GATE PEDIA

COMPUTER SCIENCE & INFORMATION TECHNOLOGY

Published By:

Physics Wallah

ISBN: 978-93-94342-51-4

Mobile App: Physics Wallah (Available on Play Store)

Website: www.pw.live

Email: support@pw.live

Rights

Disclaimer

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

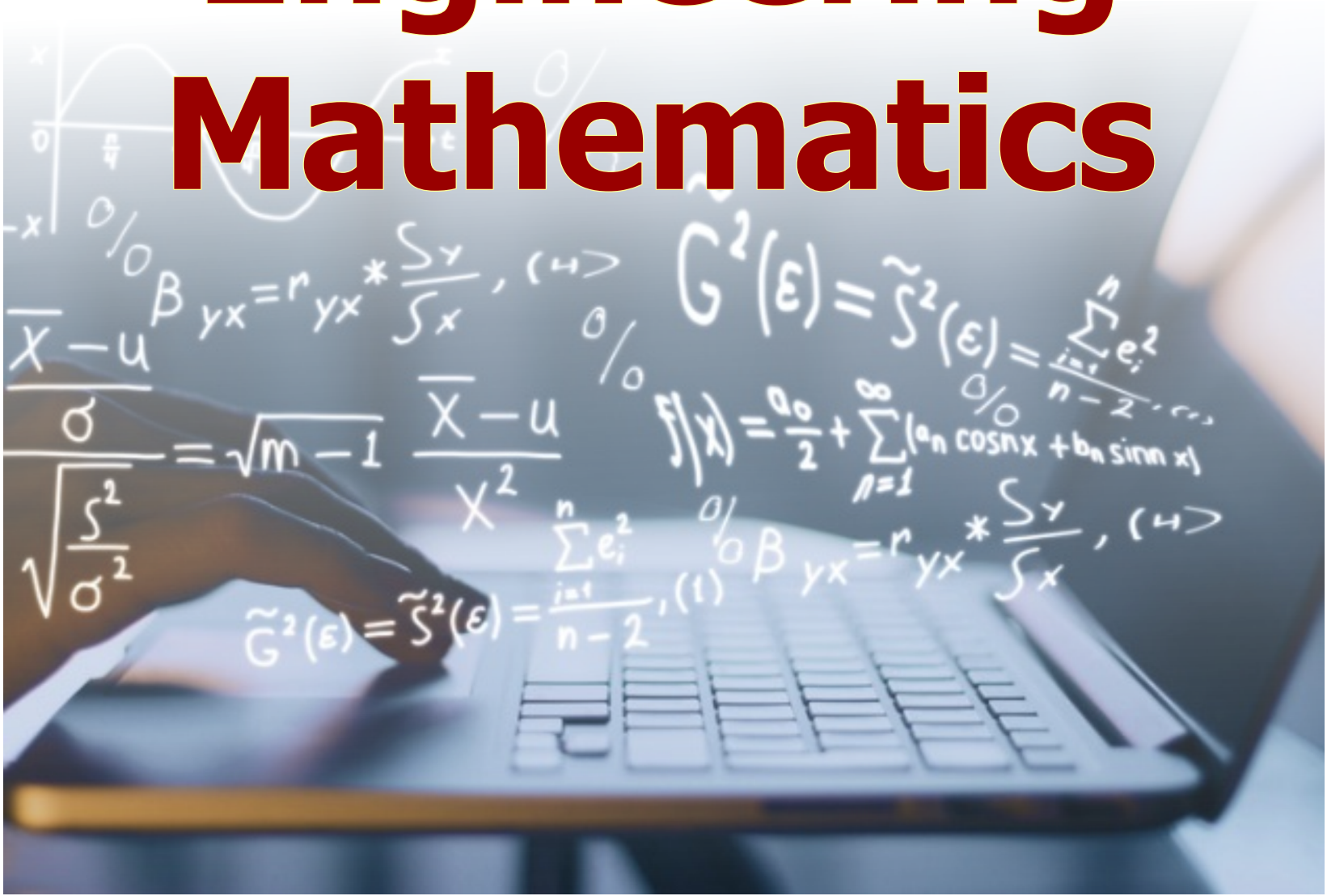
While the author and publisher have used their best efforts in preparing these books. The content has been checked for accuracy. As the book is intended for educational purposes, the author shall not be responsible for any errors contained in the book.

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

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

INDEX

Engineering Mathematics



Engineering Mathematics

INDEX

GATE Exam 2025?

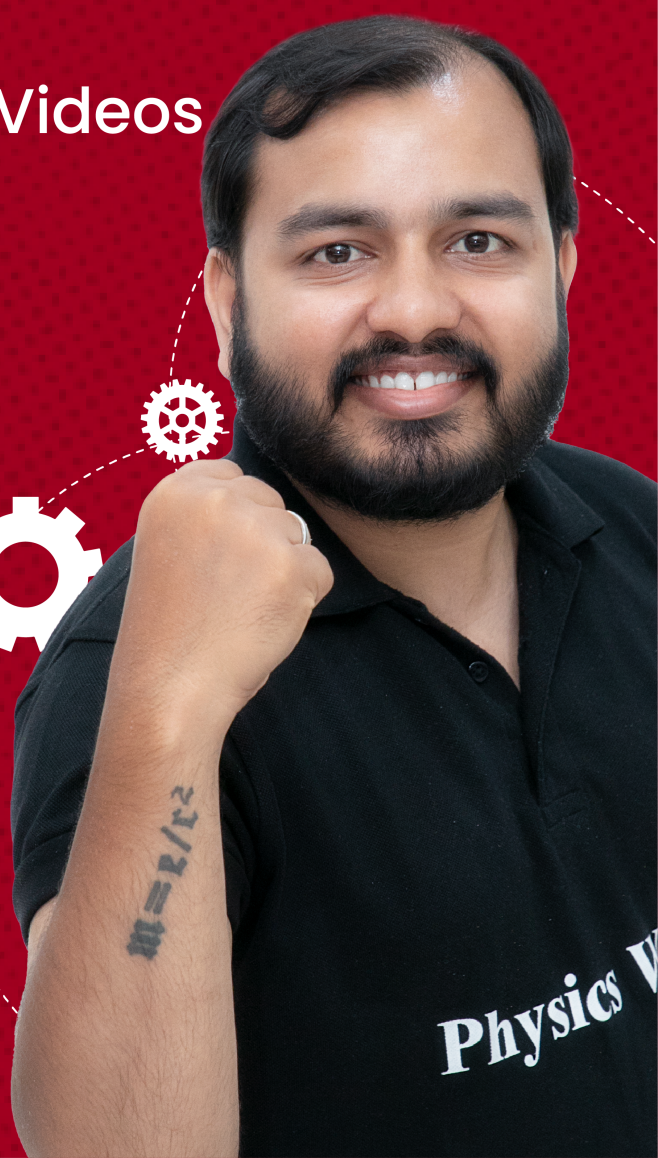
SHRESHTH BATCH

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!



1 BASIC CALCULUS

1.1. Introduction

1.1.1 Limits, Continuity and Differentiability

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

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

It is mathematically defined as

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

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

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

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

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

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

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

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).

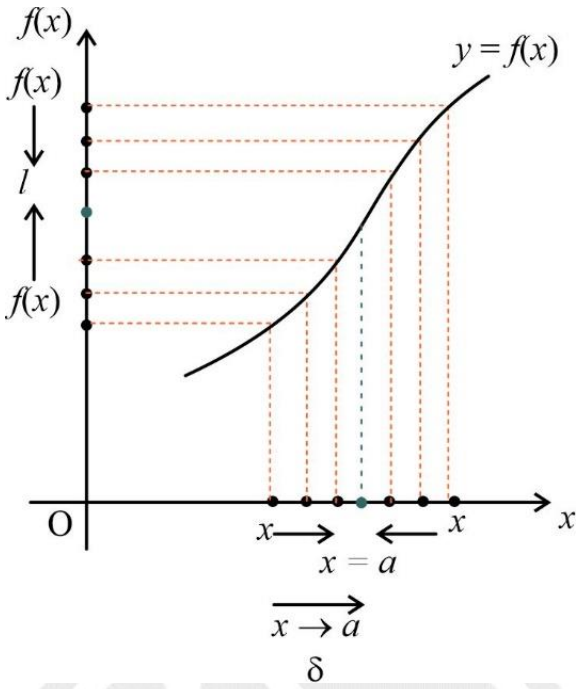


Fig. 1.1

(b) Concept of differentiability

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

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

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

(c) Some Standard Derivatives

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

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

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

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

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

(vi) $\frac{d}{dx}(\sec x) = \sec x\,.\tan x$

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

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

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

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

(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) $\frac{d}{dx}(cosec^{-1}x) = \frac{-1}{|x|\sqrt{x^2-1}}$; $|x| > 1$

(xiv) $\frac{d}{dx}(log_a x) = \frac{1}{x\, log_e a}$

(xv) $\frac{d}{dx}(log_e x) = \frac{1}{x}$

(xvi) $\frac{d}{dx}(a^x) = a^x.log_e a$

(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) $\frac{d}{dx}(sinh\, x) = cos\, h\, x$

(d) Product rule of differentiation

(i) $\frac{d}{dx}\big(f(x).g(x)\big) = f(x).g'(x) + f'(x).g(x)$

(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 from :

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

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

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

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

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

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

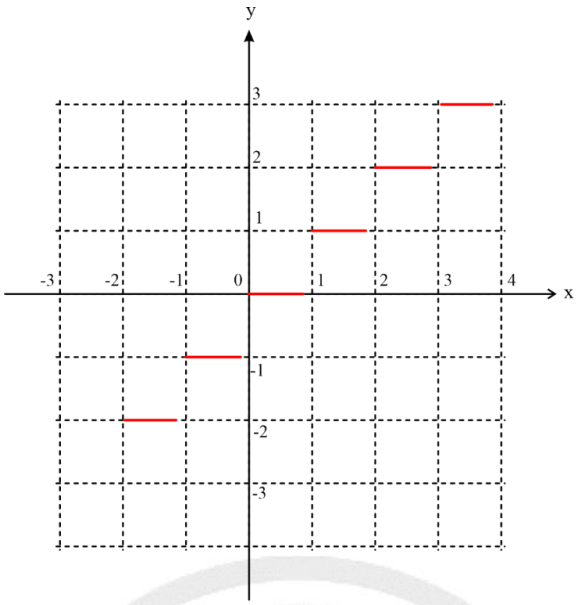


Fig.1.2. Greatest Integer

(i) Properties of Limits

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

(ii) $\lim_{x\to a}\big(f(x).g(x)\big) = \lim_{x\to a} f(x).\lim_{x\to a} g(x)$

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

(iv) If $\lim_{x\to a} f(x)$ exists and $\lim_{x\to a} g(x) = \nexists$, then $\lim_{x\to a} f(x).g(x)$ **may exist**

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

But $\lim_{x\to 0} \sin x.\frac{1}{x} = 1$

(v) Indeterminate form III (0^0, 1^∞, ∞^0)

If $y = \lim_{x\to a}\left[f(x)\right]^{\phi(x)}$

Then, $\log y = \lim_{x\to a}\phi(x)\log\left[f(x)\right]$

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

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

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

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

(j) Some Standard Limits

(i) $\lim_{x\to 0}\frac{\sin x}{x}=1$

(ii) $\lim_{x\to 0}\frac{\tan x}{x}=1$

(iii) $\lim_{x\to 0}\frac{1-\cos ax}{x^2}=\frac{a^2}{2}$

(iv) $\lim_{x\to \infty}\frac{\sin x}{x}=0$

(v) $\lim_{x\to \infty}\frac{\cos x}{x}=0$

(vi) $\lim_{x\to 0}(1+ax)^{b/x}=e^{ab}$

(vii) $\lim_{x\to \infty}\left(1+\frac{a}{x}\right)^{bx}=e^{ab}$

(viii) $\lim_{x\to 0}\left(\frac{a^x+b^x}{2}\right)^{1/x}=\sqrt{ab}$

(ix) $\lim_{x\to 0}\left(\frac{1^x+2^x+3^x+....+n^x}{n}\right)^{1/x}=\sqrt[n]{n!}$

(x) $\lim_{x\to 0}\frac{a^x-1}{x}=\log_e a\,;\lim_{x\to 0}\frac{e^x-1}{x}=1$

(xi) $\lim_{x\to 0}x.\sin\left(\frac{1}{x}\right)=0$

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 ∃ at least one point 'c' such that c ∈ (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

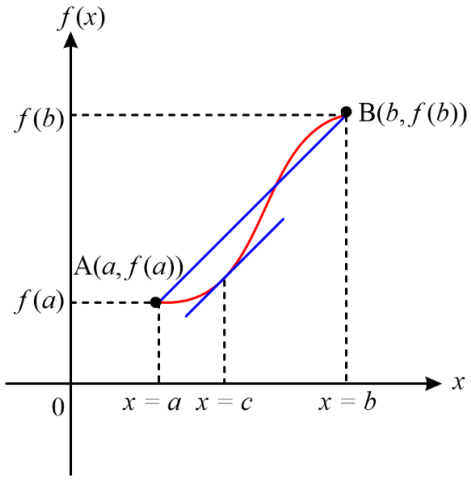


Fig.1.3. LMVT

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 $f'(c) = 0$.

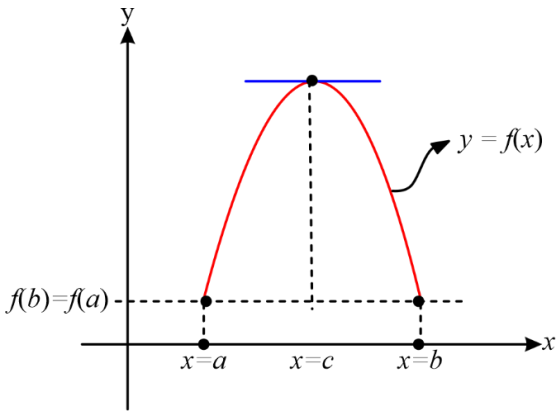


Fig. 1.4. Rolle's mean value

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 $\left(\frac{\pi}{2}, \pi\right)$

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 $a_1, a_2, \ldots$ then $f(x)$ is stationary at $x = a_1, a_2, \ldots\ldots$ Thus $x = a_1, a_2, \ldots \ldots$ are the only points at which $f(x)$ can be maximum or a minimum.

(3) Find $f''(x)$ and substitute in it by terms $x = a_1, a_2, \ldots\ldots$ 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 $f(x) = x^3$

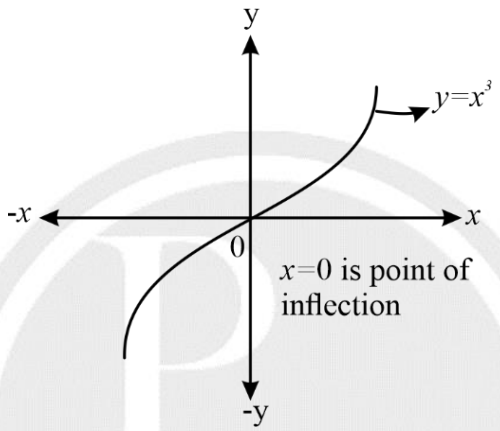


Fig. 1.5. Graph of x^3

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

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

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

- **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 $(f'(x), f''(x), f'''(x), \ldots\ldots \text{exists})$ then the Taylor series expansion of $f(x)$ about the point $x = a$ is given by

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

If $a = 0$, then $f(x) = f(0) + \frac{f'(0)}{1!}x + \frac{f''(0)}{2!}x^2 + \frac{f'''(0)}{3!}x^3 + \ldots\ldots\infty$ (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 $f(x+h) = f(x) + h.\frac{f'(x)}{1!} + h^2.\frac{f''(x)}{2!} + h^3.\frac{f'''(x)}{3!} + \ldots\ldots\infty$

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

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

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

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

$\Rightarrow \quad e^x = 1 + \frac{x}{1!} + \frac{x^2}{2!} + \frac{x^3}{3!} + \ldots..$

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

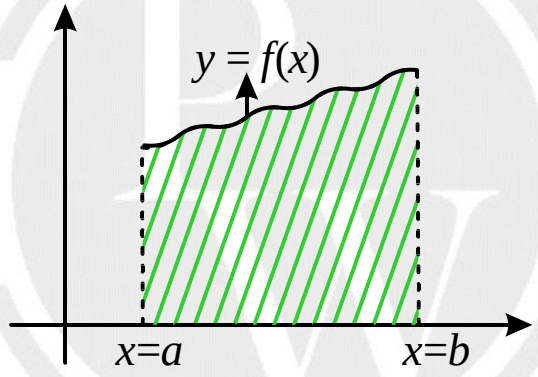


Fig.1. 6. Integration of continuous function

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

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

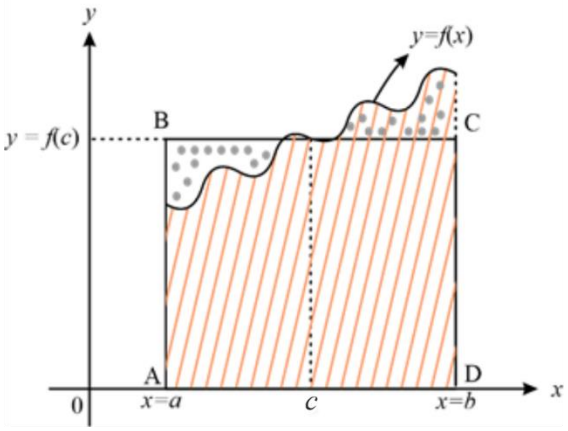


Fig. 1.7. Mean value of integration

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

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

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

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

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. $\int \operatorname{cosec} x dx = \log_e |\operatorname{cosec} x - \cot x| + C$

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

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

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

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

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,

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

$= 0$; if $f(-x) = -f(x)$, ("$f(x)$ is odd function")

4. $\int_0^{2a} f(x)\,dx = 2 \times \int_0^a f(x)\,dx$, if $f(2a - x) = f(x)$

5. $\int_a^b f(x)\,dx = \int_a^b f(a + b - x)\,dx$

6. $\int_a^b \frac{f(x)}{f(x)+f(a+b-x)} = \left(\frac{b-a}{2}\right)$

Example:

(i) $\int_0^{\pi/2} \frac{\sin x}{\sin x+\cos x} = \frac{\pi}{4}$

(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) $\int_2^3 \frac{\sqrt{x}}{\sqrt{x}+\sqrt{5-x}} = \left(\frac{3-2}{2}\right) = \frac{1}{2}$

(iv) $\int_0^{\pi/2} \frac{\sqrt{\tan x}}{\sqrt{\tan x}+\sqrt{\cot x}}\,dx = \frac{\pi}{4}$

7. $\int_0^{\pi/2} \sin^m x\,dx = \int_0^{\pi/2} \cos^m x\,dx = \frac{(m-1)\times(m-3)\times(m-5)}{m\times(m-2)\times(m-4)} \times \ldots \left(\frac{1}{2}\right)$ (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 $s = \int_a^b \sqrt{\left[1+\left(\frac{dy}{dx}\right)^2\right]}\,dx$

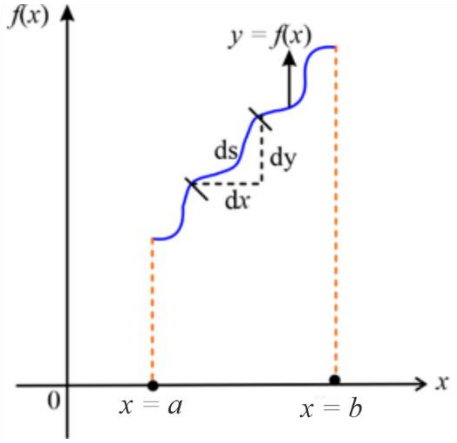


Fig.1.8. Length of the curve

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

(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 $s = \int_\alpha^\beta \sqrt{\left[r^2+\left(\frac{dr}{d\theta}\right)^2\right]}d\theta$

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

Elemental Surface Area

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

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

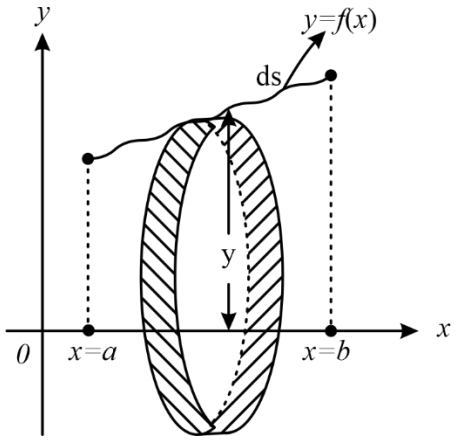


Fig.1.9. Surface area

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

$\Rightarrow \quad V \approx \int_{x=a}^{x=b} \pi y^2 dx$

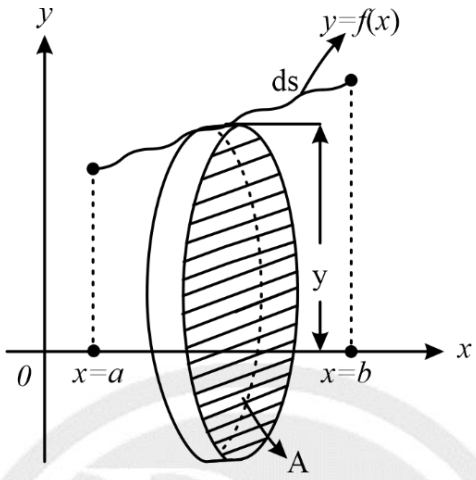


Fig. 1.10. Volume of the solid

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

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

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

1.14.1 Properties of β function

(i) $\beta(m,n) = \frac{\Gamma m.\Gamma n}{\Gamma(m+n)}$

(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) $sin^p\theta.cos^q\,\theta\,dx = \frac{1}{2}\beta\left(\frac{p+1}{2}, \frac{q+1}{2}\right), (p,q > -1)$

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

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

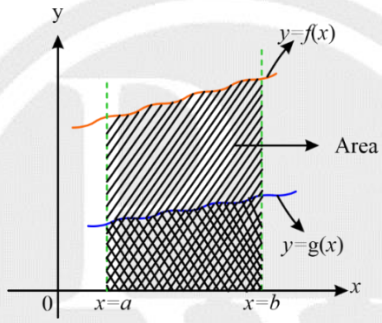


Fig. 1.11. Area under curve

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 $\lim_{\substack{x\to a\\ y\to b}} f(x,y) = f(a,b)$

(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

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

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

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

(c) Homogenous Function

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

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

$\Rightarrow f(kx,ky) = (kx)^3 - 3(kx)^2(ky) + 3(kx).(ky)^2 + (ky)^3$

$= k^3(x^3 - 3x^2y + 3xy^2 + y^3)$

$= k^3.f(x,y) \Rightarrow f(x,y)$ is a homogenous function of degree '3'.

(d) Euler's Theorem

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

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

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

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

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

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

(e) Total derivative:

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

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

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

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

(iv) If x and y are connected by an equation of the form f(x, y) = 0, 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 $r = \left.\frac{\partial^2 f}{\partial x^2}\right|_{(x_0,y_0)}$; $s = \left.\frac{\partial^2 f}{\partial x.\partial y}\right|_{(x_0,y_0)}$; $t = \left.\frac{\partial^2 f}{\partial y^2}\right|_{(x_0,y_0)}$

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

$$I = \iint_R f(x,y)\,dxdy$$

If there is a double integral,

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

Then I = area of the region bounded by the curves, $y = \phi(x)$; $y = \Psi(x)$, $x = a$ and $x = b$ 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

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

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,

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

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

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

1.19 Change of Variables

(i) Cartesian to polar co-ordinates :

$x = r\cos\theta$

$y = r\sin\theta$

$J = r$

$dx\,dy = rdrd\theta$

(ii) Cartesian to cylindrical polar co-ordinate :

$x = r\cos\theta$

$y = r\sin\theta$

$z = z$

$J = r$

$dxdydz = rdr\,d\theta\,dz$

(iii) Cartesian to spherical polar co-ordinates :

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

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

$z = \rho\cos\phi$

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

$dx\,dy\,dz = \rho^2\sin\phi\,d\rho\,d\phi\,d\theta$

❑❑❑

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 × n (read as m by n)

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

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 × n matrix each of whose elements are zero. Then, A + O = A = O + A for every m × 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.

⇒ Matrix –A is additive inverse of A. Because (–A) + A = O = A + (–A). Here O is null matrix of order m × n.

(vi) Cancellation laws holds good in case of addition of matrices, which is X = –A.

⇒ A + X = B + X ⇒ A = B

⇒ X + A = X + B ⇒ A = B

(vii) The equation A + X = 0 has a unique solution in the set of all m × 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.

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

$$A = \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix}_{3\times3} \quad B = \begin{bmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \\ b_{31} & b_{32} \end{bmatrix}_{3\times2} \quad A_{3\times3} \times B_{3\times2}$$

$$\Rightarrow \begin{bmatrix} a_{11}.b_{11} + a_{12}\cdot b_{21} + a_{13}.b_{31} & a_{11}b_{12} + a_{12}b_{22} + a_{13}b_{32} \\ a_{21}.b_{11} + a_{22}.b_{21} + a_{23}.b_{31} & a_{21}b_{12} + a_{22}b_{22} + a_{23}b_{32} \\ a_{31}.b_{11} + a_{32}\cdot b_{21} + a_{33}\cdot b_{31} & a_{31}b_{12} + a_{32}b_{22} + a_{33}b_{32} \end{bmatrix}_{3\times2}$$

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

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

(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

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

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

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 $A = diag\,[d_1, d_2, d_3]$

(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 → Identity Matrix,

If K = 0, then A → 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 $A.A^T = I$

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) $(\overline{A+B}) = \bar{A} + \bar{B}$

(c) $(\overline{KA}) = \bar{K}\bar{A}$ (d) $(\overline{AB}) = \bar{A}\bar{B}$

(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 η 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) $\left|adj\left(adj(A)\right)\right| = (|A|)^{(n-1)^2}$

(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 $(A_{n\times n})\left('n' is\ odd\right)$ 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 × n' determinant is $\underline{n!}$

(xvi) Value of the determinant is invariant under row and column interchange i.e., $\left|A^T\right| = |A|$

(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 the 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| × |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 :

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

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

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
- $A^{-1} = \dfrac{Adj(A)}{|A|}$

2.11 Inverse of a matrix

Inverse of a matrix only exists for square matrices.

