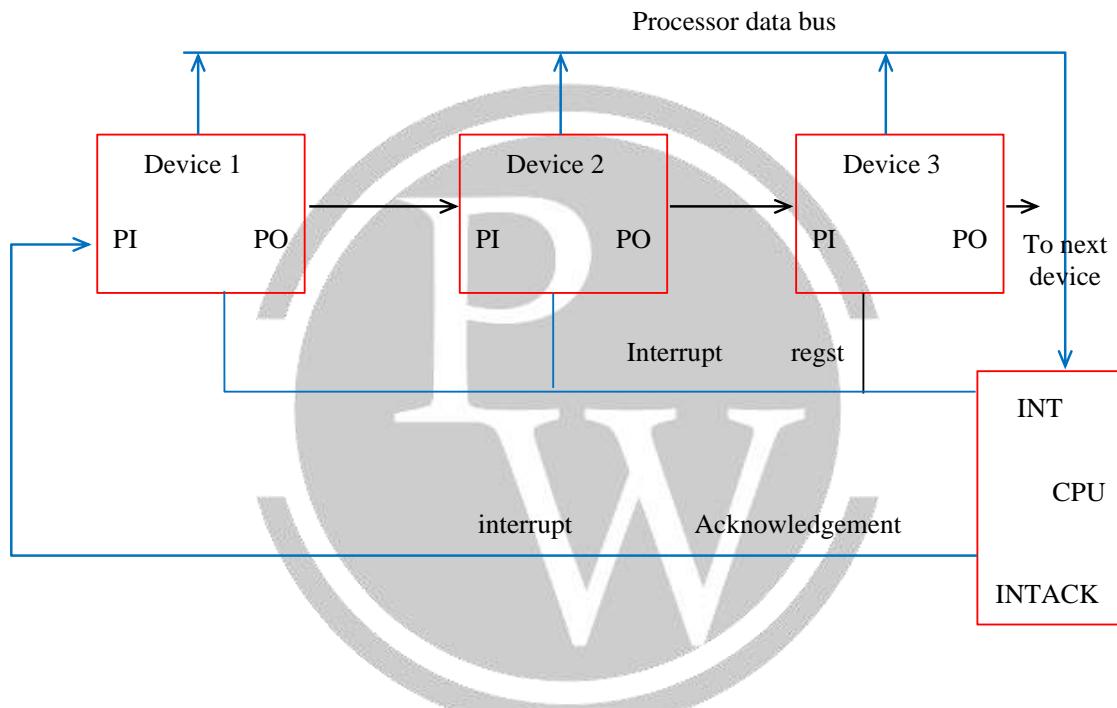
- (Data transfer from I/o device to CPU) The device transfers bytes of data one at a time as they are available. When a byte of data is available the device places it on I/o bus and enables its data valid line. The Interface accepts the byte into its data register and enables the data accepted line. The interface sets the Flag bit. Then the device disables the data valid line, but it will not transfer another byte until the data accepted line is disabled by the interface.

6.2.2. Interrupt – Initiated I/O

- In programmed I/O, the CPU stays in a program loop until the I/O unit indicates that it is ready to transfer. Hence this process keeps processor busy needlessly. meanwhile keeps monitoring the device. When the interface determining that the device is ready, it generates an interrupt request
- Priority Interrupt:** It is a system that establishes a priority over the various sources to determine which condition is to be serviced first when two or more requests arrive simultaneously. It also determines which conditions are permitted during processing of an

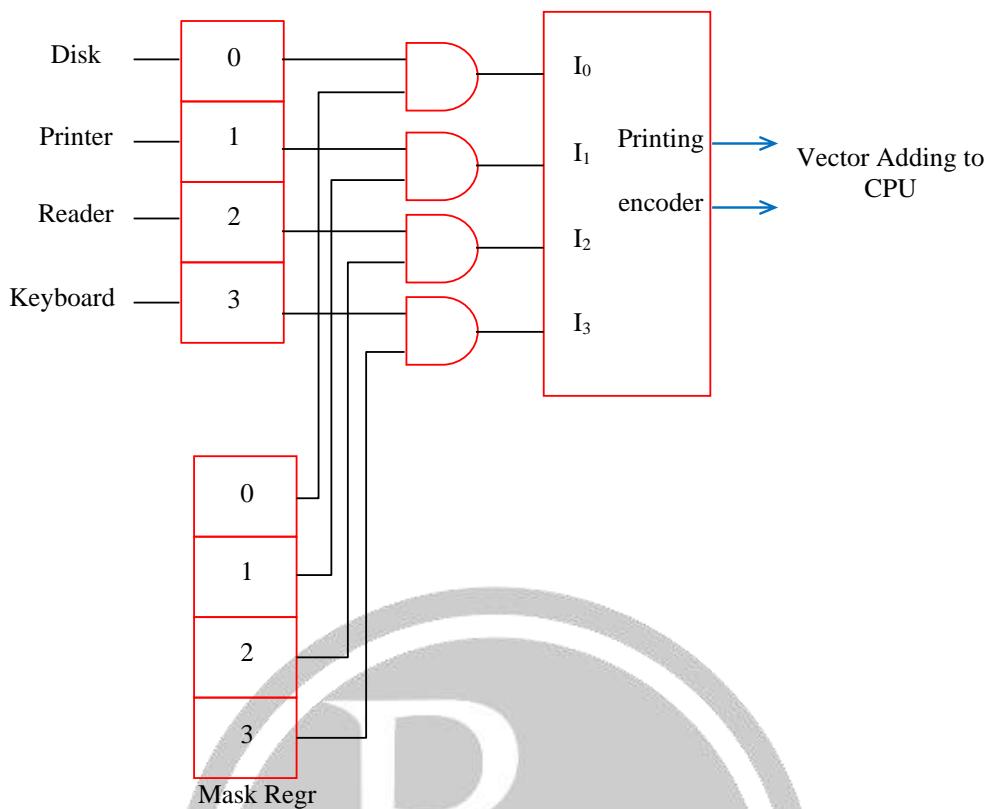
6.2.3. Daisy – Chaining Priority (Serial – Priority Interrupt)

The system consists of a serial connection of all devices that request an interrupt. The device with the highest priority is placed in first position followed by lower – priority devices. The lowest priority will be placed last in the chain.



6.2.4. Parallel Priority Interrupt

- This method uses a register whose bits are set separately by the interrupt signal from each device. Priority is established according to the position of bits in the register. The CKT will also include a mask register to control the starting of each interrupt request. The mask register can be programmed to disable low-priority interrupts while a high – priority device is being serviced. If will also allow a high-priority device to interrupt the CPU while low – priority device is being serviced.



6.2.5. Direct Memory Access (DMA)

- The speed of data transfer can be increased by removing CPU from the path and the peripheral device to manage the memory buses directly. This kind of transfer technique is called DMA transfer during DMA transfer the CPU is idle and has no control of the memory buses. The DMA controller takes control of buses to manage the transfer directly b/n i/o device & memory.
- To keep CPU idle to use bus, the “bus request (BR) input is used by the DMA controller to request the bus to relinquish control of the buses. When BR is active, CPU terminates current execution and places data, control & address lines in high impedance state. Then the bus behaves like an open circuit. The CPU then activates. “Bus grant” (BG) output to DMA, then DMA takes control of the bus to transfer without CPU intervention, when the transfer completes DMA disables the Bus request & CPU disables the bus grant. Then the CPU gets the control of the buses.

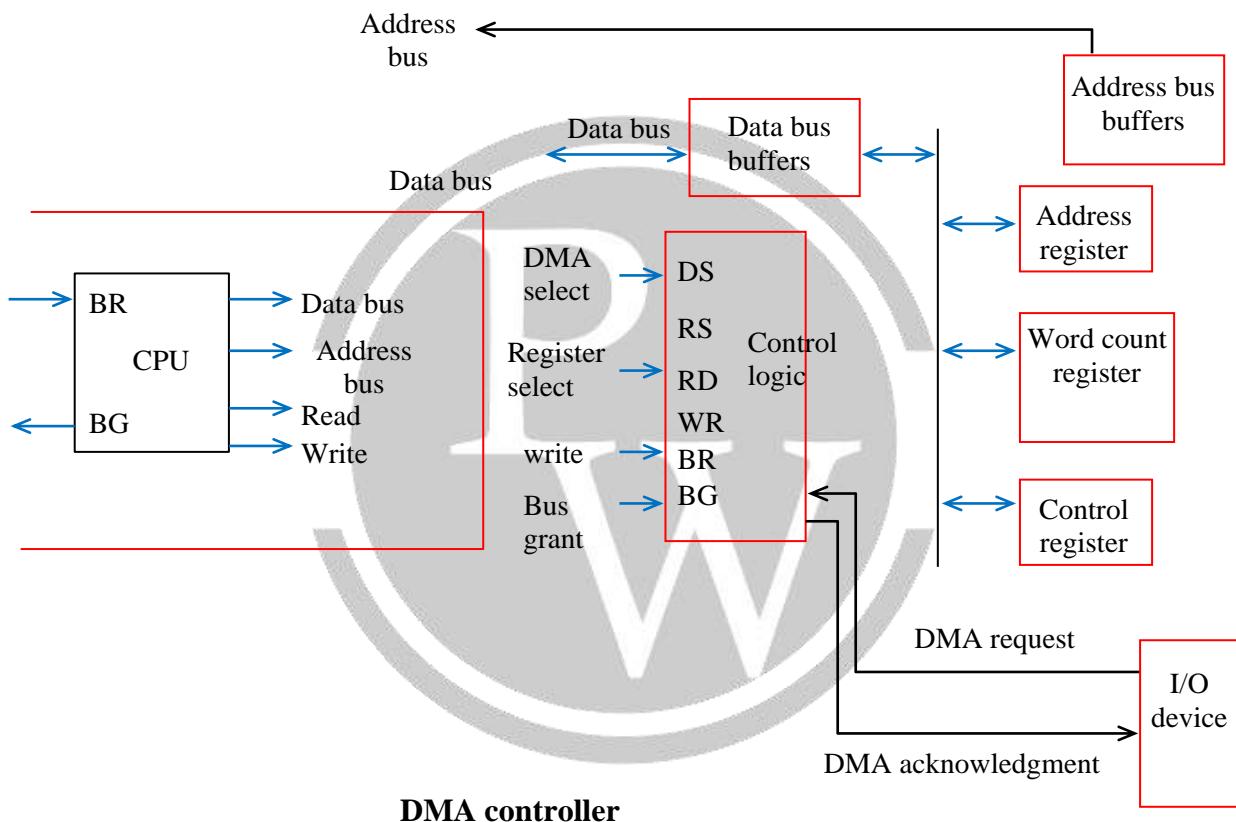
6.3. The DMA Transfer Take Place in Two Modes

(1) Burst transfer:

A block sequence consisting of a number of memory words is transferred in a continuous burst while DMA controller is master of the memory buses. Used to transfer fast devices like magnetic disks for large volumes. In this mode the data transfer will not be stopped, until an entire block is transferred.

(2) Cycle Stealing:

In this mode, DMA controller transfers one word at a time, after which it must return control of the buses to the CPU, later it will ‘steal’ memory cycle when CPU is idle.



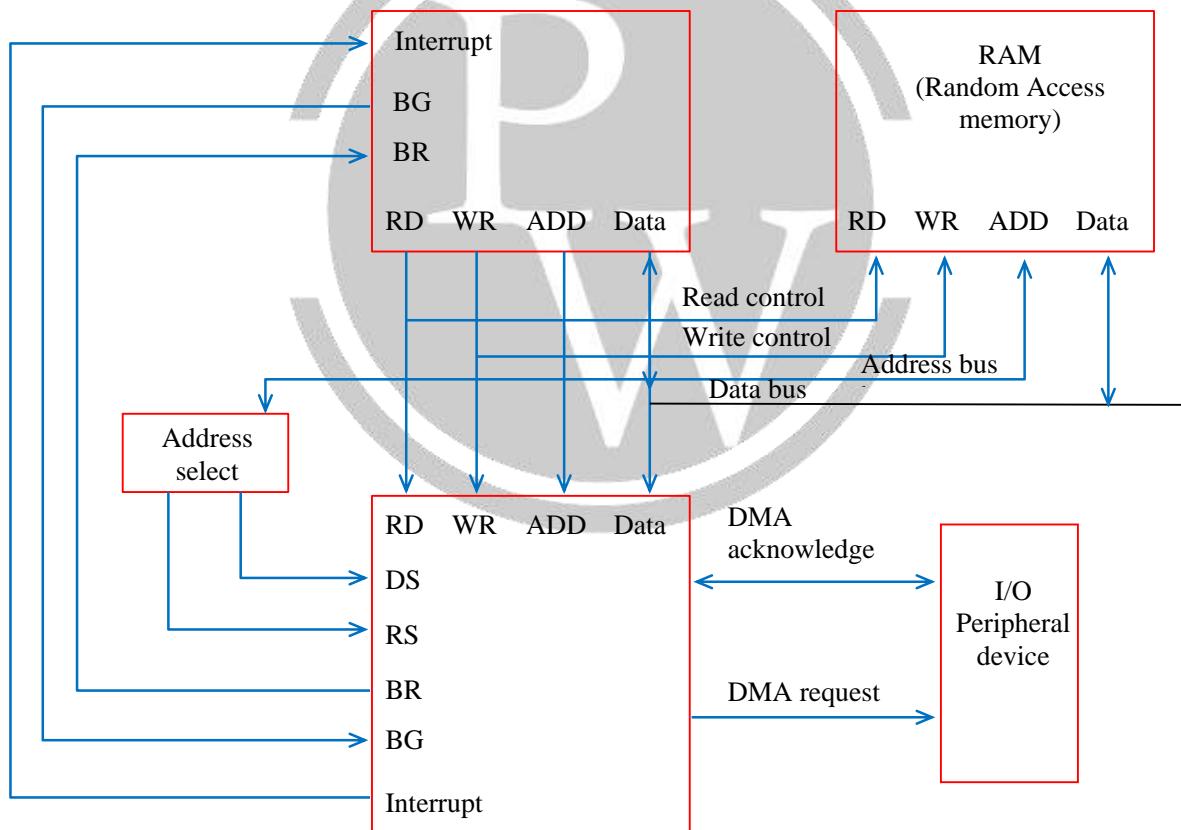
- The DMA controller communicates with the CPU the data bus and control lines. The register in DMA are selected by CPU through address bus by enabling DS & RS inputs. When BG = 1 (bus grant) the CPU has relinquished the buses and the DMA can communicate directly with the memory by specifying address in address bus & activating the RD or WR control. The DMA communicates with the external device through the request and acknowledge lines.

6.4. The CPU Initialize the DMA by Sending

- (1) The starting address of memory block where data are available (for read) or where data are to be stored (for write)
 - (2) Word count, the no of words in memory block.
 - (3) Control of specify mode of transfer such as read or write.
 - (4) A control to start the DMA transfer.
- Once DMA is initialized, the CPU stops communicating with the DMA unless it receives an interrupt signal.

6.5. DMA Transfer

When the peripheral device sends a DMA request, the DMA controller activates the BR line, informing the CPU to relinquish the buses. The CPU responds with its BG line, informing DMA that its buses are disabled. DMA then puts the current value of its address register into the adder bus and initiates RD or WR signal. It then sends DMA acknowledge to the peripheral device. When $BG = 0$, then CPU communicates with the internal DMA registers, when $BG = 1$ then RD & WR lines are used from DMA controller to RAM to specify read or write operation.



When the device receives DMA acknowledge, it puts a word in the data bus (write) or receives one word from the data bus (read). In this way the peripheral communicates with memory without any involvement of CPU. For each word transfer DMA increments the address register and decrements its word count register. When count reaches to zero, the DMA disables bus request line so that the CPU can continue to execute its program. DMA transfer is very useful for fast transfer of data.



7

PIPELINING

7.1 Introduction

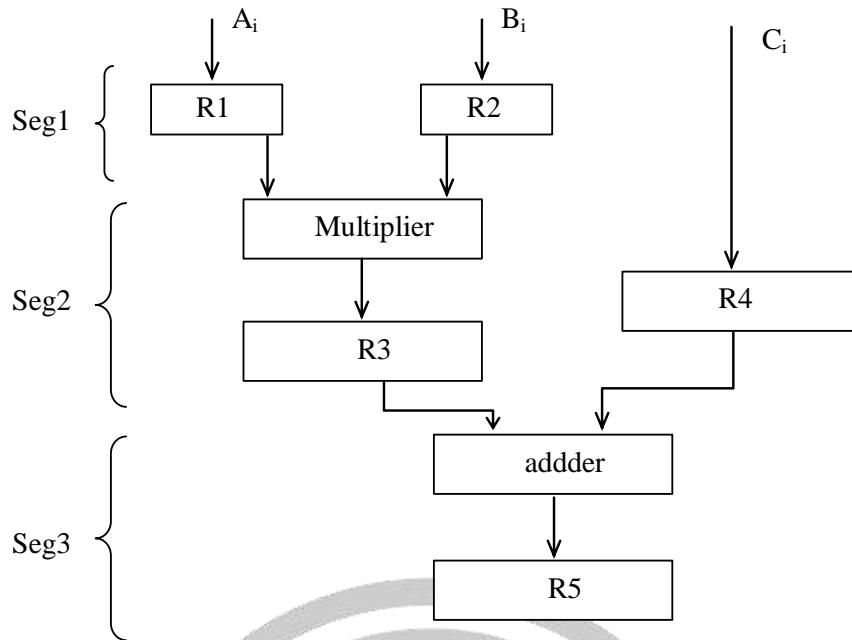
- “Parallel processing” denotes a large class of techniques that are used to provide simultaneous data-processing tasks for the purpose of increasing the computational speed of a computer system.
- The purpose of parallel processing is to speed up the computer processing capability and increases its throughput. “Through put” is the amount of processing that can be accomplished during a given interval of time”.
- Parallel processing can be viewed from various levels as
 - (i) At registers level, Registers with parallel load operate with all the bits of the word simultaneously. (Instead of shift registers). This is at lowest level.
 - (ii) Higher level of complexity can be achieved by having a multiplicity of functional units that perform identical or different operations simultaneously.
 - (iii) By distributing the data among the functional units in multiple. Eg: The arithmetic logical & shift operations can be separated.
 - (iv) Using multiple processors. (Flynn’s classification) (SISD, SIMD, MISD, MIMD).
 - (v) Using pipelining in unit processor systems.

7.2 Pipelining

- Pipelining is a technique of decomposing a sequential process into sub operations, with each sub process being executed in a special dedicated segment that operates concurrently with all other segments.
- The result obtained from one segment is transferred to the next segment in the pipeline. The final result is obtained after the data have passed through all segments.

Example:

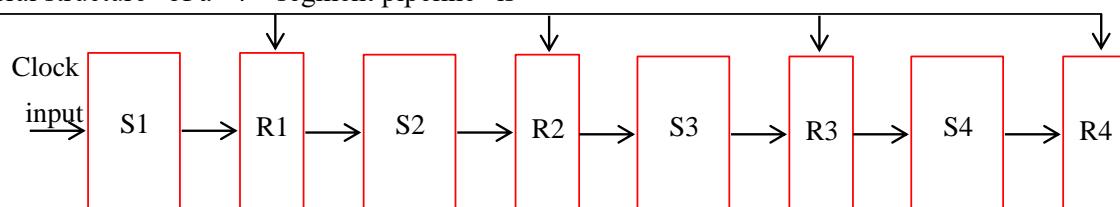
The perform $A_i * B_i + C_i$, for $i = 1$ to 6. Each sub operation multiply & add implemented in a segment with in a pipeline. Each segment has one or more registers.



- R1 through R5 are registers that receive new data with every clock pulse.
- The effect of each clock pulse can be shown as

Clock pulse Number	Segment - 1		Segment - 2		Segment - 3
	R1	R2	R3	R4	R5
1	A1	B1	---	---	---
2	A2	B2	A ₁ * B ₁	C1	---
3	A3	B3	A ₂ * B ₂	C2	A ₁ * B ₁ + C ₁
4	A4	B4	A ₃ * B ₃	C3	A ₂ * B ₂ + C ₂
5	A5	B5	A ₄ * B ₄	C4	A ₃ * B ₃ + C ₃
6	A6	B6	A ₅ * B ₅	C5	A ₄ * B ₄ + C ₄
7	---	---	A ₆ * B ₆	C6	A ₅ * B ₅ + C ₅
8	---	---	---	---	A ₆ * B ₆ + C ₆

- The “General structure” of a “4 – segment pipeline” is



- The operands pass through all four segments in a fixed sequence.
- Each segment consists of combinational circuit S_i that performs a sub operation.
- The segments are separated by registers R_i that holds the intermediate results between stages.
- Information flows between adjacent stages under the control of common clock applied to all registers simultaneously.
- A task as the total operation performed going through all the segments in the pipeline.
- The behaviour of a pipeline can be illustrated with “Space – time diagram”. It shows the segment utilization as a function of time. The following diagram is for tasks T1 to T6 executed in 4 – segment pipeline.

Segments	1	2	3	4	5	6	7	8	9	Clock cycles
S – 1	T1	T2	T3	T4	T5	T6				
S – 2		T1	T2	T3	T4	T5	T6			
S – 3			T1	T2	T3	T4	T5	T6		
S – 4				T1	T2	T3	T4	T5	T6	

- “An Arithmetic pipeline divides an arithmetic operation into sub operations for execution in the pipeline segments.”

7.2.1 Pipeline performance evaluation

- Consider a K – segment pipeline with a clock cycle time t_p is used to execute n tasks.
 - To complete n – tasks using a K – segment pipeline requires $K + (n-1)$ clock cycles.
- (1) The number of clock cycles needed in a pipeline to execute 100 tasks in 6 segments.

Ans. $K = 6$

$$n = 100$$

$$\Rightarrow 6 + (100-1) = 105 \text{ clocks}$$

- The processing time in each stage is called as “stage delay”, In a pipeline. (t_p).
- The time delay due to interstage transfer of data is “interstage delay” (t_d)
- The interstage delays can be same between the stages, stage delay very from stage to stage based on segment operation.

The time period for clock cycle, ‘ t_p ’ is

$$t_p = \max \{ t_i \} + t_d$$

$$t_p = t_m + t_d$$

Since $t_m > > t_d$, maximum stage delay denotes the clock period.

$$\text{i.e. } t_p = t_m$$

- The total time required in pipeline execution is

$$T_p = [K + (n-1)] * t_p.$$

7.2.2 Speed up Ratio

- The speed up of a K – stage pipeline over an equivalent non-pipelined processor is defined as

$$S = \frac{\text{time without pipeline}}{\text{time with pipeline}}$$

- Consider a non-pipeline processor that performs the same operation as pipelined and takes a time equal to “ t_n ” to complete each task.

$$S = \frac{n * t_n}{(K + (n - 1)) * t_p}$$

- As the number of tasks increases, $K + n - 1$ approaches to n

Since $t_n \approx K * t_p$

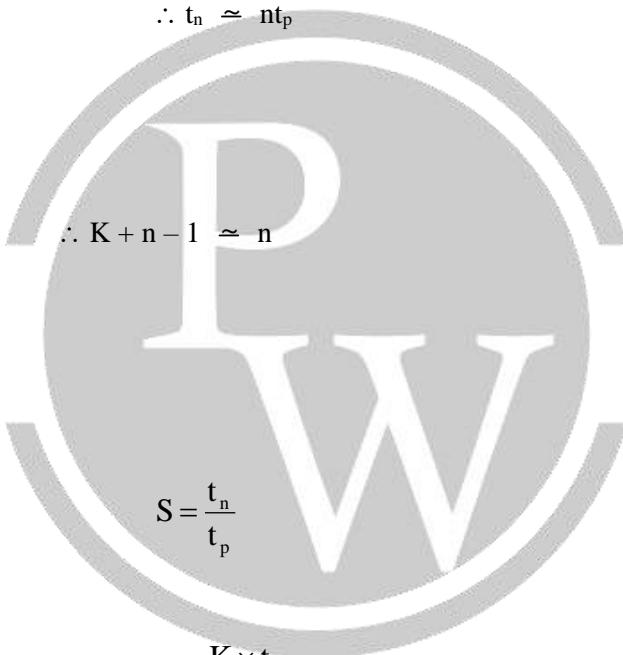
$$S = \frac{n * K * t_p}{(K + (n - 1)) * t_p}$$

$$\therefore t_n \approx n t_p$$

$$S = \frac{nK}{K + (n - 1)}$$

$$S = \frac{n t_n}{n * t_p}$$

$S = \frac{t_n}{t_p}$



For large number of tasks

$$S = \frac{nk}{K + n - 1}$$

Or

$$= \frac{nK}{n}$$

$$S = \frac{K * t_p}{t_p}$$

$S = K$

$S = K$

- The maximum speed up that can be achieved in a pipelined processor is equal to “number of stages”. [as n is large, $K + n - 1$ is K].

$$S_{\text{ideal}} = S_{\text{max}} = K$$

7.2.3 Efficiency

- The efficiency of a pipeline is defined as the ratio of speed up factor and the number of stages in the pipeline.

$$E_k = \frac{S}{K} = \left(\frac{nk}{K + (n - 1)} \right) / K$$

$$E_K = \frac{n}{K+n-1}$$

$$E_K = \frac{S}{K}$$

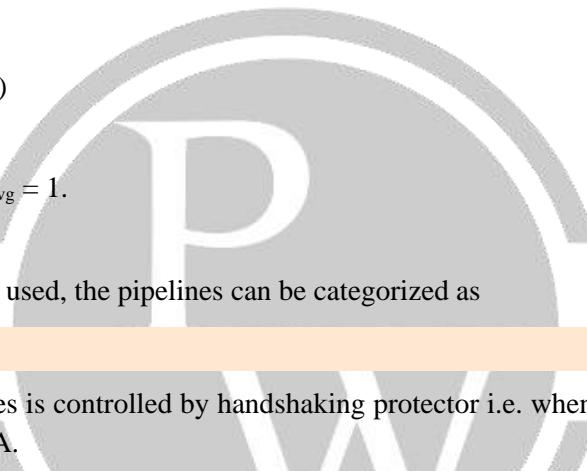
7.2.4 Throughput

- It is the number of tasks that can be completed by a pipeline per unit time.

$$H_K = \frac{n}{(K+(n-1)) \times t_p}$$

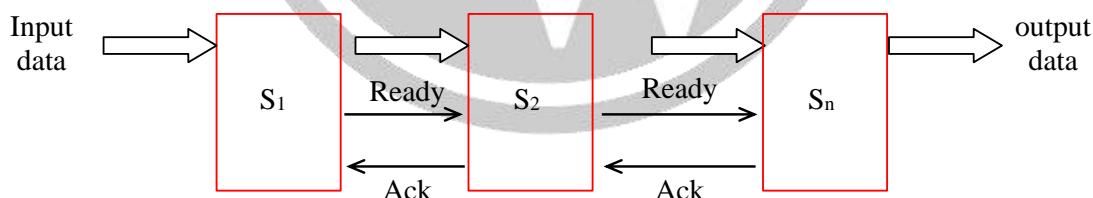
7.2.5 Stall cycle

- The performance of a pipeline is influenced by
 - Number of instructions
 - Uneven stage delays
 - Buffer overhead (Interstage delay)
 - Dependencies.
- The objective of pipelines is $CPI_{avg} = 1$.
(i.e. Clocks per instruction – CPI)
- Depending on control mechanism used, the pipelines can be categorized as



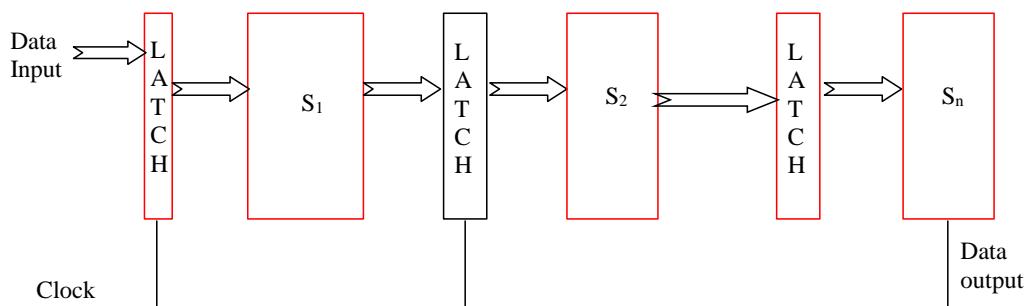
7.2.6 I Asynchronous pipeline

- Data flow along the pipeline stages is controlled by handshaking protocol i.e. when S_i is ready to send/transmit, it sends ready signal to S_{i+1} and S_{i+1} sends Ack to S_i after receiving.



7.2.7 II Synchronous pipeline

- Clocked high speed latches are used to interface between stages. At the falling edge of the clock pulse, all latches transfer data to the next stages simultaneously.



7.2.8 Instruction pipelining

- An instruction pipeline operates on a stream of instructions by overlapping the Fetch, Decode, Execute and other phases of instruction cycle.
- The instruction cycle with 4 – segments is

Instructions	1	2	3	4	5	6	7	8	9	Clock
I 1	FI	DA	FO	FX						
I 2		FI	DA	FO	EX					
I 3			FI	DA	FO	EX				
I 4				FI	DA	FO	EX			
I 5					FI	DA	FO	EX		

Here FI - fetches an instruction

DA - Decodes the instruction & calculates effective address

FO - fetches the operand

EX - Executes the instruction

- "In general, each stage in a pipeline is expected to complete its operation in one clock cycle, hence the clock period must allow the longest task to be completed."
 - "The performance of a pipeline is high if different stages require about same amount of time."
 - The use of cache solves the memory access problem.
- (5) Consider a 4–stage pipeline, where different instructions require different number of clock cycles, at different stages. The total number of clocks required for execution of 4 instructions is _____

Ans.:

	1	2	3	4	
I 1	2	1	2	2	7
I 2	1	3	3	2	9
I 3	2	2	2	2	8
I 4	1	3	1	1	6
					30

Through sequential process 30 clocks. But using pipeline the number of clock cycles required is 14. That is,

	1	2	3	4	5	6	7	8	9	10	11	12	13	14
I1	S1	S1	S2	S3	S3	S4	S4							
I2			S1	S2	S2	S2	S3	S3	S3	S4	S4			
I3				S1	S1	-	S2	S2	-	S3	S3	S4	S4	
I4						S1	-	-	S2	S2	S2	S3	-	S4

7.2.9 Data Hazards

- A data hazard is situation in which the pipeline is stalled because the data to be operated on are delayed for some reason, as illustrated in Figure. We will now examine the issue of availability of data in some detail.
- Consider a program that contains two instructions, I_1 followed by I_2 . When this program is executed in a pipeline, the execution of I_2 can begin before the execution of I_1 is completed. This means that the results generated by I_1 may not be available for use by I_2 . We must ensure that the results obtained when instructions are executed in a pipelined processor are identical to those obtained when the same instructions are executed sequentially. The potential for obtaining incorrect results when operations are performed concurrently can be demonstrated by a simple example. Assume that $A = 5$, and consider the following two operations:

$$A \leftarrow 3 + A$$

$$B \leftarrow 4 \times A$$

- When these operations are performed in the order given, the result is $B = 32$. But if they are performed concurrently, the value of A used in computing B would be the original value, 5, leading to an incorrect result. If these two operations are performed by instructions in a program, then the instructions must be executed one after the other, because the data used in the second instruction depend on the result of the first instruction. On the other hand, the two operations.

$$A \leftarrow 5 \times C$$

$$B \leftarrow 20 + C$$

- Can be performed concurrently, because these operations are independent.

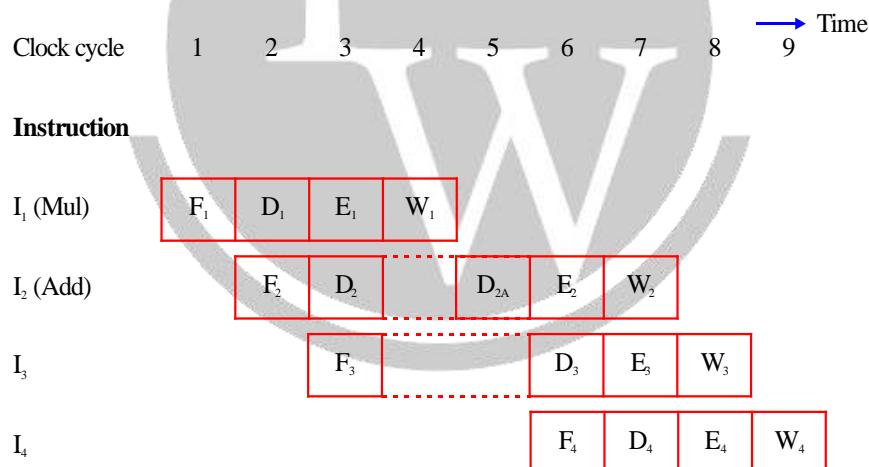


Fig: Pipeline stalled by data dependency between D₂ and W₁

- This example illustrates a basic constraint that must be enforced to guarantee correct results. When two operations depend on each other, they must be performed sequentially in the correct order. This rather obvious condition has far-reaching consequences. Understanding its implications is the key to understanding the variety of design alternatives and trade-offs encountered in pipelined computers.
- Consider the pipeline in Figure. The data dependency just described arises when the destination of one instruction is used as a source in the next instruction. For example, the two instructions

Mul R2.R3.R4

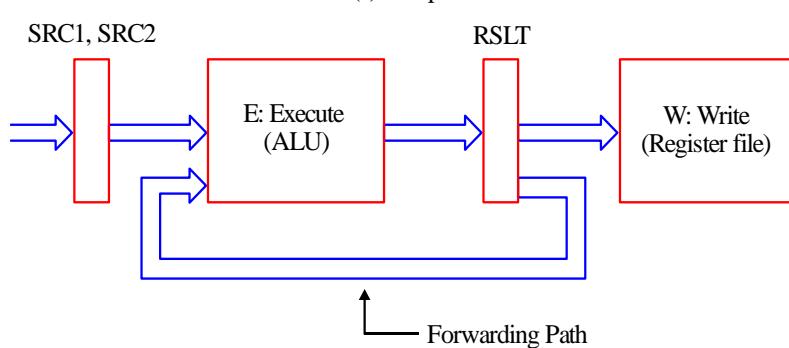
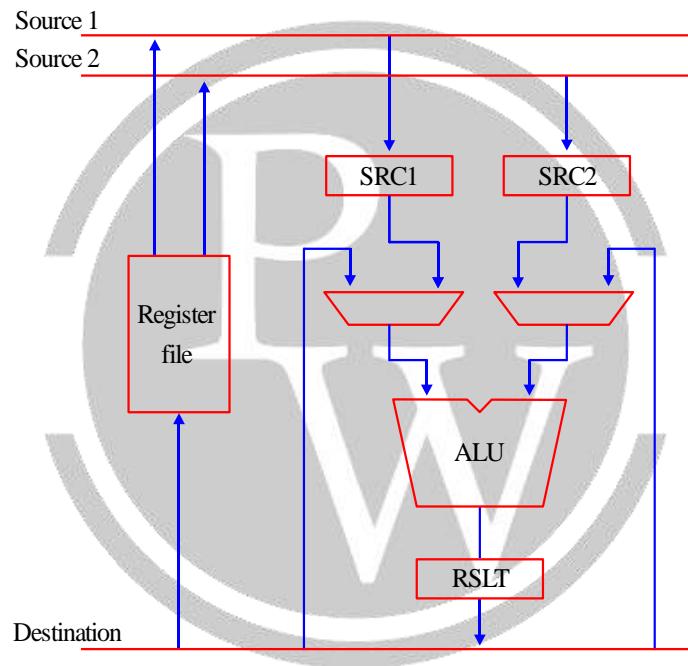
Mul R5.R4.R6

- Give rise to a data dependency. The result of the multiply instruction is placed into register R4, which in turn is one of the two source operands of the Add instruction. Assuming that the multiply operation takes one clock cycle to complete; execution would proceed as shown in Figure. As the Decode unit decodes the Add instruction in cycle 3, it realizes that R4 is used as a source operand. Hence, the D step of that instruction cannot be completed until the W step of multiply instruction has been completed. Completion of step D₂ must be delayed to clock cycle 5, and is shown as step D_{2A} in figure. Instruction I₃ is fetched in cycle 3, but its decoding must be delayed because step D₃ cannot precede D₂. Hence, pipelined execution is stalled for two cycles.

7.2.10 Operand Forwarding

- The data hazard just described arises because one instruction, instruction I₂ in Figure, is waiting for data to be written in the register file. However, these data are available at the output of the ALU once the Execute stage completes step E₁. Hence, the delay can be reduced, or possibly eliminated, if we arrange for the result of instruction I₁ to be forwarded directly for use in step E₂.

Figure shows a part of the processor data path involving the ALU and the register file.



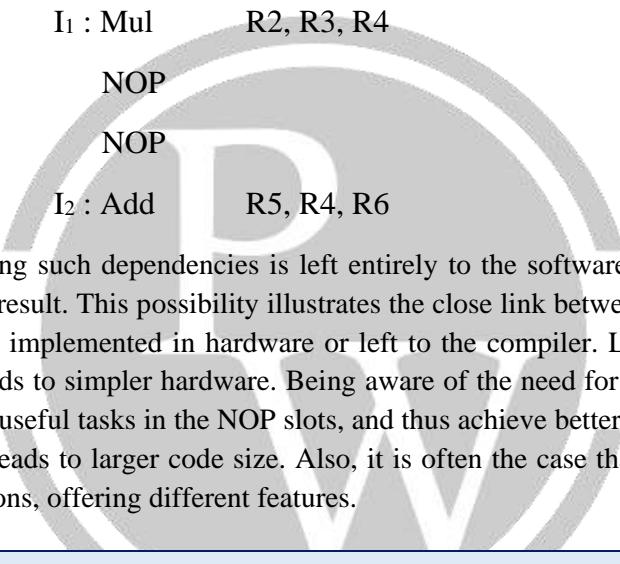
- Interstage buffers needed for pipelined operation, as illustrated in Figure. With reference to Figure, registers SRC1 and SRC2 are part of buffer B2 and RSLT is part of B3. The data forwarding mechanism is provided by the blue connection

lines. The two multiplexes connected at the inputs to the ALU allow the data on the destination bus to be selected instead of the contents of either the SRC1 or SRC2 register.

- When the instructions in Figure 8.6 are executed in the data path of Figure, the operations performed in each clock cycle are as follows. After decoding instruction I_2 and detecting the data dependency, a decision is made to use data forwarding. The operand not involved in the dependency, register R2, is read and loaded in register SRC1 in clock cycle 3. In the next clock cycle, the product produced by instruction I_1 is available in register RSLT, and because of the forwarding connection, it can be used in step E₂. Hence, execution of I_2 proceeds without interruption.

7.2.11 Handling Data Hazards in Software

- In Figure, we assumed the data dependency is discovered by the hardware while the instruction is being decoded. The control hardware delays reading register R4 unit cycle 5, thus introducing a 2-cycle stall unless operand forwarding is used. An alternative approach is to leave the task of detecting data dependence and dealing with them to the software. In this case, the compiler can introduce the two-cycle delay needed between instructions I_1 and I_2 by inserting NOP (No-operation) instructions, as follows:



- If the responsibility for detecting such dependencies is left entirely to the software, the compiler must insert the NOP instructions to obtain a correct result. This possibility illustrates the close link between the compiler and the hardware. A particular feature can be either implemented in hardware or left to the compiler. Leaving tasks such as inserting NOP instructions to the compiler leads to simpler hardware. Being aware of the need for a delay, the compiler can attempt to reorder instructions to perform useful tasks in the NOP slots, and thus achieve better performance. On the other hand, the insertion of NOP instructions leads to larger code size. Also, it is often the case that a given processor architecture has several hardware implementations, offering different features.

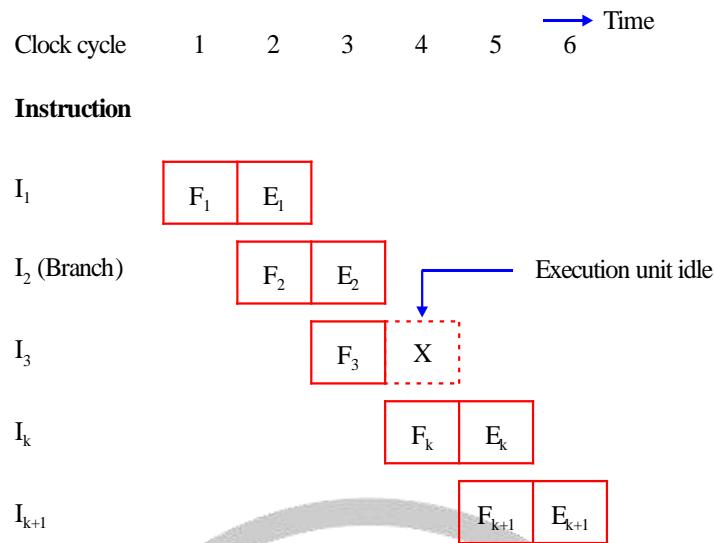
7.3 INSTRUCTION HAZARDS

- The purpose of the instruction fetch unit is to supply the execution units with a steady stream of instructions. Whenever this stream is interrupted, the pipeline stalls, as Figure illustrates for the case of cache miss. A branch instruction may also cause the pipeline to stall. We will now examine the effect of branch instructions and the techniques that can be used for mitigating their impact. We start with unconditional branches.

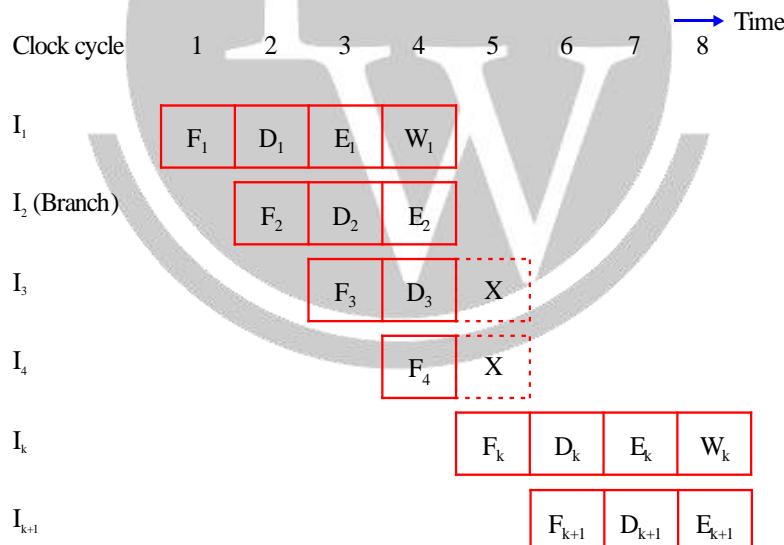
7.4 UNCONDITIONAL BRANCHES

- Figure shows a sequence of instructions being executed in a two-stage pipeline. Instructions I₁ to I₃ are stored at successive memory addresses, and I₂ is a branch instruction. Let the branch target be instruction I_h. In clock cycle 3, the fetch operation for instruction I₃ is in progress at the same time that the branch instruction is being decoded and the target address computed. In clock cycle 4, the processor must discard I₃, which has been incorrectly fetched, and fetch instruction I_h. In the meantime, the hardware unit responsible for the Execute (E) step must be told to do nothing during that clock period. Thus, the pipeline is stalled for one clock cycle.
- The time lost as a result of a branch instruction is often referred to as the *branch penalty*. In Figure, the branch penalty is one clock cycle. For a longer pipeline, the branch penalty may be higher. For example, Figure shows the effect of a branch

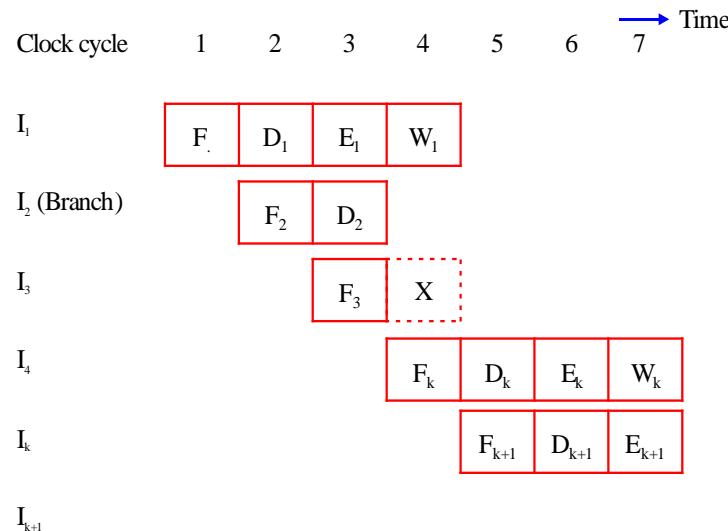
instruction on a four-stage pipeline. We have assumed that the branch address is computed in step E₂. Instructions I₃ and I₄ must be discarded, and the target instruction, I_h, is fetched in clock cycle 5. Thus, the branch penalty is two clock cycles.



- Reducing the branch penalty requires the branch address to be computed earlier in the pipeline. Typically, the instruction fetch unit has dedicated hardware to identify a branch instruction and compute the branch target address as quickly as possible after an instruction is fetched. With this additional hardware, both of these tasks can be performed in step D₂, leading to the sequence of events shown in Figure. In this case, the branch penalty is only one clock cycle.



(a) Branch address computed in Execute stage



(b) Branch address computed in Decode stage

7.4.1 Instruction Queue and Prefetching

- Either a cache miss or a branch instruction stalls the pipeline for one or more clock cycles. To reduce the effect of these interruptions, many processors employ sophisticated fetch units that can fetch instructions before they are needed and put them in a queue. Typically, the instruction queue can store several instructions. A separate unit, which we call the *dispatch unit*, takes instructions from the front of the queue and

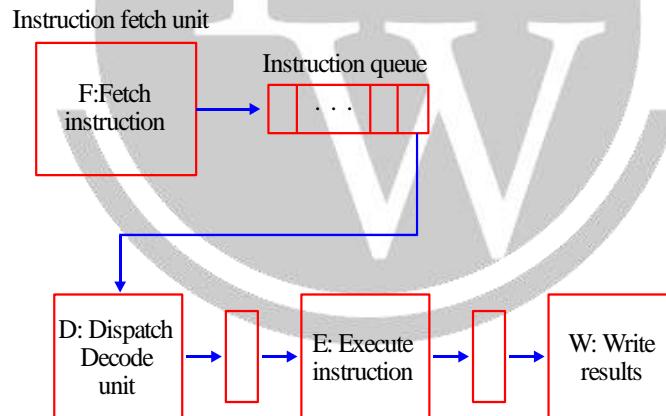


Fig: Use of an instruction queue in the hardware organization of fig (b)

- Sends them to the execution unit. This leads to the organization shown in Figure 8.10. The dispatch unit also performs the decoding function.
- To be effective, the fetch unit must have sufficient decoding and processing capability to recognize and execute branch instructions. It attempts to keep the instruction queue filled at all times to reduce the impact of occasional delays when fetching instructions. When the pipeline stalls because of a data hazard, for example, the dispatch unit is not able to issue instructions from the instruction queue. However, the fetch unit continues to fetch instructions and add them to the queue. Conversely, if there is a delay in fetching instructions because of a branch or a cache miss, the dispatch unit continues to issue instructions from the instruction queue.

7.5 CONDITIONAL BRANCHES AND BRANCH PREDICTION

- A conditional branch instruction introduces the added hazard caused by the dependency of the branch condition on the result of a preceding instruction. The decision to branch cannot be made until the execution of that instruction has been completed.
- Branch instructions occur frequently. In fact, they represent about 20 percent of the dynamic instruction count of most programs. (The dynamic count is the number of instruction executions, taking into account the fact that some program instructions are executed many times because of loops.) Because of the branch penalty, this large percentage would reduce the gain in performance expected from pipelining. Fortunately, branch instructions can be handled in several ways to reduce their negative impact on the rate of execution of instructions.

7.5.1 Delayed Branch

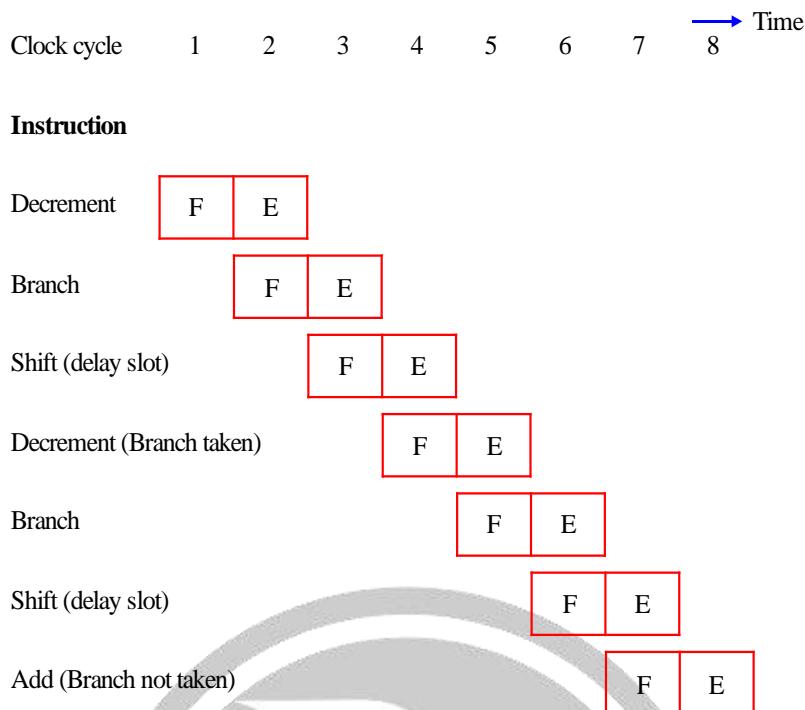
- In Figure the processor fetches instruction I_3 before it determines whether the current instruction, I_2 , is a branch instruction. When execution of I_2 is completed and a branch is to be made, the processor must discard I_3 and fetch the instruction at the branch target. The location following a branch instruction is called a *branch delay slot*. There may be more than one branch delay slot, depending on the time it takes to execute a branch instruction. For example, there are two branch delay slots in Figure and one delay slot in Figure. The instructions in the delay slots are always fetched and atleast partially executed before the branch decision is made and the branch target address is computed.
- A technique called *delayed branching* can minimize the penalty incurred as a result of conditional branch instructions. The idea is simple. The instructions in the delay slots are always fetched. Therefore, we would like to arrange for them to be fully executed whether or not the branch is taken. The objective is to be able to place useful instructions in these slots. If no useful instructions can be placed in the delay slots, these slots must be filled with NOP instructions.

LOOP	Shift left	R1
	Decrement	R2
	Branch=0	LOOP
NEXT	Add	R1.R3

(a) Original program loop

LOOP	Decrement	R2
	Branch=0	LOOP
	Shift left	R1
NEXT	Add	R1.R3

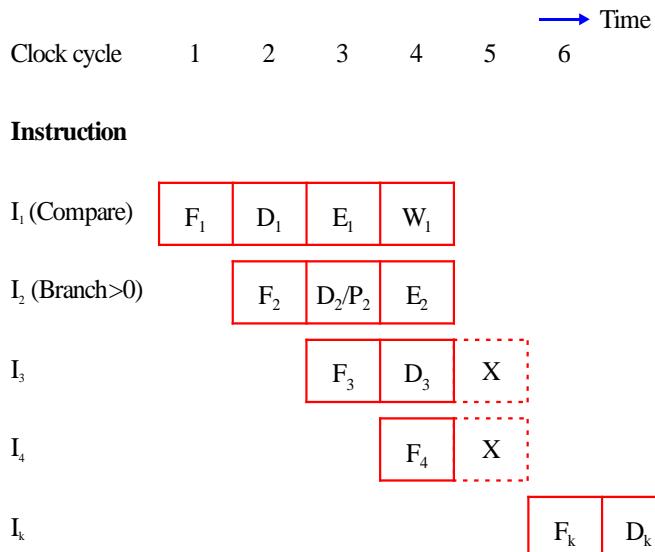
(b) Reordered instructions



- The effectiveness of the delayed branch approach depends on how often it is possible to reorder instructions as in Figure. Experimental data collected from many programs indicate that sophisticated compilation techniques can use one branch delay slot in as many as 85 percent of the cases. For a processor with two branch delay slots, the compiler attempts to find two instructions preceding the branch instruction that it can move into the delay slots without introducing a logical error. The chances of finding two such instructions are considerably less than the chances of finding one. Thus, if increasing the number of pipeline stages involves an increase in the number of branch delay slots, the potential gain in performance may not be fully realized.

7.5.2 Branch Prediction

- Another technique for reducing the branch penalty associated with conditional branches is to attempt to predict whether or not a particular branch will be taken. The simplest form of branch prediction is to assume that the branch will not take place and to continue to fetch instructions in sequential address order.
- Until the branch condition is evaluated, instruction execution along the predicted path must be done on a speculative basis. Speculative execution means that instructions are executed before the processor is certain that they are in the correct execution sequence.
- Hence, care must be taken that no processor registers or memory locations are updated until it is confirmed that these instructions should indeed be executed. If the branch decision indicates otherwise, the instructions and all their associated data in the execution units must be purged, and the correct instructions fetched and executed.



7.5.3 Dynamic Branch Prediction

- The objective of branch prediction algorithms is to reduce the probability of making a wrong decision, to avoid fetching instructions that eventually have to be discarded. In dynamic branch prediction schemes, the processor hardware assesses the likelihood of a given branch being taken by keeping track of branch decision every time that instruction is executed.
- In its simplest form, the execution history used in predicting the outcome of a given branch instruction is the result of the most recent execution of that instruction. The processor assumes that the next time the instruction is executed, the result is likely to be the same.

7.6 INFLUENCE ON INSTRUCTION SETS

- We have seen that some instructions are much better suited to pipeline execution than others. For example, instruction side effects can lead to undesirable data dependencies. In this section, we examine the relationship between pipelined execution and machine instruction features. We discuss two key aspects of machine instructions – addressing modes and condition code flags.

7.6.1 Condition Codes

- In many processors, the condition code flags are stored in the processor status register. They are either set or cleared by many instructions, so that they can be tested by subsequent conditional branch instructions to change the flow of program execution. An optimizing compiler for a pipelined processor attempts to reorder instructions to avoid stalling the pipeline when branches or data dependencies between successive instructions occur. In doing so, the compiler must ensure that reordering does not cause a change in the outcome of a computation. The dependency introduced by the condition-code flags reduces the flexibility available for the compiler to reorder instructions.

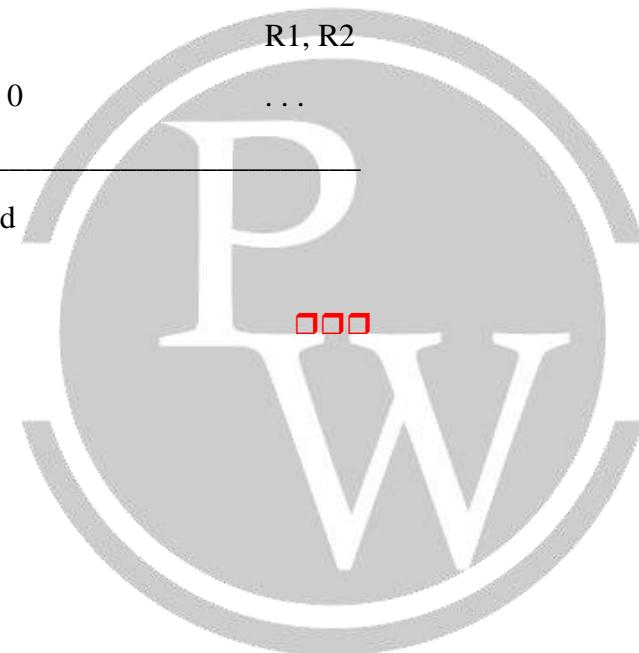
- Consider the sequence of instructions in Figure, and assume that the execution of the Compare and Branch = 0 instructions proceeds as in Figure. The branch decision takes place in step E₂ rather than D₂ because it must await the result of the Compare instruction. The execution time of the Branch instruction can be reduced

Add	R1, R2
Compare	R3, R4
Branch = 0	...

(a) A program fragment

Compare	R3, R4
Add	R1, R2
Branch = 0	...

(b) Instructions reordered

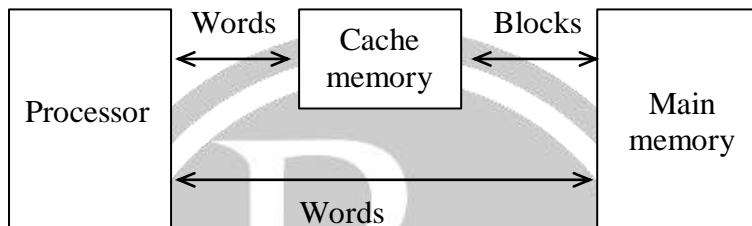


8

CACHE MEMORY

8.1. Introduction

- The speed of the main memory is very low in comparison with the speed of modern processors. For good performance an efficient solution is to use a fast cache memory.



- It is the smallest and fastest memory component in the hierarchy.
 - By placing active portions of the program and data in a fast small memory, the average access time can be reduced.
 - The effectiveness of cache mechanism is based on principle called "**locality of reference**"
 - The Locality of reference states that, "The references to memory at any given interval of time tend to be confined within a few localized areas in memory. i.e. many instructions in localized areas of the program are executed repeatedly. It can be
 - Temporal:** It means that recently executed instruction is likely to be executed again very soon.
 - Spatial:** It means that instructions in close proximity to CI recently executed instruction,
- Example:** loops, nested loops, procedure calls. Etc.

- The performance of cache is measured using Hit ratio

$$\text{Hit ratio} = \frac{\text{no. of hits}}{\text{Total CPU references}} \times 100$$
$$\text{Hit ratio} = \frac{\text{no. of hits}}{\text{no.of hits} + \text{no.of misses}} \times 100$$

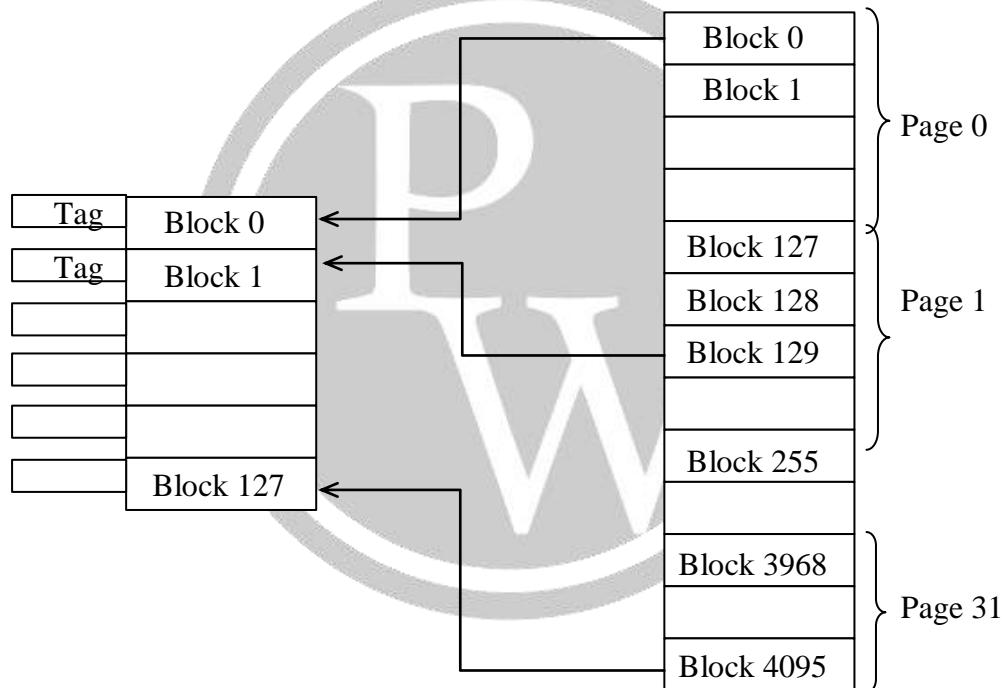
- The performance of cache can be analysed with the following characteristics.
 - (1) Cache size (Small in KB's)
 - (2) Block or line size
 - (3) No. of levels of cache
 - (4) Cache mapping process.
 - (5) Cache replacement algorithm
 - (6) Cache updating scheme.

8.2. Cache Mapping Functions

- Consider a cache consisting of 128 blocks of 16 words cache, assume the main memory is addressable by a 16-bit address. The main memory is addressable by a 16-bit address. The main memory has 64K words viewed as 4K blocks of 16 words each.
- The various mapping functions are
 - (i) Direct mapping
 - (ii) Associative mapping
 - (iii) Set – Associative mapping

8.2.1 Direct Mapping

- The simplest way to determine cache locations in which to store memory blocks.
- Each block from the main memory has only one possible location in cache.
- Block j of the main memory maps to block $j \bmod n$. Where n is the no. of blocks in the cache.



- If the processor needs to access same memory locations from two different pages of the main memory frequently, then miss & will be more.

- Easy to implement but hit ratio is less.

- To implement this the address is divided into Three fields.

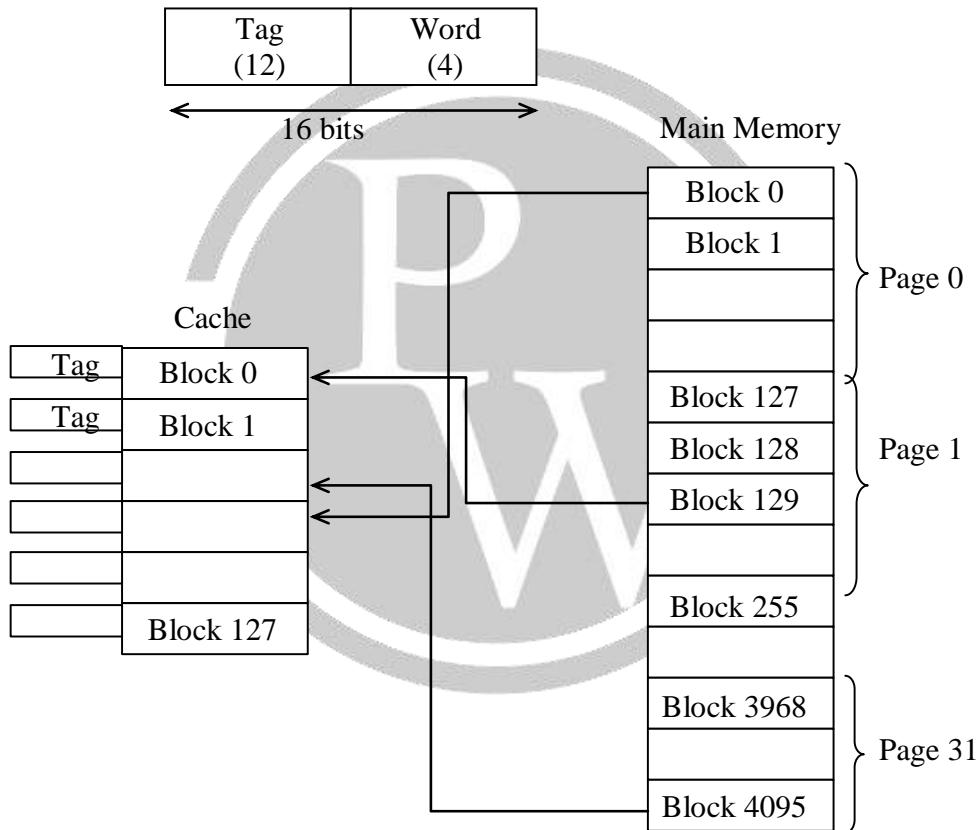
- Example : Tag Block Word

5	7	4
---	---	---

- As execution proceeds, the higher order tag bits of the address are compared with the tag bits associated with that cache location. If they match, then the desired word is in that block of cache. If there is no match, then miss will occur.
- Requires only one comparison.

8.2.2 Associative Mapping

- In which a main memory block can be placed into any cache block position.
- It gives complete freedom in choosing the cache location in which to place the memory block. Thus the space in the cache can be used effectively.
- A new block that has to be brought into the cache has to replace an existing block if the cache is full.
- The cost is high, because of need to search all tag patterns to determine whether a given block is in the cache, i.e. comparison circuit is more complex.
- Requires maximum of n comparisons. Where n is the number of cache blocks.
- The no. of tag comparators = no. of cache blocks.
- The address is divided into 2 fields.

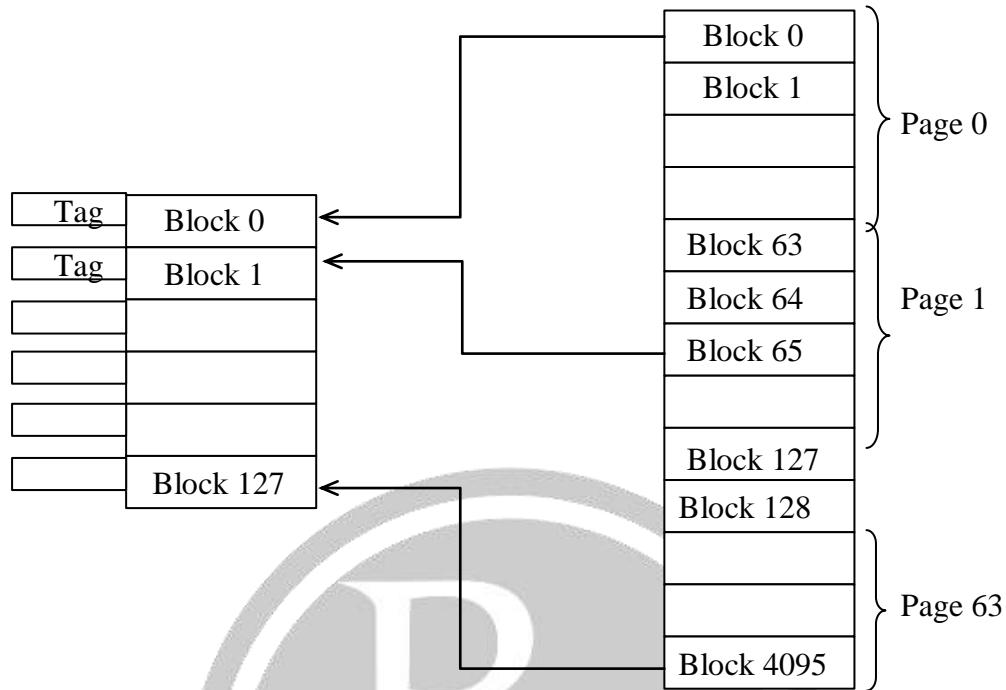


- We need an algorithm to select the block to be replaced.

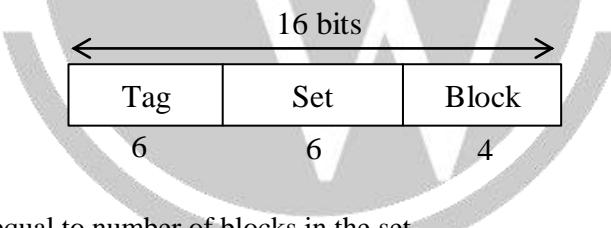
8.2.3 Set - Associative Mapping

- For efficiency, a combination of direct and associative mapping techniques can be used.
- Blocks of the cache are grouped into sets, and the mapping allows a block of the main memory to reside in any block of a specific set. i.e. contains several groups of directed mapped caches in parallel.
- The contention problem of the direct mapped method is eased by having a few choices for block placement.

- The hardware cost is reduced for comparing tags unlike associative mapping.
- A block can occupy either of blocks in the set.



- A cache that has K blocks per set, then it is referred as K-way set-associative cache.
- The address is divided into 3- fields.



- Number of tag comparisons is equal to number of blocks in the set.

8.3. Cache Replacement Policy

- In direct mapped cache, the position of each block is predetermined hence no replacement strategy exists.
- In Associative and set – Associative caches there exists some flexibility, when a new block is to be brought into the cache and all positions that it may occupy are full. Hence replacement strategy exists.
- The replacement policies are aimed for reducing the penalty (miss) to feature references.

The various block replacement policies are

- (1) **FIFO:** The block which enters first will be the candidate for replacement. i.e. Assumes that, since it has spent long time in cache all the references to it are slightly exhausted. Implemented using queue data structure.
- (2) **LRV:** The block in the set which has been in the cache longest with no references to it is selected for the replacement. (Least recently used).
- (3) **LFU:** The block in the set which has the fewest references is selected for replacement. (Least frequently used).

- (4) **Random:** No specific criteria for replacement of any block. The existing blocks are replaced randomly.

8.4. Cache Updation Policy

- The two cache updation schemes employed are
- **Write – through:** The simplest and commonly used approach. Updation of cache and main memory are done simultaneously.
 - Main memory contains the same data as the cache. Which is important for DMA transfers.
- **Write-back:** In this method, only the cache location is updated during a write operation. The location is then marked by a flag so that later when the word is removed from the cache it is copied into main memory.
 - i.e. updation of main memory is postponed until the cache block selected for replacement.

8.5. Cache Memory & Arrays

An array is a homogeneous collection of data items stored in contiguous memory locations either in row major order or column major order.

Example:

$$\begin{bmatrix} 3 & 1 & 2 \\ 5 & 6 & 9 \\ 8 & 4 & 7 \end{bmatrix}$$

Row major: 00 01 02 10 11 12 20 21 22

3	1	2	5	6	9	8	4	7
---	---	---	---	---	---	---	---	---

Col major: 00 10 20 01 11 21 02 12 22

3	5	8	1	6	4	2	9	7
---	---	---	---	---	---	---	---	---

- Q.1.** Consider an array is A[100] and each element occupies 4 – words, 32 – word cache is used and it is divided into 8 – word blocks (a) what is the hit ratio for the following instruction.

- (a) For (i = 0 ; i < 100 ; i++)
 $A[i] = A[i] + 10;$

Ans.

	R	W
A[0]	M	H
A[1]	H	H
A[2]	M	H
A[3]	H	H

A(0)
A(1)
A(2)
A(3)

- (b) How many times block ‘0’ is modified in the cache memory.

Ans. 0, 8, 16, ----- 80, 88, 96. \Rightarrow 13 times.

- (c) How many times block ‘0’ is replaced.

Ans. 12 times.



9

MAIN MEMORY

9.1. Introduction

- The main memory is the central storage unit in a computer system. The principal technology used for the main memory is based on semiconductor integrated circuits. Main memory is made up of RAM and ROM chips. Different types of integrated circuit memories are given in Table 1.
- Integrated circuit RAM chips are available in two possible operating modes, static and dynamic. The static RAM consists of interval flip flops that store the binary information. The stored information remains valid as long as power is applied to the unit i.e. ROM is volatile. The dynamic RAM stores the binary information in the form of electric charges that are applied to capacitors.

Memory	Access	Volatile	Features
Read/Write	Random or serial	Yes	Data can be read or written with equal ease. Used whenever data is needed fast and often, and is frequently changed.
Read-only (ROM)	Random	No	(a) Data is stored permanently by a masking step during manufacture. (b) Used whenever data is not to be changed as in fixed tables, constants or computer instruction sets.
Programmable Read-only (PROM)	Random	No	(a) Can be programmed after IC is manufactured. (b) Data is written by opening fusible metal links or P-N junction's with high currents. (c) Can only be written into once. Same use as a ROM, but is cheaper in small quantities.
Re-programmable read-only	Random	No	Are ROMs that can be written into many times. Are really “read-mostly” rather than “read-only” memories Two main types: (a) Those in which writing is by voltage application and erasing (all data at once) by exposing chip to ultra-violet radiation through a window on the IC (b) Those in which reading and writing is electrical. Data remains stored even with no power applied through the use of MNOS transistors. Used where frequent or at least more than one change in a program is required as in debugging a program.
Content-addressable (CAM)	Random	Yes	This extracts all data in an address when a part of the contents of that address match a specified number. Used in associative memories to obtain stored data related in some way to the input data.

Charge-coupled device (CCD)	Serial	Yes	(a) Digital input is converted to charge and stepped through a shift register. (b) Requires refresh circuitry to prevent data loss.
Programmable logic array (PLA)	Random	No	(a) Is a memory structure that is mask or field (FPLA) programmed as is a ROM. However, it implements logic function. (b) Inputs are functions of the input variables and the logic operation stored in the address. (c) Is more flexible than random (hard-wired separate IC) circuits because PLAs and FPLAs are programmable.

Refreshing is done by cycling through the words every few milliseconds to restore the decaying charge. The dynamic RAM offers reduced power consumption and larger storage capacity in a single memory chip. The static RAM is easier to use and has shorter read and write cycles. The Table 2 shows the summary of various types of integrated circuit memories.

(a) Types of Memory Access	
Random access	Any address can be accessed at random, that is, without going through other address first. Data retrieval time is relatively fixed.
Serial access	Typified by shift register or charge-coupled-device (CCD) memories. Data retrieval is serial in a fixed order. All data ahead of required data must be read first.
(b) Static verses Dynamic Storage	
Static storage	Data is stored in flip-flops or other memory cells in which the data does not deteriorate with time.
Dynamic storage	Data is stored in leaky capacitors so that refresh circuitry is required to prevent data loss.

9.2. Memory Interface

A computer uses memory capacity of 512 bytes of RAM and 512 bytes of ROM. The IC sizes of RAM and ROM are 128×8 and 512×8 bits respectively.

- (a) Find the number of RAM ICs.
- (b) Find the number of ROM ICs.
- (c) Give the memory map of the system
- (d) Mention the size of decoder.
- (e) Give the diagram of memory connection to the CPU.

Solution:

$$\begin{aligned}
 (a) \quad \text{The number of RAM ICs} &= \frac{\text{Total RAM size}}{\text{RAM IC size}} \\
 &= \frac{512 \times 8}{128 \times 8} \\
 &= 4
 \end{aligned}$$

(b) The number of ROM ICs = $\frac{\text{Total ROM size}}{\text{ROM IC size}}$

$$= \frac{512 \times 8}{512 \times 8}$$

$$= 1$$

(c) The memory map of this system is given by Table. 3

Component	Hexadecimal address	Address bus											
		10	9		8	7	6	5		4	3	2	1
RAM 1	0000–007F	0	0		0	×	×	×		×	×	×	×
RAM 2	0030–00FF	0	0		1	×	×	×		×	×	×	×
RAM 3	0100–017F	0	1		0	×	×	×		×	×	×	×
RAM 4	0180–01FF	0	1		1	×	×	×		×	×	×	×
ROM	0200–03FF	1	×		×	×	×	×		×	×	×	×

(d) 2×4 decoder

Each RAM receives the seven low-order bits of the address bus to select one of 128 possible bytes. The particular RAM chip selected is determined from lines 8 and 9 in the address bus. This is done through a 2×4 decoder. The outputs of decoder are given to the CSI inputs in each RAM chip. Thus when address lines 8 and 9 are equal to 00, the first RAM chip is selected when 01, the second RAM chip is selected and so on.

(e) The connection of memory chips to the CPU is shown in Fig. RAM and ROM chips are connected to a CPU through the data and address buses. The low order lines in the address bus selected the byte within the chips and other lines in the address bus select a particular chip through its chip select inputs.

□□□

10

SECONDARY STORAGE

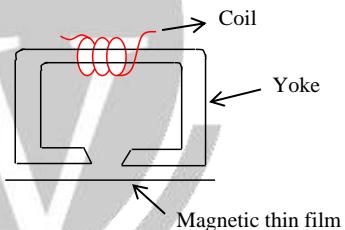
10.1 Introduction

- The cost per bit of stored information is high in semiconductor memories (main memory). This limits the use of these memories for large storage. Large storage requirements of most of computers are economically fulfilled by magnetic disks, magnetic tapes and optical disk memories. These memories are referred as secondary storage devices.

10.2 Magnetic Disks

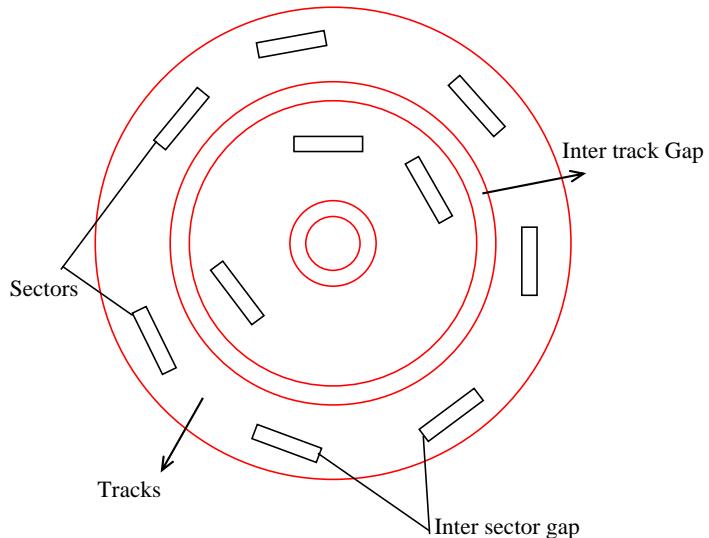
- A magnetic disk is a thin, circular metal plate. It is coated with a thin magnetic film usually on both sides.
- Digital information is stored on the magnetic disk by magnetizing the magnetic surface in a particular direction.
- Magnetic disks are semi random access memories.

The head consists of a magnetic yoke and the magnetizing coil. The Digital information can be stored on the film by applying current pulses of suitable polarity.



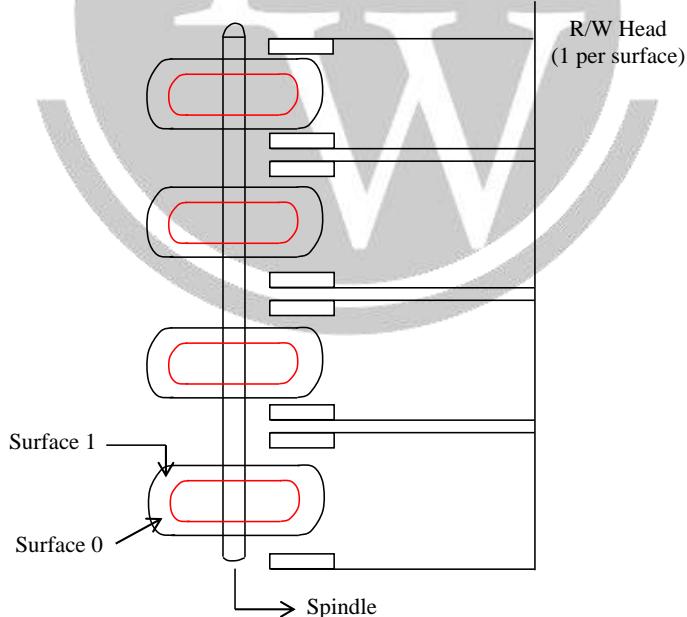
10.2.1. Data organization & Formatting

- The data on the disk is organized in a concentric set of rings. These concentric set of rings are called tracks.
 - Each track has same width as head and adjacent tracks are separated by gaps.
 - Each track is divided into manageable units called sectors.
 - Each sector stores a block of data which can be transferred to or from the disk.
 - The universal size of sector is 512 bytes.
 - The R/W head is capable of reading a sector from disk and writing a sector to disk.
 - Placing R/W head on desired track is random access and reading concerned sector is serial. Hence disks are semi random access devices.



10.2.2. Physical Characteristics

- | | |
|-----------------------|--|
| (i) Head Motion | - Fixed head (one per track)
Movable head (one per surface) |
| (ii) Disk portability | - Non removable or Removable |
| (iii) Sides | - Single sided, Double – sided. |
| (iv) Disk/surface | - Single surface, Multiple – surfaces |
| (v) Head Mechanism | - Contact, fixed Gap, Aerodynamic Gap. |



A vertical set of all of the tracks with the same number on each surface of a diskette or hard disk is called “cylinder.”

- (vi) Platters - Single platter,
Multiple platters – some disk
Accommodate multiple platters
Stacked vertically a fraction of an inch apart.

- (1) If all the tracks are having constant capacity then the disk moves with “constant angular capacity” and “recording density” varying from track to track.
- (2) If each track is having variable capacity then the disk is moving with “constant linear velocity”
 - Placing the control information in the sector is called as formatting.
 - The disk controller is the hardware interface to control the operation of a disk drive. Used to control more than one drive. The major functions are seek, read, write and error checking.
 - The start and end of each sector is determined by the control data stored in the sector.
 - Disk performance parameters:

Seek Time: It is the time required to move the disk arm to the required track.