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

Properties:

(a) $AA^{-1} = A^{-1}A = I$

(b) $(AB)^{-1} = B^{-1}A^{-1}$

(c) $(ABC)^{-1} = C^{-1}B^{-1}A^{-1}$

(d) $(A^T)^{-1} = (A^{-1})^T$

(e) The inverse of 2 × 2 matrix should be remembered,

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

(i) Interchange the diagonal elements and put negative sign on the rest.

(ii) 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. ρ(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

(i) $\rho(A_{m\times n}) \leq (m, n)$

(ii) $\rho(AB) \leq min\{\rho(A), \rho(B)\}$

(iii) Rank of transpose of matrix is equal to rank of matrix

(iv) Elementary operations do-not affect the rank the matrix

(v) $\rho(A + B) \leq \{\rho(A) + \rho(B)\}$

2.12.2 Row Echelon Form

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

(i) Number of zeroes before the 1st Non-zero element in any row is less than the number of such zeroes in its succeeding row.

(ii) 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

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

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

Ax = B

Coefficient Matrix — Variable Matrix — Constants 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

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

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

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

$$\text{Ax} = 0 \Rightarrow \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix} \begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix} [A/B] = \begin{bmatrix} a_{11} & a_{12} & a_{13} & 0 \\ a_{21} & a_{22} & a_{23} & 0 \\ a_{31} & a_{32} & a_{33} & 0 \end{bmatrix}_{3\times4}$$

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

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

If $x_1 = [a_1 \quad b_1 \quad c_1]; x_2[a_2 \quad b_2 \quad c_2]; x_3 = [a_3 \quad b_3 \quad c_3]$, 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 $k_1x_1 + k_2x_2 + \ldots\ldots + k_nx_n$ is said to be linearly independent if $k_1x_1 + k_2x_2 + \ldots\ldots\ldots + k_nx_n = 0$ when $k_1 = k_2 = k_3 = \ldots\ldots k_n = 0$

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 $\lambda_1, \lambda_2, \lambda_3, \ldots\ldots, \lambda_n$ are 'n' Eigen Values of $A_{n\times n}$, then

(a) Sum of Eigen Values of 'A' = $\lambda_1 + \lambda_2 + \lambda_3 + \ldots + \lambda_n = \sum_{i=1}^{n} \lambda_i = trace(A)$ = Sum of Principal diagonal elements

(b) Product of all the Eigen Values of 'A' = $\lambda_1 \cdot \lambda_2 \cdot \lambda_3 \cdot \ldots\ldots \lambda_n = \prod_{i=1}^{n} \lambda_i = |A|$

(c) Eigen Values of A^m are $\lambda_1^m, \lambda_2^m, \lambda_3^m, \ldots\ldots \lambda_n^m$

(d) Eigen Values of adj(A) are $\frac{|A|}{\lambda_1}, \frac{|A|}{\lambda_2}, \frac{|A|}{\lambda_3}, \ldots\ldots, \frac{|A|}{\lambda_n}$

(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

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

(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 $X_{n\times1}$ 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 $c_0\lambda^n + c_1\lambda^{-1} + ... + c_{n-1}\lambda + c_n = 0$ is the characteristic equation of a square matrix *A* of order *n*, then

$c_0A^n + c_1A^{n-1} + ... + c_{n-1}A + c_nI = 0$ …(i)

Note: When λ is replaced by A in the characteristic equation, the constant term c_n should be replaced by c_nI 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:

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

  $r_1x_1 + r_2x_2 + …. + r_k x_k = 0$

Note: Let $X_1, X_2…….X_n$ be 'n' vector of matrix *A*.

- If rank (*A*) = number of vectors then vector $X_1, X_2…..X_n$ are linearly independent.
- If rank (*A*) ≠ number of vectors then vector $X_1, X_2 ….. X_n$ are linearly dependent.

2.16.3 Vector Space R^n

If n positive integer, then an ordered n-tuple is a sequence of n real numbers ($\alpha_1, \alpha_2, \ldots \alpha_n$). 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 $V_n(F)$ it 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 $X_1, X_2 \ldots X_n$ each of n dimensions is orthogonal then they are surely linearly independent and form the basis for n-dimension space.
- The set of the vector is orthonormal if they are orthogonal and have unit magnitude.

2.17 Similar Matrices

- Two matrix A and B are similar if there exist a non singular matrix P such that $B = P^{-1}AP$
- Similar matrix has same eigen valves
- 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 = {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 with in the sample space 'S' of a RE, then $0 \le P(A) \le 1$

$$\frac{0}{n(S)} \le \boxed{\frac{n(A)}{n(S)}} \le \frac{n(S)}{n(S)}$$

$$\boxed{0 \le P(A) \le 1}$$

(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 $A \cap B = \phi$.

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

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

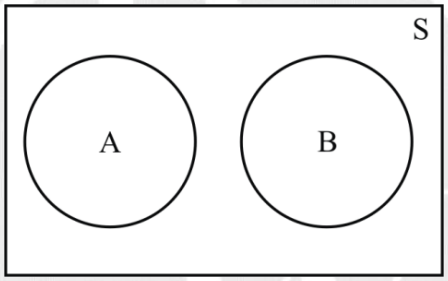


Fig. 5.1. Mutually exclusive event

(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

$A \cup B = S$

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

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

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

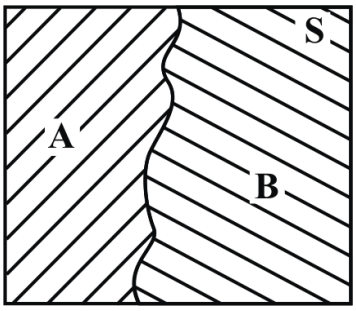


Fig. 5.2. Mutually exhaustive event

(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 $P(A \cap B) = P(A) \cdot P(B)$.

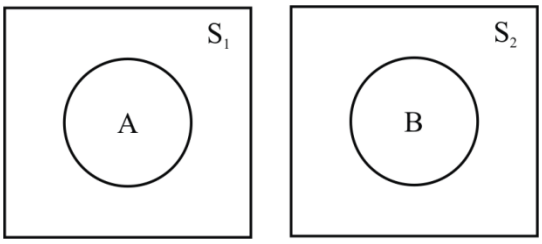


Fig. 5.3. Independent Event

(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

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

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

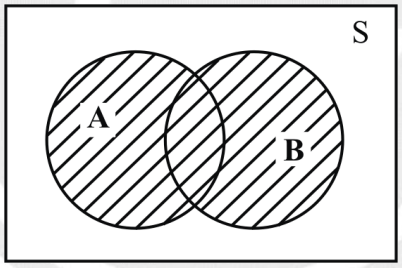


Fig. 5.4. Addition theorem

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

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

  $= P(E_1) + P(E_2) + P(E_3) + \ldots\ldots p(E_n)$

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
  $f(x, y) = f(x) \cdot f(y)$
- $P(a\le x\le b, c\le y\le d)=\int_a^b\int_c^d f(x,y)dydx$
- $f(x)=\int_{-\infty}^{\infty} f(x,y)dy$
- $f(y)=\int_{-\infty}^{\infty} f(x,y)dx$

3.3.5 Multiplication Theorem of Probability:

If A, and B are two events within a sample space 'S', then P(A/B) · P(B) = P(B/A) · 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 $E_1, E_2, E_3, \ldots\ldots E_n$ are 'n' mutually exclusive ($E_i\cap E_j=\phi;\forall i\neq j$) and collectively exhaustive event ($E_1\cup E_2\cup E_3\cup\ldots\ldots\cup E_n = S$) and 'A' is any event with in the sample space 'S', then

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

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

3.3.7 Baye's Theorem

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

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

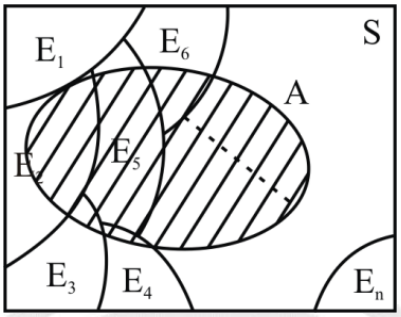


Fig. 5.5. Baye's theorem

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

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

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

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

Result:

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

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

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

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

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

Permutations with Repetition

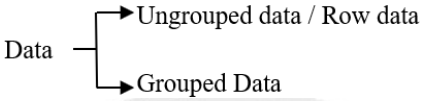
The number of permutations of n objects, where *p* objects are of one kind, q objects are of another kind and the rest, if any, are of a different kind is $\frac{{}^{n}P_{r}}{p!q!}$.

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

(1) The range of the data = R = max{$x_1, x_2, \ldots\ldots, x_n$} – min{$x_1, x_2, x_3, \ldots\ldots, x_n$}

(2) Arithmetic mean **:** Mean of the data is equal to sum of observaions divided by the total number of observations.

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

- 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 $(x_1, x_2, x_3, \ldots\ldots x_n)$ When the data is arranged in either ascending or descending order.

If $x_1, x_2, x_3, x_4, \ldots\ldots x_n$ 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 → Bimodel Data

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

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

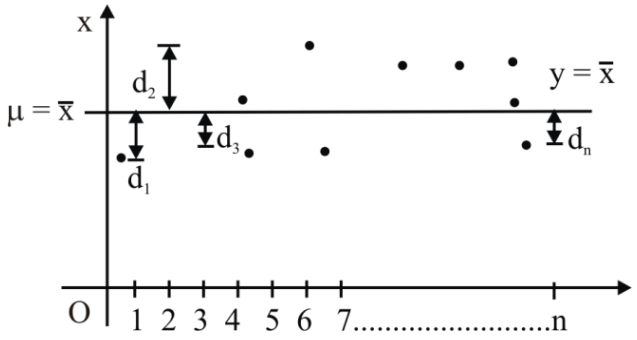


Fig. 5.6. Discrete data

Sum of derivations of all the observations = $\Sigma d_i = (x_1 - \bar{x}) + (x_2 - \bar{x}) + \ldots\ldots 0 + (x_n - \bar{x})$

$$= \Sigma d_i = (x_1 + x_2 + \ldots\ldots + x_n) - n\bar{x}$$

$$\boxed{\Sigma d_i = 0}$$

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

3.4.4 Absolute Mean Deviation

If $x_1, x_2, x_3, \ldots\ldots, x_n$ 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.

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

3.4.5 Standard Deviation

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

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

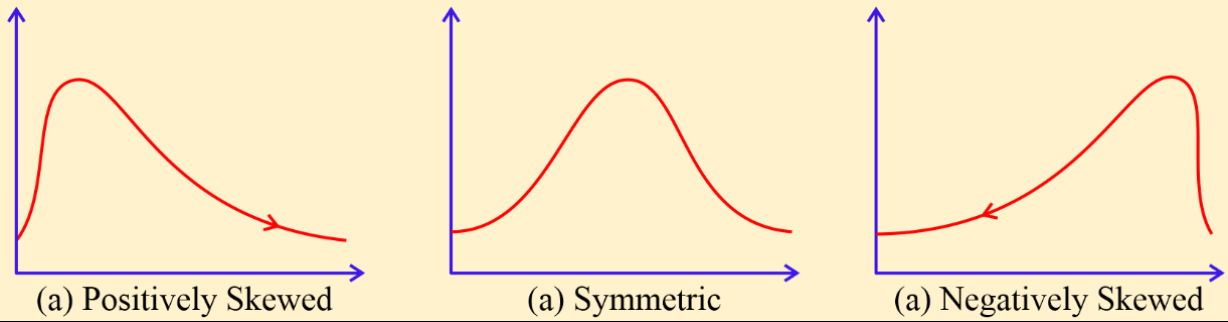
Sample Standard derivation: $\sigma = \sqrt{\frac{1}{(n-1)}\Sigma(x_i - \bar{x})^2}$, n$\rightarrow$ size of sample

Generally (n > 29 $\rightarrow$ population) (n < 29 $\rightarrow$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).



(a) Positively Skewed (a) Symmetric (a) Negatively Skewed

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
  (i) Expectation of 'x' $= E(x) = \sum_{i=1}^{n} x_i P(x = x_i)$
  (ii) 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 $x_1, x_2, ...x_n$ 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

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

2. Expected value of X^2 is,

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

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'

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

Or, 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 ± Y] = E(x) ± E(Y).

(v) If X and Y are random variables E(X. Y) = E(X). E(Y / X).

(vi) If X and Y independent random variables E(X. Y) = E(X). E (Y).

3.5.7 Properties of Variance

(i) Var[C] = 0, C is constant.

(ii) Var (aX) = a^2V(X) where X is random variable and 'a' constant.

Var(–X) = $(-1)^2$ Var(X) = Var(X) Variance is always positive.

(iii) Var(ax + b) = a^2 Var(X) + 0

(iv) If X and Y are independent random variables.

Var(X + Y) = Var(X) + Var(Y)

Var(X – Y) = Var(X) + Var(Y)

(v) Var(ax + by) = a^2 v(x) + b^2 v(y) + 2ab Cov (x, y)

(vi) Cov (x, y) = E(x, y) – E(x) E(y)

(vii) For independent random variables 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 valueof 'x' at which P(x) is maximum is called mode.

3.6.4 Variance

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

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

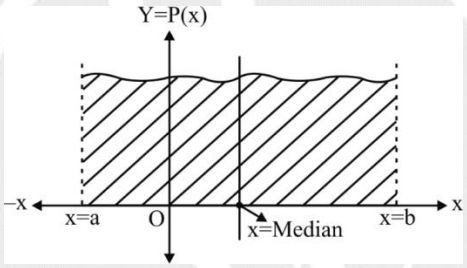


Fig. 5.7. Continuous random variables

3.7 Continous RV distributions

(1) Gaussian/Normal Distributon:

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

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

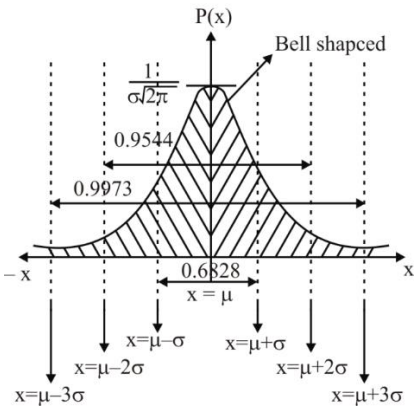


Fig. 5.8. Normal distribution

Mean = Median = Mode = μ

$$P(\mu - \sigma \le x \le \mu + \sigma) = 0.6828$$

$$P(\mu - 2\sigma \le x \le \mu + 2\sigma) = 0.9544$$

$$P(\mu - 3\sigma \le x \le \mu + 3\sigma) = 0.9973$$

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

(2) Standard Normal Distribution:

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

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

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

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

Note:

1. The normal distribution curve is bell shaped curve
2. The points of infelection 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 distrbuted random variable such that $a \le x \le b$ then the Pdf is given by

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

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

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

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

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

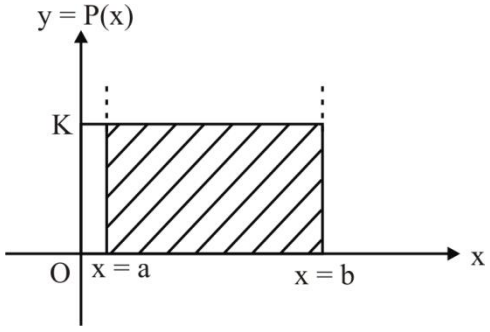


Fig. 5.9. Uniform Distribution

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

(1) Exponential Dirtibution:

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

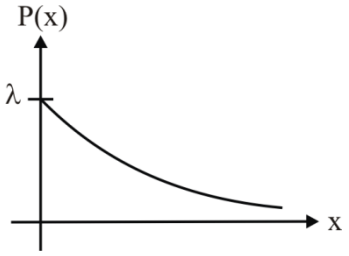


Fig. 5.10. Exponential distribution

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

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

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

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

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 occuring 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 $\rightarrow$ Probability of Success,

q $\rightarrow$ Probability of Failure

p + q = 1

Mean = np,Variance = npq = σ^2, 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

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

Mean = λ.

Mean = Variance = λ

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

❑❑❑

GATE Exam 2024?

PARAKRAM BATCH

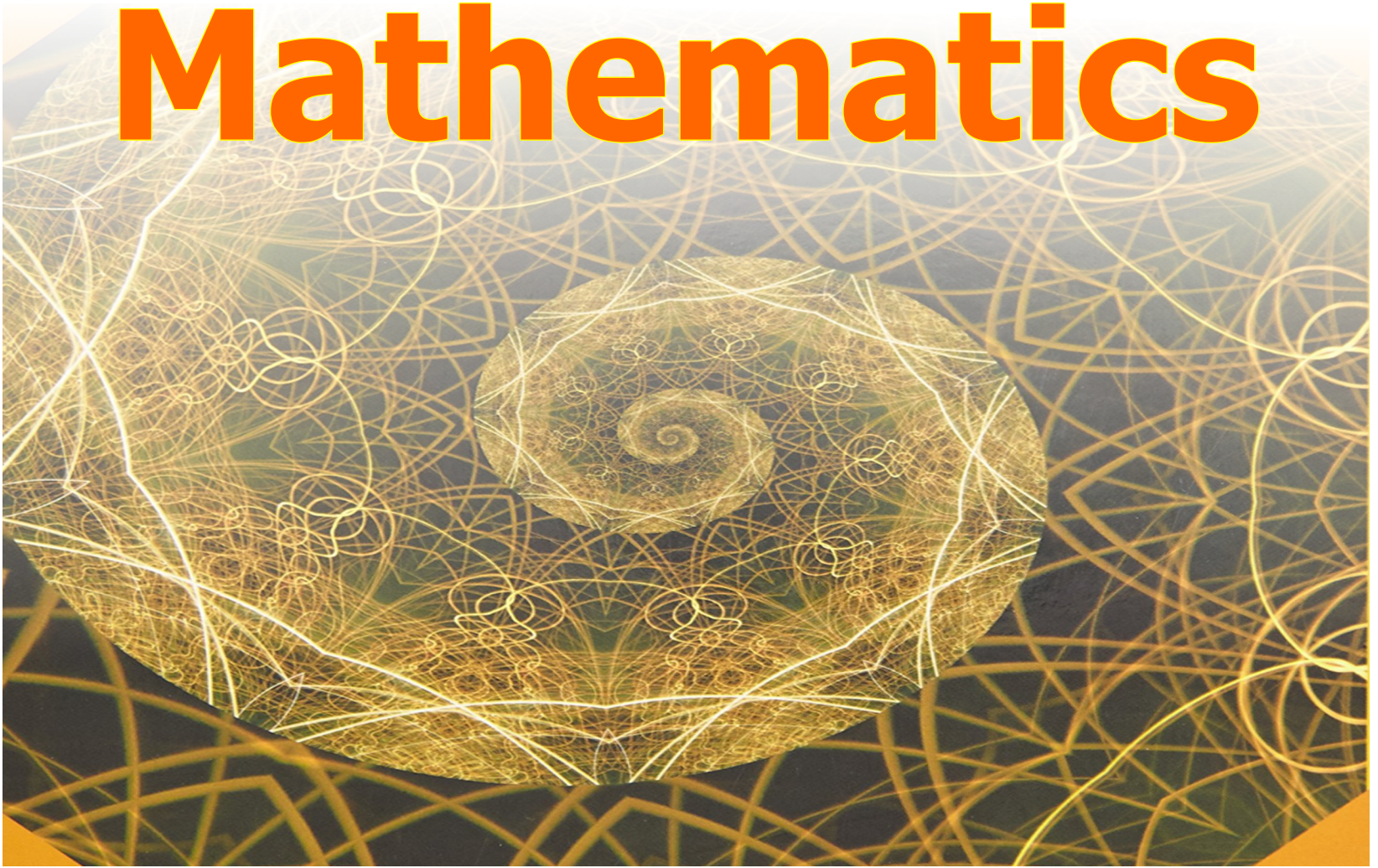
ME | CE | EE | EC | CS & IT

- 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

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 ≤ n – 1.

Note:

Hand Shaking Lemma: Sum of all degrees = 2 × 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:

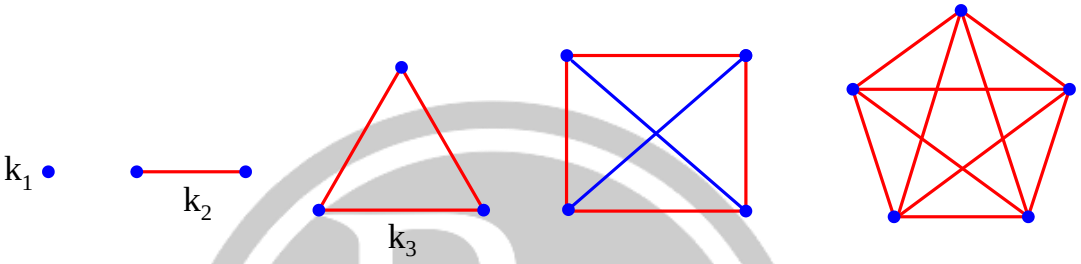
In a simple graph at least two vertices with have same degree ($n \geq 2$)
Example: {5, 4, 3, 2, 1} is not graphical

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) ($n \geq 1$):



Degree of every vertex is n – 1

(or)

There direct edge b/w every pair of vertices

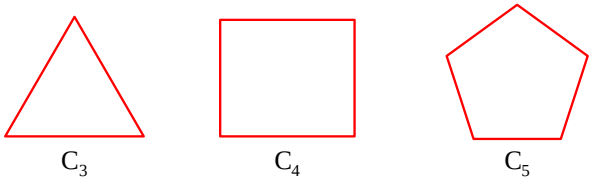
- no. of edges is,

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

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) ($n \geq 3$):



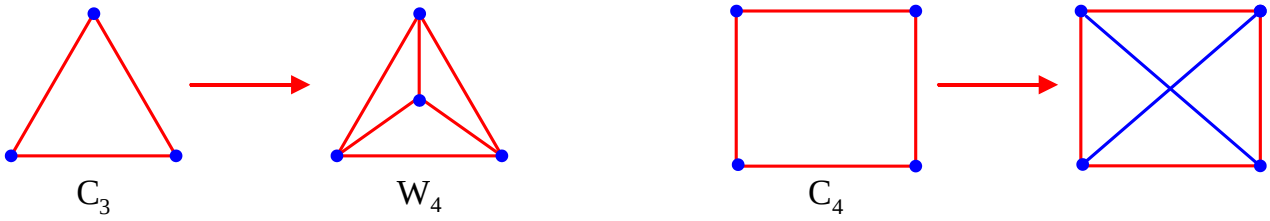
If given degree sequence is all 2's, then it's not guaranteed that it's a cycle graph
G: [2, 2, 2, 2, 2, 2]

Degree of all vertices is 2 in a cycle graph

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

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

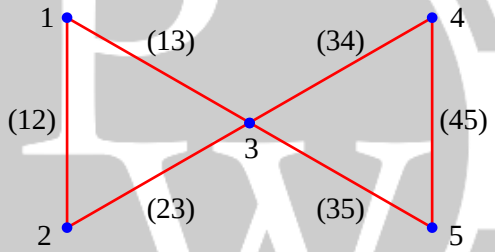
A wheel graph w_n is obtained by adding a vertex (hub) to c_{n-1}(Cycle Graph) such that this vertex is adjacent to all the other vertices.



Number of edges = 2 (n - 1)

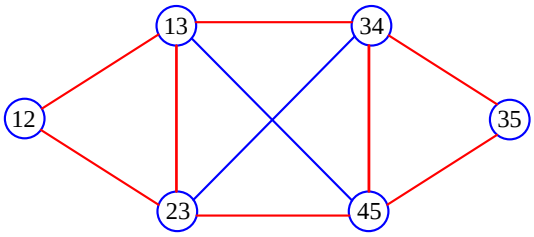
1.1.10 Line Graph (L(G)):

Consider below graph, G



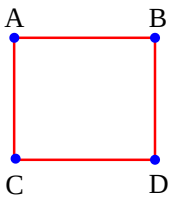
Step to construct L(G)

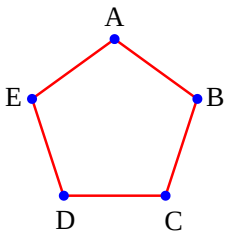
(i) define edges in G
(ii) label these edges as vertices in L(G)
(iii) Connect vertices with common number.

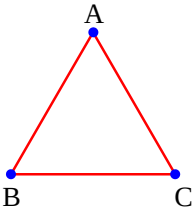


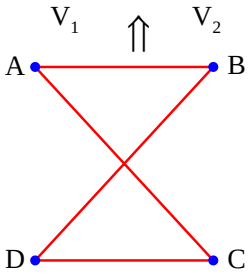
Line graph of every cycle graph is also a cycle.

1.1.11 Bipartite (L(G)):

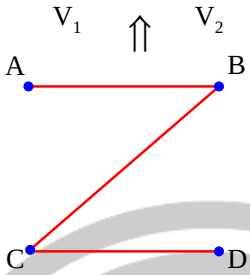








$\therefore C_4$ is a bipartite graph



Now adding E in either of the sets violates the property of bipartite graph

$\therefore C_5$ is not a bipartite graph

adding C is not possible

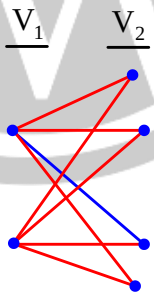
$\therefore C_3$ is not a bipartite graph

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:



$K_2, 4$ $\quad |v_1| = m$

$| v_2 | = n$

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

number of edges = mn

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

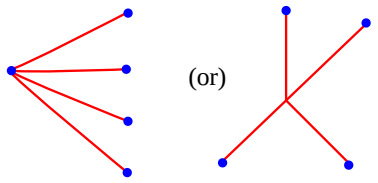
1.1.13 Theorem 5 :

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

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

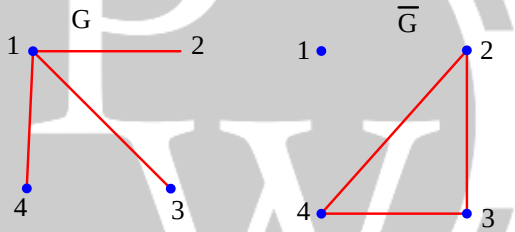
It is complete bipartite graph with one vertex in one set and rest of the vertices in other set.
(or)
Star graph (k_1, k_{n-1}) is complete bipartite graph possible with n vertices and minimum no. of edges.

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.



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

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 $d_1, d_2 \ldots\ldots\ldots d_n$ is degree sequence for G then
$(n-1-d_1), (n1-d_2), \ldots.. (n-1-d_n)$ 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

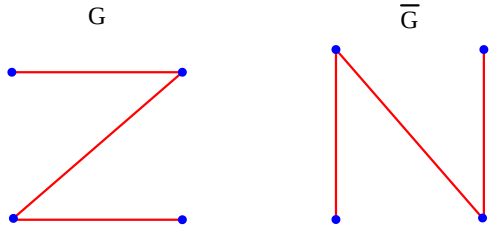
$$k_6 \rightarrow 5, 5, 5, 5, 5, 5$$
$$G \rightarrow 5, 2, 2, 2, 2, 1$$
$$\bar{G} \rightarrow \{0, 3, 3, 3, 3, 4\}$$

Isomorphic Graphs:
Let G1 = (V1, E1) and G2 = (V2, E2) be two undirected graphs. A function f: V1 $\rightarrow$ V2 is called a graph isomorphism if (a) f is one-to-one and onto, and (b) for all a, b $\in$ V1. {a, b} $\in$ E1 if and only if {f(a), f(b)} $\in$ E2. When such a function exists, G1 and G2 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

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

- $e = \frac{n(n-1)}{4}$ 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.

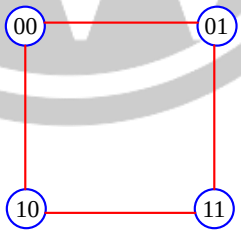
Hypercube (Q_n):

Q_1

2' vertices – 0, 1

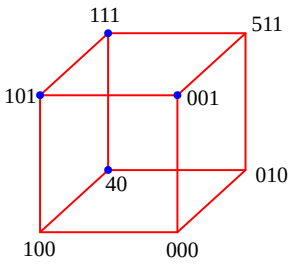
Q_2

4' vertices – 00, 01, 10, 11



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

Q_3: 2^3 vertices



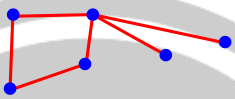
Number of edges = $\frac{n \times 2^n}{2} = n \times 2^{n-1}$

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

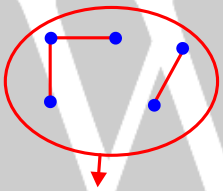
1.2.1 Connected graph:

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.

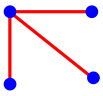


Disconnected graph

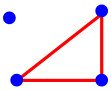
- If G is connected then $\bar{G}$ may be connected or disconnected

G Connected

$\overline{G}$ Connected



G Connected

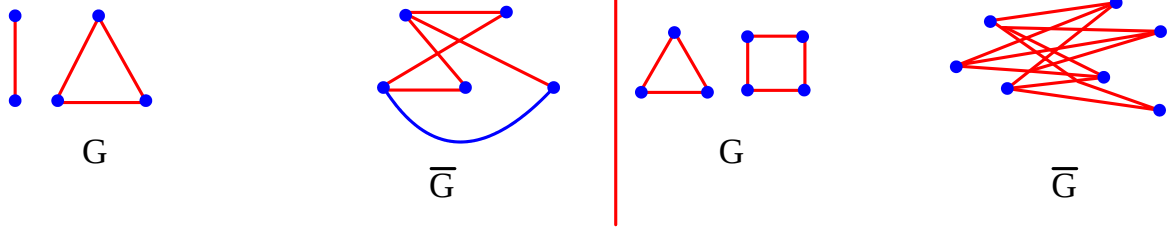


$\overline{G}$ disconnected

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 in vertices is n-1.

$$(n-1)_{Tree} \le e \le \frac{n(n-1)}{2}$$

- 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, these 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

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

Proof:

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

$\therefore n_1 + n_2 + \ldots\ldots + n_2 : k$

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

$\therefore$ min no of edges is

$N_1 - 1 + n_2 - 1 + \ldots\ldots + n_k - 1$

$= (n_1 + n_2 + \ldots.. n_k) - k = n - k$

Note:

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

deg u + deg v ≥ n – 1

nonadjacent vertices u and v of G, then G is connected and diam(G) ≤ 2.

2. If G is a graph of order n with $\delta(G) \ge (n-1)/2$, 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. $k(G) \le \lambda(G) \le \min_{v \in v} \deg(v)$.

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 (≥2) vertices. If deg(v) ≥ (n - 1)/2 for all v ∈ 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 ∈ 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 s 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| = ν 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 ν – e + r = 2.
Let G = (V, E) be a loop-free connected planar graph with |V| = ν, |E| = e > 2, and r regions. Then 3r ≤ 2e and e ≤ 3ν – 6.

1.3.1 Coloring:

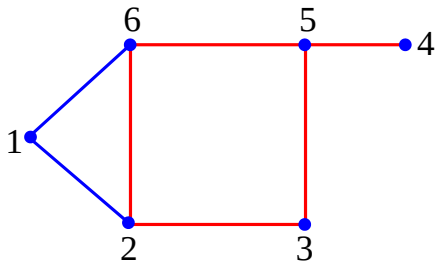
Note: Every MIS will always be MDS. But reverse need not to be true
Domination number ≤ Independence number.
α(G) = ≤ β(G

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

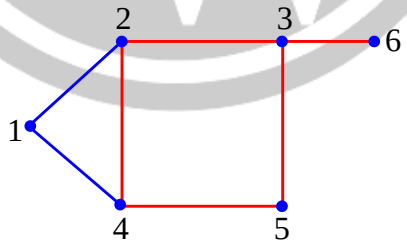
1.5 Perfect Matching:

A matching is said to be prefect 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 indeed degree

Example:



{12, 45, 36} is perfect matching

Note: Every perfect matching is maximal, but reverse need not to be true.
If prefect 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 prefect matching possible for a complete graph with 2n vertices is $\frac{(2n)!}{2^n.n!}$

❑❑❑

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.

Operator	*Precedence*
$\neg$	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.

$\rightarrow$ Every contingency is satisfiable, but reverse need not to be true 1.

$\rightarrow$ Every tautology is satisfiable, but revers 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) $\dfrac{p \quad p \rightarrow q}{\therefore q}$	$[p \wedge (p \rightarrow q)] \rightarrow q$	Rule of Detachment (Modus Ponens)
2) $\dfrac{p \rightarrow q \quad q \rightarrow r}{\therefore p \rightarrow r}$	$[(p \rightarrow q) \wedge (q \rightarrow r)] \rightarrow (p \rightarrow r)$	Law of the syllogism
3) $\dfrac{p \rightarrow q \quad \neg q}{\therefore \neg p}$	$[(p \rightarrow q) \wedge \neg q] \rightarrow \neg p$	Modus Tollens

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

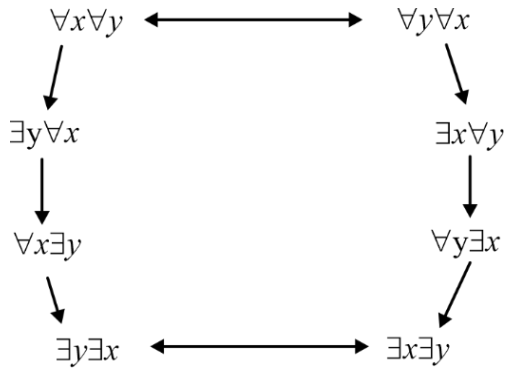
$\forall x \neg p(x)$	For every replacement a from The universe, $p(a)$ is false and its negation $\neg p(a)$ is true	There is at least one replacement a from the universe for which $\neg p(a)$ is false and $p(a)$ is true.

$$\neg [\forall x\, p(x)] \Leftrightarrow \exists x \neg p(x)$$
$$\neg [\exists x\, p(x)] \Leftrightarrow \forall x \neg p(x)$$
$$\neg [\forall x \neg p(x)] \Leftrightarrow \exists x \neg\neg p(x) \Leftrightarrow \exists x\, p(x)$$
$$\neg [\exists x \neg p(x)] \Leftrightarrow \forall x \neg\neg p(x) \Leftrightarrow \forall x\, p(x)$$

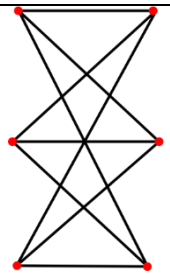
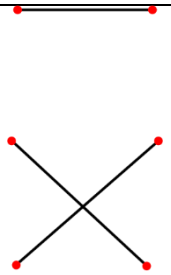
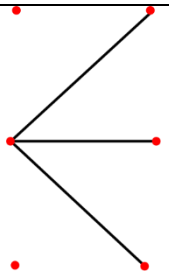
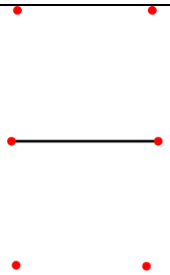
$\exists x\,[P(x) \vee Q(x)] \equiv \exists x\, P(x) \vee \exists x\, Q(x)$
$\forall x\,[P(x) \vee Q(x)] \equiv \forall x\, P(x) \vee \forall x\, Q(x)$
$\exists x\,[P(x) \wedge Q(x)] \rightarrow \exists x\, P(x) \wedge \exists x\, Q(x)$
$\forall x\,[P(x) \wedge \forall\, Q(x)] \rightarrow \forall x\,[P(x)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

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

$\exists x \exists y P(x, y)$ $\exists y \exists x P(x, y)$	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***
$\dfrac{\forall x P(x)}{\therefore P(c)}$	Universal instantiation
$\dfrac{P(c) \text{for an arbitary } c}{\therefore \forall x P(x)}$	Universal generalization
$\dfrac{\exists x P(x)}{\therefore \text{for some element } c}$	Existential instantiation
$\dfrac{P(c) \text{for some elemet } c}{\therefore \exists x P(x)}$	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 ≡ No graphs are connected	$\forall x[G(x) \rightarrow \neg C(x)] \equiv \forall x[G(x) \rightarrow \neg C(x)]$

❑❑❑

3 SET THEORY

3.1 Introduction

Collection of unordered, district and well defined objects.

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

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

  $A = \{z|z \in N \quad x \leq 3\}$ – 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:

  $a \in A$.

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.

$A = B \Leftrightarrow \forall_x (x \in A \leftrightarrow x \in B) \Leftrightarrow A \subseteq B \wedge B \supseteq A$

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 $A \subset B$

  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

  $\phi \subseteq A$

  If A = ϕ, then

  $\phi \subset A$

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, $\bar{A} = A^c = \{x | x \in \cup \wedge x \notin A\}$

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.

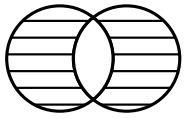
"complement of B in A"

(v) **Symmetric Difference:**

$A \Delta B = \{x | x \in A \vee x \in B$, 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 ϕ.

3.1.6 Loss involving set operations:

- $\left.\begin{aligned} A\cup A&=A\\ A\cap A&=A\end{aligned}\right\}$ Idempotent law
- $\left.\begin{aligned} A\cap\phi&=\phi\\ A\cup U&=U\end{aligned}\right\}$ Domination law
- $\left.\begin{aligned} A\cap U&=A\\ A\cup\phi&=A\end{aligned}\right\}$ Indentity law
- $\left.\begin{aligned} A\cup(B\cup C)&=(A\cup B)\cup C\\ A\cup(B\cap C)&=(A\cap B)\cap C\\ A\oplus(B\oplus C)&=(A\oplus B)\oplus C\end{aligned}\right\}$ Associative law

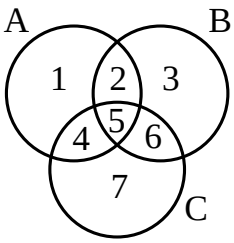
  $\because$ XOR operation is associative
- $\left.\begin{aligned} A\cup(B\cup C)&=(A\cup B)\cup(A\cup c)\\ A\cap(B\cap C)&=(A\cap B)\cup(A\cap C)\end{aligned}\right\}$ Distributive law
- $\left.\begin{aligned} A\cup(A\cap B)&=A\\ A\cap(A\cup B)&=A\end{aligned}\right\}$ absorption law
- $\left.\begin{aligned} \overline{A\cup B}&=\bar{A}\cup\bar{B}\\ \overline{A\cup B}&=\bar{A}\cap\bar{B}\end{aligned}\right\}$ Demorgan's law

Note:

$\left.\begin{aligned} A\oplus B=A\oplus C&\Rightarrow B=C\\ A\oplus C=B\oplus C&\Rightarrow A=C\end{aligned}\right\}$ *The proot can be obtained by relating this to XOR operation in* log*ic.*

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(sd) (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

$A \cup B = B$

Dual is $A \cap B = B$

i.e., $B \subseteq A$

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

Duality principle should be applied only for general can but no for particular cases

3.1.8 Cartesian Product:

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

- Cartesian product is associative

  However

$$\underset{\text{it has ordered pairs t type } ((a,b),c)}{(A \times B) \times C} \neq \underset{(a,(b,c))}{A \times (B \times C)}$$

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

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

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

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

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

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

3.1.9 Multisets:

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

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

Where m_1 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 × B.

If |A| = M and |B| = n, then |A × B| = mn

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

If relation B 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' on 'k' relates y

3.2.1 Inverse of relations:

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

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

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

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}	✓	✓
Irreflective	$\forall a \in A$ ($a \not R\ a$)	2^{n^2-n}	✓	✓
Symmetric	$\forall x, y \in A(xR_y \Rightarrow yR_x)$	$2^n.2^{\frac{n^2-n}{2}}$	✓	✓
Antisymmetric	$\forall x, \forall y\ (xR_y \wedge yR_x \Rightarrow x = y)$	$2^n\ 3^{\frac{n^2-n}{2}}$	×	✓
Asymmetric	$\forall x, \forall y\ (xR_y \Rightarrow y \not R_x)$	$3^{\frac{n^2-n}{2}}$	×	✓
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)∈R and (b, c)∈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

$R_o\Delta_A = R$ and Δ_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 prpperty.

(i) **Reflexive Closure (R*):**

Reflexive clouse of a relation R,

$R^{\#} = RU\Delta_A$

(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
  $R\# = R \cup R^2 . \cup R^3 \cup \ldots\ldots . R^{n-1} \cup R^n$
  '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,
- $Rn \leq R \ \forall n \geq 1$
- also R^n is transitive relation.

3.2.7 Partion:

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

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

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

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

- It R is an equivalence relation, the below two sentences means the same
  (i) aRb
  (ii) $[a]_R = [b]_R$

  aRb & $[a]R \phi [b]$ 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

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

  Where sterling 2nd kind number

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

  The below is a shortcut to find the Bell number

  $B_0 \rightarrow (1)$

  $B_1 \rightarrow (1)$ 2

  $B_2 \rightarrow (2)$ 3 5

  $B_3 \rightarrow (5)$ 7 10 15

  $B_4 \rightarrow (15)$ 20 27 37 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 post [A;R] is called toset it every pair of element 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:

  (i) Create vertex corresponding to every element of set A.
  (ii) 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:

- $\left.\begin{matrix} a \wedge a = a \\ a \vee a = a \end{matrix}\right\}$ Idempotent
- $\left.\begin{matrix} a \vee b = a \vee a \\ a \wedge v = a \wedge a \end{matrix}\right\}$ Commutative
- $\left.\begin{matrix} a \vee (b \vee c) = (a \vee b) \vee c \\ a \wedge (b \wedge c) = (a \wedge b) \wedge c \end{matrix}\right\}$ Associative
- $\left.\begin{matrix} a \vee (a \wedge b) = a \\ a \wedge (a \vee b) = a \end{matrix}\right\}$ Absorption law
- $a \le b \Rightarrow a \vee c \le b \vee c$
  $a \le b \Rightarrow a \wedge c \le b \wedge c$

  If a ≤ b and c ≤ d then

  a ∨ c ≤ b ∨ d

  a ∧ c ≤ b ∧ d

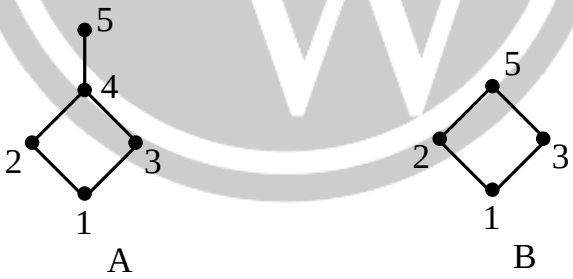
3.4.2 Sublattice:

If A is a lattice B is called

Sublattice of A iff

(i) B itself is a lattice.

(ii) 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 $\left.\begin{matrix} in\ A\ glb\ (2,3) = 4 \\ in\ \mathrm{B}\ glb\ (2,3) = 5 \end{matrix}\right]$ not equal $\therefore$ not sublatice

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.
- glb (a, b) = greatest element and lub (a, b) = 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 satisfises 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.
- [Dn : 1] is a distributive lattice for any n.
- It is because distributive properties hold for lcm & gcd.
- [Dn : 1] is a Boolean lattice if n is a square free number.
- [P(s); ≤] is a Boolean lattice.
- Every Boolean lattice with 2n elements, $\forall \cap \geq 0$.
  Every Boolean lattice contains 2^nelements is isomorphic to the lattice (P(s); ≤) 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 in 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 $(f_1 + f_2)(x)$
  $f_1(x).\ f_2(x)$ is denoted as $(f_1.f_2)(x)$
  $\frac{1}{f(x)}$ is denoted as $\frac{1}{f}(x)$, ($\frac{1}{f}$ 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):

$f : A \to B$ is one to one
iff
$\forall a \forall b(f(a) = f(b) \Rightarrow a = b)$

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

3.6.2 Onto function (Surjection):

$f : A \to B$ is onto iff
$\forall b \in B \quad \exists a \in A$ such that $f(a) = b$

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

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

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

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

A function $f : A \to B$ is Bijection iff
f is both one-one and onto

- If $f : A \to B$ is bijection, then $|A| = |B|$
- If $f : A \to B$ is 1 – 1 and $|A| = |B|$
  then f is a bijection
- If $f : A \to B$ 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 → A is a function and A is a finite set then
  A is 1–1 ⇔ A is onto

3.6.4 Inverse of a function:

f : A → B is invertible if its inverse relation f^{i} is a function from B to A.

f : A → B is invertible ⇔ 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 → B is a function and S ≤ B

Inverse image of S = {a∈A|f(a) ⊆ 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 → A is called an identity function if

I_A = {(x, x)/x∈A}

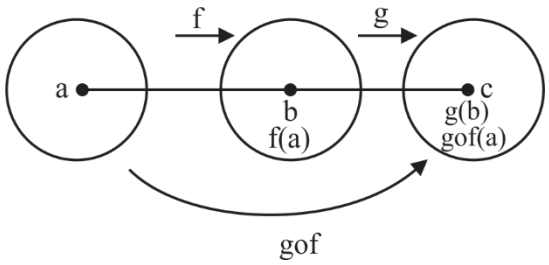
3.6.7 Constant function:

A function f : A → B is said to be a constant function if

f(x) = c ∀x ∈A

3.6.8 Composition of functions:

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

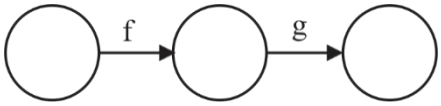


gof(x) = g(f(x))

- In gof : A → C
  A is domain of gof
  C is codomain of gof

Range of gof is image of (range of f under g)

- If $f : A \to B$ and $g : B \to C$ are tow functions
  gof is defined for every case



But fog is defined iff

range of $g \leq$ domain of f

fog maps from B to B i.e., $fog : B \to B$

$fog \neq gof$ (i.e., composition is not commutative)

(fog)oh = fo(goh) (i.e., Associative)

$(fog)^{-1} = g^{-1} of^{-1}$ (Think why)

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

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

$f^{-1}of : A \to A$ is an identity function I_A

$fof^{-1} : B \to B$ is an identity function I_B

Note:

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

3.6.9 Partial Functions:

$f: A \to B$ is called partial function if 'f' is defined only for some of $a \in A$.
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., $(a*b) \in A \ \forall a,b \in A$

3.7.2 Binary Structure (or) Algebraic Structure:

A nonempty set A is called binary structure with respect is a binary operator *, if * is binary operation on A. it is demoted 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
  $a^{-1} = a, \forall a \in G$
- $(\{0,1,2,3 \ldots\ldots m-1\}, \oplus_m)$ 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 ever raw 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 $\Leftrightarrow (ab)^{-1} = b^{-1}a^{-1} \ \forall a, b \in G$
- $\forall a \in G, a^{\circ} = e$

3.7.7 Homomorphism

- If (G, *) and (G', $\oplus$) are two groups then a function $f : G \to G'$ is called a homomorphism
  if $f(a*b) = f(a) \oplus f(b)$
- If $f : G \to G'$ is a bijection, then we call the homomorphism isomorphism. It is denote 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 $f : G \to G'$ 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.
- Order (a) = order (a^{-1}) $\forall a \in G$
- $a^{-n} = b^n$ 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 order(H) divides 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) $a^{-1} \in H \qquad \forall a, b \in H$
- 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

$$\text{order}(a^m) = \frac{n}{\gcd(n,m)}$$

- 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 <a> where

$\langle a \rangle = \{a^n / \forall n \in Z\}$

Also <a> is a cyclic subgroup.

order of the subgroup <a> = 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 n1 outcomes possible for the first step, n_2 outcomes possible for the second step, ... , and nm outcomes possible for the mth step, the total number of possible outcomes is

$$n_1 \cdot n_2 \cdot \cdot \cdot n_m$$

For a collection of m disjoint sets with n_1 elements in the first, n_2 elements in the second, . . ., and nm elements in the mth, the number of ways to choose one element from the collection is

$$n_1 + n_2 + \cdot \cdot \cdot + n_m$$

$$(x+y)^n = \sum_{k=0}^{n} \binom{n}{n-k} x^k y^{n-k}.$$

$$\sum_{i=0}^{n} C(n,i)(-1)^i = 0$$

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 $x_1^{n_1} x_2^{n_2} x_3^{n_3} \ldots x_t^{n_t}$ in the expansion of $(x_1 + x_2 + x_3 + \ldots + x_t)^n$ is

$$\frac{n!}{n_1!n_2!n_3!\ldots n_t!},$$

where each n_i is an integer with $0 \le n_i \le n$, for all $1 \le i \le t$, and $n_1 + n_2 + n_3 + \cdots + n_t = n$.

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

$x_1 + x_2 + \ldots + x_n = r,\ x_i \geq 0,\ 1 \leq i \leq n.$

(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.

$$b_n = \binom{2n}{n} - \binom{2n}{n-1} = \frac{1}{n+1}\binom{2n}{n}, \qquad n \geq 1, b_0 = 1.$$

The numbers $b_0, b_1, b_2, \ldots$ are called the Catalan numbers, after the Belgian mathematician

Order Is Relevant	Repetitions Are Allowed	Type of Result	Formula
Yes	No	Permutation	$P(n, r) = n!/(n – r)!,\ 0 \leq r \leq n$
Yes	Yes	Arrangement	$N^r,\ n, r \geq 0$
No	No	Combination	$C(n, r) = n!/[r!(n – r)!] = \binom{n}{r},\ 0 \leq r \leq n$
No	Yes	Combination with repetition	$\binom{n+r-1}{r},\ n, r \geq 0$

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 rth type, where $n_1 + n_2 + \ldots + n_r = n$, then there are $\frac{n!}{n_1!n_2!\ldots n_r!}$ (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) $|A \backslash B| = |A| - |A \cap B| \geq |A| - |B|$

(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) $|A \times B| = |A| - |B|$

Given a finite number of finite sets, $A_1, A_2, \ldots, A_n$, the number of elements in the union $A_1 \cup A_2 \cup \ldots \cup A_n$ is

$$|A_1 \cup A_2 \cup \ldots \cup A_n| = \sum_i |A_i| - \sum_{i<j} |A_i \cap A_j| + \sum_{i<j<k} |A_i \cap A_j \cap A_k| - \ldots + (-1)^{n+1} |A_1 \cap A_2 \cap \ldots \cap A_n|,$$

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

$$\boxed{D_n = n!\left(1 - \frac{1}{1!} + \frac{1}{2!} - \frac{1}{3!} + \ldots + (-1)^n \frac{1}{n!}\right).}$$

Let $a_0, a_1, a_2, \ldots$ be a sequence of real numbers. The function

$$f(x) = a_0 + a_1x + a_2x^2 + \ldots = \sum_{i=0}^{\infty} a_i x^i$$

is called the generating function for the given sequence.