Rotational Delay: The time taken for the beginning of sector to reach. (latency)

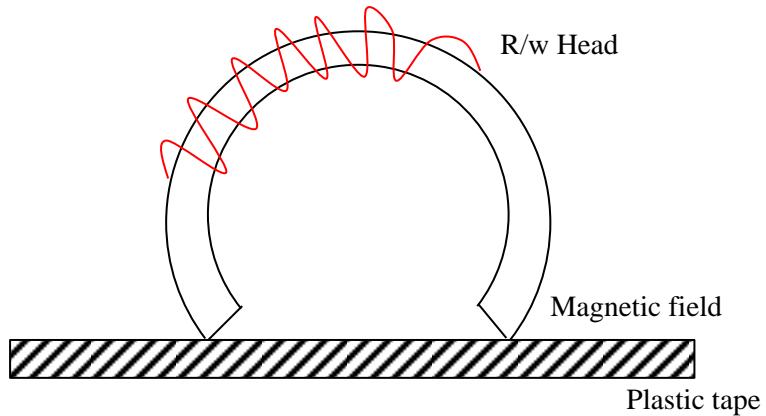
Access time: The sum of seek time and the rotational delay.

- The average time to disk access is
- $$t_{avg} = t_{seektime} + t_{rotational\ delay} + t_{data\ transfer} + t_{over\ head}$$
- Maximum Recording Density (P) = No. of Bytes/cm
 - Data transfer rate (D) = Number of Bytes/Sec.

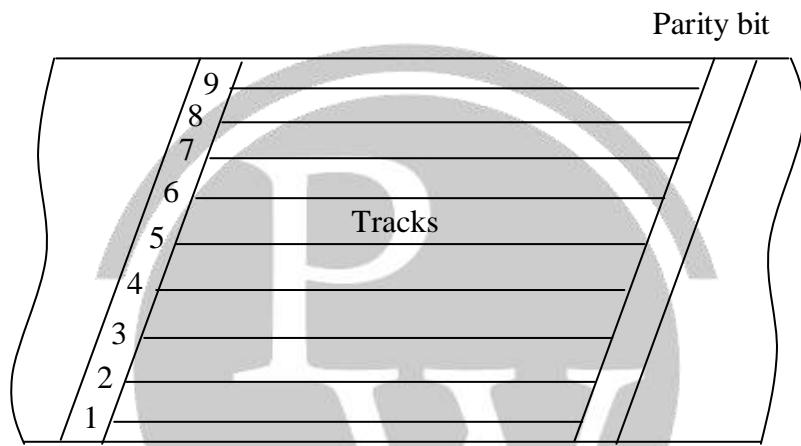
10.3 Secondary Storage Devices

10.3.1. Magnetic Tapes

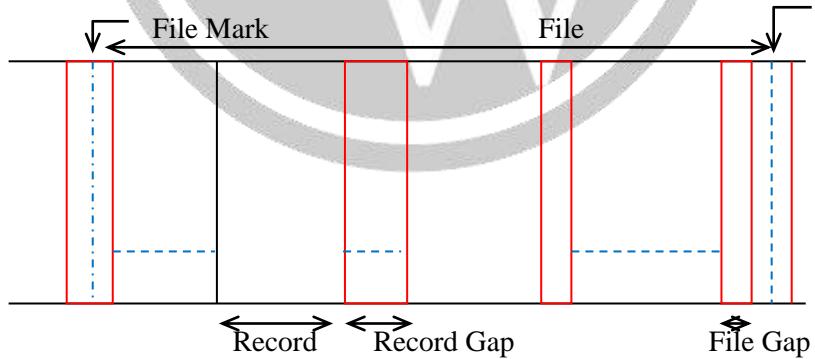
- Magnetic tape is one of the most popular storage medium for large data that are sequentially accessed and processed.
- The tape is formed by depositing magnetic film on a very thin and $\frac{1}{2}$ or $\frac{1}{4}$ inch wide plastic tape.
- Usually, iron oxide is used as a magnetizing material.
- The information is recorded on the tape with the help of read/write head. It magnetizes or non magnetizes tiny invisible spots (1's & 0's).
- Usually 7 or 8 bits are recorded (a character) in parallel across the width of the tape, perpendicular to the direction of motion. A separate R/Wo head is provided for each bit position on the tape.



- Data on the tape are organized in the form of records separated by gaps.
- A set of related records constitutes a file.



- The data on a 9th track (a) tape is stored in parallel consists of data byte and a parity bit.



- Data transfer takes place when the tape is moving at constant velocity relative to a read/write head.
- Hence the maximum data transfer rate depends largely on the storage density along the tape and speed of the tape.
- The file mark is used as header or identifier.
- The information on the magnetic tape is organized into blocks. These blocks have a fixed length and separated by gap between them.
- If the block length is B_L and inter block gap length is G_L , then the utilization factor of the tape (u) is given by

$$u = \frac{B_L}{B_L + G_L}$$

- The unit of data transfer is a record.
- The tape is moving with a linear velocity and Read/write head is constant.
- Let L is the length of the tape,
 - N is the number of parallel tracks,
 - P is the constant recording density.

(i) Capacity of the tape

$$C = L \times N \times P$$

(ii) Let V is the linear velocity of the tape, then the data transfer rate

$$D = V * N * P$$

- ✓ With utilization factor, the effective data transfer rate is

$$D_{\text{eff}} = D * \frac{B_L}{B_L + G_L}$$

- ✓ Due to inter block gap and time needed to start and stop and tape between accesses, the effective data transfer rate d_{eff} is actually less than the maximum rate d .

$$d_{\text{eff}} = \frac{t_D \times d}{t_D + t_G + t_{ss}}$$

t_D = time to scan a data block

t_G = time to scan the inter block gap.

T_{ss} = time to start and stop the tape.

OUR YOUTUBE CHANNELS



**GATE
WALLAH**
EE, EC & CS



GATE Wallah



**GATE
WALLAH**
ME, CE & XE

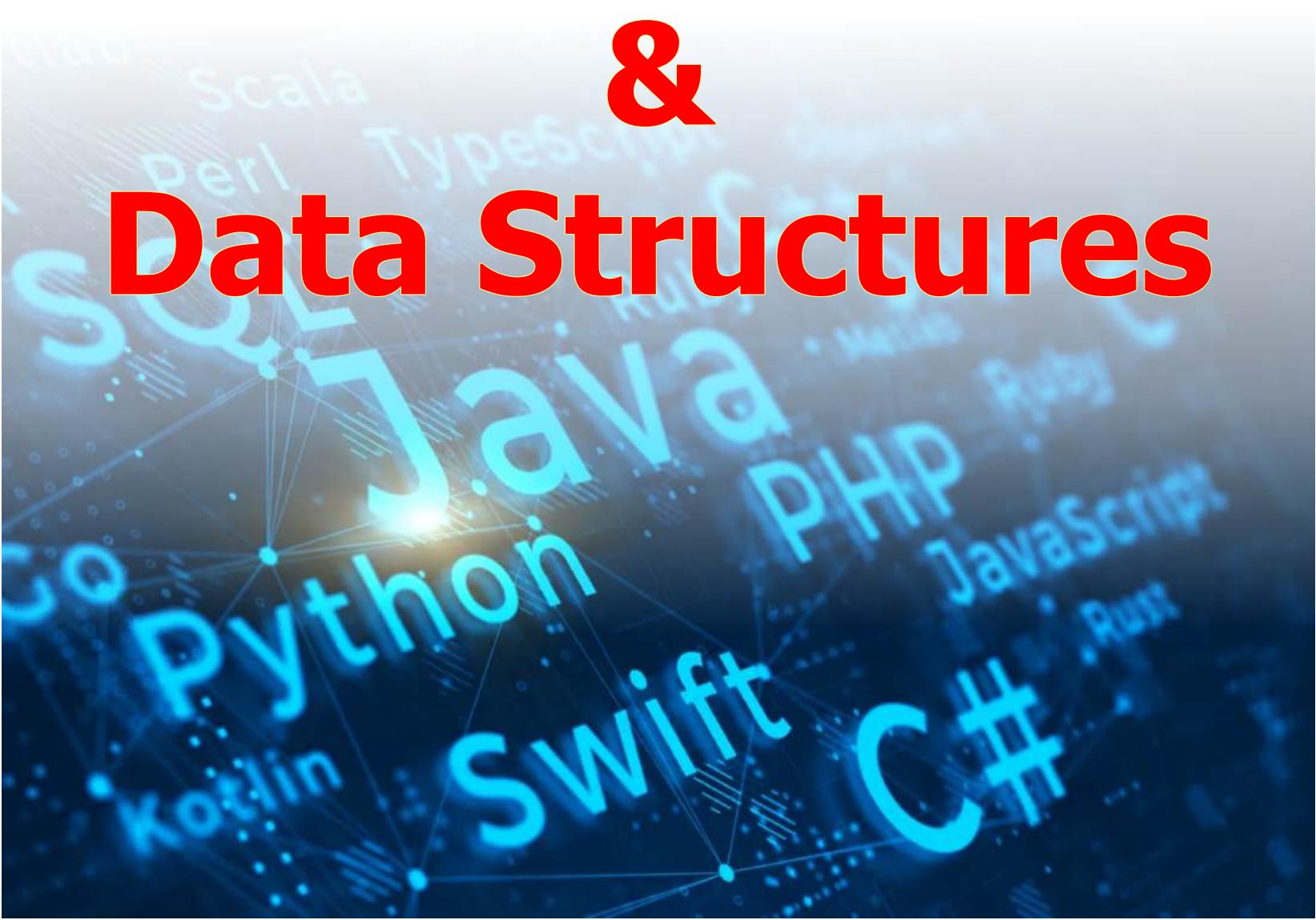


GATE Wallah (English)

ACCESS QUALITY CONTENT

For Free

Programming & Data Structures



Programming & Data Structure

INDEX

- | | | |
|-----|---|-------------|
| 1. | Data Types and Operators | 5.1 – 5.5 |
| 2. | Control Flow Statements..... | 5.6 – 5.8 |
| 3. | Storage Class & Function | 5.9 – 5.12 |
| 4. | Arrays and Pointers | 5.13 – 5.22 |
| 5. | Strings | 5.23 – 5.26 |
| 6. | Types of Data Structure, Array & Linked List..... | 5.27 – 5.31 |
| 7. | Stack and Queue | 5.32 – 5.33 |
| 8. | Tree Data Structure..... | 5.34 – 5.38 |
| 9. | Graph Traversal | 5.39 |
| 10. | Hashing..... | 5.40 – 5.41 |

1

DATA TYPES AND OPERATORS

1.1 Data Types

1.1.1 Primitive Data Type

(a) Integer Types:

- ✓ short int, unsigned short int
 - ✓ int, unsigned int
 - ✓ long int, unsigned long int
 - ✓ long long int, unsigned long long int

(b) Character Types:

- ✓ char,unsigned char

(c) Floating Types:

- ✓ float, double, long double

(d) Other:

- ✓ void, bool

1.1.2 Non – Primitive Data Type

(a) Derived data type:

- ✓ Array
 - ✓ Pointer
 - ✓ String

(b) User defined data type:

- ✓ Structure
 - ✓ union
 - ✓ Enum
 - ✓ typedef

- C standard does not specify how many bits or bytes to be allocated for every type and different compiler may choose different ranges.
 - Only restriction is that short and int types are at least 16 bits, long types are at least 32 bits and short is no longer than int which is no longer than long type.

1.2 Operators

Depending upon the number of operands, an operator in C language can be unary, binary or ternary.

1.2.1 Arithmetic Operators

- (i) Addition (+)
 - (ii) Subtraction (-)
 - (iii) Multiplication (×)
 - (iv) Division (/)
 - (v) Modulus (%)
- Both operands for % operator must be integer types, otherwise error will be raised.
 - The sign of the result for modulo operator is machine dependent.

1.2.2 Relational Operators

- (i) Less than (<)
- (ii) Greater than (>)
- (iii) Less than equal to (<=)
- (iv) Greater than equal to (>=)
- (v) Equal checking (==)
- (vi) Not Equal to (!=)

- The result of these operators is always 0 to 1

Ex.1: `printf ("%d", 20 > 10)`

O/P: 1

Ex.2: `printf ("%d", 20 == 10)`

O/P: 0

1.2.3 Assignment Operator (=)

- Used to assign a value or value of an expression to a variable.
- Typically, the system is
Lvalue = Rvalue
- Lvalue must be a variable.
- Lvalue cannot be a literal or expression.
- Rvalue can be an expression, variable, literal.

Ex1.

The following are invalid

`10 = a;`

`a + b = c;`

- The result of assignment statement is the value we are assigning i.e.....

`int x;`

`x = 3 + (x = 10);` is perfectly valid.

Firstly `x = 10` evaluated and

(1) 10 is assigned to `x` (2) then the result of `(x = 10)` will be 10

so,

`x = 3 + 10`

x = 13

Finally, 13 is assigned to x.

1.2.4 Logical Operators

- (i) Logical AND (&&)
- (ii) Logical OR (||)
- (iii) Logical NOT (!)

- Just like rational operators, the result of every logical operator is either 0 or 1.
- Logical NOT is a unary operator
- Logical NOT converts a non-zero operand into 0 and a zero operand in 1.

1.2.5 Short Circuiting in Logical Operators

- In case of logical AND, the second operand is evaluated only if the first operand is evaluated to be true.
If the first operand is evaluated as false then the second operand is not evaluated.
- In case of logical OR operation, the second operand is not evaluated if the first operand is evaluated as true.

Example1: void main () {

```
printf ("Hello") || printf ("Pankaj");  
}
```

O/P: Hello

printf function display everything written within double quotes on the monitor and the result / value returned by printf () is the number of characters successfully displayed on the screen.

So, printf ("Hello") prints Hello and return 5.

So, the expression

```
printf ("Hello") || printf ("Pankaj");
```

becomes

```
5 || printf ("Pankaj")
```

As the first operand for logical OR is evaluated as true, second printf () will never execute.

1.2.6 Increment and Decrement Operators

- (i) pre increment ++x
- (ii) post increment x++
- (iii) pre decrement - -x
- (iv) post decrement x - -

- unary operator.
- can't be applied on constant / literals.
- Pre increment: can be viewed as 2 step operators.
1st step: Increment the value of variable.
2nd step: After increment, use the value of variable.
- Post increment: can be viewed as 2 step operators
1st step: Use the value.
2nd step: Increase the value of variable by 1.

1.2.7 Bitwise operators

- (i) Bitwise AND (&)
 - (ii) Bitwise OR (|)
 - (iii) Bitwise XOR (^)
 - (iv) Bitwise Left Shift (<<)
 - (v) Bitwise Right Shift (>>)
 - (vi) Bitwise Not (~)
- All these operators work on binary representation of numbers.
 - Typically, faster than arithmetic operators and other operators.

```
#include<stdio.h>
int main () {
    int x = 3, y = 6;
    printf ("%d\n", x & y); // output 2
    printf ("%d\n", x|y); // output 7
    printf ("%d\n", x^y); // output 5
    return 0;
}
binary representation of 3: 00000000...0011
binary representation of 6: 00000000...0110
3 & 6: 00000000...0010
3 | 6: 00000000...0111
3 ^ 6: 00000000...0101
```

XOR of two bits is 1 if both bits are different.

XOR of two bits is 0 if both bits are same.

- $\sim x$ has output as $-(x + 1)$
- `printf ("%d\n", ~2)` has output -3 i.e., $-(2 + 1)$
- Comma has lowest precedence among all operators in C language.
- `sizeof()` is used
 - (i) To find the number of elements present in an array.
 - (ii) To dynamically allocate block of memory.

1.2.8 Ternary operator (? :)

- It requires 3 operands i.e., Left, Middle, Right
Left ? Middle : Right
- If left expression evaluates as true, then the value returned is middle argument otherwise the returned value is Right expression
`int a;`
$$a = 7 > 2 ? 3 \times 2 + 5 : 6! = 7$$

 \Rightarrow Left expression: $7 > 2$ is true
 \Rightarrow So, the value returned would be the middle argument.
i.e., $3 \times 2 + 5 = 11$
so, `a` will get 11.

Note:

- (a) No of ‘?’ and ‘:’ should be equal.
- (b) Every colon (:) should match with just before ‘?’.
- (c) Every ? followed by : not immediately but following.

Example: int a;

a = 2 > 5 ? 10 : !5! = 2 > 5 ? 20 : 30

— — —
L₁ M₁ R₁

2 > 5 is false.

so, for Leftmost ? the value returned is its right expression.

a = ! 5 ! = 2 > 5 ? 20 : 30 :

Left Mid Right

Left expression:

!5! = 2 > 5

0! = 2 > 5

0! = 0

0 i.e., false.

As left operand is false, the final value is right expression

a = 30

Operators	Associativity
() [] -> .	Left to right
! ~ ++ -- * (type) size of	Right to left
* / %	Left to right
+ -	Left to right
<< >>	Left to right
< <= > >=	Left to right
== !=	Left to right
&	Left to right
^	Left to right
	Left to right
&&	Left to right
	Left to right
? :	Right to left
= += -= *= /= %= &= ^= = <<= >>=	Right to left
,	Left to right



2

CONTROL FLOW STATEMENTS

2.1 Switch Statement

- “Case Value” can be of character type and int type.
- There can be one or many number of cases.
- Statements associated with a matching case executes until a break statement is reached.
- The position of default case does not matter. It can be placed anywhere.
- Default case is optional.
- Duplicate case labels are not allowed.
- Statements written before all the cases are ignored by the compiler.
- The break statement is optional.
- Case label cannot be a variable.

2.2 Iterative statements (Loop)

An iterative statement, or loop, repeatedly executes a group of statements, known as body of loop, until the controlling expression is false.

2.2.1 For Loop

- The for statement evaluates 3 expressions and executes the loop body until second controlling expression executes to false.
- It is recommended to use for loop when the number of iterations is known in advance.
- Syntax of for loop is:

```
for (expression – 1(optional); expression – 2(optional); expression – 3(optional))
{
    statement – 1
    statement – 2
    :
    statement – n
}
```
- for loop executes the loop body 0 or more times.
- for statements works as follows:
 - (a) expression – 1 is evaluated once before the first iteration of the loop.
 - (b) expression – 2 is a expression that determines whether to terminate the loop. expression – 2 is evaluated before every iteration.
If the expression is true (non - zero), the loop body executed.
If the expression is false (zero), execution of the for statement is terminated.
 - (c) expression – 3 is evaluated after each iteration.

- (d) The for statement executes until expression-2 is false (0), or until a jump statement terminates execution of the loop.

- All 3 expression can be omitted

- (a) If we omit expression-2, the condition is considered as true for an example:

for (i = 0; ; i++) } represent an infinite loop
statement.

2.3 While Statement

- The while statement evaluates a controlling expression before every execution of the loop body.
- If the controlling expression is true (non-zero), the loop body is executed. If the controlling expression is false (zero), then the while statement terminates.
- Syntax:

while (expression)	while (expression)
	{
OR	statement-1
statement	statement-2
	:
	statement-n
	}

2.4 Do-while Loops

- Both for and while loop checks the loop termination condition before every iteration but do while loop check the condition after executing the loop body.
- do while loop body executes at least 1 time, no matter whether the loop termination condition is false or true.
- Syntax is:

do	do {
statement	statement-1
	statement-2
OR	:
while (expression);	statement-n
	}
	while (expression);

2.5 Break Statement

- The break statement provides an early exit from for, while and do while.
- A break statement causes the innermost loop or switch to be exited immediately.
- In implementation, when we know the maximum number of repetition but some condition is there, where we require to terminate the repetition process, then use break statement.

Example:

```
void main ()  
{  
int i = 1;  
while (i <= 15)  
{
```

```
printf("%d\t",i);
if (i > 4)
    break;
i = i + 1;
}
printf("End");
}
```

Output: 1 2 3 4 End

In above code, when i becomes 5, the condition $i > 4$ becomes true, so break statement executes & control passed outside of the loop body i.e., printf("End") executed.

2.6 Continue Statement

- Continue statement is related to break but it is not used that much frequently.
- Whenever continue is encountered in a loop, it skips the remaining statement of the current iteration and continues with the next iteration of the loop.
- Only applicable with loops, not with switch statement.

Example:

```
void main () {
    int i = 1;
    for (i = 1; i <= 10; i++)
    {
        if(i%2 == 0)
            continue;
        printf("%d",i);
    }
}
```

Output: 13579 (All odd numbers between 1 to 10)

whenever i becomes even, condition $i \% 2 == 0$ becomes true & continue causes the control to go to $i++$ & printf is skipped.



3

STORAGE CLASS & FUNCTION

3.1 Memory Organization of C Program

3.1.1 Code Segment

- It is also known as text segment.
- It contains executable instructions and this segment is a read only segment.
- Usually, this section is sharable.

3.1.2 Uninitialized data segment

- This section contains all static and global variables that are not initialized by the programmer and hence initialized to zero (default value).

3.1.3 Initialized data segment

- This section contains all static and global variables that are initialized by the programmer.

3.1.4 Stack

- In this local variables are stored and some other information is also saved.
- A set of values stored for a function is called as stack frame which atleast contain return address.

3.1.5 Heap

- This area is responsible for dynamic memory allocation. The heap area is managed by malloc(), calloc(), ralloc() and free() which may be use brk() and sbrk() system calls.

3.1.6 Static Memory

- Compile time allocation
- Static variable
- Global Variable

3.2 Storage Classes

Storage Class	Scope	Life Time	Default	Storage area
auto	Local	Within function	Garbage Value	RAM
static	Local	Till end of main program	0	RAM

extern	Global	Till end of main program	0	RAM
Register	Local	Within function	Garbage	Register

- Stack Overflow: Abnormal Termination
- If conflict between global and local variable occurs, then local variable gets preference.
- Declaration of a register variable is only a recommendation/request not a command.
- Cannot apply '&' operator with register variable.
- No physical memory is allocated using 'extern' keyword.
- Extern declaration is mandatory when an external variable is referred before it is defined or if it is defined in some other source file from the file where it is being used.
- Declaration of an external variable tells about the properties like its type, while definition leads to storage allocation.

3.3 Recursion

Definition: When the body of a function call the function itself directly or indirectly.

- Certain arguments for which the function does not call itself are called as base argument, base values.
- In general, for recursion to be non-cyclic whenever a function calls itself the formal arguments must get closer to the base argument.

Example 1:

Consider the factorial code:

```
int factorial (int n)
{
    if (n == 0)
        return 1;
    else
        return n * factorial (n - 1);
}
```

If we call factorial (3), it will call factorial of 2 and so on ...the argument will get closer to 0 i.e., the base argument.

- Recursion code is shorter than iterative code.
- Overhead is present.
- Some standard problems are best suited to be solved by using recursion for example- Tower of Hanoi, Factorial, merge Sort.

3.4 Static and dynamic scoping

3.4.1 Static Scoping

Static scoping is also called as lexical scoping. In this scoping, the binding of a variable can be determined by the program text. In this type of scoping compiler first looks in the current block (local), then in global variables i.e., local and then ancestors' strategy is followed.

Example 1:

```
int a = 10;
void main ()
{
    int a = 1;
    {
        int a = 2
        printf("%d",a);
    }
}
```

Output: 2

As printf is referring to a, compiler first check in the current block & gets a variable whose value is 2.

Example 2:

```
int a = 10;
void main ()
{
    int a = 1;
    {
        int b;
        printf("%d",a);
    }
}
```

Output: 1

Compiler looks for 'a' in scope S₁, because S₁ does not have variable a, it will look into higher scope S₂ that contains a variable name a whose value is 1.

3.4.2 Dynamic Scoping

In this type of scoping, the compiler first searches the current block and then successively searches all calling functions.

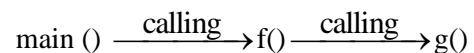
Example:

```
int i;
program main ()
{
    i = 10;
    call f ();
}
procedure f ()
{
    int c = 20;
    call g();
}
Procedure g() {
    print i;
}
```

Assuming that the above program is written in a hypothetical programming language which allows global variables

and dynamic scoping, let's try to find the output.

The order of function calls is:



g() is printing i, which is not present inside current block. So, because of dynamic scoping compiler will go to the function that calls g() i.e., compiler will search f() for variable i. Hence 20 is printed.

◻◻◻



4

ARRAYS AND POINTERS

4.1 Array

- An array is defined as the collection of similar type of data items stored at contiguous memory locations.
- Arrays are the derived data type in C programming language which can store the primitive type of data such as int, char, double etc.
- It has the capability to store the collection of derived data types such as pointer, structure etc.
- Each element of an array is of same data type and carries the same size example int = 4 byte.
- Elements of the array are stored at contiguous memory locations, meaning, the first element is stored at the smallest memory location.
- Elements of the array can be randomly accessed since we can calculate the address of each element of array with the given base address and size of the data element.

4.1.1 Advantages of C Array

- 1) Code Optimization: Less code to access the data.
- 2) Ease of traversing: By using the for loop, we can retrieve the elements of an array easily.
- 3) Ease of sorting: To sort the elements of the array, we need a few lines of code only.
- 4) Random Access: We can access any element randomly using the array.

4.1.2 Disadvantages of Array

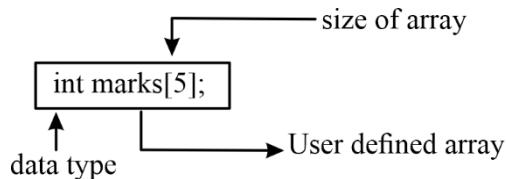
- **Fixed Size:** 1) Whatever size, we define at the time of declaration of the array, we can't exceed the limit.
2) It does not grow the size dynamically like linked list.

Declaration of C Array

Syntax:

```
data type array name [array_size]
```

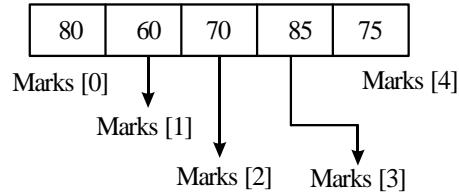
Example:



Initialization of C array:

We can initialize each element of the array by using the index. Suppose array `int marks [5]`. Then

```
marks [0] = 80
marks [1] = 60;
marks [2] = 70;
marks [3] = 85;
marks [4] = 75;
```



C Array Example:

```
# include <stdio.h>
int main ()
{
    int i = 0;
    int marks [5];
    marks [0] = 80;
    marks [1] = 60;
    marks [2] = 70;
    marks [3] = 85;
    marks [4] = 75;
    for(i = 0; i<5; i++)
    {
        printf("%d\n",marks[i]);
    }
    return 0;
}
```

4.1.3 Array: Declaration with Initialization

- We can initialize the C array at the time of declaration.
Example: `int marks [5] = {20, 30, 40, 50, 60};`
- In such case, there is no requirement to define the size.
Example: `int marks [] = {20, 30, 40, 50, 60}`

Example:

```
# include<stdio.h>
int main ()
{
    int i = 0;
    int marks [5] = {20, 30, 40, 50, 60}
    for(i = 0; i < 5; i++)
    {
        printf("%d\n",marks[i]);
    }
    return 0;
}
```

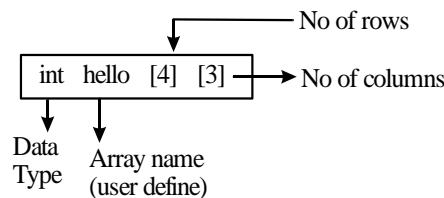
4.1.4 Two-Dimensional Array

- The two-dimensional array can be defined as an array of arrays.
- The 2D array is organized as matrices which can be represented as the collection of row and columns.

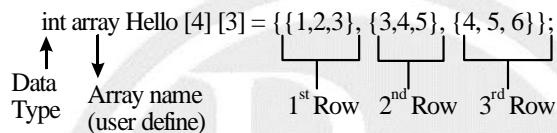
4.1.5 Declaration of two-dimensional array in C syntax

```
data_type array_name [rows] [columns]
```

Example:



Initialization of 2D Array



Example: Program of Two-Dimensional Array

```
# include<stdio.h>
int main()
{
    int i = 0, j = 0;
    int arrayHello[4][3] = {{1, 2, 3}, {2, 3, 4}, {4, 5, 6}};
    for(i = 0; i < 3; i++)
    {
        for(j = 0; j < 3; j++)
        {
            printf("arrayHello[%d][%d]=%d\n", i, j, arrayHello[i][j])
        }
    }
    return 0;
}
```

Output:

```
arrayHello[0][0] = 1
arrayHello[0][1] = 2
arrayHello[0][2] = 3
arrayHello[1][0] = 2
arrayHello[1][1] = 3
arrayHello[1][2] = 4
arrayHello[2][0] = 3
arrayHello[2][1] = 4
arrayHello[2][2] = 5
arrayHello[3][0] = 4
```

```
arrayHello[3][1] = 5  
arrayHello[3][2] = 6
```

4.1.6 Passing array to a function

Example:

```
# include<stdio.h>  
User defined function  
void Helloarray (int marks [ ]) {  
    for(int i = 0; i < 5; i++)  
    {  
        printf("%d", marks[i]);  
    }  
}  
int main()  
{  
    int marks[5] = {45, 67, 34, 78, 90};  
    Helloarray(marks);  
    return 0;  
}
```

Note: void Helloarray(int marks[]) Without return type function

4.1.7 Passing Array to a Functions as a pointer

Now, we will see how to pass an array to a function as a pointer.

```
# include<stdio.h>  
void helloarray(char * marks)  
{  
    printf("Elements of array are:");  
    for(int i = 0; i < 5; i++)  
    {  
        printf("%c", marks[i])  
    }  
}  
int main()  
{  
    char marks[5] = {'A', 'B', 'C', 'D', 'E'};  
    helloarray(marks);  
    return 0;  
}
```

Note:

Whenever you pass an array, it is always call by reference. In previous 2 programs, we passed an address & formal argument in both the cases is a pointer internally.

How to return an Array from a function

```
# include<stdio.h>
int *fun()
{
    int marks [5];
    print f ("Enter the element in an array");
    for (int i = 1; i < 5; i++)
    {
        scanf("%d",& marks[i]);
    }
    return marks;
}
int main()
{
    int *n;
    n = fun();
    printf("\n Elements of array are:");
    for(int i = 0; i < 5; i++)
    {
        printf("%d",n[i]);
    }
    return 0;
}
```

Note:

fun() function returns a variable ‘marks’.

It returns a local variable, but it is an illegal memory location to be returned, which is allocated with a function in the stack. Since the program control comes back to the main () function, and all the variables in the stack are freed.

Therefore, we can say that this program is returning memory location, which is already de-allocated, so the output of the program is a **segmentation fault**.

4.1.8 Array declaration and initialization

1. int A[] = {10, 20, 30}: valid
2. int A[3] = {10, 20, 30}: valid
3. int A[] ; invalid
4. int A[3] ; valid
5. int A[2][3]; valid
6. int A[] [3]; = {10, 20, 30}: invalid
7. int A[2] [3]; = {1, 2, 3, 4, 5, 6}: valid
8. int A[] [3]; = {1, 2, 3, 4, 5, 6}: valid
9. int A[2] [3] [2]; invalid
10. int A[] [3] [2]; invalid
11. int A[] [2] []; invalid
12. int A[] [] [2]; invalid
13. int A[2] [3] [2]; = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12,}; valid

- 14. int A[] [3] [2]; = { 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12,}; valid
- 15. int A[2] [] [2]; = { 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12,}; invalid
- 16. int A[2] [3] []; = { 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12,}; invalid

Note:

- while declaring an array, it is mandatory to provide the size of each dimension. (3), (6), (10), (11), (12) are not providing sizes of dimensions.
- While initializing an array, you have flexibility to omit only first dimension i.e., you are allowed to other dimension (1), (8),(14) are following this rule while (15), (16) are not following this rule.

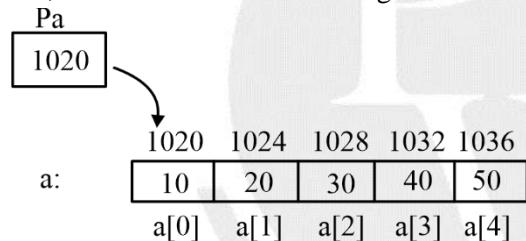
4.1.9 Pointers

- In C, there is a strong relationship between arrays and pointers. An operation that can be achieved by arrays subscripting also be done with pointers.
- Consider the declaration
 - int a[5] = { 10, 20, 30, 40, 50};
 - int* Pa;
 - Pa = &a[0];
- Pointers are special variable that can hold address of other variables

Syntax: data type * Identifier

int *Pa: Pa is a pointer to integer variable

i.e., Pa can hold address of integer variable



Pa contains address of a[0]

- Two operators are important to understand for understanding pointers: &, *
- i. &: address of operator
- ii. *: value at operator
- Pa is equivalent address 1020
- Pa is equivalent to value at (Memory location 1020)
i.e *Pa is same as 10
- Pa is pointing to same element of array, then by definition Pa +1 points to next element, Pa + i points to elements after i elements before Pa.
i.e, In our example Pa points to a[0]
So, Pa+1 will point [1]
Pa+2 will point to a[2]
i.e
Pa+i is address of a[i]
 $\Rightarrow Pa + i \equiv \& a[i]$
 $\Rightarrow *(Pa+i) \equiv * \& a[i] \equiv a[i]$
- Array name always represent address of the very first element
i.e., Pa = &a[0];
and

$\text{Pa} = \text{a};$

both are same.

- $\text{a}[i] \equiv *(\text{a} + i) \equiv *(\text{i} + \text{a}) \equiv \text{i}[\text{a}]$

All are same and can be used interchangeably in program except in declaration.

- In declaration, we can not write

int 3[a]; instead of int a[3]

- Array name represent a constant address.

int a[10];

- a++, ++a, --a, a-- are all invalid as we cannot apply increment/decrement operator on constants:

- Array name cannot be Lvalue of on assignment statement.

i.e.

array_name = any expression/address/value is invalid

- On the other hand, pointer being a variable, the following operations are valid.

int a[5]

int = *Pa;

Pa = a : valid

Pa++ ;

Pa-- :

++ Pa :

-- Pa :

all are valid

- Multi-level pointer

int x : x can store value

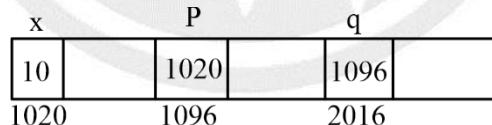
int *P : p can store address of integer variable

int **q : q can store address of pointer variable

x = 10 : valid

p = &x : valid

q = &p : valid



$\text{p} \equiv 1020$ i.e address of x

$*\text{P} \equiv$ value at (memory location 1020)

$*\text{p} \equiv 10$

$\text{q} \equiv 1096$ i.e address of variable p

$*\text{q} \equiv$ value at (memory location 1096)

$\equiv 1020$ which is again an address

$*\text{q} \equiv$ value at (memory address 1020)

$**\text{q} \equiv 10$

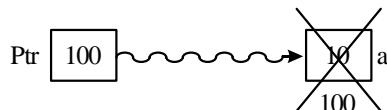
1. **Dangling pointer:** The pointer pointing to a deallocated memory block is known as dangling pointer.

This situation raises an error known as dangling pointer problem. Dangling pointer occurs when a pointer pointing to a variable goes out of scope or when a variable's memory gets deallocated.

Consider the code.

int *f() {

```
int a = 10; // a is a local variable and goes out of scope after execution of f( )
return &a :
}
int main () {
    int * ptr = f (); //Ptr points to something which is not valid
    print ("%d",ptr);
    return 0;
}
```

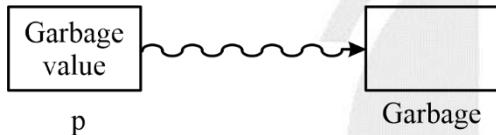


- To overcome this problem, just make variable a as static when x become static it has scope throughout the program.

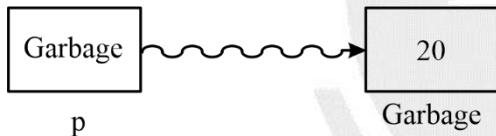
2. Uninitialized pointer: An uninitialized pointer also known as wild pointer

A pointer which has not been initialized to anything can be dangerous

- The value saved in an uninitialized pointer could be randomly pointing anywhere in memory
- int *p



- int *p



- Storing a value using an uninitialized pointer has the potential to overwrite anything in your program including your program itself.
- System may crash.
- Always initialize with NULL.

3. NULL pointer: It is a pointer which is pointing to nothing. It is a specially designed pointer that stores a defined value but not a valid address of any element or object.

NULL pointer does not hold valid address and that is why if you try to dereference it, you will get an error

```
int *p = NULL;
```

```
printf ("%d",*p); //NULL pointer dereferencing
```

4. void pointer: void pointer is a pointer that points to some location in memory but it does not have any specific type.

It can point to any type of data and any pointer type is convertible to a void pointer.

Its declaration:

```
void *ptr :
```

is saying that ptr is a pointer that can hold an address

cannot be dereferenced directly

i.e., void *ptr

```
int x = 10;
```

```
ptr = &x :  
printf("%d", *ptr); //invalid
```

you need to typecast the void pointer before dereferencing.

i.e.,

```
void *ptr  
int x = 10;  
ptr = &x;  
printf("%d", *(int*)ptr);  
          └─> typecasting
```

Here, `(int*)ptr` does type casting of void

`*(int*) ptr` dereferences the typecasted void pointer variable

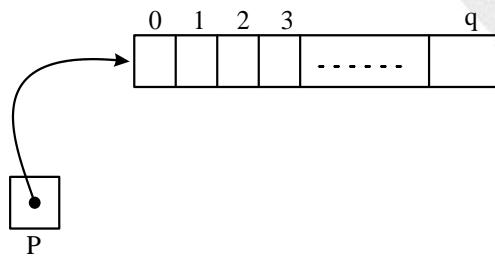
- Pointer arithmetic is not possible on void pointer.
i.e..... `ptr++`, `++ptr`, `ptr--`, `ptr-1`, `ptr + 1` is not allowed on void pointer.
- An uninitialized pointer holds a garbage value while a NULL pointer holds a defined value but not a valid address.

4.1.10 Memory leakage problem

- Whenever a programmer allocated memory dynamically then it is the responsibility of the programmer to free that memory after usage.
- Memory leakage problem occurs when a programmer allocates memory dynamically but does not de-allocate it after using it.
- It reduces the performance of the computer by reducing the amount of available memory.

4.1.11 Understanding Declaration

1. `int *(P[10])` : P is an array of 10 pointer to integer
2. `int (*P)[10]` : P is a pointer to an array of 10 integers



3. `int (*P)()` : P is pointer to a function that takes no argument and returns an integer.
4. `int (*P)(int, int)` : P is a pointer to a function that takes 2 integer arguments and returns an integer.
5. `int *P(char*)` : P is a pointer to a function that takes a pointer to character argument and return a pointer to integer.

4.1.12 Pointer to Functions / Function Pointer

- Just like normal pointers we can have pointers to functions.

`int(*p)(int, int)`

`P` is a pointer to a function that takes 2 integer arguments and returns an integer value.

- **Declaration of function pointer:** declaring a pointer to function is almost same as declaring the function except that in function pointer are prefix is with an asterik * symbol.

For example: if the function is

```
int add(int, int)
```

Declaration of a function pointer for add() function is :

```
int (*ptr)(int, int);
```

- How to call a function through function pointer : In order to call a function using function pointer, first we need to assign the address of function code to the pointer

Ex 1:

```
int add (int, int);
void main () {
    int (*ptr) (int, int);
    ptr = &add;      // ptr is a pointer to add() function
    printf ("The sum of 10 and 20 is", (*ptr )(10, 20)); //calling add()
}
int add (int, int);
```

Ex 2:

```
void main() {
    int (*ptr) (int, int);
    ptr = add ; // same as ptr = & add
    printf (" The sum of 10 and 20 is", (*ptr) (10, 20));
```

- A function can be called using following 4 ways using pointer to function.

<pre>int add (int int); void main() { int (*ptr) (int, int); ptr = &add; printf ("%d", (*ptr)(10, 20)); }</pre>	<pre>int add (int int); void main() { int (*ptr) (int, int); ptr = add; printf ("%d", (*ptr) (10, 20)); }</pre>
<pre>int add (int, int) ; void main() { int(*ptr) (int, int); ptr= &add; printf ("%d", (*ptr)(10, 20)); }</pre>	<pre>int add (int, int); void main() { int (*ptr) (int, int); ptr= add; printf ("%d", (*ptr) (10, 20)); }</pre>

- Using function pointer we are able to pass a function as an argument to other function and can also be returned from function.

Array of function pointer

```
int (*ptr[3])(int, int)
```

Ptr is an array of 3 pointers to a function that takes 2 arguments and returns an integer.



5

STRINGS

5.1 Strings

- Sequence of characters terminated by a null character ‘\0’.

Syntax:

```
char str_name[size];
```

- Initializing a string in c

➤ `char str[] = "Pankaj";`

Here, str is internally a pointer (holds the address of first character)

➤ `char str[20] = "Pankaj";`

Here, we redefine the size as 20 but we should always take one extra space along with size of string.
i.e atleast 7 size must be there to store “Pankaj” (6+1)

➤ `char str[7] = {'P', 'a', 'n', 'k', 'a', 'j', '\0'};`

must set the end character as ‘\0’

➤ `char str[] = {'p', 'a', 'n', 'k', 'a', 'j', '\0'};`

5.1.1 Reading a string

- `scanf()`:

when we use `scanf()` to read, we use `%s` as a format specifier without using “`&`” to access the variable address because an array name acts as a pointer.

```
#include <stdio.h>
int main (){
    char name [20];
    printf("enter your name \n");
    scanf("%s",name);
    printf("Hello %s",name);
}
```

Output:

```
Enter your name
Pankaj Sharma
Hello Pankaj // why not Hello Pankaj Sharma
```

The above code did not print Hello Pankaj Sharma as expected because `scanf` halts reading as soon as a whitespace or a newline character is encountered, and that is why it reads only Pankaj.

- In order to read a string containing space, we use `gets()` function. `gets()` ignores the whitespace & stops reading when a newline is found.

```
# include<stdio.h>
```

```
void main(){
    char name [20];
    printf("Enter your name :");
    gets(name);
    printf("Hello %s", name);
}
```

Output :

```
Enter your name: Pankaj Sharma
Hello Pankaj Sharma
```

- A string literal always represent base address of a string. i.e address of first character.
- “Pankaj” represents address of character ‘P’
Hence “Pankaj” [0] means ‘P’

$$\begin{aligned} \text{“Pankaj” [1]} &\equiv *(\text{“Pankaj”} + 1) \equiv *(\text{Address of ‘P’} + 1) \\ &\equiv *(\text{Address of ‘a’}) \\ &\equiv ‘a’ \end{aligned}$$

5.2 Strings and pointers

- Array name represents the address of its first element. Similar to character arrays, we can create a character pointer to represent a string that will hold the starting address i.e address of first character of string.

Example - 1

```
# include<stdio.h>
void main() {
    char *ptr = "pankaj";
    printf("%s",ptr);
}
```

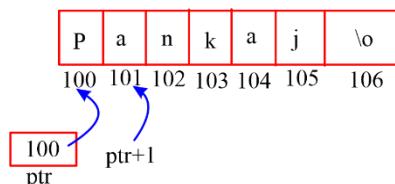
Output : pankaj



Example – 2

```
# include<stdio.h>
void main() {
    char *ptr = "Pankaj";
    printf("%s",ptr+1);
}
```

Output: ankaj



5.3 Predefined functions for string

- (1) **strlen ()**: returns the length of the string passed as an argument.
- (2) **strcpy ()**: copies the contents of one string to another. strcpy (S₁,S₂) copies the content of string S₂ to string S₁.
- (3) **strcmp ()**: compares the first string with the second string. If both are same it returns 0.
strcmp (str1,str2) returns 0 if both strings are same.
- (4) **strcat (S₁,S₂)**: concat S₁ string with S₂ string and the result i.e concatenation of S₁,S₂ is stored in S₁.

5.4 Important concepts

```
(1) # include<stdio.h>
void main(){
    char name [20] = "Pankaj";
    name = "Neeraj";           // Error
    printf("%s", name);
}
```

- Array name being a constant can't be presented at the left side of the assignment statement i.e it can't be L-value. You can't re-assign another string to array.

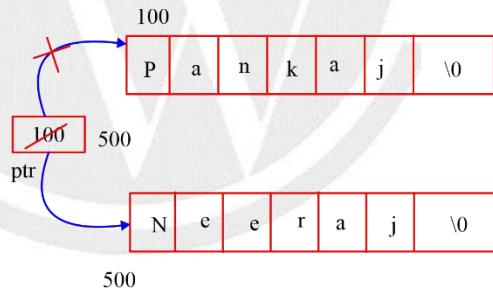
```
(2) # include<stdio.h>
void main(){
    char name [20] = "Pankaj";
    name [1] = 'u';
    printf("%s", name);
}
```

Output: Punkaj

- We are allowed to change content of array and hence we can update any character.

```
(3) # include<stdio.h>
void main(){
    char * ptr = "Pankaj";
    ptr = "Neeraj";
    printf("%s", ptr);
}
```

Output: Neeraj

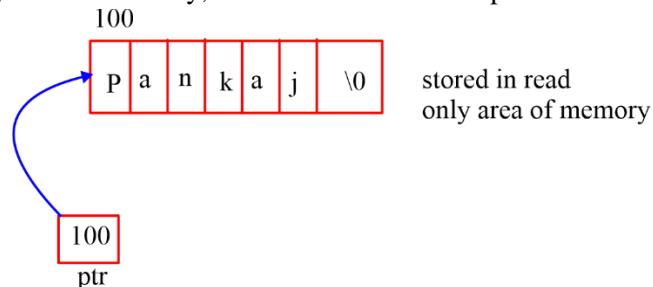


- Pointer ptr being a variable can hold different address at different time. Hence, we can assign another string to a character pointer any time.

```
(4) # include<stdio.h>
void main()
{
    char * ptr = "pankaj";
    ptr [1] = 'u'; // Invalid
    printf("%s", ptr);
}
```

- String literals are stored in read only memory area i.e "pankaj" is stored first & then the address of character 'p' is given to ptr.

- As they are stored in read only area of memory, we are not allowed to update them.



□□□



6

TYPES OF DATA STRUCTURE, ARRAY & LINKED LIST

6.1 Introduction

6.1.1 Linear data structure

Every element can have almost 2 neighbours. Ex. arrays, linked list, stack, queue.

6.1.2 Non-Linear data structure

Element can have more than 2 neighbours. Ex. Tree, graph.

6.2 Arrays

6.2.1 1-D array

Theoretically index can start from any integer value. Let A be a 1-D array of n elements and the size of each element is w bytes, then the address of A[i] element.

$$\text{Base Address } (A) + (i - \text{starting index}) * w$$

6.2.2 2-D Array

Analogous to matrix with rows and columns. Can be implemented in RMO (Row-major order) or CMO (Column-major order).

Let A[M] [N] be a 2-D array, where size of each element is w-bytes, then the address of A[i] [j] is :

(a) In RMO

$$\text{Address } (A[i] [j]) = \text{Base Address } (A) + [(j-\text{starting index}) + (i - \text{starting index}) \times M] \times w$$

(b) In CMO

$$\text{Base Address } (A) + [(i - \text{starting index}) + (j - \text{starting index}) \times M] \times w$$

6.2.3 Sparse Matrices

Most of the element of the matrix have 0 value and representing such matrix by a 2D array leads to wastage of memory and that is why, it is better than we will only store non-zero elements (Efficient method).

6.2.4 Lower Triangular Matrix

$$n \times n$$

$$A_{ij} = 0 \quad \text{if} \quad i < j$$

$$A_{ij} \neq 0 \quad \text{if} \quad i > j$$

For ex.

$$A = \begin{bmatrix} A_{11} & 0 & 0 & 0 \\ A_{21} & A_{22} & 0 & 0 \\ A_{31} & A_{32} & A_{33} & 0 \\ A_{41} & A_{42} & A_{43} & A_{44} \end{bmatrix}$$

6.2.5 Upper triangular Matrix

$$A = \begin{bmatrix} A_{11} & A_{12} & A_{13} & A_{14} \\ 0 & A_{22} & A_{23} & A_{24} \\ 0 & 0 & A_{33} & A_{34} \\ 0 & 0 & 0 & A_{44} \end{bmatrix}$$

6.2.6 Tri-diagonal Matrix

Matrix that has non-zero elements only in the main diagonal, diagonal just below main diagonal and diagonal just above main diagonal. All other elements are zero.

$$\begin{bmatrix} A_{11} & A_{12} & 0 & 0 & 0 \\ A_{21} & A_{22} & A_{23} & 0 & 0 \\ 0 & A_{32} & A_{33} & A_{34} & 0 \\ 0 & 0 & A_{43} & A_{44} & A_{45} \\ 0 & 0 & 0 & A_{54} & A_{55} \end{bmatrix}$$

6.2.7 Efficient method to store sparse matrix

Using Row-major order for storing lower triangular matrix, the element at index (i, j) can be represented as :

$$\text{Index of } A_{ij} = \left[\frac{i(i-1)}{2} + (j-1) \right]$$

Using column-major order for storing upper triangular matrix, the element at index (i, j) is sparse matrix can be represented as:

$$\text{Index of } A_{ij} = (i-j) + \left[(j-1)N - \frac{(j-i)(i-2)}{2} \right]$$

Matrix order

$N \times N$

Using Row-major order for storing upper triangular matrix, the element at index (i, j) in sparse matrix can be represented as :

$$\text{Index of } A_{ij} = (j-i) + \left[(i-1)N - \frac{(i-i)(i-2)}{2} \right]$$

Where N : Number of rows/columns

Using column-major order for starting upper-triangular matrix the element at index (i, j) can be represented as

$$\text{Index of } A_{ij} = \left[(i-1) + \frac{j(j-1)}{2} \right]$$

Using Row-major order for storing tri-diagonal matrix,

$$\text{Index of element } A_{ij} = 2i + j - 3$$

Using column-major order for storing tri-diagonal matrix,

$$\text{Index of element } A_{ij} = i + 2j - 3$$

6.3 Linked List

6.3.1 Linked List

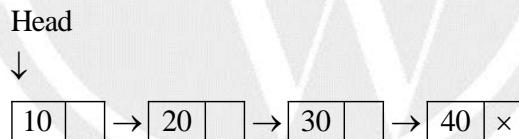
Linked list is collection of elements called node, where each node contains atleast 2 fields.

- (i) **Data field** : that contains information.
- (ii) **Link field** : contains address of next note.

6.4 Types of Linked List

6.4.1 Singly Linked List

Every node contains data and a pointer to the next node in the linked list, we can traverse the linked list only in forward direction.



- Head : Pointer that contains address of 1st node.
- Head contains NULL represent empty Linked List.
- If Ptr contains address of some note (Ptr is a pointer to node) then to go to the next node, we need.

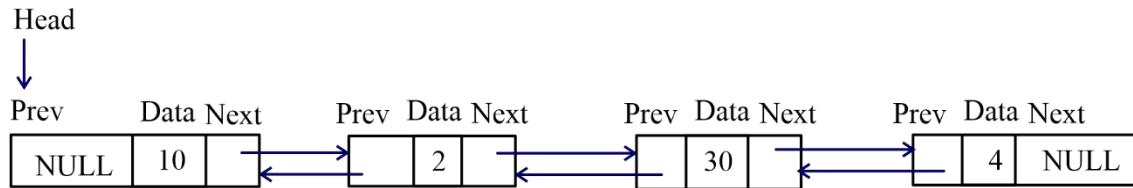
$$\text{Ptr} = \text{Ptr} \rightarrow \text{Link}$$
- Node can be implemented by structure in C

```

Struct Node {
    int data;
    Struct Node * Link;
}
  
```

6.4.2 Doubly Linked List

A two-way linked list in which every node contains a pointer to the next node as well as pointer to previous node in the list we can traverse it in backward direction also.



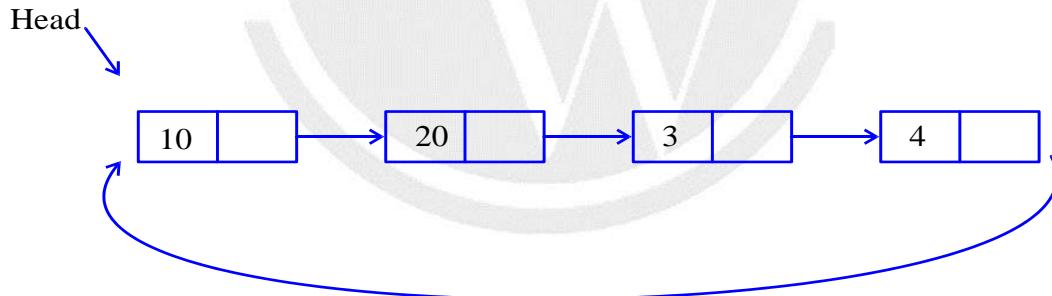
Structure of a Node :

```
Struct Node {  
    struct Node * Prev;  
    int data;  
    struct node * next;  
};
```

6.4.3 Circular Linked List

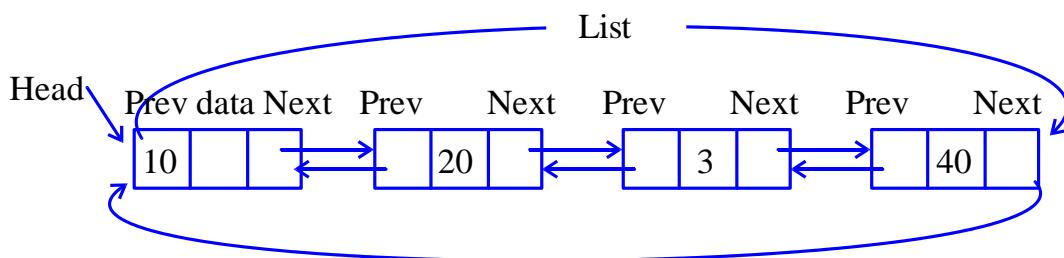
A linked list in which last node points back to the first node in the list.

- Circular linked list has no beginning and no end.



6.4.4 Doubly circular linked list

It is a 2-way circular linked list.

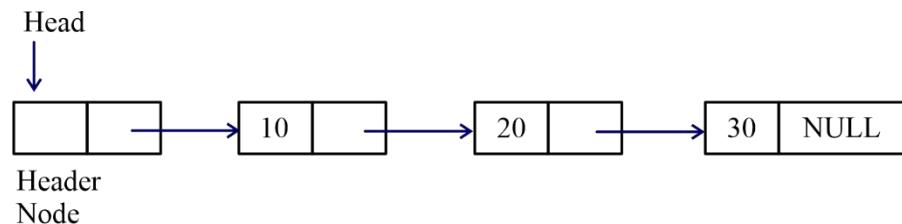


6.4.5 Header Linked List

A linked list that contains a special node called header node at the beginning of the list.

- Head will not point to first node.

- Head points to the header node.

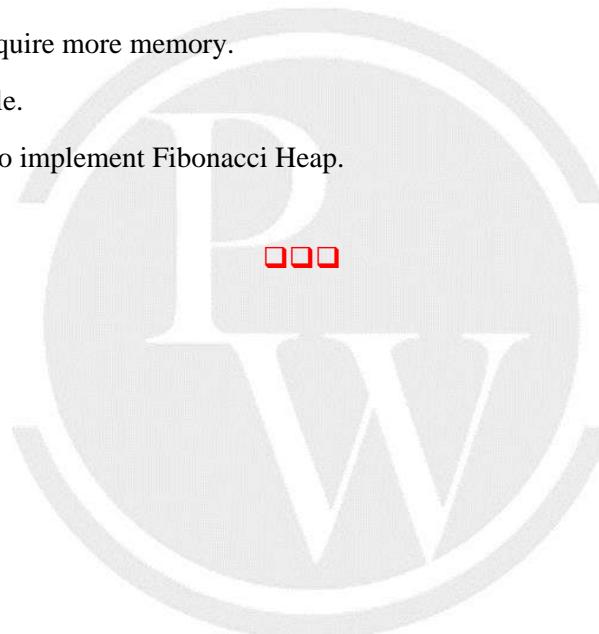


6.4.6 Applications

- used to implement other data structure link stack, queue data structure.
- used for dynamic memory allocation.

6.4.7 Advantages and Disadvantage

- Insertion and deletion are efficient.
- Using pointers in every node, require more memory.
- Random accessing in not possible.
- Circular linked list can be used to implement Fibonacci Heap.



7

STACK AND QUEUE

7.1 Introduction

Linear data structure in which both insertion and deletion operation are performed at one end called TOP of the stack. Works on LIFO (Last In First Out) Policy.

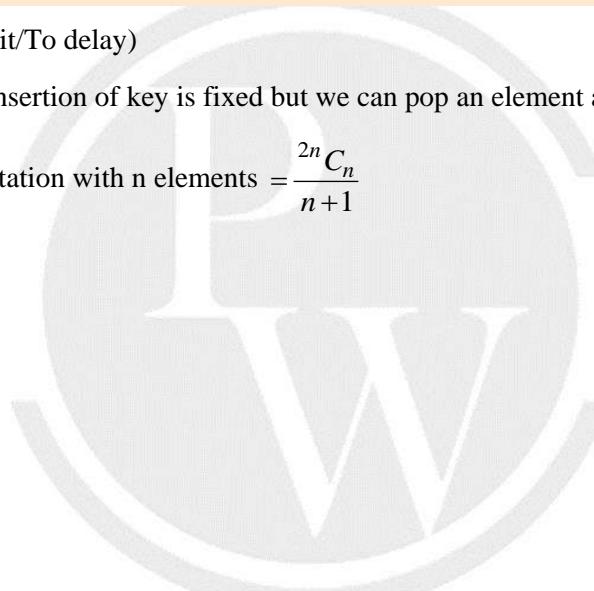
7.1.1 Application

To post-pone certain decision (To wait/To delay)

- **Stack Permutation :** Order of insertion of key is fixed but we can pop an element any time.

$$\text{Number of possible stack permutation with } n \text{ elements} = \frac{2^n C_n}{n+1}$$

- Infix to postfix
- Infix to prefix
- Prefix Evaluation
- Postfix Evaluation
- Recursion
- Tower of Hanoi



7.2 Queue Data Structure

Linear data structure in which insertion is done at one end called rear of the queue and deletion is performed at other end called front end of the queue.

Works on FIFO (First In First Out) policy.

7.2.1 Applications of Queue Data Structure

- Whenever a resource is being shared among many users.
- When data is transferred asynchronously.
- FCFS scheduling
- Algorithms like BFS.

7.3 Implementation

Can be implemented using arrays, linked list.

(a) Using Array :

- Easy to implement and memory efficient.
- Not dynamic

(b) Using Linked List :

- Dynamic
- Require extra memory because of pointer field.

7.4 Standard Operations

7.4.1 Stack

- (a) Push () : Insertion of an element onto stack.
- (b) Pop () : Deletion of an element.
- (c) Is_empty () : Returns true if stack is empty otherwise return false.

7.4.2 Queue

- (a) Enqueue :** Insertion of an element.
Performed at rear end of the queue.
- (b) Dequeue :** Deletion of an element.
Performed at front end of the queue.

7.5 Circular Queue

Extended version of simple queue in which last element is connected to the first element.

In simple queue, even though space is available, we are not able to insert a new element and declaring it a overflow. Circular queue solves this problem.

Queue Full

- (i) Front == 0 & & Rear == SIZE - 1 OR
- (ii) Front == (Rear + 1) % SIZE

7.6 Priority Queue

A queue in which a priority is associated with every element and elements are processed according to their priorities and if two elements have same priority then they are processed as per their arrival in the queue.



8

TREE DATA STRUCTURE

8.1 Introduction

8.1.1 Terminology

- (i) Internal Node : Node with atleast 1 child.
- (ii) Leaf Node : Node without any child.
- (iii) Root Node : Only node that does not have any parent.
- (iv) Complete binary Tree : Binary tree in which nodes are inserted from left to right at every level and we can not insert at node at $(K+1)$ th level until all levels upto K are fully filled.
- (v) Full Binary Tree : A binary tree in which every internal node has exactly two children and all the leaf nodes are at some level.
Binary tree that contains maximum number of nodes.
- (vi) Strict Binary Tree : A binary tree in which every internal node has exactly two children.
- (vii) Complete K-ary tree : Tree in which every internal node has exactly K-children.

8.1.2 Mathematical Result

- (i) For a binary tree of height h
 - Maximum number of nodes = $2^{(h+1)} - 1$
 - Minimum number of nodes = $h + 1$

- (ii) For a complete binary tree of height h
 - Maximum number of nodes = $2^{(h+1)} - 1$
 - Minimum number of nodes = 2^h

- (iii) For a full binary tree :

$$\text{Number of nodes} = 2^{h+1} - 1$$

$$(iv) \text{ Total number of unlabelled binary trees with } n \text{ nodes} = \frac{2nC_n}{n+1}$$

$$(v) \text{ Total number of labelled binary tree with no keys} = \frac{2nC_n}{n+1} \times n!$$

(vi) Total number of binary trees with a given preorder (n length) = $\frac{2^n C_n}{n+1}$

(vii) Total number of binary trees with a given preorder/postorder/inorder = $\frac{2^n C_n}{n+1}$

(viii) Number of binary trees with a given preorder and inorder = 1

(ix) Number of binary trees with a given postorder and inorder = 1

(x) Number of leaf nodes = Number of nodes with two children + 1

8.2 Binary Tree Traversal

(1) Preorder :

- (i) Process the root.
- (ii) Traverse the left subtree of root in preorder.
- (iii) Traverse the right subtree of root in preorder.

(2) Inorder

- (i) Traverse the left subtree of root in inorder.
- (ii) Process the root.
- (iii) Traverse the right subtree of root in inorder.

(3) Post-order

- (i) Traverse the left subtree of root in post-order.
- (ii) Traverse the right subtree of root in post-order.
- (iii) Process the root.

In-order

- **Morrison Traversal :**

In-place traversal of a binary tree

No recursion

Constant extra space.

8.3 Binary Search Tree

A binary tree in which every node satisfy the property that all the keys in the left subtree of the node are smaller than node's key and all the keys in the right subtree of the node are greater than node's key.

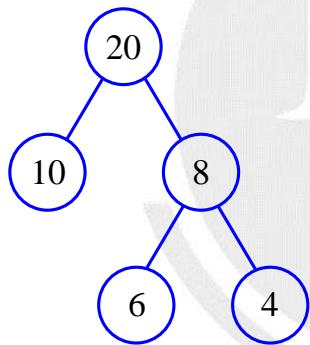
- Inorder traversal of a binary search tree is ascending order of keys in the BST.
- Insertion in a BST
 - (a) $O(\log n)$ best case
 - (b) $O(n)$ worst case.

- Deletion from a BST :
 - (a) Deletion of leaf node : can be done directly
 - (b) Deletion of node with one child : copy the child to the node and delete the child.
 - (c) Deletion of a node with two children : first find the inorder successor of the node, copy the contents of this successor to the node and delete the inorder successor.
We can also use inorder predecessor.
- TC : O (log n) best case
O (n) worst case

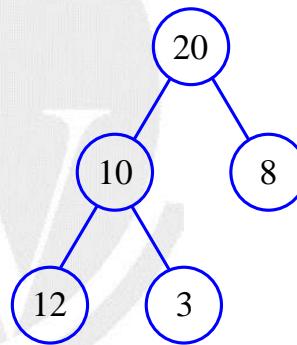
8.4 Heap

A complete binary tree in which every node satisfy heap property (min heap or max heap)

- (i) Min heap : A complete binary tree in which every node satisfy the property that the value of the node is smaller than both its children.
- (ii) Max heap : A complete binary tree in which every node satisfy the property that the value in the node is greater than both its children.



Not a heap



Not a heap

- Construction of Heap :
 - (i) By inserting keys one after another : O (n log n)
 - (ii) Using build-heap method/heapify algo : O(n)
- Number of min-heaps possible with n keys
 $T(n) = T(k) \cdot T(n - k - 1) \cdot {}^{n-1}C_k$
K : Number of elements in the left subtree of root node.
- Deletion :
 - (1) Swap A[1], A[n]
 - (2) Apply Heapify on A[1] considering there are only (n - 1) nodes.
 $T.C = O (\log n)$

- Heapsort (Assuming max heap)
 - (i) Build Max Heap $O(n)$
 - (ii) Replace root element with last element ($A[1] \leftrightarrow A[n]$) reduce the heap size by 1 and then apply heapify at root node.
 - (iii) Repeat step 2 while heap size is greater than 1.

8.5 AVL Search Tree

- Height balanced BST.
- Balance factor (B_f) is given by

$$B_f = |h_L - h_R| \leq 1$$

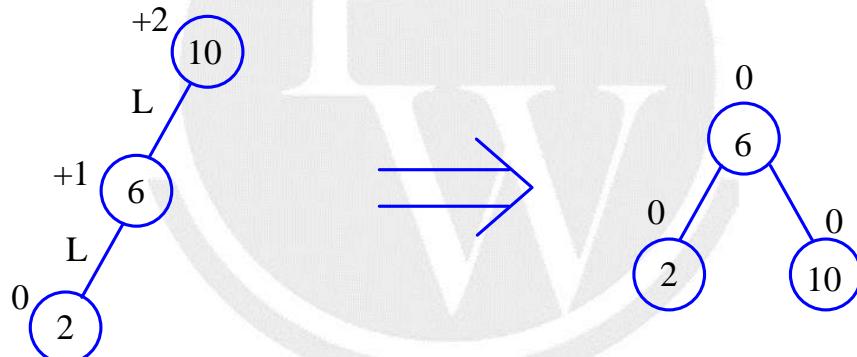
i.e., Balancing factor of a node can be $+1, -1$ or 0 .

8.5.1 Insertion

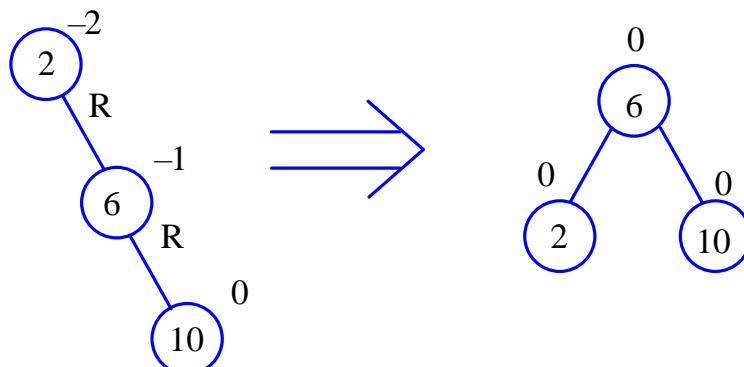
To ensure that the tree remains AVL tree, after insertion some re-balancing is performed i.e., certain rotations are needed.

8.5.2 Types of Rotation

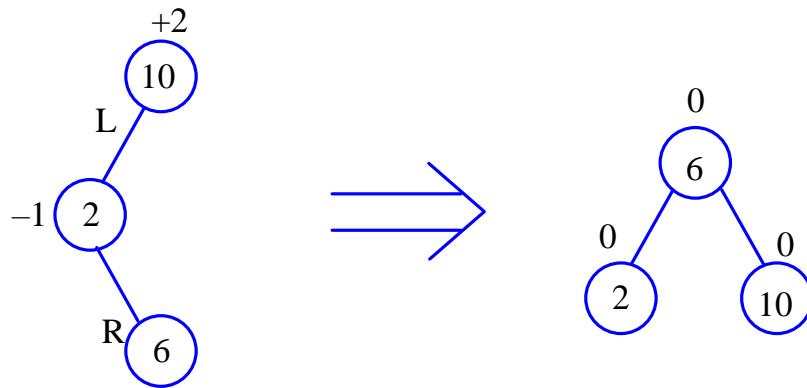
- (i) Left-Left (LL) Rotation :



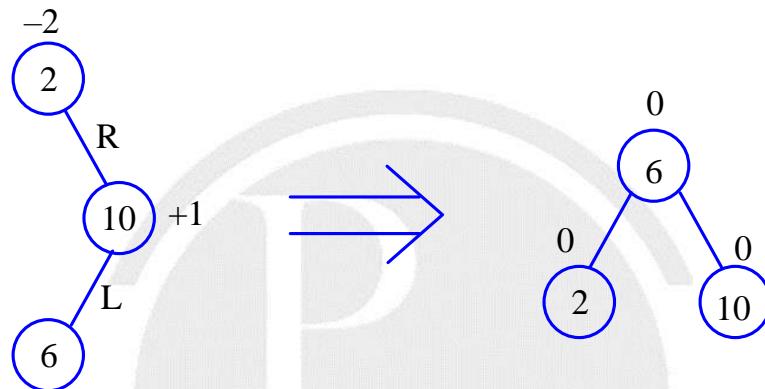
- (ii) Right-Right (RR) Rotation :



- (iii) Left-Right (LR) Rotation :



(iv) Right-Left (RL) Rotation :



- LL, RR are single rotation.
- LR, RL are double rotation.
- Just work no tri-node structure.
- Minimum number of nodes in an AVL tree of height h is given by recurrence.

$$n(h) = 1 + n(h-1) + n(h-2) \quad h \geq 2$$

$$n(0) = 1$$

$$n(1) = 2$$

i.e., it also forms a Fibonacci series

1, 2, 4, 7, 12,



9.1 Introduction

9.1.1 Breadth First Search

- Uses a queue data structure.
- To find whether a graph is connected or not.
- To find number of connected components in a given graph (Breadth First traversal is used)
- To detect cycle in a graph.
- To find whether a given graph is bipartite or not.
- T.C. = $O(V + E)$ using Adjacency List
= $O(V^2)$ using Adjacency matrix.
- To find shortest path in undirected graph without weights.

9.1.2 Depth First Search

- Uses stack.
- To detect a cycle in a graph.
- To find whether a graph is connected or not.
- To find whether given graph is bipartite or not.
- To find number of connected components in the graph.
- To find bridges in the graph.
- To find articulation points in the graph.
- To find topological sort of a graph.
- To find biconnected components.
- To find strongly connected components.



10

HASHING

10.1 Introduction

- Searching technique.
- Mapping keys into hash table using a hash function.
- Efficiency based on hash function used for mapping.

10.1.1 Terminology

(i) Collision : When two keys mapped to same location, collision occurs.

$H(k_1) = H(k_2)$, where $H(k)$ is the hash function used.

(ii) Load factor $\lambda = \frac{\text{no. of keys}}{\text{Hash table size}}$

10.1.2 Collision resolution technique

(i) Open addressing (closed hashing)

- (a) Linear probing
 - (b) Quadratic probing
 - (c) Double hashing
- $\lambda \leq 1$

(ii) Separate chaining

10.1.3 Linear Probing

Whenever there is a collision at memory location ‘L’, then we will search for empty slot sequentially starting from ‘L’ then $L + 1, L + 2, L + 3, \dots$

10.1.4 Problem with linear Probing

Linear probing suffers with primary clustering problem, consecutive keys forms a cluster and the time to find a free slot increases.

10.1.5 Quadratic Probing

Hash function $h(x) = x \bmod m$. It leads to a collision and it is i^{th} collision for key x then the collision resolution function is given as:

$$H(x, i) = (h(x) + i^2) \bmod m$$

Searching order : $L, L + 1, L + 4, L + 9, \dots$

- Free from primary clustering
- Suffers with secondary clustering problem.

10.1.6 Double Hashing : Using 2 hash functions

$(\text{hash} - 1(\text{key}) + i * \text{hash-2}(\text{key})) \% m$

M : table size

- Hash-2 (key) : Secondary hash function must not give output 0.
- * One of the best probing.
 - * Uniform distribution.
 - * Free from primary and secondary clustering.
 - * Computation overhead because of two hash function computation.

10.2 Chaining

Uses the concept of linked list (chain) when more than one element are hashed to same location (slot) then elements are inserted into a singly linked list (chain).

- Deletion is easy compared to open addressing.
- Deleting in open addressing require rehashing remaining keys.



GATE Exam 2025?



SHRESHTH BATCH

ME | CE | EE | EC | CS & IT

- Live Classes & Recorded Videos
- DPPs & Solutions
- Doubt Solving
- Batches are available in *Hindi*

Weekday & Weekend
Batches Available

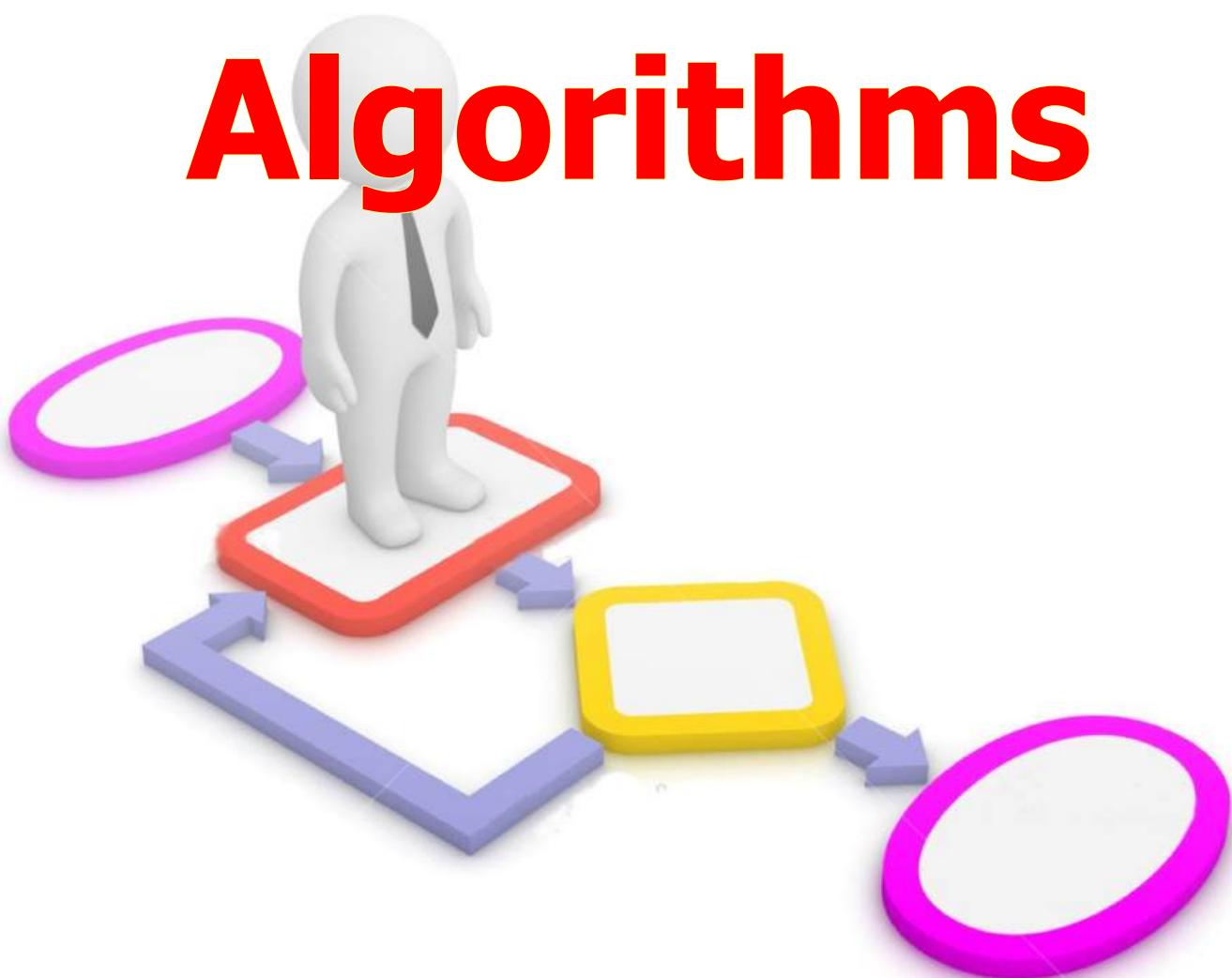


JOIN NOW!



Physics W

Algorithms



Algorithm

INDEX

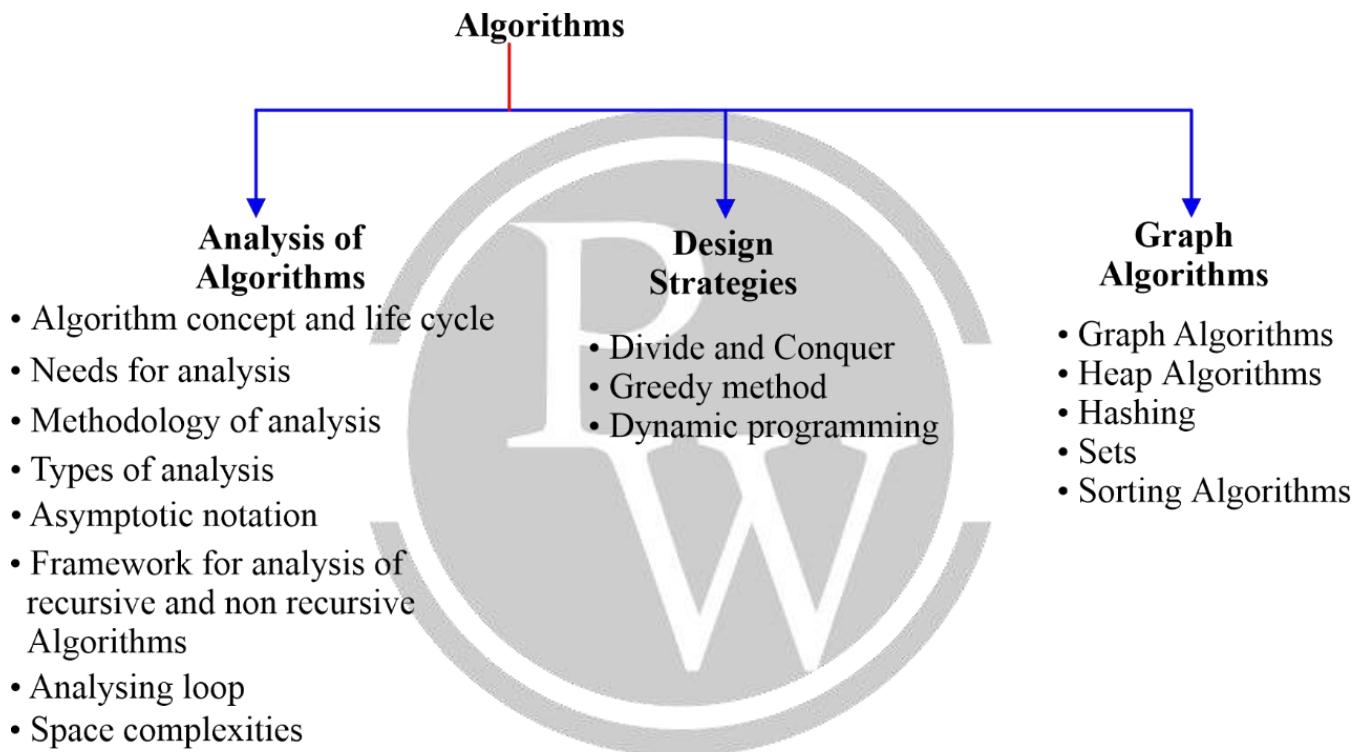
- | | | |
|----|---------------------------|-------------|
| 1. | Asymptotic Notation | 6.1 – 6.18 |
| 2. | Divide and Conquer..... | 6.19 – 6.25 |
| 3. | Greedy Technique | 6.26 – 6.29 |
| 4. | Dynamic Programming..... | 6.30 – 6.34 |



1

ASYMPTOTIC NOTATION

1.1 Introduction of Course

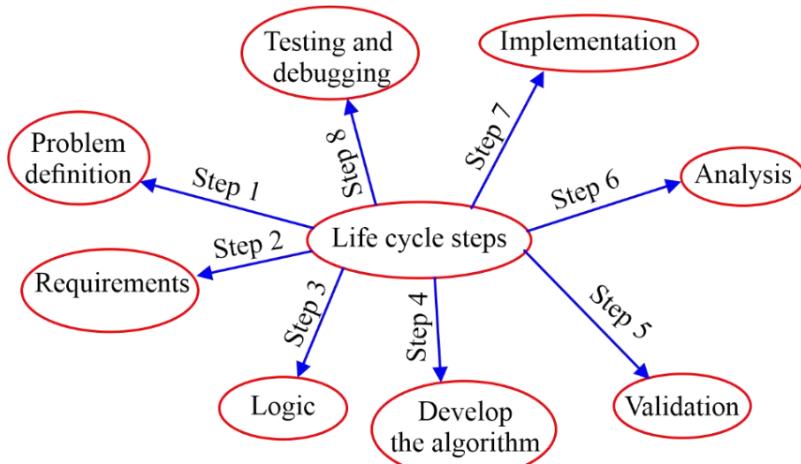


1.2 Algorithm Concept and Life Cycle Steps

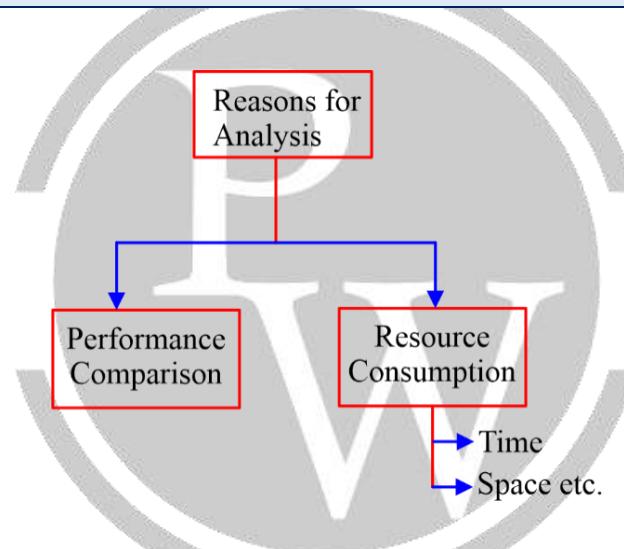
1.2.1 Algorithm

- An Algorithm consists finite number of steps to solve any problem.
- Every step involves some operations and each operation must be definite and effective.