4.1.2 Extended Binomial coefficient

$$\binom{-n}{r} = \frac{(-n)(-n-1)(-n-2)\ldots(-n-r+1)}{r!}$$

$$= \frac{(-1)^r (n)(n+1)(n+2)\ldots(n+r-1)}{r!}$$

$$= \frac{(-1)^n (n+r-1)!}{(n-1)!r!} = (-1)^r \binom{n+r-1}{r}.$$

For all m, n ∈ Z^+, a ∈ R,

(1) $(1 + x)^n = \binom{n}{0} + \binom{n}{1}x + \binom{n}{2}x^2 + \ldots + \binom{n}{n}x^n$

(2) $(1 + ax)^n = \binom{n}{0} + \binom{n}{1}ax + \binom{n}{2}a^2x^2 + \ldots + \binom{n}{n}a^nx^n$

(3) $(1 + x^m)^n = \binom{n}{0} + \binom{n}{1}x^m + \binom{n}{2}x^{2m} + \ldots + \binom{n}{n}x^{nm}$

(4) $(1 - x^{n+1})/(1 - x) = 1 + x + x^2 + \ldots. + x^n$

(5) $1/(1 - x) = 1 + x + x^2 + x^3 + \ldots = \sum_{i=0}^{\infty} x^i$

(6) $1/(1 - ax) = 1 + (ax) + (ax)^2 + (ax)^3 + \ldots$

$= \sum_{i=0}^{\infty} (ax)^i = \sum_{i=0}^{\infty} a^i x^i$

$= 1 + ax + a^2x^2 + a^3x^3 + \ldots$

(7) $1/(1+x)^n = \binom{-n}{0} + \binom{-n}{1}x + \binom{-n}{2}x^2 + \ldots$

$$= \sum_{i=0}^{\infty}\binom{-n}{i}x^i$$

$$= 1 + (-1)\binom{n+1-1}{1}x + (-1)^2\binom{n+2-1}{2}x^2 + \ldots$$

$$= \sum_{i=0}^{\infty}(-1)^i\binom{n+i-1}{i}x^i$$

(8) $1/(1-x)^n = \binom{-n}{0} + \binom{-n}{1}(-x) + \binom{-n}{2}(-x)^2 + \ldots$

$$= \sum_{i=0}^{\infty}\binom{-n}{i}(-x)^i$$

$$= 1 + (-1)\binom{n+1-1}{1}(-x) + (-1)^2\binom{n+2-1}{2}(-x)^2 + \ldots$$

$$= \sum_{i=0}^{\infty}\binom{n+i-1}{i}x^i$$

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 $k \geq 0$,

$$c_k = a_0b_x + a_1b_{x-1} + \ldots + a_{k-1}b_1 + a_kb_0 = \sum_{j=0}^{k}a_j b_{k-j}.$$

Objects Are Distinct	Containers Are Distinct	Some Container(s) May Be Empty	Number of Distributions
No	Yes	Yes	$\binom{n+m-1}{m}$
No	No	Yes	(1) p(m), for n = m (2) p(m, 1) + p(m, 2) + … + p(m, n), for n < m
No	Yes	No	$\binom{n+(m-n)-1}{(m-n)} = \binom{m-1}{m-n} = \binom{m-1}{n-1}$
No	No	No	p(m, n)

Consider the nonhomogeneous first-order relation

$$a_n + C_1a_{n-1} = kr^n,$$

where k is a constant and $n \in Z^+$. If r^n is not a solution of the associated homogeneous relation

$$a_n + C1a_{n-1} = 0,$$

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 $a_n^{(h)} = c_1r^n + c_2r_n^n$, 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 $C_0a_n + C_1a_{n-1} + C_2a_{n-2} + \ldots + C_ka_{n-k} = f(n)$, 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, $t \in Z^+$	$A_tn^t + A_{t-1}n^{t-1} + \ldots. + A_1n + A_0$
r^n, $r \in R$	Ar^n
sin θn	A sin θn + B cos θn
cos θn	A sin θn + B cos θn
$n^t r^n$	$r^n (A_tn^t + A_{t-1}n^{t-1} + \ldots + A_1n + A_0)$
r^n sin θn	Ar^n sin θn + Br^n cos θn
r^n cos θn	Ar^n sin θn + Br^n cos θn

❑❑❑

GATE Exam 2024?

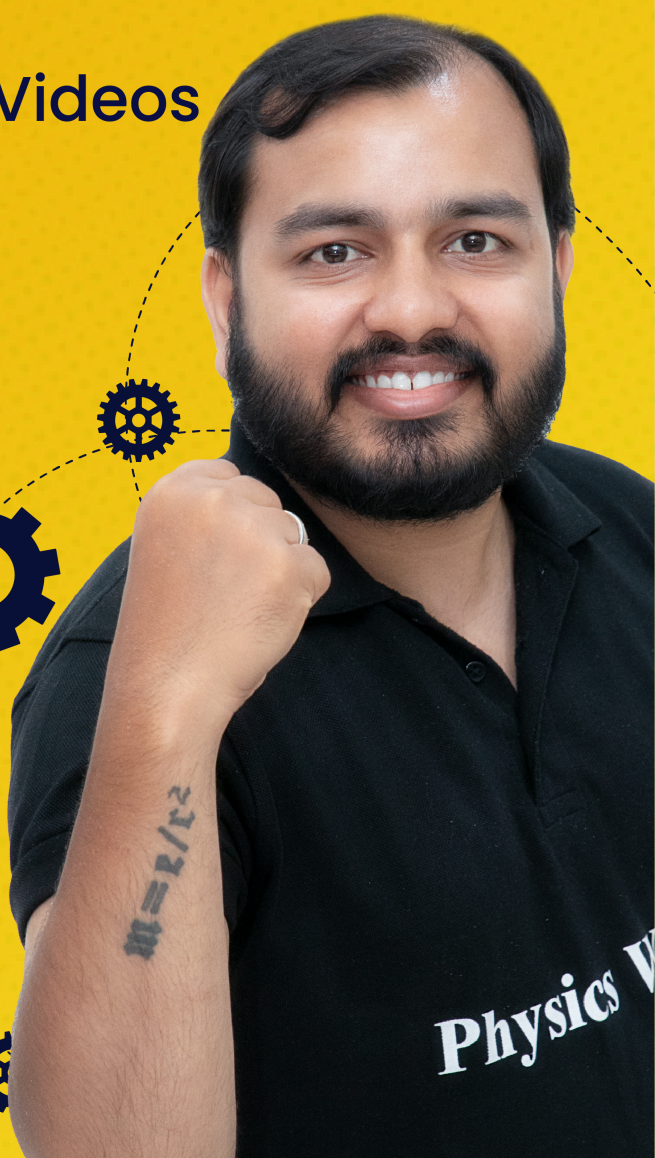
PARAKRAM BATCH

ME | CE | EE | EC | CS & IT

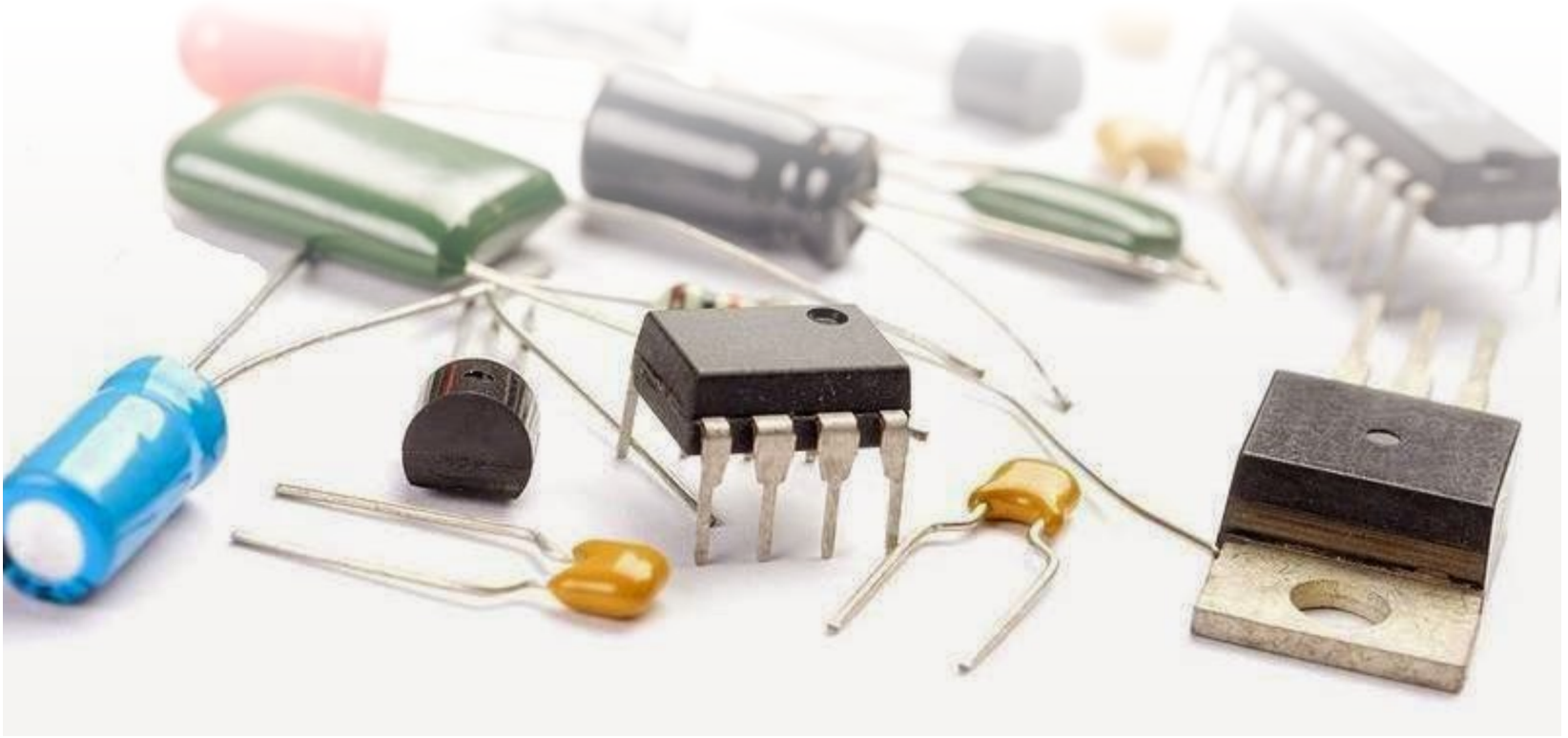
- Live Classes & Recorded Videos
- DPPs & Solutions
- Doubt Solving
- Batches are available in ***English***

Weekday & Weekend Batches Available

JOIN NOW!



Digital Logic



Digital Logic

INDEX

1 LOGIC GATE

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.

$A \xrightarrow{\text{NOT}} \bar{A}$ or A' (Complementation law)

and $\bar{\bar{A}} = A \Rightarrow$ 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.

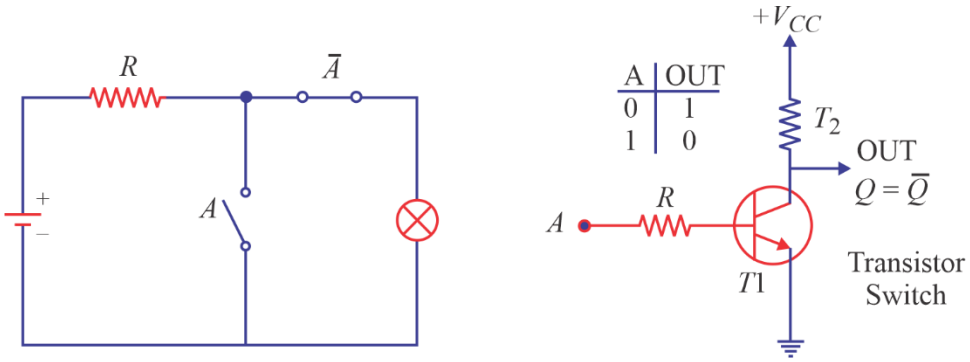
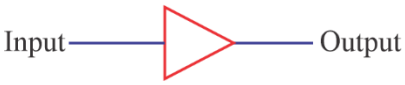


Fig. 1.2.

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.

"Buffer" gate



Input	Output
0	0
1	1

1.1.2. AND Operation

Symbol:

$A.A = A, A.0 = 0, A.1 = A, A\bar{A} = 0$

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.

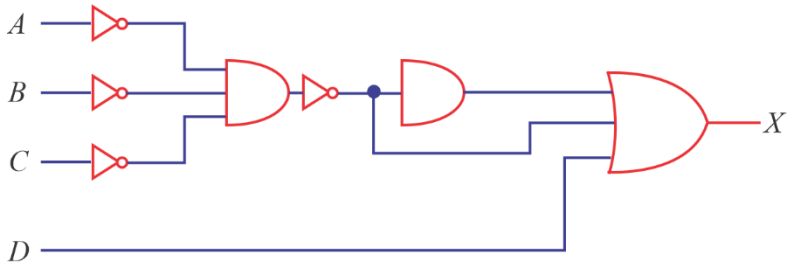


Fig. 1.3.

Solution:

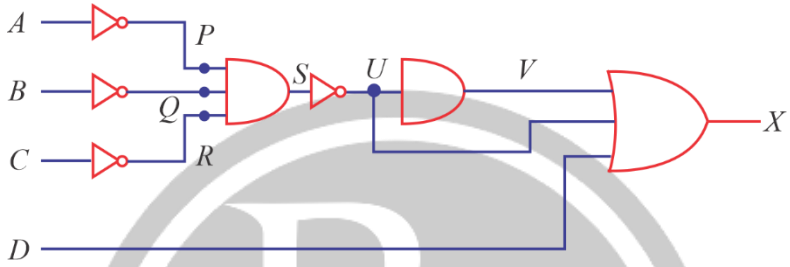


Fig. 1.4.

$P = \bar{A}, Q = \bar{B}, R = \bar{C}$

$S = \bar{A} \cdot \bar{B} \cdot \bar{C}$

$V = U = \overline{\bar{A}\bar{B}\bar{C}}$

$X = U + V + D$

$= \overline{\bar{A}\bar{B}\bar{C}} + \overline{\bar{A}\bar{B}\bar{C}} + D$

$= A + B + C + D$

Fig. 1.5.

Enable or Disable Input:

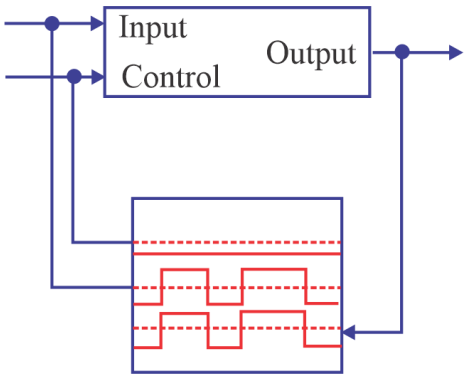


Fig. 1.6.

Enable:

- Allow a signal to pass when the control signal is HIGH.
- Prevent a signal from passing when the control signal is LOW.

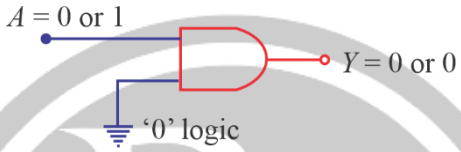
Disable:

- 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.

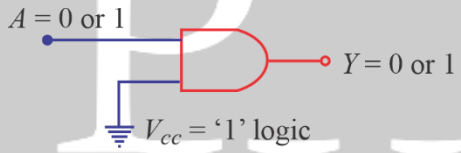
For a two input AND gate:

For a two input OR gate:

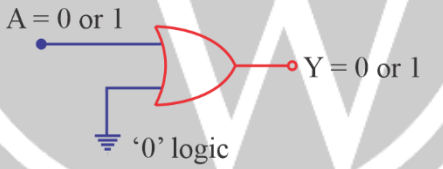
- **Control '0' disable**



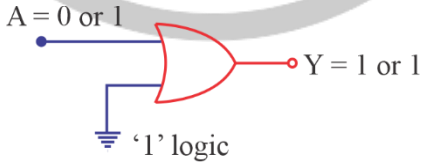
- **Control '1' enable (Buffer)**

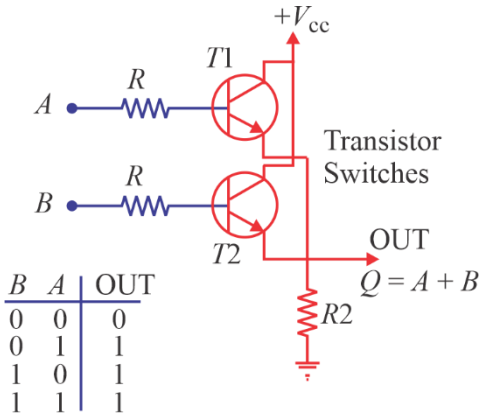


- **Control '0' enable (Buffer)**



- **Control '1' Always enable**





B	A	OUT
0	0	0
0	1	1
1	0	1
1	1	1

1.1.4. Switch Diagram for AND/OR gate

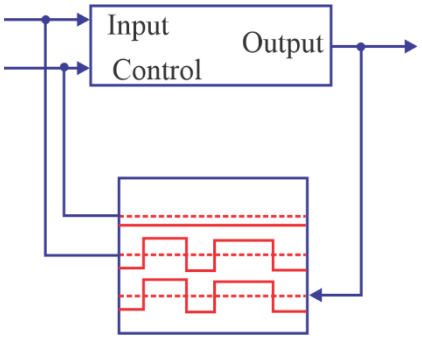


Fig. 1.7.

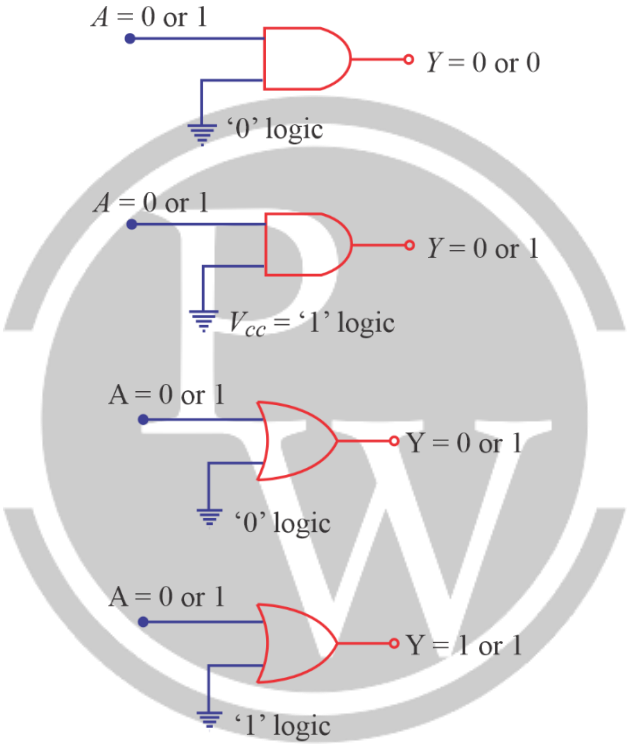


Fig. 1.8.

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.

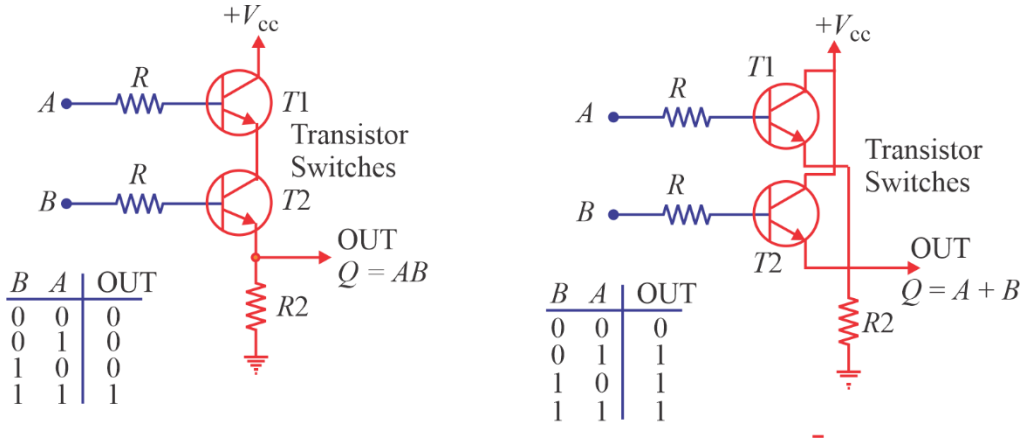


Fig. 1.9.

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.

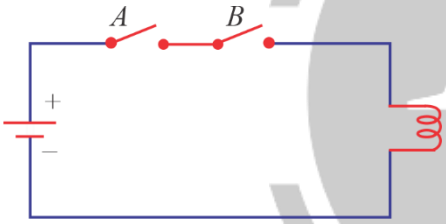


Fig. 1.10.

The circuit shown below shows the switch representation of OR gate which is basically the parallel connection of switches A and B.

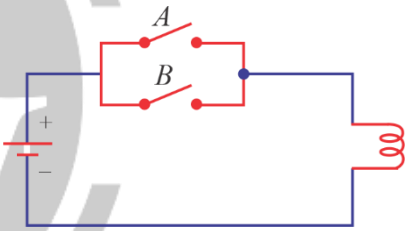
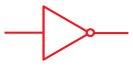
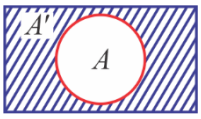
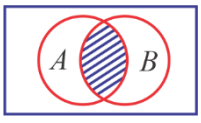
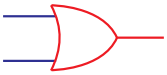


Fig. 1.11.

1.1.8. Venn Diagram:

NOT			$\bar{A}$	A		Output
				0 1		1 0
AND			A.B	A	B	Output
				0 0 1 1	0 1 0 1	0 0 0 1
OR			A + B	A	B	Output
				0 0 1 1	0 1 0 1	0 1 1 1

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:

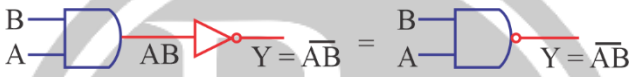


Fig. 1.12.

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

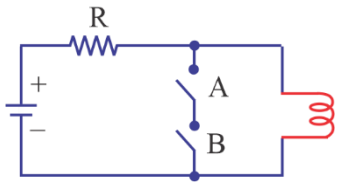


Fig. 1.13.

NAND gate acts as Universal Gate

Logic Gates using only NAND Gates

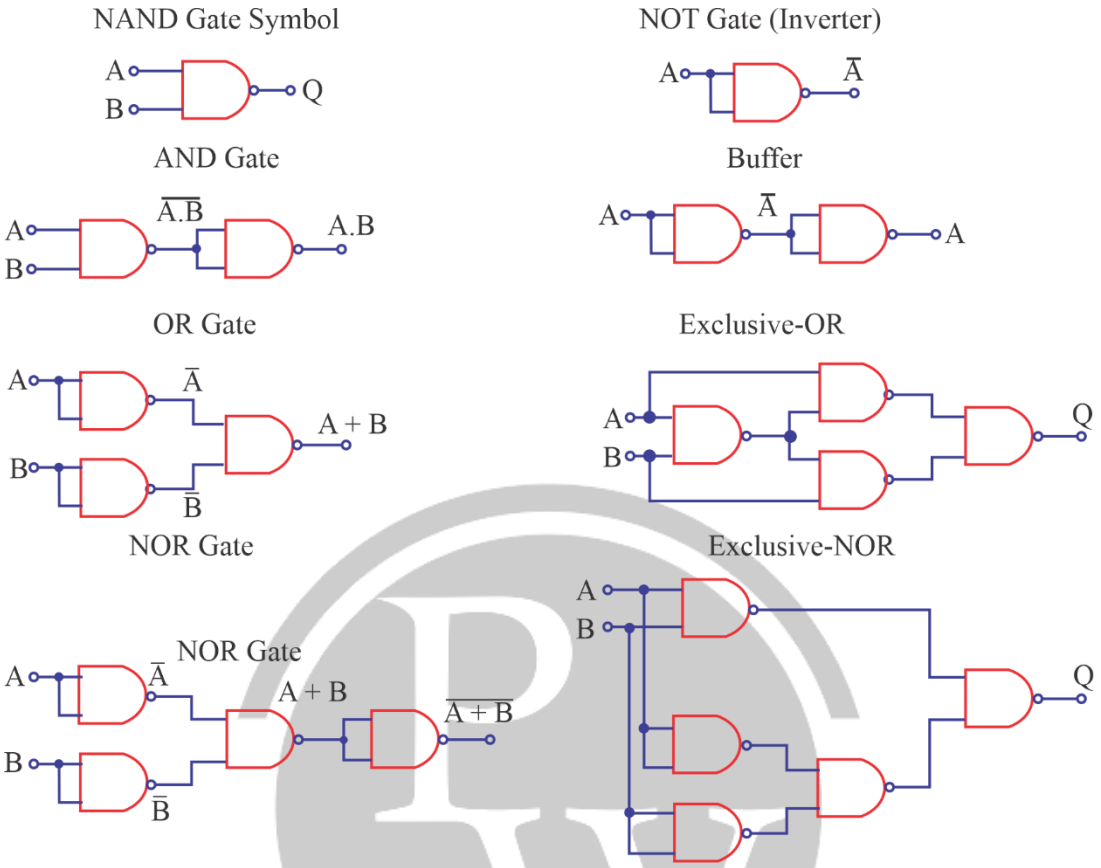


Fig. 1.14.

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

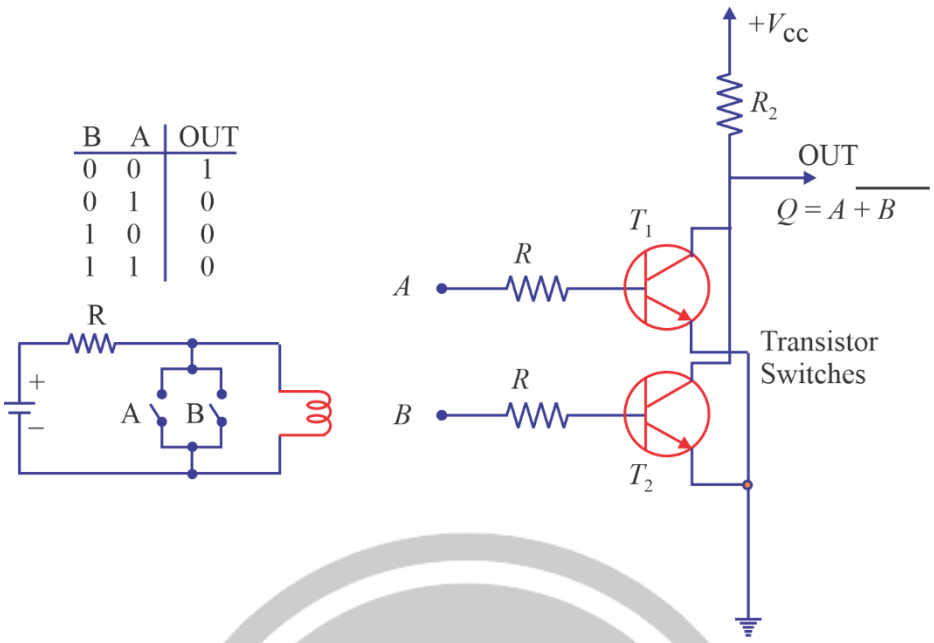


Fig. 1.15.

NOR gate acts as Universal Gate.

Logic Gates using only NOR Gates

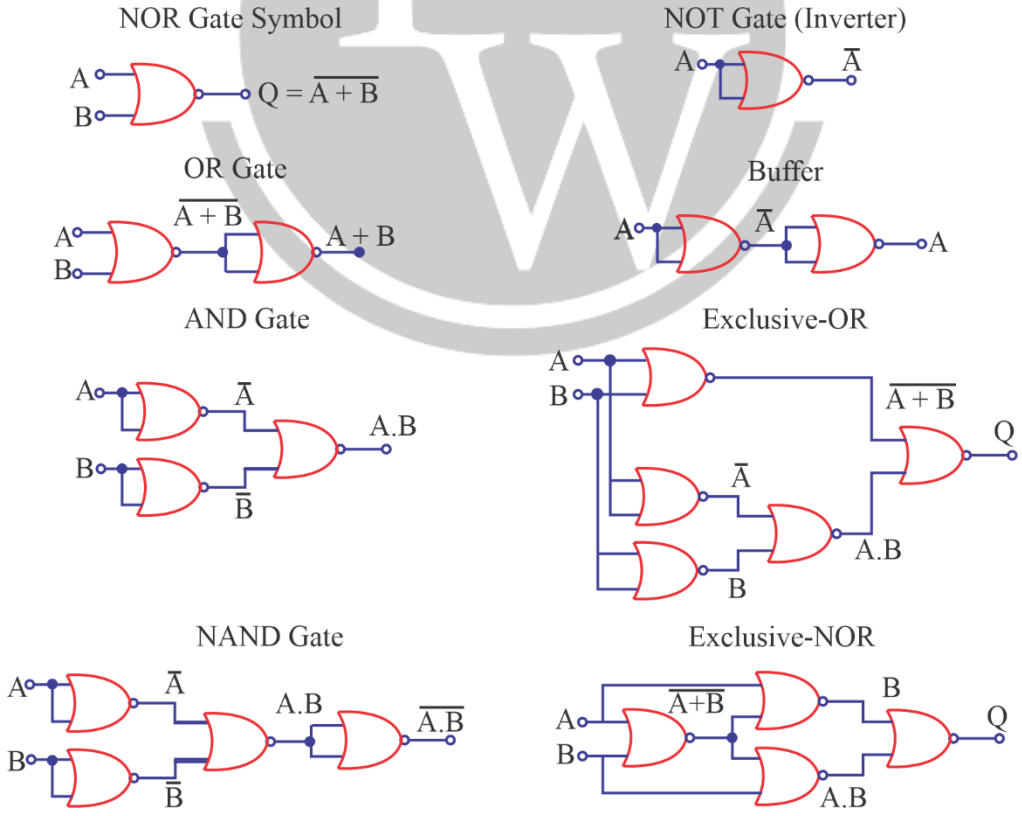
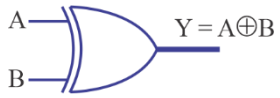


Fig. 1.16.

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

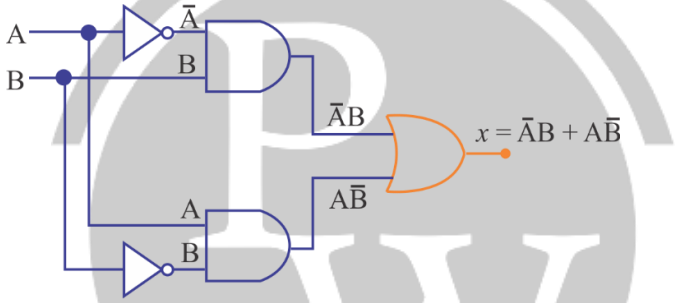
Symbol of two input XOR gate



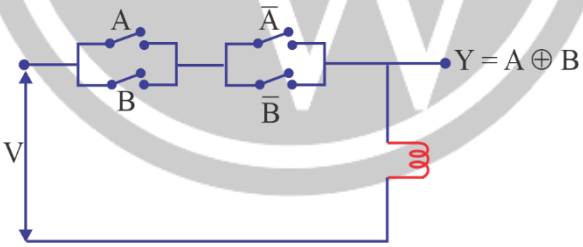
Truth table for 2-input XOR gate

Input		Output
A	**B**	**Y = A ⊕ 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



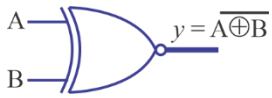
Truth Table:

A	B	Y
0	0	0
0	1	1
1	0	1
1	1	0

$$Y = (A + B)(\bar{A} + \bar{B})$$
$$= \bar{A}B + A\bar{B}$$
$$= A \oplus B$$

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 ⊙ B
0	0	1
0	1	0
1	0	0
1	1	1

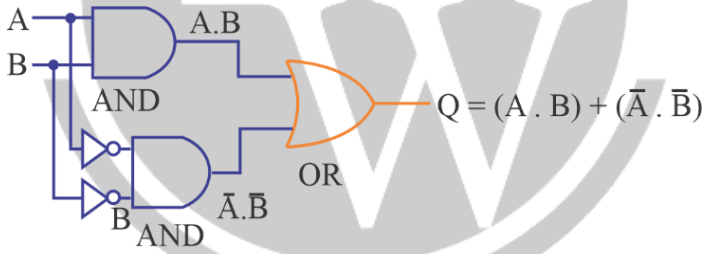
Boolean expression for EX-NOR gate is $Y = \overline{A \oplus B}$

Apply De-Morgan's theorem:

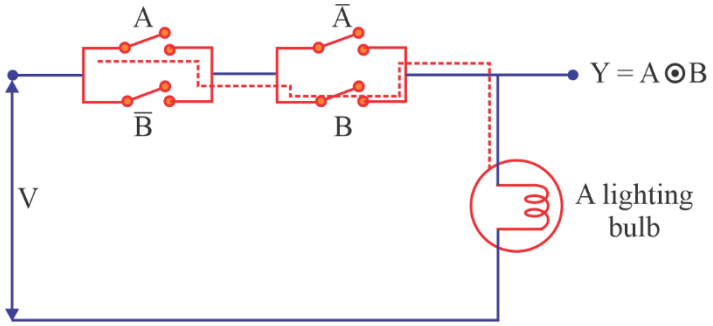
$$\overline{A \oplus B} = \overline{\bar{A}B + A\bar{B}} = \overline{\bar{A}B} \cdot \overline{A\bar{B}} = \left(A + \bar{B}\right)\left(\bar{A} + B\right) = AB + \bar{A}\bar{B}$$

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

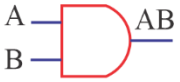
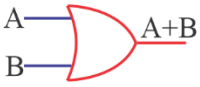
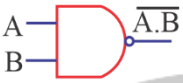


Truth Table:

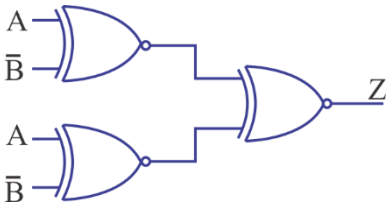
A	B	Y
0	0	1
0	1	0
1	0	0
1	1	1

$$Y = (A + \bar{B})(\bar{A} + \bar{B})$$
$$= AB + \bar{A}\bar{B}$$
$$= A \oplus B$$

1.3. Alternate Logic Gate Representation

Logic	Normal symbol	Alternate symbol
NOT		
AND		
OR		
NAND		
NOR		

Example: In the following circuit, the find the output Z?

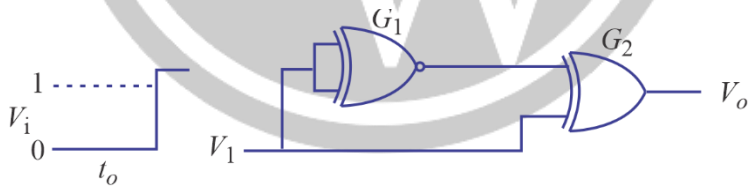


Solution: From the given circuit, we can observe that input to last XNOR is same, so, the XNOR output is given by (let input is X)

$$Z = X.X + \overline{X}.\overline{X} = X + \overline{X} = 1$$

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.

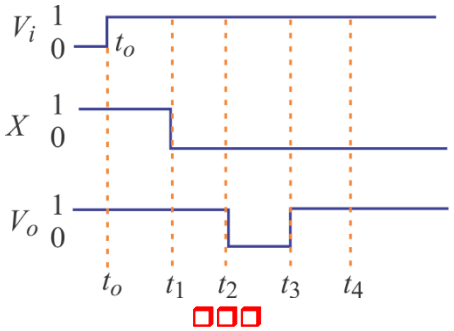


If input V_i makes an abrupt change from logic 0 to 1 at t = t_0, then find the output waveform V_0?

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:	0.0 = 0	0 + 0 = 0	$\bar{1} = 0$
Axioms 2:	0.1 = 0	0 + 1 = 1	$\bar{0} = 1$
Axioms 3:	1.0 = 0	1 + 0 = 1	
Axioms 4:	1.1 = 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 : if A = 0 then $\bar{A} = 1$

4 : $\bar{\bar{A}}$ = A (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: $A \cdot 0 = 0$ (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.

I. $\overline{A.B} = \bar{A} + \bar{B}$

II. $\overline{A + B} = \bar{A}.\bar{B}$

The above two laws can be extended for 'n' variables as,

$\overline{A_1.A_2.A_3... + A_n} = \bar{A}_1 + \bar{A}_2 + ...\bar{A}_n$ and $\overline{A_1 + A_2 + ... + A_n} = \bar{A}_1.\bar{A}_2...\bar{A}_n$

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 $f = \left(AB + \bar{A}\bar{B}\right)$

$$\text{Complement of } \bar{f} = \overline{AB + \bar{A}\bar{B}} = \overline{AB} \cdot \overline{\bar{A}\bar{B}}$$

$$= \left(\bar{A} + \bar{B}\right)\left(A + B\right)$$

$$= A\bar{A} + A\bar{B} + B\bar{A} + B\bar{B} = A\bar{B} + \bar{A}B$$

Example : Show that

$$AB + B\bar{C} + AC = AC + B\bar{C}$$

Solution : LHS = $AB + B\bar{C} + AC$

$$= AB\left(C + \bar{C}\right) + B\bar{C}\left(A + \bar{A}\right) + A\left(B + \bar{B}\right)C$$

$$= ABC + AB\bar{C} + AB\bar{C} + \bar{A}B\bar{C} + ABC + A\bar{B}C$$

$$= ABC + AB\bar{C} + \bar{A}B\bar{C} + A\bar{B}C$$

$$= AC\left(B + \bar{B}\right) + B\bar{C}\left(A + \bar{A}\right) = AC + B\bar{C}$$

= RHS

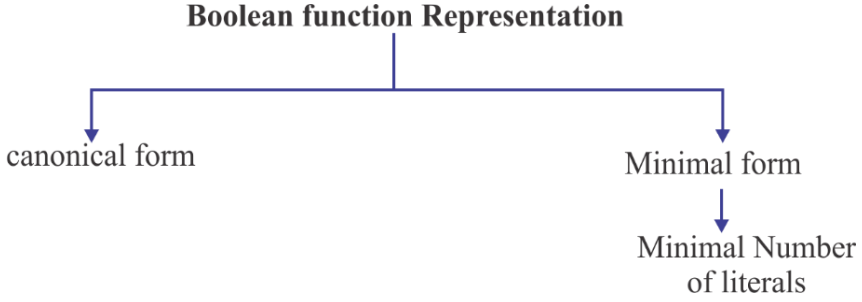
2.4. Minimization of Boolean function:

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.... 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.

$$Y = \overline{A}BC + A\overline{B} + AC$$

$$Y = A\overline{B} + B\overline{C}$$

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:

$f(A, B, C) = \Sigma m(3, 5, 6, 7)$

$Y = m_3 + m_5 + m_6 + m_7$

Also, $Y = \overline{A}BC + A\overline{B}C + AB\overline{C} + ABC$

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: $Y = \left(A+\overline{B}+C\right)\left(B\overline{C}+D\right)$

Each individual term in standard POS form is called maxterm.

POS forms are used to write logical expression for output be coming logic '0'.

we get f(A, B, C) = πM(0, 1, 2, 4)

$Y = M_0.M_1.M_2.M_4$

$Y = \left(A + B + C\right)\left(A + B + \overline{C}\right)\left(A + \overline{B} + C\right)\left(\overline{A} + B + C\right)$

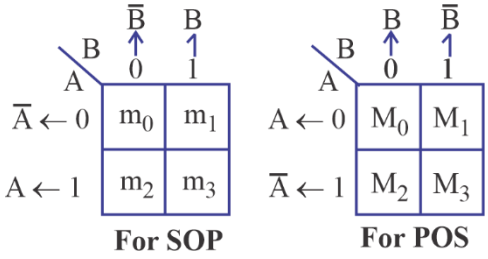
∴ We can also conclude from Table 2 and from above equations:

if Y = Σm(3, 5, 6, 7) or Y = πM(0, 1, 2, 4)

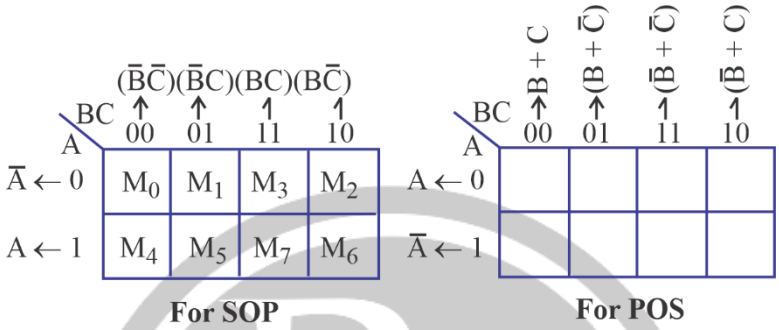
2.6. Karnaugh Map (K-MAP)

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.

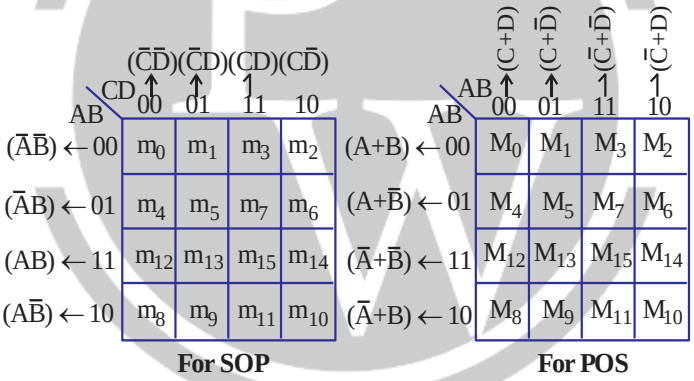
2.6.1. Two variable K-map



2.6.2. Three variable K-map



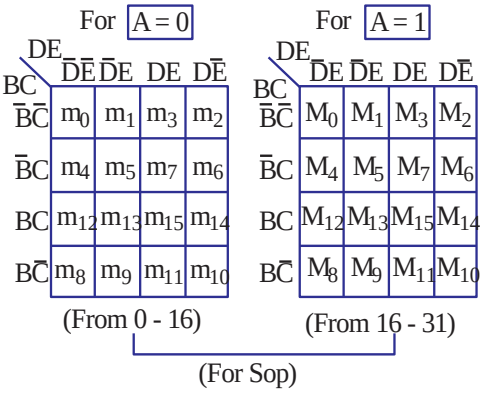
2.6.3. Four Variable K-map



2.6.4. Five variable K-map

- 32 cells
- 32 Minterms (Maxterms)

Here, we have *f*(A, B, C, D, E)



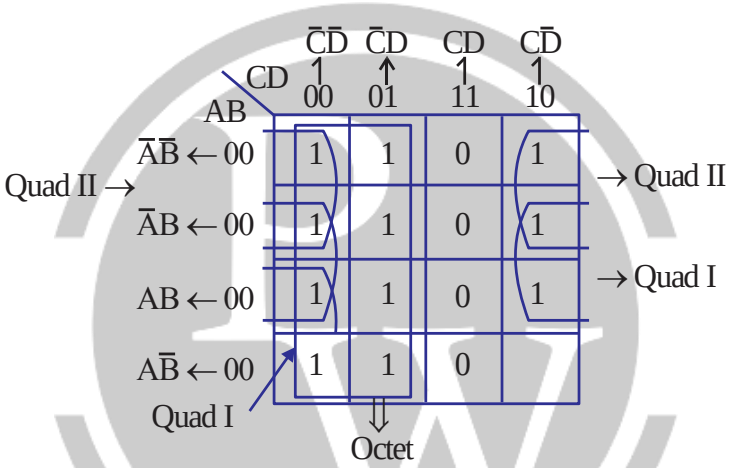
2.7. Simplification Rules

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

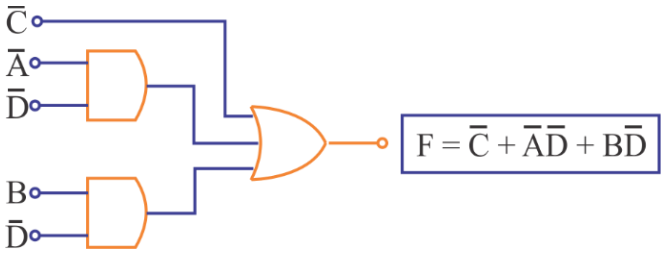
$f(A, B, C, D) = \Sigma m(0, 1, 2, 4, 5, 6, 8, 9, 12, 13, 14)$ also implement the simplified expression with AND-OR logic.

Solution:



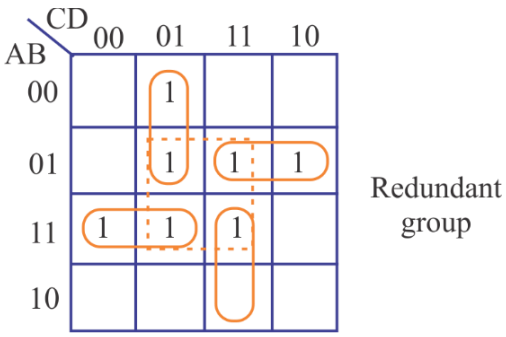
$$\therefore \quad f = \underset{\text{octet}}{\bar{C}} + \underset{\text{quad - II}}{\bar{A}\bar{D}} + \underset{\text{quad - I}}{B\bar{D}}$$

⇒ 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: Simply the given equation in part (i) and (ii)

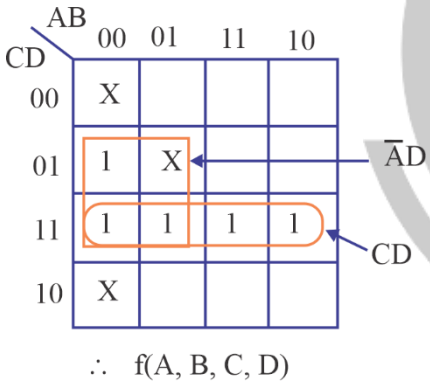
(i) In terms of SOP. and don't care conditions

$f(A, B, C, D) = \Sigma m(1, 3, 7, 11, 15) + d(0, 2, 5)$

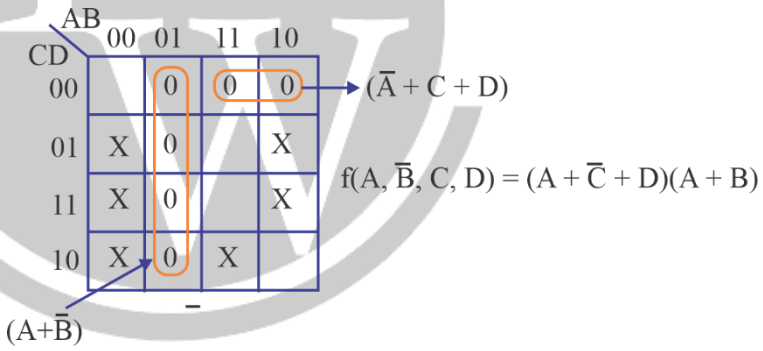
(ii) In terms of POS and don't care conditions.

$f(A, B, C, D) = \pi M(4, 5, 6, 7, 8, 12) + d(1, 2, 3, 9, 11, 14)$

Solution:



(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)

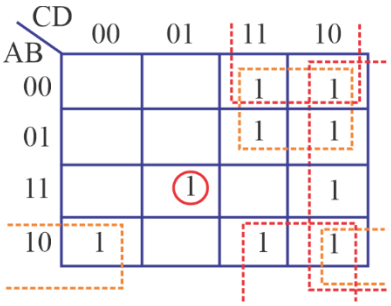
Prime implicant is a smallest possible product term of the given function,

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 :

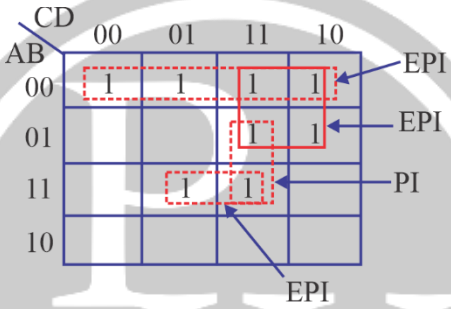


$F = AB\bar{C}D + A\bar{B}\bar{D} + \bar{A}C + \bar{B}C + C\bar{D}$

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

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.

ωx / yz	00	01	11	10
00	0	x	0	x
01	x	x	x	1
11	0	x	1	0
10	0	1	x	0

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 :

ωx / yz	00	01	11	10
00	0	x	0	x
01	x	x	x	1
11	0	x	1	0
10	0	1	x	0

$= xy + \bar{y}z$

❐❐❐

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.

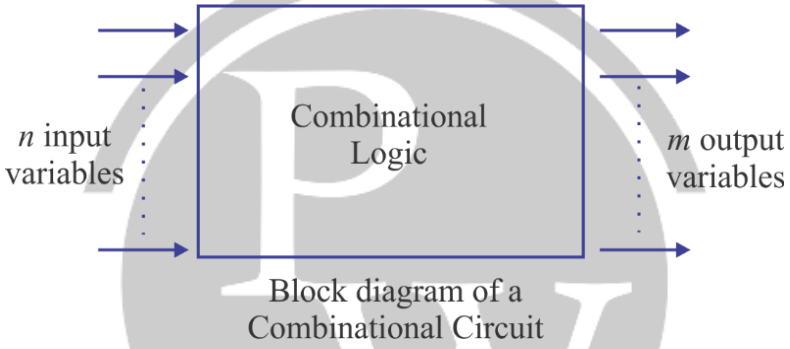


Fig. 3.1. Block diagram of a Combinational Circuit

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).