1.2.2 Life Cycle Steps



1.3 Needs of Analysis



In performance comparison comparing different algorithms for optimal solution.

1.3.1 Time Complexity

Time complexity of an algorithm quantifies the amount of time taken by an algorithm to run as a function of the input size.

1.3.2 Space Complexity

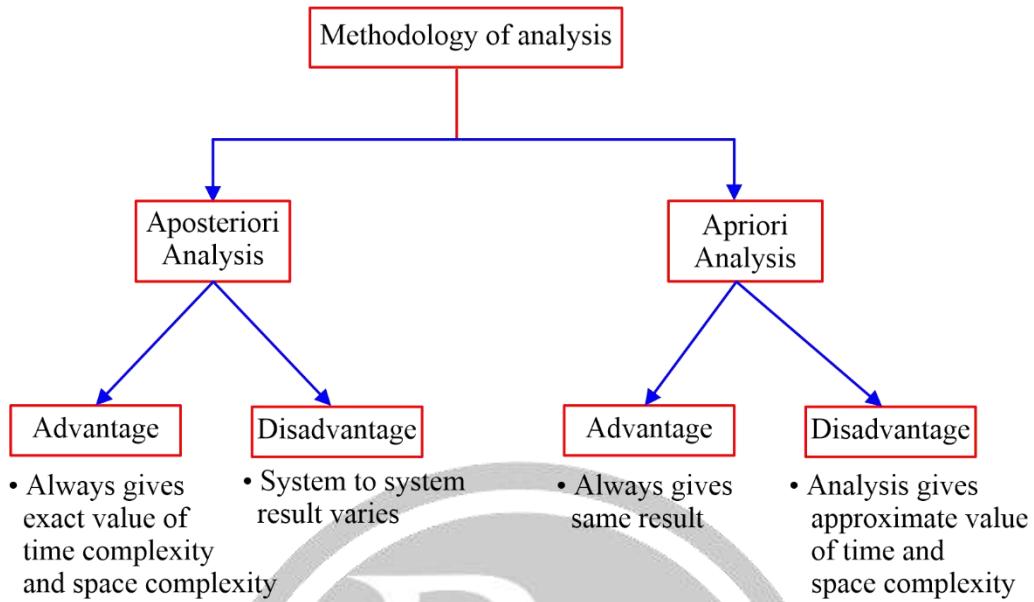
Space complexity of an algorithm quantifies the amount of space or memory taken by an algorithm to run as a function of input size.

Note:

To find the time complexity of an algorithm, find the loops and also consider larger loops.

Space complexity is dependent on two things input size and some extra space (stack space link, space list etc).

1.4 Methodology of Analysis



1.5 Types of Analysis

Worst Case

The input class for which the algorithm does maximum work and hence, take maximum time.

Best Case

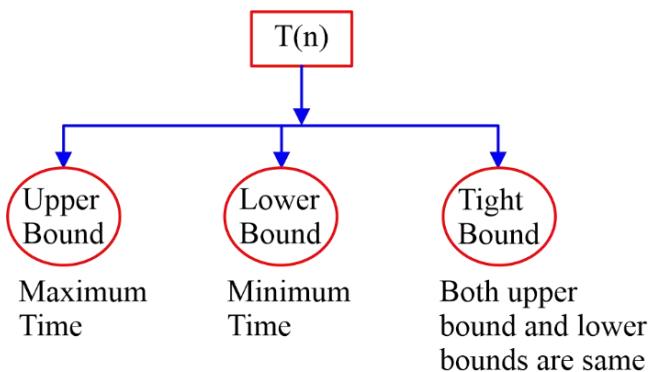
The input class for which the algorithm does minimum work hence, take minimum time.

Average Case

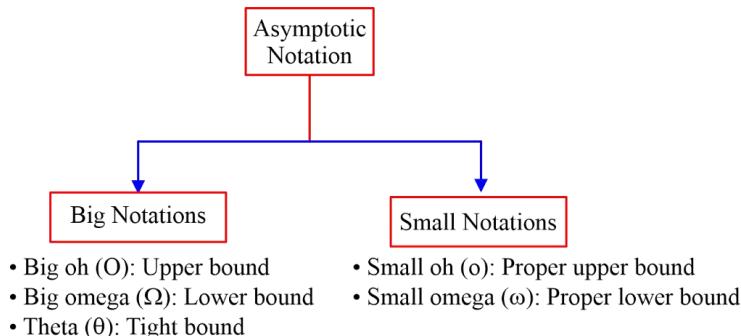
Average case can be calculated from best case to worst case.

1.6 Asymptotic Notations

Suppose, $T(n)$ be a function of time for any algorithm.



1.7 Types of Asymptotic Notations



1.7.1 Big O - Notation

Two Functions $f(n), g(n)$

$$f(n) = O(g(n))$$

When the growth of $g(n)$ is same or higher than $f(n)$ like $a \leq b$

Example:

$$f(n) = 3n + 10, g(n) = n^2 + 2n + 5$$

$$f(n) = O(g(n))$$

1.7.2 Ω - Notation

$$f(n) = \Omega(g(n))$$

$$\therefore f(n) \geq C \cdot g(n)$$

$$(a \geq b)$$

Example: $3^n = \Omega(2^n)$

1.7.3 Θ - Notation

If $f(n) \leq g(n)$

And

$$f(n) \geq g(n)$$

$$f(n) = g(n)$$

$$\therefore f(n) = \Theta(g(n))$$

Example:

$$f(n) = 2n^2, g(n) = n+10$$

$f(n) > g(n)$ here

$$\text{so, } f(n) = \Omega(g(n)) \text{ or } g(n) = O(f(n))$$

1.7.4. Properties with respect to asymptotic notations

	Reflexive	Symmetric	Transitive	Transpose symmetric
Big oh (O)	✓	✗	✓	✓
Big omega (Ω)	✓	✗	✓	✓
Theta (θ)	✓	✓	✓	✗
Small oh (o)	✗	✗	✓	✓
Small omega (ω)	✗	✗	✓	✓

Example 1. Consider the following function

$$f(n) = \sum_{p=1}^n p^3 = q$$

Which of the following is/are true for 'q'

- (a) $\theta(n^4)$ (b) $\theta(n^5)$ (c) $O(n^5)$ (d) $\Omega(n^3)$

Solution: (a, c, d)

$$\begin{aligned} f(n) &= \sum_{P=1}^n P^3 \\ &= 1^3 + 2^3 + 3^3 + 4^3 \dots \dots \dots + n^3 \\ &= \left(n \left(\frac{n+1}{2} \right) \right)^2 \\ &= O(n^4) \text{ or } \Omega(n^4) \\ &= \theta(n^4) \end{aligned}$$

Example 2. Consider the following functions:

$$f(n) = \sum_{P=1}^n P^{1/2} = q$$

Find the value of q in terms of asymptotic notation.

$$\begin{aligned} \text{Solution: } f(n) &= \sum_{P=1}^n P^{1/2} \\ &= 1 + (2)^{1/2} + (3)^{1/2} + \dots \dots \dots \\ &= \frac{2}{3} \left[n^{3/2} - 1 \right] \\ &= \frac{2}{3} n^{3/2} - \frac{2}{3} \\ &= O(n^{1.5}) \\ &= O(n\sqrt{n}) \end{aligned}$$

Example 3. Arrange the following functions in increasing order.

$$\begin{aligned} f_1 &= n \log n, f_2 = \sqrt{n}, f_3 = 2^n, f_4 = 3^n, f_5 = n!, f_6 = n^n, f_7 = \sqrt{\log n}, f_8 = 100n \log n \\ \rightarrow f_7 < f_2 < f_1 &= f_8 < f_3 < f_4 < f_5 < f_6 \end{aligned}$$

Example 4. Arrange the following functions in increasing order.

$$f_1 = 10, f_2 = \sqrt{n}, f_3 = \log \log n, f_4 = (\log n)^2, f_5 = n^2$$

$$f_6 = n \log n, f_7 = n!, f_8 = 2^n, f_9 = n^n, f_{10} = n^2 \log n$$

$$\rightarrow f_1 < f_3 < f_4 < f_2 < f_6 < f_5 < f_{10} < f_8 < f_7 < f_9$$

Example 5. Arrange the following functions in increasing order.

$$f_1 = \log \log n \quad f_9 = n \log \log n$$

$$f_2 = \log n \quad f_{10} = n^2 \log n$$

$$f_3 = (\log n)^2 \quad f_{11} = n^3$$

$$f_4 = \sqrt{\log n} \quad f_{12} = 2^n$$

$$f_5 = n^{1/10} \quad f_{13} = e^n$$

$$f_6 = n \quad f_{14} = n!$$

$$f_7 = n^2 \quad f_{15} = n^n$$

$$f_8 = n \log n \quad f_{16} = n^{3/2}$$

$$f_1 < f_4 < f_2 < f_3 < f_5 < f_6 < f_9 < f_8 < f_{16} < f_7 < f_{10} < f_{11} < f_{12} < f_{13} < f_{14} < f_{15}$$

$$a^{\log_b c} \Leftrightarrow c^{\log_b a}$$

$$\therefore 2^{\log_2 n} \Leftrightarrow n^{\log_2 2} = n$$

Example 6. Arrange the following functions in increasing order.

$$f_1 = n!, f_2 = n^n$$

$$f_1 = n \times (n-1)(n-2) \times \dots \times 3 \times 2 \times 1$$

$$f_2 = n \times n \times n \times n \times \dots \times n \times n \times n$$

$$f_2 > f_1$$

$$\therefore f_1 = O(f_2)$$

$$2^n < 3^n < 4^n < n! < n^n$$

Question.

Which of following is TRUE?

- | | |
|--|-------|
| (1) $2^{\log_2 n} = O(n^2)$ | TRUE |
| (2) $n^2 \cdot 2^{3\log_2 n} = O(n^5)$ | TRUE |
| (3) $2^n = O(2^{2n})$ | TRUE |
| (4) $\log n = O(\log \log n)$ | FALSE |
| (5) $\log \log n = O(n \log n)$ | TRUE |

Solution:

$$(1) \quad 2^{\log_2 n} = O(n^2)$$

$$= n^{\log_2 2}$$

$$= n$$

$$= n = O(n^2)$$

(2) $n^2 \cdot 2^{3\log_2 n} = O(n^5)$

$$= n^2 \cdot n^{3\log_2 2}$$

$$= n^2 \cdot n^3$$

$$= n^5$$

$$= n^5 = O(n^5)$$

(3) $2^n = 2^{2n}$

$$2^n = 2^n \cdot 2^n$$

$$2^n \leq 2^{2n}$$

$$2^n = O(2^{2n}) \text{ True}$$

(4) $\log n > \log \log n$

$$\log n \neq O(\log n)$$

False

(5) $\log \log n \leq n \log n$

$$\log \log n = O(n \log n)$$

True



1.8. Analysis of an Algorithm

Without Loop

Algorithms

Interactive Algorithm

Recursive Algorithm

1.8.1 Without loop

Example:

```
int fun (in + n)
{
    return n*(n+1)/2;
}
```

Solution.

Here 1 multiply, 1 division, 1 addition

$\therefore O(1)$ [no loops, no recursion]

1.8.2. Iterative Algorithm Analysis

Example 1:

```
for (i=1; i ≤ n; i=i*2)
printf("Sushil")
```

Solution.

$i=1, 2, 2^2, 2^3 \dots, 2^k$

\rightarrow

$2^k \leq n$

$k \log 2 \leq \log n$

$$k \leq \frac{\log n}{\log 2}$$

$\therefore k \leq \log_2 n$

$$k = \lfloor \log_2 n \rfloor$$

So, this will execute $\lfloor \log_2 n \rfloor + 1$ time and Complexity $O(\log_2 n)$

Example 2:

```
For (i=1; i ≤ n; i=i*3)
printf("Aaveg");
```

Solution.

So, this will execute $\lfloor \log_3 n \rfloor + 1$ time and complexity $O(\log_3 n)$

➤ $i = 1 \rightarrow 2 \rightarrow 4 \rightarrow 8 \rightarrow 16 \rightarrow \dots \rightarrow n$
 $i = n \rightarrow n/2 \rightarrow n/4 \rightarrow n/8 \rightarrow \dots 1$

Example 3:

```
for (i = 1; i ≤ n; i++)
{
    for (j=1; j ≤ 10; j++)
    {
        printf("Dhananjay");
    }
}
```

Solution.

This will execute $10 \cdot n$ times and complexity $O(n)$

Example 4:

```
for (i = 1; i <= n; i = i*3)
    for (j = 1; j ≤ n; j++)
        printf("Prapti");
```

Solution.

Total $n(\lfloor \log_3 n \rfloor + 1)$ time execute and Complexity = $O(n \log_3 n)$

1.8.3. Recursive Algorithm Analysis

Example 1:

```

void fun (in + n)      T(n)
{
    if (n > 0)          1 compare; C1 time
    {
        if ("% d", n);   ← C2 time
        fun (n - 1);    ← T(n-1)
    }
}

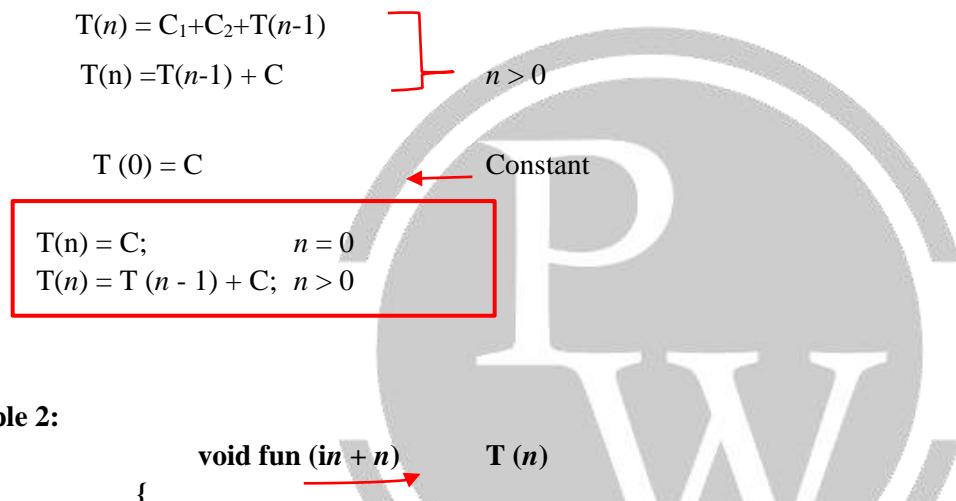
```

Let $T(n)$ be the Complexity time taken by algo for n size i/p

Solution.

$$T(n) = C_1 + C_2 + T(n-1)$$

$$T(n) = T(n-1) + C$$



Example 2:

```

void fun (in + n)      T(n)
{
    if (n > 0)          ← C1 time
    {
        for (i = 1; i <= n; i + 1) ← n time
            printf("Hello");
        fun (n - 1);    ← T(n - 1)
    }
}

```

Solution.

$$T(n) = C_1 + n - 1 + T(n - 1)$$

$$= T(n - 1) + n$$

$$T(0) = C$$

Example 3:

```

void fun (in + n)      T(n)
{
    if (n > 0)          ← C1
    {
        for (i = 1; i <= n; i = i*2) ← ⌊ log2 n ⌋
    }
}

```

```

        printf("Divyajyoti");
        fun (n - 1); ← T (n - 1)
    }
}

```

Solution. $T(n) = T(n - 1) + O(\log_2 n)$; $n > 0$

or

$$T(n) = T(n - 1) + \log_2 n$$

$$\begin{aligned} T(0) &= C \\ T(0) &= O(1) \end{aligned}$$

1.9 Solving Recurrence Relation

1.9.1 Substitution Method

Example: (1)

$$T(n) = T(n - 1) + C$$

$$T(1) = C$$

n size on problem $n - 1$ size x_1 convert them

$$T(n) = T(n - 1) + C$$

$$\begin{array}{l} \downarrow \\ [T(n - 2) + C] + C \end{array}$$

$$T(n) = T(n - 2) + 2C$$

$$\begin{array}{l} \downarrow \\ = T(n - 3) + 3C \end{array}$$

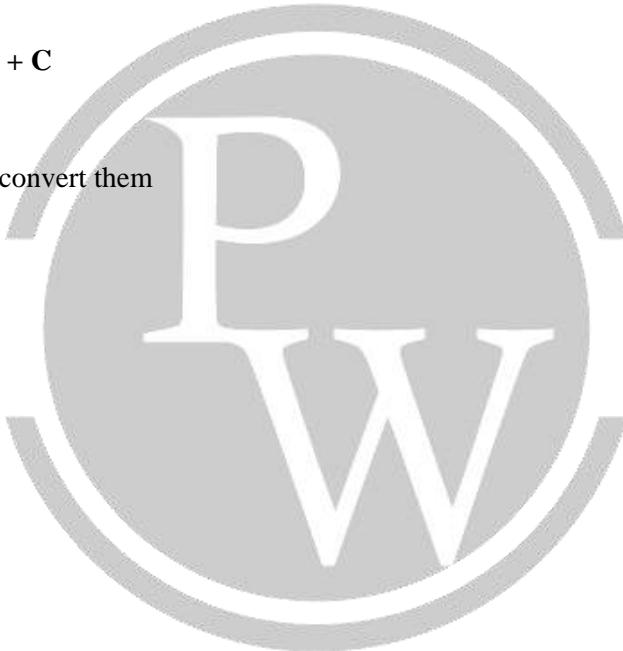
$$T(n) = T(n - k) + kC$$

$$\therefore n - k = 1$$

$$T(n) = T(1) + (n - 1)C$$

$$= C + (n - 1)C$$

$$T(n) = O(n)$$



Example (2)

$$T(n) = T(n - 1) + C \cdot n$$

$$T(1) = C$$

Solution.

$$\begin{aligned} \therefore T(n) &= T(n - 1) + C \cdot n \\ &= [T(n - 2) + C \cdot (n - 1)] + C \cdot n \\ &= [T(n - 3) + C \cdot (n - 2)] + C \cdot (n - 1) + C \cdot n \\ &= T(n - 3) + (n - 2) \cdot C + (n - 1) \cdot C + n \cdot C \\ &= T(n - k) + C \cdot (n - k + 1) + C \cdot (n - k + 2) + \dots + C \cdot (n - k + k) \\ \therefore n - k &= 1 \end{aligned}$$

$$T(n) = T(1) + T(2) + C(3) + C(4) + \dots + C(n - 1) + C(n)$$

$$= C + C(2) + (3)C + 4(C) + \dots + (n - 1)C + (n)C$$

$$= C[1 + 2 + 3 + \dots + n]$$

$$= C \cdot n \frac{(n+1)}{2}$$

$$= O(n^2)$$

Example (3)

$$T(n) = T(n/2) + C$$

$$T(1) = 1$$

Solution.

$$T(n) = T(n/2) + C$$

$$= [T(n/2^2) + C] + C$$

$$= T(n/4) + 2C$$

$$= T(n/2^3) + 3C$$

$$T(n) = T(n/2^k) + kC$$

$$= (n/2^k) = 2$$

$$T(n) = T(2) + (\log_2 n - 1)C$$

$$= 1 + (\log_2 n - 1)C$$

$$= O(\log n)$$


Example (4)

$$T(1) = 1$$

$$T(n) = 2T(n/2) + C$$

Solution.

$$T(n) = 2 \left[2T\left(\frac{n}{2^2}\right) + C \right] + C$$

$$= 2^2 T\left(\frac{n}{2^2}\right) + 2^2 C + C$$

$$= 2^2 \left[2T\left(\frac{n}{2^3}\right) + C \right] + 2C + C$$

$$= 2^3 T\left(\frac{n}{2^3}\right) + 2^2 C + 2C + C$$

$$= 2^k T\left(\frac{n}{2^k}\right) + 2^{k-1} C + 2^{k-2} C + \dots + 2^1 \cdot C + C$$

$$\begin{aligned}
 \frac{n}{2^k} &= 1 \quad \therefore n = 2^k \\
 \rightarrow T(n) &= nT(1) + 2^{k-1} \cdot C + 2^{k-2} \cdot C + \dots + 2C + C \\
 &= 2^k + 2^{k-1} \cdot C + 2^{k-2} + \dots + 2C + C \\
 &= 2^k + C(2^{k-1} + 2^{k-2} + \dots + 2^1 + 2^0) \\
 &= 2^k + C \frac{(2^k - 1)}{2 - 1} \\
 &= 2^k + C(2^k - 1) \\
 &= 2^k + 2^k \cdot C - C \\
 &= n \cdot C \\
 &= O(n)
 \end{aligned}$$

1.9.2 Master's Method

$$T(n) = aT\left(\frac{n}{b}\right) + \Theta(n^k (\log n)^p)$$

$a \geq 1, b > 1, k \geq 0, p = \text{real number}$

$\text{If } a > b^k \text{ or } \log_b a > k$
 $T(n) = \Theta(n^{\log_b a})$

Question 1. $T(n) = 2T\left(\frac{n}{2}\right) + (n)^0 \log n$

Solution. $a = 2, b = 2, k = 0$
 $a > b^k; 2 > 2^0; 2 > 1$
 $T(n) = \Theta(n)$

Question 2. $T(n) = 2T\left(\frac{n}{2}\right) + n$

Solution. $a = 2, b = 2, k = 1, p = 0$
 $T(n) = \Theta(n \cdot \log n)$

Question 3. $T(n) = 2T\left(\frac{n}{2}\right) + n \log n$

Solution. $a = 2, b = 2, k = 1, p = 1$

$$\therefore T(n) = \Theta(n (\log n)^2)$$

(b) If $p < 0$ then $T(n)$
 $T(n) = O(n^k)$

Question 4. $T(n) = T\left(\frac{n}{2}\right) + C$

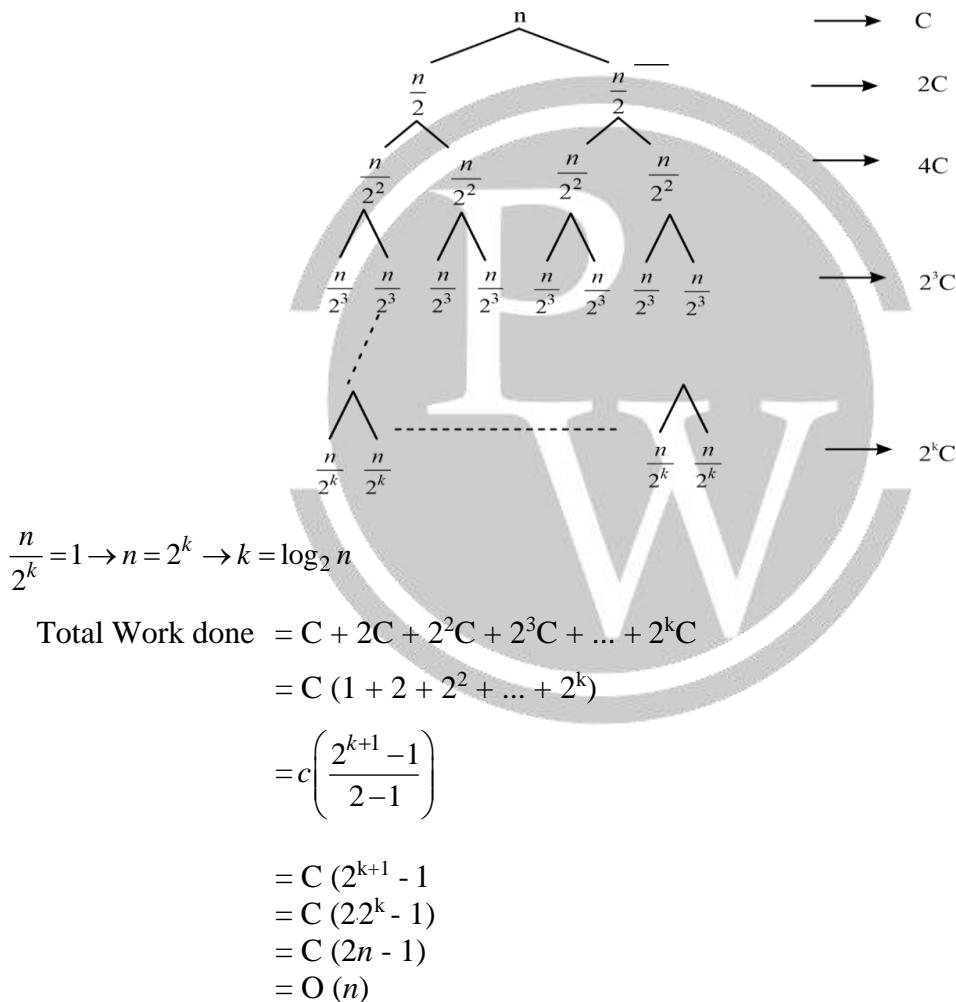
Solution.

$$T(n) = \Theta(n^2 \log n)$$

1.9.3. Recursive Tree

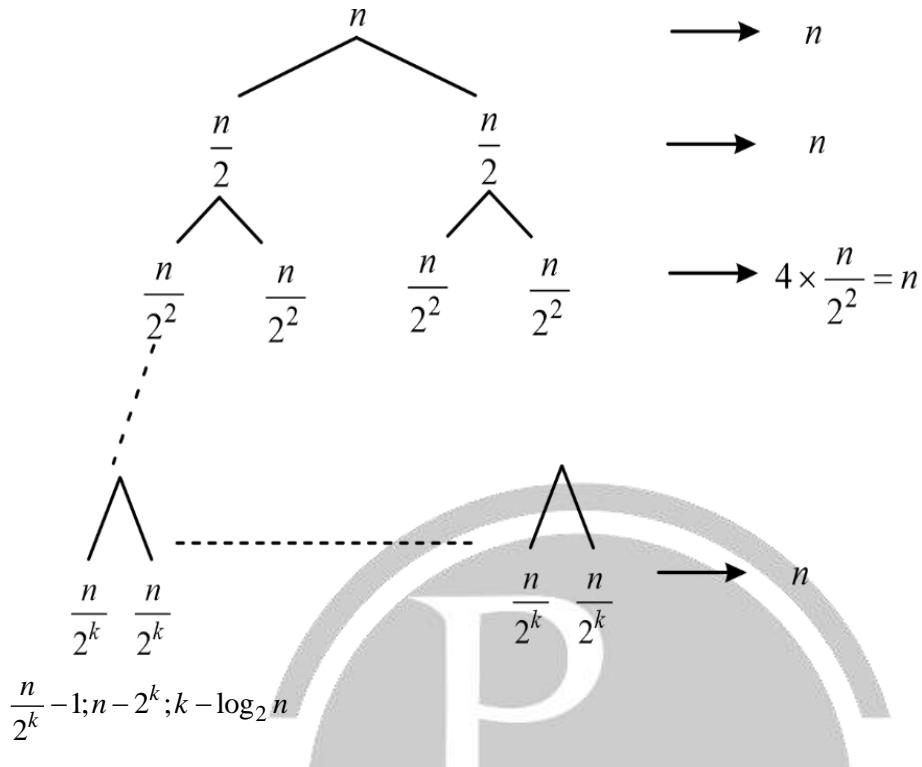
$$(1) \quad T(n) = 2T\left(\frac{n}{2}\right) + C$$

$$T(1) = C$$





$$(2) \quad T(n) = 2T\left(\frac{n}{2}\right) + n$$



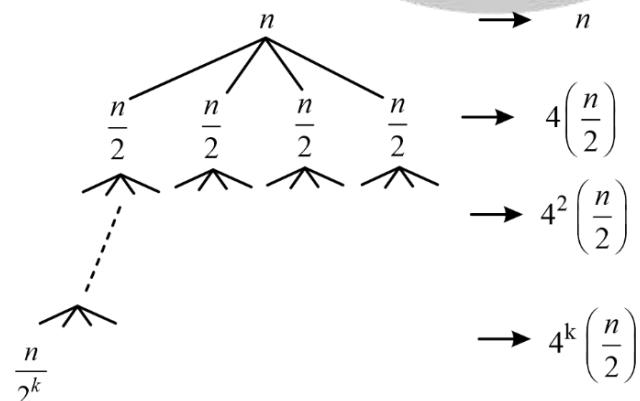
$$\therefore n + n + n + \dots + n$$

$$= k n$$

$$= n \log_2 n$$

$$= O(n \log_2 n)$$

$$(3) \quad T(n) = 4T\left(\frac{n}{2}\right) + n$$



$$n = 2^k, k = \log_2 n$$

$$= n + 4\left(\frac{n}{2}\right) + 4^2\left(\frac{n}{2}\right) + \dots + 4^k\left(\frac{n}{2}\right)$$

$$= n \left[1 + 2 + 2^2 + 2^3 + \dots + 2^k \right]$$

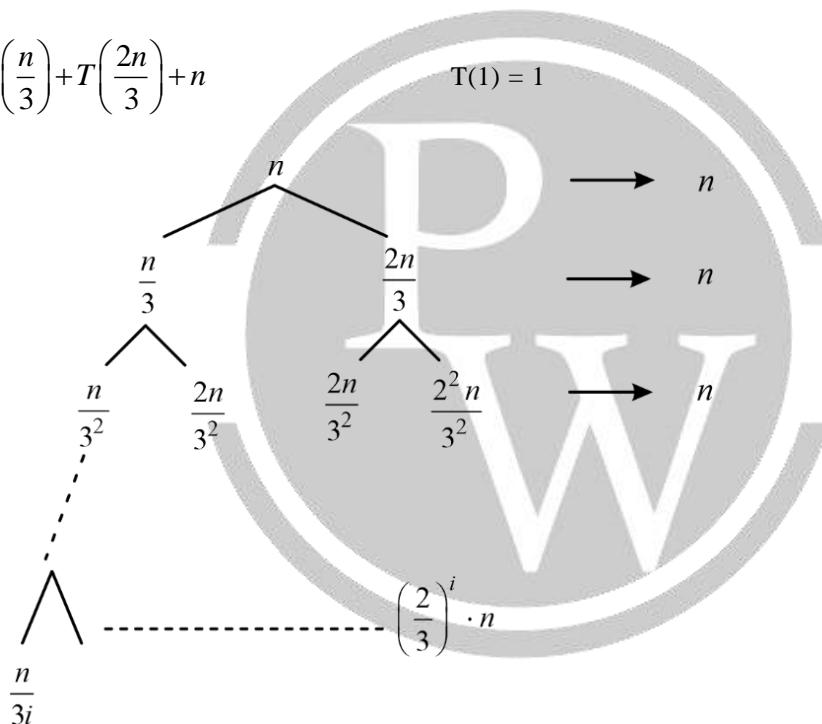
$$= n \left(\frac{2^{k+1} - 1}{2 - 1} \right)$$

$$= n (2 \cdot 2^{k-1})$$

$$= n (2n) - 1$$

$$= O(n^2)$$

(4) $T(n) = T\left(\frac{n}{3}\right) + T\left(\frac{2n}{3}\right) + n$



$$\frac{n}{3i} - 1; n - 3i; i - \log_3 n$$

$$= n + n + \dots + \log_3 n T(n)$$

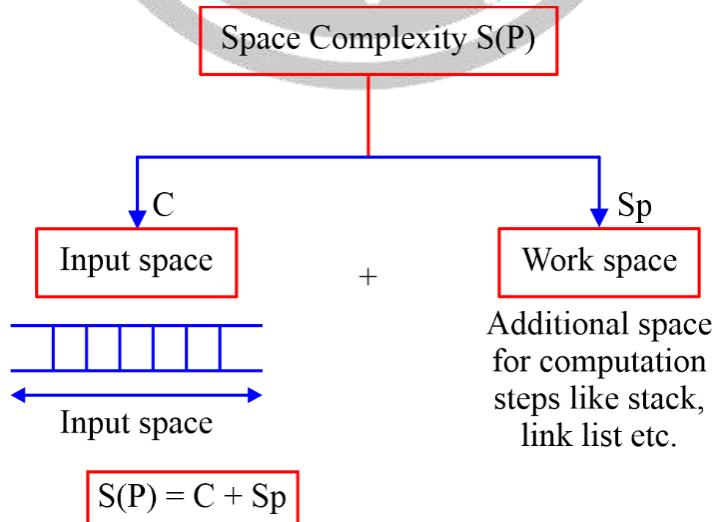
$$= (n + n + \dots + \log_3 n) \geq n + \dots + \log_3 n$$

$$\Omega(n \log_{3/2} n)$$

1.10 Recurrence Relations and their Time Complexity

$T(n) = C; n = 2$	$O(\log n)$
$T(n) = 2 T(\sqrt{n}) + C; n > 2$	
$T(n) = C; n = 2$	$O(n)$
$T(n) = T(n - 1) + C; n > 2$	
$T(n) = C; n = 1$	$O(n^2)$
$T(n) = T(n - 1) + n + C; n > 2$	
$T(n) = C; n = 1$	$O(2^n)$
$T(n) = 2T(n - 1) + C; n > 1$	
$T(n) = C; n = 1$	$\Theta(n)$
$T(n) = 2T\left(\frac{n}{2}\right) + C; n > 1$	
$T(n) = C; n = 1$	$\Theta(n \log n)$
$T(n) = 2T\left(\frac{n}{2}\right) + n; n > 1$	
$T(n) = C; n = 1$	$\Theta(\log n)$
$T(n) = T\left(\frac{n}{2}\right) + C; n > 1$	
$T(n) = 1; n = 2$	$\Theta(\log \log n)$
$T(n) = T(\sqrt{n}) + C; n > 2$	

1.11 Space Complexities



```

Int n, A[n];
Algorithm Rsum(A, n)
{
    if (n = 1) return (A(1));
    else;
        return (A[n] + RSum(A, (n-1)));
}

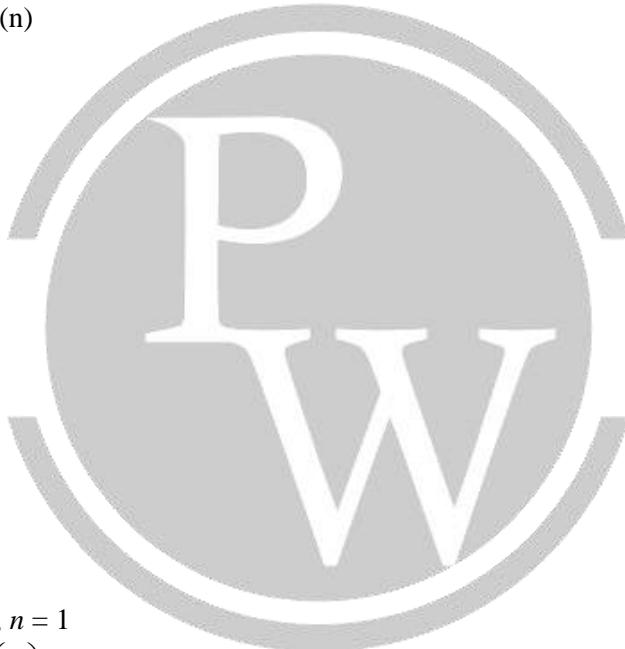
```

- **Time Complexity** = $O(n)$
- **Space Complexity**
- We need stack space
- Stack is used to store activation records of function calls
- Size of activation records is trivial
- Stack size that we need = $O(n)$
- Space complexity = $O(n)$

```

Algorithm A(n)
{
    if (n = 1) return;
    else;
    {
        A( $\frac{n}{2}$ );
    }
}

```



Recurrence relation

$$T(n) = C; n = 1$$

$$T(n) = T\left(\frac{n}{2}\right) + C; n > 1$$

Time Complexity = $O(\log n)$

Space Complexity

- Space complexity will depend on number of activation record pushed into the stack
Suppose, $n = 16$

A (1)
A (2)
A (4)
A (8)
A (16)

For $n = 2^k$ we are pushing
the ' k ' activation record

∴ Space Complexity

$$\begin{aligned} n &= 2^K \\ \log n &= K \log_2 2 \\ K &= \log_2 n \end{aligned}$$

Space Complexity = $O(\log n)$

Example 3

```
Algorithm A(n)
{
    if (n = 2) return;
    else;
    return (A  $\sqrt{n}$ );
}
```

Solution:

$$\begin{aligned} T(n) &= 1; n = 2 \\ T(n) &= T(\sqrt{n}) + C; n > 2 \end{aligned}$$

Time Complexity = $O(\log \log n)$

Space Complexity

Suppose $n = 16$

A(1)
A(2)
A(4)
A(16)

∴ For $\frac{n}{2^k}$ manner we are pushing in stack

$$2^{\frac{n}{2^k}} \geq 2$$

$$\frac{n}{2^k} \log_2 2 \geq \log_2 2$$

$$n \geq 2^K$$

$$K \leq \log_2 n$$

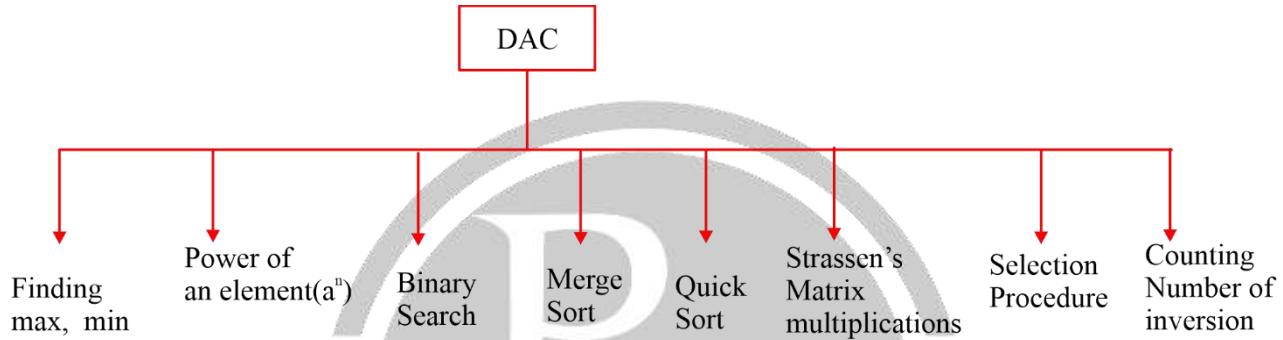
Space complexity = $O(\log_2 n)$



2

DIVIDE AND CONQUER

2.1 DAC Application



2.2 Finding Maximum Minimum element

Recurrence Relation:

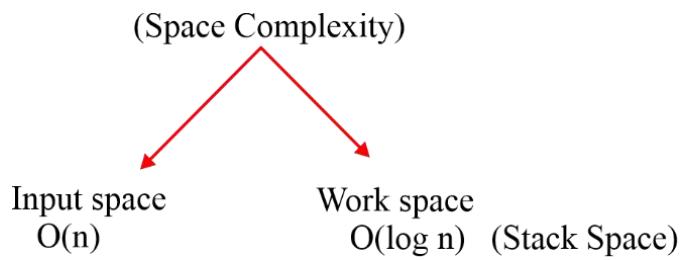
$$T(n) = \begin{cases} 1 & \text{if } n=1 \text{ or } n=2 \\ 2T\left(\frac{n}{2}\right) + 1 & \text{if } n > 2 \end{cases}$$

Time Complexity:

$$T(n) = O(n)$$

- Time complexity is same for every case (Best case/Worst case).

Space Complexity:



$$\begin{aligned}\text{Space Complexity} &= O(n) + O(\log n) \\ &= O(n)\end{aligned}$$

Number of comparisons to find maximum / minimum element on an given array of n elements:

$$\text{Comparison} = \frac{3n}{2} - 2$$

2.3 Power of an Element

Recurrence relation:

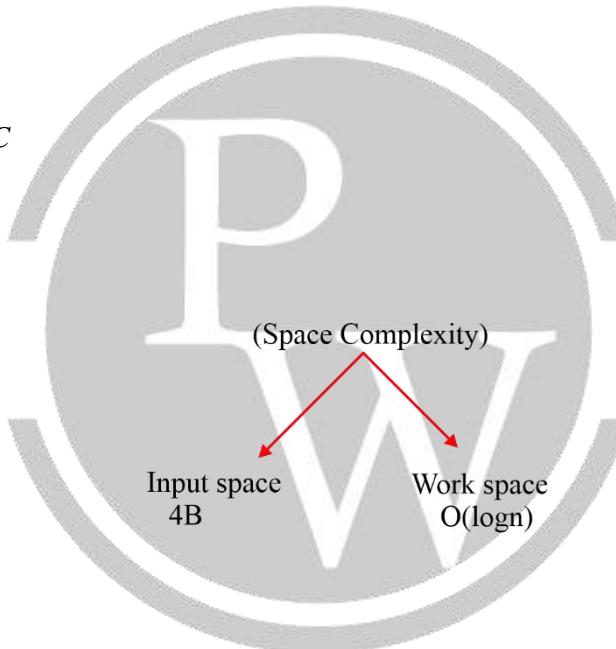
$$T(n) = \begin{cases} 1 & \text{if } n=1 \\ T\left(\frac{n}{2}\right) + 1 & \text{if } n > 1 \end{cases}$$

Time Complexity:

$$T(n) = T\left(\frac{n}{2}\right) + C$$

$$T(n) = O(\log n)$$

Space Complexity:



$$\begin{aligned}\text{Space Complexity} &= 4B + O(\log n) \\ &= O(\log n)\end{aligned}$$

Number of multiplications to find a^n

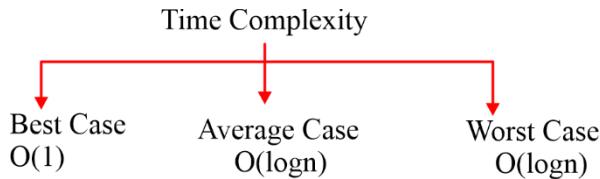
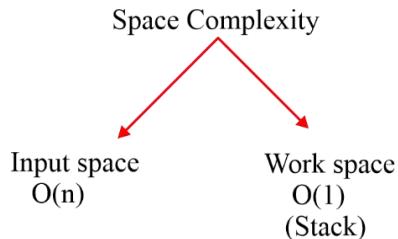
$$\text{Multiplication} = O(\log n)$$

2.4 Binary Search

Given a sorted array and an element x, need to return the index of element x if it is present then 1, otherwise – 1.

Recurrence relation:

$$T(n) = \begin{cases} 1 & \text{if } n=1 \\ T\left(\frac{n}{2}\right) + C & \text{if } n > 1 \end{cases}$$

Time Complexity:**Space Complexity:**

$$\text{Space complexity} = O(n) + O(1)$$

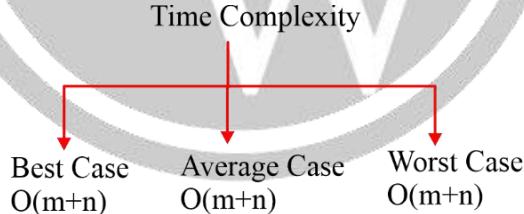
$$= O(n)$$

2.5 Merge Algorithm

- Merging two sorted sub arrays of input size m,n.
- Number of comparisons to merge two sorted sub arrays of size m,n.

$$\text{Comparisons} = m + n - 1 \text{ (worst case)}$$

$$\text{Number of moves} = m + n \text{ (Outplace Algorithm)}$$



Number of comparisons in best case of merging two sorted subarrays of size m, n.

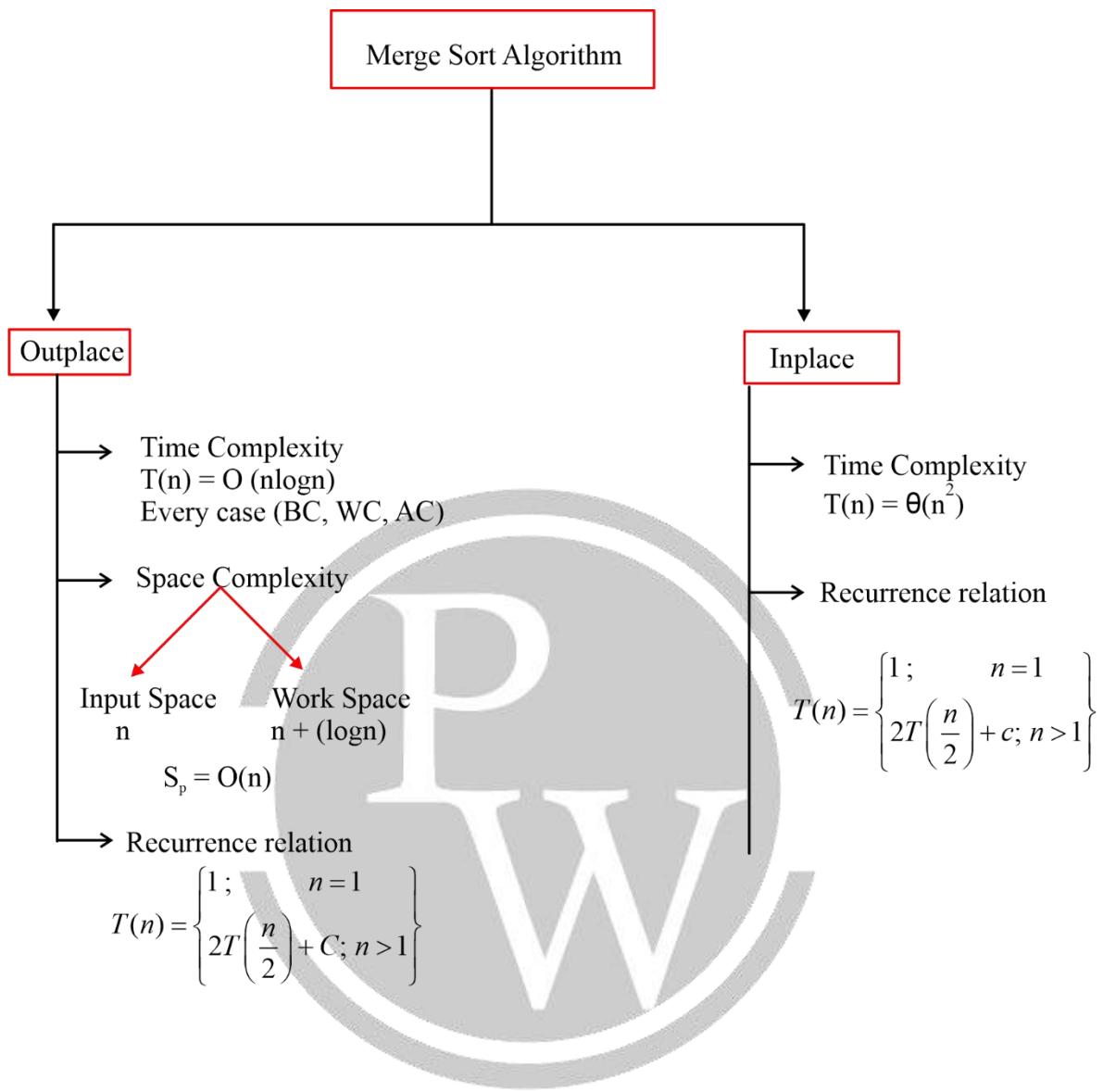
$$\text{comparisons} = \min(m, n)$$

$$\text{Moves} = m + n \text{ (Always)}$$

Note:

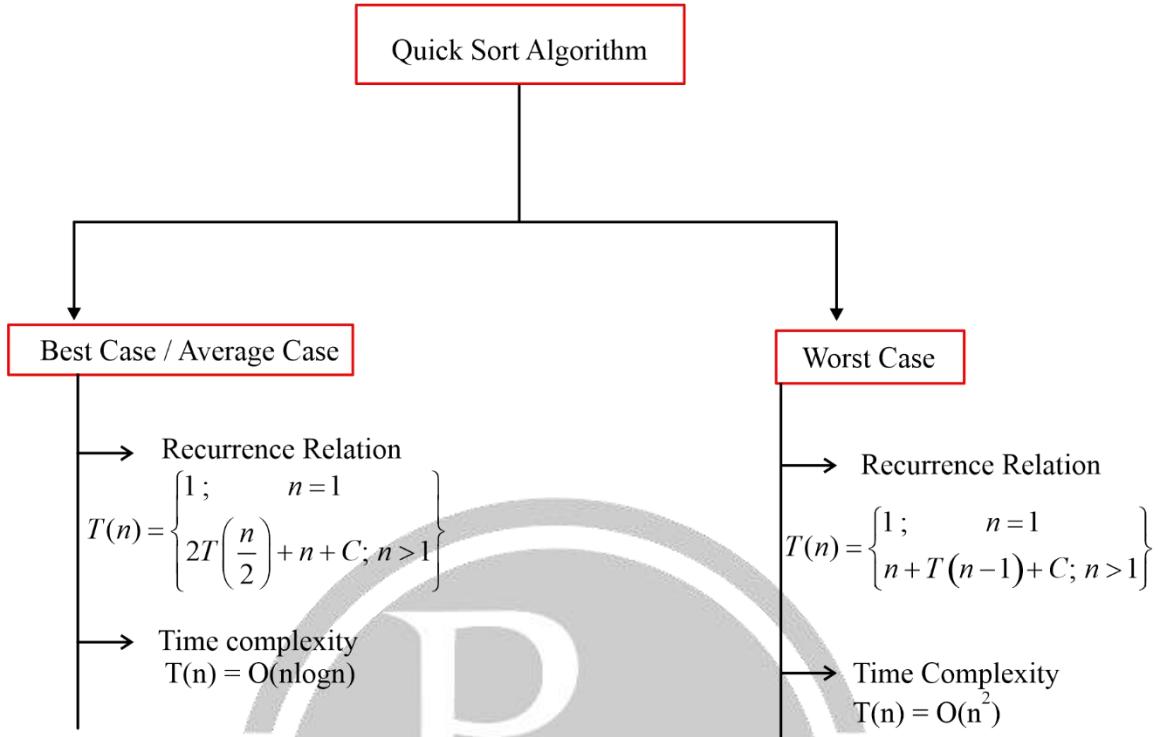
Best Case comes in comparisons no effect on moves.

2.5.1 Merge Sort Algorithm:

**Note:**

- In GATE exam if merge sort given then always consider outplace.
- If array size very large, merge sort preferable.
- If array size very small, then prefer insertion sort.
- Merge sort is stable sorting technique.

2.6 Quick Sort Algorithm



Example 1: In Quick for sorting n elements, the $\left(\frac{n}{16}\right)^{\text{th}}$ smallest element is selected as pivot. what is the worst-case time Complexity?

Solution.

$$\begin{aligned}
 T(n) &= T\left(\frac{n}{16}\right) + T\left(\frac{15n}{16}\right) + O(n) \\
 &= (\text{solve by recursive tree method})
 \end{aligned}$$

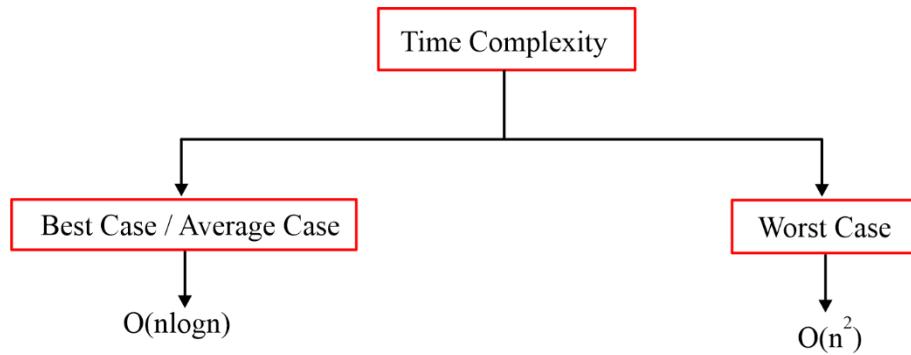
Example 2: The median of n elements can be found in $O(n)$ time then, what is the time complexity of quick sort algo in which median selected as pivot?

Solution.

$$\begin{aligned}
 T(n) &= O(n) + C + O(n) + T(n/2) + T(n/2) \\
 &\quad \downarrow \quad \downarrow \quad \downarrow \\
 &\quad \text{Find median} \quad \text{swap median} \quad \text{Partition algo} \\
 &= 2T(n/2) + C \cdot n \\
 &= O(n \log n)
 \end{aligned}$$

2.6.1 Randomized Quick Sort

- In Randomized quick sort algorithm selection of pivot element can be taken randomly.



2.7 Counting Number of Inversion

- Counting number of inversion on given an array of an element.

Time complexity $T(n) = O(n\log n)$

2.8 Selection Procedure

Find K^{th} smallest on given an array of an element and integer K .

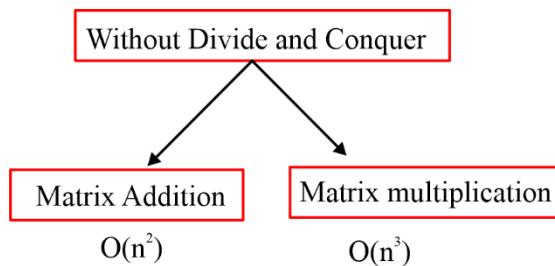
Time Complexity:

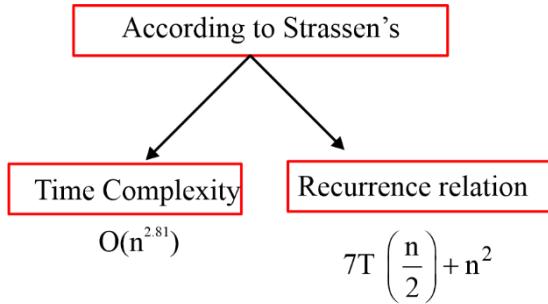
$$T(n) = O(n^2)$$

Space complexity:

$$\text{Space Complexity} = O(n)$$

2.9 Strassen's matrix Multiplication





2.10 Comparison Based Sorting Algorithms

Sorting Algorithm	Basic logic of sorting Algo	BC	AC	WC	Stable sorting	Inplace sorting
Quick sort	Choose pivot element place in correct position	$\Theta(n \log n)$	$\Theta(n \log n)$	$\Theta(n^2)$	No	Yes
Merge sort	Divide to equal parts recursively sort each sub part & merge them	$\Theta(n \log n)$	$\Theta(n \log n) = n \log n$	$\Theta(n \log n) = n \log n$	Yes	No
Heap sort	Build heap(max) delete max place	$\Theta(n \log n)$	$\Theta(n \log n)$	$\Theta(n \log n)$	No	Yes
Bubble sort	Compare exchange	$\Theta(n)$	$\Theta(n^2)$	$\Theta(n^2)$	Yes	Yes
Selection sort	Find position of min element from [1 to n]	$\Theta(n^2)$	$\Theta(n^2)$	$\Theta(n^2)$	No	Yes
Insertion sort	Insert a [i + 1] into correct position	$\Theta(n)$	$\Theta(n^2)$	$\Theta(n^2)$	Yes	Yes



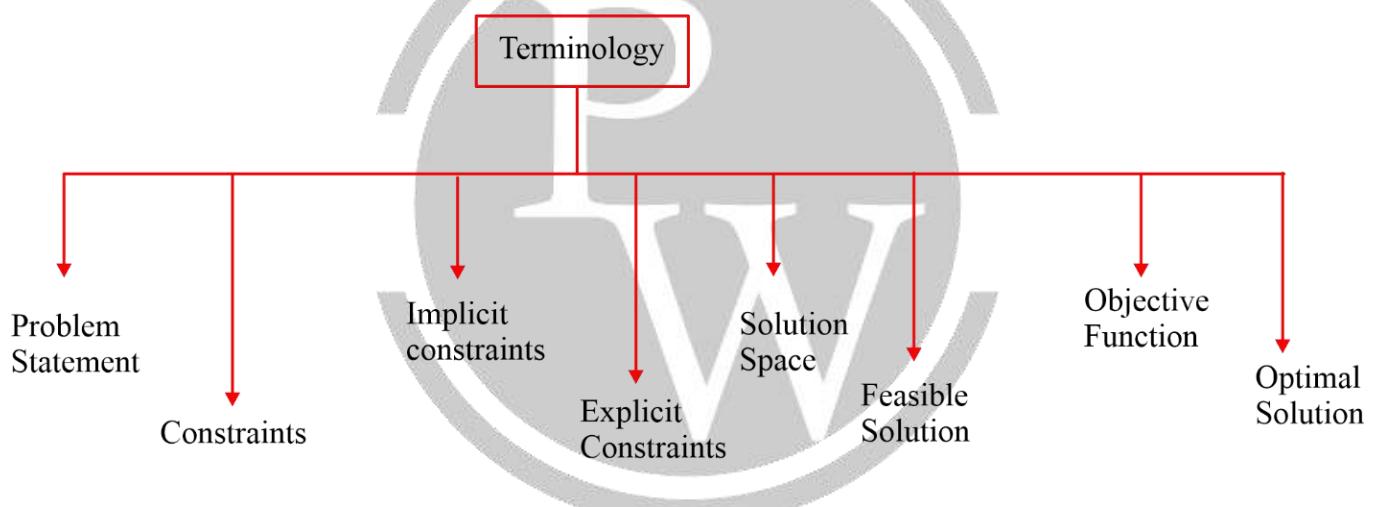
3

GREEDY TECHNIQUE

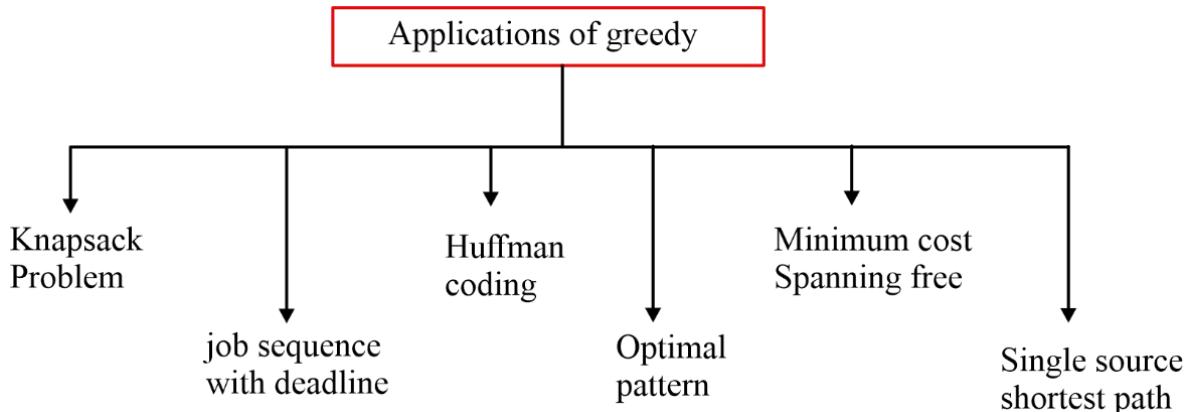
3.1 Greedy Technique

- Greedy method is an algorithm design strategy used for solving problems where solution are seen as result of making a sequence of decisions.
- A problem may contain more than one solution.

3.2 Terminology



3.3 Applications of greedy

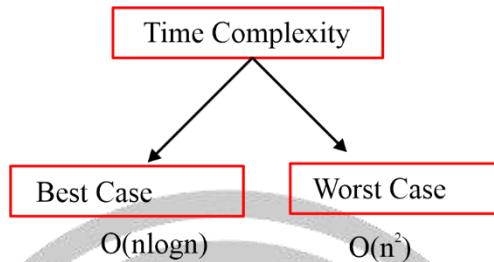


3.4 Knapsack Problem

Time complexity $T(n) = O(n \log n)$

3.5 Job Sequence with Deadline

- Single CPU only.
- Arrival time of each job is same.
- No pre-emption.



3.6 Optimal Merge Pattern

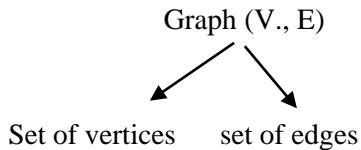
- This is a problem related to merging of files. Given a set of n -files in sorted order. It is required to merge them into a single sorted file with 2-way merging.
- This problem is like merging process in merge sort. In merge sort we were interested in number of comparisons but in optimal merge pattern we are interested in record movement (i.e moving a record from one file to another file).
 - If file F1 has ' n ' records and file 'F2' has ' m ' records then number of record movement will be ' $m+n$ '. 1 2
- At any point choose two records with least weight merge them and put them in list and continue it until all records are merged.
- Time complexity $T(n) = O(n \log n)$
- Space complexity = $O(n)$

3.7 Huffman Coding

- Huffman coding is essentially a non-uniform encoding with convention that the character with higher frequency (probability) of occurrence will be enclosed with less number of bits.
- It comes under data compression technique.
- Time complexity $T(n) = O(n \log n)$

3.8 Minimum Cost Spanning Tree

3.8.1 Graph



- Let $G(V, E)$ be a simple graph then

$$\text{Maximum edges} = \frac{V(V-1)}{2}$$

$$E \leq \frac{V(V-1)}{2}$$

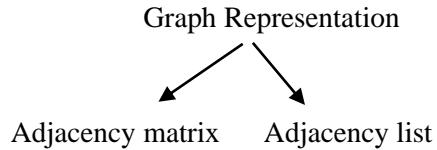
$$E \leq C \cdot V^2 \quad C \text{ is constant}$$

Note:

$$E = O(V^2)$$

$$\log E = O(\log V)$$

3.8.2 Graph Representation



- For more edges (Dense Graph) Adj. matrix is better (density more).
- For less edge (sparse graph) Adj list is better.

	Matrix	List
(1) Finding degree of vertex \Rightarrow Time Complexity	$O(V)$ Every Case	$O(1)$ Best Case $O(V)$ Worst Case
(2) Finding total edges \Rightarrow Time Complexity	$O(V^2)$ Every Case	$O(V+2E)$ Worst Case $O(V)$ Best Case
(3) Finding 2-vertices adjacent (or)not \Rightarrow Time Complexity	$O(1)$	$O(V-1)$ Worst Case $O(1)$ Best Case
(4) $G(V, E) \Rightarrow$ space	$O(V^2)$ Every Case	$O(V+E)$ Every Case

3.8.3 What is Spanning Tree

A subgraph $T(V, E')$ of $G(V, E)$ where E' is the subset of $(E' \subseteq E)$ is a spanning tree iff 'T' is a tree.

A sub graph $G(V, E')$ of $G(V, E)$ is said to be spanning tree.

- (1) T' should contain all vertices of G
- (2) T' should contain $(V-1)$ edged where V is number of vertices without cycle.

(3) T' should be connected.

3.8.4 Minimum Cost Spanning Tree

Minimum cost spanning tree is the one in which cost of the spanning tree formed should be minimum.

3.8.5 Prims Algorithm

- Select Any vertex

$$\begin{aligned} \text{Time complexity} &= V + V \log V + 2E + E \log V \\ &= O(E + V) \log V \end{aligned}$$

Using Sorted Array & Adjacency List

$$V + 2E + E \times V = O(EV)$$

Using Sorted Array & Adjacency List

$$V \times O(1) + V^2 + E \times V = O(EV)$$

3.8.6 Kruskal algorithm

- Take first minimum edge

$$\begin{aligned} \text{Time complexity} &= E \log E + (V + E) \\ &= O(E \log E) = O(E \log V) \end{aligned}$$

If edges are already sorted

$$TC = O(E + V)$$

3.9 Single Source Shortest Path

3.9.1 Dijkstra Algorithm

- Using min heap & adjacency list = $O(E + V) \log V$
- Using adjacency Matrix & Min heap = $O(V^2 E \log V)$
- Using adjacency list & Unsorted Array = $O(V^2)$
- Using adjacency list & Sorted Doubly Linked List = $O(EV)$

3.9.2 Bellman-Ford

- Time Complexity = $O(EV)$
- If negative edge weight cycle then for some vertices Incorrect answer.



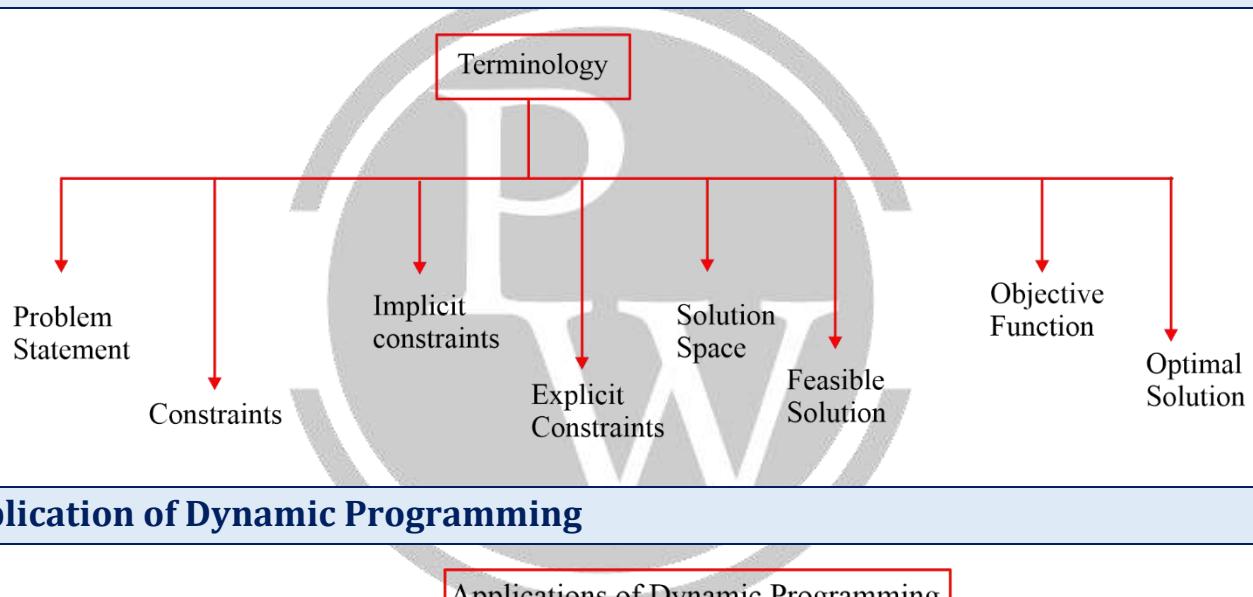
4

DYNAMIC PROGRAMMING

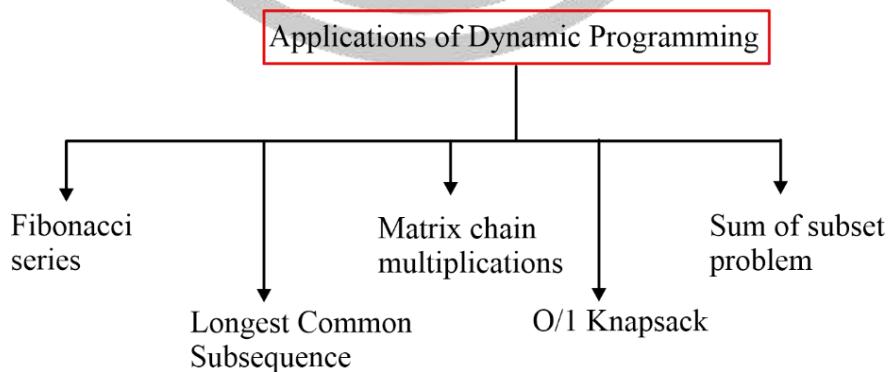
4.1 Dynamic Programming

In dynamic programming for optimal solution always computes distinct function calls.

4.2 Terminology



4.3 Application of Dynamic Programming

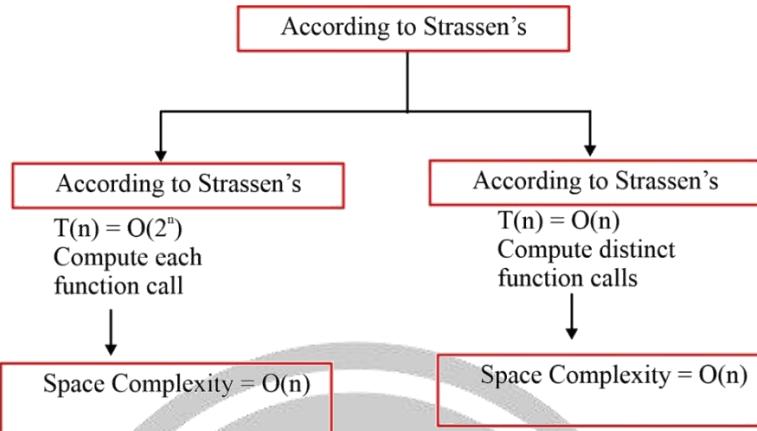


4.4 Fibonacci Series

- Time complexity $T(n) = O(n\log n)$
- Computes distinct function calls.

4.5 Job Sequence with Deadline

- Single CPU only
- Arrival time of each job is same
- No pre-emption



4.6 Longest Common Subsequence (LCS)

- For common subsequence always consider two strings:
- $P = <\text{ABCDB}>$ – $Q = <\text{BDCABA}>$
- Common subsequences for both 'p' are
- $S = <\text{A}>$
- $S = <\text{AB}>$
- $S = <\text{CAB}>$
- $S = <\text{BDAB}>$
- A common subsequence of longest length is known as longest common subsequence.
- For above problem longest common subsequence will be of length u.

4.6.1 Applications of LCS

1. Genomics
2. Software engineering applications
3. Plagiarism
4. Data gathering system of search engines

4.6.2 Algorithm for LCS

```
LCS (p,q)
{
1. For     $i \leftarrow 0$  to  $n-1$ 
         $L[i-1] = 0$ 
2. For     $j \leftarrow 0$  to  $m-1$ 
         $L[-1, j] = 0$ 
```

```

3.   For       $i \leftarrow 0$  to  $n-1$ 
      For       $j \leftarrow 0$  to  $m-1$ 
          If ( $p[i] = q[j]$ ) then
               $L[i, j] = 1 + L(i-1, j-1);$ 
          else
               $L[i, j] = \max\{L[i, j-1], L[i-1, j]\}$ 
      }
  
```

- Time complexity of step 1 = $O(n)$
- Time complexity of step 2 = $O(m)$
- Time complexity of step 3 = $O(mn)$
- Total Time complexity = $O(n) + O(m) + O(mn)$
 $= O(mn)$
- Space complexity = $O[(M+1).(n+1)]$
 $= O(mn)$

4.7 Matrix Chain Multiplications

Two matrices ‘A’ and ‘B’ are compatible if and only number of column of first matrix must be equal to number of rows of second matrix.

4.7.1 Brute force method

Number of parenthesizing for a given chain is given by Catalan number: $\left[\frac{1}{n+1} {}^{2n}C_n \right]$

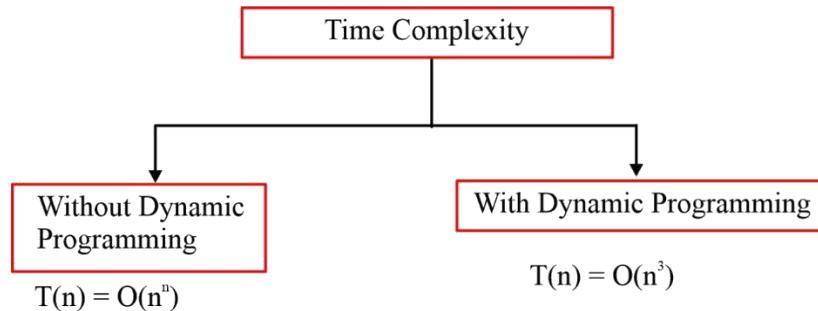
Time complexity = $O(n^n)$

Space complexity = $O(n)$

4.7.2 Algorithm For Matrix Chain Multiplication

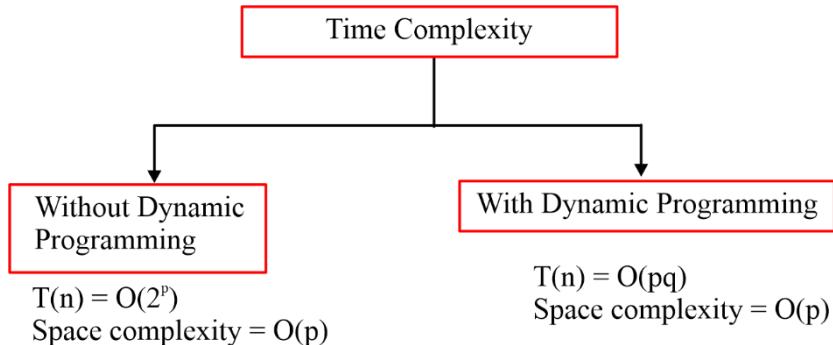
The time complexity of multiply the given chain of n matrices $\langle A_1, A_2, A_3, \dots, A_n \rangle$ using dynamic programming (district function call) is $O(n^3)$

Space complexity = $O(n^2)$



4.8 0/1 Knapsack Problem

The maximum profit can be achieved by O/1 knapsack problem where capacity of problem is 'p' and number of objects are 'q'.



4.9 Sum of Subset Problem

- Given n-elements and an integer 'm', it is required to determine whether there exists a subset of given n elements, whose sum equal M.
- This is a decision problem (True/False).

4.9.1 Algorithm for Sum of Subset Problem

```
SoS(n, M, A)
// A [1 . . . n] is an array of elements
{
    1. for i = 0 to n
        for j = 0 to M
            if (i >= 0 and j = 0)
                SoS [i, j] = T
            else
                if (i = 0 and j > 0)
                    SoS [i, j] = F;
                else
                    if (A[i] > j)
                        SoS [i, j] = SoS [i - 1, j]
                    else
                        SoS [i, j] = SoS [i - 1, j] or
                        SoS [i - 1, j - A[i]]
}
```

4.9.2 Time Complexity of SoS

Two for loops are there thus repeating for $(n * m)$ times. Thus, time complexity = $O(n * m)$

Time complexity of SoS becomes exponential if $M = 2^n$

$$T.C = O(n \times 2^n)$$



GATE Exam 2024?



PARAKRAM BATCH

ME | CE | EE | EC | CS & IT

- Live Classes & Recorded Videos
- DPPs & Solutions
- Doubt Solving
- Batches are available in **Hindi**

Weekday & Weekend
Batches Available



JOIN NOW!

Theory of Computation



Theory of Computation

INDEX

- | | | |
|----|---------------------------------------|-------------|
| 1. | Basics of Theory of Computation | 7.1 – 7.5 |
| 2. | Finite Automata | 7.6 – 7.21 |
| 3. | Push Down Automata | 7.22 – 7.29 |
| 4. | Turning Machine | 7.30 – 7.39 |

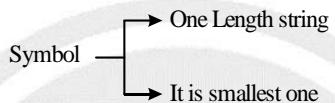


1

BASICS OF THEORY OF COMPUTATION

1.1 Symbol

Symbol represents very unique in the world. Any small thing that never be broken into any other is called as symbol.



1.2 Alphabet (Σ)

It is a set of finite number of symbols.

Example:

- English alphabet = {a, b, ... z}
- Binary alphabet = {0, 1}
- Decimal alphabet = {0, 1, 2, ... 9}
- We can create our own alphabet = {gate, cs, it, exam}
where gate, cs, it, exam all are symbols
Alphabet (Σ) = {gate, cs, it, exam}
 $\downarrow \quad \downarrow \quad \downarrow \quad \downarrow$
{ w , x , y , z }

1.3 String

- It is sequence of symbols defined over given alphabet.
let $\Sigma = \{a, b\}$
- Strings possible are \in, a, b, aa, bb, \dots
- Strings over English Alphabet: deva, gate, exam, etc.
- Strings over Binary Alphabet: 0010, 1011, 1101, 1111, etc.
- Strings over Decimal Alphabet: 3012, 2345, 5438, etc.

Note:

- Different length strings over given alphabet
 - I Zero length string → Empty string / Null string \in or λ is used to denote empty string length of empty string. $|\in| = 0$.
 - II One length string over $\Sigma = \{a, b\}$ are a, and b (2 strings).
 - III Two length strings over $\Sigma = \{a, b\}$ are da, ab, ba, and bb (4 strings).

1.4 Operations on Strings

There are various operations on strings:

- Unary and Binary operations

1. length of a string: Length of a string is denoted as $|w|$ and is defined as the number of positions for the symbol in the string.

Example: $w = aba$
 $|w| = |aba| = 3$

2. Reversal of a string: It will reverse or changes the order of a given string w .

Example: $w = abb$
 $w^R = bba$

1.4.1 Concatenation of Two Strings

Given two strings w_1 and w_2 , we define the concatenation of w_1 and w_2 to be the string as w_1w_2 .

Example:

$w_1 = a$, and $w_2 = ba$

Then $w_1w_2 = a.ba = aba$.

1.4.2 Prefix of a String

A substring with the sequence of beginning symbols of a given string is called a “prefix”.

Example:

(i) $w = aaaa$

Prefixes = { \in , a, aa, aaa, aaaa}

(ii) $w = abcd$

Prefixes = { \in , a, ab, abc, abcd}

Note:

If $|w| = n$ then $|w^{Rev}| = n$.

If length of w_1 is n_1 and length of w_2 is n_2 then length of $w_1w_2 = |w_1 w_2| = n_1 + n_2$.

- Let w be n length string.

- Number of prefixes in $w = n+1$
- Number of non-empty prefixes of $w = n$
- Number of different length prefixes of $w = n + 1$
- Number of different length prefixes of w excluding zero length = n

1.4.3 Suffix of a String

A substring with the sequence of ending symbols of a given string is called a “suffix”.

Example:

(i) $w = \text{aaaa}$

Suffixes = { \in , a, aa, aaa, aaaa}

(ii) $w = \text{abcd}$

Suffixes = { \in , d, cd, bcd, abcd}

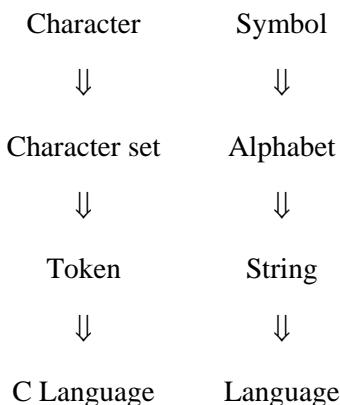
Note :

- Let w be n length string.
 - (i) Number of suffixes of $w = n + 1$
 - (ii) Number of non-empty suffixes of $w = n$
 - (iii) Number of different length suffixes of $w = n + 1$
 - (iv) Number of different length suffixes of w excluding zero length = n

1.4.4 Substring of a String

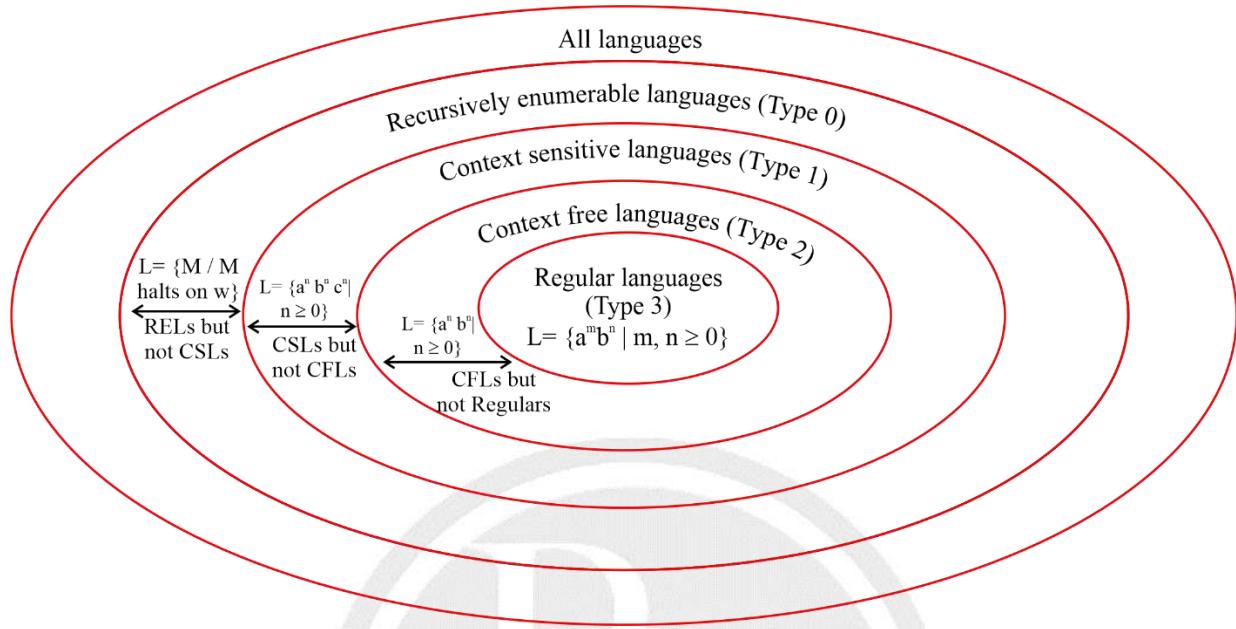
- Let w be n length string.
- | | |
|--|--|
| (i) Number of substrings of w | |
| | $\begin{aligned} \text{minimum} &= n + 1 \\ &\text{(over 1 symbol)} \\ \text{maximum} &= 1 + n + (n - 1) + \dots + 1 \\ &\text{(all characters distinct)} = 1 + \sum_{i=1}^{n-1} i \\ &= \frac{n(n+1)}{2} \end{aligned}$ |
| (ii) Number of non-empty substrings of w | |
| | $\begin{aligned} \text{minimum} &= n \\ \text{maximum} &= \sum_{i=1}^n i \end{aligned}$ |
- Number of different length substrings of any given n length string = $n+1$
 - Number of different length substrings of any given n length string excluding zero length = n .