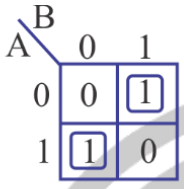
Fig. 3.2. Block Diagram of a 2-bit Half Adder

The truth table of half adder: where A and B are the inputs and sum and carry

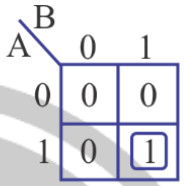
Table 1: Truth Table of Half Adder

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:



K-map for sum output



K-map for carry output

Sum, $S = \bar{A}B + A\bar{B} = (A \oplus B)$

Carry, $C = AB$

Logic Diagram:

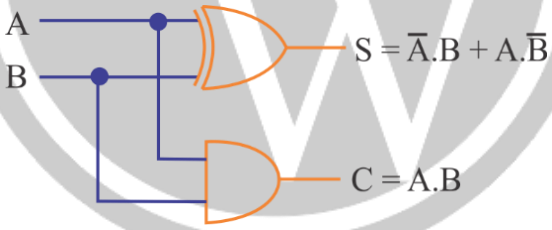


Fig. 3.3. Logic Diagram of Half Adder

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.

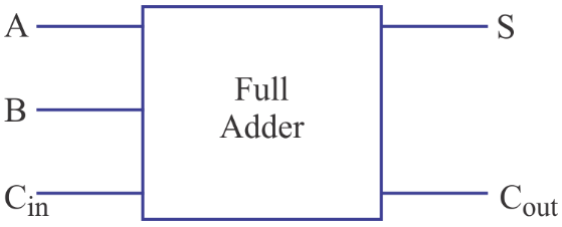


Fig. 3.4. Block Diagram of a Full Adder

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:

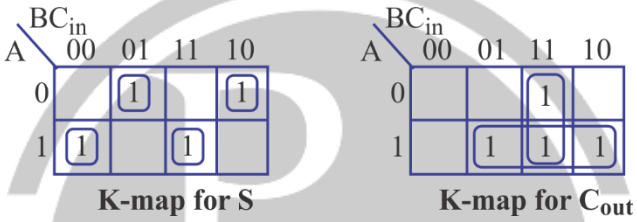


Fig. 3.5.

Simplified Boolean expressions for the sum (S) and carry (C_{out}) output from K-maps is

$$\text{Sum,} \quad S = \bar{A}\bar{B}C_{in} + \bar{A}B\bar{C}_{in} + ABC_{in} + A\bar{B}\bar{C}_{in} = C_{in}\left(\bar{A}\bar{B} + AB\right) + \bar{C}_{in}\left(\bar{A}B + A\bar{B}\right)$$

$$= C_{in}\left(A \odot B\right) + C_{in}\left(A \oplus B\right)$$

$$= C_{in}\left(\overline{A \oplus B}\right) + C_{in}\left(A \oplus B\right)$$

$$\text{Sum,} \quad S = C_{in} \oplus A \oplus B$$

$$\text{Carry,} \quad C_{out} = AB + AC_{in} + BC_{in}$$

Logic Diagram:

We can realize logic diagram of a full adder using gates as shown in below figure:

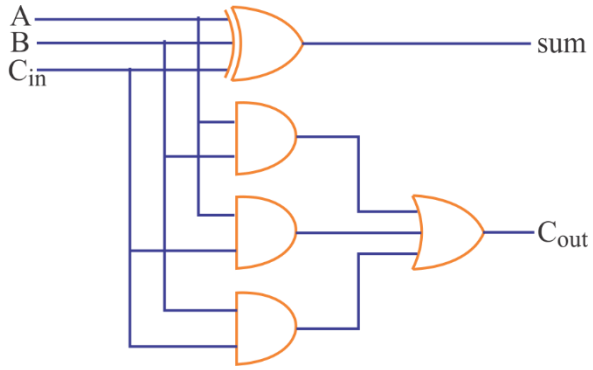


Fig. 3.6. Logic Diagram of Full Adder

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:

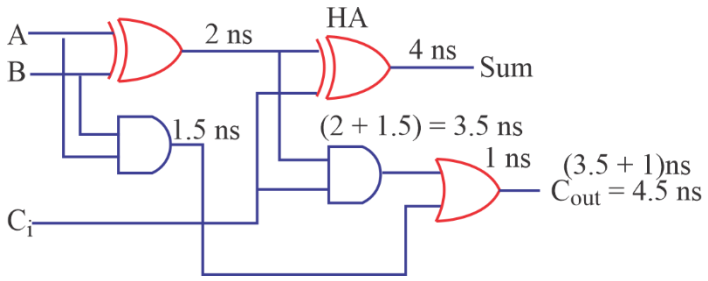


Fig. 3.7.

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.

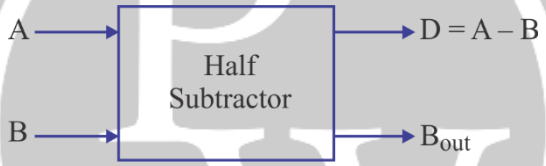


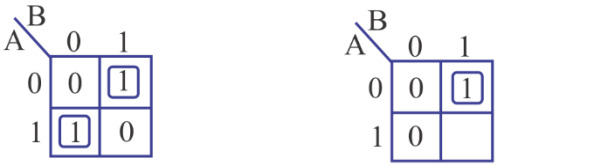
Fig. 3.8. Block diagram of a half subtractor

The truth table of half – subtractor, where A, B are the inputs, and difference (D) and borrow (B) 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 diffrence 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:

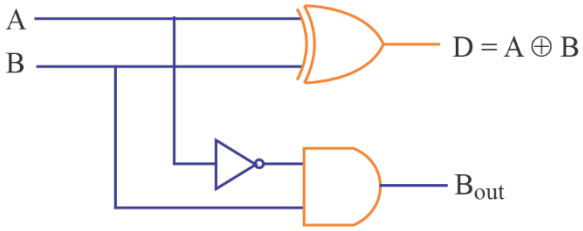


Fig. 3.9. Logic Diagram of a Half subtractor

3.3.2. Full Subtractor

A full subtractor performs subtraction operation on two bits, a minuend and a subtrahend.

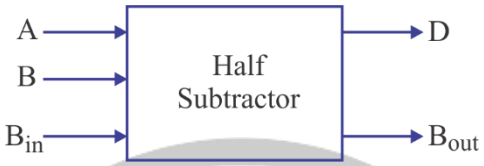


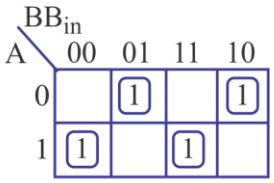
Fig. 3.10. Block Diagram of a Full subtractor

Truth table of Full subtractor:

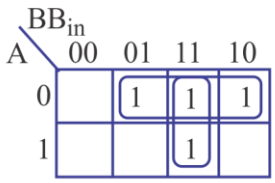
ble: Truth table of Full subtractor

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, $D = \bar{A}\bar{B}B_{in} + \bar{A}B\bar{B}_{in} + A\bar{B}\bar{B}_{in} + ABB_{in}$

$= B_{in}\left(AB + \bar{A}\bar{B}\right) + \bar{B}_{in}\left(A\bar{B} + \bar{A}B\right)$

$= B_{in}\left(\overline{A \oplus B}\right) + B_{in}\left(A \oplus B\right)$

$D = A \oplus B \oplus B_{in}$

Borrow, $B_{in} = \bar{A}B + \bar{A}B_{in} + BB_{in}$

Logic Diagram

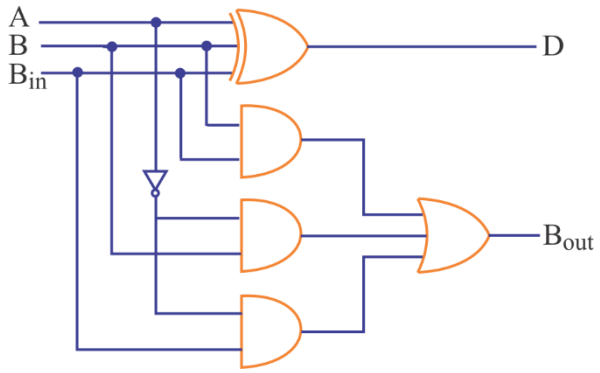


Fig. 3.11. Logic Diagram of a Full Subtractor

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.

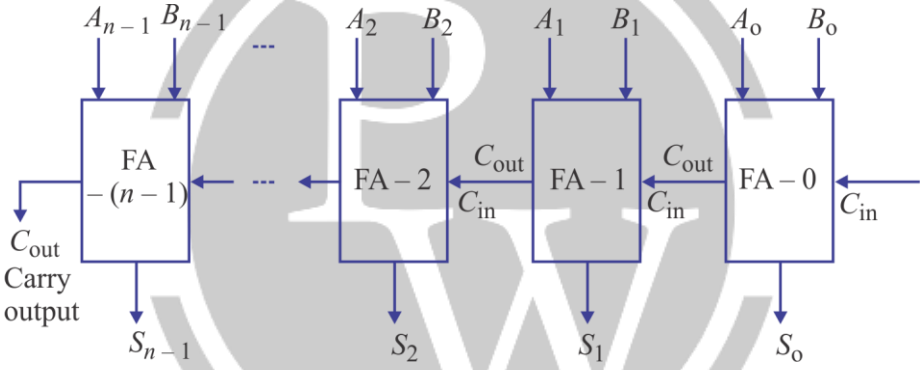


Fig. 3.12. n-bit Binary Adder

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 n^{th}, then

$$G_n = A_n \cdot B_n$$

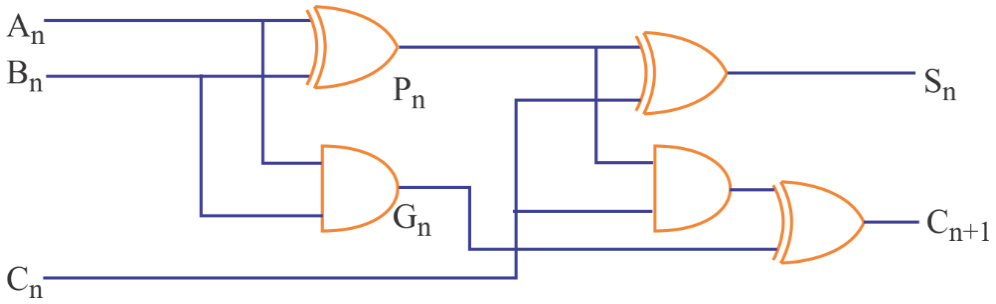


Fig. 3.13. Carry Look – ahead Generator Circuit

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

$$S_n = P_n \oplus C_n$$

$$C_{n+1} = G_n + P_n \cdot C_n$$

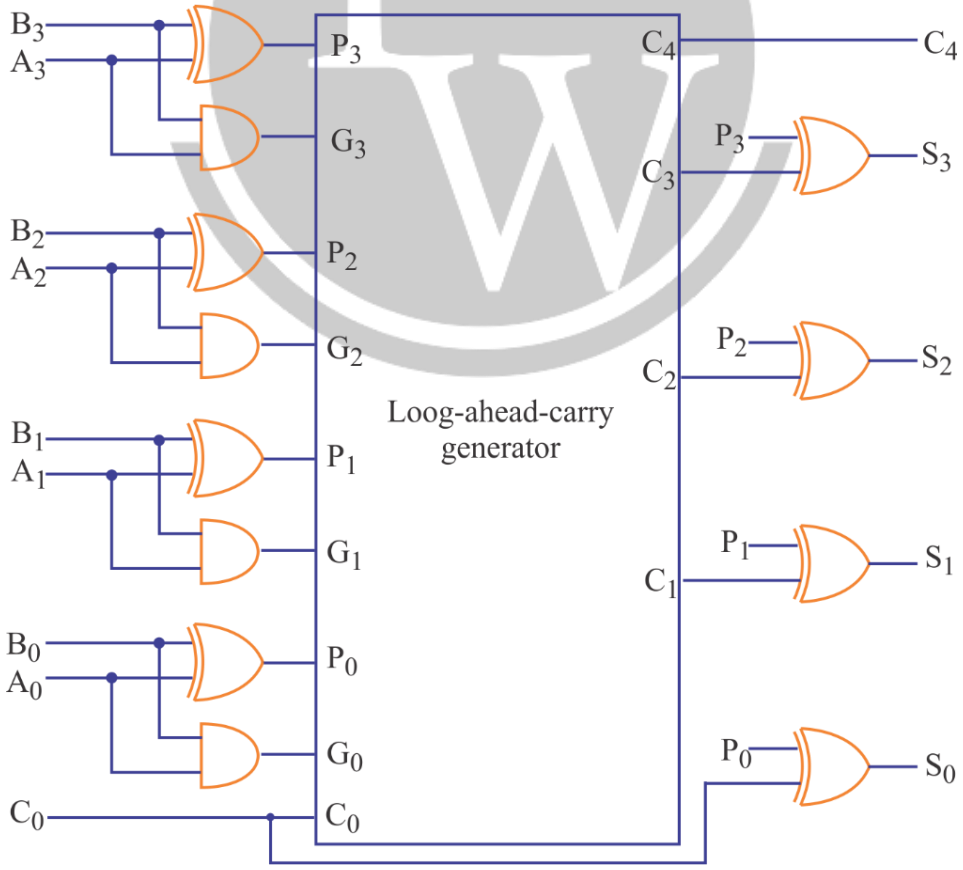


Fig. 3.14. 4-bit Full Adder with a look Ahead Carry Generator

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:

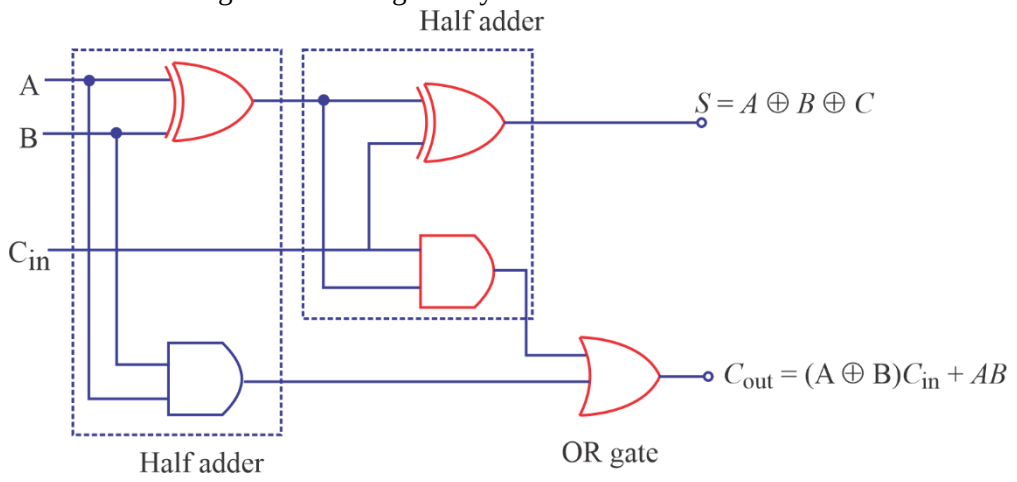


Fig. 3.15.

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.

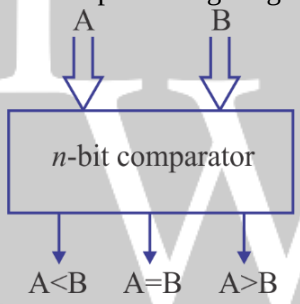


Fig. 3.16. Block diagram of digital comparator

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 $(A = B)$, $Y = \bar{A}_0 \bar{B}_0 + A_0 B_0 = \overline{A_0 \oplus B_0}$

For $(A > B)$, $Z = A_0 \bar{B}_0$

Logic Diagram of 1-bit Comparator:

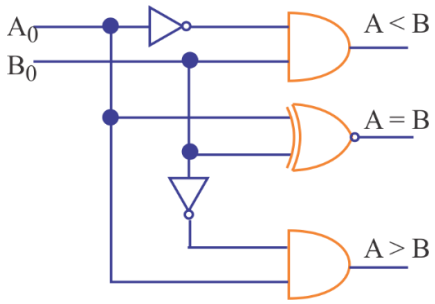


Fig. 3.17. Logic Diagram of 1-bit Comparator

Example: The circuit shown in given figure is:

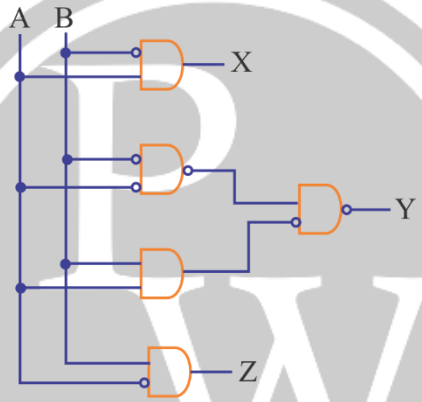


Fig. 3.18.

Solution: From the logic circuit shown in the figure, we obtain the following results,

$X = A\bar{B}$

$Y = \overline{(\bar{A}\bar{B})}\overline{(AB)} = \bar{A}\bar{B} + AB = A \odot B$

$Z = \bar{A}B$

So, we obtain the truth table for the above function as shown below.

From truth table, we deduce the following results.

It $A > B$, then $x = 1$

It $A = B$, the $y = 1$

It $A < B$, then $z = 1$

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

Therefore, it is a comparator circuit.

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.

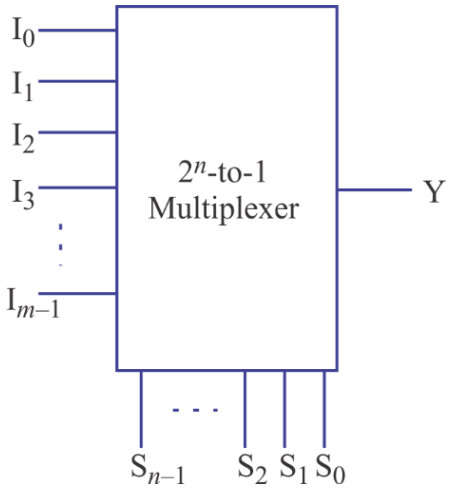


Fig. 3.19. Block diagram of a 2^n to 1 multiplexer

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 I_0 and I_1, one select input S, and one output.

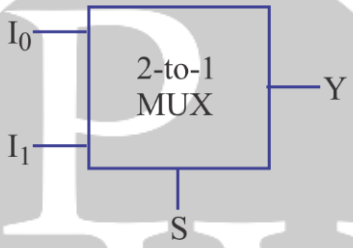


Fig. 3.20. Schematic block diagram of 2:1 Multiplexer

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,

$$Y = I_0\bar{S}_0 + I_1S_0$$

Realization of a 2:1 MUX using Logic Gates:

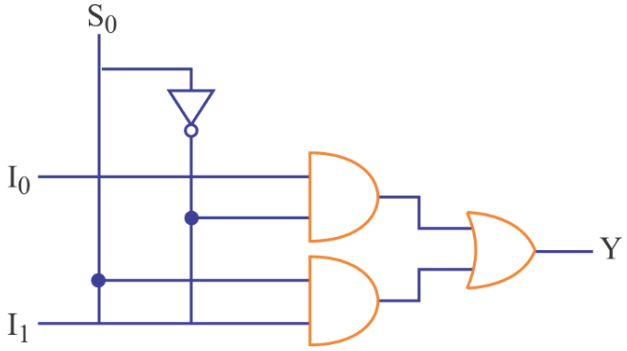


Fig. 3.21. Logic Diagram of a 2 x 1 Multiplexer

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.

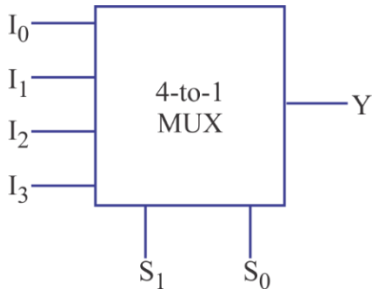


Fig. 3.22. Schematic block diagram of 4 x 1 MUX

Truth Table of a 4-to-1 Multiplexer

Select Inputs S_1	Select Inputs S_0	Output 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

$$Y = I_0\bar{S}_1\bar{S}_0 + I_1\bar{S}_1S_0 + I_2S_1\bar{S}_0 + I_3S_1S_0$$

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.

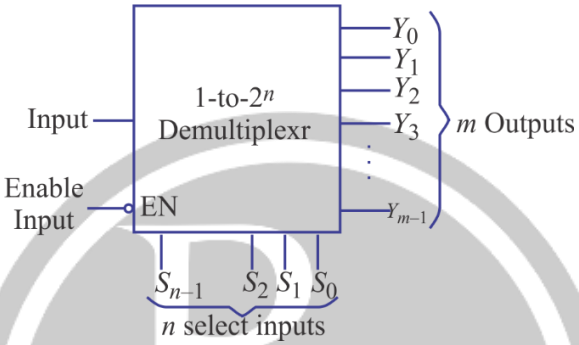


Fig. 3.23. Block Diagram of m-output Demultiplexer

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 × 1, it requires only one control (select) line.

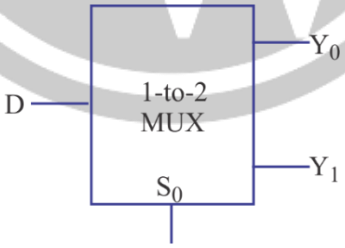


Fig. 3.24. Logic Diagram of 1 x 2 De-MUX

Truth table of a 1-to-2 demultiplexer

Table: Truth table of a 1-to-2 demultiplexer

Input S	Select input S_0	Output Y_1	Output Y_0
D	0	0	D
D	1	D	0

Thus, the Boolean expressions for the outputs can be written as

$$Y_0 = D\bar{S}_0 \ \& \ Y_1 = DS_0$$

Realization of a 1 x 2 Demultiplexer using Logic Gates:

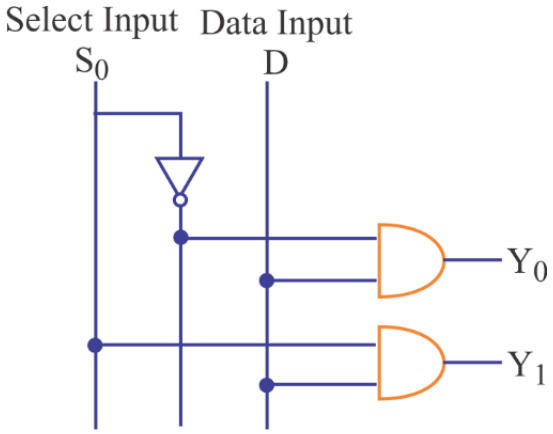


Fig. 3.25. Logic Diagram of 1 × 2 Demultiplexer

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.

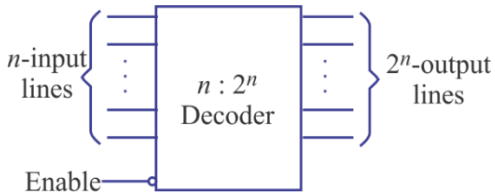


Fig. 3.26. Block Diagram of n-to-2^n Decoder

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 $m \leq 2n$.

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.

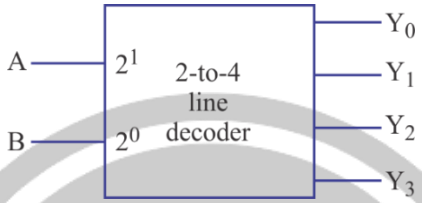


Fig. 3.27. Block Diagram of a 2 to 4 Line Decoder

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:

$$Y_0 = \overline{A}\,\overline{B} \text{ and } Y_1 = \overline{A}B$$

$$Y_2 = A\overline{B} \text{ and } Y_3 = AB$$

Realization of a 2 to 4 Line Decoder using Logic Gates

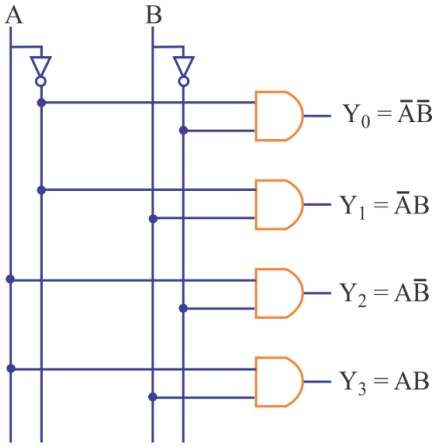


Fig. 3.28. Logic Diagram of a 2 to 4 Line Decoder

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.

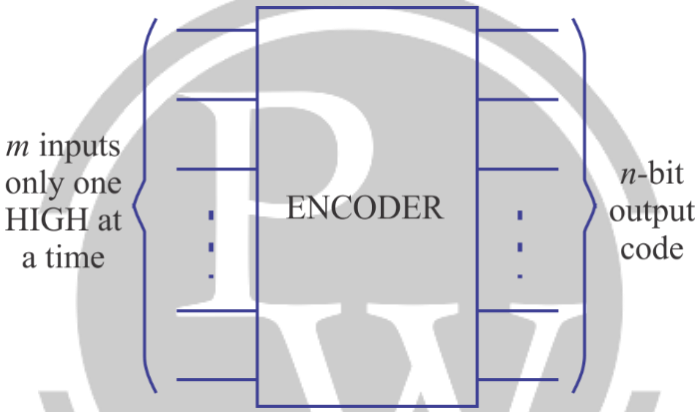


Fig. 3.29. Block Diagram of Encoder

3.13.1. Octal to Binary Encoder

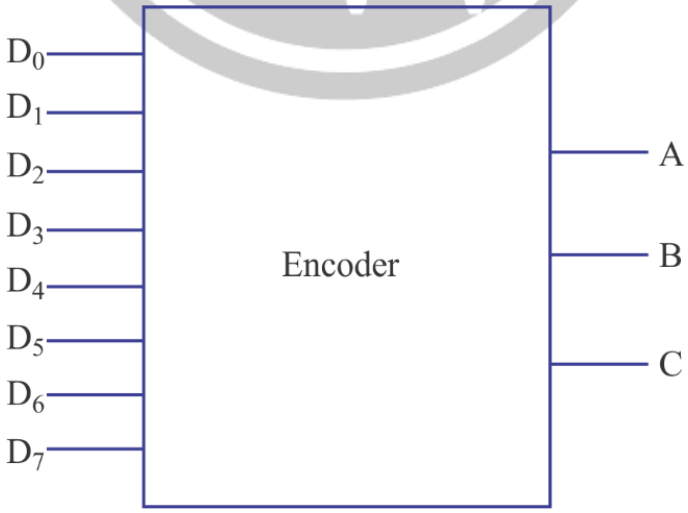


Fig. 3.30. Octal to Binary Encoder

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

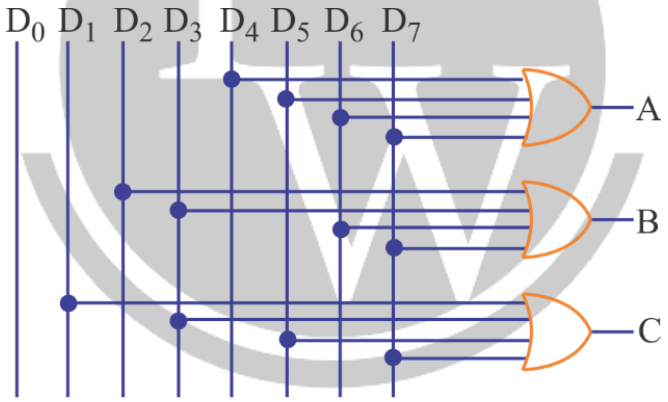


Fig. 3.31. Logic Diagram of Octal-to Binary Encoder

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.