1.5 Relation between Symbol, Alphabet, String and Language



1.5.1 Chomsky Hierarchy

- Chomsky hierarchy includes all the problems in the world classified into classes.



- Type 3 is the smallest class, and Type 0 is the biggest class.

	Type 3	Type 2	Type 1	Type 0
Language:	Regular	Context free	Context sensitive	Recursively Enumerable
Automata:	Finite Automata	Push down Automata	Linear Bounded Automata	Turing Machine
Grammar:	Regular	Context free	Context Sensitive	Unrestricted

1.6 Language

- Language is set of strings defined over alphabet (Σ).
- Let $\Sigma = \{a, b\}$. Then $\Sigma = \Sigma^0 \cup \Sigma^1 \cup \Sigma^2 \cup \dots$
 - = set of all strings
 - = universal language
 - = $\{\epsilon, a, b, aa, ab, ba, \dots\}$
- $\Sigma^2 = \Sigma^1 \cdot \Sigma^1 = \{a, b\} \cdot \{a, b\} = \{aa, ab, ba, bb\}$
- Language: It is a subset of Σ^*
 - $\therefore L \subseteq \Sigma^*$
- A language is a collection of strings that must be a subset of Σ^* where Σ^* is a universal language.

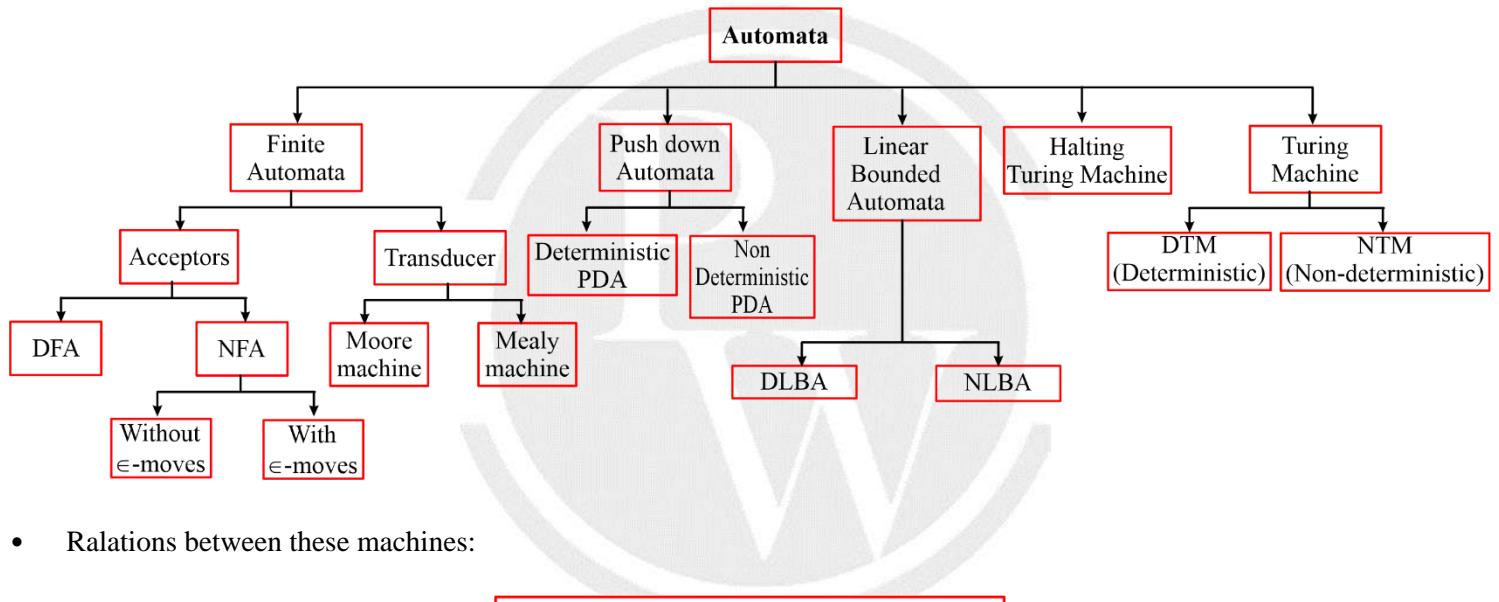
Example:

$$L = ab^* = \{a, ab, abb, \dots\}$$

1.7 Types of Languages

- Finite Language
- Infinite Language
- Regular language
- DCFL (Deterministic CFL)
- CFL
- CSL
- Recursive Language
- Recursive Enumerable Language (REL)

1.8 Types of automata



1. **Less power:** It can Represent less number of languages.
2. **More power:** It can Represent more number of languages Compare to all these machines.

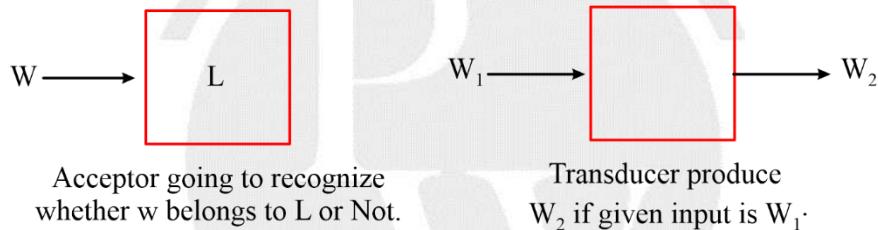


2

FINITE AUTOMATA

2.1 Introduction

- Finite Automata describes or represents a regular language.
- Finite Automata is of two types:
 - (i) Acceptor: Accepts or rejects given string.
 - (ii) Transducer: Produces output string for given input string.



Example:

1's complement of binary number: Input = 10100, Output = 01011

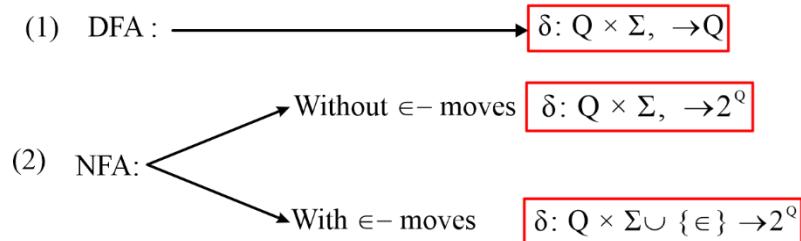
2.2 Finite Acceptor (Finite Automata)

- Finite Automata:

$$FA = (Q, \Sigma, \delta, q_0, F)$$

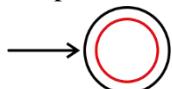
Annotations for the components of the FA definition:

- Set of final states
- Initial state
- Transition function
- Set of input symbols
- Set of finite number of states

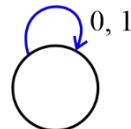
Transition function (δ)


2.3 Construction of DFA

1. If epsilon belongs to L, then initial state must be final in DFA.

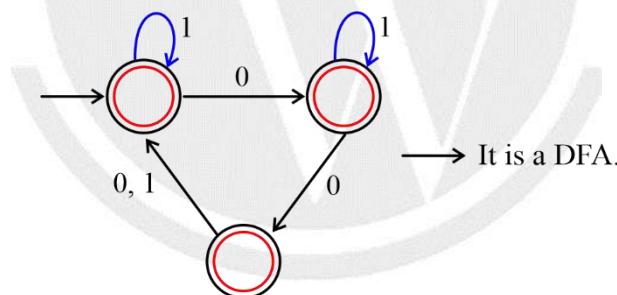


2. **Dead state:** It is non-final state but it never contain a path to final.

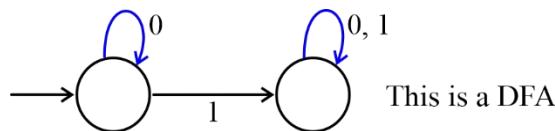


- Once we reach the dead state, there is no way to reach to the final state.
- If every state is final then every string accepted in the DFA.

3. If all states are finals in DFA then $L(DFA) = \Sigma^*$



4. If every state is non final in a DFA then $L(DFA) = \emptyset$ or $L(DFA) = \{\}$,



$$\bar{L} = \Sigma^* - L$$

$$L \cup \bar{L} = \Sigma^*$$

$$L \cap \bar{L} = \emptyset$$

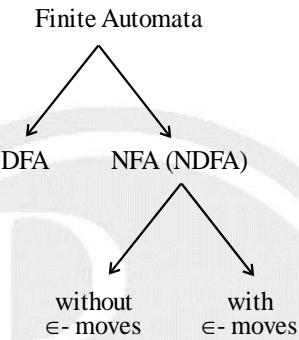
- If $|w| = n$ or $|w| \leq n$, then number of states in minimum DFA = $(n + 2)$ states
- If $|w| \geq n$ then number of states in minimum DFA = $n + 1$ states.
- Start condition, exactly, atmost length question requires Dead state but end condition, contain substring, atleast length questions do not require dead state.

- Over one symbol $\Sigma = \{a\}$. Length of string is equal to number of a's in string. $|w| = n_a(w)$.
- Let $L = \{w | w \in \{a,b\}^*, n_a(w) \text{ is divisible by } m, n_b(w) \text{ is divisible by } n\}$. Then Number of states in DFA = $m \times n$.
- Let $L = \{w | w \text{ belongs to } \{a, b\}^*, n^{\text{th}} \text{ symbol of } w \text{ from begin is 'a'}\}$. Then Number of states in DFA = $(n + 2)$.
- Number of equivalence classes for any regular set L = Number of states in minimal DFA that accepts L .

2.4 Non-Deterministic Finite Automata

2.4.1 Introduction

- Finite Automata can be designed in two ways:



- All these finite state machines are equivalent. $DFA \equiv NFA$
- We can convert one finite state machine to any other finite state machine.
- For every regular language, we can design infinite equivalent DFAs or infinite equivalent NFAs but minimum DFA is unique for given regular language.
- For every regular language, one or more minimum NFAs may exist.

Note : For every regular language:

- Unique minimum DFA exists.
- One or more minimum NFAs exists.

2.5 Comparison of NFA and DFA

	$ w = n$	$ w \leq n$	$ w \geq n$
Number of states in NFA	$n + 1$	$n + 1$	$n + 1$
Number of states in DFA	$n + 2$	$n + 2$	$n + 1$

- If every string has n^{th} symbol from begin is 'a' over binary alphabet {a, b} then $(n + 1)$ states in minimum NFA but $(n + 2)$ states in minimum DFA.

- If every string has n^{th} symbol from end is 'a' over binary alphabet {a, b} then 2^n states in minimum DFA and $(n + 1)$ states in minimum NFA.

2.5.1 COMPARISON OF DFA AND NFA (NFA vs DFA)

	NFA	DFA
(1) Transition Function (δ)	$Q \times \Sigma \rightarrow 2^Q$	$Q \times \Sigma \rightarrow Q$
(2) Number of paths for string	For valid string: 1 path For invalid string: $>= 0$ paths	For valid string: 1 path For invalid string: 1 path

2.5.2 Number of states in DFA and NFA for Regular Languages

Language	NFA states	DFA states
(1) $\{w \mid w \in \{a, b\}^*, w = n\}$	$n + 1$	$n + 2$
(2) $\{w \mid w \in \{a, b\}^*, w \leq n\}$	$n + 1$	$n + 2$
(3) $\{w \mid w \in \{a, b\}^*, w \geq n\}$	$n + 1$	$n + 2$
(4) $\{w \mid w \in \{a, b\}^*, w \text{ starts with } a\}$	2	3
(5) $\{w \mid w \in \{a, b\}^*, n^{\text{th}} \text{ symbol from begin is } a\}$	$n + 1$	$n + 2$
(6) $\{w \mid w \in \{a, b\}^*, n^{\text{th}} \text{ symbol from end is } 'a'\}$	$n + 1$	2^n

2.5.3 Finding number of states in DFA using NFA

NFA (n states)

↓ Subset Construction

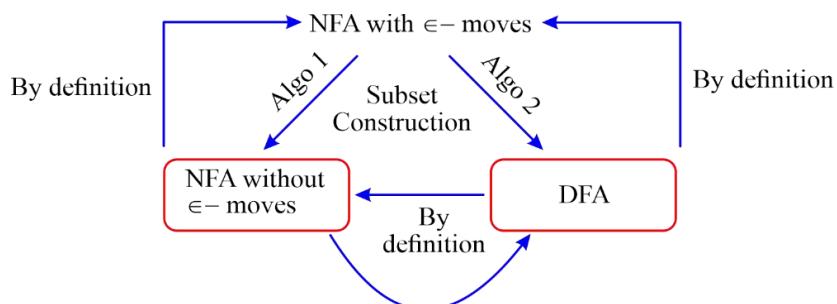
DFA (2^n states exists)

↓ Partition algorithm

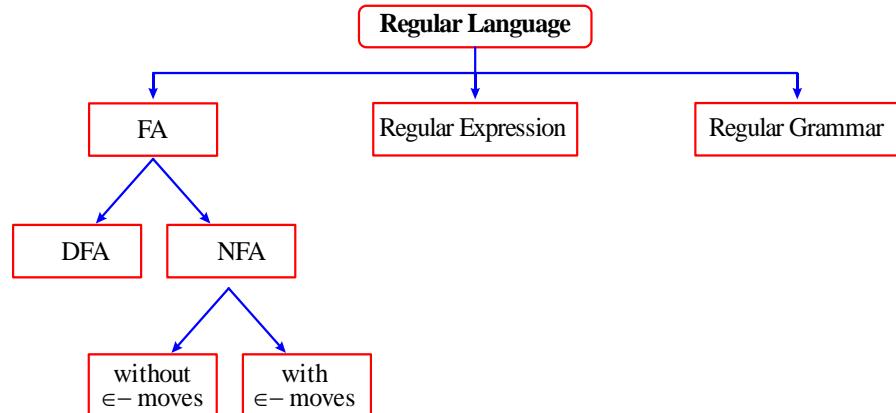
Minimum DFA ($\leq 2^n$ states) atmost 2^n states

- Every DFA is NFA, but NFA need not be DFA.

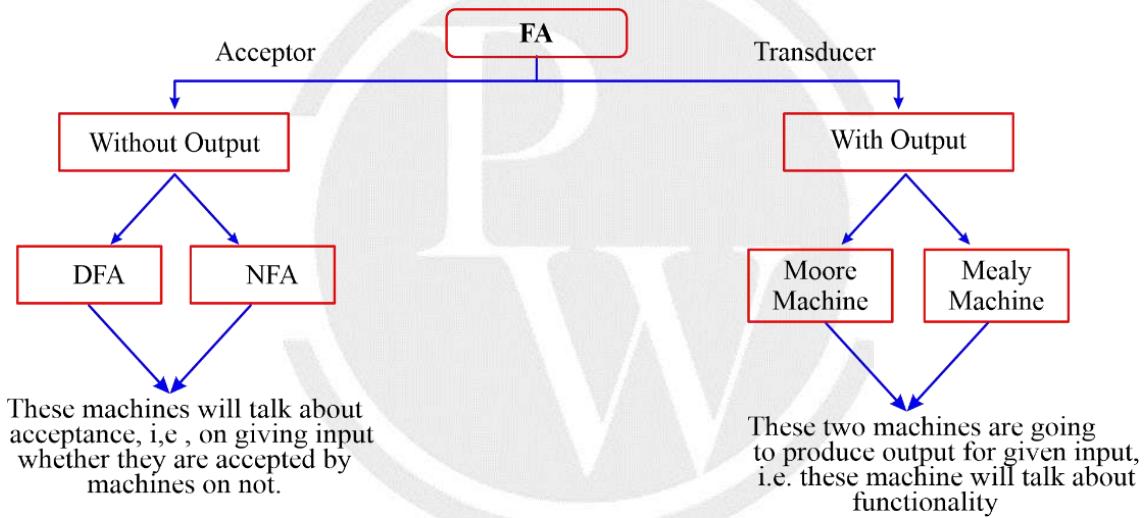
2.5.4 Relation between NFA with epsilon, NFA without epsilon, and DFA



2.6 Regular language Representation



2.7 FA Classification



2.8 Moore machine and mealy machine

Moore machine	Mealy machine
Transition Function $\delta : Q \times \Sigma \rightarrow Q$ Output Function $\lambda : Q \rightarrow \Delta$ Output is associated with every state.	$\delta : Q \times \Sigma \rightarrow Q$ $\lambda : Q \times \Sigma \rightarrow \Delta$ Output is associated with transition.

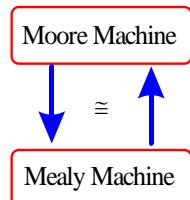
2.8.1 Example for Moore machine and Mealy machine

Moore	Mealy
<p> $Q = \{A, B\}$ $\Sigma = \{0, 1\}$ $\Delta = \{x, y\}$ Output is associated with state. </p> <p> I/P O/P 011 xyx Ignore </p> <p> $A \xrightarrow{0} A \xrightarrow{1} B \xrightarrow{1} A$ x y x </p> <p> This is not expected. </p> <ul style="list-style-type: none"> Extra one output is produced other than desired output. So, we can ignore this extra output as this is the machine property. 	<p> $Q = \{A, B\}$ $\Sigma = \{0, 1\}$ $\Delta = \{x, y\}$ Output is associated with transition. </p> <p> I/P O/P 011 0 $E \xrightarrow{0} E \xrightarrow{1} F \xrightarrow{1} E$ x y y </p> <ul style="list-style-type: none"> No extra output is produced. For 3 length input, we are getting the 3 length output.

2.9 Difference between Moore machine and Mealy machine

		Moore machine	Mealy machine
1.	δ	$Q \times \Sigma \rightarrow Q$	$Q \times \Sigma \rightarrow Q$
2.	λ	$Q \rightarrow \Delta$	$Q \times \Sigma \rightarrow \Delta$
3.	Length of O/p	If n length I/P (assume 1 length O/P symbol is taken at each state) then O/P length is (n+1).	If n length input (assume 1 length O/P symbol is taken at each transition) is given then O/P length is n.
4.	By default	DFA no final state.	DFA no final state.

2.10 Construction of FA with output



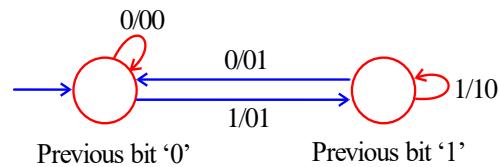
Note :

For every problem, if moore machine exists the we can also construct equivalent mealy machine.

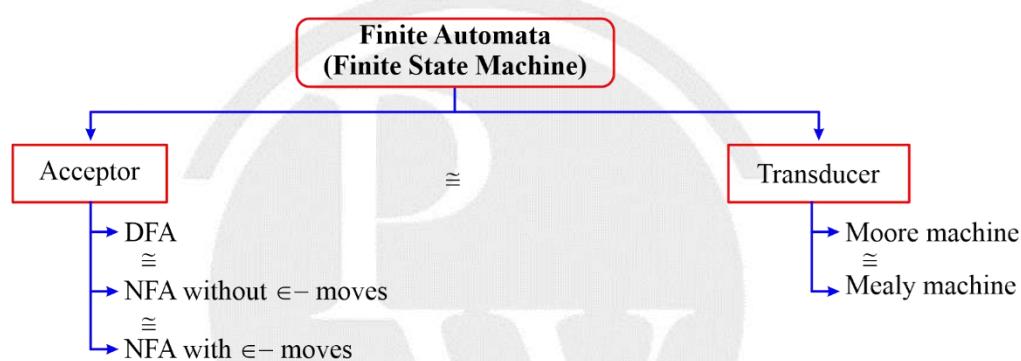
Example:

Sum of present bit and previous bit.

Mealy Machine



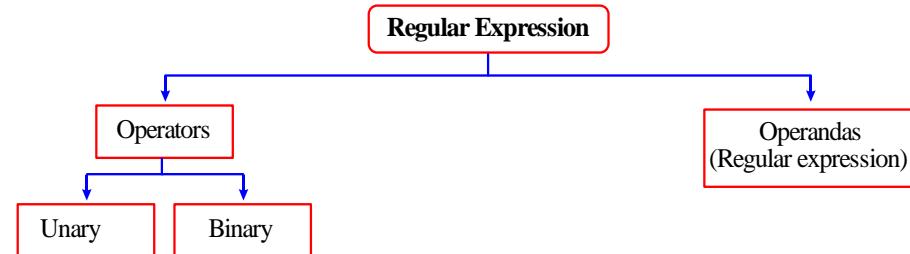
2.11 Classification of Finite State Machine (FSM)



2.12 Regular Expression

Definition:

- Regular expression represents a regular language.
- It describes a regular set.
- L (regular expression) is regular set.
- It is a kind of declarative way to represents a regular language.
- Regular expression generates a regular set.
- It uses 4 operators to represent a regular language.



2.13 Operators of Regular Expression

1. OR (+)
 2. Concatenation(.)
 3. Kleenestar (*)
 4. Kleenestar (+)
- Binary Operator
Unary Operator

Regular Expression	Equivalent Regular Set
$a + b$	$L(a + b) = \{a, b\}$
$a + a = a$	$L(a + a) = \{a\}$
$a + \phi = a$	$L(a + \phi) = \{a\}$
$a + \epsilon = \epsilon + a$	$L(a + \epsilon) = \{a, \epsilon\}$
$\epsilon + \epsilon = \epsilon$	$L(\epsilon + \epsilon) = \{\epsilon\}$
$\phi + \phi = \phi$	$L(\phi + \phi) = \{\phi\}$
$a \cdot b = ab$	$L(a \cdot b) = \{ab\}$
$a \cdot \epsilon = a$	$L(a \cdot \epsilon) = \{a\}$
$\epsilon \cdot \epsilon = \epsilon$	$L(\epsilon \cdot \epsilon) = \{\epsilon\}$
$\phi \cdot \phi = \phi$	$L(\phi \cdot \phi) = \{\phi\}$
$a \cdot a = aa = a^2$	$L(a \cdot a) = aa = \{a^2\}$

2.14 Kleene star/ kleene closure / closure

R^* (Kleene closure of R):

$$R^* = R^0 + R^1 + R^2 + R^3 + \dots = \epsilon + R + RR + RRR + \dots$$

Example:

$$\begin{aligned} a^* &= \{a^0, a^1, a^2, a^3, \dots\} = \{\epsilon, a, aa, aaa, \dots\} = \text{Set of all strings over } a. \\ \phi^* &= \phi^0 + \phi^1 + \phi^2 + \phi^3 + \dots = \epsilon + \phi + \phi + \phi + \dots = \epsilon + \phi = \epsilon \end{aligned}$$

2.15 Positive Closure (Kleeme Plus)

R^+ (Positive Closure of R):

$$R^+ = R^1 + R^2 + R^3 + \dots$$

$R^* = R^{\geq 0}$ i.e. repeat R any number of time

$R^+ = R^{\geq 1}$ i.e. repeat R atleast 1 time.

$$R^* = R^+ + R^0$$

Example:

$$(1) a^+ = \{a, aa, aaa, \dots\} = \{a^n \mid n \geq 1\}$$

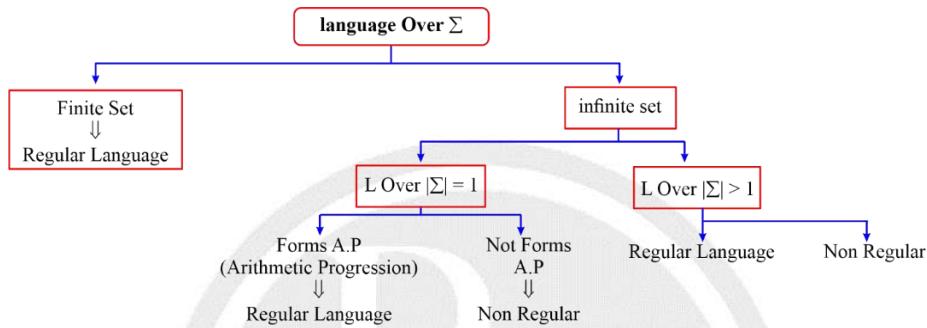
$$(2) \epsilon^+ = \epsilon = \epsilon^*$$

$$(3) \phi^+ = \phi$$

Properties :

		OR	Concatenation
1.	Identity	ϕ	\in
2.	Associative	Yes	Yes
3.	Commutative	Yes	No
4.	Annihilator	Σ^*	ϕ

2.16 Identification of Regular Languages



1. $L = \{a^n b^m \mid n, m \geq 0\} = a^* b^*$ (Regular language)
2. $L = \{a^n b^m \mid m < n < 10\}$ (Finite Set, Regular language)
3. $L = \{a^n b^m \mid m > n > 10\}$ (Non Regular language)
4. $L = \{a^n b^m \mid m = n, m < 10\}$ (Regular language)
5. $L = \{a^n b^m \mid m = n, m > 10\}$ (Non Regular language)
6. $L = \{a^m b^n \mid \gcd(m, n) = 1\}$ (Non Regular language)
7. $L = \{a^m b^n \mid \text{LCM}(m, n) = 1\}$ (Regular language)
8. $L = \{a^n b^n \mid n \geq 0\}$ (Non Regular language)
9. $L = \{a^n b^{2n} \mid n \geq 0\}$ (Non Regular language)
10. $L = \{a^m b^n \mid m = \text{even}, n = \text{odd}\}$ (Regular language)
11. $L = \{a^m b^n \mid m = n = \text{even}\}$ (Non Regular language)

OR

$$L = \{a^{2n} b^{2n} \mid n = \text{even}\}$$

12. $L = \{a^*\}$ over $\Sigma = \{a\}$

= Regular language

13. $L = \{a^{2n} \mid n \geq 0\}$ over $\Sigma = \{a\}$.

$$L = a^{2n} = (aa)^*$$

= Regular language

14. $L = \{a^{\text{Prime}}\}$ over $\Sigma = \{a\}$

= Non regular language

15. $L = \{a^{n^2} \mid n \geq 0\}$ over $\Sigma = \{a\}$

$L = \{\epsilon, a, aaaa, a^9, a^{16}, \dots\}$, FA Not possible for L. So, it's Non-Regular language.

16. $L = \{a^{2^n} \mid n > 0\}$ over $\Sigma = \{a\}$

Non Regular language

17. $L = \{a^{2^n} \mid n \leq 10\}$

= Regular language

18. $L = \{a^{n!} \mid n \geq 100\}$ over $\Sigma = \{a\}$

= Non Regular language

19. $L = \{a^{n^n} \mid n \geq 10\}$ over $\Sigma = \{a\}$

= Non Regular language

20. $L = \{a^{\text{Prime}}\}^*$ over $\Sigma = \{a\}$

$L = \text{complement of } \{a\} = \Sigma^* - \{a\} = \{\epsilon, a^2, a^3, a^4, a^5, a^6, a^7, \dots\}$ = Regular language

21. $L = \{a^{\text{prime}} \mid \text{prime} < 100\}$ is finite language (regular)

22. $L = \{w \# w \mid w \in a^*\}$

$$L = \{a^n \# a^n\}$$

↑
dependency

= Non Regular language

23. $L = \{w \# w \mid w \in (a+b)^*\}$

= Non Regular language

24. $L = \{w \# w^R \mid w \in a^*\}$ is non regular

$$L = \{a^n \# a^n\}$$

↑
dependency

25. $L = \{w x w \mid w \in \{a, b\}^*, x \in \{a, b\}^+\}$

Put $w = \epsilon$, and $x = (a + b)^+$

$$L = (a + b)^+$$

= Regular language

26. $L = \{x w w \mid w \in \{a, b\}^* x \in \{a, b\}^+\}$

$$L = (a + b)^+$$

= Regular language

27. $L = \{w w^R x \mid w \in \{a, b\}^*, x \in \{a, b\}^+\}$

$$L = (a + b)^+$$

= Regular language

28. $L = \{w x w^R \mid w \in (a + b)^*, x \in \{a, b\}^+\}$

$$L = (a + b)^+$$

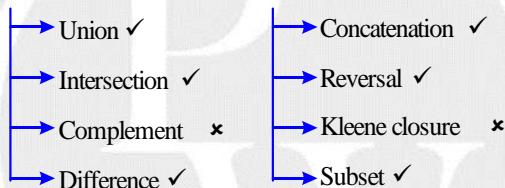
= Regular language

29. $L = \{x w w^R \mid w \in \{a, b\}^*, x \in \{a, b\}^+\}$

$$L = (a + b)^+$$

2.17 Closure Properties of Regular Languages

1. Closure properties for finite languages:



2.18 Table for FINITE sets

		Finite sets	Closed/Not Closed
(1)	Union (\cup)	$F_1 \cup F_2 \Rightarrow$ Finite	✓
(2)	Intersection (\cap)	$F_1 \cap F_2 \Rightarrow$ Finite	✓
(3)	Complement (\bar{L})	$\bar{F} \Rightarrow$ NOT finite	✗
(4)	$L_1 - L_2$	$F_1 - F_2 \Rightarrow$ Finite	✓
(5)	$L_1 \cdot L_2$	$F_1 \cdot F_2 \Rightarrow$ Finite	✓
(6)	L^*, L^+	$F^*, F^+ \Rightarrow$ May or may not be finite	✗
(7)	Subset (L)	Subset (Finite set) \Rightarrow Finite	✓

2.18.1 Table for Infinite sets

		Infinite sets
(1)	Union (\cup)	Infinite \cup Infinite \Rightarrow Infinite
(2)	Intersection (\cap)	Infinite \cap Infinite \Rightarrow Need not be infinite
(3)	Complement (\bar{L})	Complement of Infinite \Rightarrow May or may not be infinite
(4)	$L_1 - L_2$	Need not be infinite
(5)	$L_1 \cdot L_2$	Infinite
(6)	L^*, L^+	Infinite
(7)	Subset (L)	Need not be infinite

2.19 Closure Properties of Regulars

$L_i \rightarrow$ Regular

1.	\cup	2.	\cap
3.	\bar{L}	4.	$L_1 - L_2$
5.	$L_1 \cdot L_2$	6.	L^{Rev}
7.	L^+	8.	L^*
9.	Subset (L) is not closed for regular languages	10.	Prefix (L)
11.	Suffix (L)	12.	Substring (L)
13.	Substitution (L)	14.	Homomorphism (L)
15.	ϵ -free homomorphism (L)	16.	$h^{-1}(L)$ (Inverse homomorphism (L))
17.	L_1/L_2 (Quotient)	18.	Symmetric difference
19.	Half (L)	20.	Second half (L)
21.	One-third (L) [$1/3(L)$]	22.	Middle $1/3(L)$
23.	Last $1/3(L)$	24.	New $(L_1, L_2) = \text{Suffix}(L_1 \cup L_2^+)$
25.	Finite Union	26.	Finite Intersection
27.	Finite Difference	28.	Finite Concatenation

29.	Finite Subset	30.	Finite Substitution
31.	Infinite Union	32.	Infinite \cap
33.	Infinite Difference	34.	Infinite Concatenation
35.	Infinite Subset	36.	Infinite Substitution

- Out of the given 36 closure properties, how many are closed for regular languages?

Subset operation, and 6 infinite operations are not closed, remaining all are closed for regular languages.

Out of 36 operations total 7 operations are not closed.

2.19.1 Operations over regular languages

Examples:

$$(1) \quad \left. \begin{array}{l} L_1 = a^* \\ L_2 = a^+ \end{array} \right\} \Rightarrow L_1 \cdot L_2 = a^*$$

$$(2) \quad \left. \begin{array}{l} L_1 = \phi \\ L_2 = \text{Any} \end{array} \right\} \Rightarrow L_1 + L_2 = L_2$$

$$(3) \quad \left. \begin{array}{l} L_1 = a^* \\ L_2 = b^* \end{array} \right\} \Rightarrow L_1 + L_2 = a^* + b^*$$

$$(4) \quad \left. \begin{array}{l} L_1 = \Sigma^* \\ L_2 = \text{Any language over same } \Sigma \end{array} \right\} \quad L_1 + L_2 = \Sigma^* = L_1$$

$$(5) \quad \left. \begin{array}{l} L_1 = a\Sigma^* \\ L_2 = b\Sigma^* \end{array} \right\} \quad \begin{aligned} L_1 + L_2 &= a\Sigma^* + b\Sigma^* \\ &= (a+b)\Sigma^* \\ &= \Sigma^+ \end{aligned}$$

Note :

$$1. \quad \left. \begin{array}{l} L_1 = \phi \\ L_2 = \text{Any} \end{array} \right\} \Rightarrow L_1 \cdot L_2 = \phi$$

$$2. \quad \left. \begin{array}{l} L_1 = \Sigma^* \\ L_2 = \text{Any} \end{array} \right\} \Rightarrow L_1 + L_2 = L_2$$

2.19.2 Properties of regular languages

I. If both L_1 and L_2 are regular sets then $L_1 \cap L_2$ is Regular.

II. IF $L_1 \cap L_2$ is regular set then

- L_1 “need not be regular”

- L_2 “need not be regular”.

III. If L is regular then \bar{L} is regular.

IV. If \bar{L} is regular then L is regular.

V. L is regular iff \bar{L} is regular

VI. L is not regular iff \bar{L} is not regular.

Example:

$a^n b^n$ is not regular and $\overline{a^n b^n}$ is not regular.

2.19.3 Arden's Lemma and Kleene Method

Arden's Lemma:

If $R = Q + RP$ and P does not contain \in then $R = QP^*$

$$R = Q + RP \quad \dots(1)$$

$$= Q + (Q + RP) P = Q + QP + RP^2 \quad \dots(2)$$

Substitute R one more time in equation (2)

$$R = Q + QP + QP^2 + RP^3$$

If we do repetitive substitution infinite time, we will get $R = QP^*$

Kleene Method:

Kleene Method

Dynamic programming $O(n^3)$

 R_{ij}^k where i is the initial state, j is the final state
 k represents number of states

$$R_{ij}^k = R_{ij}^{k-1} + R_{ik}^{k-1} (R_{kk}^{k-1})^* R_{kj}^{k-1}$$

Regular Grammar

It represents a regular set

It can be either left linear grammar (LLG) or right linear grammar (RLG)

LLG	RLG
Each production in LLG follows $V \rightarrow VT^*$ OR $V \rightarrow T^*$	Each production in RLG follows $V \rightarrow T^*V$ OR $V \rightarrow T^*$

2.20 Identify the Language Generated by Regular Grammar

Regular Grammar	Regular Language
1. $S \rightarrow \in$	$L = \{\in\}$
2. $S \rightarrow \in a$	$L = \{\in, a\}$
3. $S \rightarrow aa \mid abc \mid d$	$L = \{aa, abc, d\}$

4.	$S \rightarrow Aa$ $A \rightarrow b$	By default S is a start symbol here. $L = \{ba\}$
5.	$S \rightarrow Aa$	Useless production. It has no meaning. $L = \{\} = \emptyset$
6.	$S \rightarrow \boxed{Ab} b$ ↓ Useless	$L = \{b\}$
7.	$\uparrow S \rightarrow Aa Ba$ $A \rightarrow a$ $B \rightarrow b$ Look from bottom to top	$L = \{aa, bb\}$
8.	$\uparrow S \rightarrow Aa$ $A \rightarrow \epsilon b$	$L(A) = \{\epsilon, b\}$ $L = L(S) = L(A) \cdot a$ $= \{a, ba\}$

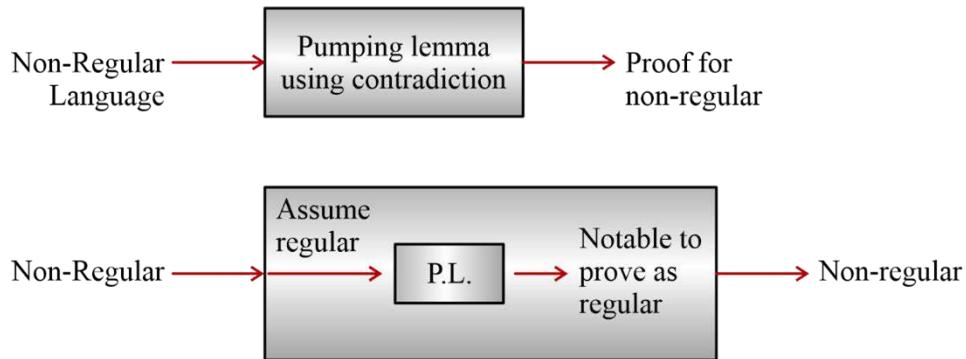
Regular Language	Regular Grammar
a^*	(I) $S \rightarrow Sa \epsilon$ OR (II) $S \rightarrow aS \epsilon$
a^+	(I) $S \rightarrow Sa a$ OR (II) $S \rightarrow aS a$
$(a + b)^*$	(I) $S \rightarrow Sa Sb \epsilon$ OR (II) $S \rightarrow aS bS \epsilon$
$(a + b)^+$	(I) $S \rightarrow Sa Sb a b$ OR (II) $S \rightarrow aS bS a b$
$(ab)^*$	(I) $S \rightarrow Sab \epsilon$ OR (II) $S \rightarrow abS \epsilon$

2.21 Pumping-Lemma (PL)

2.21.1 Pumping Lemma for Regular Languages



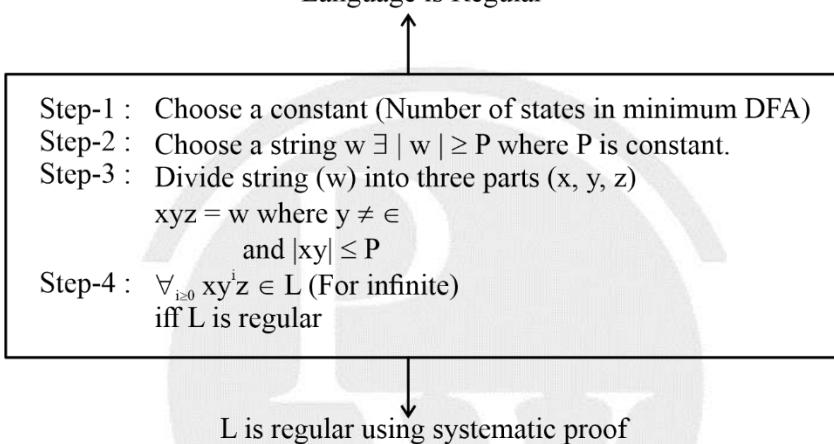
2.21.2 Pumping Lemma for Non-Regular Languages using contradiction



2.21.3 Proof for Regular Languages

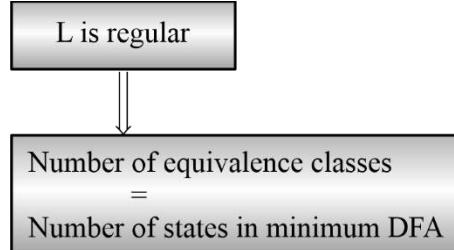
- If L is Regular language, then how pumping lemma proves it as regular?

Language is Regular



2.22 Equivalence Classes

- L is regular iff L has finite number of equivalence classes.
- L is not regular iff L has infinite equivalence classes.

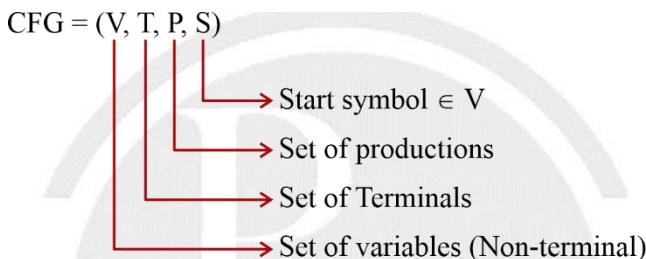


3

PUSH DOWN AUTOMATA

3.1 Context Free Grammar

CFG: It represents a Context Free Language.



- Rule of each production in P :
 $V \rightarrow (V \cup T)^*$
 V = only 1 variable in LHS
- To derive a string, following derivations can be used.
 1. **Linear Derivation:** Linear derivation is two types
 - (a) Left Most Derivation (LMD)
 - (b) Right Most Derivation (RMD)
 2. **Non- linear Derivation:** Non- linear Derivation OR Parse Tree OR Derivation Tree

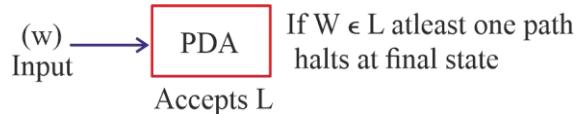
3.2 Types of Context Free Grammars

There are two types of CFG:

1. **Ambiguous CFG:** At least one string has more than one derivation.
2. **Unambiguous CFG:** Every string (w) generated by CFG has exactly one derivation.

3.3 Pushdown Automata (PDA)

PDA accepts context free language (CFL). PDA also called as NPDA.



3.4 PDA acceptance mechanisms

- PDA acceptance mechanisms are three types:
 - PDA acceptance using final state.
 - PDA acceptance using empty stack.
 - PDA acceptance using both final state and Empty stack.
All PDA acceptance mechanisms are equivalent.
- DPDA acceptance mechanism are two types:
 - DPDA using final stack.
 - DPDA using both final state and empty stack.

3.5 PDA configuration

$PDA = (Q, \Sigma, \delta, q_0, F, Z_0, \Gamma)$.

where Q = Set of states

Σ = input alphabet

δ = Transition Function (PDA/NPDA $\delta : Q \times \Sigma \times \Gamma \rightarrow 2^{Q \times \Gamma^*}$)

F = Set of Final state

Z_0 = Bottom symbol or initially TOS

Γ = Stack Alphabet

- DPDA transition Function is $[\delta : Q \times \Sigma \times \Gamma \rightarrow Q \times \Gamma^*]$
- Difference between DPDA and PDA

DPDA	PDA
[1] $\delta : Q \times \Sigma \times \Gamma \rightarrow Q \times \Gamma^*$	$\delta : Q \times \Sigma \times \Gamma \rightarrow 2^{Q \times \Gamma^*}$
[2] Every DPDA is PDA	Every PDA need not be DPDA
[3] DPDA acceptance with <ol style="list-style-type: none"> Final state mechanism Both Final and empty stack 	PDA acceptance with <ol style="list-style-type: none"> Final state mechanism Empty stack mechanism Both Empty stack and Final state

- Relation between Regular, DCFL and CFL.

Regular languages(FA)
 DCFLs (DPDA)
 CFLs (PDA or NPDA)

- I If L is regular language, then it is also DCFL and CFL.
- II Every DCFL is CFL, but it need not be regular.

3.6 Closure Properties of CFLs

Operation	Closed / Not Closed
Union ($L_1 \cup L_2$)	✓
Intersection ($L_1 \cap L_2$)	✗
Complement (\bar{L})	✗
Set difference ($L_1 - L_2$)	✗
Concatenation ($L_1 \cdot L_2$)	✓
Reversal (L^{Rev})	✓
Kleene Closure (L)	✓
Kleene Plus (L^*)	✓
Subset (L)	✗
Prefix (L)	✓
Suffix (L)	✓
Substring (L)	✓
Substitution (L)	✓
Homomorphism (L)	✓
\in - free Homomorphism $h(L)$	✓
Inverse Homomorphism $h^{-1}(L)$	✓
quotient (L_1, L_2) = L_1/L_2	✗
Symmetric difference (L_1, L_2)	✗
Finite Union	✓
Finite Intersection	✗
Finite difference	✗
Finite concatenation	✓
Finite subset	✓
Finite substitution	✓
Infinite Union	✗
Infinite intersection	✗
Infinite difference	✗
Infinite concatenation	✗
Infinite Subset	✗
Infinite Substitution	✗
Union with Regular ($L \cup \text{Regular}$)	✓
$L \cup \text{Regular}$	✓
$L - \text{Reg}$	✓
$\text{Reg} - L$	✗

Note :(i) $CFL \cap CFL = \text{Need not be CFL}$

- (ii) $CFL \cap \text{Regular} = CFL (\text{Need not be DCFL})$
- (iii) $CFL \cap \text{DCFL} = \text{May or may not be CFL}$
- (iv) $CFL \cap \text{Finite} = \text{Finite}$
- (v) $CFL \cap \text{infinite} = \text{Need not be CFL}$

3.7 Closure Properties of DCFLs

Operation	Closed / Not Closed
Union ($L_1 \cup L_2$)	✗
Intersection ($L_1 \cap L_2$)	✗
Complement (\bar{L})	✓
Set difference ($L_1 - L_2$)	✗
Concatenation ($L_1.L_2$)	✗
Reversal (L^{Rev})	✗
Kleene Closure ($L = L^*$)	✗
Kleene Plus ($L = L^+$)	✗
Subset (L)	✗
Prefix (L)	✓
Suffix (L)	✗
Substring (L)	✗
Substitution (L)	✗
Homomorphism (L)	✗
\in - free Homomorphism $h(L)$	✗
Inverse Homomorphism $h^{-1}(L)$	✓
quotient (L_1, L_2)	✗
Symmetric difference (L_1, L_2)	✗
Finite Union	✗
Finite Intersection	✗
Finite difference	✗
Finite concatenation	✗
Finite subset	✓
Finite substitution	✗
Infinite Union	✗
Infinite intersection	✗
Infinite difference	✗
Infinite concatenation	✗
Infinite Subset	✗
Infinite Substitution	✗
Union with Regular ($L \cup \text{Regular}$)	✓
$L \cup \text{Regular}$	✓
$L - \text{Regular}$	✓
Regular – L	✓

Note :

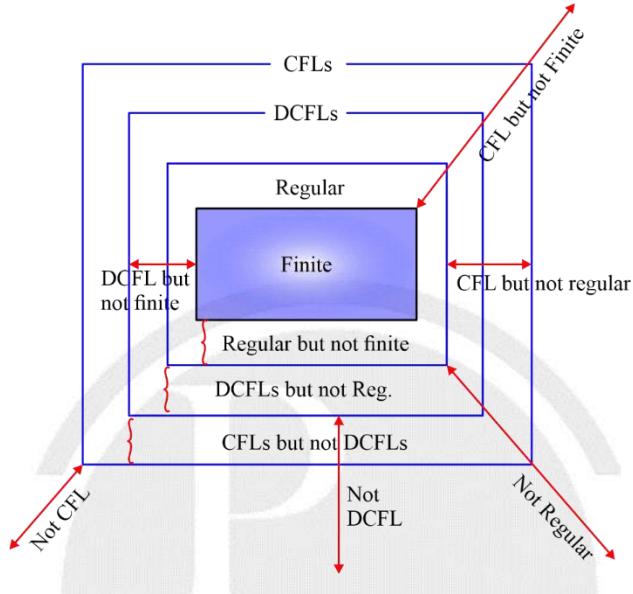
DCFL \cup Regular : DCFL
DCFL \cap Regular : DCFL
DCFL - Regular : DCFL
Regular - DCFL : DCFL
DCFL \cup CFL : CFL (need not be DCFL)
DCFL \cap CFL : Need not be CFL
DCFL - CFL : Need not be CFL
CFL - DCFL : Need not be CFL
DCFL \cup Finite : DCFL
DCFL \cap Finite : Finite
DCFL - Finite : DCFL
Finite - DCFL : Finite

3.8 Comparison of Regular Grammars and CFGs

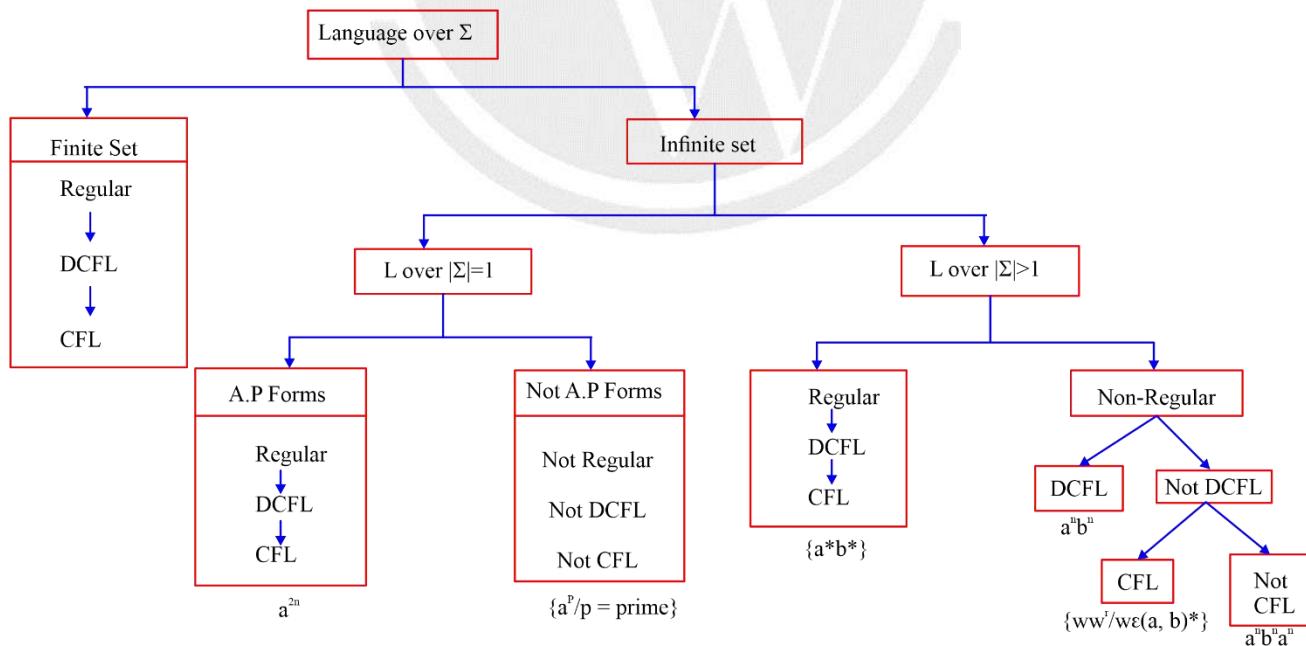
- CFL and CFG both are equivalent. $CFL \equiv CFG$
- LLG: Left Linear Grammar, RLG: Right Linear Grammar, RG: Regular Grammar, LG: Linear Grammar, CFG: Context Free Grammar.
- Every LLG is RG
- Every RLG is RG
- Every RG is LG
- Every RG is CFG
- Every LG is CFG
- Every RG need not be RLG
- Every RG need not be LLG
- Every LG need not be RG
- Every CFG need not be RLG
- Every CFG need not be LG

3.9 Context Free Languages and DCFLs

I. Comparison of various languages



II. Identification of Regulars, DCFLs, and CFLs



[1] $L = \{a^n b^n c^k \mid n, m, k \geq 0\}$

$= a^* b^* c^*$

= Regular

[2] $L = \{a^n b^n \mid n \geq 0\}$

$L = \text{DCFL}$ but not regular

$S \rightarrow aSb \mid \in$

[3] $L = \{a^n b^{2n} \mid n \geq 0\}$

$L = \text{DCFL}$ but not regular

$S \rightarrow aSbb \mid \in$

[4] $L = \{a^{2n} b^n \mid n \geq 0\}$

$L = \text{DCFL}$

$S \rightarrow aaSb \mid \in$

[5] $L = \{a^{2n} b^{2n} \mid n \geq 0\}$

$L = \text{DCFL}$

$S \rightarrow aaSbb \mid \in$

[6] $L = \{a^n b^n c^*\}$ Assume always $n \geq 0$ in all examples

OR

$= \{a^m b^n c^* \mid m = n\}$

DCFL but not regular

[7] $L = \{a^n b^* c^n\}$ DCFL

[8] $L = \{a^* b^n c^n\}$ DCFL

[9] $L = \{a^m b^n c^* \mid m \neq n\}$

DCFL

[10] $L = \{a^m b^n c^* \mid m < n\}$

DCFL

[11] $L = \{a^m b^n c^* \mid m > n\}$

DCFL

[12] $L = \{a^m b^n c^* \mid c \leq n\}$

DCFL

[13] $L = \{a^m b^n c^* \mid m \geq n\}$

DCFL

[14] $L = \{a^m b^n c^{m+n} \mid m, n \geq 0\}$ is DCFL

[15] $L = \{ a^{m+n}b^{n+k}c^{k+m} \mid m, n \geq 0 \}$ is CFL

[16] $L = \{ ww^r \mid w \text{ belongs to } \{a, b\}^* \}$ is CFL but not DCFL

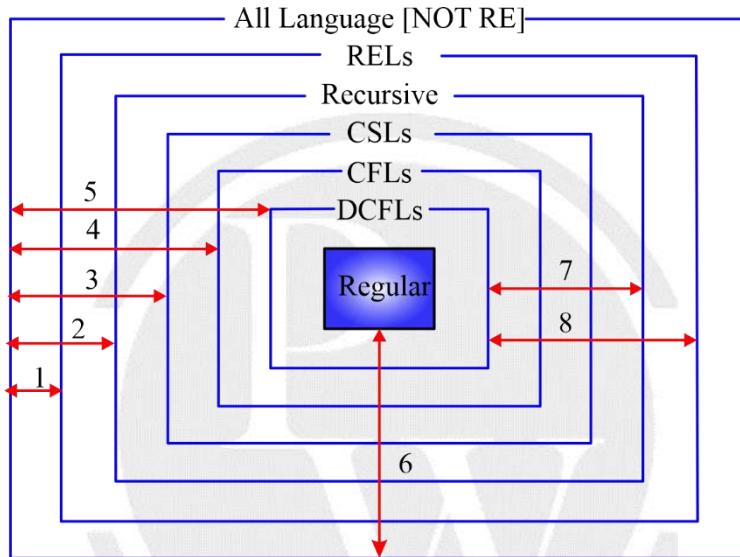
□□□



4

TURNING MACHINE

4.1 Classification of Languages



1. All languages which are not RELs
2. All not recursive languages
3. All not CSLs
4. All not CFLs
5. All not DCFLs
6. All not regulars
7. All recursive languages which are not DCFLs
8. All RELs which are not DCFLs

4.2 There are Two types of TM

- (a) DTM (Deterministic TM)
- (b) NTM (Non-Deterministic TM)

DTM

$$\delta : Q \times \Gamma \rightarrow Q \times \Gamma \times \left\{ \begin{array}{c} L, R \\ \downarrow \quad \downarrow \\ \text{Left} \quad \text{Right} \end{array} \right\}$$

NTM

$$\delta : Q \times \Gamma \rightarrow 2^{Q \times \Gamma \times \{L, R\}}$$

4.3 Unrestricted Grammar (UG) and Context Sensitive Grammar (CSG)

Unrestricted grammar (UG) also called as

- RE grammar
 - Phase structure grammar
- Context sensitive grammar (CSG) is also called as
- Non contracting grammar
 - Bound restricted grammar

4.4 Grammars

Left Linear Grammar (LLG): $V \rightarrow VT^* \mid T^*$

Right Linear Grammar (RLG): $V \rightarrow T^*V \mid T^*$

Linear Grammar (LG): $V \rightarrow T^*VT^* \mid T^*$

Context Free Grammar (CFG): $LHS \rightarrow RHS, |LHS| \leq |RHS|$

Unrestricted Grammar (UG): $LHS \rightarrow RHS$

4.5 Equivalence of various TMs

TM \cong Single tape TM

TM \cong One-way infinite tape TM

TM \cong Two-way infinite tape TM

TM \cong Multi tape and multi head TM

TM \cong Universal TM

TM \cong Two stack PDA

TM \cong Multi stack PDA

TM \cong FA with two stacks

TM \cong FA + R/W tape + Bidirectional head

4.6 Restrictions on TM

(1) If TM tape is read only tape, this TM accepts regular.

TM \cong FA (TM with read only tape)

(2) If TM head is unidirectional then L(TM) = Regular.

(3) If TM tape is read only and unidirectional head then L(TM) = Regular.

(4) If TM always halts then L(TM) is recursive language.

(5) If TM always halts and uses linear bound tape then L(TM) is CSL.

4.7 LBA, HTM and TM

HTM: It is a TM that always halts for every input.

TM: It halts for every valid string and for invalid strings either halts at “non final state” or “never halts”.

LBA: It is HTM but length of the tape we use linearly bounded.

- LBA accepts CSL languages.
- HTM accepts recursive languages.
- TM accepts Recursive Enumerable (RE) languages.

4.8 Recursively Enumerable Language

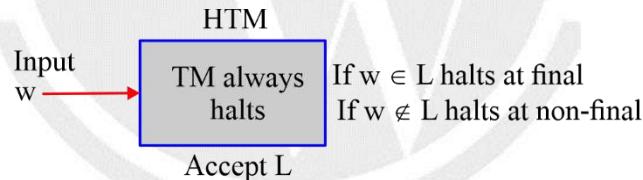
It is also called as:

- Enumerable language
- TM recognizable language
- TM Enumerable language
- Partially decidable language
- Semi-decidable language

4.9 Recursive Language

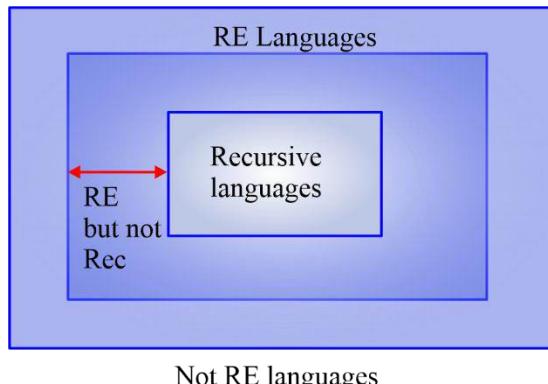
- Recursive language is acceptable by HTM, and hence acceptable by TM.
- Recursive also called as decidable language.
- Recursive also called as Turing decidable language.

If TM always halts, then TM is called as HTM.



4.10 Difference between Recursive and REL

- All Recursive languages are RE languages.



Note :

1. Union:

$$\text{REL} \cup \text{Finite} \Rightarrow \text{REL}$$

$$\text{REL} \cup \text{Regular} \Rightarrow \text{REL}$$

$$\text{REL} \cup \text{CFL} \Rightarrow \text{REL}$$

$$\text{REL} \cup \text{Recursive} \Rightarrow \text{REL}$$

$$\text{REL}_1 \cup \text{REL}_2 \Rightarrow \text{REL}$$

2. Intersection:

$$\text{REL} \cap \text{Finite} \Rightarrow \text{REL} (\text{Finite})$$

$$\text{REL} \cap \text{CFL} \Rightarrow \text{REL}$$

$$\text{REL} \cap \text{Rec} \Rightarrow \text{REL}$$

$$\text{REL}_1 \cap \text{REL}_2 \Rightarrow \text{REL}$$

4.11 Closure Properties of Recursive languages

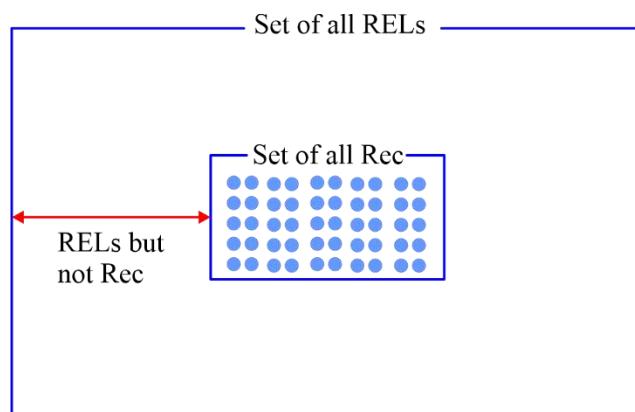
I. The following operations are not closed for recursive languages:

- Subset
- Substitution
- Homomorphism
- Finite substitution
- Infinite union
- Infinite intersection
- Infinite concatenation
- Infinite difference
- Infinite substitution

II. The following operations are closed for recursive languages:

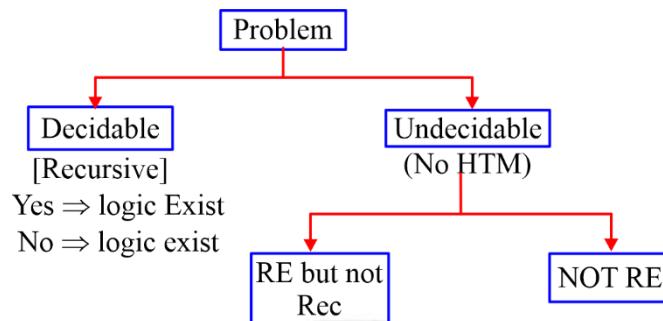
- Complement
- Difference
- Finite difference
- Infinite followed by $\cup, \cap, -, \bullet, \subseteq, f$
- Remember not closed operations

4.12 Complement of Recursive Vs Complement of REL



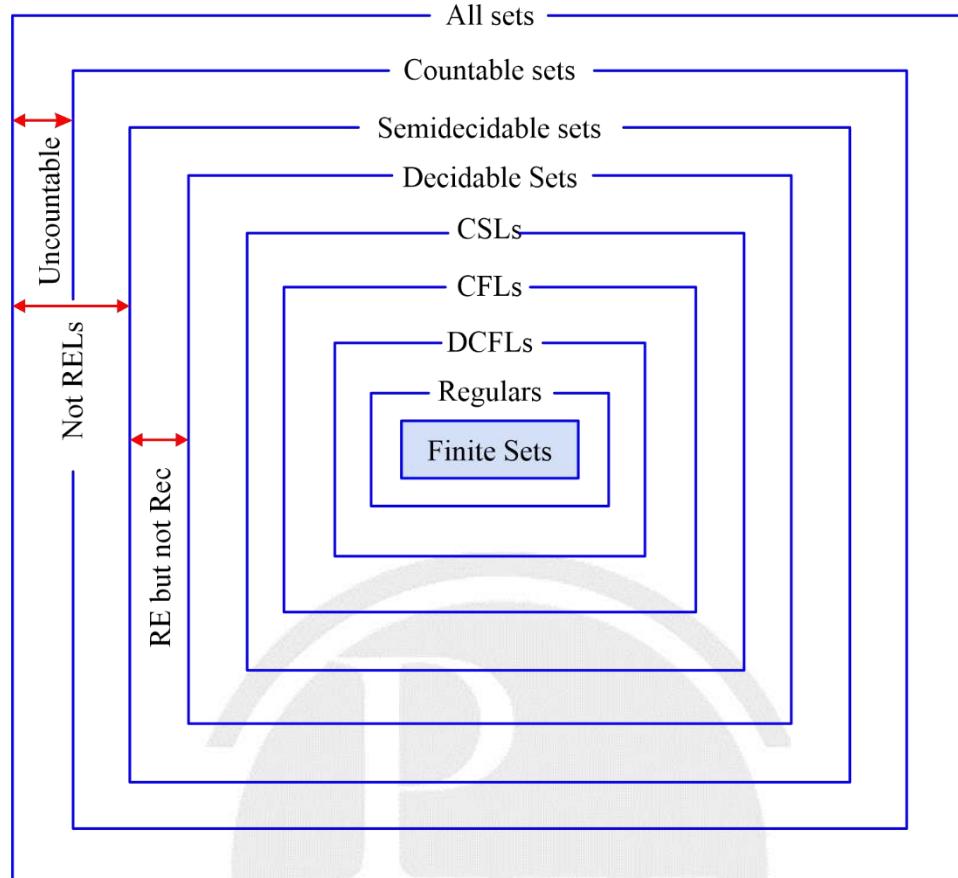
- Complement of Recursive set is Recursive.
- Complement of RE is either Recursive or non-RE.
- Complement of RE never be “RE which is not recursive”.

4.13 Decidable and Undecidable



4.14 Decidable Vs Undecidable, RE Vs Not RE, Countable Vs Uncountable

- [1] Decidable and Undecidable
- ↓
- HTM exist
(Rec language) HTM not exist
(not Rec language)
- [2] RE and Not RE
- ↓
- TM exist TM not exist
- [3] Countable set and Uncountable set
- ↓
- X is countable set (X is not countable set)
iff
X → f(X) is bijective where f(X) is known countable set.

**Note :**

- If problem p is decidable then \bar{p} is also decidable.
- If problem p is Undecidable then \bar{p} is also UD.
- If problem p is RE but not recursive then \bar{p} is not RE.
- If problem p is not RE then \bar{p} is either “Not RE” or “RE but not Recursive”.

4.15 Decision Properties Table

- D: Decidable
- UD: Undecidable

	FA	DPDA	PDA	LBA/HTM	TM
H (Halting)	D	D	D	D	UD
M (Membership)	D	D	D	D	UD
E _m (Emptiness)	D	D	D	UD	UD
F (Finiteness)	D	D	D	UD	UD

T (Totality)	D	D	UD	UD	UD
E _q (Equivalence)	D	D	UD	UD	UD
D (Disjoint)	D	UD	UD	UD	UD
S (Set Containment)	D	UD	UD	UD	UD

4.16 Decidable problem for DFA / NFA/ FA/ Regular

- (1) Halting problem for FA / Reg/ DFA / NFA is decidable.
- (2) Non-halting problem for FA / Reg/ DFA / NFA is decidable.
- (3) Membership problem for FA / Reg/ DFA / NFA is decidable.
- (4) Non-membership problem for FA / Reg/ DFA / NFA is decidable.
- (5) Emptiness for FA / Reg/ DFA / NFA is decidable.
- (6) Non-emptiness problem for FA / Reg/ DFA / NFA is decidable.
- (7) Fitness problem for FA / Reg/ DFA / NFA is decidable.
- (8) Non-fitness problem for FA / Reg/ DFA / NFA is decidable.
- (9) Totality problem for FA / Reg/ DFA / NFA is decidable.
- (10) Non-totality problem for FA / Reg/ DFA / NFA is decidable.
- (11) Equivalence problem for FA / Reg/ DFA / NFA is decidable.
- (12) Non-equivalence problem for FA / Reg/ DFA / NFA is decidable.
- (13) Disjointness problem is decidable for FA / Reg/ DFA / NFA.
- (14) Non-disjointness problem is decidable for FA / Reg/ DFA / NFA.
- (15) Set containment problem for FA / Reg/ DFA / NFA is decidable
- (16) Non-set containment problem for FA / Reg/ DFA / NFA is decidable.

4.17 Decidable problems for CFLs/DCFLs

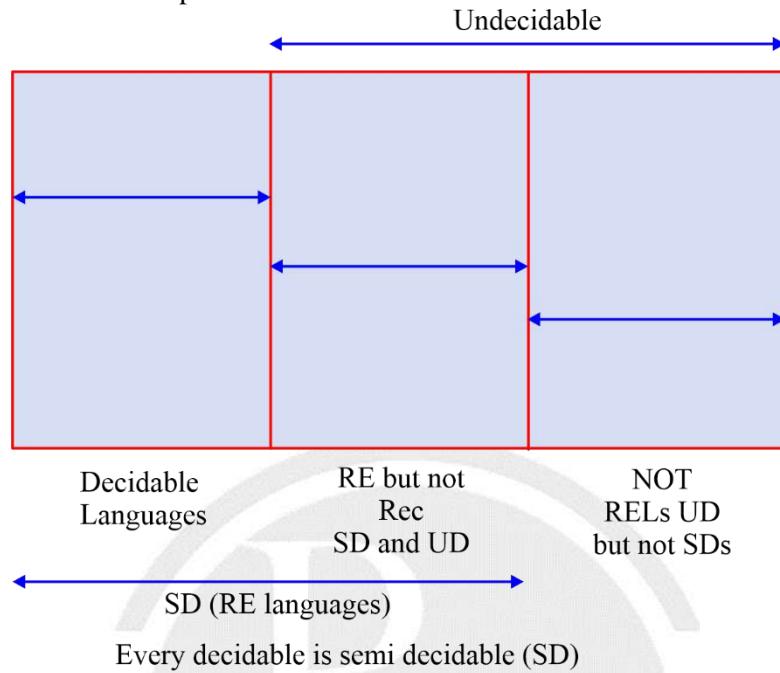
	Problems	DCFLs	CFLs
CYK algo	Halting	D	D
	Non-Halting	D	D
	Membership	D	D
	Non-membership	D	D
Simplification algo	Emptiness	D	D
	Non-emptiness	D	D
	Finiteness	D	D
	Non-finiteness	D	D
Dependency graph	Totality	D	UD
	Non-totality	D	UD
	Equivalence	D	UD
	Disjointness	UD	UD
	Non-disjointness	UD	UD
	Set containment	UD	UD
	Non-set containment	UD	UD

4.18 Decidability problems for Recursive languages

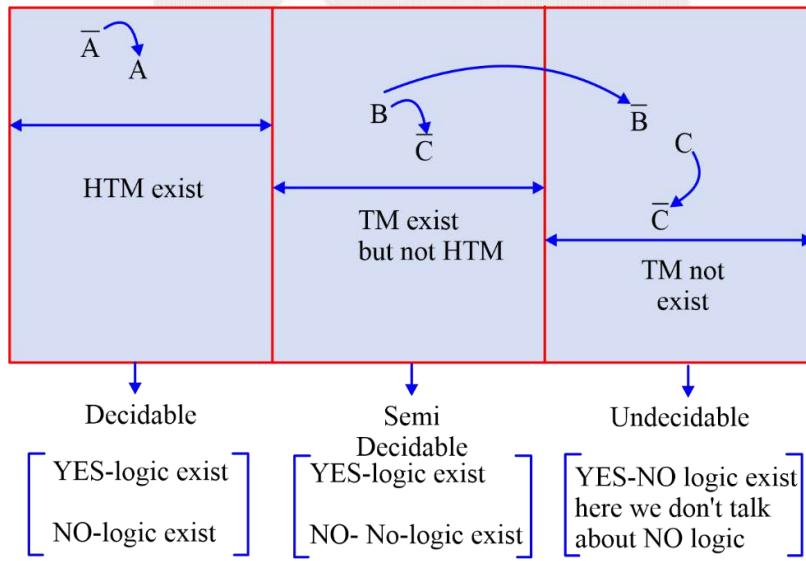
	Problems	Recursive
1.	Halting	D
2.	Non-Halting	D
3.	Membership	D
4.	Non-membership	D
5.	Emptiness	UD [Not REL]
6.	Non-emptiness	UD [RE but not Rec] [SD but UD]
7.	Finiteness	UD [Not RE]
8.	Non-finiteness	UD [Not RE]
9.	Totality	UD [Not RE]
10.	Non-totality	UD [RE but not Rec] [SD but UD]
11.	Equivalence	UD [Not RE]
12.	Non-equivalence	UD [RE but not Rec]
13.	Disjointness	UD [Not RE]
14.	Non-disjointness	UD [RE but not Rec]
15.	Set containment	UD [Not RE]
16.	Non-set containment	UD [RE but not Rec]

4.19 Classification of Languages based on Decidability

All RE languages can be classified into 3 important classes.



4.19.1 Decidability Vs Turing Machine



4.20 Decidable languages

- (1) Finite set \Rightarrow Decidable
- (2) $\Sigma = \{a, b\} \Rightarrow$ Decidable
- (3) $\Sigma \cong$ Set of finite number of symbols \Rightarrow Decidable
- (4) Σ^* over alphabet $\Sigma = \{a, b\} \Rightarrow$ Decidable
- (5) $\{M \mid M \text{ is DFA, } M \text{ accepts } ab\} \Rightarrow$ Decidable

- (6) $\{\text{TM} \mid \text{Number of states in TM} = 2\} \Rightarrow \text{Decidable}$
- (7) $\{\text{TM} \mid \text{TM reaches state q within 100 steps}\} \Rightarrow \text{Decidable}$
- (8) $\{\text{TM} \mid \text{TM accepts REL}\} \Rightarrow \text{Decidable}$



GATE Exam 2024?



PARAKRAM BATCH

ME | CE | EE | EC | CS & IT

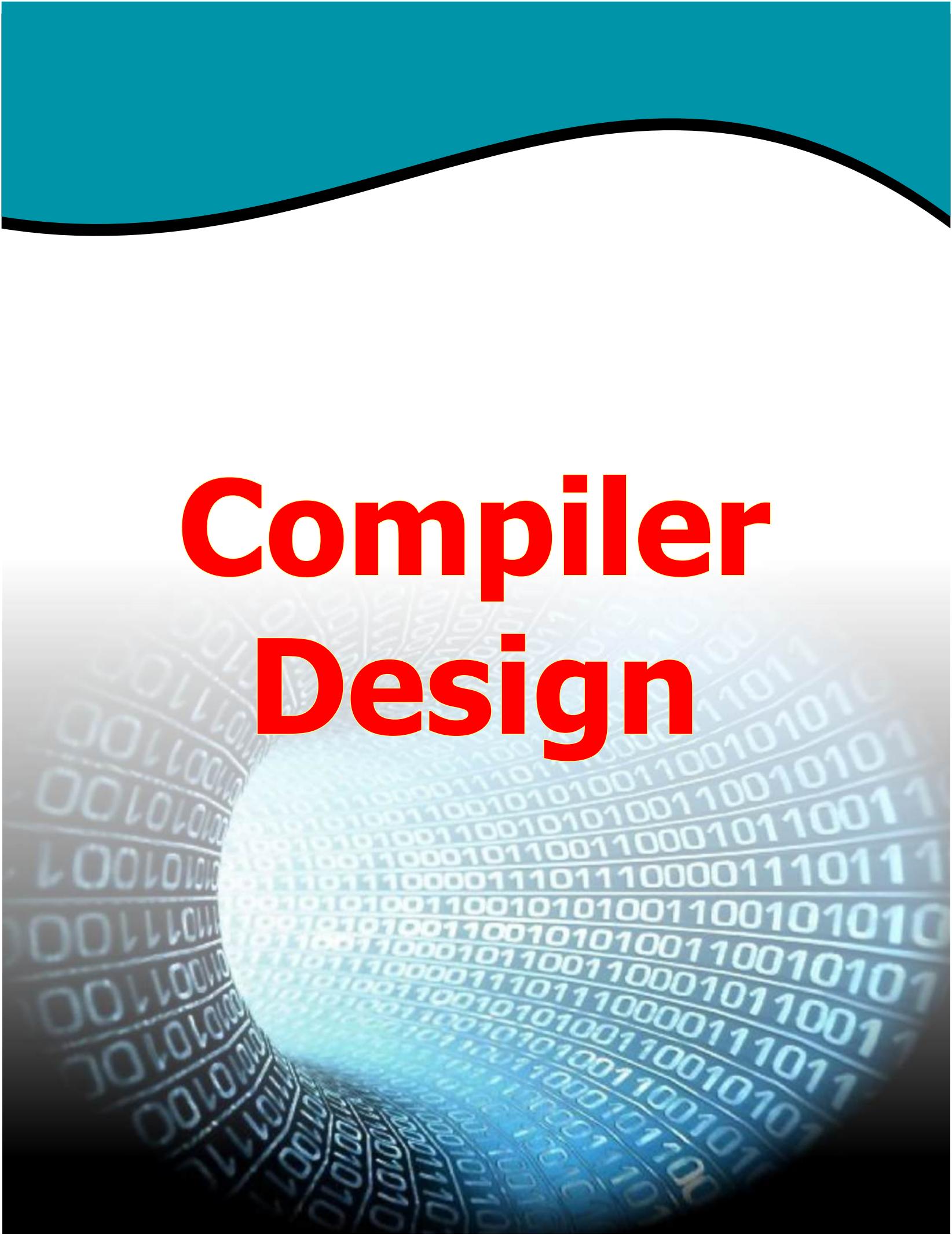
- Live Classes & Recorded Videos
- DPPs & Solutions
- Doubt Solving
- Batches are available in *English*

Weekday & Weekend
Batches Available



JOIN NOW!

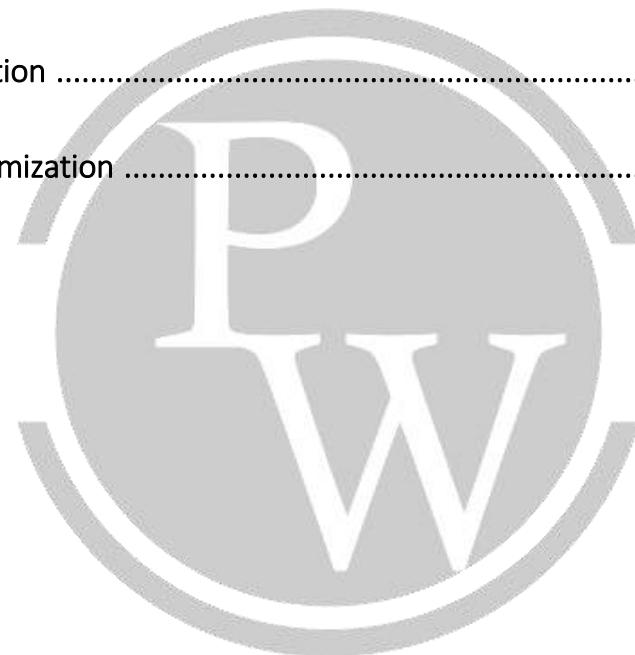
Compiler Design



Compiler Design

INDEX

- | | | |
|----|---|-------------|
| 1. | Introduction and Lexical Analysis | 8.1 – 8.2 |
| 2. | Parsing | 8.3 – 8.11 |
| 3. | Lexical Analysis | 8.12 – 8.13 |
| 4. | Syntax Directed Transaction | 8.14 – 8.16 |
| 5. | Intermediate, Code Optimization | 8.17 – 8.23 |



1

INTRODUCTION AND LEXICAL ANALYSIS

1.1 CD Introduction

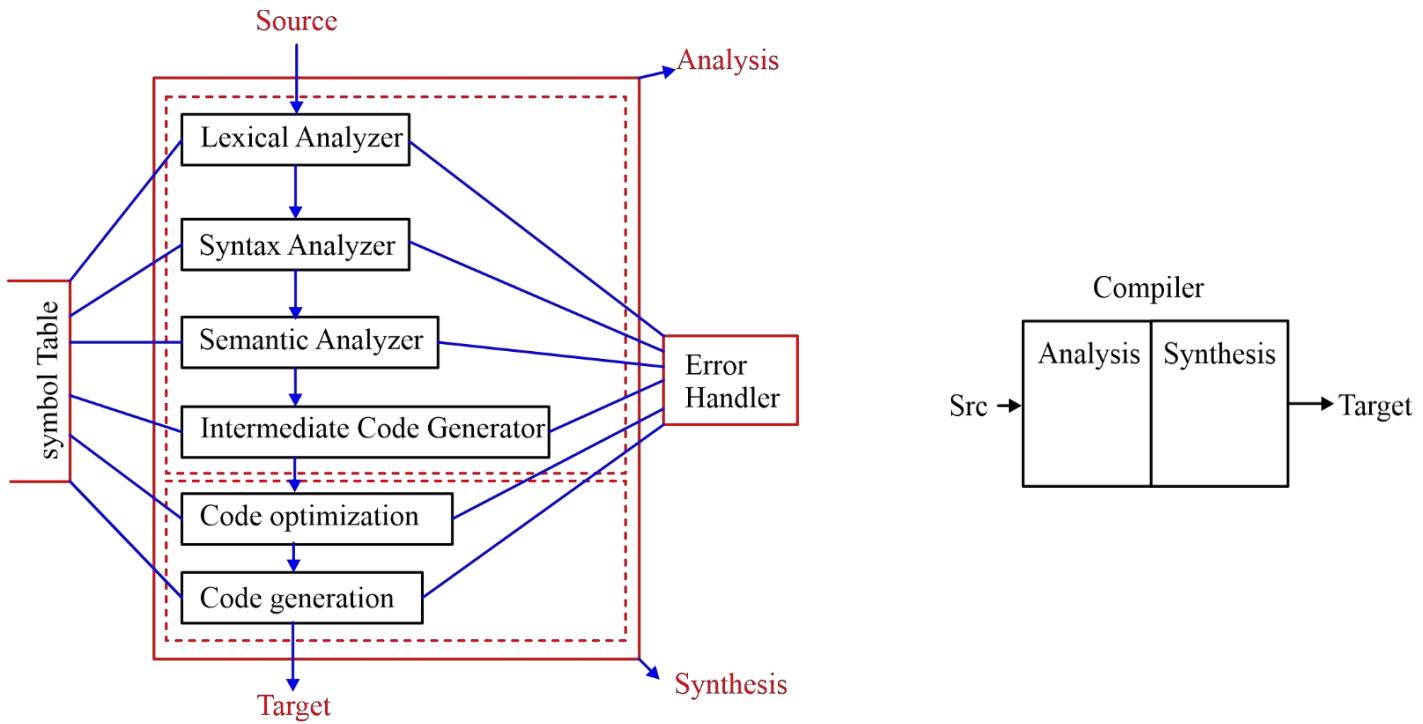
1.1.1 Definition

Convert High Level Language to Low Level Language.



- High Level Language can perform more than one operation in a single Statement.
- Low Level Language can perform atmost one operation in a statement.

1.2 Analysis and Synthesis model of Compiler



- There are 6 phases of the Compiler

1. Lexical Analyzer:

- Program of DFA, it checks for spelling mistakes of program.
- Divides source code into stream of tokens.

2. Syntax Analyzer:

- Checks grammatical errors of the program (Parser).
- Parser is a DPDA.

3. Semantic Analyzer:

Checks for meaning of the program.

Example:

Type miss match, stack overflow

4. Intermediate Code Generation:

- This phase makes the work of next 2 phases much easier.
- Enforces reusability and portability.

5. Code optimization:

- Loop invariant construct
- Common sub expression elimination
- Strength Reduction
- Function inlining

Deadlock elimination

6. Symbol Table:

- (1) Data about Data (Meta data)
- (2) Data structure used by compiler and shared by all the phases.



2

PARSING

2.1 CD - Grammar

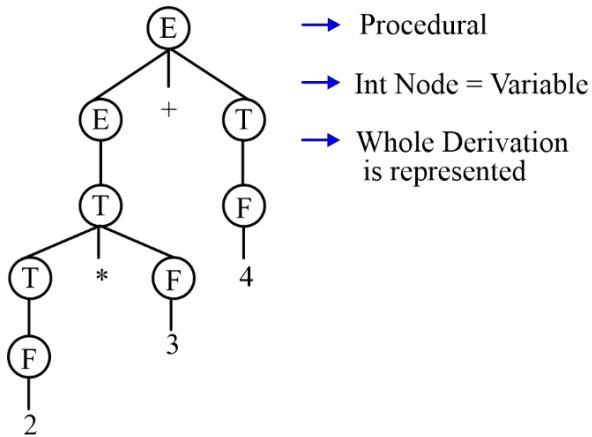
- In compiler we only use: Type – 2 (CFG) and Type – 3 Regular grammar.
- Compiler = Program of Grammar
- Compiler = Membership algorithm
- Every programming Language is Context Sensitive Grammar (Context Sensitive Language)

2.1.1 Parse Tree and Syntax Tree

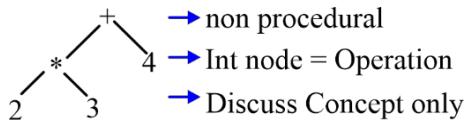
G: $E \rightarrow E + T / T \quad E \rightarrow E + T \rightarrow T + T \rightarrow T * F + T \rightarrow F * F + T \rightarrow 2 * F + T \rightarrow 2 * 3 + T \rightarrow 2 * 3 + F$
 $T \rightarrow T * F / F$

$$E \rightarrow 2 * 3 + 4$$

Parse Tree:

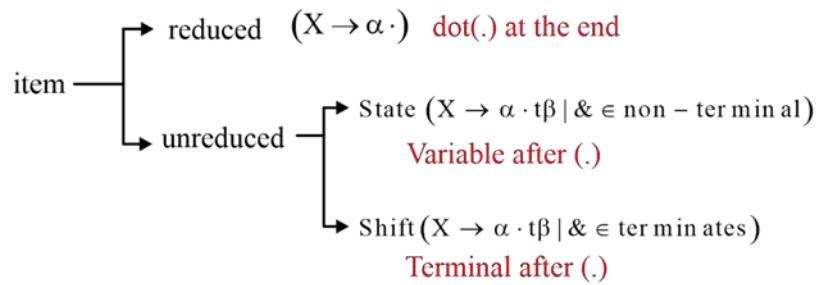


Syntax Tree:



- To check to priority / Associativity:
Randomly derive till you have enough operators, then check which one is done first.
- If priority of 2 operators is same and both Left and Right associative → **Ambiguous Grammar** [Useless]

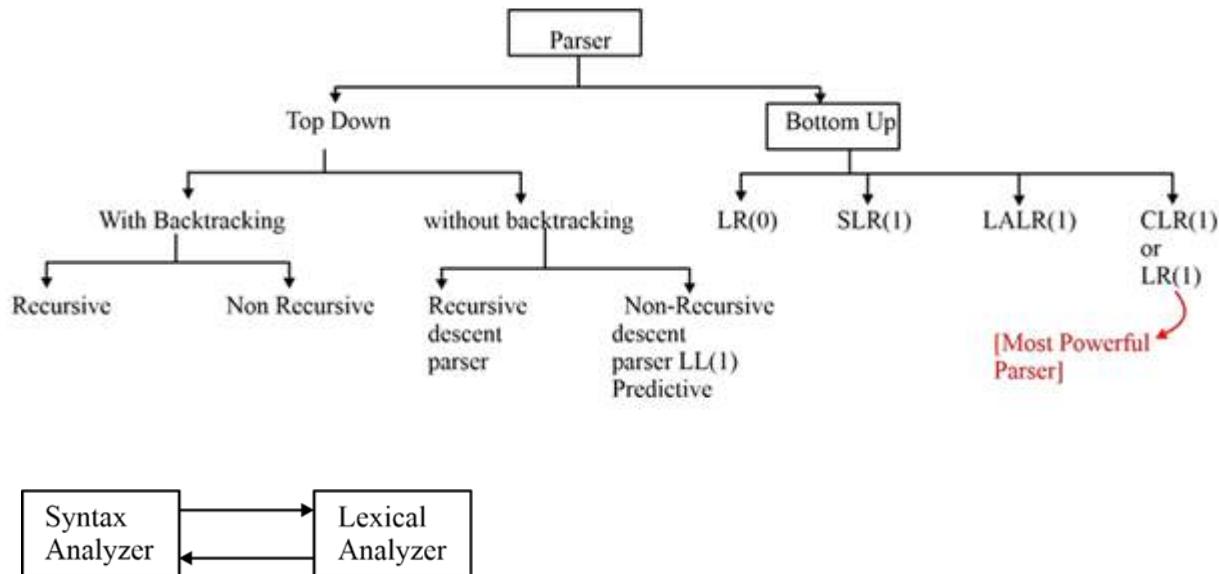
2.1.2 Types of Parsers in Bottom Up: [CD - Parser]



2.2 CD-Syntax Analysis / Parsing

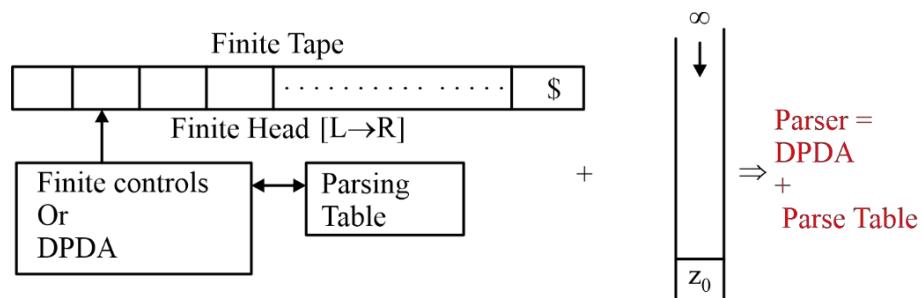
Grammatical Errors are checked by the help of parsers.

- Parsers are basically DPDA



- All of these parsers are table driven.

2.2.1 Mathematical Model of Parser



- Parsers generate Parse Tree, for a given string by the given grammar.

2.2.2 Top-Down Parser (LL (1))

- It uses LMD and is equivalent to DFS in graph.

2.2.3 Algorithm to construct Parsing Table

1. Remove Left Recursion if any.
 2. Left Factor [Remove Common Prefix]
 3. Find first and follow set
 4. Construct the table.
- If we increase the look ahead symbol:

2.2.4 Removal of common Prefix: (Left Factor)

$$1. \quad S \rightarrow a \mid a b \mid a A \quad S \rightarrow a Y$$

$$Y \rightarrow \in \mid b \mid A$$

$$2. \quad A \rightarrow a b A \mid a A \mid b$$

$$A \rightarrow a \times \mid b \quad A \rightarrow a \times \mid b$$

$$X \rightarrow b A \mid A \quad X \rightarrow b A \mid a x \mid b$$

$$A \rightarrow a X \mid b$$

$$X \rightarrow a X \mid b$$

$$Y \quad Y \rightarrow A \mid \in$$

2.2.5 First and Follow

- First Set → extreme Left terminal from which the string of that variable starts.
 - it never contains variable, but may contain ‘ \in ’.
 - we can always find the first of any variable.
- Follow set → Follow set contains terminals and $\$$.
 - It can never contain variable and ‘ \in ’.
 - How to find follow set?
 1. Include $\$$ in follow of start variable.
 2. If production is of type →
 $A \rightarrow \alpha B \beta \quad \alpha \beta \rightarrow \text{strings of grammar symbol}]$.
follow (B) = first (β).
If, $\beta \rightarrow \in$, i.e. $A \rightarrow \alpha B$, then follow(B) = follow(A).
- Productions like: $A \rightarrow \alpha A$ gives no follow set.

2.2.6 Example of first and follow set

1.	S	\rightarrow	AB		CD
	A	\rightarrow	aA		a
	B	\rightarrow	bB		b
	C	\rightarrow	cC		c
	D	\rightarrow	dD		d

	First	Follow
S	a, c	\$
A	a	b
B	b	\$
C	c	D
D	D	\$

2.2.7 Entity into Table: Top Down

1. No of Rows = number of unique variable in Grammar.
 2. No of columns = [Terminals + \$]
 3. For a Variable (Row) fill the column (Terminal) if its P, there in its first with the production required.
 4. If \in is in first put $V \rightarrow \in$ under \$ and its follow.
- If any cell has multiple items, then its not possible to have LL (1) Parser, since that will be ambiguous.
 - In top down we do: Derivation
In Bottom up we do: Production.

Question: Construct LL (1) Parsing Table for the given Grammar: $E \rightarrow E + T \mid T; T \rightarrow T * F \mid F; F \rightarrow (E) \mid id; \Rightarrow G_0$

\Rightarrow Removing Left Recursion:

$$\begin{aligned} E &\rightarrow TE' \\ E' &\rightarrow + TE' \mid \in \\ T &\rightarrow FT' \quad G_1 \\ T' &\rightarrow *FT' \mid \in \\ F &\rightarrow (E) \mid id \end{aligned}$$

	First	Follow
E	C, id	\$,)
E'	+, \in	\$,)
T	C, id	+, \$,)
T'	*, \in	+, \$,)
F	C, id	*, +, \$,)

- Left Factoring not Required:

Construction of Table: [LL (1)]

	+	*	()	id	\$
E	Error	Error	$E \rightarrow TE'$	Error	$E \rightarrow TE'$	Error
E'	$E' \rightarrow TE'$	Error	Error	$E' \rightarrow \in$	Error	$E' \rightarrow \in$
T	Error	Error	$T \rightarrow FT'$	Error	$T \rightarrow FT'$	error
T'	$T' \rightarrow \in$	$T' \rightarrow *FT'$	Error	$T' \rightarrow \in$	Error	$T' \rightarrow \in$
F	Error	Error	$F \rightarrow (\in)$	Error	$F \rightarrow id$	error

- Since for G_1 , Table constructed with no multiple entries. Hence successfully completed. Hence G_1 is LL (1)

Question: Construct LL (1) Parsing Table for the following Grammar:

$$S \rightarrow L = R \mid R; L \rightarrow * R \mid id; R \rightarrow L \Rightarrow G_0$$

Solution: Left Factoring:

$$\begin{array}{l}
 S \rightarrow L = R \mid L \quad S \rightarrow LX \\
 L \rightarrow *R \mid id \Rightarrow \times \Rightarrow R \in \Rightarrow G_1 \\
 R \rightarrow L \quad \quad \quad L \rightarrow *R \mid id \\
 \quad \quad \quad \quad \quad R \rightarrow L
 \end{array}$$

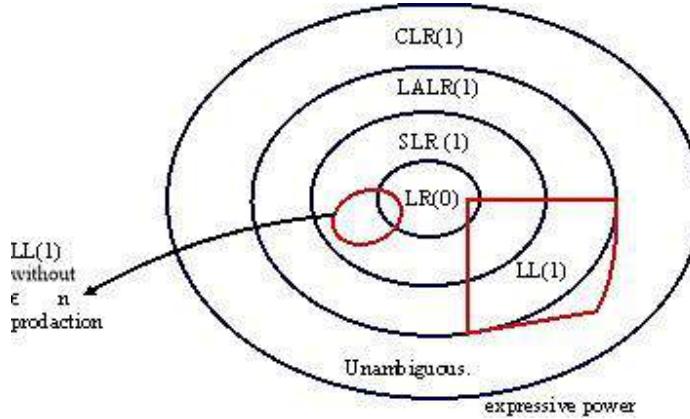
	First	Follow
S	*, id	\$
\times	=, \in	\$
L	*, id	\$
R	*, id	\$

Construction of Table:

	*	=	Id	\$
S	$S \rightarrow LX$	Error	$S \rightarrow L$ X	Error
L	$L \rightarrow *R$	Error	$L \rightarrow id$	Error
R	$R \rightarrow L$	Error	$R \rightarrow L$	Error
\times	Error	$X \rightarrow = R$	Error	$X \rightarrow \in$

- G_1 is a LL (1) Grammar.

2.2.8 Hierarchy of Parsers: [for \in - free Grammar]



- For \in - producing grammars every LL (1) may not be LALR (1).

Note:

We can't construct any parser for ambiguous grammar. Except: Operator precedence, parser possible for some ambiguous grammar.

Example:

$G: S \rightarrow a S a \mid b S b \mid a \mid b$ (Unambiguous but no parser) $L(G) = \omega(a + b) \omega R$
(Odd palindrome)

- Every RG is not LL (1) as it may be ambiguous, or Recursive or Common Prefix.
- Parsers exists only for the grammar if its Language is DCFL.
- There are some grammars whose Language is DCFL but no parser is possible for it.

2.4.9 Operator Precedence Grammar

Format: (1) No 2 or more variable side by side
 (2) No \in production.

Example:

$$\begin{array}{ll}
 E \rightarrow E + T \mid T & E \rightarrow E + E \\
 T \rightarrow T * F \mid F & \text{or} \quad E \rightarrow E * E \quad \text{or} \quad S \rightarrow aSa \mid bSb \mid a \mid b \\
 F \rightarrow (E) \mid id & E \rightarrow a / b \quad \text{or} \quad O.G \\
 O.G. & O.G \\
 & \text{Or} \\
 S \rightarrow AB & \\
 A \rightarrow aA \mid \epsilon & \boxed{\text{Not O.G}} \\
 B \rightarrow bB \mid \epsilon &
 \end{array}$$

2.2.10 Checking LL (1) without Table

$A \rightarrow \alpha_1 | \alpha_2 | \alpha_3$ then \rightarrow

$$\begin{array}{ll}
 \text{first } (\alpha_1) \cap \text{first } (\alpha_2) = \emptyset & \text{first } (\alpha_1) \cap \text{first } (\alpha_2) = \emptyset \\
 \text{first } (\alpha_1) \cap \text{first } (\alpha_3) = \emptyset & \text{first } (\alpha_1) \cap \text{first } (\alpha_3) = \emptyset \\
 \text{first } (\alpha_2) \cap \text{first } (\alpha_3) = \emptyset & \text{first } (\alpha_2) \cap \text{first } (\alpha_3) = \emptyset \\
 & \text{follow } (A) \cap \text{first } (\alpha_1) = \emptyset \\
 & \text{follow } (A) \cap \text{first } (\alpha_2) = \emptyset \\
 & \text{follow } (A) \cap \text{first } (\alpha_3) = \emptyset
 \end{array}$$

2.2.11 Bottom-UP Parser

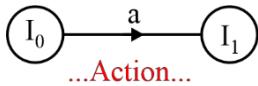
- It uses RMD in reverse and has no problem with:
 - (a) Left recursion
 - (b) Common Prefix
- SLR (1) LR (0) CLR (1) LR (1)
- LR (0) items LALR (1) items
- $LR(1) = LR(0) + 1$ Lookahead
- No parser possible for ambiguous grammar.
- There are some unambiguous grammars for which, there are no Parser.

2.2.12 Basic Algorithm for Construction

- Augment the grammar and expand it, and give numbers to it
- Construct LR (0) or LR (1) item set.
- From that fill the entries in the Table Accordingly.

2.2.13 Types of Entries

- (1) **Shift Entries:** Transitions or Terminals



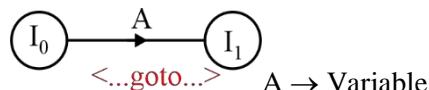
Entry:

	a
I ₀	S ₁

a ∈ Terminal, I₀, I₁: Item sets.

- Shift entries are some for all Bottom – up Parser.

- (2) **State Entry:** Transition or non – terminal (variable)



Entry:

	A
I ₀	I

- Same for all Bottom-up Parser.

- (3) **Reduce Entity:** done for each separate production in the item set of type:

i > x → α

where, i → Production Number

x → Producing Variable

α → Grammar String

- (a) LR (0) Parser:

Put R_i in every cell of the set-in action table (**ALL**).

- (b) SLR (1) Parser:

Put R_i only in the follow(x) form the Grammar. (**Follow(x)**)

- (c) LALR (1) and CLR (1):

Put R_i only in the look ahead of the production (**Lookaheads**)

2.2.14 Conflicts

LR (0) Parser:

SR: Shift Reduce Conflict

$$S \rightarrow X \rightarrow \alpha \cdot t\beta \Rightarrow \text{then SR}$$

$$R \rightarrow Y \rightarrow \delta \cdot \quad \text{conflict}$$

RR: Reduce Reduce conflict

$$R \rightarrow X \rightarrow \alpha \cdot \Rightarrow \text{then RR}$$

$$R \rightarrow X \rightarrow \beta \cdot$$

SLR (1) Parser:

SR:

$$\begin{array}{l} S \rightarrow X \rightarrow \alpha \cdot t\beta \\ R \rightarrow Y \rightarrow \delta \cdot \end{array} \left. \begin{array}{l} \$ \\ \text{conflict} \end{array} \right\} \quad t \in \text{follow}(Y)$$

RR:

$$\begin{array}{l} X \rightarrow \alpha \cdot \\ Y \rightarrow \beta \cdot \end{array} \left. \begin{array}{l} \$ \\ \Rightarrow \text{then RR} \end{array} \right\}$$

Follow(x) \cap follow(y) $\neq \emptyset$

LALR (1) and CLR (1): Same as SLR (1), but instead

SR:

$$\begin{array}{l} X \rightarrow \alpha \cdot t \beta \cdot L_1 \\ Y \rightarrow \delta \cdot L_2 \\ T \in L_2 \end{array} \left. \begin{array}{l} \$ \\ \text{conflict} \end{array} \right\}$$

RR:

$$\begin{array}{l} X \rightarrow \alpha \cdot, L_1 \\ Y \rightarrow \beta \cdot, L_2 \end{array} \left. \begin{array}{l} \$ \\ \text{conflict} \end{array} \right\} \quad L_1 \cap L_2 \neq \emptyset$$

2.4.15 Inadequate Static

A static having ANY conflict is called a conflicting state or independent state.

Note:

The state $S' \rightarrow S$. or $S' \rightarrow S, \$$ is accepted state, and this is not a reduction.

- The only difference between CLR (1) and LALR (1) is that, the states with the similar items, but different Lookaheads are merged together to Reduce space.

Important Points

- If CLR (1) doesn't have any conflict, then conflict may or may not arise after merging in LALR (1).
- If LALR (1) has SR – conflict, then we can conclude that CLR (1) also has SR – Conflict.
- LALR (1) has SR – conflict if and only if CLR (1) also has SR.

- We can construct parser for every unambiguous regular grammar: [CLR (1) Parser].

				L Left to right scan of i/p	R Using reverse RMD	(0) No Lookhead
S	L	R	(1)			
Simple	L to R Scan	Using Reverse RMD	Lookahead			
L	A	L	R	(1)		
Look	Ahead	L to R Scan	Revers RMD		Lookahead	
C	L	R	(1)			
Canonical	L to R Scan	reverse RMD	Lookahead			

Very Important Point:

LALR (1) Parser can Parse non LALR (1) grammar, when only non-SR – Conflict by favouring shift over reduce.

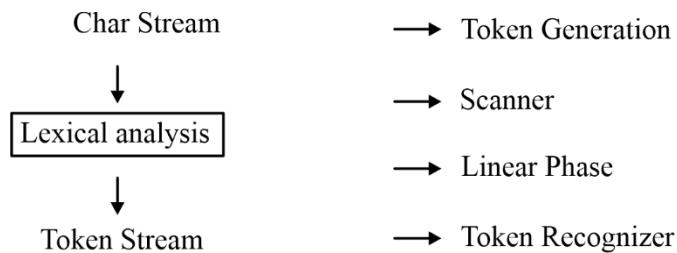
Example: $E \rightarrow E + E \mid E * E \mid id \mid 2 + 3 * 5 \Rightarrow E + E \cdot * 5$



3

LEXICAL ANALYSIS

3.1 Lexical Analysis



3.1.1 Definition

Scan the whole program, char by char and produces the corresponding token.

- Also produces /Lexical Errors (if any).
- Functions of Lexical Analyzer.
 - (1) Scans all the character of the program.
 - (2) Token recognizer.
 - (3) Ignores the comment & spaces.
 - (4) Maximal Munch Rule [Longest Prefix Match].

Note:

The Lexical Analyser uses, the Regular Expression.

Prioritization of Rules.

Longest Prefix match .

- Lexeme → smallest unit of program or Logic.
- Token → internal representation of Lexeme.

3.1.2 Types of Token

- (1) Identifier
- (2) Keywords
- (3) Operators
- (4) Literals/Constants
- (5) Special Symbol

3.1.3 Token Separation

- (1) Spaces
- (2) Punctuation

3.1.4 Implementation

- LEX tool \Rightarrow Lex.yy.e
- All identifiers will have entry in symbol Table/ LA, gives entries into the symbol Table.
Regular Expression \rightarrow DFA \rightarrow Lexical Analyzer

3.1.5 Find number of Tokens

- (1) void main () {
 Printf("gate");
}
 {11 Tokens}
- (2) int x, *P;
 x = 10; p = &x; x++;
[18 Tokens]
- (3) int x;
 x = y;
 x == y;
[11 Tokens]
- (4) int 1x23; [Lexical Error]
- (5) char ch = 'A';
[5 Token]
- (6) char ch = 'A;
Lexical Error.
- (7) char *p = "gate";
[6 Tokens]
- (8) char * p = "gate;
Error.
- (9) int x = 10;
/* comment x = x + 1; ERROR
x = x + 1; [11 Tokens]



4

SYNTAX DIRECTED TRANSACTION

4.1 Syntax Directed Transaction

CFG + Transition }
Syntax + Transition

SDT

SDT: CFG + Transition → 1) Meaning 2) Semantic

4.1.1 Application of SDTL

- (1) Used to perform Semantic Analysis.
- (2) Produce Parse Tree.
- (3) Produce intermediate representation.
- (4) Evaluate an expression.
- (5) Convert infix to prefix or postfix.

Example: $S \rightarrow S_1S_2$ [S.count = $S_1.\text{count} + S_2.\text{count}$]

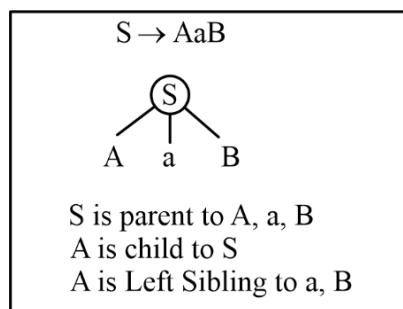
$S \rightarrow (S_1)$ [S.count = $S_1.\text{count} + 1$]

$S \rightarrow \epsilon$ [S.count = 0]

- Count is an attribute for non-terminal

4.1.2 Attributes

- (1) Inherited Attribute
- (2) Synthesized Attribute



- (1) Inherited Attribute: (RHS)

$S \rightarrow AaB$ {A.x = f(B.x | S.x)}

- The computation at any node (non-terminal) depends on parent or siblings (S).

- In above Example, x is inherited attribute.

(2) Synthesized Attribute: (LHS)

$$S \rightarrow A \ B \ \{S.x = f(A.x \mid B.x)\}$$

x is synthesized attribute.

The computation at any node (non-terminal) depends on children.

4.1.3 Identifying Attribute Type

- Always check every Translation.

- | | |
|---|--|
| 1) $D \rightarrow T : L ; \{L.Type = T.Type\}$ inherited
$L \rightarrow L1, id ; \{L1.Type = L.Type\}$ inherited
$L \rightarrow id$
$T \rightarrow integer \{T.Type = int\}$ synthesized | Type is neither inherited nor synthesized |
| 2) $E \rightarrow E1 + T \{E.val = E1.val + T.val\}$ synthesized
$E \rightarrow T \{E.val = T.val\}$ Synthesized
$T \rightarrow T1 - F \{T.val = T1.val - F.val\}$ Synthesized
$F \rightarrow id \{F.val = id.val\}$ synthesized | Val is synthesized attribute |
| 3) $S \rightarrow AB \{A.a = B.x; S.y = A.x\}$ x is inherited y is synthesized
$A \rightarrow a \{A.y = a\}$ y is synthesized
$B \rightarrow b \{B.y = a.y\}$ y is synthesized | x \Rightarrow inherited
y \Rightarrow synthesized |
| 4) $D \rightarrow T \ L \ \{L.in = T.type\}$ inherited(in)
$T \rightarrow int \ \{T.type = int\}$ /synthesized
$L \rightarrow id \ \{\text{Add type}(id.entry, L.in)\}$ | in \Rightarrow inherited
type \Rightarrow synthesized |

4.1.4 Syntax Directed Definitions (SDDs): (Attribute Grammar)

(1) L-Attributed Grammar

- Attribute is synthesized or restricted inherited. (1) Parent 2) Left sibling only).
- Translation can be appended anywhere is RHS of production.
- Example: $S \rightarrow AB \{A.x = S.x + 2\}$
 or, $S \rightarrow AB \{B.x = f(A.x \mid S.x)\}$
 or, $S \rightarrow AB \{S.x = f(A.x \mid B.x)\}$
- Evaluation: In Order (Topological).

(2) S-Attributed Grammar:

- Attribute is synthesized only.
- The transaction is placed only at the end of production.
- Example: $S \rightarrow AB \{S.x = f(Ax \mid Bx)\}$.
- Evaluation: Reverse RMD (Bottom-Up Parsing).

4.1.5 Identify SDD

(1) $E \rightarrow E_1 + E_2 \{E.type = \text{if } (E_1.type == \text{int} \& \& E.type == \text{int}) \text{ then int}\}$ Synthesizer else type – error.
 $E \rightarrow \text{id } \{E.type = \text{Lookup (id.entry)}\}$ synthesizer.

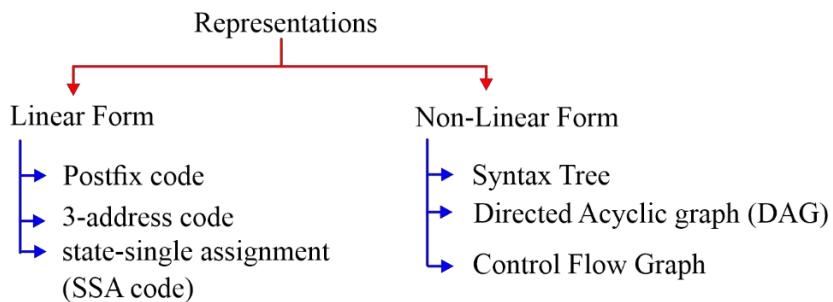
- type is synthesized, hence S-attributed and also L-attributed Grammar.
- Every S-attributed Grammar is also L-attributed Grammar.
- For L-attributed Evaluation, use the In-order of annotated Parser Tree.
- For S-attributed, reverse of RMD is used.
 - find RMD Order.
 - Consider its reverse.



5

INTERMEDIATE, CODE OPTIMIZATION

5.1 Introduction



Example Expression:

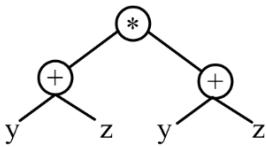
$$(y + z) * (y + z)$$

Post fix → yz + yz + *

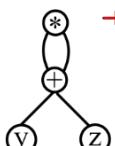
SSA → $t_1 = y + z$
 $t_2 = t_1 * t_2$ ⇒ f_1 and f_2 cannot be reassigned

3AC → $t_1 = y + z$
 $t_2 = t_1 \times t_2$

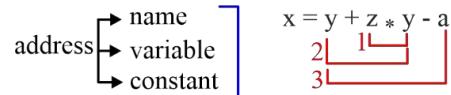
Syntax Tree →



DAG → → Reuse the already existing common sub expression

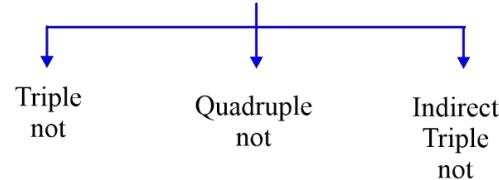


3-address Code: Code in which, at most 3 addresses. [including LHS]



→ [$t_1 = z * y$
 $t_2 = y + t_1$
 $t_3 = t_2 - a$] Equivalent
 3 - address
 Code
 $x = t_3$

Representation of 3AC



0	*	z	y
1	+	y	(0)
2	-	(1)	a
3	=	(2)	

0	*	z	y	t_1
1	+	y	t_1	t_2
2	-	t_2	a	t_3
3	=	t_3		x

6	(0)
7	(1)
8	(2)
9	(3)

→ Indirect notation
 pointers to the
 numbers of Triple

- Triple Notation Quadruple
 → Space efficient → space inefficient
 → time inefficient → time efficient

- 3AC done using operator precedence.

Find minimum number of variables required in equivalent 3AC:

1. $x = u - t$ $x * y \Rightarrow x * (t - z) \Rightarrow (y + w) * (t - z)$

$\Rightarrow ((x * v) + w) * (t - z)$

$\Rightarrow (((u - t) * v) + w) * (t - z)$

$y = t - z$

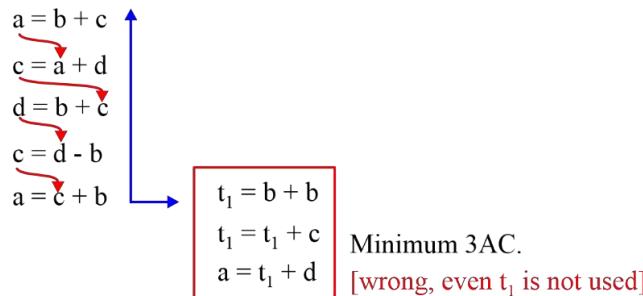
$y = *x * y$

7 variables

$$\begin{bmatrix} u = u - t & u = t - z \\ v = u * u & z = w * v \\ w = v + w \end{bmatrix} \Rightarrow 5 \text{ variables only}$$

⇒ Evaluating the expression:

$$\begin{aligned} a &\Rightarrow c + b \Rightarrow d - b + b \Rightarrow b + c - b + b \\ &\Rightarrow b + a + d - b + b \\ &\Rightarrow b + b + c + d - b + b \\ &\Rightarrow b + b + c + d \end{aligned}$$



∴ Minimum:

$$\begin{cases} b = b + b \\ c = b + c \\ a = c + d \end{cases} \Rightarrow \text{only 3 variables [most optimal]}$$

5.1.1 Static Single Assignment Code: (SSA Code)

Every variable (address) in the code has single assignment [single meaning] + 3AC.

1.

$$\begin{array}{l} x = u - t \\ y = x * u \\ x = y + w \\ y = t - z \\ y = x * y \end{array}$$

Find SSA?

⇒ [u, t, v, w, z] are already assigned so we can't use them.

Equivalent SSA Code:

$$x = u - t \quad y = x * v$$

$$p = y + w$$

$$q = t - x$$

$$r = p * q$$

in use: x, y, p, q, r ⇒ additional

∴ Total Variable ⇒ 10.

2.

$$\begin{array}{l} p = a - b \\ q = p * c \\ p = u * v \\ q = p + q \end{array}$$

⇒ [a, b, c, u, v] are already assigned,

Equivalent SSA Code:

$$p = a - b$$

$$q = p * c$$

$$p_1 = v * u$$

$$q_2 = p_1 + q$$

in use: p, q, p₁, q₂

∴ Total Variable = 9

5.1.2 Control Flow Graphs

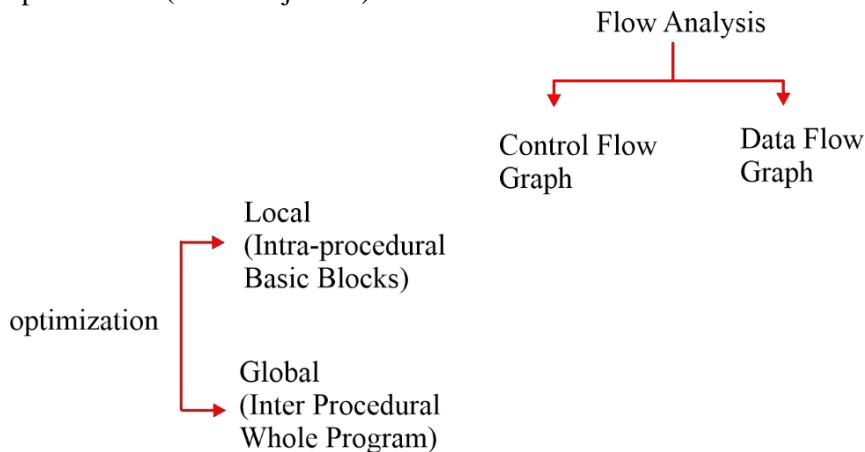
- CFG contain group of basic blocks and controls. CFG has nodes and edges to define basic blocks and controls.
- Basic Blocks: Sequence of 3-address code statements, in which control enters only from 1st statement (called as leader), and leaves from last statement.

Example:

```
1. i = 1          } LB1
2. j = 1          } LB2
3. t1 = 5 * i    }
4. t2 = t1 + 5   }
5. t3 = 4*t2    }
6. t4 = t3      }
7. a[t4] = 1    }
8. j = j + 1    }
9. if j ≤ 5 goto 3 } LB4
10. i = i + 1   }
11. if i < 5 goto 2 } LB6
```

5.2 Code Optimization

Saves space / time. (Basic Objective)



5.2.1 Optimization Methods

1. Constant folding
2. Copy propagation
3. Strength Reduction
4. Dead code elimination
5. Common sub expression elimination.
6. Loop Optimization
7. Peephole Optimization

1. Constant Folding

(i) $x = 2 * 3 + y \Rightarrow x = 6 + y$

Folding

ii) $x = 2 + y * 3 \}$ can't fold the constant

2. Copy Propagation

i) Variable Propagation:

$x = y;$

$z = y + 2;$

$\Rightarrow z = x + 2;$

ii) Constant Propagation:

$x = 3$

$z = 3 + a;$

$\Rightarrow z = x + a$

3. Strength Reduction:

Replace expensive statement / instruction with cheaper one.

(i) $x = 2 * y$ **costly** $\Rightarrow x = y + y$; **Cheap**

(ii) $x = 2 * y$ $\Rightarrow x = y << 1$; **Much Cheaper**

(iii) $x = y / 8$ $\Rightarrow x = y >> 3;$

4. Dead Code Elimination:

$x = 2;$ **FALSE**

if ($x > 2$)

printf("code"); **DEAD CODE CAN BE REMOVED**

else

printf("optimization");

$x = 2;$
printf("optimization");

- Hence, above dead code never executes during execution. We can always delete such code.

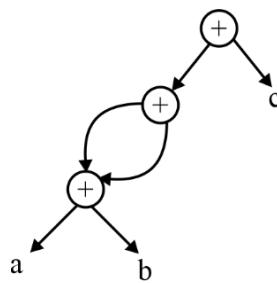
5. Common Sub Expression Elimination:

DAG is used to eliminate common sub expression.

Example: $x = (a + b) + (a + b) + c;$

$\Rightarrow t_1 = a + b$

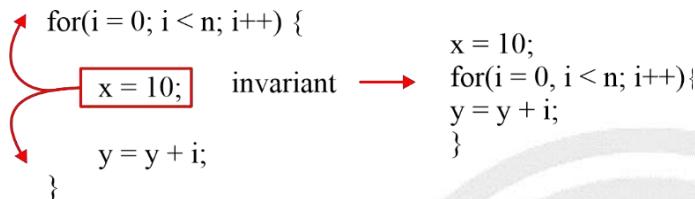
$x = t_1 + t_1 + c$



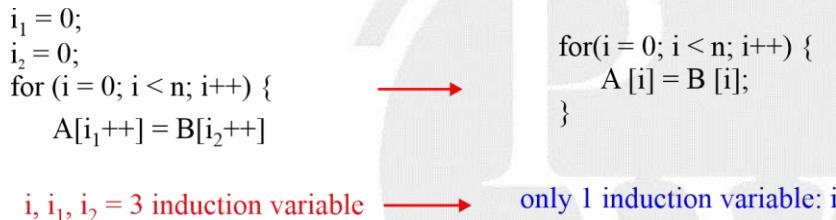
6. Loop Optimization:

(i) Code Motion – Frequency Reduction:

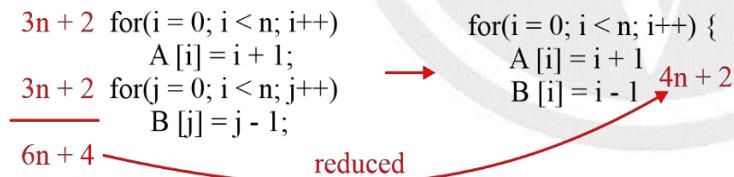
Move the loop invariant code outside of loop.



(ii) Induction Variable elimination:



(iii) Loop Merging / Combining: (Loop Jamming)



(iv) Loop Unrolling:

1) for ($i = 0; i < 3; i++$)
 printf("CD");
 printf("CD");
 printf("CD");

$3 \times 3 + 2 = 11$ Statements → 3 statements

2) for ($i = 0; i < 2n; i ++$) {
 printf("CD");
 }
 for ($i = 0; i < n; i++$) {
 printf("CD");
 printf("CD");
 }
 $(2 \times 3n + 2) = 6n + 2$ → $(4n + 2)$

7. Peephole Optimization:

Examines a short sequence of target instructions in a window (*peephole*) and replaces the instructions by a faster and/or shorter sequence when possible.

- Applied to intermediate code or target code
- Following Optimizations can be used:

- Redundant instruction elimination
- Flow-of-control optimizations
- Algebraic simplifications
- Use of machine idioms



GATE Exam 2025?



SHRESHTH BATCH

ME | CE | EE | EC | CS & IT

- Live Classes & Recorded Videos
- DPPs & Solutions
- Doubt Solving
- Batches are available in *English*

Weekday & Weekend
Batches Available

JOIN NOW!



Operating System



Operating System

INDEX

1. Introduction and Background of OS **9.1 – 9.3**
2. Process Concepts **9.4 – 9.6**
3. CPU Scheduling **9.7 – 9.10**
4. Multithreading **9.11 – 9.12**
5. Synchronization **9.13 – 9.26**
6. Deadlock **9.27 – 9.29**
7. Memory Management **9.30 – 9.35**
8. Virtual Memory **9.36 – 9.38**
9. File Systems **9.39 – 9.42**
10. Disk Scheduling **9.43 – 9.44**

1

INTRODUCTION AND BACKGROUND OF OS

1.1 What is OS

Operating system is an interface between the user applications and the computer hardware to develop and execute programs.

1.2 Goals of OS

- The primary goal of the operating system is to provide an easy-to-use environment to the user.
- The secondary goal of the operating system is the efficient use of the computer resources. (Operating system is also called as the resource allocator)
- Modularity
- Abstraction
- Ease of debugging

1.3 Functions of OS

- Process Management
- Memory management
- Resource Allocation
- File systems management
- Protection and Security

1.4 Types of OS

1.4.1. Batch Operating System

- Jobs with similar requirements are grouped into batches by the operating system.
- The idea is to execute the jobs onto the CPU one at a time. Another job cannot occupy the CPU time until the previous job has completed execution.
- Increased CPU idleness.

- Decreased throughput of the system.

1.4.2 Multi-programmed/multi-tasking operating system

- Multiple programs/jobs reside in the main memory for execution. The operating system selects and executes one of these jobs on to the CPU.
- If the job in execution requires an I/O operation, another job which is ready for execution is scheduled on the CPU.
- Increased CPU utilization.
- Increased throughput of the system.
- Multi-tasking is a logical extension of multi-programming systems. The jobs are executed on the CPU in time sharing mode.
- The main advantage of multi-tasking systems is good response time.

1.4.3 Real time operating system

- It has well-defined and fixed time constraints.
- Processing of the programs must be done in the defined constraints or the system will fail.
- They are categorized into-hard and soft real time systems.

1.4.4 Distributed operating system

- A distributed operating system handles jobs that are executed by multiple processors networked to each other.
- They are also termed as loosely coupled systems.
- The advantages of distributed systems include resource sharing, computation speed up and reliability.

1.5 Dual Mode Operations

A processor supports two modes of instruction execution:

- User mode/ non-privileged mode
- Kernel mode/ Privileged mode/ Monitor mode

The primary goal of the dual mode operation is to provide protection and security to the user application programs and the operating system from the unauthorized users. The mode bit is used to determine the particular mode in which an instruction is executing.

Mode bit: 0	Kernel mode
Mode bit: 1	User mode

- A process running in kernel mode has direct access to the hardware and full access to the machine instruction set. The operating system always runs in kernel mode.
- Other examples of privileged instructions include I/O operations, context-switching, clearing the memory map etc.

- A process running in user mode has no access to the hardware and limited access to the machine instruction set.
- Other examples of non-privileged instructions include reading the time of the clock, reading the status of the processor, generate trap etc.

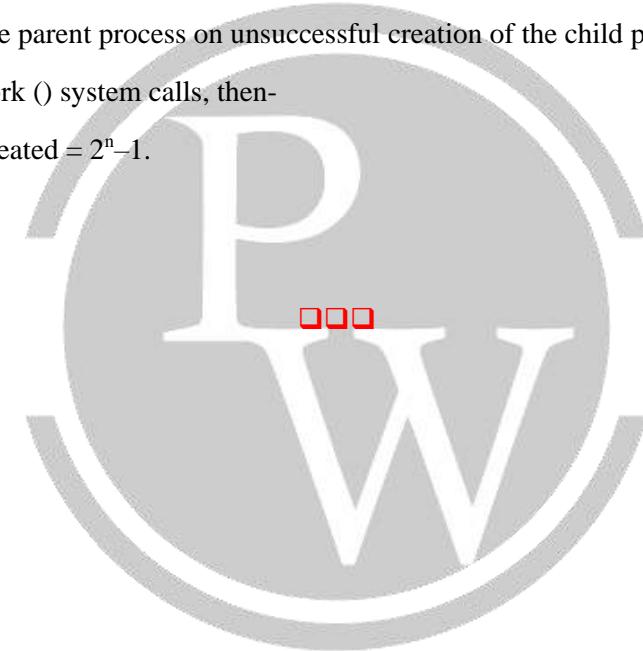
1.6 System Call

- System calls provide the services of the operating system to the user application programs.
- They act as entry points into the kernel system.

1.6.1 Fork () System Call

- The fork () system call is used to create the child processes.
- It returns 0 to the newly created child process.
- It returns the process id of the child process to the parent process on successful creation of the child process.
- It returns negative value to the parent process on unsuccessful creation of the child process.
- If the program contains ‘n’ fork () system calls, then-

Number of child processes created = $2^n - 1$.



2

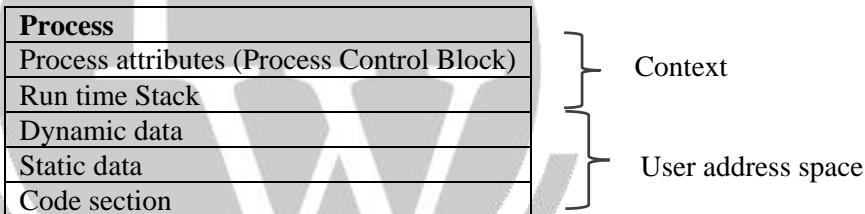
PROCESS CONCEPTS

2.1 Program Versus Process

Program	Process
Program is a set of instructions and data.	A program under execution is called a process.
It resides in the secondary memory.	It resides in the main memory.
It is passive.	It is active and dynamic.
It is not allocated any resources.	The operating system allocates resources to the process for its execution.

2.2 Process as ADT

The process image can be described as:



Any process has the following attributes:

- Process identification information
- Priority
- Process state information
- Program counter
- Memory limits
- List of files
- List of open devices
- Protection information

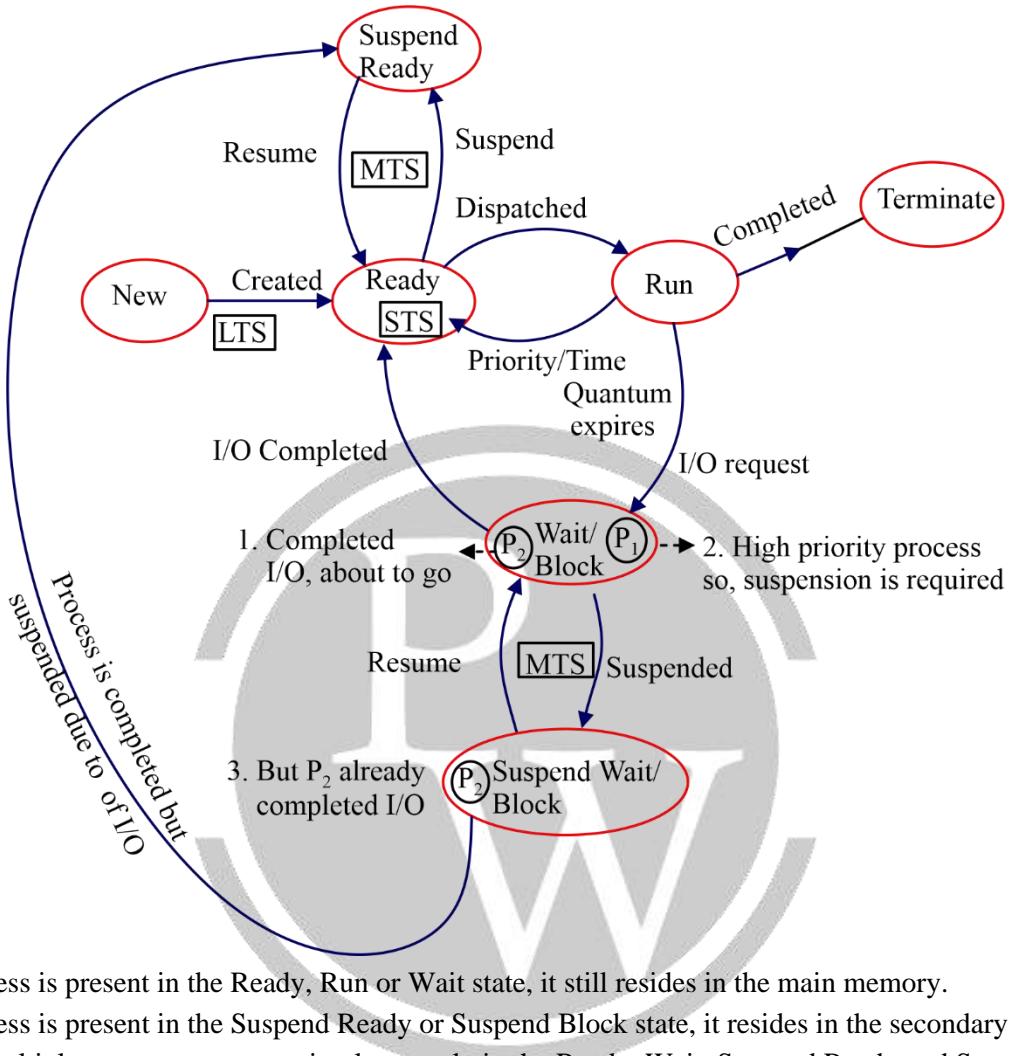
All the attributes of the process are stored in the Process Control Block (PCB). The features of Process Control Block are as follows:

- Every process has its own process control block.
- The process control blocks of all the processes are stored in the main memory.
- They are implemented using **double linked list** data structure.

The **context** of a process includes the process attributes and the stack information.

2.3 Process State Transition Diagram

A process passes through multiple states in its lifetime as follows:



- When the process is present in the Ready, Run or Wait state, it still resides in the main memory.
- When the process is present in the Suspend Ready or Suspend Block state, it resides in the secondary memory.
- There can be multiple processes present simultaneously in the Ready, Wait, Suspend Ready and Suspend Wait states.
- In the Run state, only one process can exist at a time.

2.4 Schedulers

The operating system deploys three kinds of schedulers:

- Long-term scheduler.
- Medium-term scheduler.
- Short-term scheduler.

2.4.1 Long-term scheduler

- It is responsible for the creation and bringing of new processes into the main memory.
- It controls the degree of multi-programming.
- It should generate a good mix of CPU and I/O bound process for efficient resource utilization.
- It is involved in the New → Ready state transition.

2.4.2 Medium-term scheduler

- The medium-term scheduler acts as the swapper by doing swap-out (suspending the process from the main to secondary memory) and swap-in (resuming the process by bringing it from the secondary to main memory) operations.
- It is involved in Ready \Leftrightarrow Suspend Ready and Block \Leftrightarrow Suspend Block state transitions.

2.4.3 Short-term scheduler

- The short-term scheduler selects a process from the ready state to be executed.
- It is involved in the Ready \rightarrow Run state transition.

2.5 Dispatcher

- The dispatcher is responsible for loading the job (selected by the short-term scheduler) onto the CPU. It performs context switching. Context switching refers to saving the context of the process which was being executed by the CPU and loading the context of the new process that is being scheduled to be executed by the CPU.



3

CPU SCHEDULING

3.1 Scheduling Criteria

CPU scheduling will occur when:

- A new process arrives into the Ready state.
- A process undergoes Wait → Ready state transition.
- A process undergoes Run → Wait state transition for an I/O request.
- A process undergoes Run → Ready state transition every q second where q is the time slice.
- The priority of a ready process is higher than the priority of a running process.

3.2 Goals of CPU Scheduling

- Maximize CPU utilization.
- Minimize the response time and waiting time of the processes.

3.3 Process times

3.3.1 Arrival Time (AT)

The time at which the process arrives into the Ready state is called arrival time of the process.

3.3.2 Burst Time (BT)

The time required by the process for its complete execution is called burst time/service time of the process.

3.3.3 Completion Time (CT)

The time by which a process completes its execution post arrival is called completion time of the process.

3.3.4 Turnaround Time (TAT)

The time difference between completion and arrival times of a process is called as the turnaround time of the process.

$$TAT = CT - AT$$

3.3.5 Waiting Time (WT)

The time spent waiting for the CPU to complete the execution is called as waiting time of the process.

$$WT = TAT - BT$$

3.3.6 Response Time (RT)

The time difference between the first response and arrival times of a process is called the response time of the process.

3.4 Gantt Chart

Gantt chart shows the execution time period of the processes. For example:



- Process P₁ started execution at time t = 0 and finished just before time t = 10.
- The CPU was idle from t = 10 to t = 12.
- Process P₂ started execution at time t = 12 and finished at time t = 17.
- Process P₃ started execution at time t = 17 and finished at time t = 23.

3.5 Scheduling Algorithms

3.5.1 FCFS

- It is a non pre-emptive algorithm.
- Processes are assigned to the CPU based on their arrival times. If two or more processes have the same arrival times, then the processes are assigned based on their process ids.
- It is free from starvation.
- It suffers from *Convoys effect*.

3.5.2 Shortest Job First (SJF)

- It is a non-pre-emptive algorithm.
- Processes are assigned to the CPU based on the shortest burst time. If two or more processes have the same burst times, then the processes are assigned to the CPU based on their arrival times.
- If all the processes have the same burst times, SJF behaves as FCFS.
- In SJF, the burst times of all the processes should be known prior execution.
- It minimizes the average response time of the processes.
- There is a chance of starvation.

3.5.3 Shortest Remaining Time First (SRTF)

- It is a pre-emptive algorithm.
- Processes are assigned to the CPU based on their shortest remaining burst times.
- If all the processes have the same arrival times, SRTF behaves as SJF.
- It minimizes the average turnaround time of the processes.
- If shorter processes keep on arriving, then the longer processes may starve.

3.5.4 Longest Remaining Time First (LRTF)

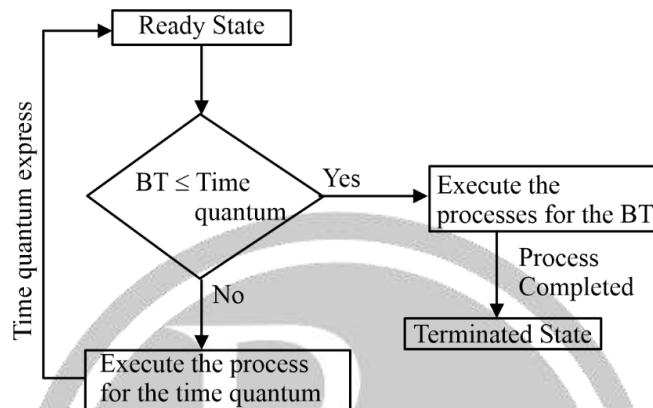
- It is a pre-emptive algorithm.
- Processes are assigned to the CPU based on their longest remaining burst times.
- It minimizes the average response time of the processes.
- If favours CPU bound processes.
- It is free from starvation.

3.5.5 Priority based Scheduling

- Priority scheduling can either be pre-emptive or non-pre-emptive.
- In pre-emptive priority scheduling, a process is voluntarily pre-empted whenever a higher priority process arrives.
- In non-pre-emptive priority scheduling, the scheduler picks up the highest priority process.
- If all the processes have equal priority, then priority scheduling behaves as FCFS.

3.5.6 Round Robin Scheduling

- RR scheduling is a pre-emptive FCFS based on the concept of time quantum or time slice.



- If the time quantum is too small, then the number of context switches (overhead) will increase and the average response time will decrease.
- If the time quantum is too large, then the number of context switches (overhead) will decrease and the average response time will increase.
- If the time quantum is greater than the burst times of all the processes, RR scheduling behaves as FCFS.

3.5.7 Highest Response Ratio Scheduling

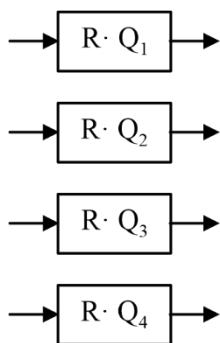
- It is a non pre-emptive algorithm.
- Processes are assigned to the CPU based on their highest response ratio.
- Response ratio is calculated as-

$$\text{Response Ratio} = \frac{WT + BT}{BT}$$

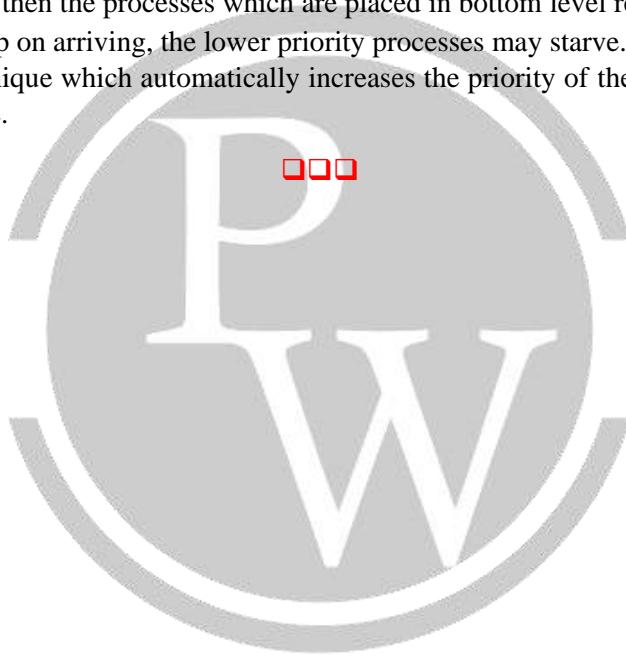
- It is free from starvation.
- It favours the shorter jobs and limits the waiting time of the longer jobs.

3.5.8 Multilevel Queue Scheduling

Multi-Level Queue Scheduling



- Depending on priority of the process, in which particular ready queue, the process has to be placed will be decided.
- High priority processes will be placed in top-level ready queue and low priority process will be placed in bottom level ready queue.
- Only after completion of all the processes, from top level ready queue the further level ready queue processes will be scheduled.
- If this is the strategy followed, then the processes which are placed in bottom level ready queue suffer from starvation. If higher priority processes keep on arriving, the lower priority processes may starve. This problem can be solved with the help of aging. Aging is a technique which automatically increases the priority of the processes that have been waiting in the system for a very long time.



4

MULTITHREADING

4.1 Threads

- A thread is a light-weight process.
- In multithreading, the threads of the same process share user address space, files, signal and signal handlers etc.
- In multithreading, each thread has its own stack, thread id, CPU state information (control and status registers, stack pointers) and scheduling information (thread state, priority etc.).

4.2 Benefits of Threads

- Improved response time.
- Faster context switches.
- Effective utilization of multiprocessor systems
- Enhanced throughput of the system.
- Economical.
- Effective resource sharing and utilization.

4.3 Types of Threads

4.3.1 Based on the number of threads

There are two types of threads:

- (i) Single thread process
- (ii) Multi thread process

4.3.2 Based on level

There are two types of threads:

- (i) User-level threads
- (ii) Kernel-level threads

4.3.3 User level threads versus kernel level threads

User level threads	Kernel level threads
These threads are managed at user level.	These threads are managed at kernel level.
These threads are not recognized by the kernel.	These threads are recognized by the kernel.
They are implemented as dependent threads.	They are implemented as independent threads.
All user-level threads of a process can run on one processor only and only one thread runs at a time.	The kernel-level threads of a process can run on different processors concurrently in a multi-processor environment.
Blocking one user-level thread of a process blocks the entire process.	Blocking one kernel-level thread of a process does not affect the other threads of the process.
These threads have less context.	These threads have more context.
Scheduling of user-level threads is done by the thread libraries.	Scheduling of kernel-level threads is done by the operating system.
No hardware support is required.	Hardware support is required.
Implementation is easy and simple.	Implementation is complicated and difficult.

4.4 Multithreading Models

4.4.1 Many-to-One

- Many user level threads are mapped to single kernel-level thread.

4.4.2 One-to-One

- Each user level thread is mapped to single kernel-level thread.

4.4.3 Many-to-Many

- Many user level threads are mapped to multiple kernel-level threads.
- It allows the OS to create a sufficient number of kernel threads

4.5 Thread Libraries

- Thread libraries provide programmer with API for creating and managing threads.
- There are two primary ways of implementing thread libraries:
 - Library entirely in user space
 - Kernel-level library supported by the OS
- Examples include pthread libraries, java threads, green threads etc.

4.6 Disadvantages of Multithreading

- Blocking: Blocking one user-level thread of a process blocks the entire process. If the kernel thread is single threaded, then blocking the kernel thread will block the whole process. Consequently, CPU may remain idle during this period.
- Security: Since, there is an extensive sharing among threads, there is a potential problem of security.
- Maintaining Thread Control Block (TCB) is considered as overhead for the system.



5

SYNCHRONIZATION

5.1 Synchronization

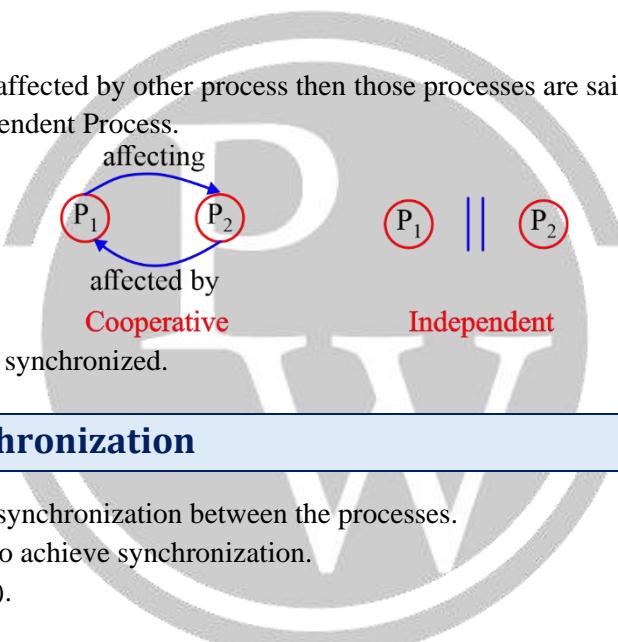
Processes are categorized into two types

Process



Execution of one process affects or affected by other process then those processes are said to be cooperative process.

Otherwise, they are said to be Independent Process.



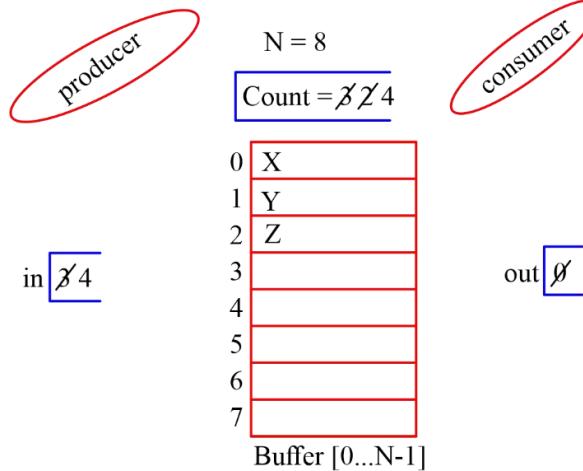
Processes must be cooperative to be synchronized.

5.2 Understanding Synchronization

- (1) The problems arise not having synchronization between the processes.
- (2) The conditions to be followed to achieve synchronization.
- (3) The solutions (wrong and right).

5.3 Problems arises for not having synchronization between the processes

5.3.1 Producer Consumer



```
int count = 0; // Initially buffer will be empty and only producer will be allowed to execute first.
```

```
void producer (void)
```

```
{
```

```
    int itemp;
```

```
    while(true)
```

```
{
```

```
        produce_item(itemp);
```

```
        while (count == N); // Buffer Full
```

```
        Buffer [in] = itemp;
```

```
        in = (in + 1)modN; // ← to repeat in value from 0 - 7
```

```
        count = count + 1;
```

```
}
```

```
}
```

Register of producer memory

- I. Load R_{p,m}[count]
- II. INCR R_p
- III. Store m[count], R_p

```
void consumer(void)
```

```
{
```

```
    int itemc;
```

```
    while(true)
```

```
{
```

```
    while (count == 0);
```

```
    itemc = Buffer [out];
```

```
    out = (out + 1) modN;
```

```
    count = count - 1;
```

```
    process_item(itemc);
```

```
}
```

```
}
```

- ↓
- I. Load R_{c,m}[count]
 - II. DECR R_c
 - III. Store m[count], R_c

- IN is a variable used by the producer to identify the next empty slot in the buffer.
- OUT is a variable used by the consumer to identify from where it has to consume the item.

- Count is a variable used both producer and consumer to identify no of items present in the buffer at any point of time.
- **Shared Resources:**
 - Count Variable
 - Buffer
- **Two conditions to be followed:**
 - If the buffer is full, producer is not allowed to produce the item into the buffer.
 - If the buffer is empty, consumer is not allowed to consume the item from the buffer.

5.3.2 Universal Assumption

The running process can get pre-empted at any point of time after completion of current instruction.

Analysis:

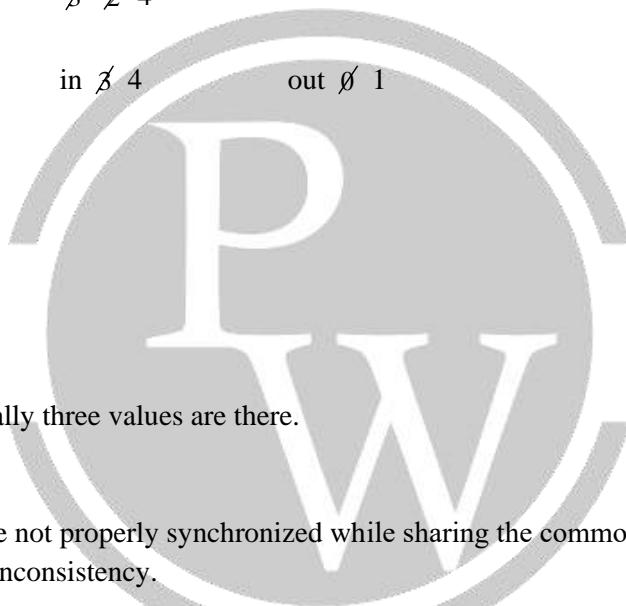
Itemp = 'A'
Itemc = 'X'
count
 $\cancel{3} \cancel{2} 4$

P → I Rp $\boxed{3}$
P → II Rp $\cancel{\boxed{4}}$

C → I Rc $\cancel{\boxed{2}}$
C → II
C → III

P → III

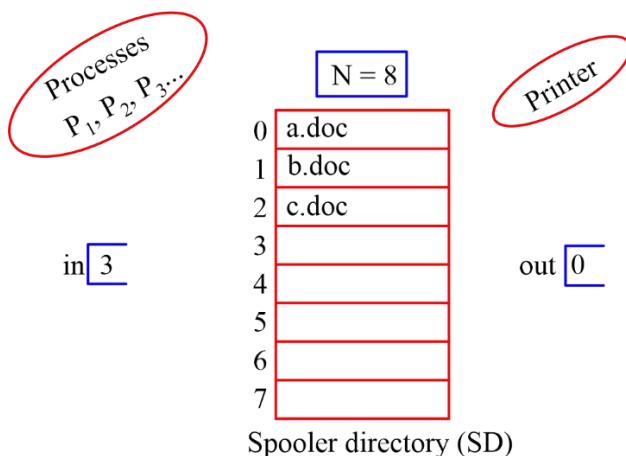
Final count value = 4. Though actually three values are there.



(1) Inconsistency:

The producer and consumer are not properly synchronized while sharing the common variable "count". Hence it is leading to the problem of inconsistency.

5.3.4 Printer Spooler Daemon (Daemon → Background process)



5.3.5 Definitions

(1) Critical Section:

The portion of program text where shared variables or shared resources will be placed.

Example:

Count = Count + 1

OR

Count = Count - 1

(2) Non-Critical Section:

The portion of program text where the independent code of the processes will be placed.

Example:

$T_n = (in + 1) \bmod N$

(3) RACE condition:

The final value of any variable depends on execution sequence of the processes. This type of condition is called as RACE condition.

- To avoid the RACE condition, only one process is allowed to enter into critical section.

5.3.6 Conditions to be followed to achieve Synchronization

(1) Mutual Exclusion (M.E):

- No two processes may be simultaneously present inside the critical section at any point of time.
- Only one process is allowed to enter into critical section at any point of time.

(2) Progress:

- No process running outside the critical section should block the other intersected process from entering into critical section when critical section is free.
- If there is only one process trying to enter into critical section then it should be definitely allowed to enter into critical section.
- If two or more processes are trying to enter into critical section then one process should be definitely allowed to enter critical section.

(3) Bounded Waiting:

- No process should have to wait forever to enter into critical section.
- There should be a bound on getting chance to enter into critical section.
- Some process is indefinitely waiting to enter into critical section because critical section is always busy by some other processes. This situation should not arise.
- If the bounded waiting is not satisfied then it is possible for starvation.

(4) No assumptions related to hardware and the processor speed:

- Number of processes.

Solutions:**I. Software Type:**

- (a) Lock Variables
- (b) Strict alteration or Decker's Algorithm
- (c) Petersons Algorithm

II. Hardware Type:

- (a) TSL Instruction Set
- (b) Test and Set Lock

III. OS Type:

- (a) Counting semaphore
- (b) Binary Semaphore

IV. Programming Language Compiler Support Type:

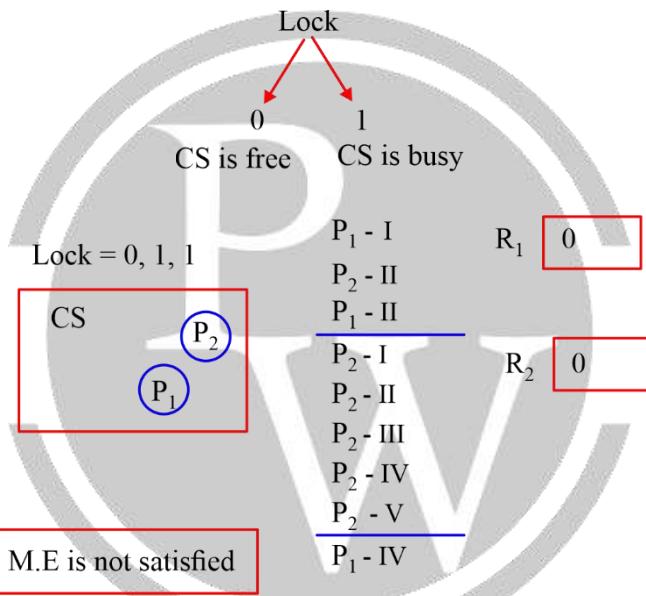
- (a) Monitors

I. Software Types:

(a) Lock Variables:

Entry Section:

- I. Load R_i, m[Lock] (R_i → Respective Process Register)
- II. Cmp R_i, #0
- III. JNZ to Step (I)
- IV. Store m[Lock], #1
- V. C.S
- VI. Store m[Lock], #0



Analysis:

We have proved that both the processes P₁ and P₂ are entering into the critical section at the same time, Hence mutual exclusion is not satisfied and the solution is bound to be incorrect.

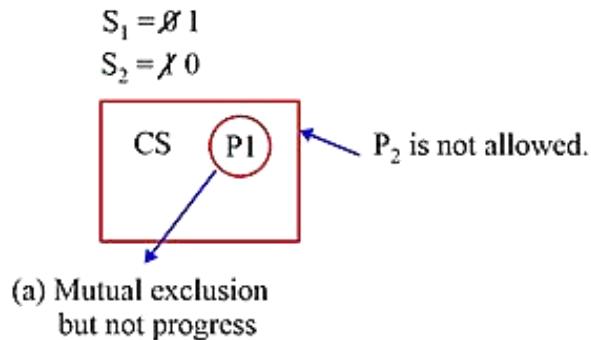
(b) Strict Alteration or Decker's Algorithm:

(Process takes 'Turn' to enter into C.S)

Process 'P ₀ ' code	Process 'P ₁ ' code
<pre>while (true) { non_cs(); while (turn != 0); c.s turn = 1; }</pre>	<pre>while (true) { non_cs(); while(turn != 1); c.s turn = 0; }</pre>

Important Points:

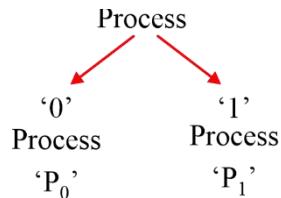
- The pre-emption is just a temporary stop and the process will come back and continue the remaining execution.
- If there is any possibility of solution becoming wrong by taking the pre-emption then consider the pre-emption.
- If any solution is having deadlock the progress is not satisfied.



(c) Peterson's Algorithm:

(Two Process Solution)

```
#define N 2
#define TRUE 1
#define FALSE 0
int turn;
int interested [N];
void enter_Region (int process)
{
    1. int other
    2. other = 1 - process;
    3. interested [process] = TRUE;
    4. turn = process;
    5. while (turn == process && interested [other] == TRUE);
}
C.S
void leave_Region(int process)
{
    interested [process] = FALSE;
}
initially
interested [0] = FALSE;
interested [1] = FALSE
```



TURN is a shared variable used by both the processes P_0 and P_1 , interested [N] is also shared by both the processes.

5.3.7 Hardware Type

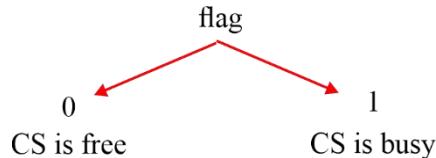
(a) TSL Instruction Set: (Test and Set Lock):

- **TSL Register Flag:**

Copies the current value of flag into register and stores the value of '1' into flag in a single atomic cycle without any pre-emption.

- **Entry Selection:**

1. TSL Ri, m[flag]
2. Cmp Ri, ≠ 0
3. JMP to step (1)
4. c.s
5. Store m[flag], ≠ 0



Algorithm	M.E.	Progress	Bounded Waiting
1. Lock Variable	✗	✓	✗
2. Strict alteration or Decker's Algorithm	✓	✗	✓
3. Peterson's Algorithm	✓	✓	✓
4. TSL Instruction	✓	✓	✗

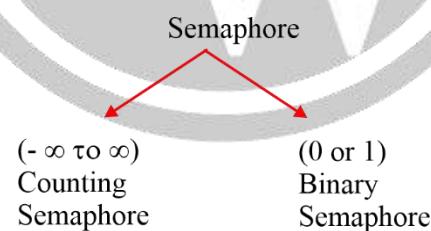
5.3.8 OS Type

Semaphore:

Semaphore is an integer variable which is used by the various processes in a mutual exclusive manner to achieve synchronization.

Improper usage of semaphore will also give the wrong results.

Semaphore is categorised into 2 types:



The two different operations will be performed on the semaphore variable.

- (1) Down, or wait (); or p ()
- (2) up (); or signal (); or v (); or release ();

(a) Counting Semaphore:

```

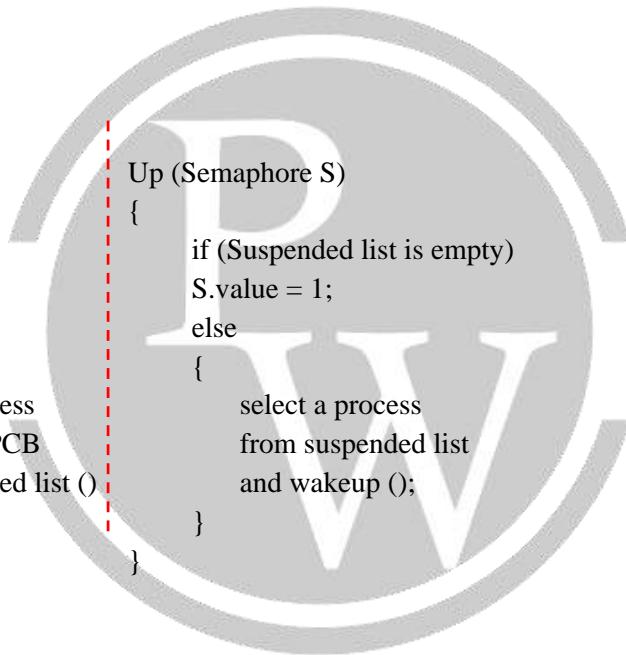
Down (Semaphore s)
{
    s.value = s.value - 1;
    if (s.value < 0)
    {
        Block the process and
        place its PCB in the
  
```

```
suspended list ();
}
}
Up (Semaphore s)
{
    s.value = s.value + 1;
    if (s.value ≤ 0)
    {
        Select a process from
        suspended list and
        wakeup ();
    }
}
```

(b) Binary Semaphore:

```
Down (Semaphore S)
{
    if (S.value == 1)
        S.value = 0;
    else
    {
        block the process
        and place its PCB
        in the suspended list ()
    }
}

Up (Semaphore S)
{
    if (Suspended list is empty)
        S.value = 1;
    else
    {
        select a process
        from suspended list
        and wakeup ();
    }
}
```

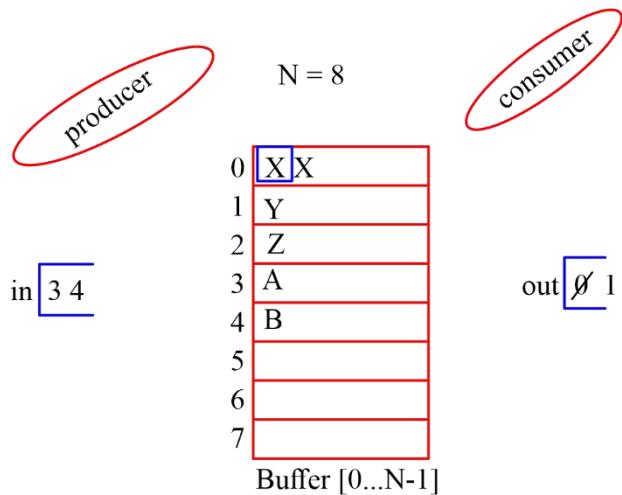


Notes: (Applicable for both counting and binary semaphore).

- ⇒ Every semaphore variable will have its own suspended list.
- ⇒ The down and up operations are atomic [OS will prevent the interrupts]
- ⇒ If more than one process is in the suspended list then every time when we perform one up operation one process will wakeup from the suspended list and this will be based on (FIFO → to ensure bounded waiting)
- ⇒ If two or more processes are in the suspended list and there is no other process to wakeup these processes then those processes are said to be involved in the deadlock.

5.4 Classical problems of IPC (Inter Process Communication)

- Producer consumer with semaphore:



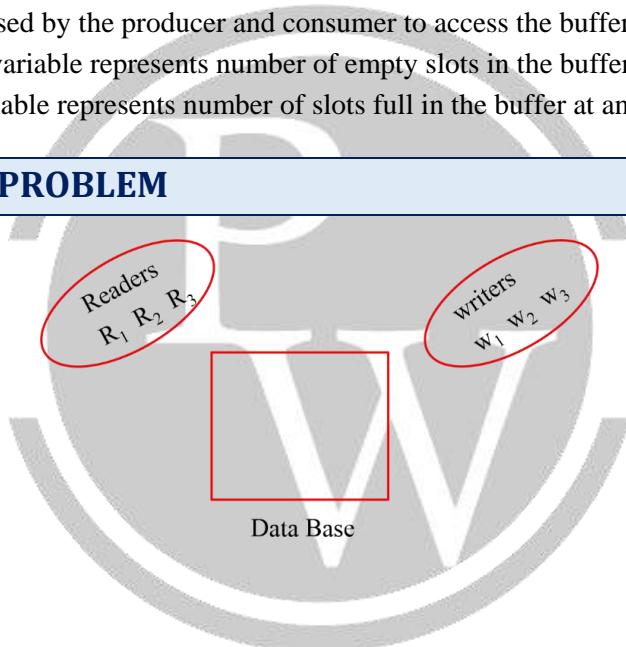
```
semaphore mutex = 1;  
semaphore empty = N;  
semaphore full = 0;  
void producer(void)  
{  
    int item p;  
    while(true)  
    {  
        produce_item (item p);  
        down(empty);  
        down(mutex);  
        buffer [in] = item p;  
        in = (in + 1) mod N  
        up(mutex);  
        up(full);  
    }  
}
```

```
void consumer (void)  
{  
    int item c;  
    while (true)
```

```
{  
    down (full);  
    down (mutex);  
    item c = buffer [out];  
    out = (out + 1)mod N;  
    up (mutex);  
    up (empty);  
    process_item (item c);  
}  
}
```

- mutex is a binary semaphore used by the producer and consumer to access the buffer in a mutual exclusive manner.
- empty is counting semaphore variable represents number of empty slots in the buffer at any point of time.
- full is counting semaphore variable represents number of slots full in the buffer at any point of time.

5.5 READERS WRITERS PROBLEM



```
int rc = 0;  
semaphore mutex = 1;  
semaphore db = 1;  
void reader (void)  
{  
    while (true)  
    {  
        down (mutex);  
        rc = rc + 1;  
        if (rc == 1) down (db);  
        up (mutex);  
    }  
}
```

```
}
```

```
void writer (void)
```

```
{
```

```
    while (true){
```

```
        down (db);
```

```
        D.B
```

```
        up (db);
```

```
    }
```

```
}
```

5.5.1 conditions to be followed

1) R - W ✗

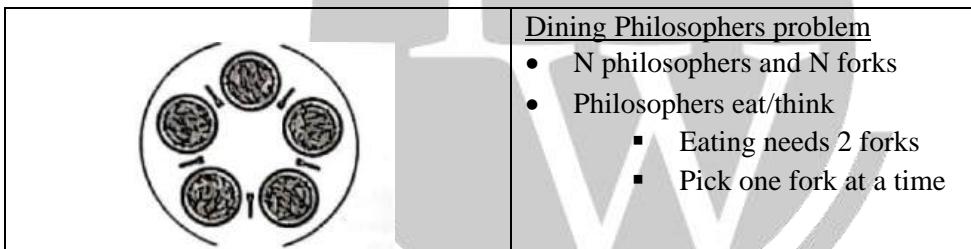
2) R - R ✓

3) W - R ✗

4) W - W ✗

- rc is a integer variable represents readers count i.e number of readers present in the data base at any point of time.
- mutex is a binary semaphore used by the readers in mutual exclusive manner.
- db is a binary semaphore variable used by readers and writers in a mutual exclusive manner.