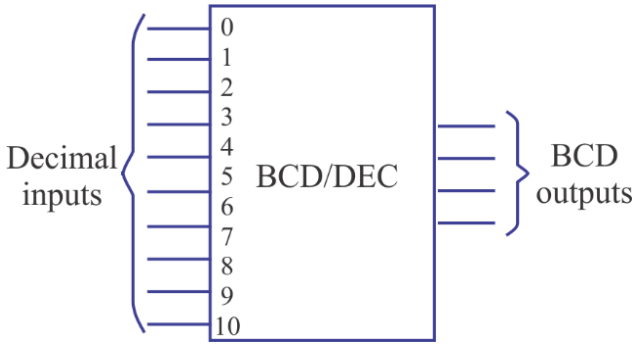


Fig. 3.32. Block Diagram of a Decimal-to-BCD Encoder

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				
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	1	0	0
0	0	0	0	0	1	0	0	0	0	0	1	0	1
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	1	1
0	0	0	0	0	0	0	0	1	0	1	0	0	0
0	0	0	0	0	0	0	0	0	1	1	0	0	1

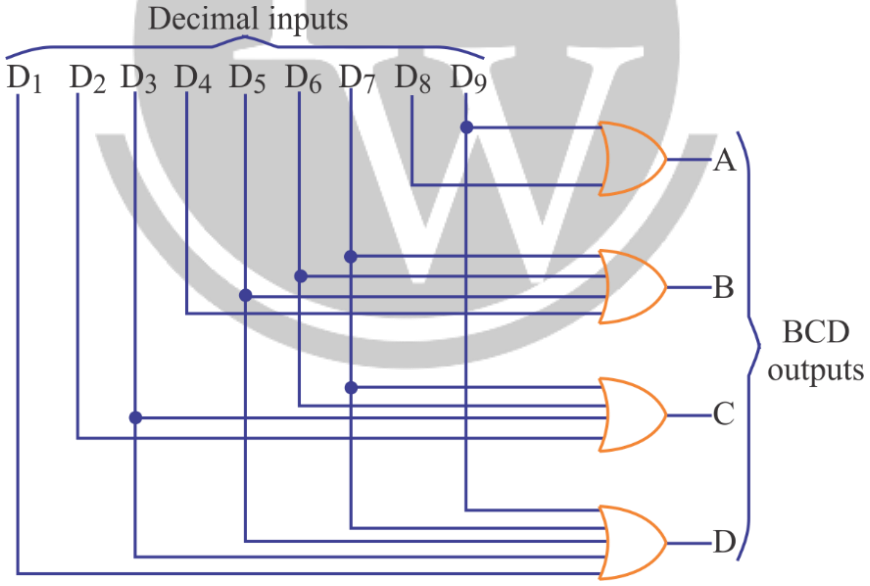


Fig. 3.33. Logic Diagram of Decimal-to-BCD encoder

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:

$$B_4 = AB + AC$$

$$B_1 = \overline{A}C + AB\overline{C}$$

$$B_2 = \overline{A}B + BC$$

$$B_3 = A\overline{B}\,\overline{C}$$

$$B_0 = D$$

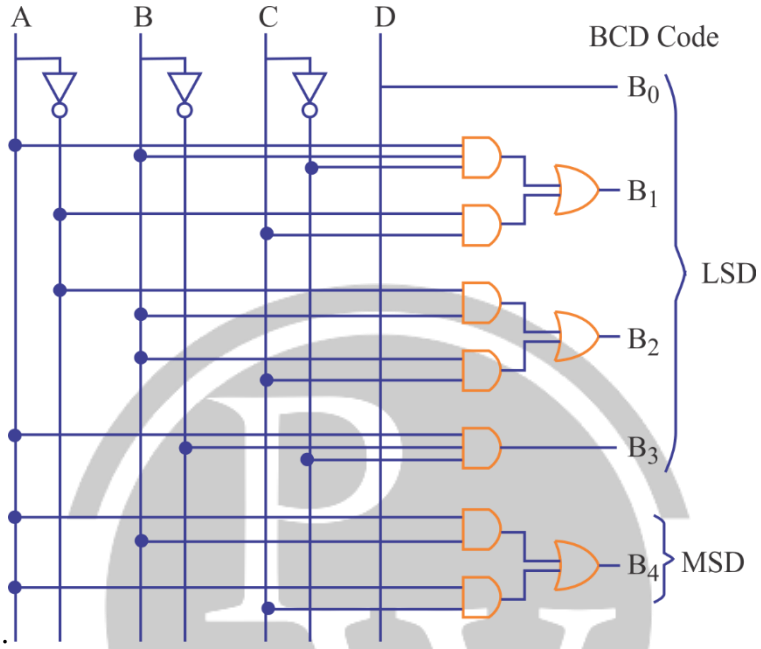


Fig. 3.34. Logic Diagram of a Binary-to-BCD Code Converter

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

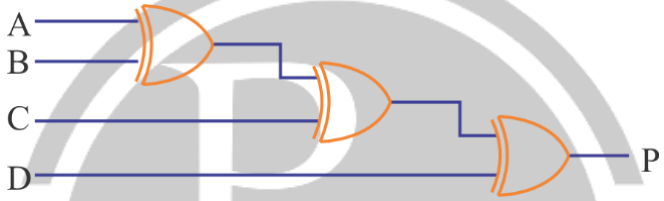


Fig. 3.35. Logic diagram of even parity generator

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 generator on 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.

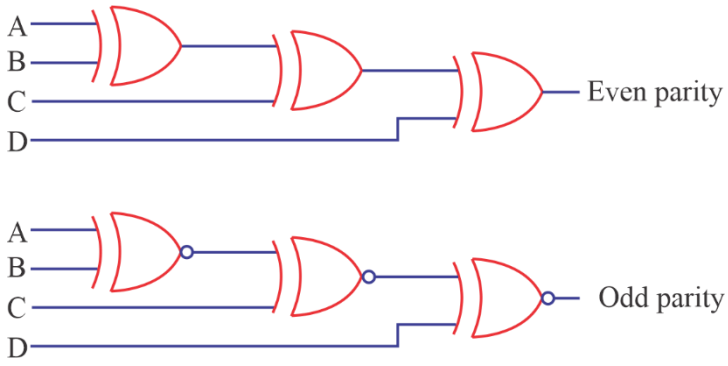


Fig. 3.36

∴ 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.

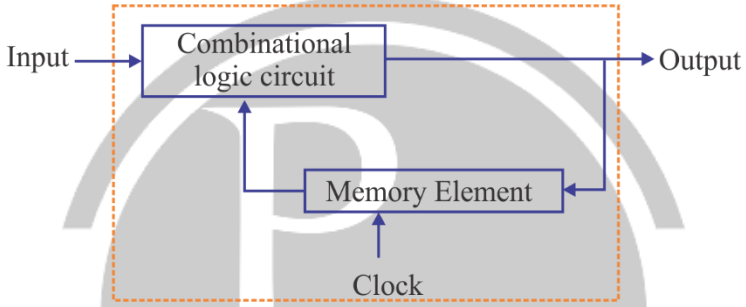


Fig. 4.1. General Block diagram of Sequential Logic Circuit

There are two types of sequential circuits:

4.1.1. Synchronous Circuits

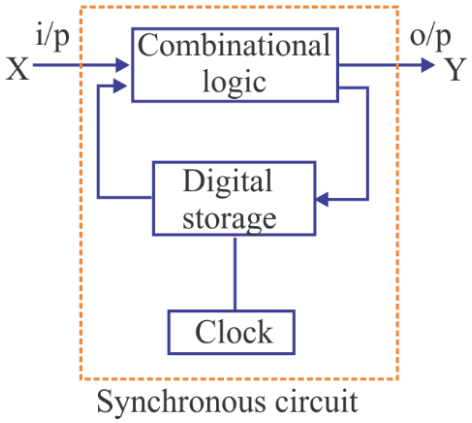
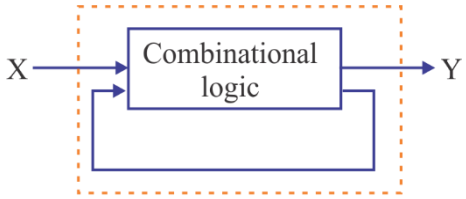
The sequential circuits which are controlled by a clock are called synchronous sequential circuits. These circuits get activated only when clock signal is present.

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.

5.	 Synchronous circuit	 Asynchronous circuit

4.3. Latches

Flip-flop is an electronic circuit or device which is used to store a data in binary form. Actually, flip-slop 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 $\overline{Q}$. The Q output is the normal output of the latch and $\overline{Q}$ 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 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:

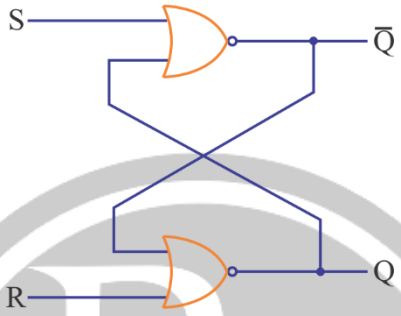


Fig. 4.1. Logic circuit of SR latch.

The state table for the SR latch is:

S	R	Q	Q^+	$\overline{Q^+}$
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:

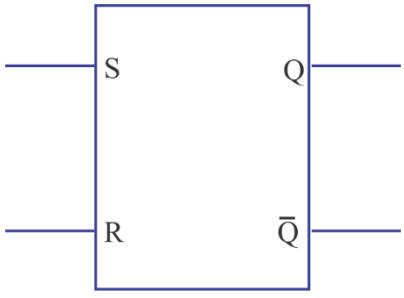


Fig. 4.2.

Obtaining the characteristic equations of the NOR gate based latch are; we get

$$Q^+ = \overline{R} \times S + \overline{R} \times Q = \overline{R} \times (S + Q) \text{ and } \overline{Q}^+ = \overline{S} \times R + \overline{S} \times \overline{Q} = \overline{S} \times (S + \overline{Q})$$

Note: It must be noted that the complementing Q^+ does not yield $\overline{Q}^+$.

Hence, the truth table for SR latch is

S	Q	Q^+	$\overline{Q^+}$	
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

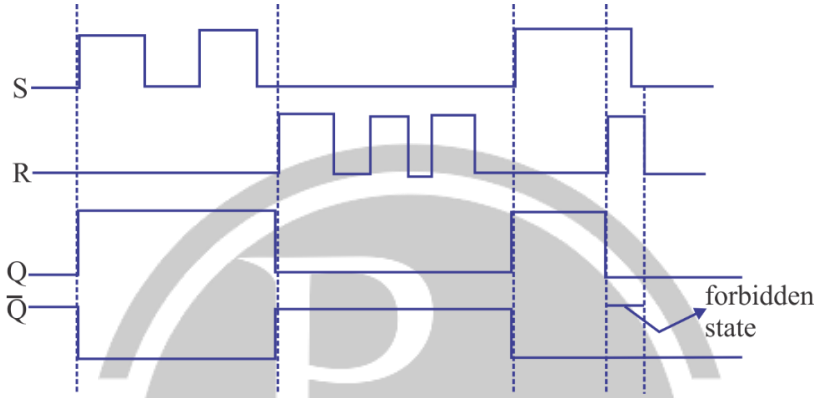


Fig. 4.3.

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

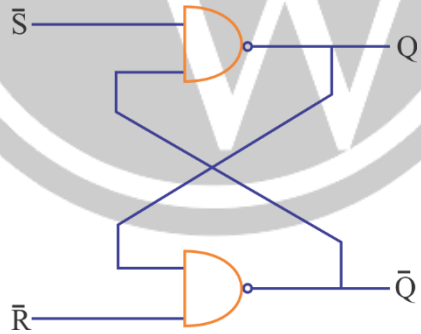


Fig. 4.4. Logic circuit for $\bar{S}\bar{R}$ Latch

The $\bar{S}\bar{R}$ latch is said to be set-dominant 1,

The symbol for $\bar{S}\bar{R}$ latch is shown below:

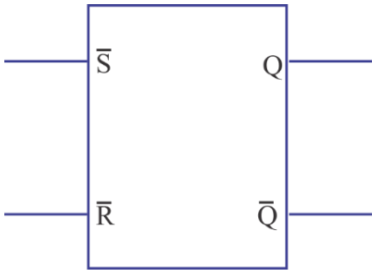


Fig. 4.5.

The truth table for $\overline{S}\overline{R}$ latch is given as:

$\overline{S}$	$\overline{R}$	Q^+	$\overline{Q^+}$	
1	1	Q	$\overline{Q}$	⇒ 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 $\overline{S}\overline{R}$ latch: The application of $\overline{S}\overline{R}$ latch is in switch bouncing i.e. contact bounces of a push-button switch during its opening or closing can be eliminated by using $\overline{S}\overline{R}$ 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

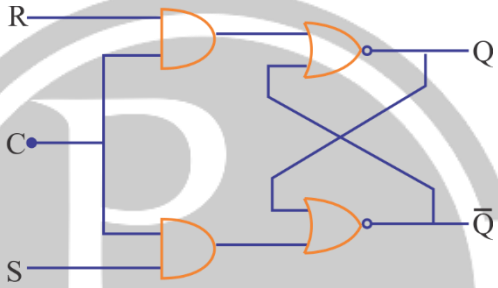


Fig. 4.6. Logic circuit of clocked SR latch.

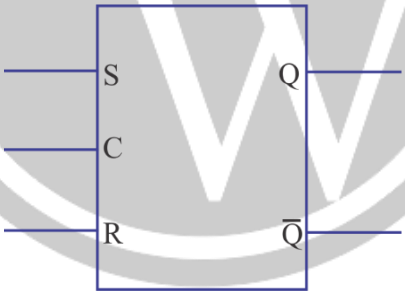


Fig. 4.7. Symbol for clocked SR latch

C	S	R	Q^+	$\overline{Q^+}$	
0	×	×	Q	$\overline{Q}$	} No change state
1	0	0	Q	$\overline{Q}$	
1	0	1	0	1	⇒ Reset
1	1	0	1	0	⇒ Set
1	1	1	0	0	⇒ Forbidden state

4.4.4. Gated $\overline{S}\overline{R}$ Latch or enable $\overline{S}\overline{R}$ Latch or clocked $\overline{S}\overline{R}$ Latch

Gated $\overline{S}\overline{R}$ latch is implemented using two NAND gates and an $\overline{S}\overline{R}$ latch.
The logic circuit diagram, symbol and truth is given as

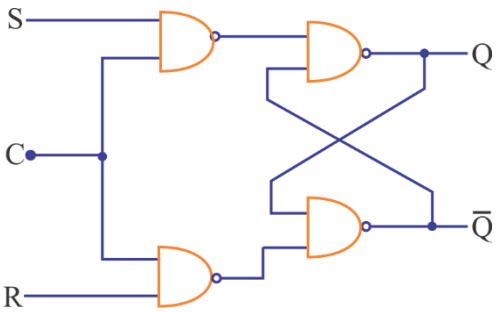


Fig. 4.8. Logic diagram of clocked $\overline{S}\overline{R}$ latch.

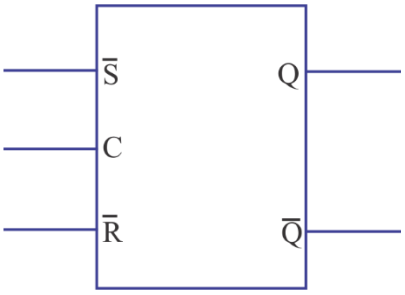


Fig. 4.9.

The truth table of the gated SR latch based on a $\overline{S}\overline{R}$ latch:

C	J	K	Q	$\overline{Q^+}$
O	**X**	**X**	**Q**	$\overline{Q}$
1	0	0	Q	$\overline{Q}$
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

$$Q^+ = Q_{n+1} = S + \overline{R}Q_n = S + \overline{R}Q$$

4.5. Flip-Flops

Flip-flops are synchronous bistable devices also known as bistable multivibrator. Its output change its state only at a verified paint (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 check 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.

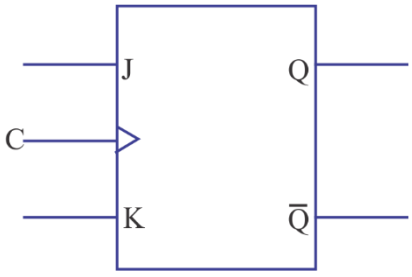


Fig. 4.10. Positive edge triggered flip-flop.

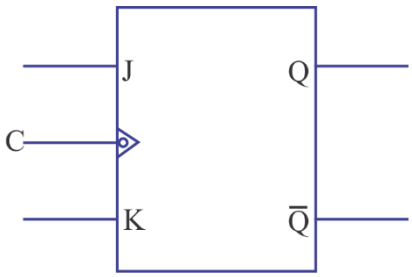


Fig. 4.11. Negative edge-triggered flip-flop.

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:

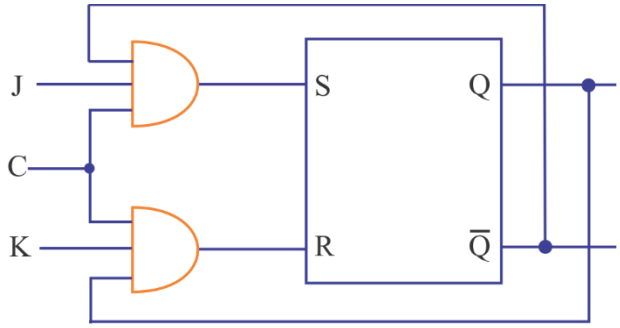


Fig. 4.12. Logic circuit diagram of clocked JK flip-flop.

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
1	1	1	1	0

Hence, the truth table becomes,

C	S	R	Q^+	$\overline{Q}^+$	
0	X	X	Q	$\overline{Q}$	} No change state
1	0	0	Q	$\overline{Q}$	
1	0	1	0	1	⇒ Reset
1	1	0	1	0	⇒ Set
1	1	1	0	0	⇒ 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 & $\overline{Q}$ forced the flip-flop is 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

$$Q_{n+1} = J\overline{Q}_n + \overline{K}Q_n \text{ or } Q^+ = J\overline{Q} + \overline{K}Q$$

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.

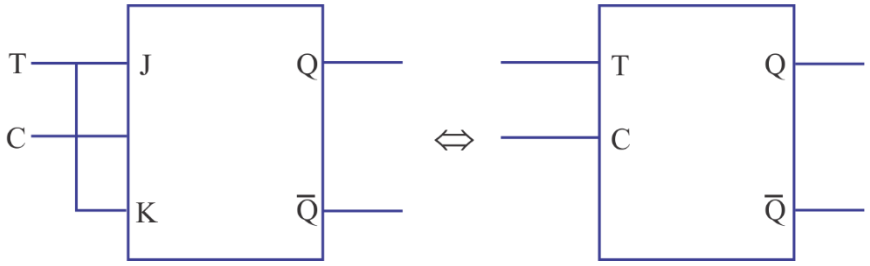


Fig. 4.13.

The state or characteristic table for T flip-flop is

C	T	Q	Q^+
O	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

$$Q^+ = T \times \overline{Q} \times C + (\overline{T} + \overline{C}) \times Q$$

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^+	$\overline{Q^+}$	
0	×	Q	$\overline{Q}$	⇒ No change state
1	0	Q	$\overline{Q}$	
1	1	$\overline{Q}$		⇒ 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

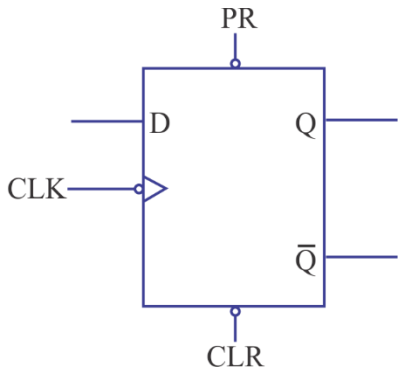


Fig. 4.15.

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.

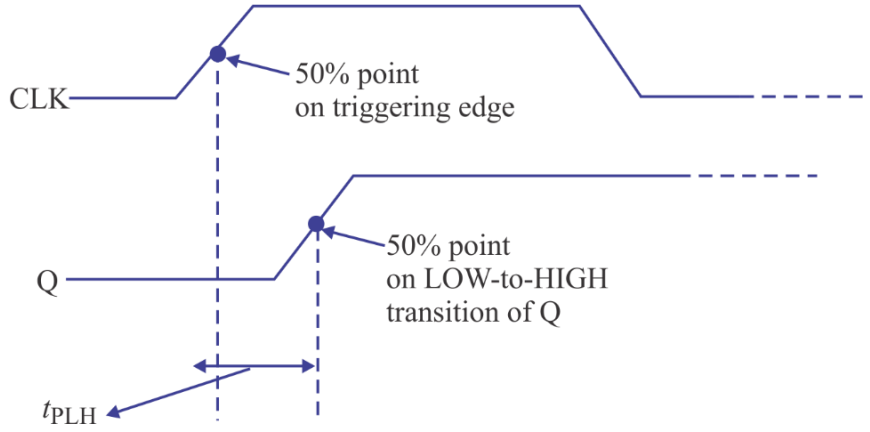


Fig. 4.16.

B. Propagation delay t_{PHL}, it is measured from the triggering edge of the clock pulse to HIGH-to-LOW transition of the output.

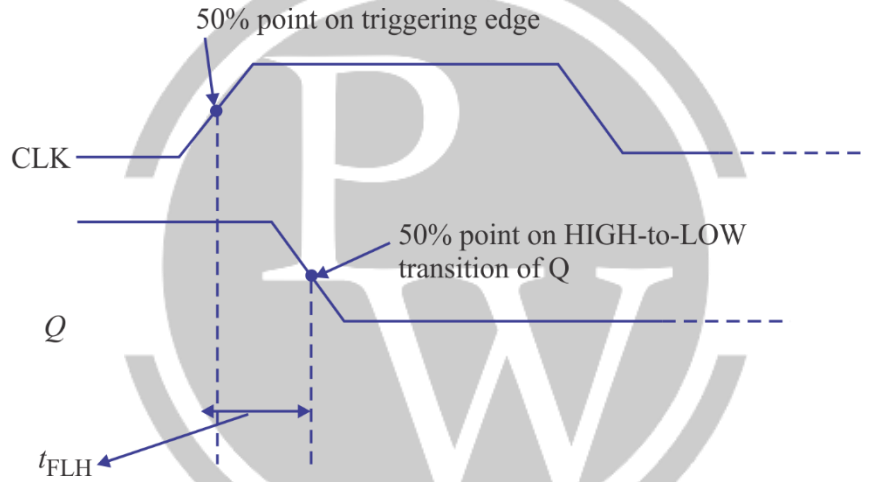


Fig. 4.17.

C. Propagation delay t_{PHL}, it is measured from the leading edge of the PRESET input to LOW-to-HIGH transition of the output.

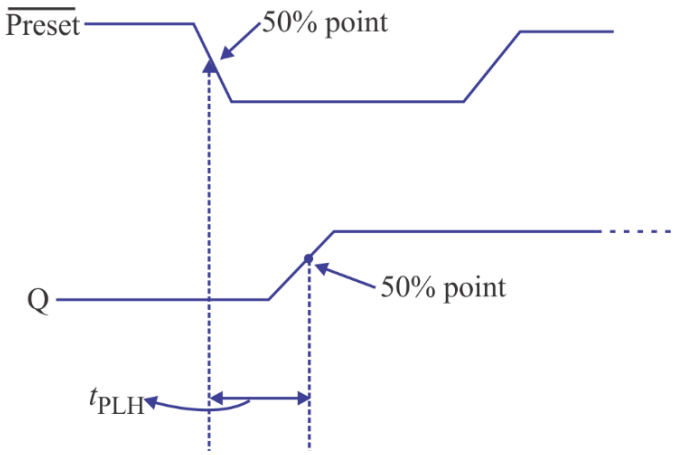


Fig. 4.18.

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.