5.5.2 Dining Philosophers' problem



5.5.3 Solution 1 for Dining Philosophers problem

```
# define N5                      /* number of philosophers */  
Void philosophers (int i)          /* i: philosophers' number from 0 to 4 */  
{  
    while (TRUE) {  
        think ();  
        take_fork (i);  
        take_fork ((i+1)%N)  
        eat ();  
        put_fork (i);  
        put_fork ((i+1)%N)  
    }  
}
```

- This solution to dining philosopher problem suffers from Deadlock.
- Everyone picks up their left fork first, then waits for right fork... ⇒ this leads to Starvation!

- This solution to dining philosophers' problem suffers from Deadlock everyone picks up their left fork first, then waits for right fork => This leads to Starvation!

5.5.4 Solution 2 – state based

```
# define N          5           /* number of philosophers */
# define LEFT        (i+N-1)%N   /* number of I's left neighbours */
# define RIGHT       (i+1)%N    /* number of I's right neighbours */
# define THINKING    0          /* philosopher is thinking */
# define HUNGRY      1          /* philosopher is trying to get forks */
# define EATING       2          /* philosopher is eating */
typedef int semaphore
int state [N];
semaphore mutex = 1;
semaphore s[N];

void philosopher (int i)
{
    while (TRUE) {
        Think();
        Take_forks(i);
        Eat();
        Put_forks(i);
    }
}

void take_forks(int i)
{
    down(&mutex);
    state [i] = HUNGRY;
    test(i);
    up(&mutex);
    down(&s[i]);
}

void put_forks(i)
{
    down(&mutex);
    state [i] = THINKING;
    test(LEFT);
    test(RIGHT);
    up(&mutex);
}

void test(i)
{
    if (state[i] == HUNGRY && state [LEFT] == EATING && state [RIGHT] == EATING) {
        state [i] = EATING;
        up(&s[i]);
    }
}
```

Made picking up left and right chopsticks an atomic operation ξ or, $N - 1$ philosophers but N chopsticks ξ ...both changes prevent deadlock.

5.6 Monitors:

- Monitors is a programming language compiler support type of solution to achieve synchronization.
- Monitors is collection of variables and procedures combined together in a special kind of module or package.
- The process running outside the monitor cannot directly access the internal variables of the monitor but however they can call the procedures of the monitor.
- Monitors has an important property that only one process can be active inside the monitor at any point of time.

5.6.1 Syntax

Monitor example

{

Variables;

Condition variables;

Procedure P₁

{

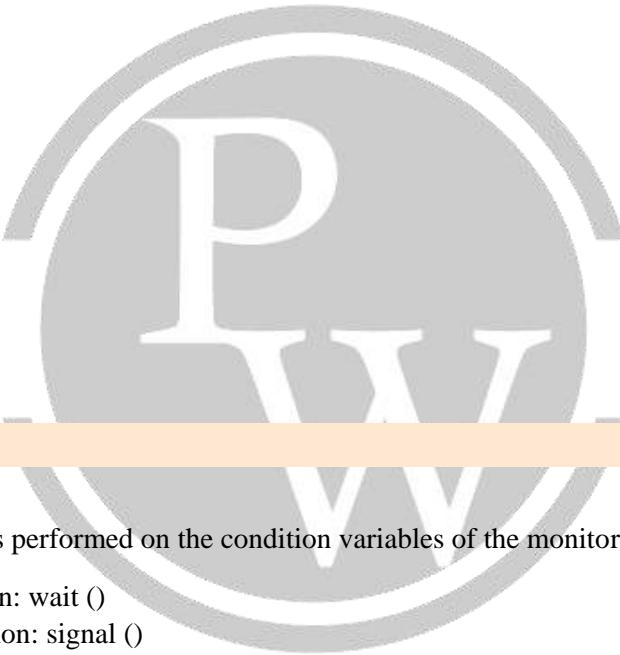
}

Procedure P₂

{

}

}



5.6.2 Condition variables

Condition x,y;

The two different operations performed on the condition variables of the monitor.

1. Wait operation: wait ()
2. Signal operation: signal ()

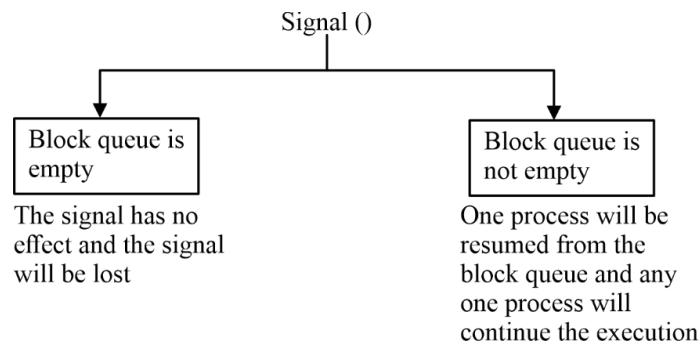
- **Wait ()**

Ex – x.wait (); or wait (x)

The process performing wait operation on any condition variable will be suspended and suspended process will be placed in the “block queue” of respective condition variable.

- **Signal ()**

Ex – x.signal (); or signal(x)



5.6.3 Concurrent Programming

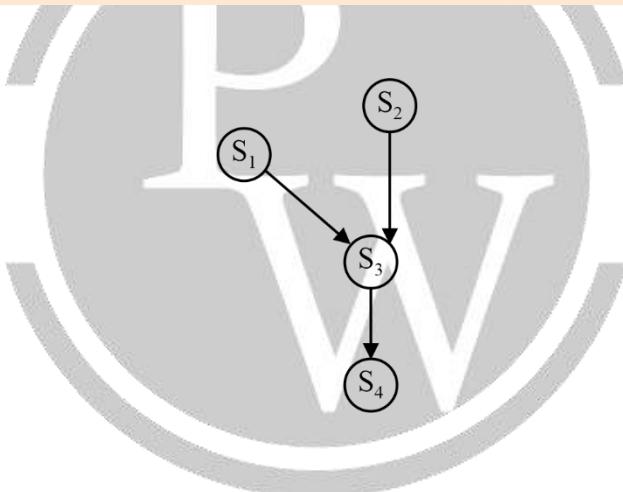
$S_1: a = b + c ;$
 $S_2: d = e * f ;$
 $S_3: g = a / d ;$
 $S_4: h = g * i ;$

Read set = {b, c, e, f, a, d, g, i}

Write set = {a, d, g, h}

5.6.4 Precedence graph

S_1, S_2 can execute concurrently



- Any two statements S_i and S_j can be executed concurrently or parallel if they are following the conditions.
 - (1) $R(S_i) \cap W(S_j) = \emptyset$
 - (2) $W(S_i) \cap R(S_j) = \emptyset$
 - (3) $W(S_i) \cap W(S_j) = \emptyset$
 - The real concurrent programming is possible only on the multiprocessor system.
 - Concurrent has 3 different meanings
 - They can execute concurrently or parallel
 - They don't have any dependency
- Anyone can start first [for single processor this will be applicable]



6

DEADLOCK

6.1 Concept of Deadlock

- A set of blocked processes each holding a resource and waiting to acquire a resource held by another process in the set.
- Example:
 - System has 2 disk drives.
 - P_1 and P_2 each hold one disk drive and each needs another one.

6.2 System Model

- Resource types R_1, R_2, \dots, R_m
- Resources include CPU cycles, memory space, I/O devices
- Each resource type R_i has W_i instances.
- Each process utilizes a resource as follows:
 - request
 - use
 - release

6.3 Deadlock Characteristics

6.3.1 Mutual Exclusion

There should be a one-to-one relationship between a resource and a process.

6.3.2 Hold and Wait

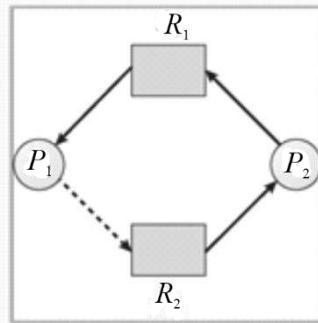
This condition arises when a process is requesting another resource while holding on to a resource.

6.3.3 No Pre-emption

The resource has to be voluntarily released by the process after its execution. It is not allowed to forcibly pre-empt a process to release its held resources.

6.3.4 Circular Wait

The processes are circularly waiting on each other for the resources.

**Note:**

If all the deadlock characteristics simultaneously exist in the system, then the system is in deadlock.

6.4 Deadlock Prevention

It ensures that the system will *never* enter into a deadlock state. Deadlock can be prevented by unsatisfying one of the mentioned deadlock characteristics.

- Mutual exclusion cannot be unsatisfied because of the concept of sharable and non-sharable resources.
- Hold and Wait characteristic can be unsatisfied through any of the following strategies:
 - (i) A process should be assigned all the required resources before the start of its execution. This may lead to low device utilization.
 - (ii) The process should release all the existing resources before making a new request. This may eventually lead to starvation.
- Pre-emption can be achieved as:
- Suppose a process P_1 is requesting for a resource R which is held by another process P_2 . If P_2 is already in execution then P_1 has to wait; else, the resource R will be pre-empted from P_2 and allocated to P_1 .
- Circular wait can be avoided by the following algorithm:
 - (i) Assign unique numbers to resources. (assume there exists single instance of a resource).
 - (ii) A process can request for a resource in either increasing or decreasing order of enumeration.

6.5 Deadlock Avoidance

Deadlock avoidance is achieved by implementing Banker's algorithm. Banker's algorithm ensures that the system never enters in unsafe state.

- When a process requests an available resource, system must decide if immediate allocation leaves the system in a safe state.
- System is in safe state if there exists a sequence $\langle P_1, P_2, \dots, P_n \rangle$ of ALL the processes in the systems such that for each P_i , the resources that P_i can still request can be satisfied by currently available resources + resources held by all the P_j .
- If a system is in safe state \Rightarrow no deadlocks.
- If a system is in unsafe state \Rightarrow possibility of deadlock.

6.5.1 Data Structures for Banker's algorithm

Let n = number of processes and m = number of resources, then-

- **Available:** Vector of length m . If Available [j] = k , there are k instances of resource type R_j available.
- **Max:** $n \times m$ matrix. If Max [i,j] = k , then process P_i may request at most k instances of resource type R_j .
- **Allocation:** $n \times m$ matrix. If Allocation [i,j] = k then P_i is currently allocated k instances of R_j .
- **Need:** $n \times m$ matrix. If Need [i,j] = k , then P_i may need k more instances of R_j to complete its task.
- $\text{Need } [i,j] = \text{Max}[i,j] - \text{Allocation } [i,j]$.

6.5.2 Banker's Algorithm

- Algo Bankers(i , Request, Available, Allocation, Need){
 If(Need _{i} ≤ Max _{i})
 {
 If(Need _{i} ≤ Available)
 {
 Available = Available – Need _{i} ;
 Allocation = Allocation + Need _{i} ;
 Max _{i} = Max _{i} - Need _{i} ;
 Run safety Algorithm;
 If the system is in safe state, then grant the Need _{i} ;
 Else block the process;
 }
 }
 }

6.6 Deadlock Detection

- A cycle in the resource allocation graph represents deadlock only when resources are of single-instance type.
- If the resources are of multiple instance type, then the safety algorithm is used to detect deadlock.

6.7 Deadlock Recovery

A system can recover from deadlock through adoption of the following mechanisms:

- Process termination
- Resource Pre-emption
- Ostrich Algorithm:
- Ignore the problem and pretend that deadlocks never occurred in the system; used by most operating systems, including UNIX.



7

MEMORY MANAGEMENT

7.1 Introduction

- In multiprogramming system, the task of subdividing the memory among the various processes is called memory management.
- The task of the memory management unit is the efficient utilization of memory and minimize the internal and external fragmentation.

7.2 Logical versus Physical Address

- The address generated by CPU is called the logical address.
- The address perceived by the memory unit is called physical address.

7.3 Memory Management Unit

- Hardware device that maps virtual to physical address.
- In MMU scheme, the value in the relocation register is added to every address generated by a user process at the time it is sent to memory.
- The user program deals with *logical* addresses; it never sees the *real* physical addresses.

7.3.1 Loading

It is defined as bringing the program from the secondary to the main memory.

It is classified into three types:

- (i) Absolute Loading
 - A given program is always loaded into the same memory location whenever loaded for execution.
- (ii) Relocatable Loading
 - A given program is loaded into any desired memory location whenever loaded for execution.
 - The compiler must generate relative address for the program.
- (iii) Dynamic Loading
 - Routine is not loaded until it is called better memory-space utilization (unused routine is never loaded, postponed until execution time).
 - Useful when large amounts of code are needed to handle in frequently occurring cases.
 - No special support from the operating system is required. It is implemented through program design
 - Address translation is performed through the hardware.

7.3.2 Linking

Linking is the process of collecting and combining various pieces of code and data into a single file that can be loaded (copied) into memory and executed. It can be performed at compile time, load time or run time.

It is classified into two types:

- **Static Linking**

Static linkers take as input a collection of relocatable object files and command-line arguments and generate as output a fully linked executable object file that can be loaded and run.

- **Dynamic Linking**

A shared library is an object module that, at run time, can be loaded at an arbitrary memory address and linked with a program in memory. This process is dynamic linking.

7.3.3 Address Binding

Association of the program instructions and data into the actual physical memory locations is called as address binding. Address binding of instructions and data to memory addresses can happen at three different stages

- **Compile time:** If memory location known a priori, **absolute code** can be generated. It must recompile code if starting location changes.
- **Load time:** It must generate **relocatable code** if memory location is not known at compile time.
- **Execution time:** Address binding is delayed until run time if the process can be moved during its execution from one memory segment to another. It needs hardware support for address mapping (e.g., base and limit registers).

7.4 Memory Management Techniques

7.4.1 Contiguous Memory Allocation

It comprises of – Fixed Partition Scheme and Variable Partition Scheme

Fixed Partition Scheme / Static Partitioning

- The memory is divided into a fixed number of partitions of equal/unequal sizes.
- Every partition is associated with limit registers.
- The degree of multiprogramming is restricted by the number of partitions.
- Partition size may not be large enough for any waiting process.
- Internal fragmentation may be present.

Variable Partition Scheme / Dynamic Partitioning

- Initially, memory is available as a single continuous free block. When a process arrives, a hole large enough to accommodate it is created in the memory.
- There is no internal fragmentation.
- It suffers from external fragmentation.
- It is associated with the overhead of compaction.

Note

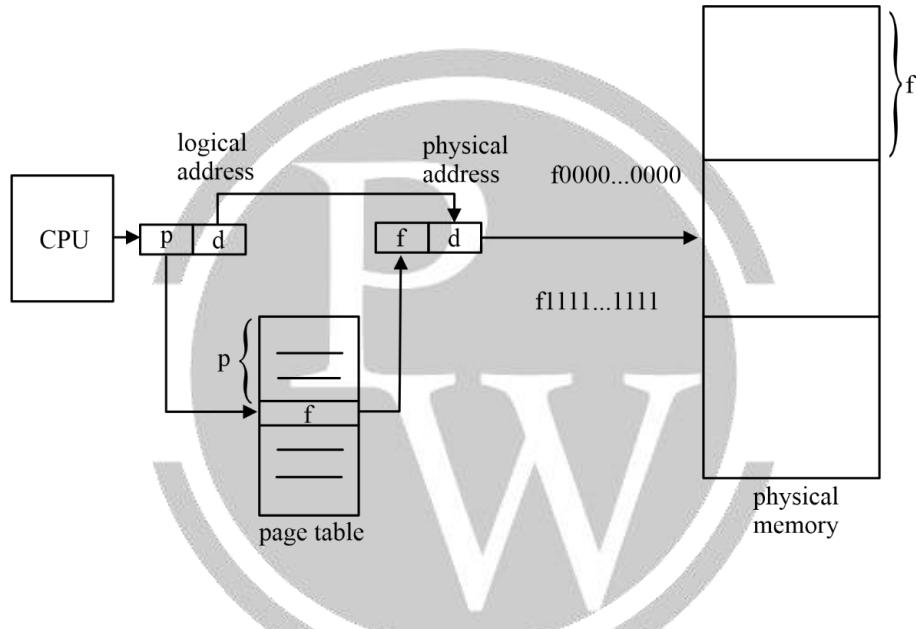
If more than one partition is sufficient to accommodate a process, then any of the following dynamic allocation methods can be adopted:

- First Fit: Allocate the first partition that is big enough starting from the beginning of the memory.
- Next Fit: It behaves similar to first fit except that it scans the memory after the ‘last allocation point’.
- Best Fit: It scans the entire memory to find the smallest sufficient partition to accommodate the process.
- Worst Fit: It scans the entire memory to find the largest sufficient partition to accommodate the process.

7.4.2 Non-Contiguous Memory Allocation

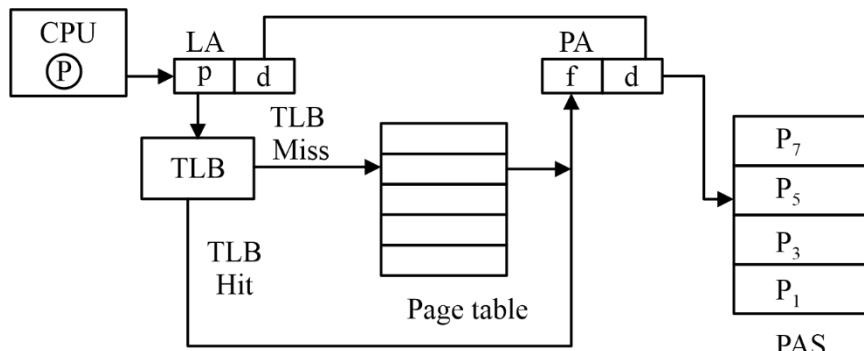
Paging

- The technique of mapping CPU generated logical address to physical address is called paging.
- Logical address space is divided into equal size pages. Physical address space is divided into equal size frames. In paging, the frame size is equal to the page size.
- When a process is created, paging will be applied on the process. A page table will be maintained in the main memory for a process. The base address of the page table will be stored in the process control block.
- The number of entries in the page table is equal to the number of the pages in the logical address space. Each page table entry contains the frame number. Hence, the page table is also known as the **address translation table**.
- There is no external fragmentation in paging. Internal fragmentation may exist in the last page and is formulated as $\left\lceil \frac{p}{2} \right\rceil$ where p is the page size.



Paging with TLB

- Translation Look Aside Buffer** is a hardware device implemented using associative registers.
- TLB is added to improve the performance of paging. The TLB access time will be very less compared to the main memory access time.
- TLB contains frequently referred page numbers and their corresponding frame numbers.



- TLB access time = c
- TLB hit ratio = x

Then the formula for effective memory access time:

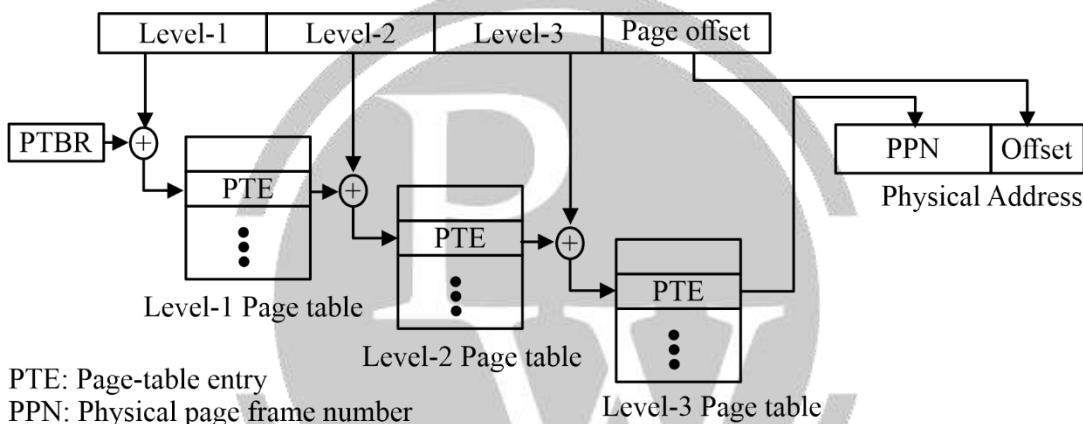
$$\text{E.M.A.T} = x(c + m) + (1 - x) \cdot 1 \cdot (c + \underline{2m})$$

The diagram shows the formula $\text{E.M.A.T} = x(c + m) + (1 - x) \cdot 1 \cdot (c + \underline{2m})$. Three arrows point downwards from the terms $x(c + m)$, $(1 - x) \cdot 1 \cdot (c + \underline{2m})$, and $\underline{2m}$ respectively, to the corresponding text below.

1 TLB access	1 MR for	1 MR for PT 1 MR for actual page
		actual page

Multilevel Paging

- To avoid the overhead of maintaining large size page tables, multilevel paging will be applied.
 - In multilevel paging, pages of the page table are brought into the main memory.
 - The page tables of all the levels of multi-level paging are kept in the memory.
 - The entries of the level-1 page table are pointers to the level-2page table.
 - The entries of the level-2 page table are pointers to the level-3 page table.
 - The entries of the last level page table will have frame numbers of the actual page.
 - All levels page table entry must contain frame number.
 - Address translation is done as:



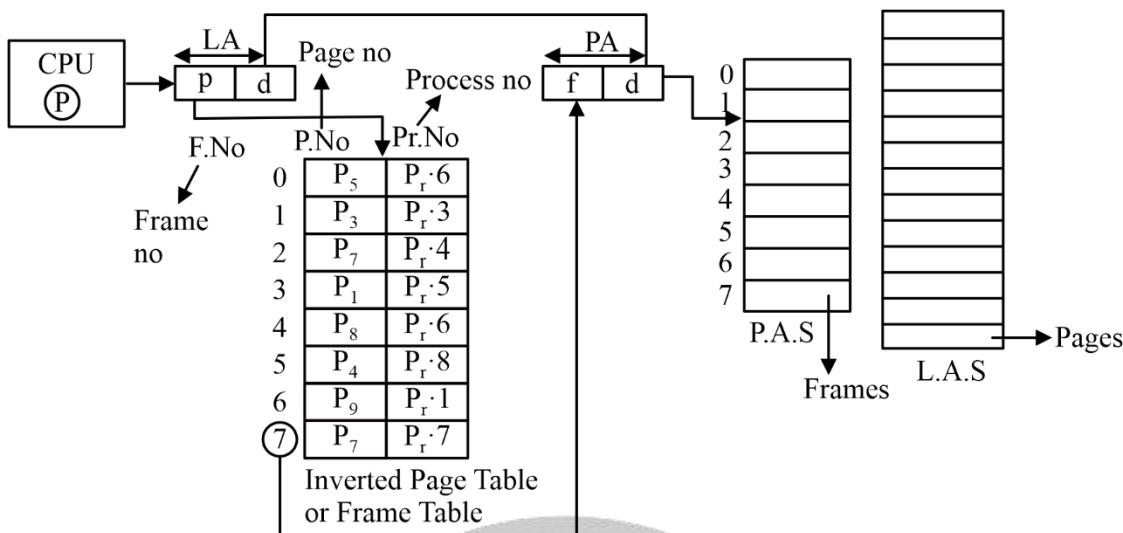
- Performance of multi-level paging with TLB:
Assume, TLB hit ratio = p , TLB access time = c , Main Memory Access time = m
If the system uses n -level paging, then effective memory access time is given as
$$EMAT = p(c + m) + (1 - p)(c + (n + 1)m)$$

$$EMAT = p(c + m) + (1 - p)(c + (n + 1)m)$$

Inverted Paging

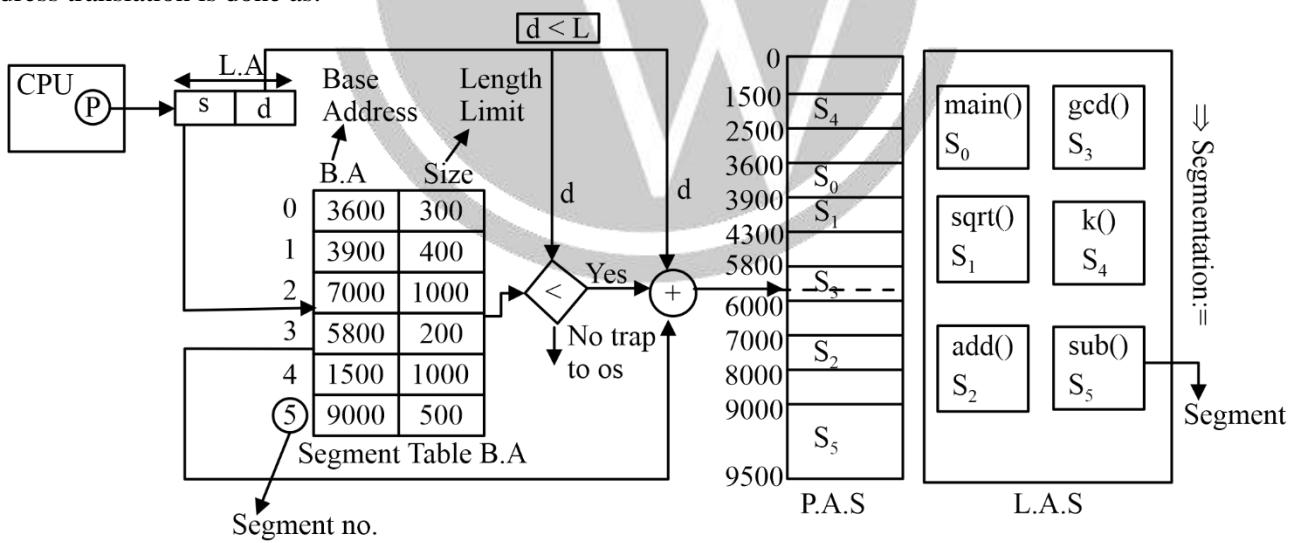
- Inverted paging maps physical frames to virtual pages.
 - Instead of maintaining multiple page tables for different processes, an inverted page table can be maintained.
 - The number of entries in the inverted page table is equal to the number of the frames in the physical address space.
 - Each inverted page table entry contains a page number and a process identifier. It gives an idea about ‘which page of which process is allocated in which frame’.
 - The disadvantage of inverted paging is the increased lookup time and hard to implement.

- Address translation is done as:



Segmentation

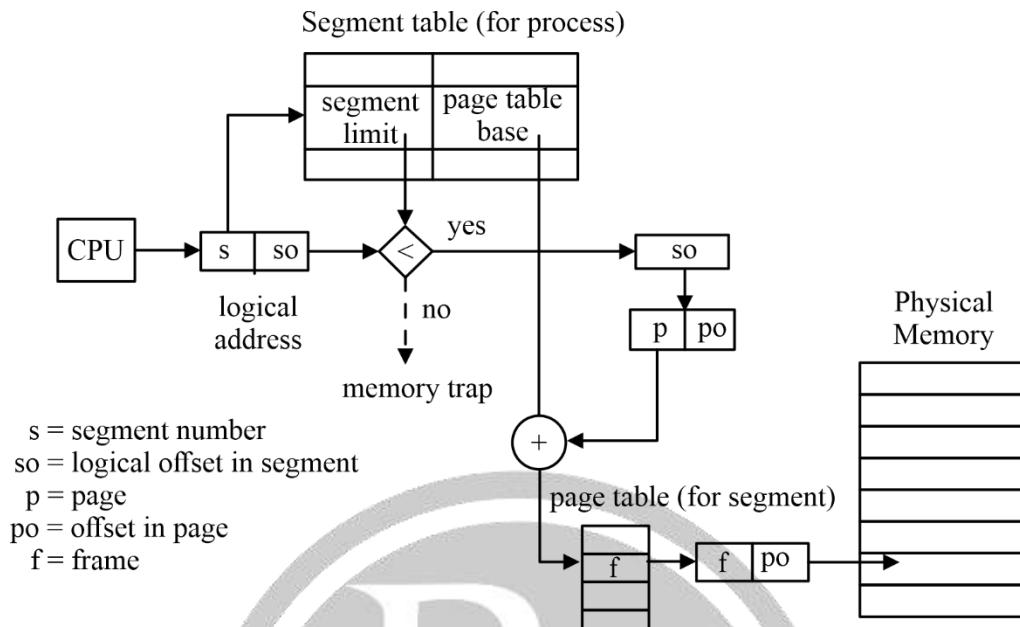
- Paging does not follow user's view of memory allocation. Instead of dividing the process into equal sized pages, it can be divided into variable length segments.
- A segment table is maintained for each process. The number of entries in the segment table is equal to the number of segments of a process.
- Each segment table entry contains the base (starting address) and limit(length) of the segment.
- Segmentation does not suffer from internal or external fragmentation.
- Address translation is done as:



Segmented Paging:

- To avoid the overhead of bringing large sized segments of a process into the main memory, paging will be applied on the segments.
- Pages of the segment will be loaded into the main memory.
- A page table will be maintained for each segment of the process.
- The number of entries in the page table is equal to the number of the pages of the segment in the logical address space.
- Lookup time increases.

- It suffers from internal fragmentation only.
- Address translation is done as:



08

VIRTUAL MEMORY

8.1 VM Concept

- It gives an illusion to the programmer that programs having larger size than the physical memory can be executed.
- It allows address spaces to be shared by several processes.

8.2 VM Implementation

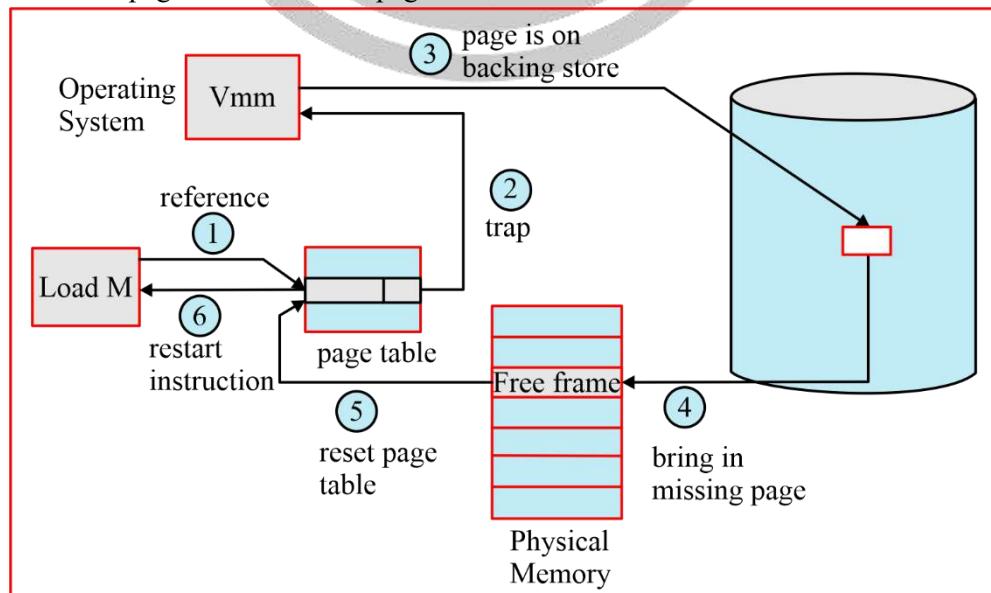
Virtual memory can be implemented through:

- Demand Paging
- Demand Segmentation

8.2.1 Demand Paging

Loading the pages from the secondary memory to the main memory on demand is called ‘demand paging’.

- When the CPU tries to refer a page which is not available in the main memory, ‘page fault’ occurs. A signal is sent to the OS regarding the page fault. The OS locates the page into the Logical address space and loads it into the main memory frame. If the memory frames are full, appropriate page replacement algorithm is used. The respective page table entries are also updated accordingly.
- The time taken to service a ‘page fault’ is called ‘page fault service time’.



8.3 Performance of virtual Memory

8.3.1 Temporal Issues

Let,

Page fault service time = 'S'

Main memory access time = 'M'

Page fault rate = 'P' where $0 \leq P \leq 1$

The effective memory access time is formulated as-

$$\text{EMAT} = P \times S + (1 - P) \times M$$

(Assume, page table access time is neglected)

8.3.2 Demand Paging with TLB

Assume, TLB hit ratio = 'h'

TLB access time = 'c'

Page fault service time = 'S'

Main memory access time = 'M'

Page fault rate = 'P' where $0 \leq P \leq 1$

The effective memory access time is formulated as-

$$\text{EMAT} = h(c + m) + (1 - h)(c + (P \times S + (1 - P) \times M))$$

8.3.3 Page Replacement Algorithms

FIFO

- When a page fault occurs and all the memory frames are full, FIFO algorithm replaces the oldest page to allocate the page referred by the CPU.
- It is implemented using a queue or time-stamp on pages.
- Sometimes, even on increasing the number of frames, the page fault rate increases. This situation is called Belady's Anomaly.

LRU

- LRU algorithm replaces the page that has not been used for the longest period (least recently used page).
- It is implemented using a stack or counter.

Optimal

- Optimal algorithm replaces the page that will not be used for the longest period in future.
- For a fixed number of frames, optimal algorithm gives the least page fault rate.
- It cannot be implemented in real time as it requires future references.

Note

Reference String is a set of successively unique pages referred in a given list of virtual addresses.

8.3.4 Thrashing

When CPU utilization is low, the OS may increase the degree of multiprogramming. After a certain point, the throughput of the system will gradually decrease as the system spends more time in page replacements owing to the lack of frames. This phenomenon is called ‘thrashing’.

8.3.5 Working Set Model

It is defined as the set of the unique pages referred during the past ‘ Δ ’ references.

- If $\Delta <$ (total number of frames), the OS can bring in more processes.
- If $\Delta >$ (total number of frames), the OS should instruct the mid-term scheduler to suspend some of the processes to avoid thrashing.



9

FILE SYSTEMS

9.1 File

- Collection of logically related entities/records

9.1.1 Attributes

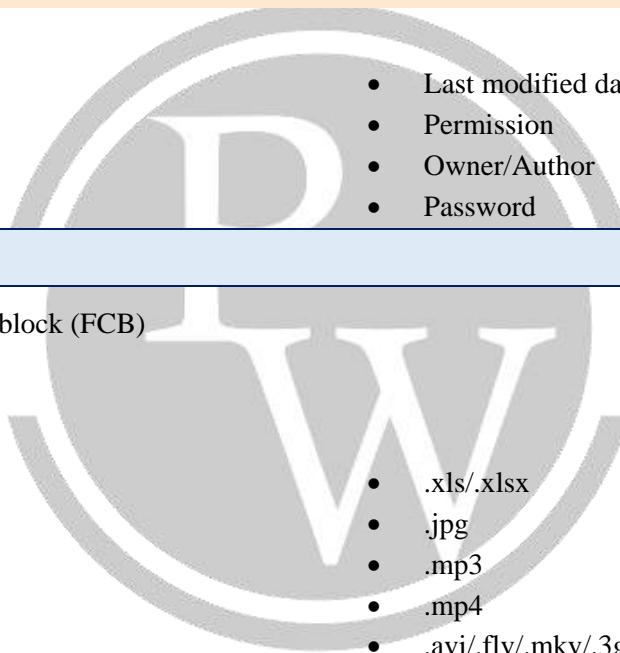
- File Name
- File Type
- File Size
- Location of file
- Generation date
- Last modified date
- Permission
- Owner/Author
- Password

9.2 File Context

File context is stored in file control block (FCB)

File will have various types—

- .doc
- .txt
- .pdf
- .exe
- .obj
- .dll
- .png
- .apk
- .xls/.xlsx
- .jpg
- .mp3
- .mp4
- .avi/.flv/.mkv/.3gp
- .c/.cpp/.java/.xml/.html



9.3 Operations performed on file

- create
- open
- write
- read
- hide
- save
- save as
- close
- copy
- paste
- cut
- delete
- rename
- send/share
- print

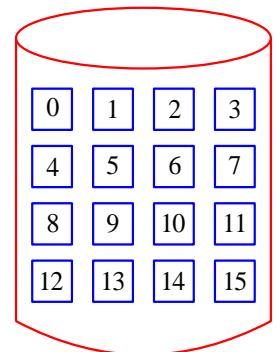
9.4 Access methods

- Sequential Access
- Random Access
- For better classification of files, files will be stored in directory.

9.5 Disk Space Allocation Methods

9.5.1 Contiguous Allocation

File	Starting Disk Block Address	Size w.r.t no. of DB
Abc.docx	2	4
Xyz.docx	9	5



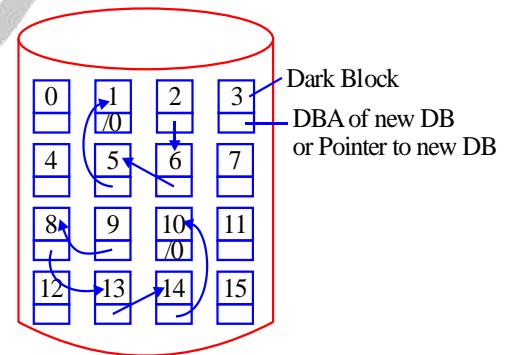
- In contiguous allocation, whenever file is created, disk blocks are allocated in a continuous manner.
 - Every file is associated with two parameters
- (1) Starting DBA
 - (2) Size
- Increasing the file size may not be possible always.
 - It suffers from external fragmentation.
 - Internal fragmentation may exist in the last disk block of the file.
 - It supports both sequential and random access of file.

Starting disk block Address + Offset Value \Rightarrow Random location

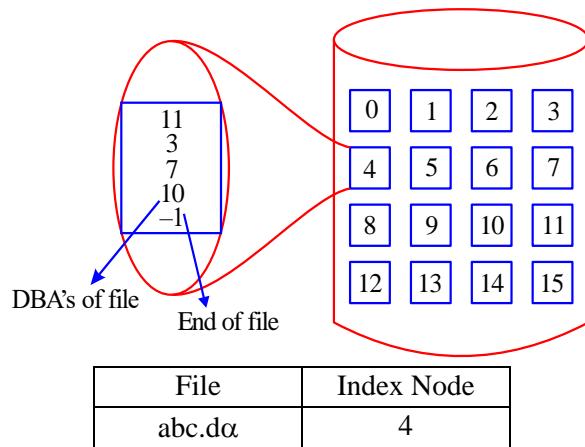
9.5.2 Linked Allocation/Non-contiguous allocation

File	Starting DBA	Ending DBA
abc.docx	2	1
xyz.docx	9	10

- In the linked allocation disk blocks are allocated in a non-continuous manner.
 - Every file is associated with 2 parameters–
- (1) Starting DBA
 - (2) Ending DBA
- Increasing file size is always possible if free disk block is available.
 - There is no external fragmentation.
 - Internal fragmentation may exist in last disk block of the file.
 - There is an overhead of maintaining pointer in every disk blocks.
 - If the pointer of any disk block is lost, the file will be truncated.
 - It supports only sequential access of the file.



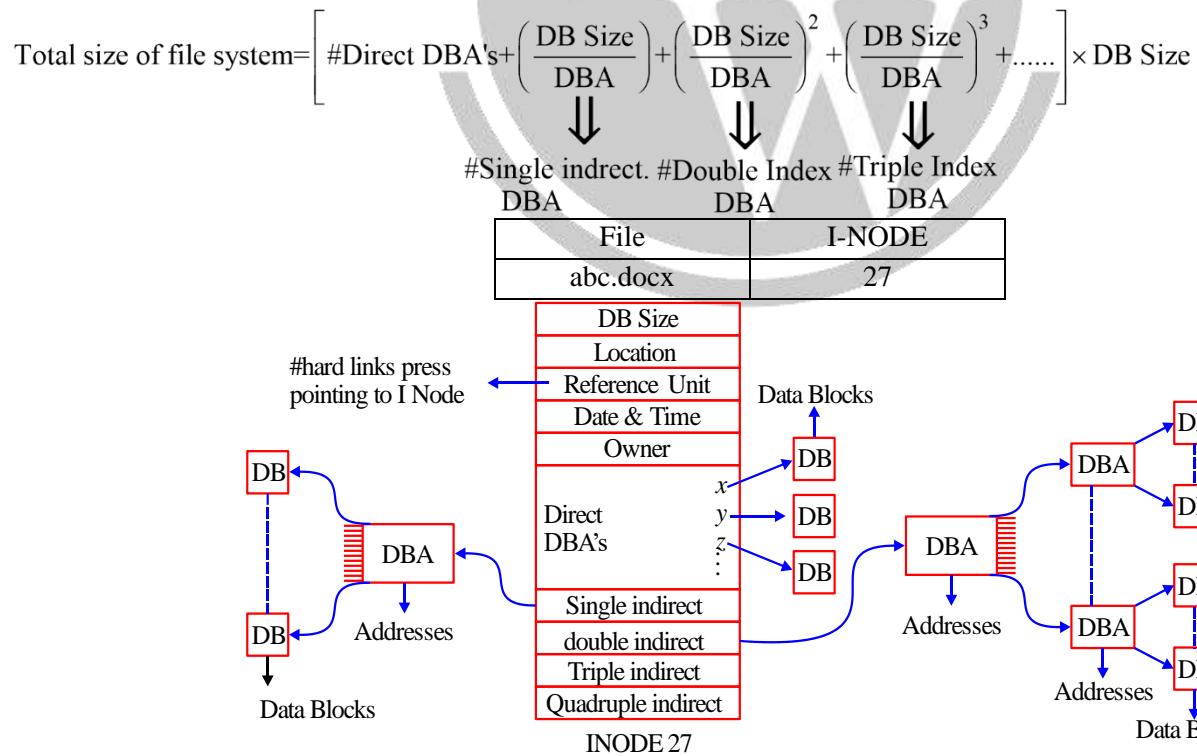
9.5.3 Indexed Allocation



- In the indexed allocation, every file is associated with its own index node.
- Index node contains all the disk block address of the file.
- There is no external fragmentation but internal fragmentation may exist in the last disk block of file.
- If the file is very large, then one disk block may not be sufficient to store all the disk block addresses of the file.
- If the file is very small, then it is waste of using one entire disk blocks. Just to store, the addresses.

9.7 UNIX I-NODE IMPLEMENTATION

9.7.1 An extension of indexed allocation



9.8 DISK FREE SPACE MANAGEMENT

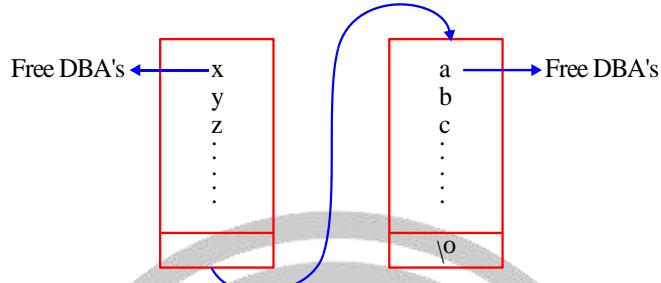
Size of Disk = 20 MB

DB Size = 1 KB

DBA = 16 Bits = 2 B

$$\# \text{ Disk blocks available on given disk} = \frac{\text{Size of disk}}{\text{DB size}} = \frac{20 \times 2^{20} B}{2^{10} B} = 20 \text{ K}$$

9.8.1 Free List Approach



- In the free list approach, some disk blocks are used just to store the free disk block addresses.

$$\# \text{ Disk block addresses possible to store in one disk block} = \frac{\text{DB Size}}{\text{DBA}} = \frac{2^{10}}{2} = 2^9$$

2^9 addresses \rightarrow 1 disk block

$$20 \text{ K addresses} \rightarrow \frac{1}{2^9} \times 2^{11} \times 10 \rightarrow 2^2 \times 10 = 40 \text{ disk block}$$

9.8.2 Bit Map Approach

- In Bit map approach, every disk block were be mapped with 1 binary bit

Free Disk Blocks

0, 2, 4, 6, 9, 11,

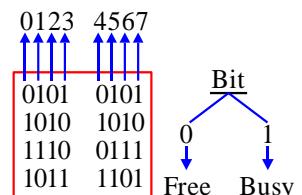
Busy Disk Blocks

1, 3, 5, 7, 8, 10,

Disk blocks we can map in 1 disk block = $1 \text{ K} \times 8 \text{ bits} = 8 \text{ K bits}$.

$8 \text{ K bits} \rightarrow 1$

$$20 \text{ K bits} \rightarrow \frac{1}{8\text{K}} \times 20\text{K} = 2.4 = 3$$



10

DISK SCHEDULING

10.1 Goal of Disk Scheduling

- Enhanced throughput.
- Minimize the average seek time of the disk.
- Fair chance to all the I/O requests.

10.2 Disk scheduling Algorithms

10.2.1 FCFS

It serves the first request in the queue.

10.2.2 SSTF

It selects the request with minimum seek time for the current head position.

10.2.3 SCAN or Elevator Algorithm

The disk arm starts from the current head position and moves towards the other end, servicing requests until it gets to the other end of the disk, where the head movement is reversed and servicing continues.

10.2.4 C-SCAN

The disk arm starts from the current head position and moves towards the other end, servicing requests until it gets to the other end of the disk, where the head movement is reversed and immediately returns to the beginning of the disk without servicing any requests at the return trip. The servicing then continues from the beginning of the disk again.

10.2.5 C-LOOK

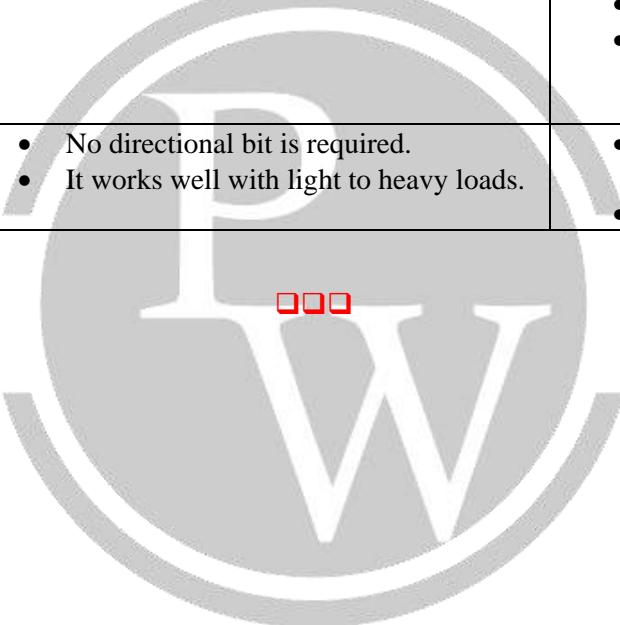
It is similar to C-SCAN where the arm goes as far as the last request in each direction, then reverses the direction immediately, without first going all the way to the end of the disk.

Note:

- Performance depends on the number and types of requests.
- I/O requests can be influenced by the file allocation methods.
- Either SSTF or LOOK is a reasonable choice for the default algorithm.

10.2.6 Comparison Among the Disk Scheduling Algorithms

Algorithm	Advantages	Disadvantages
First Come First Served	<ul style="list-style-type: none">• Easy to implement• Fair chance to all the I/O requests• It doesn't suffer from starvation	<ul style="list-style-type: none">• Seek time is not optimized.• It doesn't maximize throughput.
Shortest Seek Time First	<ul style="list-style-type: none">• It tends to minimize the arm movement.• It has better throughput than FCFS.	<ul style="list-style-type: none">• It may suffer from starvation.
SCAN/LOOK	<ul style="list-style-type: none">• It eliminates starvation.• It works well with light to heavy loads.	<ul style="list-style-type: none">• It is complex to implement.• Increased overhead.• It needs a directional bit to keep track of the head direction.
C-SCAN/C-LOOK	<ul style="list-style-type: none">• No directional bit is required.• It works well with light to heavy loads.	<ul style="list-style-type: none">• It is complex to implement.• Increased overhead.



OUR YOUTUBE CHANNELS



**GATE
WALLAH**
EE, EC & CS



GATE Wallah



**GATE
WALLAH**
ME, CE & XE



GATE Wallah (English)

ACCESS QUALITY CONTENT

For Free

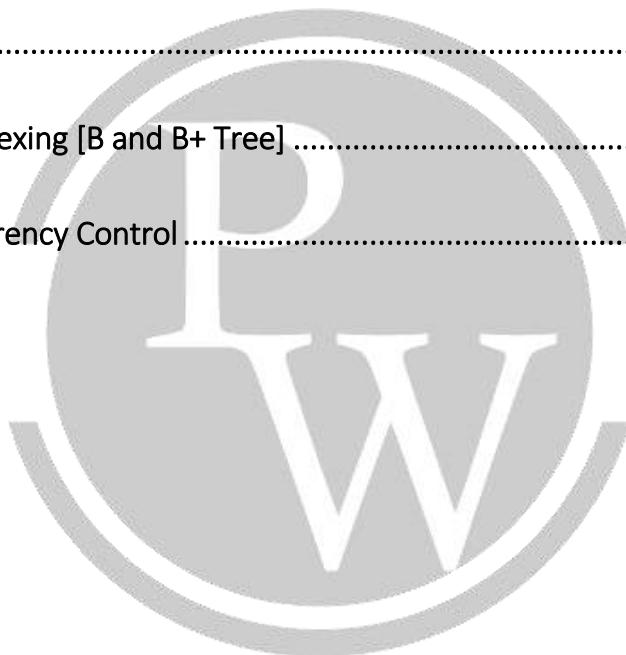
Database Management Systems



Database Management Systems

INDEX

1. Database Design 10.1 – 10.9
2. ER Model 10.10 – 10.13
3. Relation Model and SQL 10.14 – 10.25
4. File Organization and Indexing [B and B+ Tree] 10.26 – 10.31
5. Transactions and Concurrency Control 10.31 – 10.46



1

DATABASE DESIGN

1.1 Introduction

A database is an organized collection of structured data or related data.

1.2 Limitations of File System

- The physical details of the data to access the database are managed by the user.
- File system can be used to manage small database.
- When database is large, the operating system fails to control the concurrency.
- Only single user can access the whole data of the file system at a time.

1.3 Integrity Constraints Of RDBMS

- According to codd, data in database file must be stored in tabular format (set of row's and column's).
- According to codd, no two row's/records of the table should be equal.

1.3.1 Arity

Number of attributes of database table.

1.3.2 Cardinality

Number of records of database table.

1.4 Keys in database

- **Candidate Key:** A minimal set of attributes that differentiate records/row's of DB table uniquely.
- **Primary Key :** One of the candidate key whose field value cannot be null.
- **Simple Candidate Key:** A candidate key with only one attribute.
- **Compound Candidate Key:** A candidate key with atleast two attributes.
- **Overlapped Candidate Key:** Two or more candidate key with some common attribute.
- **Prime attribute:** The attribute that belongs to some candidate keys of a relation.
- **Non-prime attribute:** The attribute that does not belongs to any of the candidate keys of the relation.

Example :

Consider a relation R (ABCDE)

1. Assume candidate key : {AB, BC}

2. The above candidate keys are compound and overlapped.
3. Prime attribute {A, B, C}
4. Non-prime attribute : {D, E}

1.4.1 Difference between Primary key and Alternative key

Primary Key	Alternative Key
<ol style="list-style-type: none"> 1. Any one candidate key whose field value can not be null. 2. Atmost one primary key allowed for any DB table. 	<ol style="list-style-type: none"> 1. All candidate key of relation except primary key, whose field value can be null. 2. Many (o or more) alternative keys are allowed for DB table.

Note:

For any RDBMS table there must be atleast one candidate key, whose field value can not be null.

(Unique + Not Null) ≠ Primary key

1.4.2 Super key

The set of attributes which can differentiate records/tuples uniquely or super set of candidate key.

Example 1: R (ABCD) with CK = {A}

$$\begin{aligned}\therefore \text{Super key} &= \text{CK} \cdot [\text{Any subset of other attributes (BCD)}] \\ &= A \cdot [2^3] = 8 \text{ Super key.}\end{aligned}$$

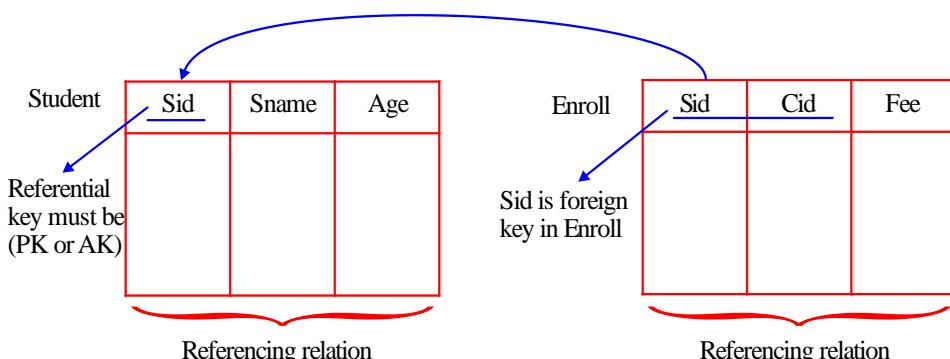
Example 2 : Let R be the relational schema with n-attributes, R (A₁, A₂, A_n) then number of superkeys possible:

- (I) With (A₁) as candidate key : 2ⁿ⁻¹
- (II) With {A₁, A₂} as candidate key : 2ⁿ⁻¹ - 2ⁿ⁻² + 2ⁿ⁻¹
- (III) With {A₁A₂, A₃A₄} as candidate key: 2ⁿ⁻² - 2ⁿ⁻⁴ + 2ⁿ⁻²
- (IV) The maximum number of super keys possible when each attribute of R is candidate key : 2ⁿ⁻¹
- (V) The minimum number of super key possible when all the attributes combined form single candidate key:
1 {A₁A₂ A_n: candidate key}

1.5 Referential Integrity Constraints

1.5.1 Foreign key

Foreign key is a set of attributes that references primary key or alternative key of the same relation or other relation.



Referenced Relation

1. **Insertion :** No violation
2. **Deletion :** [May cause violation]
 - (a) On delete no action : Means if it cause problem on delete then deletion is not allowed on table.
 - (b) On delete cascade : If we want to delete primary key value from referenced table then it will delete that value from referencing table also.
 - (c) On delete set null : If we want to delete primary key value from referenced table then it will try to set the null values in place of that value in referencing table.

Note:

If foreign key field is not null attribute then “On delete set null” is same as “on delete no action.”

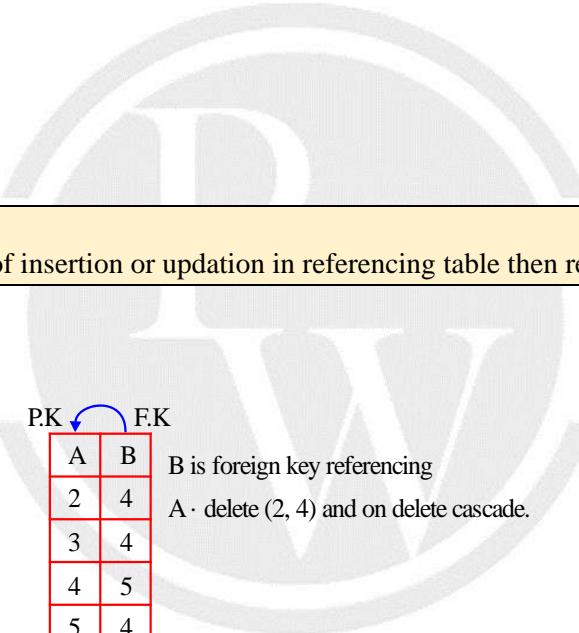
3. **Updation :** [May cause violation]
 - (a) On update no action
 - (b) On update cascade
 - (c) On update set null

Referencing Relation

1. **Insertion :** [May cause violation]
2. **Deletion :** No violation
3. **Updation :** [May cause violation]

Note

If integrity violation occurs because of insertion or updation in referencing table then restrict insertion and updation.

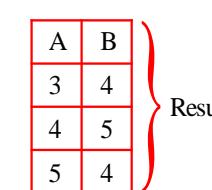
Example:


The diagram shows a 6x2 table with columns labeled A and B. A blue arrow points from the label "P.K." to the first column (A). Another blue arrow points from the label "F.K." to the second column (B). The table contains the following data:

A	B
2	4
3	4
4	5
5	4
6	2

B is foreign key referencing
A · delete (2, 4) and on delete cascade.

So, If we delete (2, 4) then PK “2”. gets deleted from the table and all the tuples in which B is referencing PK.2” also gets deleted.



The diagram shows the result of the deletion. A red brace on the right side of the table is labeled "Result". The table now contains the following data:

A	B
3	4
4	5
5	4

1.6 Database Design Goals

1. 0% redundancy
2. Loss-less join
3. Dependency preservation

1.6.1 Functional Dependency

Let R be the relational schema and x, y be the non-empty set of attributes. Consider t_1, t_2 are any tuples of relation then $x \rightarrow y$ (y functionally determined by x):

If $\forall t_1, t_2 t_1 \cdot x = t_2 \cdot x$ then $t_1 \cdot y = t_2 \cdot y$

1.6.2 Types of Functional Dependency

1. Trivial Functional Dependency

Consider relational schema R(ABCD)

A FD $x \rightarrow y$ is trivial FD only if $x \supseteq y$.

Example :

$$AB \rightarrow A$$

$$A \rightarrow A$$

$$B \rightarrow B$$

2. Non-trivial Functional Dependency

Consider relational schema R(ABCD)

A FD $x \rightarrow y$ is non-trivial only if

$x \cap y = \emptyset$ means no common attributes between x and y attribute sets.

Example :

$$A \rightarrow B$$

$$AB \rightarrow CD$$

3. Semi-Non-Trivial Functional Dependency

A combination of both trivial FD and non-trivial FD.

Example :

$$A \rightarrow AB \equiv \{A \rightarrow A, A \rightarrow B\}$$

1.6.3 Attribute Closure (X^+)

The set of all possible attributes determined by x .

Example :

$$R(ABCD) \{A \rightarrow B, B \rightarrow C, C \rightarrow D\}$$

$$\therefore (A)^+ = \{A, B, C, D\}$$

1.6.4 Armstrong's Axioms

1. **Reflexive :** If $x \supseteq y$ then $x \rightarrow y$ or $x \rightarrow x$
2. **Transitivity :** If $x \rightarrow y$ and $y \rightarrow z$ then $x \rightarrow z$
3. **Augmentation :** If $x \rightarrow y$ then $xz \rightarrow yz$
4. **Splitting :** If $x \rightarrow yz$ then $x \rightarrow y, x \rightarrow z$
5. **Union :** If $x \rightarrow y$ and $x \rightarrow z$ then $x \rightarrow yz$
6. **Pseudo transitivity :** If $x \rightarrow y, yw \rightarrow z$ then $xw \rightarrow z$

1.6.5 Finding Candidate Key [Minimal Super Key]

X is candidate key of R

If and only if

1. x must be super key of relation R
 $(x)^+ = \{\text{determine all attributes of } R\}$
2. No proper subset of 'x' is super key of relation R.
 $\forall y \subset x : (y)^+ = \{\text{not determine all attributes of } R\}$
- $x \rightarrow y$ is a non-trivial FD in R with y is a prime attribute then relation R has atleast two candidate key.

$(x \rightarrow y) : \text{Non - trivial FD}$



Prime-attribute

1.7 Membership of FD

A FD $x \rightarrow y$ is member (logically implied) of FD set F if and only if $(x)^+$ should determine 'y' in FD set F.

Example : Given FD Set F : { }

↓
 $x \rightarrow y$ FD belongs to F or not ?
 ↓
 $x^+ = \{ \dots y \dots \}$
 then $x \rightarrow y$ is member of F.

1.7.1 Testing of Two FD Sets whether they are equal or not

F and G FD sets equality test:

F and G FD sets logically equal if and only

1. **F cover's G :** All FD's of G set must be member's of F set.
 $F \supseteq G$
2. **G cover's F :** All FD's of F set must be member's of G set.
 $F \subseteq G$

F Cover G	G Cover F	Result
True	True	$F \equiv G$
True	False	$F \supset G$
False	True	$F \subset G$
False	False	$F \& G$ are not comparable

1.7.2 Minimal Cover or Canonical Cover

- Minimal Cover of given FD Set (F) is minimum possible FD's (Fm), which is logically equal to 'F' : ($F = F_m$)
- Remove extraneous attribute (useless attribute) from each determinant of FD set F.

$wxy \rightarrow z$
 Extraneous
 attribute $\{wxy \rightarrow z, w \rightarrow x\} \equiv \{wy \rightarrow z, w \rightarrow x\}$

- Every FD must be simple (RHS of any FD should have single attribute).

Example : $F = \{A \rightarrow CD\}$ then $\{A \rightarrow C, A \rightarrow D\}$

- FD set must be non-redundant FD. (eliminating unnecessary FD's)

Example : $F = \{A \rightarrow B, B \rightarrow C, A \rightarrow C\}$ then

$F = \{A \rightarrow B, B \rightarrow C\}$ because $A \rightarrow C$ is redundant.

Note:

Minimal Cover of FD Set (F) may not unique, but all minimal cover's logically equal.

$$\therefore F_{m1} \equiv F_{m2} = F$$

1.8 Normalization

Normalization used to eliminate/reduce redundancy in DB relations.

The normalization is especially meant to eliminate the following anomalies:

- Insertion anomaly
- Deletion anomaly
- Update anomaly
- Join anomaly

1.8.1 Normalization of DB table

Decompose relation into two or more sub-relations in-order to reduce or eliminate redundancy and DB anomalies.

1.8.2 Properties of Decomposition

- Loss-less Join decomposition:** Relational Schema R decomposed into $R_1, R_2, R_3, \dots, R_n$ sub-relations.
 - In general $[R_1 \bowtie R_2 \bowtie R_3 \dots, R_n] \supseteq R$
- If $[R_1 \bowtie R_2 \bowtie R_3 \dots, R_n] \equiv R$ then loss-less join decomposition.
 - If $[R_1 \bowtie R_2 \bowtie R_3 \dots, R_n] \supset R$ then lossy join decomposition.

Example :

Let R be the relational schema decomposed into R_1 and R_2 .

Given decomposition is loss-less join if–

- $R_1 \cup R_2 = R$ (all attributes covers)
- $R_1 \cap R_2 \neq \emptyset$
- $\underbrace{R_1 \cap R_2 \rightarrow R_1}_{R_1 \cap R_2 \text{ is SK of } R_1} \text{ or } \underbrace{R_1 \cap R_2 \rightarrow R_2}_{R_1 \cap R_2 \text{ is SK of } R_2}$

1.8.3 Dependency Preserving Decomposition

Relational Schema R with FD set F decomposed into R_1, R_2, \dots, R_n sub relations

Assume F_1, F_2, \dots, F_n FD sets of sub relations respectively.

In general

$$[F_1 \cup F_2 \cup \dots, F_n] \subseteq F$$

- If $[F_1 \cup F_2 \cup \dots, F_n] \equiv F$ then dependency preserving decomposition.
- If $[F_1 \cup F_2 \cup \dots, F_n] \subset F$ then not dependency preserving decomposition.

1.9 Normal Forms

Used to find degree of redundancy

Redundancy in relation because of

Non-Trivial FD
 $x \rightarrow y$
 (Single value dependancy)

Non-Trivial MVD
 $x \rightarrow\!\!> y$
 (Multivalued dependancy)

- BCNF relations have 0% redundancy over FD's whereas 4NF relations have 0% redundancy over FD and MVD.
- To Eliminate redundancy over Non-Trivial FD, relation should decompose into BCNF but BCNF may not avoid the redundancy due to MVD.
- To eliminate redundancy over non-trivial MVD, relation should decompose into 4NF (4th Normal Form).

1.9.1 First Normal Form

- Default NF of RDBMS relations.
- Relation (DB Table) R is in 1 NF if and only if no multivalued attributes in R. [Every attribute of R must be atomic/single valued].

Note:

Degree of redundancy is very high in 1NF relation.

Important Point 1

- $x \rightarrow y$ FD forms redundancy in relational schema R if and only if –
 - Non-Trivial FD
and
 - X is not super key.
- $x \rightarrow y$ FD not forms any redundancy in relation R if and only if –
 - Trivial FD [$x \supseteq y$] or Semi-trivial
or
 - x : super key

1.9.2 Second Normal Form (2NF)

Relational Schema R is in 2 NF iff

- R should be 1NF
- R should not contain any partial dependency

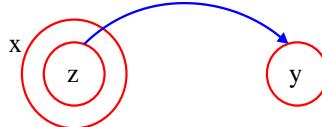
Partial Dependency

Let R be the relational schema and x, y, z are non-empty set of attributes.

X : Any candidate key of R.

Y : Non-prime attribute

Z : Proper subset of candidate key



$z \rightarrow y$ is said to be partial dependency iff–

- z is proper subset of candidate key
- y should be non-prime attribute.

Important Point 2

$$\underbrace{[\text{Proper subset of Candidate key}]}_{\text{Not super key}} \rightarrow [\text{Non-Prime attribute}]$$

The above FD forms redundancy in R.

1.9.3 Third Normal Form (3NF)

Relational schema R is in 3NF iff every non-trivial FD $x \rightarrow y$ in R with–

- (i) x must be super key (SK)
or
- (ii) y must be prime attribute.
 $\therefore \{SK \rightarrow \text{Prime/Non-Prime}, (\text{Not SK}) \rightarrow \text{Prime attribute}\}$

Important Point 3

3NF allow FD set:

$[\text{Proper subset of candidate key}] \rightarrow [\text{Proper subset of other candidate key}]$
which forms redundancy in R.

1.9.4 Boyce codd Normal Form (BCNF)

Relational schema R is in BCNF iff every non-trivial FD “ $x \rightarrow y$ ” with x must be super key.

\therefore Prime/ Non-prime attributes must be determine by super key.

Important Point 4

If R is in 3NF but not BCNF then $[\text{Proper subset of candidate key}] \rightarrow [\text{Proper subset of other candidate key}]$ must exists in R.

Important Point 5

If relational shcema are with only simple candidate key then R always in:

- I. 2NF
- II. May or may not 3NF/BCNF

Reason : $[\text{Proper subset of candidate key}] \rightarrow [\text{Non-prime attribute}]$

From the above statement, we can conclude that “partial dependency” not possible if all CK’s are simple candidate key.

Important Point 6

If relational schema R with only prime attribute (No non-prime attribute in R) then R always in:

- I. 3NF
- II. May or may not BCNF

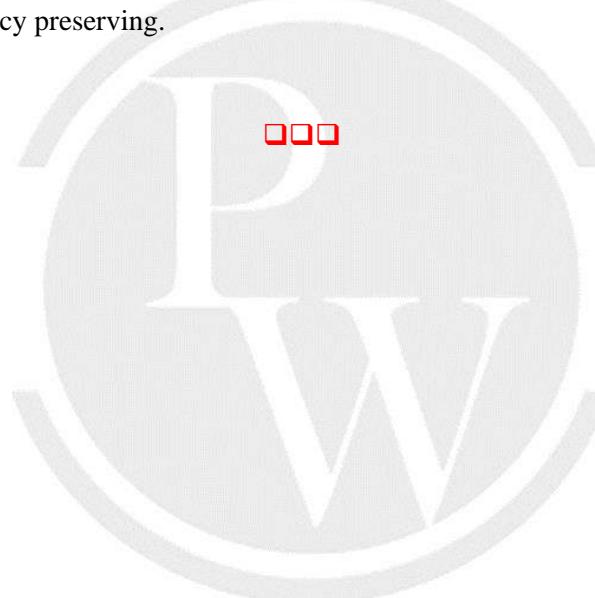
Important Point 7

If relational schema R with no non-trivial FD's then R always in BCNF.

1.9.5 Database design Table

DB design goal	1NF	2NF	3NF	BCNF
1. Loss-less join decomposition	Yes	Yes	Yes	Yes
2. Dependency preserving decomposition	Yes	Yes	Yes	May not
3. 0% redundancy	No	No	No	Yes [Over FD]

- Every relation possible to decompose into 1NF, 2NF, 3NF, BCNF with loss-less join decomposition.
- Every relation possible to decompose into 1NF, 2NF, 3NF with Dependency preserving decomposition.
- Not every relation can decompose into BCNF and 4NF with dependency preserving decomposition.
- Most accurate normal form to design simple database is 3NF because every relation is possible to decompose into 3NF with loss-less join and dependency preserving.



2

ER MODEL

2.1 Introduction

Entity relationship diagram used to represent diagrametic design (High level design) of DB.

2.2 Main components in ER Diagram

- (i) Attributes
- (ii) Entity sets
- (iii) Relationship sets

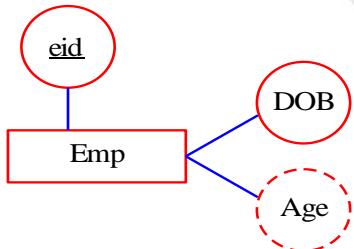
(a) Entity sets

It is a set of entities of the same type denoted by a rectangular box in ER diagram. Entity can be identified by a list of attributes which are placed in ovals.

Represent by :



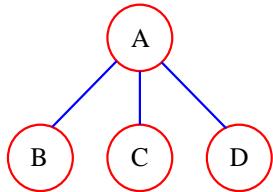
Example :



(b) Attributes

- (i) Attribute : 
- (ii) Key attribute : 
- (iii) Derived attribute : 

(iv) Composite attribute: Attribute which can be represented as two or more attributes.

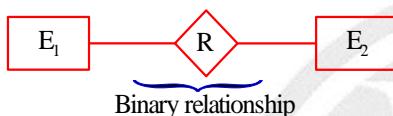


(v) Multivalued attribute:

(c) Relationship set: It is used to relate two or more entity set.

Represented by :

Example:



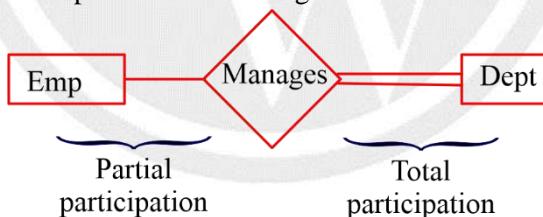
2.3 Participation

- If every entities of entity set are participated with relationship set then it is total participation (100% participation) otherwise it will be partial participation (< 100% participation)

Example :

Consider Emp and Dept entity set.

Manages relationship set such that each dept must have manager.

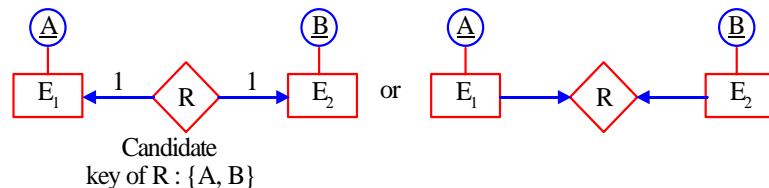


2.4 Mapping [Cardinality constraints of relationship set]

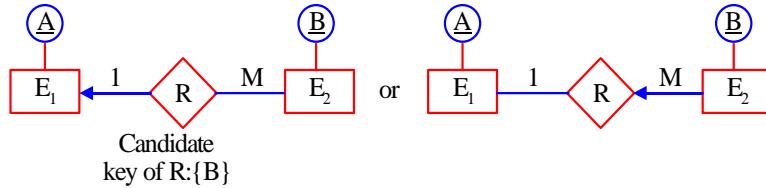
One mapping : Atmost one (0 or 1)

Many mapping : 0 or more (0 *)

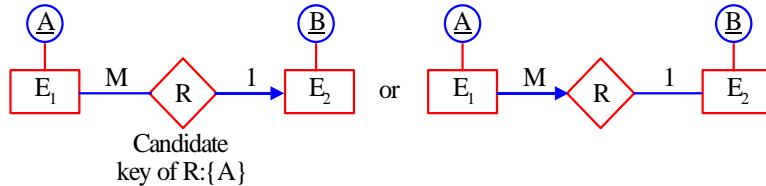
1. Binary Relationship Mapping (One : One)



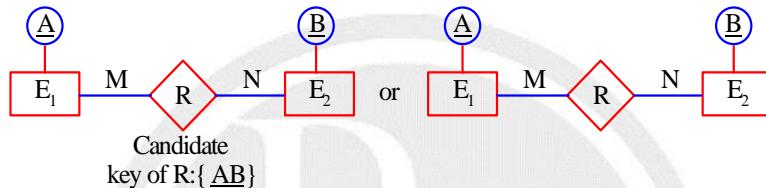
2. Binary Relationship Mapping (One : Many)



3. Binary Relationship Mapping (Many to One)



4. Binary Relationship Mapping (Many to Many)



2.5 RDBMS Design of Given ER Diagram

(For binary relationship)

1. Partial participation on both side of binary relationship

- One to Many :** Merge relationship set towards many side. So, 2 relational tables.
- Many to one :** Merge relationship set towards many side. So, 2 relational tables.
- One to one :** Merge relationship set any one side. So, 2 relational tables.
- Many to Many :** Separate table for each entity set and relationship set. so, 3 relational tables.

2. Full participation on “one” side of many to one relationship

Merge the entities and relationship set into single relational table. So, 1 table.

3. Full participation on “Many” side of Many-to-one relationship

Merge relationship set towards many side. So, 2 relational tables.

4. Full participation on any “one” side in one-to-one relationship

Merge the entity sets and relationship set into single table. So, 1 table.

5. Full participation on any “Many” side of Many-to-Many relationship

Merge relationship set towards any “Many” side of relationship. So, 2 table.

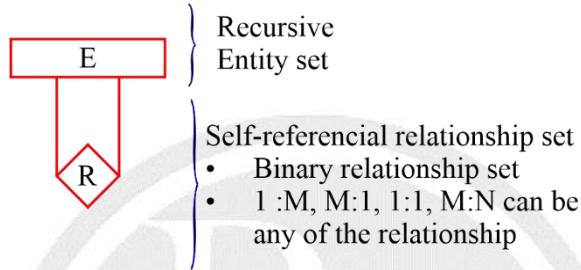
6. Full participation on both side of relationship

- 1:1
 - 1:M
 - M:1
 - M:N
- Merge the entity sets and relationship set into single relational table so, 1 relational table

2.6 Self-Referencial Relationship Set

(Recursive entity set)

Entities of entity set (E) related to some other entity of same entity set (E).



2.7 Weak Entity Set and Weak Relationship Set

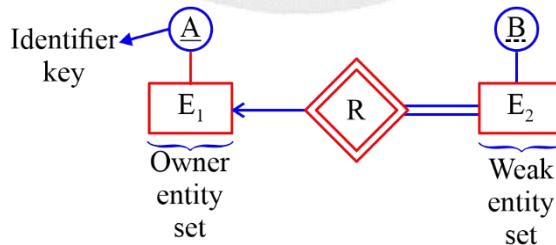
The entity set with no key. (Attributes of weak entity sets are not sufficient to differentiate entities uniquely).



Points:

- For each weak entity set there must be owner entity set, which is strong entity set.
- Relationship set between weak entity set and identifier entity set is also “weak relationship set”.
- The participation towards weak entity set end must be “total participation”.
- The mapping between identifier entity set and weak entity set must be one : many (1 : M)

Example:



Note:

Weak entity set and multivalued attributes allowed to represent in ER diagram but not allowed in RDBMS table.



3

RELATION MODEL AND SQL

3.1 Procedural Query Language and Non-procedural Query Language

Procedural Query Language	Non-procedural Query Language
Formulation of how to access data from the database table and what data required to retrieve from DB tables. “Relational Algebra”	Formulation of what data retrieve from DB tables. “Relational Calculus”

3.2 Relational Algebra

(Always generate distinct tuples)

Relational algebra refers to a procedural query language that takes relation instances as input and returns relation instances as output.

3.2.1 Basic operators

π : Projection operator

σ : Selection operator

\times : Cross-product operator

U : Union

$-$: Set difference

ϱ : Rename operator

3.2.2 Derived operators

\cap : Intersection {using “_”}

\bowtie : Join {using X, σ }

/ or \div : Division {using π , x, $-$ }

I. π : Projection

- $\pi_{\text{attribute_name}}(R)$: It is used to project required attribute from relation R.
- $\sigma_{\text{Condition}(P)}(R)$: It is used to select records from relation R, those satisfied the condition (P).

Example:

R	A	B	C	$\pi_{B,C}(R)$:	B	C	
	8	4	5		4	5	
	2	4	5		4	6	
	7	4	6		5	5	
	3	5	5	$\sigma_{A>S}(R)$:	A	B	C
					8	4	5
					7	4	6

II. Cross product (\times):

- $R \times S$: It result all attributes of R followed by all attributes of S, and each record of R paired with every record of S.
- Degree ($R \times S$) = Degree (R) + Degree (S)
- $|R \times S| = |R| \times |S|$

Note:

- Relation R with n tuples and
- Relation S with 0 tuples then
number of tuples in $R \times S$ = 0 tuples

3.3 Join (\bowtie)

I. Natural join (\bowtie)

$$R \bowtie S \equiv \pi_{\text{distinct attributes}}(\sigma_{\text{equality between common attributes of } R \text{ and } S} (R \times S))$$

Example:

- T_1 (ABC) and T_2 (BCDE)
- $\therefore T_1 \bowtie T_2 = \pi_{ABCDE} \left(\begin{array}{l} \sigma_{T_1.B=T_2.B}(T_1 \times T_2) \\ \cap T_1.C=T_2.C \end{array} \right)$
- T_1 (AB) and T_2 (CD)
- $\therefore T_1 \bowtie T_2 \equiv T_1 \times T_2 = \pi_{ABCD}(T_1 \times T_2)$

Note:

Natural join equal to cross-product if join condition is empty.

II. Conditional Join (\bowtie_c)

- $R \bowtie_c S \equiv \sigma_c(R \times S)$

III Outer Joins:

(a) **LEFT OUTER JOIN**

$R \bowtie S$: It produces

$(R \bowtie S) \cup \{\text{Records of } R \text{ those are failed join condition with remaining attributes null}\}$

(b) **RIGHT OUTER JOIN (\bowtie^R)**

$R \bowtie^R S$: It produces]

$(R \bowtie S) \cup \{\text{Records of } S \text{ those are failed join condition with remaining attributes null}\}$

(C) FULL OUTER JOIN (\bowtie)

$$R \bowtie S = (R \bowtie S) \cup (R \bowtie\bowtie S)$$

3.4 Rename operator (ϱ)

- It is used to rename table name and attribute names for query processing.

Example:

(I) Stud (Sid, Sname, age)

$\varrho(\text{Temp}, \text{Stud}) : \text{Temp} (\text{Sid}, \text{Sname}, \text{age})$

(II) $\varrho_{I,N,A} (\text{Stud}) : \text{Stud} (I, N, A)$

All attributes renaming

(III) $\varrho_{\substack{\text{Sid} \rightarrow I \\ \text{age} \rightarrow A}} (\text{Stud}) : \text{Stud} (I, \text{Sname}, A)$

$\text{age} \rightarrow A$

Some attribute renaming

Example:

Retrieve eids of female employees whose salary more than every male employee?

Result :

$$\pi_{eid} \left(\sigma_{(gen=Female)}^{(Emp)} \right) - \pi_{eid} \left(E \bowtie \delta_{I,S,G} (emp) \right)$$

$$\left(\begin{array}{l} gen = female \\ \cap \\ G = male \\ \cap \\ Sal \leq S \end{array} \right)$$

3.5 Division

- It is used to retrieve attribute value of R which has paired with every attribute value of other relation S.
- $\pi_{AB}(R)/\pi_B(S)$: It will retrieves values of attribute ‘A’ from R for which there must be pairing ‘B’ value for every ‘B’ of S.

Expansion of ‘/’ by using basic operator

- Example: Retrieve sid's who enrolled every course.

Result:

$$\pi_{sidcid}(\text{Enroll}) / \pi_{cid}(\text{Course})$$

Step 1: Sid's not enrolled every course of course relation.

(Sid's enrolled proper subset of course)

$$\pi_{sid}((\pi_{sid}(\text{Enroll}) \times \pi_{cid}(\text{course})) - \pi_{sidcid}(\text{Enroll}))$$

Step 2:

$[\text{sid's enrolled every course}] = [\text{sid's enrolled some course}] - [\text{sid's not enrolled every course}]$

$$\therefore \pi_{\text{sidcid}}(E) / \pi_{\text{cid}}(C) = \pi_{\text{sid}}(E) - \pi_{\text{sid}}(\pi_{\text{sid}}(E) \times \pi_{\text{cid}}(C) - \pi_{\text{sidcid}}(E))$$

3.6 Set operator

U: Union operator

- : Except minus

\cap : Intersection operator

- To apply set operations relations must be union compatible.
- R and S relations union compatible
- If and only if-
 - (i) Arity of R equal to Arity of S and
 - (ii) Domain of attributes of R must be same as domain of attributes of S respectively.

Example 1:

$$\pi_{\text{Sid Sname}}(\dots) \cap \pi_{\text{Sid}}(\dots)$$

{Arity not same so, set operation not allowed}

Example 2:

$$\pi_{\text{Sid S name}}(\dots) \cap \pi_{\text{Sid marks}}(\dots)$$

{Arity same but Sname domain is different from marks so, not allowed}

Example 3:

$$\pi_{\text{Sid Sname}}(\dots) \cap \pi_{\text{Stud ID, Stud name}}(\dots)$$

{Arity and domains are same so, allowed for set operation}

- 1. Set operation on relation:

R	A
2	
2	
2	
3	

S	B
2	
2	
2	
4	

$$R \cup S : \{x / x \in R\} \vee x \in S \equiv \begin{array}{|c|} \hline A \\ \hline 2 \\ \hline 3 \\ \hline 4 \\ \hline \end{array}$$

$$R - S : \{x / x \in R \wedge x \notin S\} \equiv \begin{array}{|c|} \hline A \\ \hline 3 \\ \hline \end{array}$$

$$R \cap S : \{x / x \in R \wedge x \in S\} \equiv \begin{array}{|c|} \hline A \\ \hline 2 \\ \hline \end{array}$$

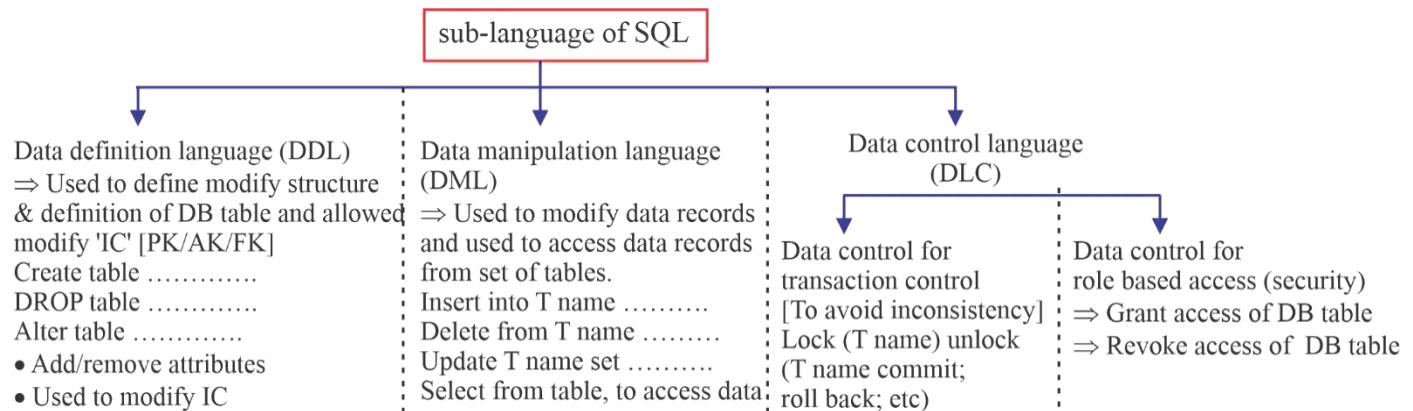
3.7 Cardinality Table

Assume relation R with n Distinct tuples and relation S with m distinct tuples:

RA Expression	Cardinality
$R \times S$	$n * m$ tuples
$R \bowtie S$	{0 to $n * m$ tuples}
$R =\bowtie S$	{n to $n * m$ tuples}
$R \bowtie= S$	{m to $n * m$ tuples}
$R =\bowtie= S$	{max (n,m) to $n * m$ tuples}
$R \cup S$	{max(n,m) to $n + m$ }
$R \cap S$	{0 to min (m, n)}
$R - S$	{0 to n tuples}
R/S	{0 to [n/m] tuples}

3.8 SQL

3.8.1 [Structure Query Lang]



3.8.2 Relational Query and SQL query

- SQL : SELECT DISTINCT A₁, A₂, A_n

FROM R₁, R₂, R_n
WHERE P;

Equivalent RA : $\pi_{A_1, A_2, \dots, A_n} (\sigma_p (R_1 \times R_2 \times \dots \times R_n))$

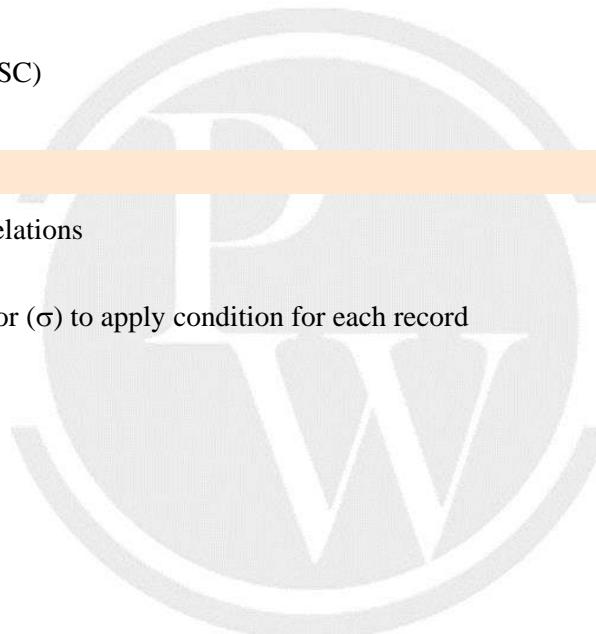
- SELECT ≡ projection of RA (π)
FROM ≡ Cross-product (x)
WHERE ≡ Selection operator of RA (σ)

3.8.3 Basic SQL Clauses

SELECT DISTINCT A₁, A₂, A_n
FROM R₁, R₂, R_n
WHERE condition
GROUP BY (attributes)
HAVING condition
ORDER BY (attributes) (DESC)

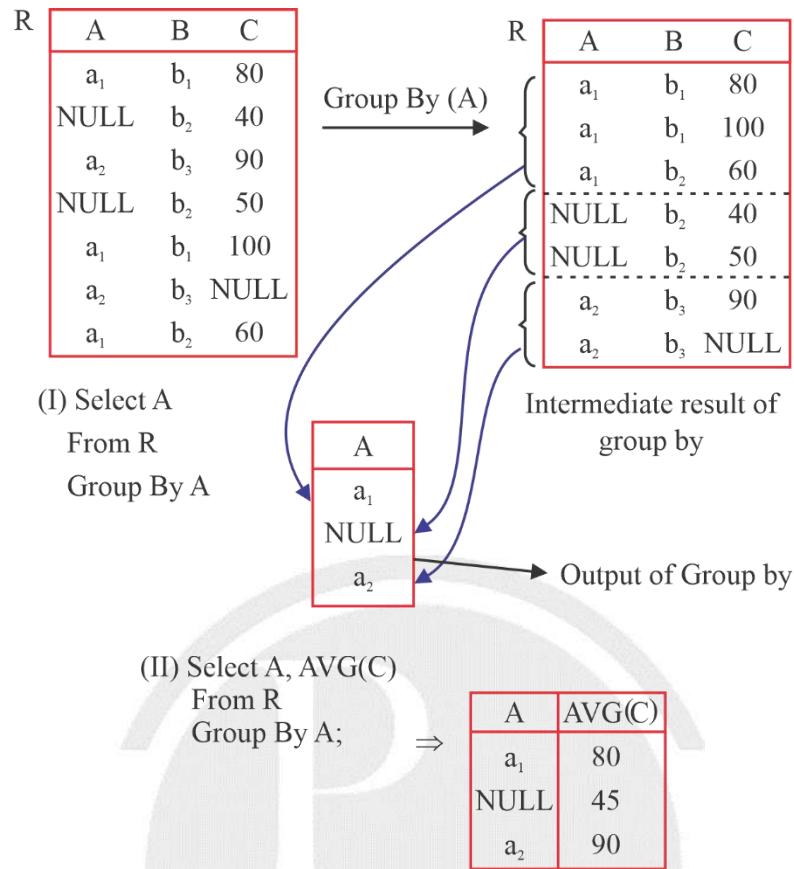
3.8.4 Execution Flow

FROM ⇒ cross-product of relations
↓
WHERE ⇒ Selection operator (σ) to apply condition for each record
↓
GROUP BY
↓
HAVING
↓
SELECT
↓
DISTINCT
↓
ORDER BY



3.8.5 GROUP BY:

- It is used to group records data based on specific attribute.
 - If GROUP BY clause used then
 - (a) Every attribute of GROUP BY clause must be selected in SELECT clause.
 - (b) Not allowed to select any other attribute in SELECT clause.
- Allowed to select aggregate function along with GROUP BY attribute in SELECT Clause.



3.8.6 HAVING Clause

- HAVING clause must be followed by GROUP BY clause.
- HAVING clause used to select groups those are satisfied having clause condition.
- HAVING clause condition must be over aggregation function such as some (), Every () etc. but not allowed direct attribute comparison

Example:

1.

```
SELECT A
  FROM R
 GROUP BY A
 HAVING AVG (c) > 60;
```
2.

```
SELECT A, AVG (c)
  FROM R
 GROUP BY A
 HAVING some (c) > 50;
```
3.

```
SELECT A
  FROM R
 GROUP BY A
 HAVING C> 60;
```

 } Error

↓
Not allowed

Note:

WHERE clause condition tested for each record but HAVING clause condition tested for each GROUP.

3.8.7 WITH Clause

- WITH clause is used to create sub-query result that can be re-used many times in query processing.

Example:

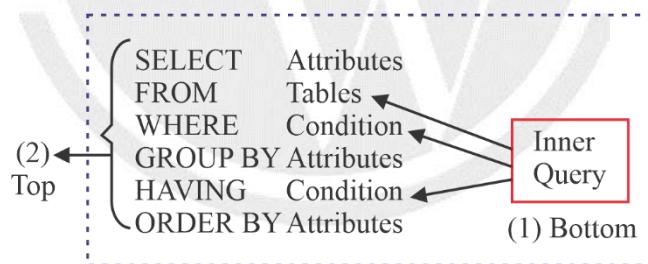
```
WITH Temp (A1, A2) as
  (SELECT T1 . A1, T2 . A2
   FROM Emp T1, Emp T2
   WHERE T1 . A1 < T2 . A1)
```

Note:

Every Query which is written by using HAVING clause can be re-written by using WHERE clause.

3.8.8 Nested Query Without Co-relations

- Inner query independent of outer query.



- Execution flow must be Bottom to Top mean first bottom (Inner query) evaluated then outer query executed.

3.8.9 Co-related Nested Query

- In Nested Co-related query inner query uses attributes from outer query tables.
- In Co-related Nested query inner query allowed in WHERE, HAVING clause of outer query.

Example: `SELECT A`

```
  FROM R
  WHERE (SELECT count(*)
        FROM S
        WHERE S.B < R.A) < 5;
```

Important point 1

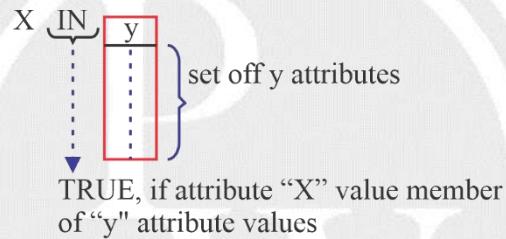
- If co-relation in WHERE clause then inner query re-computes for each record of outer query From clause.
- If correlation in HAVING clause then inner query re-computes for each group of outer query.

3.9 Function used for nested Queries

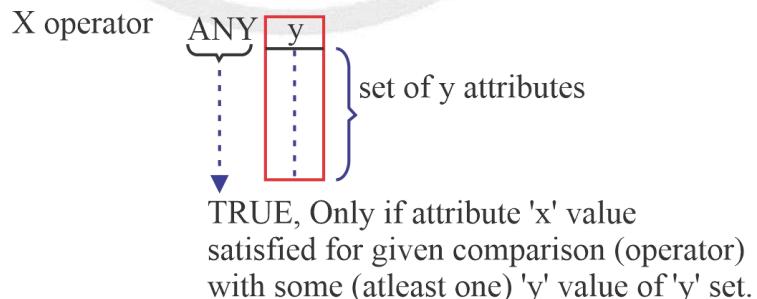
- | | |
|----------------------|-------------------------|
| 1. IN/ NOT IN | } Best suitable for |
| 2. ANY | |
| 3. ALL | } non- co-related query |
| 4. EXISTS/NOT EXISTS | |
| | } Best suitable for |
| | co-related query |

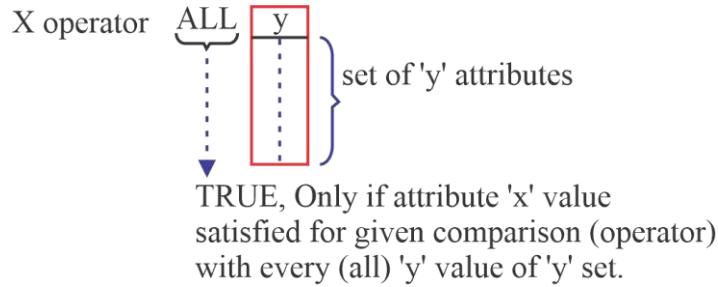
1. IN Function

It is used for membership testing.

**Note:**

Queries of "equi Join" can implement by IN function $\equiv (R \bowtie S)$

2. ANY Function (operator "<, ≤, >, ≥, =, <>")**3. ALL Function**

**Note:**

ANY function can be used as queries of conditional join query.

Example:

$$\pi_A \left(R \bowtie S \atop R.A > S.B \right) \equiv \text{SELECT } A \text{ FROM } R \text{ WHERE } R.A > \text{ANY} (\text{SELECT } B \text{ FROM } S)$$

$$\pi_A(R) - \pi_A \left(R \bowtie S \atop R.A \leq S.B \right) \equiv \text{SELECT } A \text{ FROM } R \text{ WHERE } R.A > \text{ALL} (\text{SELECT } B \text{ FROM } S)$$

Important point 2

- IN function equal to ‘= ANY’ function

$$X \text{ IN } \boxed{Y} \equiv X = \text{ANY } \boxed{Y}$$

- IN function not equal to ‘= ANY’ if more than one attribute used for IN comparison.

$$(A,B) \text{ IN } \boxed{C D} \neq (A,B) = \underbrace{\text{ANY}}_{\text{Not allowed}} \boxed{C D}$$

- NOT IN function equal to “<> ALL”.

$$X \text{ NOT IN } \boxed{Y} \equiv X <> \text{ALL } \boxed{Y}$$

4. EXISTS clause

- It is used to test result of inner query is empty or not empty.
-

EXISTS (Query)

TRUE, if result of inner query is non-empty that is atleast one record in result of inner query.

Example:

```
SELECT R.A
FROM R
WHERE EXISTS (SELECT *
               FROM S
               WHERE R.A > S.B);
Atleast one record of S relation satisfy R.A > S.B
```

Note:

EXISTS function can use as conditional join queries with join condition in inner query WHERE clause.

Emp (eid sal dno. Gen) Retrieve eids of femal's whose salary more than (some) male emp.

$$\text{RA: } \pi_{\text{eid}} \sigma_{\text{gen} = \text{Female}}(\text{Emp}) \bowtie \rho_{I,S,D,G} (\sigma_{\text{gen}=male}(\text{emp})) \\ \text{Sal} > S$$

SQL
[JOIN Query]

```
SELECT DISTINCT T1 . eid
FROM EMP T1, Emp T2
WHERE
T1.gen = Female and
T2.gen = male and
T1.sal > T2 . sal
```

SQL
[Mested Query]

```
SELECT Eid
FROM Emp
WHERE gen = Female
and sal > ANY (SELECT sal
               FROM Emp
               WHERE gen = male);
```

SQL
[Co – related Nested Query]



```
SELECT Eid  
FROM Emp T1  
WHERE T1.gen = Female and  
EXISTS (SELECT *  
        FROM Emp T2  
        WHERE T2.gen = male and T1.Sal > T2.sal);
```

□□□



4

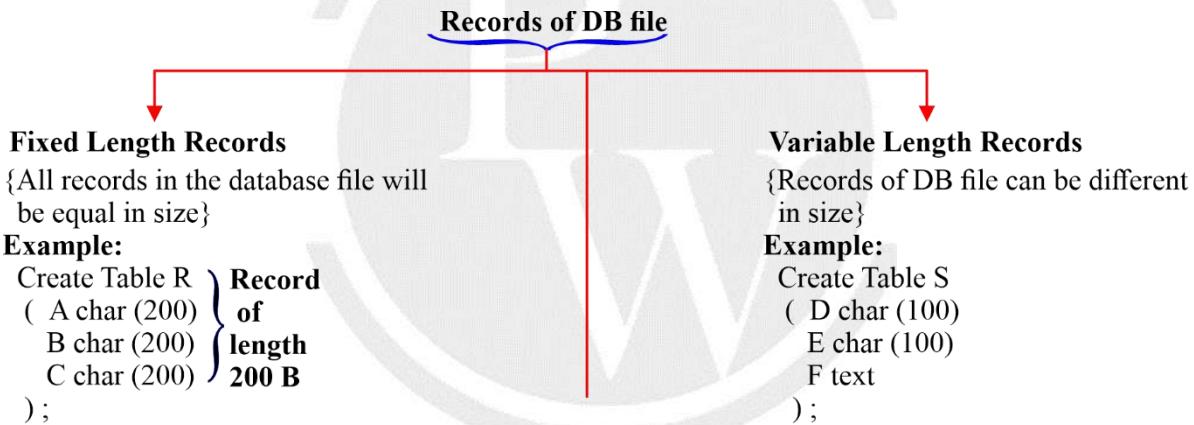
FILE ORGANIZATION AND INDEXING [B AND B⁺ TREE]

4.1 File Organization

- Database is collection of files [Tables].
- File is collection of blocks [Pages].
- Block is collection of records.

Note:

Data access from disk (DB file) to main memory block by block.



4.2 Records organization of DB file

Spanned Organization	Unspanned Organization
1. Record allowed to span in more than one block.	1. Complete record must be in one block.
2. Advantage: Possible to allocate file without any internal fragmentation.	2. Advantage: less access cost and easy to organize DB file.
3. Disadvantage: More access cost to access spanned records.	3. Disadvantage: May not possible to allocate DB file without any internal fragmentation.

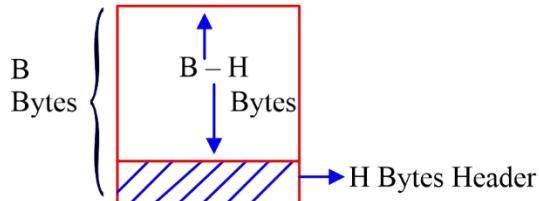
Note:

1. Spanned organization prefer to store database record of the file with variable length record.

2. Unspanned organization prefer to store DB record of file with fixed length record.

4.3 Block factor [BF]

The maximum possible records which can store in block (maximum record per block).



Consider record size = R bytes

$$1. \therefore \text{Block Factor} = \left\lfloor \frac{\text{Block size} - \text{Header size}}{\text{Record size}} \right\rfloor$$

For unspanned organization

$$2. \therefore \text{Block Factor} = \left\lfloor \frac{\text{Block size} - \text{Header size}}{\text{Record size}} \right\rfloor$$

For spanned organization

4.4 Categories of Index

4.4.1 Dense Index

For each DB record of DB file there must be index entry in index file.

4.4.2 Sparse Index

For set of DB records (blocks) of DB file there exists index entry in index file.

Important Point I

- Sparse index possible only over ordered field of DB file.
- In Sparse indexing

$$\begin{bmatrix} \text{No. of index} \\ \text{file entries} \end{bmatrix} < \begin{bmatrix} \text{No. of records} \\ \text{of DB file} \end{bmatrix}$$

4.4.3 Block Factor of Index

Assume Block size = B Bytes

Header size = H Bytes

Search key = K Bytes

Pointer size = P Bytes

$$\therefore \text{Block Factor of index} = \left\lfloor \frac{B - H}{K + P} \right\rfloor$$

Note:

By default header size will be 0 bytes, if not given in problem.

4.5 I/O Cost [Access cost]

The number of secondary memory block (DB file block) required to transfer from disk to main memory in order to access required record.

1. I/O cost to access data record without index from DB file of n blocks

- (a) Over ordered field:

$$\text{Access cost} = \lceil \log_2 n \rceil \text{ block}$$

- (b) Over unordered field:

$$\text{Access cost} = n \text{ blocks}$$

2. If Dense Index Used:

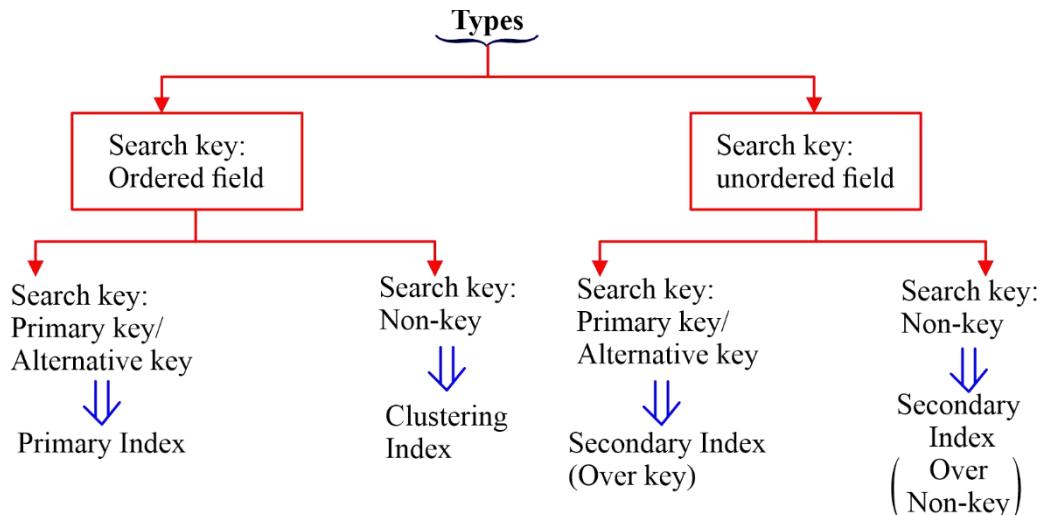
- Number of Dense Index block = $\left\lceil \frac{\text{number of records}}{\text{Block Factor of Index}} \right\rceil$
- Access cost (Number of Block access)
 $= \left\lceil \log_2 (\text{number of Dense index blocks at 1}^{\text{st}} \text{ level}) \right\rceil + 1$

3. If sparse Index Used:

- Number of DB blocks = $\left\lceil \frac{\text{number of records}}{\text{BF of DB}} \right\rceil$
- Number of sparse Index block (At 1st level)
 $= \left\lceil \frac{\text{number of DB blocks}}{\text{BF of Index}} \right\rceil$
- Access cost (Number of block access)
 $= \left\lceil \log_2 (\text{number of sparse index blocks at 1}^{\text{st}} \text{ level}) \right\rceil + 1$

4.6 Types of Index

- Search key: Fields of DB file which is used for indexing



4.6.1 Primary Index

Search Key: Ordered field and key (candidate key)

- I/O cost to access record using primary index with multilevel indexing is $(k + 1)$ blocks.
Where k is level of indexing.
- For any database relation at most one primary index is possible because, search key must be ordered field.
- Primary index can be either dense or sparse but sparse primary index preferred.

4.6.2 Clustered Index

Search Key: Ordered field and non-key.

- I/O cost:

$$\text{Single level index blocks : } \left\lceil \frac{\text{No.of distinct values over non-key}}{\text{BFof Index}} \right\rceil$$

$$\therefore \text{Access cost} = \lceil \log_2(\text{single level index blocks}) \rceil + 1$$

- I/O cost to access cluster of record using clustered index with multilevel index is:
 $k + \underbrace{(\text{one or more block of DB})}_{\text{Until next cluster begins}}$
- For any DB relation at most one clustering index is possible because search key is ordered field.

Note:

For any DB relation either primary index or clustering index is possible but not both simultaneously.

4.6.3 Secondary Index [over key]

Search Key : Unordered field and key.

- Secondary index is alternative index to access data records even primary or clustering index already exists.
- Secondary index over key is always dense index.
- I/O cost to access record using secondary index over key with multilevel index: $(k + 1)$ blocks.

4.6.4 Secondary Index [over non-key]

Search Key: Unordered field and non-key.

- I/O cost to access record using secondary index over non-key with multilevel index = $[k \text{ blocks from } k \text{ level of index}] + [\text{some more DB blocks}]$

Note:

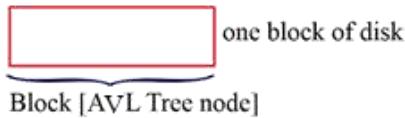
Many secondary index's possible for DB file.

4.7 Balanced Search Tree [Height Restricted Search Tree]

- The maximum height of search tree for n distinct keys should not exceed $O(\log n)$
- Worst case search cost to search element is : $O(\log n)$

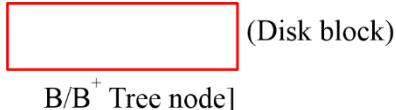
4.7.1 Why B/B+ Tree Index preferred rather than balanced BST/AVL for DB index

- (a) **AVL:** If AVL tree (balanced BST) used for DB index:



- Disadvantage: Each index block with only one key so the number of levels of index is high. (I/O cost to access data is also very high).
- Wastage of disc space, which is allocated for index (only one key stored per block)

(b) B/B⁺ Tree:

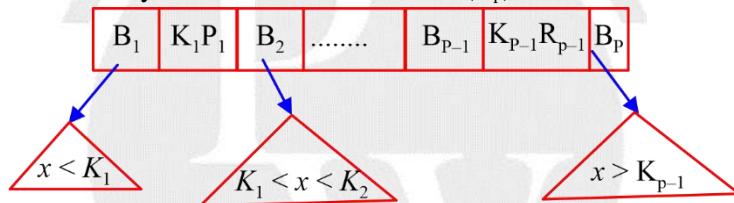


- The access cost (I/O cost) is less because many keys store in one node (number of levels of index will be less).
- The wastage of disk space is less.

4.7.2 B Tree Definition

- Order P (degree): The maximum possible child pointers can be stored in B Tree node.

Node structure: P child pointer, P – 1 Key and P – 1 Record Pointer (R_p).



\therefore order of node \Rightarrow

$$P \times (\text{size of block pointer}) + (P - 1) (\text{size of key field} + \text{size of record pointer}) \leq \text{Block size}$$

1. Every internal node except root must be atleast $\left\lceil \frac{p}{2} \right\rceil$ block pointer and $\left\lceil \frac{p}{2} \right\rceil - 1$ keys and at most P block pointer and $(P - 1)$ keys.
2. Root can have at least 2 child/block pointer and 1 keys and at most P block pointer and $(p - 1)$ keys.
3. Every leaf node must be at same level.

- **Advantages:**

B Tree index best suitable for random access of any one record.

Example:

```
SELECT *
FROM R
WHERE A = 15;
```

\therefore Worst case access cost = $(k + 1)$ blocks

Best case access cost = 2 blocks

- **Disadvantages:**

B Tree index not suitable for sequential access of range of records.

4.7.3 B⁺ Tree

Order P: The maximum possible pointers can be stored in B⁺ tree node.

1. Node Structure:

- **Leaf node:** (set of (key, R_P) pairs and one block pointer)

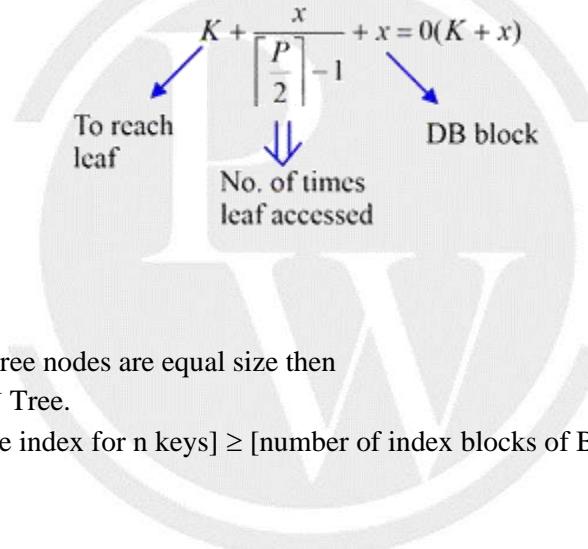
K ₁ R ₁	K ₂ R ₂	K _{P-1} R _{P-1}	B _P	→ Block Pointer {pointer to next leaf}
-------------------------------	-------------------------------	-------	-----------------------------------	----------------	--

∴ Order of leaf node = (P - 1) [K + R_P] + 1 * B_P ≤ Block size.

- **Internal Node:**

Order of internal node = P * B_P + (P - 1) * K ≤ Block size.

- B⁺ Tree best suitable for random access of any one record and sequential access of range of records.
- I/O cost to access range of 'x' records sequential:



Important point 2

If disk block allocation for B and B⁺ tree nodes are equal size then

1. Order P of B tree < order P of B⁺ Tree.
2. [number of index blocks of B tree index for n keys] ≥ [number of index blocks of B⁺ Tree nodes for n keys]
3. I/O cost ≥ I/O cost

Important point 3

If order P of B Tree is equal to order P of B⁺ Tree then

1. [number of B Tree node n distinct keys] ≤ [number of B⁺ Tree nodes for n distinct keys]
2. [B Tree level for n keys] ≤ [B⁺ Tree levels for n keys]



5

TRANSACTIONS AND CONCURRENCY CONTROL

5.1 Transaction

A set of logically related operation to perform unit of work.

5.2 Degree of concurrency

The number of users (Transactions) using data bases simultaneously (concurrently).

5.2.1 Flat File System

In flat file system 1 file is a resource so, only one user allowed at a time.

5.2.2 DBMS File System

- In DBMS file system 1 record is a resource so, it allows as many users as the number of records.
- More degree of concurrency.

5.3 Main Operations in Transactions

5.3.1 Read (A)

Access the data item ‘A’ from DB file ‘Disk’ to programmed variable (main memory) in order to use current value of ‘A’ in transaction logic.

5.3.2 Write (A)

Modification of data item ‘A’ in DB file.

5.4 ACID Properties

To preserve integrity (correctness) each transaction must satisfy ACID properties

DBMS Software	A:Atomicity } D:Durability } Recovery management component of DBMS software take cares of atomicity and durability
	I : Isolation } Concurrency control component of DBMS software is responsible for isolation.
DBA User	C : Consistency } User (DBA or DB developer) is responsible for consistency.

5.5 Atomicity

Execute all operations of transaction including commit or execute none of the operation of transaction by the time of transaction termination.

Recovery management component should roll back, if transaction fails anywhere before commit.

5.5.1 Redo Operation

- It performs all the modification of database file because of transaction commit.
- Clean all the log entries of committed transaction.
- Redo is not performed after every commit but after every check point.

5.5.2 Undo Operation

- Undone all the write operation performed by the transaction.
- Convert dirty block (updated block) into clean block.

5.5.3 Check point

- Check point issued by DBMS software in regular interval.
- If checkpoint issued
 - (I) Performs Redo operation on all the committed transaction until previous checkpoint.

Example :

9 : 00 AM	DB Started
:	
9 : 05 AM	1 st checkpoint
:	
9 : 10 AM	2 nd checkpoint
:	
9 : 15 AM	System Crash

5.5.4 System Crash

If System crash/failure happen, required operation to recover are

- (I) All committed transaction until previous checkpoint will perform Redo.
- (II) All uncommitted transaction in entire system will perform undo.
- (III) Clean all log entries.

5.5.5 Roll back (Abort)

Undo modification of database file which are done by failure transaction.

5.6 Durability

- Durability maintained by Recovery management component of DBMS Software.
- The transaction should able to recover under any case of failure.
- Transaction fails because of
 - 1. Power failure
 - 2. Software crash
 - OS restarted
 - DBMS restarted
 - 3. OS/DBMS concurrency controller may kill transaction.
 - 4. Hardware Crash (Disk failure)
RAID architecture of disk design is used to overcome the problem.

5.7 Consistency

- User responsible for consistency.
- Database should be consistent before and after the execution of the transaction.
- DB operations requested by user (SQL queries/transaction operation) must be logically correct.

5.8 Isolation

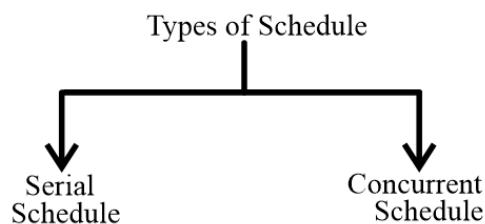
- Isolation maintained by concurrency control component.
- Isolation means concurrent execution of 2 or more transaction result must be equal to result of some serial schedule.

5.9 Schedules

Time order execution sequence of two or more transactions.

Example :

S : R₂(A) R₁(A) W₃(A)



5.9.1 Serial Schedule

- After Commit of one transaction, begins (Start) another transaction.
- Number of possible serial Schedules with ‘n’ transactions is “ $n!$ ”
- The execution sequence of Serial Schedule always generates consistent result.

Example

S : R₁(A) W₁(A) Commit (T₁) R₂(A)W₂(A) commit (T₂).

5.9.2 Advantage

- Serial Schedule always produce correct result (integrity guaranteed) as no resource sharing.

5.9.3 Disadvantage

- Less degree of concurrency.
- Through put of system is low.
- It allows transactions to execute one after another.

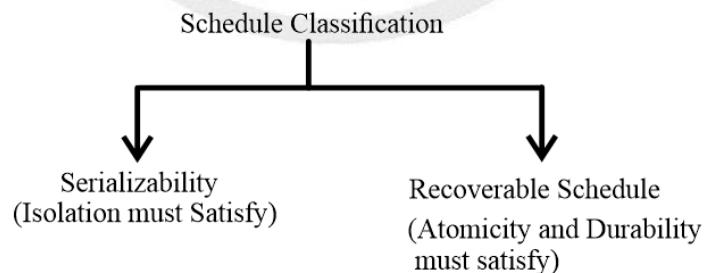
5.10 Concurrent Schedule

Transactions can execute concurrently or simultaneously

- May result inconsistency.
- Better through put and less response time.
- To maintain consistency transaction should satisfy ACID property.
- If T₁ and T₂ transaction with ‘n’ and ‘m’ operations each, then

$$\text{No. of concurrent Schedule} = \frac{(n+m)!}{n! m!}$$

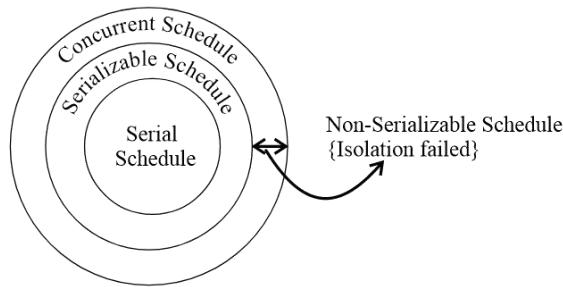
5.11 Schedule Classification



5.12 Serializable Schedule

A Schedule is serializable Schedule if it is equivalent to a Serial Schedule.

- (i) Conflict Serializability
- (ii) View Serializability



5.13 Conflict Serializability

5.13.1 Topological order

Graph traversal algorithm for directed graph

- Visit vertex (v), whose indegree is '0' and delete 'V' from the graph.
- Repeate (i) for all the vertices of graph.

5.13.2 Conflict Pairs

Two operations form Schedule (s) are conflict pair if and only if

- Atleast one write operation.
- Operation on different data item.
- Operations are from different transactions.

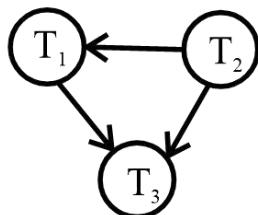
$$\left. \begin{array}{l} S : r_i(A) \dots w_j(A) \\ S : w_i(A) \dots r_j(A) \\ S : w_i(A) \dots w_j(A) \end{array} \right\} \text{Conflict Pairs}$$

5.13.3 Precedence Graph

A graph G in which vertex (v) represent the transaction of Schedule and edges (E) represent conflict pair precedence's.

Example:

S : R₂(A) R₂(B) W₁(A) W₂(B) W₃(A) W₃(B)



5.13.4 Conflict Equal Schedule

Schedule S₁ and S₂ are conflict equal Schedule if and only if Schedule S₂ can derived by inter changing consecutive non-conflict pairs of Schedule S₁.

$$S_1 \xleftarrow[\text{Consecutve non-conflict pair}]{}^{\text{Interchange}} S_2$$

Note:

- If Schedule S_1 and S_2 have
- Same set of transactions, and
 - Same precedence graph and
 - A cyclic precedence graph then
 S_1 and S_2 are conflict equal Schedule.

Important Point 1:

- If S_1, S_2 Schedule are conflict equal then precedence graph of S_1 and S_2 must be same.
- If S_1 and S_2 have same precedence graph then S_1 and S_2 may or may not conflict equal.

5.14 Conflict Serializable Schedule

Schedule (s) is conflict serializable Schedule if and only if some serial Schedule (S) must be conflict equal to Schedule (S).

$$\underbrace{\text{Schedule (s)}}_{\substack{\text{Conflict} \\ \text{Serializable} \\ \text{Schedule}}} \xleftarrow[\text{Equal}]{\text{Conflict}} (S) \text{ Serial Schedule}$$

Important Point 2:

Schedule (s) is conflict serializable Schedule if and only if

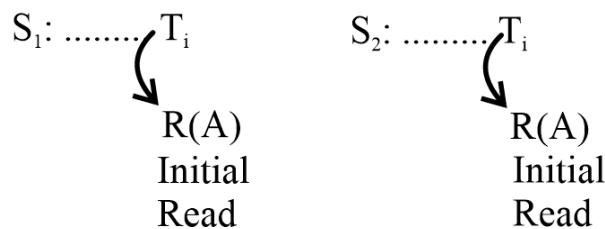
- Precedence graph is a cyclic and
- Conflict equal serial schedule are to apological order of precedence graph.

Note:

[No. of serial schedule conflict equal to schedule s] = [No. of topological order of schedules precedence graph]

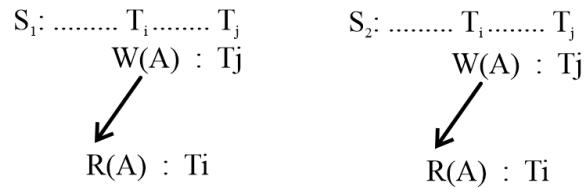
5.15 View Serializable Schedule

- Schedule (s) is view serializable if and only if some serial schedule (s') view equal to schedule (s).
- Let S_1 and S_2 be two schedule with the same set of transactions, S_1 and S_2 are view equal view equivalent if the following three conditions are met:

5.15.1 Initial Read

For each data item A, if transaction T_i reads the initial value of A in Schedule S_1 , then transaction T_i must, in Schedule S_2 , also read the initial value of A.

5.15.2 Updated Read



For each data item A if transaction T_i perform Read (A) in Schedule S_1 and that value was produced by transaction T_j , then transaction T_i must in Schedule S_2 also read the value of A that is produce by the transaction T_j .

5.15.3 Final Write



For each data item A, the transaction that perform the final write (A) operation in schedule (S_1) must perform the final write (A) operation in schedule S_2 .

Important Point 3

- If schedule S is conflict serializable schedule, then S is also view serializable schedule.
- If schedule s is not conflict serializable then it may or may not view serializable
- Every view serializable schedule that is not conflict serializable.

5.16 Classification based on Recoverability

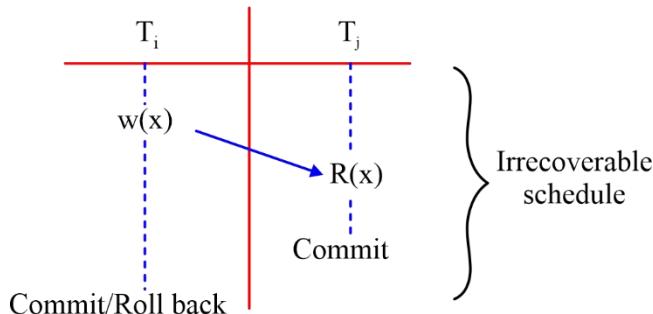
The concurrent execution may leads:

1. Irrecoverable schedule
2. Cascading rollback problem
3. Lost update problem

The above problems can occur even if the schedule is serializable (Integrity satisfied)

5.16.1 Irrecoverable Schedule

- A schedule(s) is irrecoverable if and only if transaction T_j reads data item ‘x’, which is updated by T_i and commit of T_j before commit/rollback of transaction T_i .
- Irrecoverable means unable to recover or rollback.



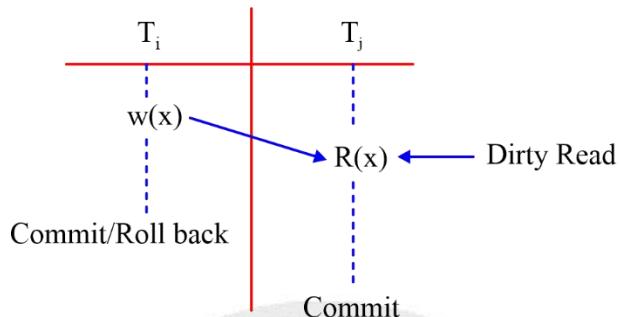
5.16.2 Recoverable Schedule

A schedule(s) is recoverable if and only if

(I) No uncommitted read (no dirty ready in schedule(s)).

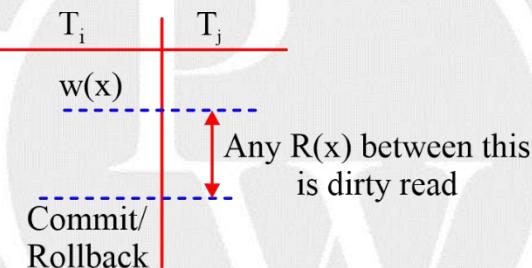
Or

(II) If transaction T_j reads data item 'x' which is updated by transaction ' T_i ', then commit of T_j must be delayed until commit/rollback of T_i .



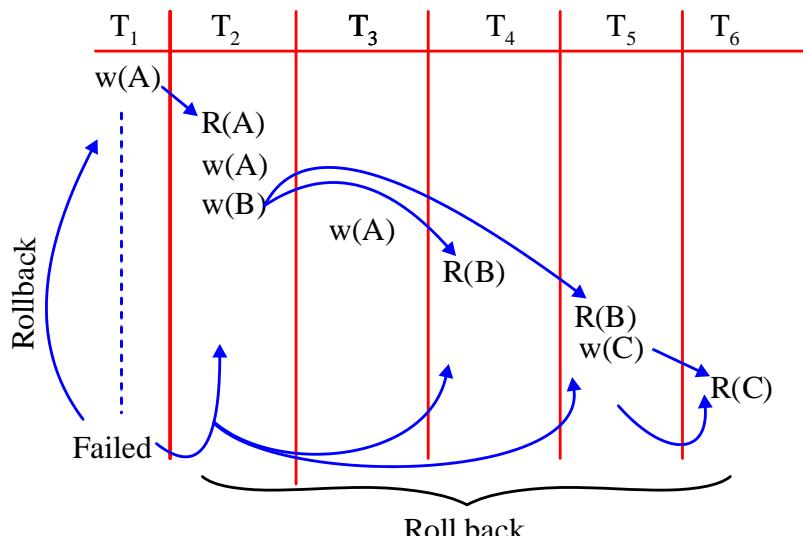
5.16.3 Uncommitted Read (Dirty Read)

Transaction T_j reads data item 'x' which is updated by uncommitted transaction. T_i



5.17 Cascading Rollback Schedule (Problem)

When some transaction reads data items which is updated by some other transaction then because of failure of the transaction that updated the data item may rollback all the dependent transaction.



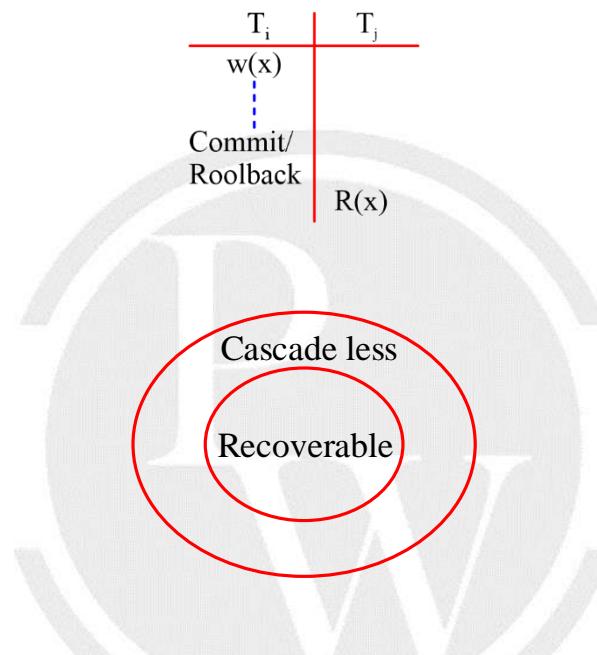
Failure of T_1 forced to rollback T_2 , T_4 , T_5 and T_6 .

- **Disadvantage of Cascading Rollback**

1. It waste the CPU execution time.
2. Wastage of I/O access cost.

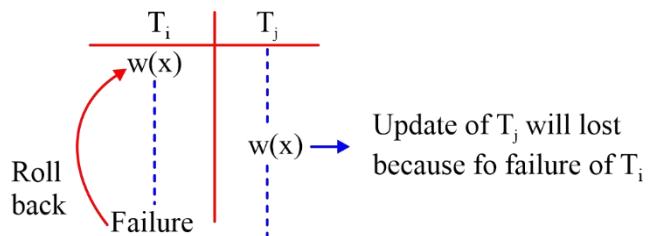
5.17.1 Cascadeless Rollback Recoverable Schedule

- No dirty read in the schedule.
- Transaction T_j should delay read (x) which is updated by T_i , until T_i commit or rollback.



5.18 Lost Update Problem

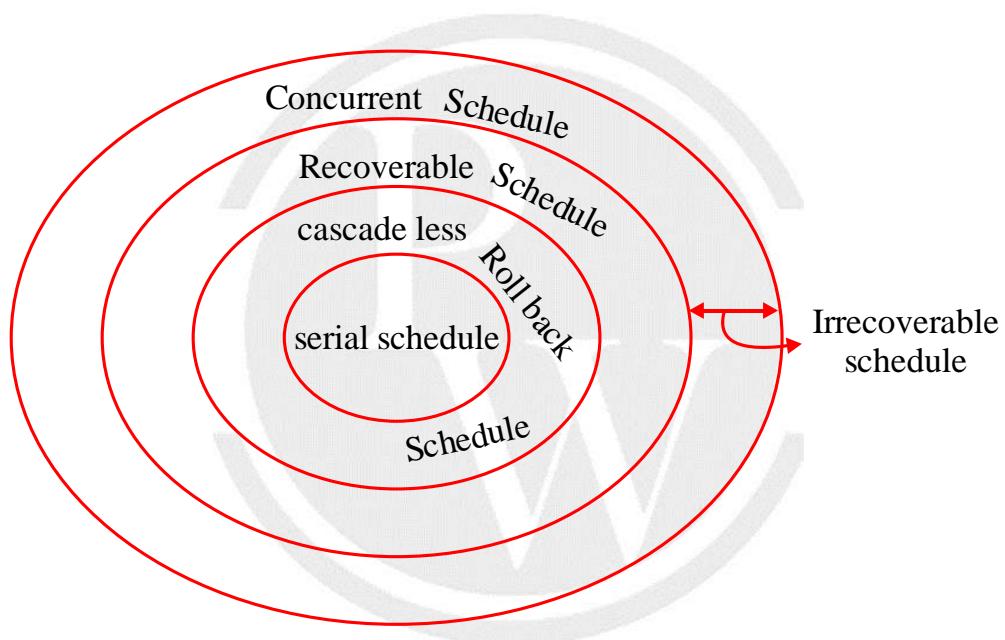
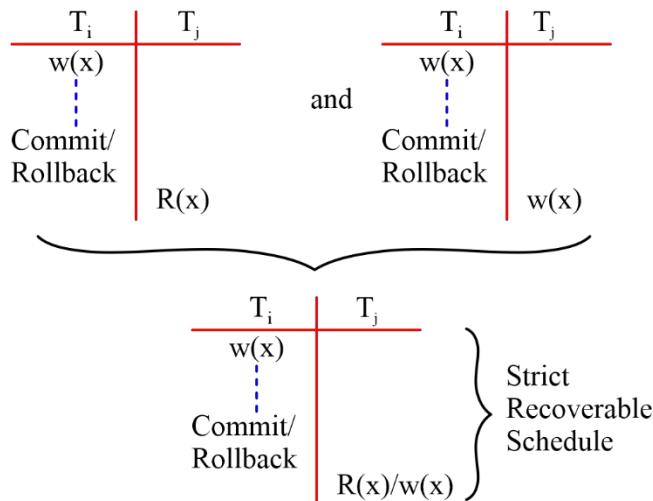
Lost update problem occur if T_j write (x) which is already written by uncommitted transaction T_i .



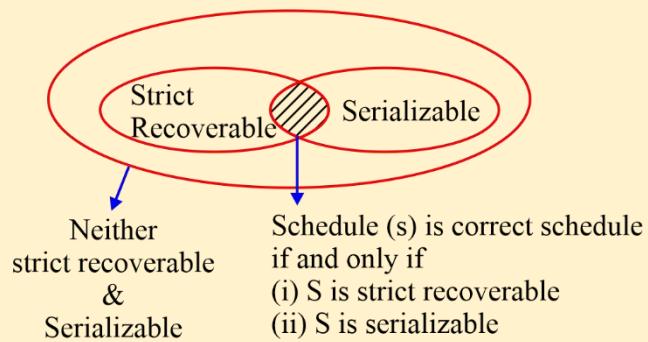
5.19 Strict Recoverable schedule

A schedule must satisfy

1. Cascadeless rollback recoverable schedule and
2. If T_i writes (x) then other transaction T_j write (x) must be delayed until T_i commit or rollback.

**Note:**

1. Strict recoverable schedule may or may not be serializable.
2. Serializable schedule may or may not be recoverable.



5.20 Concurrency control protocol

Concurrency control protocol should not allow to execute:

1. Non-serializable schedule (Violate Isolation).
2. Non-strict recoverable schedule (Violate atomicity and Durability).

5.21 Locking Protocol

Lock is a variable used to identify the status of data item.

Transaction (T_i)

Lock (A): Granted by concurrency controller

R(A)

W(A)

Lock(B): Denied by concurrency controller

5.22 Types of Lock

1. Shared Lock(s): Read only lock.
2. Exclusive lock (x): Read/write lock.

1. Shared (s mode)

- Exists when concurrent transaction granted READ access.
- Produce no conflict for Read only transactions.
- Issued when transaction wants to read and exclusive lock not held on item.

2. Exclusive (x mode)

- Exists when access reserved for locking transaction.
- Used when potential for conflict exists.
- Issued when transaction wants to update unlocked data.

Example:

Transaction (T_1)

S(A) : Granted shared lock

R(A) } only read allowed

X(B) : Granted exclusive lock

R(B) } Read and write
W(B) }

5.22.1 Lock compatible table

Requested by T_j	Data item A	S	X	→ Holded by T_i
	{ S	Yes	No	
	X	No	No	

5.23 Phase Locking Protocol (2PL)

- It guaranteed serializability.
- Transaction (T) allowed to request lock on any data item in any mode (x/s) until first unlock of transaction (T).

1. Growing Phase

Acquires all the required locks without unlocking any data. Once all locks have been acquired, the transaction is in it locked point.

2. Shrinking phase

Release all locks and can not obtain any new lock governing rules of 2 PL.

Example:

Transaction (T)

S(A)
X(B)
S(C)
X(D) *
U(C) *
U(D)
U(A)
U(B)

Growing phase
(locking phase)

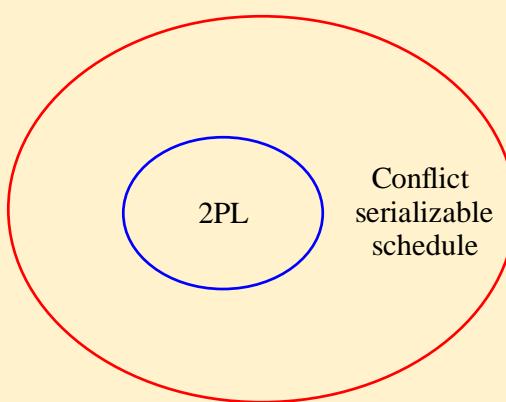
Locking point

Striking phase
(unlocking phase)

Important point 1

1. If schedule (s) is executed by 2PL then schedule guaranteed is conflict serializable schedule.
2. Conflict equal schedule is based on lock points order of the transaction.
3. If schedule is not conflict serializable schedule (cyclic precedence graph) then schedule not allowed to execute by 2 PL.

Note: Every schedule which is allowed by 2PL is always conflict serializable, but not every conflict serializable schedule is allowed by 2PL.



5.23.1 Limitations of 2PL

1. 2PL restriction may leads to deadlock.
2. 2PL restriction may leads to starvation.
3. 2PL condition not sufficient to avoid
 - (I) Ir-recoverable schedules
 - (II) Cascading rollback problem
 - (III) lost updated problem

5.24 Strict 2PL protocol

- **Basic 2PL:** Lock request of transaction (T) not allowed in shrinking phase of transaction T and

Strict recoverable: All exclusive lock of transaction must hold until commit/Rollback of transaction T.

5.24.1 Strict 2PL protocol guarantees

- (a) Serializability.
- (b) Strict recoverable.

Disadvantage:

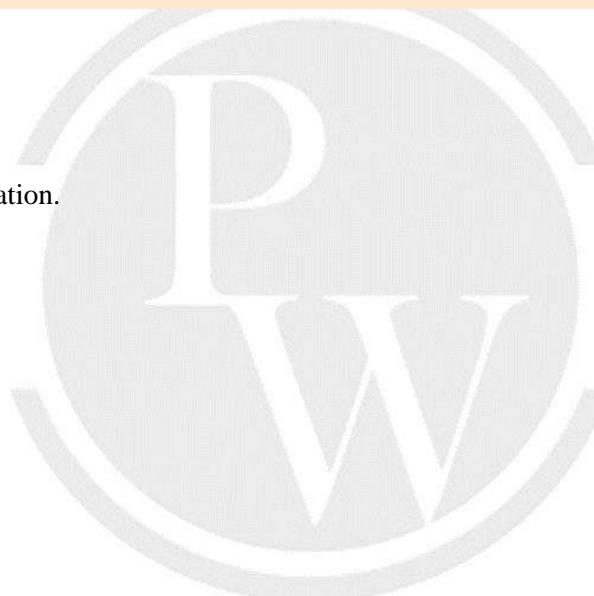
It is not free from deadlock and starvation.

Example:

Transaction (T)

$S(A)$
 $X(B)$
 $S(C)$
 $X(D)$

$U(A)$
 $U(B)$
Commit
 $U(B)$
 $U(D)$



5.25 Rigorous 2PL

Basic 2 PL : A lock request allowed only in growing phase of transaction and all locks (shared/Exclusive) of transaction T must be held until commit or Rollback.

5.26 Conservative 2PL

Transaction (T) must lock all required data items in starting phase.

If locks are not available then unlock all data items and re-request all data item lock again.

	Basic 2PL	Strict 2PL	Rigorous 2PL	Conservative 2PL
Guaranteed Serializability	Yes	Yes	Yes	Yes
Guaranteed strict Recoverable	No	Yes	Yes	No
Free from dead lock	No	No	No	Yes
Free from starvation	No	No	No	NO

5.27 Time stamp ordering protocol

Assign global unique time stamp value to each transaction and produces order for transaction submission.

The time stamp values of transaction can be used for transaction identification and to set priority between the transaction.

5.27.1 Time stamp value for each data item

- Assume data item is A
- I. **Read TS value (A):** It is the highest transaction time stamp value that has executed R(A) successfully.
- II. **Write TS value (A):** It is the highest transaction TS value that has executed W(A) successfully.

1. Basic Time stamp ordering protocol:

- a. If transaction T issues R(A) operation.

If (WTS(A)) > TS(T)

then Rollback T

else

{Allow to execute R(A) successfully}

set RTS (A) = max {RTS(A), TS(T)}

- b. If transaction T issues W(A) operation

if (RTS(A)) > TS (T)

{then Rollback trans (T)}

else if (WTS(A) > TS(T))

{then Roll back T}

Else

{Allow to execute W(A) successfully}

set WTS(A) = {TS(T)}

2. Thomas write rule stamp ordering protocol

- a. If transaction T issues R(A) operation

if (WTS(A)) > TS(T)

{then Rollback transaction T}

else

{allow to execute R(A) successfully
set RTS (A) = max {TS(T), RTS (A)}
b. If transaction T issue W(A) operation
if (RTS (A) > TS(T))
{then roll back T}
else if (WTS(A)) > TS(T)
{then ignore w(A) of transaction T and continue.}
else
{Allow to execute w(A) successfully
set WTS(A) = TS(T)}

3. Strict time stamp ordering protocol

A transition T_2 that issues a R(A) or W(A) such that $TS(T_2) > WTS(A)$ has its read operation delayed until the transaction T_1 that wrote the value x has committed or rolled back.

Basic TS ordering protocol	Thomas write TS ordering protocol	Strict TS ordering protocol
1. Ensures conflict serialization	1. Ensures view serialization	1. Ensures conflict serialization
2. Free from deadlock	2. Free from deadlock	2. Free from deadlock
3. Not free from starvation	3. Not free from starvation	3. Not free from starvation
4. Not guaranteed strict recoverable	4. Not guaranteed strict recoverable	4. Guaranteed strict recoverable



GATE Exam 2025?



SHRESHTH BATCH

ME | CE | EE | EC | CS & IT

- Live Classes & Recorded Videos
- DPPs & Solutions
- Doubt Solving
- Batches are available in *Hindi*

Weekday & Weekend
Batches Available



JOIN NOW!



Physics W

Computer Networks



NETWORK



Computer Networks

INDEX

1. IP Addressing, Subnetting & Supernetting **11.1 – 11.3**
2. Error Control..... **11.4 – 11.6**
3. Flow Control **11.7 – 11.10**
4. IPv4 Header **11.11 – 11.14**
5. TCP & UDP **11.15 – 11.19**
6. Medium Access Control[MAC] **11.20 – 11.22**
7. Routing Algorithms, Switching & IP Support Protocol..... **11.23 – 11.27**
8. Application Layer Protocol **11.28 – 11.33**
9. OSI and ICP/IP Protocol Stack **11.34 – 11.37**

1

IP ADDRESSING, SUBNETTING & SUPERNETTING

1.1 IP Addressing

Class A : 0	→	(1 - 126),	No. of IP Addresses = 2^{31}
Class B : 10	→	(128 - 191),	No. of IP Addresses = 2^{30}
Class C : 110	→	(192 - 223),	No. of IP Addresses = 2^{29}
Class D : 1110	→	(224 - 239),	No. of IP Addresses = 2^{28}
Class E : 1111	→	(240 - 255),	No. of IP Addresses = 2^{28}

1.2 Default subnet Mask

Class A : 255.0.0.0
Class B : 255.255.0.0
Class C : 255.255.255.0

1.3 Private Addresses Range

10.0.0.0 to 10.255.255.255 → 1 class A Network.
172.16.0.0 to 172.31.255.255 → 16 class B Network.
192.168.0.0 to 192.168.255.255 → 256 class C Network.

Class	Number of Networks	Number of hosts per Network
Class A	$2^7 - 2 = 126$	$2^{24} - 2 = 1,67,77,214$ hosts
Class B	$2^{14} = 16,384$	$2^{16} - 2 = 65,534$ hosts
Class C	$2^{21} = 20,97,125$	$2^8 - 2 = 254$ hosts
Class D	No NID and HID, all 28 remaining bits are used to define multicast address	
Class E	No NID and HID, it is meant for research and future purpose	

Note:

The IP address 127.x.y.z is known as loop back address and it is used to check the connectivity.

1.4 Types of Communication

- (i) Unicast communication (1 : 1)
- (ii) Broadcast communication (1 : All)
- (iii) Multicast Communication (1: Many)

1.5 Unicast Communication

- 1. Transmitting the data from one computer to another computer is called as unicast communication.
- 2. It is one to one transmission.
- 3. In Unicast communication both source and destination either present in the same network or in the different network.

1.6 Broadcast Communication



1.6.1 Limited Broadcasting

- 1. Transmitting data from one computer to all other computer in the same network is called as Limited Broadcasting.
- 2. Limited Broadcast Address = 255.255.255.255
- 3. Limited broadcast address can't be used as a source IP Address.
- 4. Limited broadcast Address will always be used as a Destination IP.

1.6.2 Direct Broadcasting

- 1. Transmitting data from one computer to all other computer in the different network is called as Limited Broadcasting.
- 2. Direct broadcast address can't be used as a source IP Address.
- 3. Direct broadcast Address will always be used as a Destination IP.

	<u>NID</u>	<u>HID</u>		
1.	—	0's	→	Network ID
2.	—	1's	→	Direct Broadcast Address (DBA)
3.	1's	1's	→	Limited Broadcast Address (LBA)
4.	0's	—	→	Host with in the Network
5.	1's	0's	→	Network Mask or Subnet Mask

1.7 Multicast communication

Transmitting a packet from one computer to many computers (0 or more) is called Multicast communication.

1.8 CIDR Rules

1. All the IP Address in the Block must be contiguous.
2. Block size must be a power of 2.
3. First IP address of the block must be divisible by size of the block.

1.9 Supernetting

The process of combining two or more network to get a single network is called as supernetting.

1.10 Advantage of Supernetting

1. Super netting Reduce Routing table entry.
2. Router will take less time for processing the packet.
3. It improve flexibility of IP Address Allotment i.e. If someone required 500 Address, then no need to purchase class B network we can combine two class C network.

1.11 Rules of Supernetting

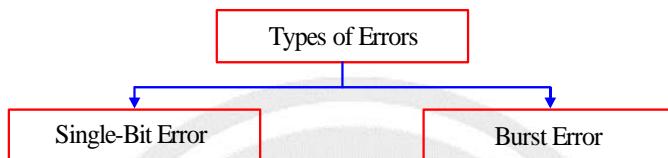
1. Network ID must be contiguous.
2. Size of the Network must be same and number of Network must be a power of 2.
3. First Network ID must be divisible by size of the supernet.



2

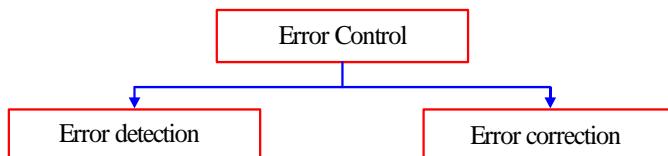
ERROR CONTROL

2.1 Error Control

**Note:**

- The number of corrupted bits or affected bits depends on the data rate and duration of noise.
- The number of corrupted bits or affected bits = Data rate * Noise duration.
- Burst error is more likely to occur than a single bit error.
- Error correction is more difficult than error detection.

Error Detection		Error Correction
1.	Once noticed error simply discard.	Capability of correcting error.
2.	Ask for retransmission.	Does not required retransmission.



1.	Simple Parity	1.	Hamming code
2.	2D parity		
3.	Check sum		
4.	CRC		

2.1.1 Hamming distance

Hamming distance between two Binary string of same size is the number of differences between corresponding bits.
Hamming distance between two Binary string is denoted by $d(x, y)$

$$d(000, 011) = 2$$

$$\begin{aligned}d(100, 011) &= 3 \\d(10101, 11110) &= 3\end{aligned}$$

Hamming distance can easily be found if we apply XOR operation (\oplus) on the two words and count the Number of 1's in the result.

2.1.2 Minimum Hamming distance

In a set of codewords, the minimum Hamming distance is the smallest Hamming distance between all possible pairs of code words.

Valid code word	$d(a, b) = 3$	$d(a, c) = 1$	$d(a, d) = 2$	$d(b, c) = 2$	$d(b, d) = 1$	$d(c, d) = 3$
0 1 0 (a)						
1 0 1 (b)						
1 1 0 (c)						
0 0 1 (d)						

Minimum Hamming distance = 1

2.1.3 Minimum Hamming distance for Error detection

To detect 'd' bit error minimum hamming distance required = $d+1$

2.1.4 Min. Hamming distance For Error Correction

To Correct 'd' bit error minimum hamming distance required = $2d+1$

2.2 Simple Parity Check Code

Simple parity

In the Simple parity concept one extra bit (parity bit) is added to each dataword.

Simple parity check can detect all single bit error .

Simple parity check can not detect an even number of errors.

Simple parity check can detect an odd number of errors .

2.3 2D Parity Check Code

2D parity

Two dimensional parity check can detect and correct all single bit error and detect two or three bit error that occur anywhere in the matrix.

However only some pattern with four or more Errors can be detected.

In a 2D-parity check code, the information bits are organized in a matrix consisting of rows and columns.

For each row and each column one parity check bits is calculated.

2.4 CRC

Length of the dataword = n

Length of the divisor = k

Append (k-1) Zero's to the original message

Perform modulo 2 division

Remainder of division = CRC

Codeword = dataword with Appended (k-1) Zero's + CRC

Note:

1. CRC must be $(k-1)$ bits.
2. If the generator has more than one term and coefficient of x^0 is 1, all single bit error can be detected.
3. If a generator can't divide $x^t + 1$ (t between 0 and $n - 1$) then all isolated Double error can be detected.
4. The generator that contains a Factor of $x + 1$ can detect all odd numbered errors.

2.4.1 A good polynomial generator needs to have the following characteristics

1. It should have at least two terms.
2. The coefficient of the term x^0 should be 1.
3. It should not divide $x^t + 1$, for t between 2 and $n - 1$.
4. It should have the factor $x + 1$.

2.5 Hamming Code

1. Hamming code is used for error correction.
2. Hamming code can correct 1 bit error only.
3. Hamming code can detect upto 2 bit error.

m = Message bits

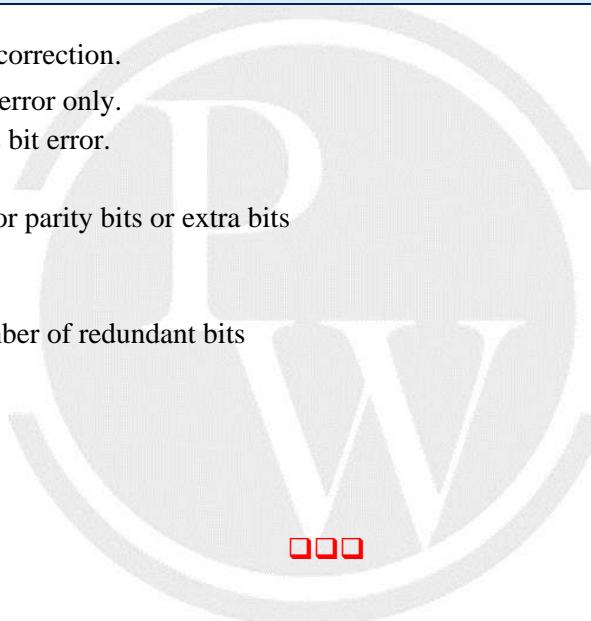
r = redundant bits or Check bits or parity bits or extra bits

$n = m + r$ (n = codeword)

According to the hamming code, number of redundant bits

$$m + r + 1 \leq 2^r$$

where r = lower limit



3

FLOW CONTROL

3.1 Delay in Computer Network

- (1) Transmission Delay (T_d)
- (2) Propagation Delay (P_d)
- (3) Queuing Delay (Q_d)
- (4) Processing Delay (P_{rd})

3.2 Transmission Delay

Amount of time taken to transfer a packet on to the outgoing link is called as Transmission delay.

$$\text{Transmission delay} (T_d) = \frac{\text{Length of packet}}{\text{Bandwidth}}$$

$$T_d = \frac{L}{B}$$

3.3 Propagation Delay

Amount of time taken to reach a packet from one (sender) point to another (receiver) point is called as propagation delay.

$$\text{Propagation delay} (P_d) = \frac{\text{distance}}{\text{velocity}}$$

$$P_d = \frac{d}{v}$$

3.4 Stop wait Protocol

3.4.1 Sender Side Rule

Rule 1 : Sender can send one data packet at a time.

Rule 2 : Sender can send the next data packet only after receiving the ACK of the previous packet.

3.4.2 Receiver Side Rule

Rule 1: Receiver will receive and consume the data packet.

Rule 2 : After consuming the data packet, Ack need to be sent.

3.5 Efficiency OR Line utilization OR Link utilization OR Sender utilization

$$\text{Efficiency} = \frac{\text{Useful time}}{\text{Total time}}$$

$$\text{Efficiency} = \frac{T_d(\text{frame})}{T_d(\text{frame}) + 2 * P_d + Q_d + P_{rd} + T_d(\text{ACK})}$$

3.6 Throughput Or Effective Bandwidth Or Bandwidth Utilization Or Maximum Data Rate Possible

$$\text{Throughput} = \frac{\text{Length of the Frame}}{\text{Total time}}$$

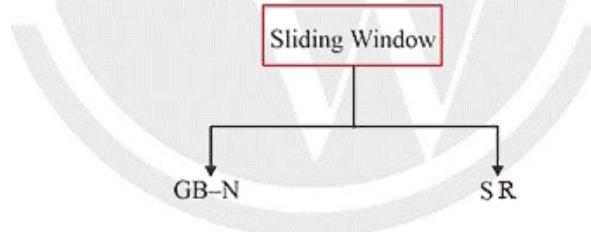
$$\text{Throughput} = \frac{L}{T_d(\text{frame}) + 2 * P_d + Q_d + P_{rd} + T_d(\text{ACK})}$$

OR

$$\text{Throughput} = \eta * B$$

3.7 Sliding Window

In the sliding window concept instead of sending one packet and wait for the acknowledgement, we send ‘w’ packet and wait for the Acknowledgement. Where ‘w’ is the sender window size.



3.7.1 GB-N(N>1)

1. In the GB – N the sender window size is N itself.
2. In the GB-N the receiver window size is equal to one always.

Note:

- (1) Out of order packet is not received by Receiver.
- (2) Timer is maintained only for the first frame (Rightmost) in window because if its timer expires then sender assume that rest of the frame are also not received by receiver (because out of order delivery is rejected).

Note:

GB–N uses cumulative Acknowledgement and Acknowledgement number defines the number of the next expected frame.

Ack timer < Time out timer

Window Receiver (W_R) size:

In the GB-N the window receiver size is equal to one always irrespective of window sender size ($W_R=1$).

Window Sender (W_S) Size:

Window sender size is calculated based on the following formula:

$$W_S + W_R \leq \text{Available Sequence Number}$$

$$W_S + 1 \leq \text{Available Sequence Number}$$

$$W_S \leq \text{Available Sequence Number} - 1$$

3.7.2 Efficiency and Throughput in GBN

$$\text{Efficiency} = \frac{\text{Useful time}}{\text{Total time}}$$

$$\boxed{\text{Efficiency} = \frac{N * T_d(\text{frame})}{T_d(\text{frame}) + 2 * P_d + Q_d + P_{rd} + T_d(\text{ACK})}}$$

$$\text{Throughput} = \frac{N * \text{Length of Frame}}{\text{Total time}}$$

$$\boxed{\text{Throughput} = \frac{N * \text{Length of Frame}}{T_d(\text{frame}) + 2 * P_d + Q_d + P_{rd} + T_d(\text{ACK})}}$$

OR

$$\boxed{\text{Throughput} = \eta * B}$$

3.8 Selective Repeat ARQ

3.8.1 Selective Repeat/Selective Reject ARQ

- (1) In SR Protocol window sender size is equal to window receiver size ($W_S = W_R$).
- (2) SR Protocol uses independent acknowledgement, and acknowledgement number defines number of error free packet received.
- (3) SR receiver can receive out of order packet but packets are delivered to upper layer in sorted order.
- (4) In SR protocol searching and sorting logic is required. Searching is done by sender and sorting is done by receiver.
- (5) Timer is maintained for each and every frame in the window at sender side.
- (6) For 1st out of order delivery or if packet received is corrupted then Negative acknowledgment (NACK) for respective packet is sent by receiver to sender.
- (7) When sender receive NACK 3 then it will search in the window for packet 3 & immediately packet 3 is retransmitted even though its timer is not expired.

3.8.2 Efficiency and Throughput of SR

$$\text{Efficiency} = \frac{\text{Useful time}}{\text{Total time}}$$

$$\text{Efficiency} = \frac{W_s \times T_d(\text{frame})}{T_d(\text{frame}) + 2*P_d + Q_d + P_{rd} + T_d(\text{ACK})}$$

$$\text{Throughput} = \eta * B$$

$$\text{Throughput} = \frac{W_s * \text{Length of the frame}}{\text{Total time}}$$

$$\text{Throughput} = \frac{W_s * \text{Length of the frame}}{T_d(\text{frame}) + 2*P_d + Q_d + P_{rd} + T_d(\text{ACK})}$$

3.9 Comparison among Stop and Wait, GBN and SR protocols

	Stop & wait	GBN	SR
Efficiency	$\eta = \frac{\text{Useful time}}{\text{Total time}}$ or $\eta = \frac{T_d(\text{frame})}{\text{Total time}}$	$\eta = \frac{\text{Useful time}}{\text{Total time}}$ or $\eta = \frac{N*T_d(\text{frame})}{\text{Total time}}$	$\eta = \frac{\text{Useful time}}{\text{Total time}}$ or $\eta = \frac{W_s * T_d(\text{frame})}{\text{Total time}}$
Throughput	$\frac{\text{Length of frame}}{\text{Total time}}$ or $\eta * B$	$\frac{N*\text{Length of the frame}}{\text{Total time}}$ or $\eta * B$	$\frac{W_s * \text{Length of the frame}}{\text{Total time}}$ or $\eta * B$
Buffer	1 + 1	N + 1	N + N
Seq No.	2	N + 1	2N
Seq. No. = K bit		$W_s = 2^K - 1$ $W_R = 1$	$W_S = 2^{K-1}$ $W_R = 2^{K-1}$

$$\text{RTT or Total Time} = T_d(\text{frame}) + 2*P_d + T_d(\text{ACK}) + P_{rd} + Q_d$$



4

IPv4 HEADER

4.1 IPv4 Header

VER (4 bits)	HL (4 bits)	Services (8 bits)	Total Length (16 bits)		
Identification number (16 bits)	Flags (3 bits)	Fragment offset (13 bits)			
Time to Live (8 bits)	Protocol (8 bits)	Header checksum (16 bits)			
Source IP Address (32 bits)					
Destination IP Address (32 bits)					
Option (0-40 bytes)					

4.1.1 Version (4 Bit)

It is used to indicate IPv4 or IPv6.

4.1.2 Header Length(4 Bit)

Header length is a 4 bit field that contains the length of header.

Minimum Header size is 20 byte.

Maximum Header size is 60 byte.

4.2 Services [8 bit]

In this Interpretation the first 3 bit are called precedence bit (Priority bit) and Next 4 bit are called types of services bits and last bit is Not used.

4.2.1 Priority

It is a 3-bit subfield ranging from 0 to 7 (000 to 111 in binary). Priority field is needed if a router is congested and need to discard some datagram, those datagrams which have the lowest priority are discarded first.

4.2.2 Types of Services

It is a 4 bit subfield. Each bit having a special meaning, although a bit can be 0 or 1. One and only one of the bits can have the value 1 in each datagram.

4.3 Total length (16 bits)

Total length = Data + Header

4.4 Identification Number (16 bits)

1. Each datagram is associated with a sequence number is called as datagram number or identification number.
2. It is used to identify all the fragment of same datagram.
3. All the fragment of same datagram will have the same identification number.

4.5 Flags

It is the 3 bit field shown in the figure.

X	D F	M F
Not Used	Don't Fragment	More Fragment

4.6 Fragment offset (13 bits)

Fragment offset indicate no of data byte ahead of this fragment in that particular packet.

4.7 TTL (8 bits)

1. TTL is used to avoid infinite looping.
2. TTL field is used to control the maximum number of hops visited by datagram.
3. When a source host sends a datagram, it stores a number in this field. Each router that process the datagram decrements this number by one. If TTL field reaches zero before the datagram arrives at its destination, then the datagram is discarded and an ICMP message is sent back to sender.

4.8 Protocol (8 bits)

1. This 8 bits field tell us which protocol is encapsulated in the IP packet.
2. At the time of traffic, some packet must be discarded. In this case it will be advantageous to know which protocol data it contains.
3. The order in which router eliminate the datagram from buffer is-
ICMP>IGMP>UDP>TCP
(01) (02) (17) (06)

4.9 Header Checksum

1. It is calculated only for header part not the data because rest of the component in packet already covered by TCP checksum.
2. Header checksum is calculated at each and every Router because IP Header might be change when packet is moving from one router to another.
3. Every router makes one modification i.e. TTL so Header checksum is calculated at every Router.
4. Fragment offset, MF, Total length, option all may be changed at a Router.

4.10 Source Address (32 bits)

This 32 bit defines the IPv4 address of source. This field remain unchanged during the time the IPv4 datagram travel from the source Host to destination Host.

4.11 Destination Address (32 bits)

This 32 bits Field defines the IPv4 address of the destination. This field remain unchanged during the time the IPv4 datagram travel from source host to destination host.

4.12 Option

The Header of IPv4 data gram is made of two parts a fixed part and a variable part. The fixed part is 20 bytes long and variable part that can be maximum of 40 bytes.

There are 5 options

1. Strict source Routing
2. Loose source Routing
3. Record Routing
4. Time stamp
5. Padding

4.12.1 Strict Source Routing

A strict source routing is used by the source to predetermine a route for data gram as it travel through the internet.

4.12.2 Loose Source Routing

A loose source route option is similar to strict source route but it is less rigid. Each router in the list must visited, but the data gram can visit other router as well.

4.12.3 Record Routing

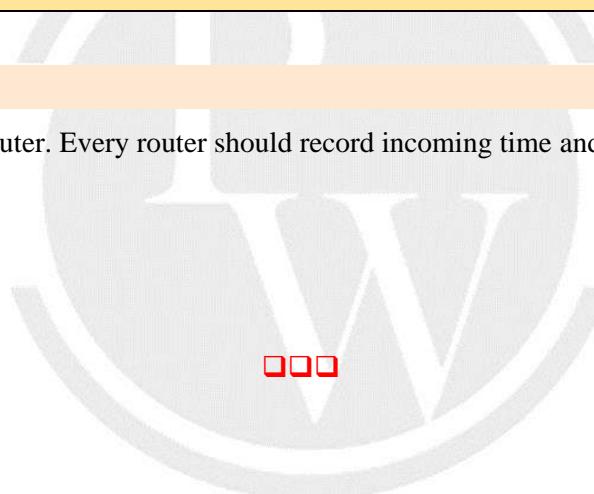
A record route option is used to record the internet routers that handle the data gram. It can list up to 9 router Address. All the Router are supposed to record their IP Address on their IP packets.

Note:

First 16 bits (2 byte) are reserved for option type (8 bit) and length (8 bit). Out of 40 bytes only 38 bytes are remaining for storing IPv4 addresses. In 38 bytes we can store 9 IPv4 addresses as each IPv4 address is of 4 byte

4.12.4 Time Stamp

It is used to find out delays at each router. Every router should record incoming time and outgoing time.

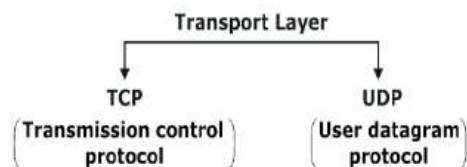


5

TCP & UDP

5.1 Introduction

Transport Layer can be connection oriented or connection less.



1. TCP is reliable process to process delivery of entire message.
2. TCP is a connection oriented.
3. TCP connection are full duplex and point to point.
4. TCP connection has 3 phases
 - i. Connection establishment
 - ii. Data transfer
 - iii. Connection termination
5. Each TCP connection is identified uniquely by Source Port + Destination Port + Source IP + Destination IP.
6. Each TCP connection is associated with Four window.
7. TCP uses three way “Handshake” to establish TCP connection.
8. TCP is not useful for Broadcasting and Multicasting.
9. TCP Header size is 20 bytes but if option (40 bytes) is added it will become 60 bytes.
10. TCP provide end to end error control and flow control.
11. Data will be received at the destination in order.
12. Data may arrive out of order and be temporarily stored by receiving TCP, but TCP guarantee that no out of order data delivered to the process.

5.2 TCP header

Source Port (16 bits)								Destination Port (16 bits)
Sequence number (32 bits)								
Acknowledgement number (32 bits)								
Header Length (4 bits)	Reserved bits (6 bits)	U R B	A C K	P S H	R S T	S Y N	F I N	Window Size (Advertisement Window) (16 bits)
Check sum (16 bits)								Urgent Pointer (16 bits)
Options (0 - 40 bytes)								

Port Number	Name
0 – 1023	Well known port No.
1024 – 49151	Registered Port No.
49152 – 65535	Dynamic Port No.

$$\text{Wrap Around time (WAT)} = \frac{\text{Total sequence No.}}{\text{Bandwidth [Bytes / Sec]}}$$

- Minimum sequence number required to Avoid wrap Around time with in Life time = $B \times LT$
- Minimum number of bits required to Avoid wrap Around time with in LT = $\lceil \log_2 B * LT \rceil$

SYN = 1 → Consume one sequence number.

Ack = 1 → Consume zero sequence number.

FIN = 1 → Consume one sequence number.

1 Data byte → Consume one sequence number.

SYN	Ack	Meaning
1	0	request
1	1	reply
0	1	Ack
0	0	Data

Time out timer in TCP

Basic Algorithm	Jacobson's Algorithm
$\text{Time Out Timer} = 2 * \text{RTT}$ $\text{Next Round Trip Time (NRTT)} = \alpha(\text{IRTT}) + (1 - \alpha)\text{ARTT}$ $0 \leq \alpha \leq 1$	$\text{Time Out Timer} = 4 * \text{ID} + \text{RTT}$ $\text{Next Round Trip Time (NRTT)} = \alpha (\text{IRTT}) + (1 - \alpha) \text{ARTT}$ $0 \leq \alpha \leq 1$ $\text{Actual Deviation (AD)} = \text{IRTT} - \text{ARTT} $ $\text{Next Deviation (ND)} = \alpha (\text{ID}) + (1 - \alpha) \text{AD}$

5.3 Congestion Control

- $W_s = \min(W_c, W_R)$

Slow start	Congestion Avoidance	Congestion Detection
1. If ACK Arrives $W_c = W_c + 1$	1. IF ACK Arrives $W_c = W_c + \frac{1}{W_c}$	1. Time out
2. After one RTT $W_c = 2 * W_c$	2. After one RTT $W_c = W_c + 1$	2. 3 duplicate ACK

5.4 Token Bucket

- Token bucket algorithm allows ideal hosts to accumulate credit for the future in the form of tokens.
- In regular interval, tokens are thrown into the bucket.
- Bucket has a maximum capacity.
- If there is a ready packet a token is removed from bucket and packet is sent.
- If there is no token in the bucket, the packet cannot be sent.

Let the capacity of token bucket is 'C' token and token enter into the bucket at rate of 'r' tokens per second. The maximum number of packet that can be enter into the network during the time interval 't' is

$$\begin{aligned}
 \text{Maximum number of packet} &= C + rt \\
 \text{Maximum average rate for token bucket } M &= \frac{C + rt}{t} \\
 Mt &= C + rt \\
 Mt - rt &= C \\
 (M - r)t &= C \\
 t &= \frac{C}{M - r}
 \end{aligned}$$

$C \rightarrow$ token Bucket capacity

$r \rightarrow$ Token Arrival rate

5.5 UDP

- UDP is message oriented connection less Datagram protocol.
- It is unreliable Transport protocol.
- It does not provide Flow control and Error control & congestion control.
- It does not add anything to the services except process to process delivery of data.
- Header is simple and fixed in size i.e. 8 byte.

UDP Header	
Source port (16 bit)	Destination port (16 bit)
Length (16 bit)	Checksum (16 bit)

Note:

Unlike TCP, the checksum calculation is not mandatory in UDP. No error control or flow control is provided by UDP. Hence UDP depends on IP and ICMP for error reporting.

5.5.1 Optional inclusion of checksum

The sender of UDP packet can choose not to calculate the checksum. In this case the checksum field is filled with all 0's before being sent.

5.5.2 Need of UDP

- [1] The application that required one request one reply. TCP is not suitable hence we use UDP.
- [2] Application that required constant dataflow TCP is not suitable hence we use UDP.
- [3] Application that required multimedia data transfer we cannot use TCP hence we use UDP.
- [4] Application that required fastness and then reliability TCP is not suitable hence we use UDP.
- [5] UDP used for management process such as SNMP (simple network management protocol).
- [6] UDP is used for some route updating protocol such as RIP.
- [7] For broadcasting & multicasting application TCP is no suitable hence we use UDP.
- [8] UDP is normally used for interactive real time applications.
- [9] UDP is suitable for a process with internal flow and error control mechanisms. For example, the Trivial File Transfer protocol(TFTP) process include flow and error control. It can easily use UDP

TCP	UDP
Connection-oriented	Connectionless
Reliability in delivery of message	Not reliable
Sequence Number.	No sequence number.
ACK number.	No ACK number.
Overhead is high(Header size 20-60 Bytes)	overhead is less(Header size = 8 Bytes)
Keep track of order (sequence)	No order
Protocols: HTTP, FTP, SMTP, POP	Protocol: DNS, SNMP, TFTP, DHCP, All real time protocol

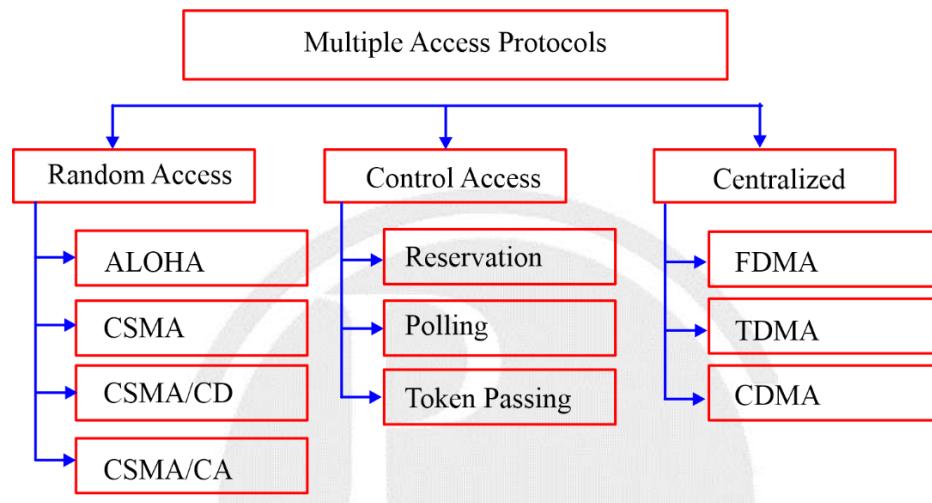


□□□



6

MEDIUM ACCESS CONTROL [MAC]

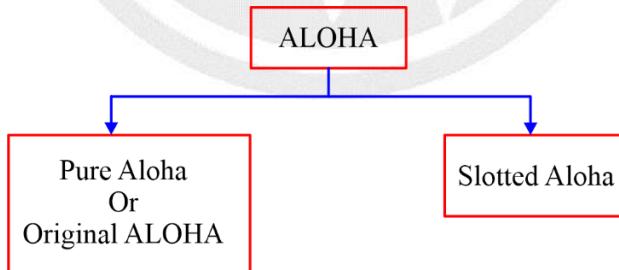


6.1 Aloha

Aloha was developed at university of Hawaii in 1970's.

It was designed for wireless LAN but it can be used in any shared medium.

Each station sends equal size frame.



Pure Aloha	Slotted Aloha
Any station transmits the data at any time.	Any station can transmit the data at the beginning of any time slot.
Vulnerable time in which collision may occur = $2 * T_f$ (T_f - Transmission time for single frame)	Vulnerable time in which collision may occur = T_f
Throughput of pure aloha = $G * e^{-2G}$	Throughput of slotted Aloha = $G * e^{-G}$
Maximum throughput $s_{max} = 18.4\%$ (When $G = 1/2$)	Maximum throughput $s_{max} = 36.8\%$ (When $G = 1$)
The main advantage of pure aloha is its simplicity in implementation	The main advantage of slotted aloha is that it reduces the number of collisions to half and doubles the throughput of pure aloha

6.2 CSMA (Carrier Sense multiple access)

CSMA requires that each station first sense the carrier before transmitting the data.

Vulnerable time for CSMA = Propagation time.

6.3 Persistence methods in CSMA

Persistent

Non-persistent

P-persistent

6.4 CSMA/CD (Carrier Sense multiple access/Collision Detection)

1. Minimum size of frame to detect the collision in Ethernet (CSMA/CD)

$$T_d \geq 2 * P_d + T_d (\text{Jam signal})$$

2. Backoff Algorithm

Waiting time = $K * \text{Slot duration}$

$$= K * \text{RTT}$$

$$= K * 2 * P_d$$

K is any random number between 0 to $2^n - 1$.

n is collision number (Collision number is respect to data packet).

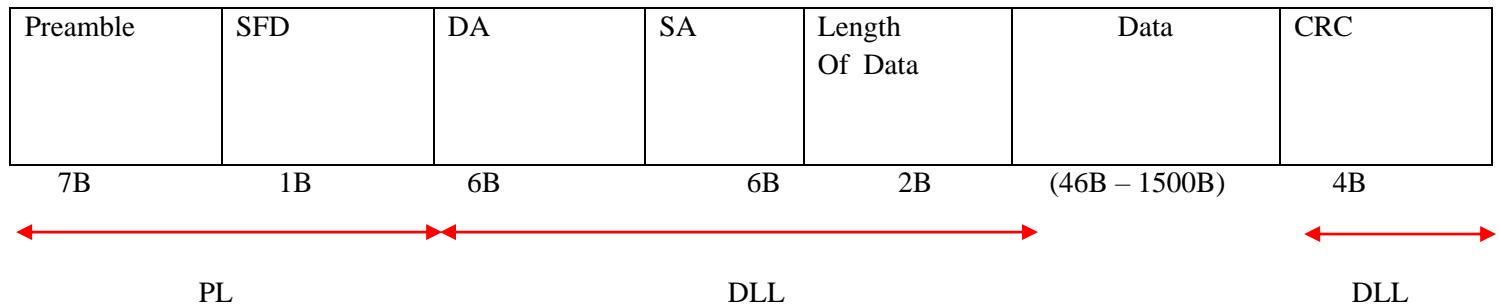
3. Efficiency in Ethernet (CSMA/CD)

$$\boxed{\eta = \frac{1}{1 + 6.44a}} \quad \text{or } \eta = \frac{\text{useful time}}{\text{Total time}} = \frac{T_d}{\text{Collision time} + T_d + P_d}$$

4. $P(1-P)^{N-1} \rightarrow$ Probability of success for single station (Throughput of Host)

$NP(1 - P)^{N-1} \rightarrow$ Probability of success for any station among all stations [Throughput of channel]

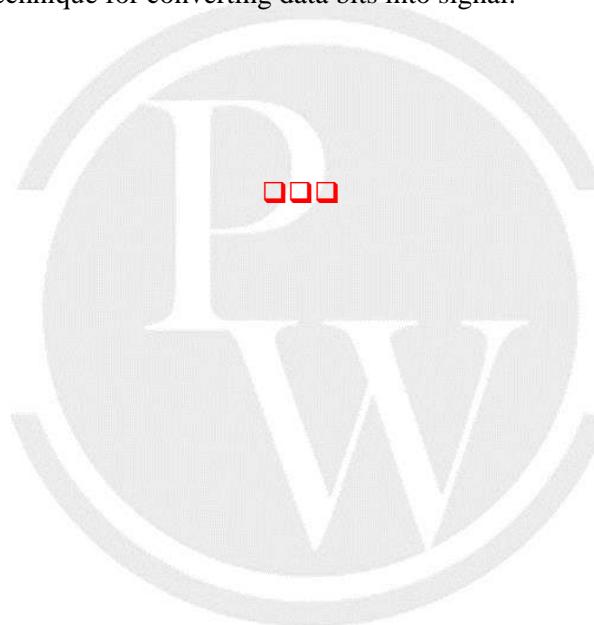
5. Ethernet Frame Structure:



Ethernet uses Manchester encoding technique for converting data bits into signal.

(Baud rate = 2 * bit rate)

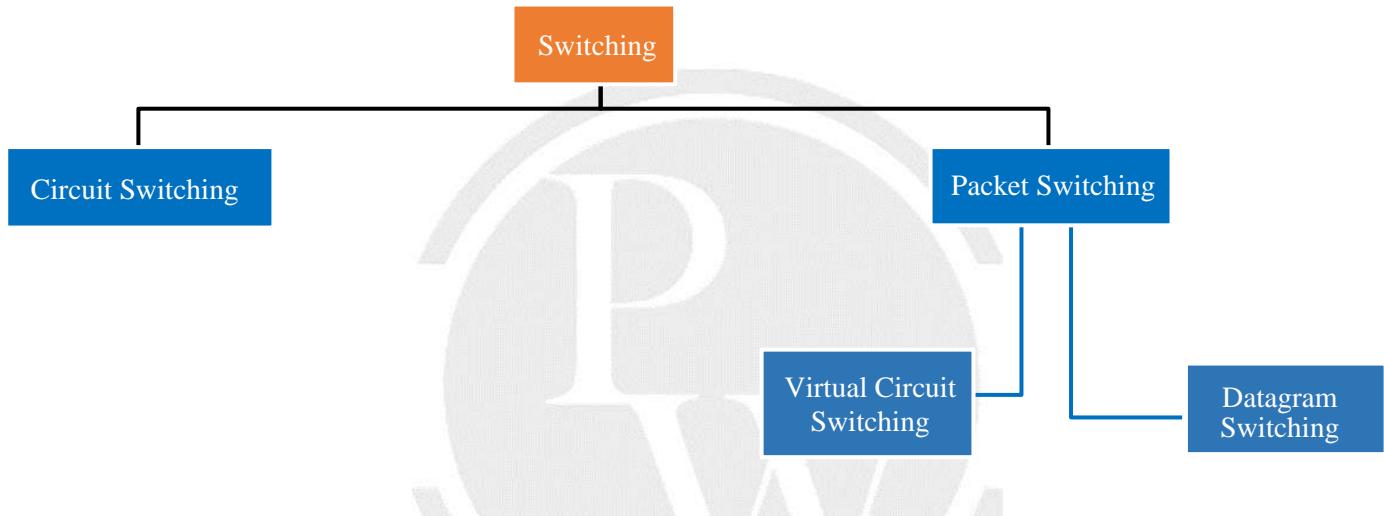
Bit rate = 1/2 baud rate



7

ROUTING ALGORITHMS, SWITCHING & IP SUPPORT PROTOCOL

7.1 Switching



Circuit Switching	Packet switching
(I) It has three phases-connection establishment, Data transfer and Connection termination.	It has only one phase-Data transfer
(2) Physical path between source and destination	No physical path
(3) All packets use same path	Packet may follow different path (travel independently)
(4) Reserves the entire bandwidth in advance	Does not reserve
(5) Bandwidth wastage	No Bandwidth wastage
(6) No store and forward transmission	Supports store and forward transmission
(7) Congestion can happen during connection establishment phase	Congestion can happen during data transfer phase

(8) It is reliable	Not reliable
(9) Better for sending large messages	Better for sending small messages
(10) Not fault tolerant technique	Fault tolerant technique
(11) Circuit switching is implemented at physical layer.	Packet switching is implemented at network layer
Total time= Setup time + $T_d + P_d + \text{Tear down time}$ $TT=S + \frac{L}{B} + X \cdot \frac{d}{V} + T$	For X Hop and N packet $\text{Total time}= X [T_d + P_d] + X - 1 [P_{rd} + Q_d] + N - 1 (T_d)$

7.2 Generalized Formula for optimal packet size(P)

M = Message size

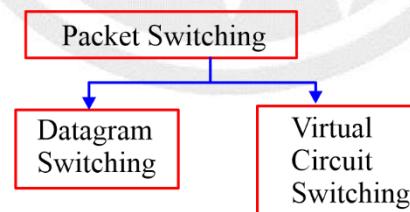
h = Header size

p = Payload/Packet data size

No. of Hops = X

$$p = \sqrt{\frac{Mh}{X-1}}$$

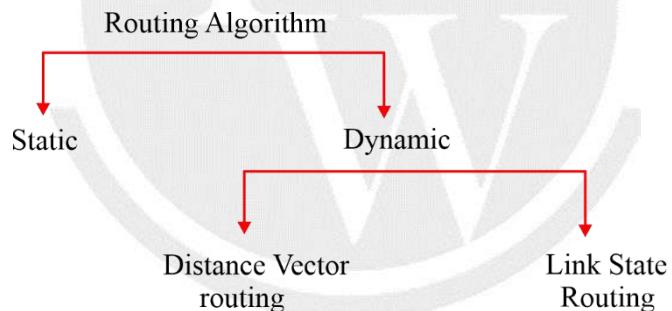
So optimum packet size P = p + h



Datagram Switching	Virtual circuit switching
(1) It is a connection less service.	It is connection-oriented service.
(2) All the packets may follow different path.	All the packet follows the same dedicated path.
(3) Data may appear out of order at the destination since the packets may follow different paths.	Data appears in order at the destination since all the packets take the same dedicated path.
(4) It is not reliable since packets may be discarded.	It is highly reliable.

(5) Cheap	Costly
(6) No resource reservation is done.	First packet reserves the resources (CPU, bandwidth and buffer) for the subsequent packets.
(7) All the packets require a global header which contains full information about the destination.	Only first packet requires a global header which identifies the path from one end to another end. All the following packets require a local header which identifies the path from hop to hop.
(8) The packets may be discarded at intermediate switches if sufficient resources are not available to process the packets.	The packets are never discarded at intermediate switches and immediately forwarded since resources are reserved for them.
(9) IP Networks uses datagram switching.	ATM (Asynchronous Transfer Mode) uses virtual circuit switching.
(10) Datagram switching is normally implemented at network layer.	Virtual circuit switching is normally implemented at data link layer.

7.3 Routing Algorithm



Distance vector Routing	Link state Routing
1. 1980's	1. 1990's
2. Bandwidth required is very less because we sent only distance vector packet.	2. Band width required is high because we sent entire link state packet
3. Local knowledge	3. Global knowledge
4. Bellman Ford algorithm	4. Dijkstra algorithm
5. Traffic is very less	5. Traffic is very high

6. Convergence is very low	6. Convergence is faster
7. Count to infinity Problem	7. No problem of count to infinity
8. Persistent Loops	8. Transient Loops
9. RIP	9. OSPF

The maximum Hop count allowed For RIP is 15 and Hop count of 16 is considered as Destination unreachable.

Note:

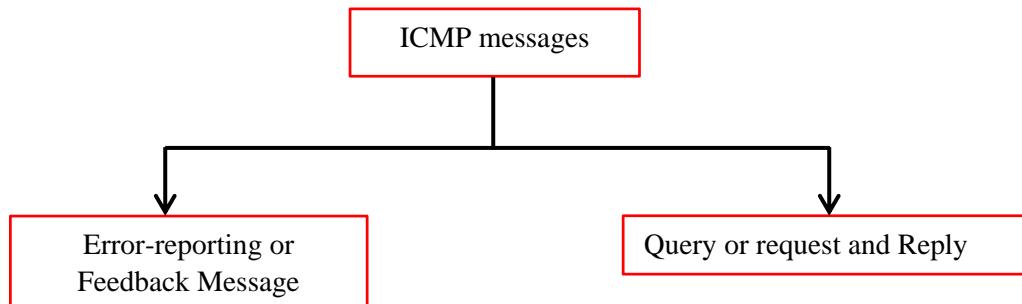
RIP uses UDP as its transport protocol with the port number – 530

7.4 IP Support Protocol

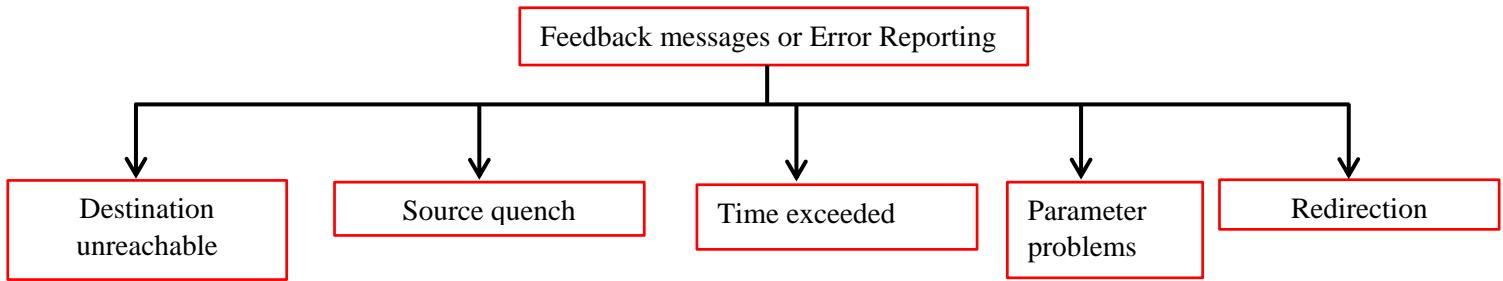
7.4.1 ARP

1. Address Resolution Protocol(ARP) is used to find the MAC(Media Access Control) address of a device from its IP address.
2. **ARP request:** ARP request is broadcasting
3. **ARP response/reply:** ARP reply is unicasting.

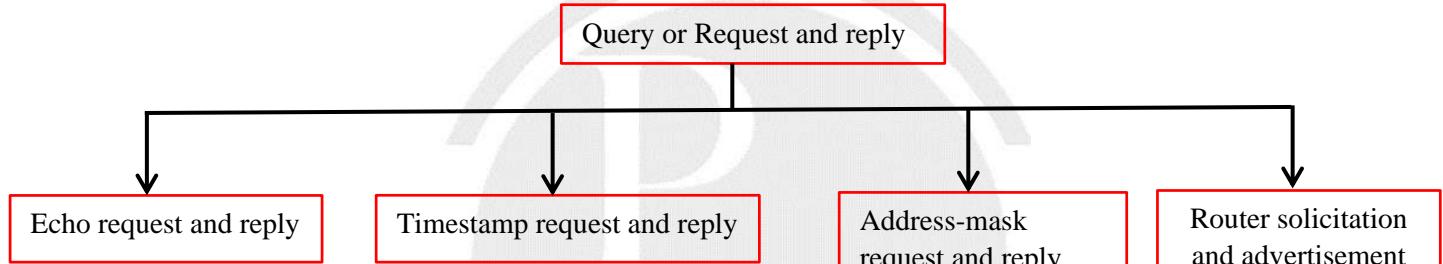
7.4.2 Internet Control Message Protocol (ICMP)



7.4.3 Internet Control Message Protocol (ICMP)



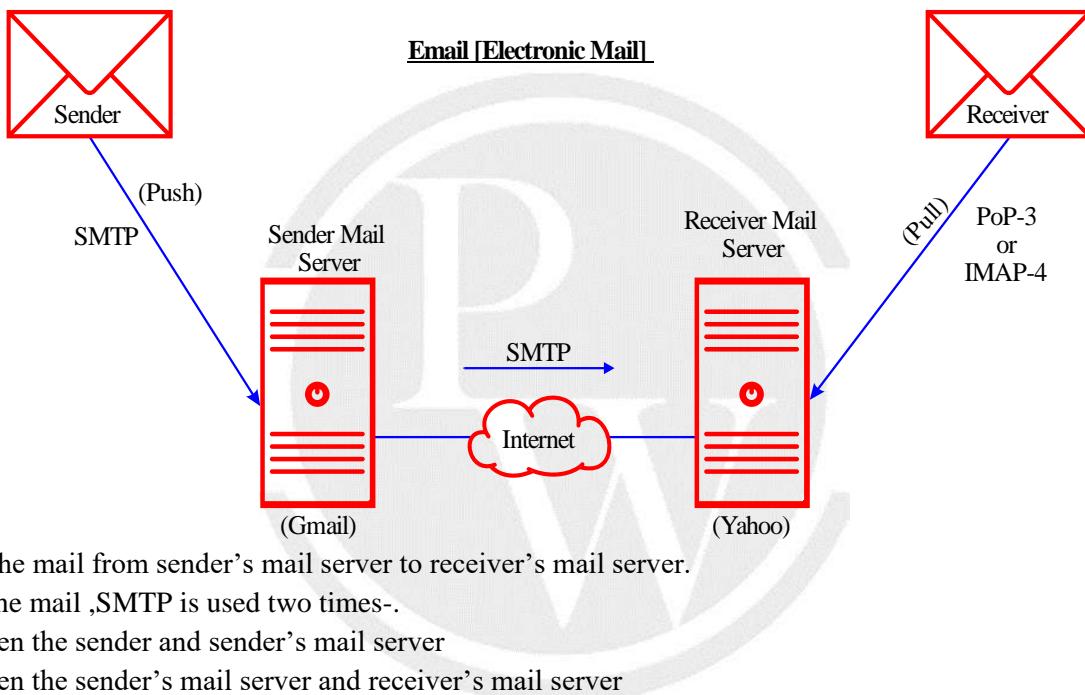
7.4.4 Internet Control Message Protocol (ICMP)



8

APPLICATION LAYER PROTOCOL

8.1 E-MAIL



SMTP transfer the mail from sender's mail server to receiver's mail server.

While sending the mail ,SMTP is used two times-.

- Between the sender and sender's mail server
- Between the sender's mail server and receiver's mail server

8.1.1 To receive or download the email,

Another protocol is needed between the receiver mail server and receiver.

The most commonly used protocols are POP3 and IMAP4.

8.1.2 SMTP(Simple mail transfer protocol)

- The objective of SMTP is to transfer the email reliably and efficiently.
- It uses port number-25 at TCP.
- In SMTP there are two components:
 - User Agent (UA)
 - Mail transfer Agent (MTA)
- User Agent prepares the message, creates the envelope and put the message in the envelope.
- Mail transfer Agent transfer the mail across the internet i.e. Actual mail transfer is done through MTA.
- To send mail, system must have a client MTA and to receive the mail. it must have a server MTA.

7. SMTP is text based protocol.
8. With the help of SMTP & POP we can send only text messages.
9. SMTP can only handle the message containing 7 bit ASCII text.
10. SMTP cannot transfer other types of data like images, video, audio, etc.
11. SMTP cannot transfer binary files or executable files.
12. SMTP cannot transfer the text data for the language other than English (such as French, Japanese, and Chinese etc.).
13. Only SMTP is not sufficient to send binary files or to send videos or audio so we require MIME (Multipurpose Internet mail extension).
14. MIME is a supplementary protocol that allows non-ASCII data to send through SMTP.
15. MIME is a set of software function that transforms non-ASCII data to ASCII data or viceversa.
16. MIME is used to convert non text data to text data and text data to non text data.
17. SMTP is stateless protocol. It does not maintain any information of user. If an e-mail is asked to be sent twice, then server resends it without saying that e-mail has already been sent.
18. SMTP is a connection-oriented protocol.
19. SMTP uses persistent TCP connections, so it can send multiple e-mail at once.
20. SMTP is an “In-Band” protocol.
21. SMTP is used for Push the e-mail.
22. SMTP Pushes the mail from client to server on other hand, It needs a pull protocol(Download).
23. POP3 and IMAP4 are used for Pulling the e-mail.

8.2 POP3(Post office Protocol version-3)

1. It is a message access protocol.
2. It is a pull protocol.
3. POP3 uses port number-110 at TCP.
4. POP3 is a connection-oriented protocol.
5. POP3 uses persistent TCP connection.
6. POP3 is a stateful protocol.
7. POP3 is an “In-Band” protocol.
8. POP3 does not allow users to partially check the content of the mail before downloading.
9. POP3 does not allow user to organize the mail on the mail server.

8.2.1 IMAP-4(Internet Mail Access Protocol version-4)

1. IMAP-4 is similar to POP3 but it has more features. IMAP-4 is more powerful and more complex.
2. IMAP-4 provides the following extra functions.
3. A user checks the email header prior to downloading.
4. A user can search the content of the email for a specific string of characters prior to downloading.
5. A user can partially download the email.

6. A user can create, delete, or rename the mail box on the mail server.
7. A user can create a hierarchy of mailbox in a folder for email storage.

8.2.2 Characteristics of IMAP

1. IMAP is a pull protocol.
2. IMAP uses port number-143 at TCP.
3. IMAP is a connection-oriented protocol.
4. IMAP uses persistent TCP connection.
5. IMAP is a tasteful protocol.
6. IMAP is an “In-Band” Protocol.

POP3	IMAP
(1) Mails can only be accessed from a single device.	Mails can be accessed from multiple device.
(2) Download the email from server to a single computer and the copy at the server is deleted.	The email message is stored on the mail server itself.
(3) User cannot organize the mails in the mail box of the mail server.	User can organize mails on the mail server.
(4) It does not allow user to sync emails.	It allows user to sync their emails.
(5) It is unidirectional i.e all the changes made on a device does not effect the content present on the server.	It is bidirectional i.e all the changes made on server or device are made on the other side too.

8.3 File Transfer protocol (FTP)

8.3.1 FTP (File Transfer Protocol)

1. File transfer protocol is a standard internet protocol for transferring files b/w computers over TCP/IP connection.
2. It uses port number - 20 & 21 on TCP.
3. It has two types of connection
 - (i) Control connection (port number. - 21)
 - (ii) Data connection (port number - 20)
4. Control connection remains connected during the entire interactive FTP session.
5. The data connection is opened and closed for each file transfer activity.
6. When user starts an FTP session, the control connection opens. While the control connection is open, the data connection can be opened and closed multiple times if several files are transferred.
7. FTP uses persistent TCP connections for control information.
8. FTP uses Non-persistent TCP connections for data information.
9. FTP is a connection-oriented protocol.
10. FTP is an “out of band” protocol as data and control information flow over different connection.
11. Some protocols send their request and data in the same TCP connection for this reason they are called as In-bound protocol.
12. HTTP & SMTP are In-Band protocol.
13. FTP is state full protocol.

8.3.2 Transmission mode

FTP can transfer a file across the data connection using one of the following three transmission modes.

- (i) Stream mode
- (ii) Block mode
- (ii) Compressed mode

8.3.3 File Type

FTP can transfer one of the following file types across the data connection:

- (i) ASCII file
- (ii) EBCDIC file (File format used by IBM)
- (ii) Image file

8.3.4 Data structure

FTP can transfer a file across the data connection using one of the following interpretation of the structure of the data:

- (i) File structure
- (ii) Record structure
- (iii) Page structure

8.4 HTTP Protocol

1. HTTP protocol is used mainly to access data on world wide web (www).
2. It is client server protocol using port number - 80 on TCP.
3. HTTP is “In-Band” protocol i.e. both request and data we will send only in one connection.
4. HTTP is a stateless protocol i.e. It does not maintain any information of user.
5. There are two types of HTTP protocol
 - (i) Non persistent (1.0)
 - (ii) Persistent (1.1)

8.5 Non Persistent (1.0)

In a Non persistent connection one TCP connection is made for each request/response. This strategy follows the following steps :-

- (i) The client opens a TCP connection and sends a request.
- (ii) Server sends the response and closes the connection.
- (iii) In this strategy , If a file contains link to N-different pictures in different files(all located on same server) the connection must be opened and closed N+1 times.

8.6 Persistent (1.1)

1. In a persistent connection the server leaves the connection open for more request after sending a response.
2. The server closes the connection at the request of client or time out has been reached.

8.6.1 Important Table

SHORT TRICK	DNS	HTTP	SMTP	POP	IMAP	FTP
Stateful/ Stateless	Stateless	Stateless	Stateless	Stateful	Stateful	Stateful
Transport Protocol Used	UDP	TCP	TCP	TCP	TCP	TCP
Connectionless/ Connection oriented	Connection less	Connection less	Connection oriented	Connection oriented	Connection oriented	Connection oriented
Persistent/Non-persistent	Non-persistent	HTTP 1.0 is non persistent HTTP 1.1 is persistent.	Persistent	Persistent	Persistent	Control connection is persistent. Data connection is non-persistent.
Push/Pull	-	-	Push	Pull	Pull	Can't
Port Number Used	53	80	25	110	143	20 for data connection. 21 for control connection.
In band/ Out-of-band	In band	In band	In band	In band	In band	Out-of- band

Application	Port Number.	Transport Protocol
DNS	53	UDP
HTTP	80	TCP
FTP	20 (Data connection) 21 (Control connection)	TCP
SMTP	25	TCP
POP	110	TCP
SNMP	161, 162	UDP
TFTP	69	UDP
IMAP	143	TCP
Telnet	23	TCP
DHCP	67 (DHCP Server) 68 (DHCP Client)	UDP

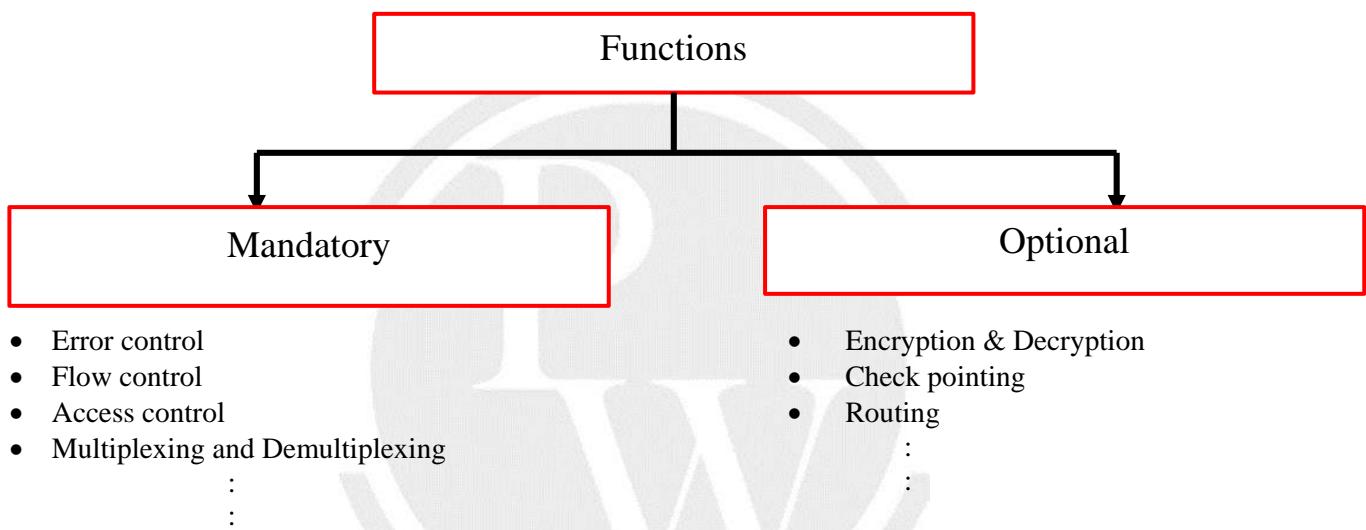
8.6.2 Commands

HTTP	FTP	SMTP
GET	USER	HELO
HEAD	PASS	MAIL FROM
PUT	ACCT	RCPT TO
POST	CWD	DATA
TRACE	REIN	QUIT
DELETE	QUIT	RSET
CONNECT	PORT*	VRFY
OPTIONS	PASV	NOOP
	TYPE	TURN
	MODE	EXPN
	PROMPT	HELP
	STRU	SEND FROM
		SMOL FROM
		SMAL FROM

9

OSI AND TCP/IP PROTOCOL STACK

9.1 Functions of Computer Network



9.2 OSI/ISO

OSI : Open systems Interconnection model.

ISO : International Standards Organization. It is a multinational body dedicated to worldwide agreement on international standard.

9.2.1 OSI Mode

- This model has been proposed by ISO.
- An open system is a set of protocols that allows any two different systems to communicate regardless of their underlying architecture (Hardware/Software).
- The purpose is to show how to facilitate communication between different systems without requiring changes to the logic of the underlying hardware & software.
- This model has got 7 separate but related layers.

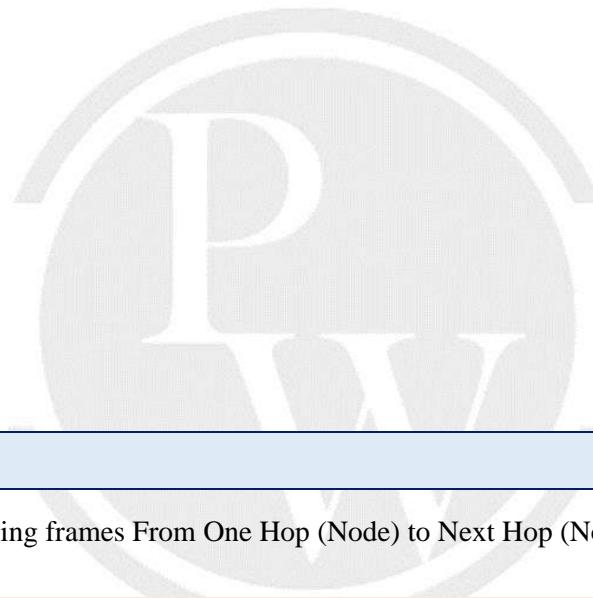
9.3 Functions of Physical layer

9.3.1 Physical Layer

Physical Layer is responsible for movement of individuals bits from one Hop to next Hop.

9.3.2 Functions of physical Layer

1. It is used to define electrical, mechanical, functional and procedural characteristic of physical link.
(physical Link)
Copper → Electrical signal
Fiber → Light signal
Wireless comm. → Electromagnetic signal.
2. It defines transmission mode:
 - a. Simplex
 - b. Half duplex
 - c. Full duplex
3. It defines topology configuration:
 - Bus topology
 - Star topology
 - Mesh Topology
 - Tree Topology
4. It is totally Hardware layer.
5. It defines link configuration:
 - i Point to Point Link
 - ii Broadcast Link
6. It defines Encoding.
7. Bits Synchronization.
8. Bit rate control.



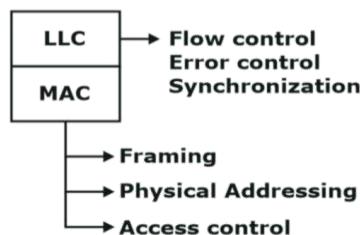
9.4 Data Link Layer

Data link layer is responsible for moving frames From One Hop (Node) to Next Hop (Node).

9.4.1 Function of data link layer

1. Flow control
2. Error control
3. Access control
4. Framing
5. Physical Addressing

Data Link Layer is divided into two parts



9.5 Network Layer

The network layer is responsible for the delivery of individual packet from source to destination.

9.5.1 Function of Network Layer

1. Host to Host connectively
2. Logical Addressing
3. Switching
4. Routing
5. Fragmentation
6. Congestion control:

9.6 Transport Layer

Transport Layer is responsible for process to process delivery. A process is an application program running on a host.

9.6.1 Function of Transport Layer

1. End to end connectively
2. Service point Addressing
3. Flow control
4. Error control
5. Segmentation and Reassembly
6. Congestion control
7. Connection Control
8. Multiplexing and Demultiplexing.

9.7 Session Layer

Session layer also known as **network dialog controller**. It establishes, maintains, synchronizes and terminates the interaction b/w sender and receiver.

9.7.1 Function of Session Layer

1. Authentication & Authorization
2. Check point or synchronization
3. Dialog control

9.8 Presentation Layer

This layer take care of syntax and semantics of the information exchange in between two communicating systems.

9.8.1 Function of Presentation Layer

1. Character translation
2. Encryption/Decryption
3. Compression

9.9 Application Layer

Application Layer is responsible for providing services to users. Users such as:

1. Mail services
2. File sharing
3. File transfer and many more

9.10 TCP/IP Model

This mode has got 5 different layer

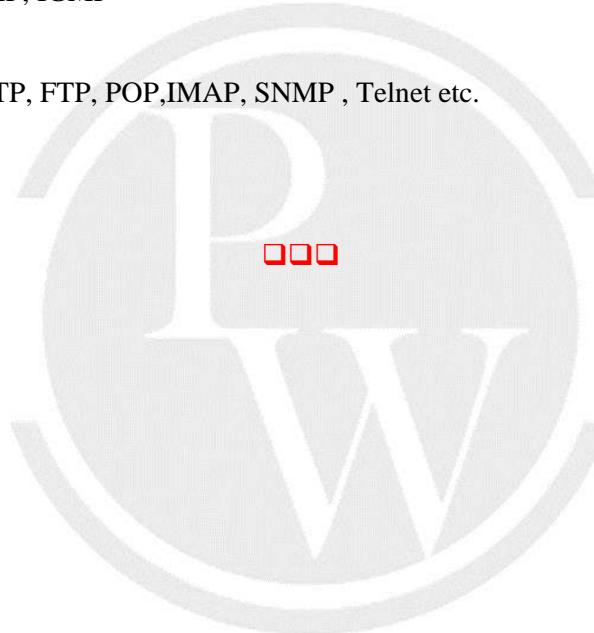
Physical : No specific protocol

Data Link : No specific protocol

Network : ARP, RARP, ICMP, IGMP

Transport : TCP, UDP,SCTP

Application : DNS, SMTP, HTTP, FTP, POP,IMAP, SNMP , Telnet etc.



GATE Exam 2024?



PARAKRAM BATCH

ME | CE | EE | EC | CS & IT

- Live Classes & Recorded Videos
- DPPs & Solutions
- Doubt Solving
- Batches are available in **Hindi**

Weekday & Weekend
Batches Available

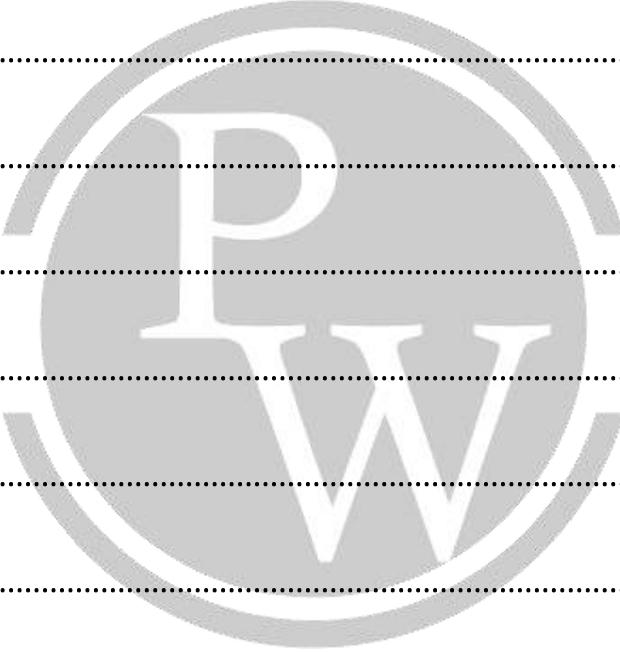


JOIN NOW!

GENERAL APTITUDE

General Aptitude

INDEX

- 
1. Percentages 12.1 – 12.5
 2. Averages & Ages 12.6
 3. Profit & Loss 12.7 – 12.9
 4. Ratio and Proportions 12.10
 5. Time and Distance 12.11 – 12.12
 6. Time and Work 12.13 – 12.14
 7. Clocks 12.15 – 12.16
 8. Calendars 12.17 – 12.19
 9. Blood Relations 12.20
 10. Directions 12.21 – 12.22
 11. Data Interpretation 12.23 – 12.24
 12. Data Sufficiency 12.25

1

PERCENTAGES

1.1. Understanding Percentages

The word percent can be understood as follows:

Per cent ⇒ for every 100.

So, when percentage is calculated for any value, it means that you calculate the value for every 100 of the reference value.

Why Percentage?

Percentage is a concept evolved so that there can be a uniform platform for comparison of various things. (Since each value is taken to a common platform of 100.)

Example:

- To compare three different students depending on the marks they scored we cannot directly compare their marks until we know the maximum marks for which they took the test. But by calculating percentages they can directly be compared with one another.
- Before going deeper into the concept of percentage, let u have a look at some basics and tips for faster calculations:

1.2. Calculation of Percentage

$$\text{Percentage} = \left(\frac{\text{Value}}{\text{Total value}} \right) \times 100$$

Example: 50 is what % of 200?

$$\text{Solution: } \text{Percentage} = \left(\frac{50}{200} \right) \times 100 = 25\% .$$

1.2.1. Calculation of Value:

$$\text{Value} = \left(\frac{\text{Percentage}}{100} \right) \times \text{total value}$$

Example: What is 20% of 200?

$$\text{Solution: } \text{Value} = \left(\frac{20}{100} \right) \times 200$$

Note: Percentage is denoted by “%”, which means “/100”.

Example: What is the decimal notation for 35%?

Solution: $35\% = \frac{35}{100} = 0.35$.

For faster calculations we can convert the percentages or decimal equivalents into their respective fraction notations.

1.3. Percentages – Fractions Conversions:

The following is a table showing the conversions of percentages and decimals into fractions:

Percentage	Decimal	Fraction
10%	0.1	$\frac{1}{10}$
12.5%	0.125	$\frac{1}{8}$
16.66%	0.1666	$\frac{1}{6}$
20%	0.2	$\frac{1}{5}$
25%	0.25	$\frac{1}{4}$
30%	0.3	$\frac{3}{10}$
33.33%	0.3333	$\frac{1}{3}$
40%	0.4	$\frac{2}{5}$
50%	0.5	$\frac{1}{2}$
60%	0.6	$\frac{3}{5}$
62.5%	0.625	$\frac{5}{8}$
66.66%	0.6666	$\frac{2}{3}$

Similarly we can go for converting decimals more than 1 from the knowledge of the above cited conversions as follows:

We know that $12.5\% = 0.125 = \frac{1}{8}$

Then, $1.125 = \frac{[8(1)+1]}{8} = \frac{9}{8}$ (i.e., the denominator will add to numerator once, denominator remaining the same.)

Also, $2.125 = \frac{[8(2)+1]}{8} = \frac{17}{8}$ (here the denominator is added to numerator twice)

$3.125 = \frac{[8(3)+1]}{8} = \frac{25}{8}$ and so on.

Thus we can derive the fractions for decimals more than 1 by using those less than 1.

We will see how use of fractions will reduce the time for calculations:

Example: What is 62.5% of 320?

Solution: Value = $\left(\frac{5}{8}\right) \times 320$ (since $62.5\% = \frac{5}{8}$) = 200.

1.4. Percentage Change

A change can be of two types – an increase or a decrease.

When a value is changed from initial value to a final value,

$$\% \text{ change} = (\text{Difference between initial and final value}/\text{initial value}) \times 100$$

Example: If 20 changes to 40, what is the % increase?

Solution: % increase = $\frac{(40 - 20)}{20} \times 100 = 100\%$.

Note:

1. If a value is doubled the percentage increase is 100.
2. If a value is tripled, the percentage change is 200 and so on.

1.5. Percentage Difference

$$\% \text{ Difference} = (\text{Difference between values}/\text{value compared with}) \times 100.$$

Example: By what percent is 40 more than 30?

Solution: % difference = $\frac{(40 - 30)}{30} \times 100 = 33.33\%$

(Here 40 is compared with 30. So 30 is taken as denominator)

Example: By what % is 60 more than 30?

Solution: % difference = $\frac{(60 - 30)}{30} \times 100 = 100\%$.

(Here is 60 is compared with 30.)

Hint: To calculate percentage difference the value that occurs after the word “than” in the question can directly be used as the denominator in the formula.

1.6. Important Points to Note

1. When any value increases by

- (a) 10%, it becomes 1.1 times of itself. (since $100+10 = 110\% = 1.1$)
- (b) 20%, it becomes 1.2 times of itself.
- (c) 36%, it becomes 1.36 times of itself.
- (d) 4%, it becomes 1.04 times of itself.

Thus we can see the effects on the values due to various percentage increases.

2. When any value decreases by

- (a) 10%, it becomes 0.9 times of itself. (Since $100 - 10 = 90\% = 0.9$)
- (b) 20%, it becomes 0.8 times of itself
- (c) 36%, it becomes 0.64 times of itself
- (d) 4%, it becomes 0.96 times of itself.

Thus we can see the effects on a value due to various percentage decreases.

Note:

1. When a value is multiplied by a decimal more than 1 it will be increased and when multiplied by less than 1 it will be decreased.
2. The percentage increase or decrease depends on the decimal multiplied.

Example: $0.7 \Rightarrow 30\%$ decrease, $0.67 \Rightarrow 33\%$ decrease, $0.956 \Rightarrow 4.4\%$ decrease and so on.

Example: When the actual value is x , find the value when it is 30% decreased.

Solution: 30% decrease $\Rightarrow 0.7 x$.

Example: A value after an increase of 20% became 600. What is the value?

Solution: $1.2x = 600$ (since 20% increase)

$$\Rightarrow x = 500.$$

Example: If 600 is decrease by 20%, what is the new value?

Solution: New value $= 0.8 \times 600 = 480$. (Since 20% decrease)

Thus depending on the decimal we can decide the % change and vice versa.

Example: When a value is increased by 20%, by what percent should it be reduced to get the actual value?

Solution: (It is equivalent to 1.2 reduced to 1 and we can use % decrease formula)

$$\% \text{ decrease} = \frac{(1.2 - 1)}{1.2} \times 100 = 16.66\%.$$

3. When a value is subjected multiple changes, the overall effect of all the changes can be obtained by multiplying all the individual factors of the changes.

Example: The population of a town increased by 10%, 20% and then decreased by 30%. The new population is what % of the original?

Solution: The overall effect = $1.1 \times 1.2 \times 0.7$ (Since 10%, 20% increase and 30% decrease)

$$= 0.924 = 92.4\%.$$

Example: Two successive discounts of 10% and 20% are equal to a single discount of ____

Solution: Discount is same as decrease of price.

So, decrease = $0.9 \times 0.8 = 0.72 \Rightarrow 28\% \text{ decrease}$ (Since only 72% is remaining).

2

AVERAGES & AGES

2.1. What is Average?

The concept of average is equal distribution of the overall value among all the things or persons present there. So the formula for finding the average is as follows:

$$\text{Average, } A = \frac{\text{Total of all things, } T}{\text{Number of things, } N}$$

Therefore, Total, $T = AN$

If any person joins a group with more value than the average of the group then the overall average increases. This is because the value in excess than the average will also be distributed equally among all the members.

Similarly when any value less than the average joins the group the overall group decreases as the deficit is divided equally among all the people present there.

Example:

Consider three people A, B and C with total of Rs. 30/-. Their average becomes Rs. 10/- for each. If another person D joins them with Rs. 50/- then he has Rs. 40/- more than actual average of Rs. 10/-.

So this Rs. 40/- will get distributed among those four and each gets Rs. 10/-. Thus the average becomes Rs. 20/- each.

Example:

The average age of a class of 30 students is 12. If the teacher is also included the average becomes 13 years. Find the teacher's age.

Solution:

- When the teacher is included there are totally 31 members in the class and the average is increased by 1 year. This means that everyone got 1 extra year after distributing the extra years of the teacher.
- So extra years of the teacher are as follow: $31 \times 1 = 31$ years.
- Age of the teacher = actual avg + extra years = $12 + 31 = 43$ years.



3

PROFIT AND LOSS

3.1. What is Profit?

When a person does a business transaction and gets more than what he had invested, then he is said to have profit. The profit he gets will be equal to the additional money he gets other than his investment.

So profit can be understood as the extra money one gets other than what he had invested.

Example: A person bought an article for Rs. 100 and sold it for Rs. 120. Then he got Rs. 20 extra and so his profit is Rs. 20.

3.2. What is Loss?

When a person gets an amount less than what he had invested, then he is said to have a loss. The loss will be equal to the deficit he got than the investment.

Example: A person bought an article at Rs. 100 and sold it for Rs. 90. Then he got a deficit of Rs. 10 and so his loss is Rs. 10.

3.3. Cost Price (CP)

- The money that the trader puts in his business is called Cost Price. The price at which the articles are bought is called Cost Price.
- In other words, Cost Price is nothing but the investment in the business.

3.4 Selling Price (SP)

- The price at which the articles are sold is called the Selling Price. The money that the trader gets from the business is called Selling Price.
- In other words, Selling Price is nothing but the returns from a business.

3.5. Marked/Market/List Price (MP):

- The price that a trader marks or lists his articles to is called the Marked Price.
- This is the only price known to the customer.

3.6. Discount

The waiver of cost from the Marked Price that the trader allows a customer is called Discount.

Note:

1. Profit or loss percentage is to be applied always to the Cost Price only.
2. Discount percentage is to be applied always to the Marked Price only.

3.7. Relationship Among CP, SP and MP:

A trader adds his profit to the investment and sells it at that increased price.

Also he allows a discount on Marked Price and sells at the discounted price.

So, we can say that,

- $SP = CP + \text{Profit}$. (CP applied with profit is SP)
- $SP = MP - \text{Discount}$. (MP applied with discount is SP)

3.8. Understanding Profit and Loss:

So, by now we came to know that if CP is increased and sold it would result in profit and vice versa.

Also whatever increase is applied to CP, that increase itself is the profit.

For Rs. 10 profit, CP is to be increased by RS. 10 and the increased price becomes SP.

For 10% profit, CP is to be increased by 10% and it is the SP.

(From previous chapter we know that any value increased by 10% becomes 1.1 times.)

So, for 10% profit, CP increased by 10% $\Rightarrow 1.1CP = SP$.

- $SP = 1.1CP \Rightarrow \frac{SP}{CP} = 1.1 \Rightarrow 10\% \text{ profit}$
- $SP = 1.07CP \Rightarrow \frac{SP}{CP} = 1.07 \Rightarrow 7\% \text{ profit}$
- $SP = 1.545CP \Rightarrow \frac{SP}{CP} = 1.545 \Rightarrow 54.5\% \text{ profit and so on.}$

Similarly,

- $SP = 0.9CP \Rightarrow \frac{SP}{CP} = 0.9 \Rightarrow 10\% \text{ loss (Since 10\% decrease)}$
- $SP = 0.7CP \Rightarrow \frac{SP}{CP} = 0.7 \Rightarrow 24\% \text{ loss and so on.}$

So, to calculate profit % or loss %, it is enough for us to find the ratio of SP to CP.

Note:

1. If $SP/CP > 1$, it indicates profit.
2. If $SP/CP < 1$, it indicates loss.

3.9. Multiple Profits or Losses

A trader may sometimes have multiple profits or losses simultaneously. This is equivalent to having multiple changes and so all individual changes are to be multiplied to get the overall effect.

Examples: A trader uses a 800gm weight instead of 1 kg. Find his profit %.

Solution: (He is buying 800 gm but selling 1000 gm.)

So, CP is for 800 gm and SP is for 1000 gm.)

$$\frac{SP}{CP} = \frac{1000}{800} = 1.25 \Rightarrow 25\% \text{ profit.}$$

Examples: A trader uses 1 kg weight for 800 gm and increases the price by 20%. Find his profit/loss %.

Solution: 1 kg weight for 800 gm \Rightarrow loss (decrease) $\Rightarrow 800/1000 = 0.8$

20% increase in price \Rightarrow profit (increase) $\Rightarrow 1.2$

So, net effect $= (0.8) \times (1.2) = 0.96 \Rightarrow 4\% \text{ loss.}$

Examples: A milk vendor mixes water to milk such that he gains 25%. Find the percentage of water in the mixture.

Solution: To gain 25%, the volume has to be increased by 25%.

So, for 1 lt of milk, 0.25 lt of water is added \Rightarrow total volume = 1.25 lt

$$\% \text{ of water} = \frac{0.25}{1.25} \times 100 = 20\% .$$

Examples: A trader bought an item for Rs. 200. If he wants a profit of 22%, at what price must he sell it?

Solution: CP=200, Profit = 22%.

So, $SP = 1.22CP = 1.22 \times 200 = 244/-$

Examples: A person buys an item at Rs. 120 and sells to another at a profit of 25%. If the second person sells the item to another at Rs. 180, what is the profit % of the second person?

Solution: SP of 1st person = CP of 2nd person = $1.25 \times 120 = 150$.

SP of 2nd person = 180.

$$\text{Profit \%} = \frac{SP}{CP} = \frac{180}{150} = 1.2 \Rightarrow 20\% .$$



4

RATIOS AND PROPORTIONS

4.1. What is a Ratio?

A ratio is a representation of distribution of a value present among the persons present and is shown as follows:

If a total is divided among A, B and C such that A got 4 parts, B got 5 parts and C got 6 parts then it is represented in ratio as A:B:C = 4:5:6.

So, 4:5:6 means that the total value is divided into $4+5+6 = 15$ equal parts and then distributed as per the ratio.

Example 1: Divide Rs. 580 between A and B in the ratio of 14:15.

Solution: A:B = 14:15 \Rightarrow 580 is divided into 29 equal parts \Rightarrow each part = Rs. 20.

So A's share = 14 parts = $14 \times 20 =$ Rs. 280

B's share = 15 parts = Rs. 300.

Example 2: If A:B = 2:3 and B:C = 4:5 then find A:B:C.

Solution: To combine two ratios the proportions common for them shall be in equal parts. Here the common proportion is B for the given ratios.

Making B equal in both ratios they become 8:12 and 12:15 \Rightarrow A:B:C = 8:12:15.

Example 3: Three numbers are in the ratio of 3: 4 : 8 and the sum of these numbers is 975. Find the three numbers.

Solution: Let the numbers be $3x$, $4x$ and $8x$. Then their sum = $3x + 4x + 8x = 15x = 975 \Rightarrow x = 65$.

So the numbers are $3x = 195$, $4x = 260$ and $8x = 520$.

Example 4: Two numbers are in the ratio of 4 : 5. If the difference between these numbers is 24, then find the numbers.

Solution: Let the numbers be $4x$ and $5x$. Their difference = $5x - 4x = x = 24$ (given).

So the numbers are $4x = 96$ and $5x = 120$.

Example 5: Given two numbers are in the ratio of 3 : 4. If 8 is added to each of them, their ratio is changed to 5 : 6. Find two numbers.

Solution: Let the numbers be a and b .

$$A:B = 3:4 \Rightarrow \frac{A}{B} = \frac{3}{4}. \text{ Also, } \frac{(A+8)}{(B+8)} = \frac{5}{6}.$$

Solving we get, $A=12$ and $B = 16$.



5

TIME AND DISTANCE

5.1. Speed

We have the relation between speed, time and distance as follows:

$$\text{Speed} = \frac{\text{Distance}}{\text{Time}}$$

So the distance covered in unit time is called speed.

This forms the basis for Time and Distance. It can be re-written as Distance = Speed X Time or

$$\text{Time} = \frac{\text{Distance}}{\text{Speed}}.$$

5.1.1. Units of Speed

The units of speed are kmph (km per hour) or m / s.

$$1 \text{ kmph} = \frac{5}{18} \text{ m/s}$$

$$1 \text{ m/s} = \frac{18}{5} \text{ kmph}$$

5.1.2. Average Speed

When the travel comprises of various speeds then the concept of average speed is to be applied.

$$\text{Average Speed} = \frac{\text{Total distance covered}}{\text{Total time of travel}}$$

Note: In the total time above, the time of rest is not considered.

Example 1: If a car travels along four sides of a square at 100 kmph, 200 kmph, 300 kmph and 400 kmph find its average speed.

Solution: Average Speed = $\frac{\text{Total distance}}{\text{Total time}}.$

Let each side of square be x km. Then the total distance = $4x$ km.

The total time is sum of individual times taken to cover each side.

To cover x km at 100 kmph, time = $\frac{x}{100}$.

For the second side time = $\frac{x}{200}$.

Using this we can write average speed = $\frac{4x}{\left(\frac{x}{100} + \frac{x}{200} + \frac{x}{300} + \frac{x}{400}\right)} = 192$ kmph.

Example 2: A man if travels at $\frac{5}{6}$ th of his actual speed takes 10 min more to travel a distance. Find his usual time.

Solution: Let s be the actual speed and t be the actual time of the man.

Now the speed is $\left(\frac{5}{6}\right)s$ and time is $(t+10)$ min. But the distance remains the same.

So distance 1 = distance 2 $\Rightarrow s \times t = \left(\frac{5}{6}\right)s \times (t+10) \Rightarrow t = 50$ min.

Example 3: If a person walks at 30 kmph he is 10 min late to his office. If he travels at 40 kmph then he reaches to his office 5 min early. Find the distance to his office.

Solution: Let the distance to his office be d . The difference between the two timings is given as 15 min = $\frac{1}{4}$ hr.

Now if d km are covered at 30 kmph then time = $d/30$. Similarly second time = $d/40$.

$$\text{So, } \frac{d}{30} - \frac{d}{40} = \frac{1}{4} \Rightarrow d = 30 \text{ km.}$$

Note: When two objects move with speeds s_1 and s_2

- In opposite directions their combined speed = $s_1 + s_2$
- In same direction their combined speed = $s_1 - s_2$.

Example 4: Two people start moving from the same point at the same time at 30 kmph and 40 kmph in opposite directions. Find the distance between them after 3 hrs.

Solution: Speed = $30 + 40 = 70$ kmph (since in opposite directions)

Time = 3 hrs

So distance = speed \times time = $70 \times 3 = 210$ km.

Example 5: A starts from X to Y at 6 am at 40 kmph and at the same time B starts from Y to X at 50 kmph. When will they meet if X and Y are 360 km apart?

Solution: Distance = 360 km, Speed = $40 + 50 = 90$ kmph.

$$\text{Time} = \frac{\text{distance}}{\text{speed}} = \frac{360}{90} = 4 \text{ hrs from 6 am} \Rightarrow 10 \text{ am.}$$



6

TIME AND WORK

6.1. Introduction

If a person can complete a work in ‘n’ days then he can do $\frac{1}{n}$ part of the work in one day.

The amount of work done by a person in 1 day is called his efficiency.

Example: A can do a work in 10 days. Then the efficiency of A is given by $A = \frac{1}{10}$.

Note: Number of days required to do a work = work to be done/work per day.

Example 1: If A can do a work in 10 days, B can do it in 20 days and C in 30 days in how many days will the three together do it?

Solution: The efficiencies are $A = \frac{1}{10}$, $B = \frac{1}{20}$ and $C = \frac{1}{30}$

So work done per day by the three $= \frac{1}{10} + \frac{1}{20} + \frac{1}{30} = \frac{11}{60} \Rightarrow$ No of days $= \frac{60}{11} = 5.45$ days.

Example 2: If A and B can do a work in 10 days, B and C can do it in 20 days and C and A can do it in 40 days in what time all the three can do it?

$$\text{Solution: } A + B = \frac{1}{10}$$

$$B + C = \frac{1}{20}$$

$$C + A = \frac{1}{40}$$

Adding all the three we get $2(A + B + C) = \frac{7}{40} \Rightarrow A + B + C = \frac{7}{80} \Rightarrow$ No of days $= \frac{80}{7}$ days.

Note: If all the people do not work for all the time then the principle below can be used:

$$mA + nB + oC = 1. \quad (1 \text{ is the total work})$$

Here, m = no of days A worked

n = no of days B worked

o = no of days C worked

A, B, C = efficiencies

Example 3: If A can do a work in 12 days, B can do it in 18 days and C in 24 days. All the three started the work. A left after two days and C left three days before the completion of the work. How many days are required to complete the work?

Solution: Let the total no of days be x .

A worked only for 2 days, B worked for x days and C worked for $x - 3$ days.

$$\text{So, } mA + nB + oC = 1$$

$$\Rightarrow 2\left(\frac{1}{12}\right) + x\left(\frac{1}{18}\right) + (x-3)\left(\frac{1}{24}\right) = 1$$

$$\Rightarrow 12 + 4x + 3(x-3) = 72$$

$$\Rightarrow x = \frac{69}{7} \text{ days.}$$

Note: The ratio of dividing wages = ratio of efficiencies = ratio of parts of work done

Example 4: A can do a work in 10 days and B can do it in 30 days and C in 60 days. If the total wages for the work is Rs. 1800 what is the share of A?

Solution: Ratio of wages = $\frac{1}{10} : \frac{1}{30} : \frac{1}{60} = 6 : 2 : 1$ (Multiplying each term by LCM 60)

So total 9 equal parts in Rs. 1800 \Rightarrow each part = Rs. 200 \Rightarrow share of A = 6 parts = Rs. 1200.

Note: When pipes are used filling the tank they are treated similar to the men working but some outlet pipes emptying the tank are present whose work will be considered negative.

Example 5: A pipe can fill a tank in 5 hrs but because of a leak at the bottom it takes 1 hr extra. In what time can the leak alone empty the tank?

Solution: Let the filling pipe be A.

$$A = \frac{1}{5}$$

$$\text{But with the leak L, } A - L = \frac{1}{6} \quad (\text{A-L because leak is outlet})$$

$$\text{So, } \frac{1}{L} = \frac{1}{5} - \frac{1}{6} = \frac{1}{30} \Rightarrow \text{Leak can empty the tank in 30 hrs.}$$



7

CLOCKS

7.1. Introduction

In a clock the most important hands are the minutes hand and the hours hand. Whatever may be the shape of the dial they move in a circular track.

The total angle of 360 degrees in a watch is divided into 12 sectors, one for each hour.

$$\text{So one hour sector} = \frac{360}{12} = 30 \text{ degrees.}$$

For every one hour (60 min),

- The minutes hand moves through 360 deg.
- The hours hand moves through 30 deg.

So for every minute,

- The minutes hand moves through 6 deg
- The hours hand moves through 0.5 deg.

They move in same direction. So their relative displacement for every minute is 5.5 deg.

This 5.5 deg movement constitutes the movements of both the hands.

So for every minute both the hands give a displacement of 5.5 deg.

Note:

1. Between every two hours i.e., between 1 and 2, 2 and 3 and so on the hands of the clock coincide with each other for one time except between 11, 12 and 12, 1.
In a day they coincide for 22 times.
2. Between every two hours they are perpendicular to each other two times except between 2, 3 and 3, 4 and 8, 9 and 9, 10.
In a day they will be perpendicular for 44 times.
3. Between every two hours they will be opposite to each other one time except between 5, 6 and 6, 7.
In a day they will be opposite for 22 times.

Examples: At what time between 5 and 6 will the hands of the clock coincide?

Solution: At 5 the angle between the hands is 150 deg.

To coincide, they collectively have to travel this distance. Every minute they travel 5.5 deg.

$$\text{So no. of minutes required to coincide} = \frac{150}{5.5} = \frac{300}{11} = 27\frac{3}{11} \text{ min.}$$

Examples: At what time between 6 and 7 will the hands be perpendicular?

Solution: At 6 the angle between the hands is 180 deg.

To form 90 deg they have to cover 90 deg (out of 180 if 90 is covered 90 will remain)

$$\text{So no. of minutes required} = \frac{90}{5.5} = \frac{180}{11} = \frac{164}{11} \text{ min.}$$

But they will be perpendicular for two times. The second one will happen after the minutes hand crosses the hours hand and then for 90 deg.

So it has to travel $180 + 90 = 270$ deg.

$$\text{So time} = \frac{270}{5.5} = \frac{540}{11} = 49\frac{1}{11} \text{ min.}$$

Examples: What is the angle between the hands of the clock at 3.45?

Solution: At 3, the angle between the hands = A = 90 deg.

In 45 min the hands will move angle of B = 45×5.5 deg (since 5.5 deg for 1 min)

B = 247.5 deg.

Required angle = A ~ B = 157.5 deg.

Examples: What is the angle between the hands at 4.40?

Solution: At 4 the angle between the hands, A = 120 deg.

In 40 min, B = $40 \times 5.5 = 220$ deg.

The required angle = A ~ B = 100 deg.

Examples: A clock loses 5 min for every hour and another gains 5 min for every hour. If they are set correct at 10 am on Monday then when will they be 12 hrs apart?

Solution: For every hour watch A loses 5 min and watch B gains 5 min.

So for every hour they will differ by 10 min.

$$\text{For 12 hrs (720 min) difference between them the time required} = \frac{720}{10} = 72 \text{ hrs}$$

So they will be 12 hrs apart after 3 days i.e., at 10 am on Thursday.



8

CALENDARS

8.1. Calendars

Here you mainly deal in finding the day of the week on a particular given date.

The process of finding this depends on the number of odd days.

Odd days are quite different from the odd numbers.

- **Odd Days:** The days more than the complete number of weeks in a given period are called odd days.
- **Ordinary Year:** An year that has 365 days is called Ordinary Year.
- **Leap Year:** The year which is exactly divisible by 4 (except century) is called a leap year.

Example: 1968, 1972, 1984, 1988 and so on are the examples of Leap Years.

1986, 1990, 1994, 1998, and so on are the examples of non leap years.

Note: The Centuries divisible by 400 are leap years.

Important Points:

- An ordinary year has 365 days = 52 weeks and 1 odd day.
- A leap year has 366 days = 52 weeks and 2 odd days.
- Century = 76 Ordinary years + 24 Leap years.
- Century contain 5 odd days.
- 200 years contain 3 odd days.
- 300 years contain 1 odd day.
- 400 years contain 0 odd days.
- Last day of a century cannot be Tuesday, Thursday or Saturday.
- First day of a century must be Monday, Tuesday, Thursday or Saturday.

Explanation:

$$100 \text{ years} = 76 \text{ ordinary years} + 24 \text{ leap years}$$

$$= 76 \text{ odd days} + 24 \times 2 \text{ odd days}$$

$$= 124 \text{ odd days} = 17 \text{ weeks} + 5 \text{ days}$$

- ∴ 100 years contain 5 odd days.
No. of odd days in first century = 5
∴ Last day of first century is Friday.
No. of odd days in two centuries = 3
∴ Wednesday is the last day.
No. of odd days in three centuries = 1
∴ Monday is the last day.
No. of odd days in four centuries = 0
∴ Sunday is the last day.

Since the order is continually kept in successive cycles, the last day of a century cannot be Tuesday, Thursday or Saturday.

So, the last day of a century should be Sunday, Monday, Wednesday or Friday.

Therefore, the first day of a century must be Monday, Tuesday, Thursday or Saturday.

8.2. Working Rules

Working rule to find the day of the week on a particular date when reference day is given:

Step 1: Find the net number of odd days for the period between the reference date and the given date (exclude the reference day but count the given date for counting the number of net odd days).

Step 2: The day of the week on the particular date is equal to the number of net odd days ahead of the reference day (if the reference day was before this date) but behind the reference day (if this date was behind the reference day).

Working rule to find the day of the week on a particular date when no reference day is given

Step 1: Count the net number of odd days on the given date

Step 2: Write:

For 0 odd days – Sunday

For 1 odd day – Monday

For 2 odd days – Tuesday

⋮ ⋮ ⋮

For 6 odd days - Saturday

Examples: If 11th January 1997 was a Sunday then what day of the week was on 10th January 2000?

Solution: Total number of days between 11th January 1997 and 10th January 2000

$$= (365 - 11) \text{ in } 1997 + 365 \text{ in } 1998 + 365 \text{ in } 1999 + 10 \text{ days in } 2000$$

$$= (50 \text{ weeks} + 4 \text{ odd days}) + (52 \text{ weeks} + 1 \text{ odd day}) + (52 \text{ weeks} + 1 \text{ odd day}) + (1 \text{ week} + 3 \text{ odd days})$$

$$\text{Total number of odd days} = 4 + 1 + 1 + 3 = 9 \text{ days} = 1 \text{ week} + 2 \text{ days}$$

Hence, 10th January, 2000 would be 2 days ahead of Sunday i.e. it was on Tuesday.

Examples: What day of the week was on 10th June 2008?

Solution: 10th June 2008 = 2007 years + First 5 months up to May 2008 + 10 days of June

2000 years have 0 odd days.

Remaining 7 years has 1 leap year and 6 ordinary years $\Rightarrow 2 + 6 = 8$ odd days

So, 2007 years have 8 odd days.

No. of odd days from 1st January 2008 to 31st May 2008 = $3+1+3+2+3 = 12$

10 days of June has 3 odd days.

Total number of odd days = $8+12+3 = 23$

23 odd days = 3 weeks + 2 odd days.

Hence, 10th June, 2008 was Tuesday.



9

BLOOD RELATIONS

9.1. Introduction

The standard definitions of relations are given below

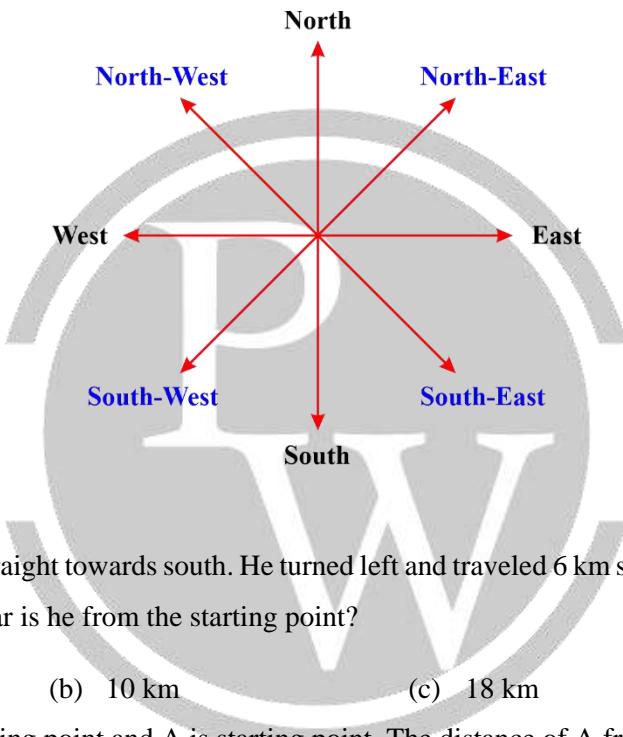
A/An ↓	is related to a PERSON as ↓
Grandfather	The father of his/her mother or father
Grandmother	The mother of his/her mother or father
Grandson	The son of his/her daughter/son
Granddaughter	The daughter of his/her daughters/son
Uncle	The brother of his/her mother or father
Aunt	The sister of his/her mother or father
Nephew	The son of his/her brother or sister
Cousin	The son or daughter of his/her aunt or uncle
Niece	The daughter of his/her brother or sister
Spouse	as her husband or his wife
Father-in-law	the father of his/her spouse
Mother-in-law	the mother of his/her spouse
Sister-in-law	the sister of his/her spouse
Brother-in-law	the brother of his/her spouse
Son-in-law	the spouse of his/her daughter
Daughter-in-law	the spouse of his/her son



10

DIRECTIONS

10.1. Introduction

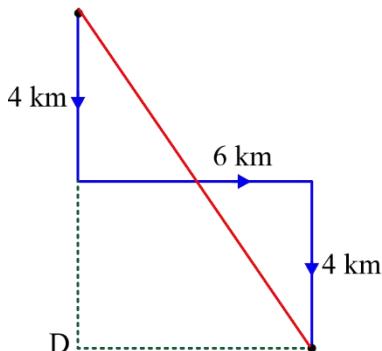


Examples:

Example 1: Ravi traveled 4 km straight towards south. He turned left and traveled 6 km straight, then turned right and traveled 4 km straight. How far is he from the starting point?

- (a) 8 km (b) 10 km (c) 18 km (d) 12 km

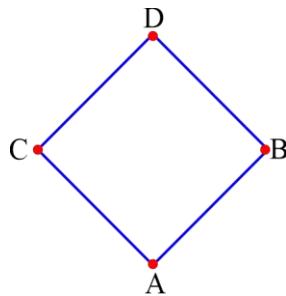
Solution. 10 km. B is the finishing point and A is starting point. The distance of A from B is



Example 2. A is to the South-East of C, B is to the East of C and North-East of A. If D is to the North of A and North-West of B. In which direction of C is D located?

- (a) North-West (b) South-West (c) North-East (d) South-East

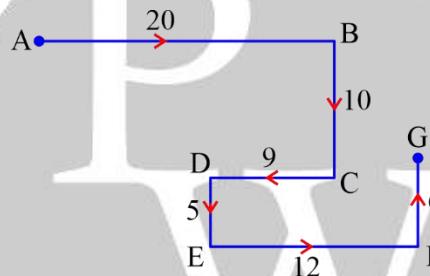
Solution. North-East D is located to the North-East of C.



Example 3. A rat runs 20' towards East and turns to right, runs 10' and turns to right, runs 9' and again turns to left, runs 5' and then runs to left, runs 12' and finally turns to left and runs 6'. Now, which direction is the rat facing?

- (a) East
- (b) West
- (c) North
- (d) South

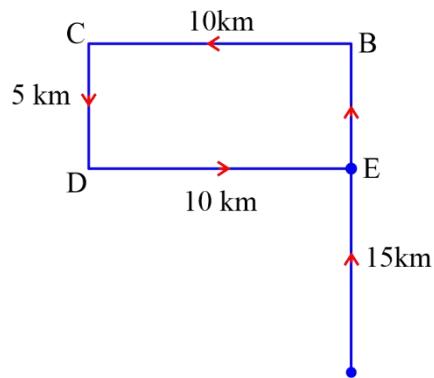
Solution. North. The movements of the rat from A to G are as shown in the fig. It is clear, rat is walking in one direction FG, i.e., North.



Example 4. From his house, Lokesh went 15 km to the North. Then he turned west and covered 10 km, then he turned South and covered 5 km. Finally, turning to East, he covered 10 km. In which direction is he from his house?

- (a) East
- (b) South
- (c) North
- (d) West

Solution. North. Starting point is A and ending point is E. E is to the north of his house at A.



11

DATA INTERPRETATION

11.1. Data Interpretation

- It deals with careful reading, understanding, organizing and interpreting the data provided so as to derive meaningful conclusions.
- Mostly used tools for interpretation of a data are
 - Ratio
 - Percentage
 - Rate
 - Average

11.2. Types of Data Interpretation:

The numerical data pertaining to any event can be presented by any one or more of the following methods.

1. Tables
2. Line Graphs
3. Bar Graphs or Bar Charts
4. Pie Charts or Circle Graphs

11.2.1. Tables

It is the systematic presentation of data in tabular form to understand the given information and to make clear the problem in a certain field of study. It has six elements namely:

- **Title:** It is the heading of the table.
- **Stule:** It is the section of the table containing row headings
- **Column Captions:** It is the heading of each column
- **Body:** It consists the numerical figures
- **Footnotes:** It is for further explanation of the table
- **Source:** It is the authority of the data

Example: Study the following table and answer the questions given below it.

Annual Income of Five Schools

Figures in '00 rupees

Sources of Income	School A	School B	School C	School D	School E
Tuition Fee	120	60	210	90	120
Term Fee	24	12	45	24	30
Donations	54	21	60	51	60
Funds	60	54	120	42	55
Miscellaneous	12	3	15	3	15
Total	270	150	450	210	280

Example: The income by way of donation to school D is what per-cent of its miscellaneous?

Solution: Required percentage = $\frac{5100}{300} = 27\%$

11.2.2. Line Graph

A line graph indicates the variation of a quantity w.r.t two parameters calibrated on X and Y-axis respectively.

Note:

1. Any part of the line graph parallel to X-axis represents no change in the value of Y parameter w.r.t the value of X parameter.
2. The steepest or maximum part of the line graph indicates maximum percentage change of the value during the two consecutive period in which the related part lies.
3. If the steepest part is a rise slope, then it is the highest percentage growth.
4. If the steepest part is a decline slope, it will represent a maximum percentage fall of the value calibrated in the other axis.

11.2.3. Bar Graph

Bar graphs are diagrammatic representation of a discrete data.

Types of Bar Graphs:

- **Simple Bar Graphs:** A simple bar graph relates to only one variable. The values of the variables may relate to different years or different terms.
- **Sub-divided Bar Graph:** It is used to represent various parts of sub-classes of total magnitude of the given variable.
- **Multiple Bar Graphs:** In this type, two or more bars are constructed adjoining each other, to represent either different components of a total or to show multiple variables.

11.2.4. Pie Chart

In this method of representation, the total quantity is distributed over a total angle of 360° which is one complete circle or pie.



12

DATA SUFFICIENCY

12.1. Introduction

Data sufficiency questions are designed to measure your ability to analyze a quantities problem, recognize which given information is relevant, and determine at what point there is sufficient information to solve a problem. In these questions, you are to classify each problem according to the five or four fixed answer choice, rather than find a solution to the problem.

Each Data sufficiency question consists of a question, often accompanied by some initial information, and two statements, labeled (1) and (2), which contain additional information. You must decide whether the information in each statement is sufficient to answer the question or- if neither statement provides enough information –whether the information in the two statements together is sufficient. It is also possible that the statements in combination do not give enough information answer the question.

Begin by reading the initial information and the question carefully. Next, consider the first statement. Does the information provided by the first statement is sufficient to answer the question? Go on the statement. Try to ignore the information given in the first statement when you consider the second statement. Now you should be able to say, for each statement, whether it is sufficient to determine the answer.

Next, consider the two statements in tandem. Do they, together, enable you to answer the question?

Give our answers as per the following statements

- A Statement (1) alone is sufficient but
Statement (2) alone is not sufficient
- B Statement (2) alone is sufficient but
Statement (1) alone is not sufficient
- C Both statements together are sufficient
but neither statement alone is sufficient
- D Each statement alone is sufficient
- E Both statement together are still not sufficient.



OUR YOUTUBE CHANNELS



**GATE
WALLAH**
EE, EC & CS



GATE Wallah



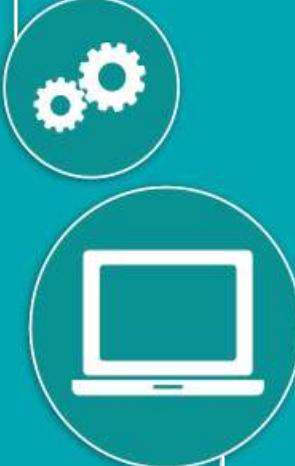
**GATE
WALLAH**
ME, CE & XE



GATE Wallah (English)

ACCESS QUALITY CONTENT

For Free



Thank you