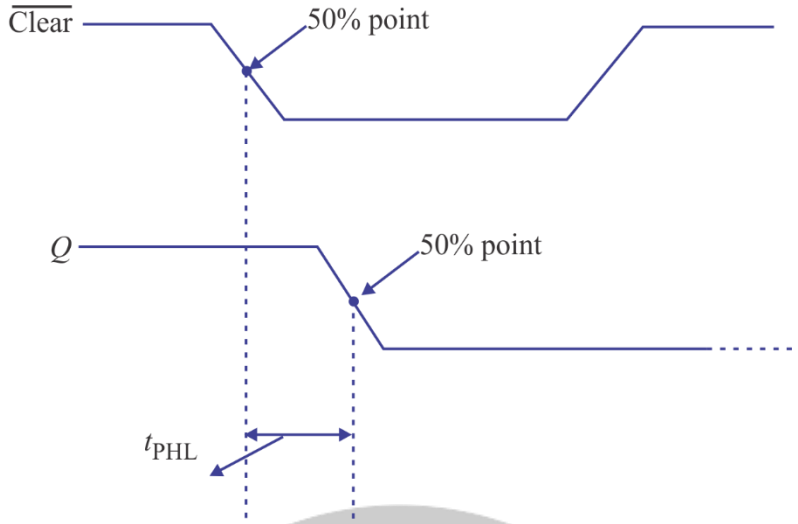


Fig. 4.19.

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.

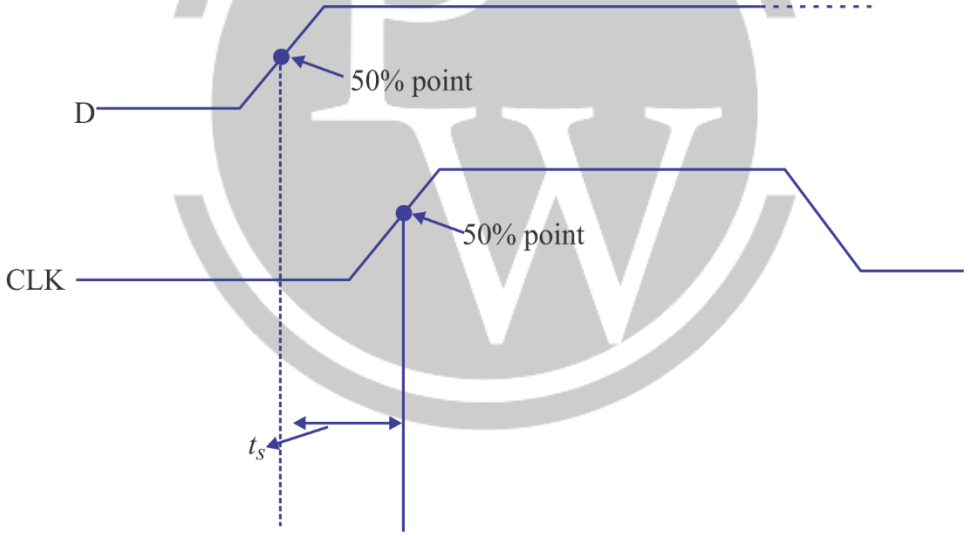


Fig. 4.20.

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.

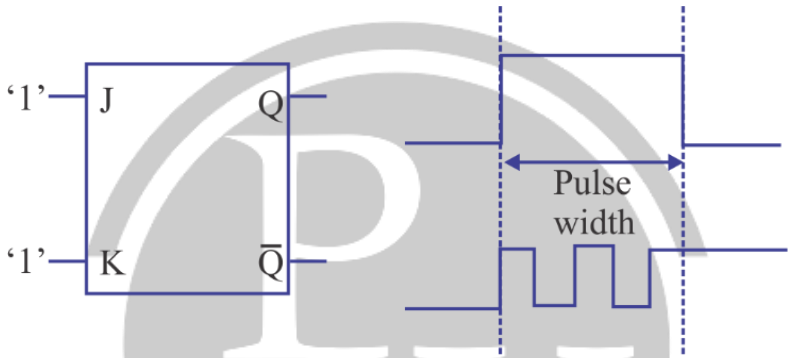


Fig. 4.21. Race Around Condition in JK Flip Flop

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:

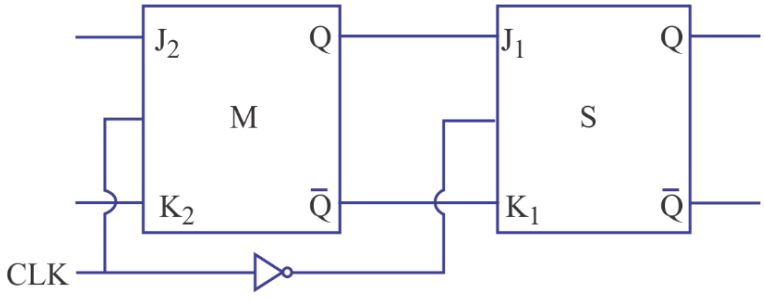


Fig. 4.22. Logic Diagram for Master Slave JK Flip Flop

(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 in 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.

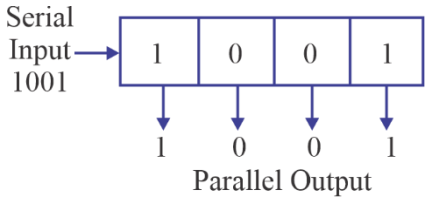


Fig. 4.24. SIPO Shift Register

- 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.

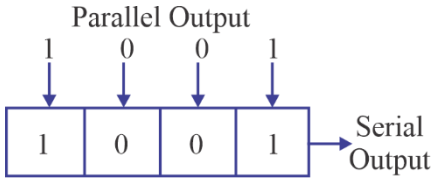


Fig. 4.25. PISO shift register

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:

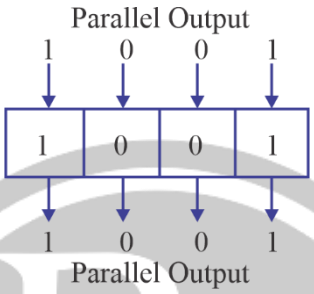


Fig. 4.26. PIPO Shift Register

- 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:

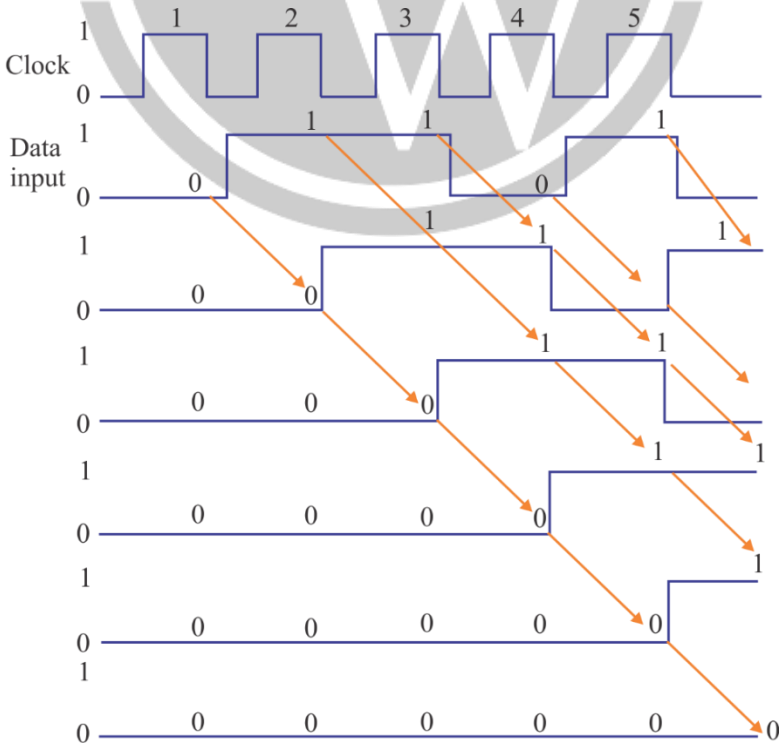


Fig. 4.27.

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 taking out in parallel from the outputs of the flip-flops. If a parallel

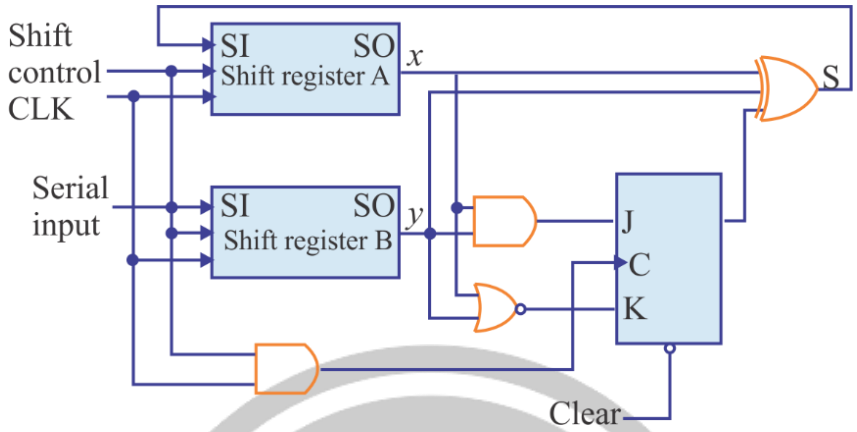


Fig. 4.28. Second form of serial adder

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:

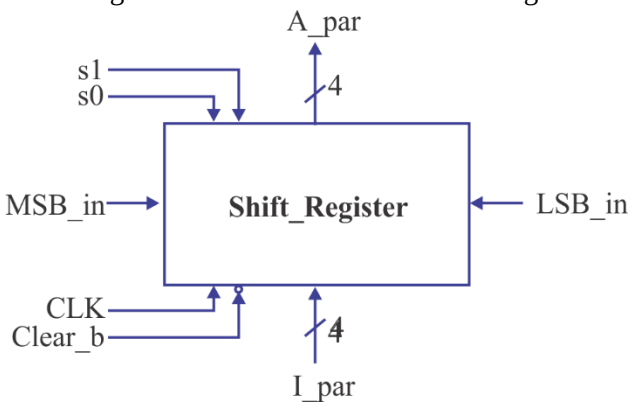


Fig. 4.29. 4-bit Universal Shift Register

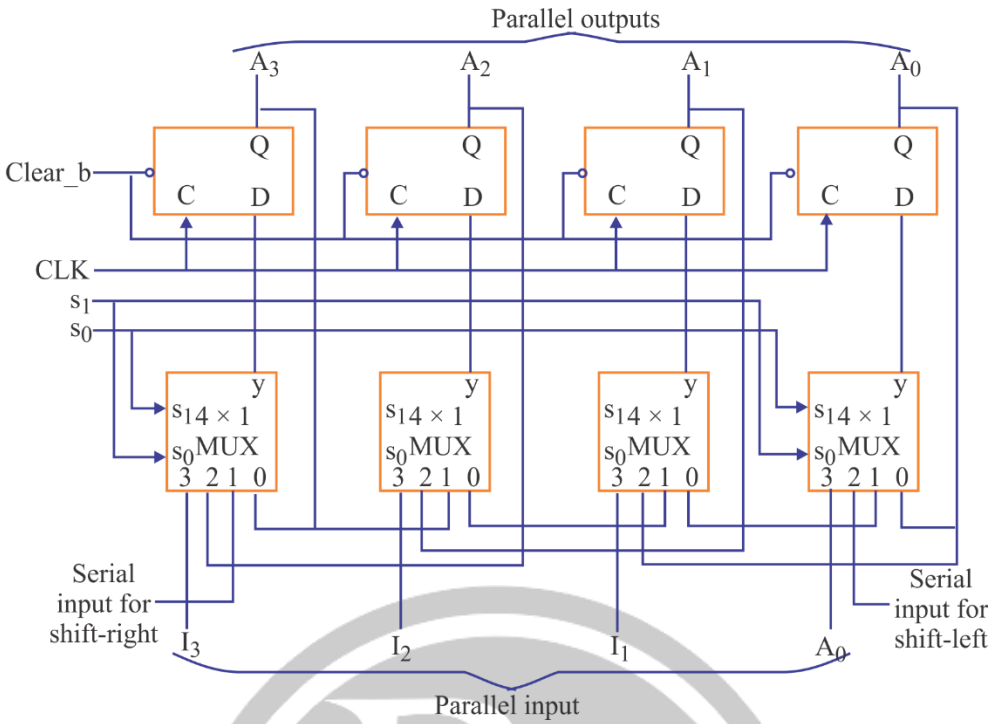


Fig. 4.30. Logic Diagram of 4-bit Universal shift register

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 the 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,

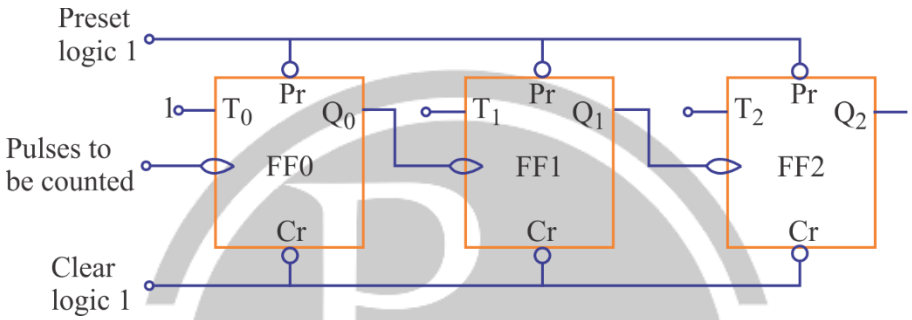


Fig. 4.31. A 3-bit Binary Counter

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:

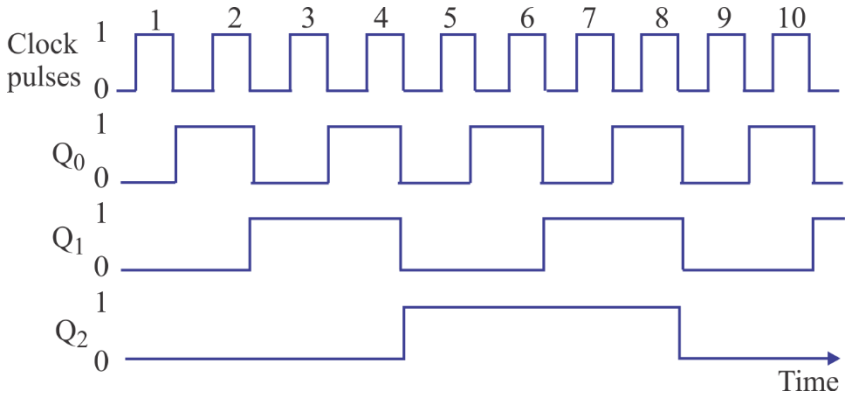


Fig. 4.32.

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 charge 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 operator 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 counters the values of pulses in descending order. Consider or Stable-8 counter ($2^3 = 8$), which uses three flip-flops.

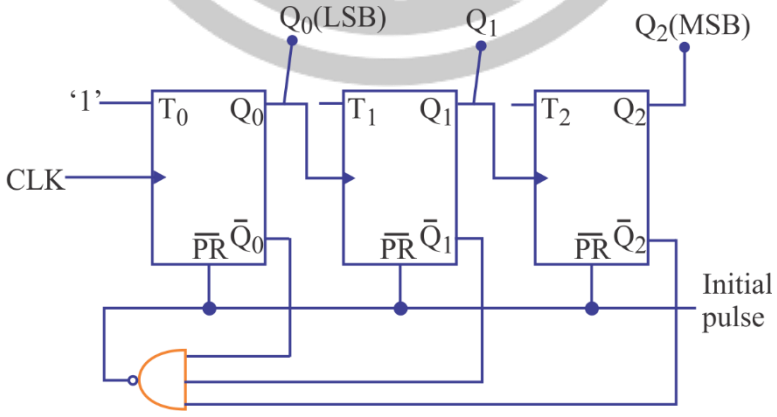


Fig. 4.33.

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).

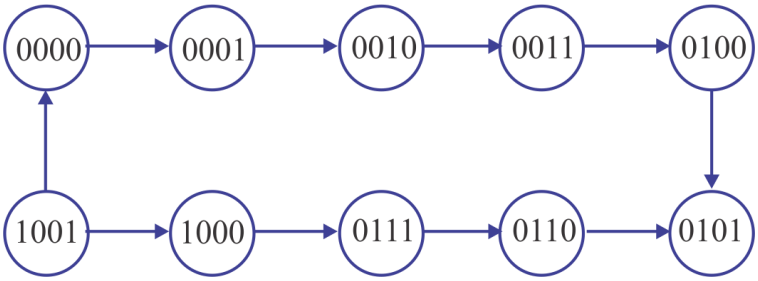


Fig. 4.34. State diagram of a decimal BCD counter.

The logic diagram of a BCD ripple counter using JK flip-flops is shown in figure below. The four outputs arc 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 Q_1 is applied to the C inputs of both Q_2 and Q_4 and the output of Q_2 is applied to the C and output of Q_2 and Q_3 applied to J through a two input AND gate.

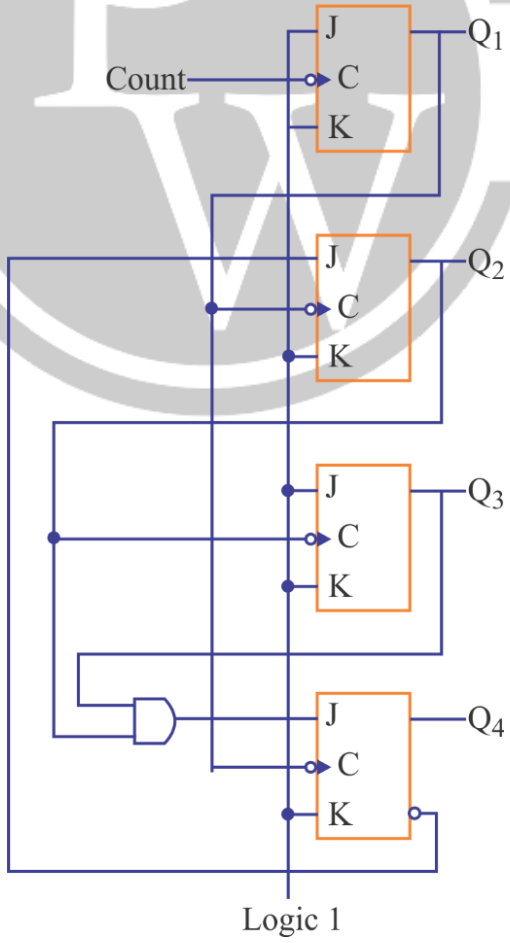


Fig. 4.35. BCD ripple counter

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-FLOPS 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-FLOPS 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-FLOPS for their memory elements, therefore, counter design using D FLIP-FLOPS 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-FLOPS.

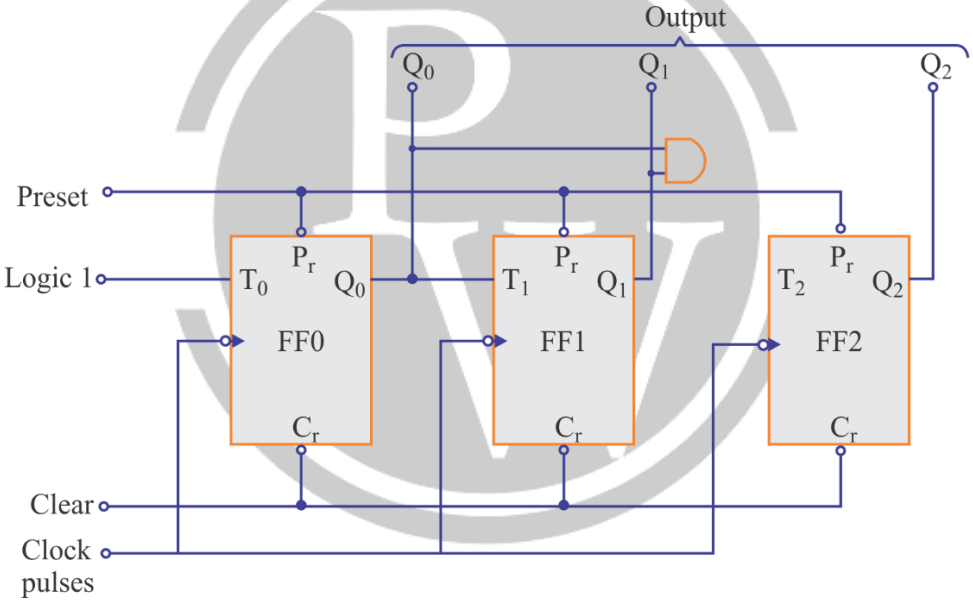


Fig. 4.36. A 3-bit Synchronous Counter

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-FLOPS.
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-FLOPS and other gates corresponding to the minimized expressions.

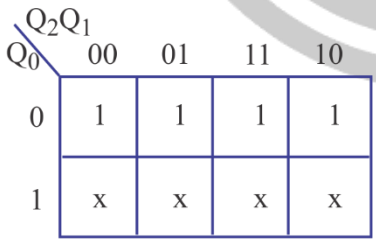
Example: Design a 3-bit synchronous counter using JK Flip-Flops.

Solution: The number of FLIP-FLOPs required is 3. Let the FLIP-FLOPs be FF0, FF1, FF2 and their inputs and outputs are given below:

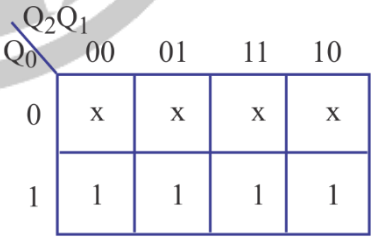
FLIP-FLOP	Inputs	Outputs
FF0	J_0, K_0	Q_0
FF1	J_1, K_1	Q_1
FF2	J_2, K_2	Q_2

The count sequence and the required inputs of FLIP-FLOPs is shown below.

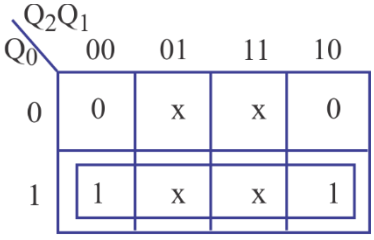
			FLIP-FLOP INPUTS					
Counter state			FF0		FF1		FF2	
Q_2	Q_1	Q_0	J_0	K_0	J_1	K_1	J_2	K_2
0	0	0	1	X	0	X	0	X
0	0	1	X	1	1	X	0	X
0	1	0	1	X	X	0	0	X
0	1	1	X	1	x	1	1	X
1	0	0	1	X	0	X	X	0
1	0	1	X	1	1	X	X	0
1	1	0	1	X	X	0	X	0
1	1	1	x	1	X	1	x	1
0	0	0						



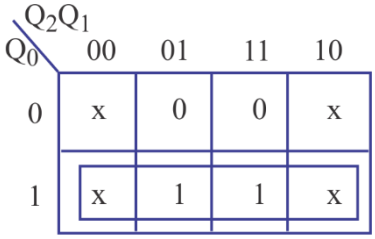
$J_0 = 1$
(a)



$K_0 = 1$
(b)



$J_1 = Q_0$
(c)



$K_1 = Q_0$
(d)

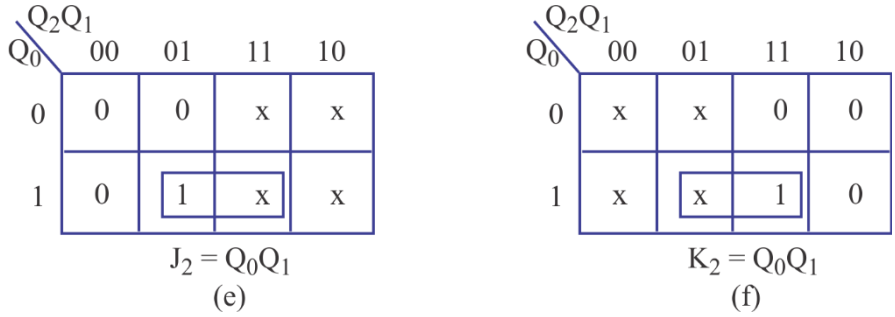


Fig. 4.37. K-Maps of 3-bit Synchronous Counter

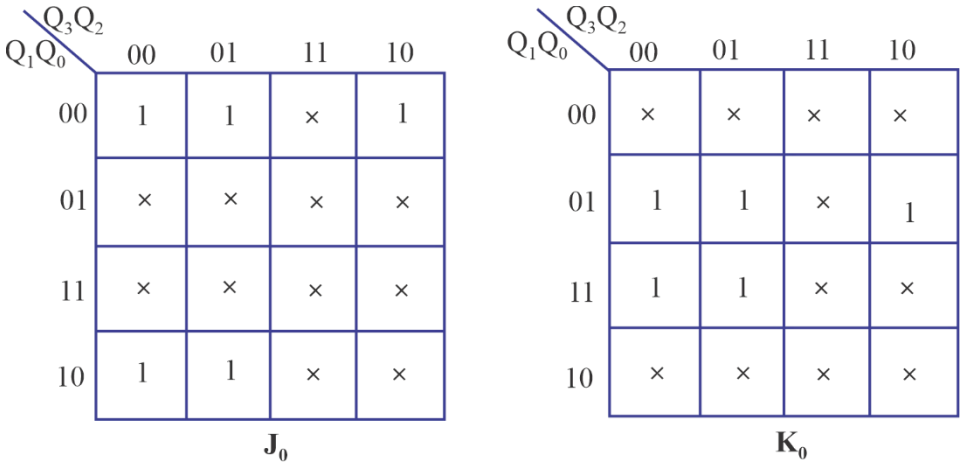
Example:

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



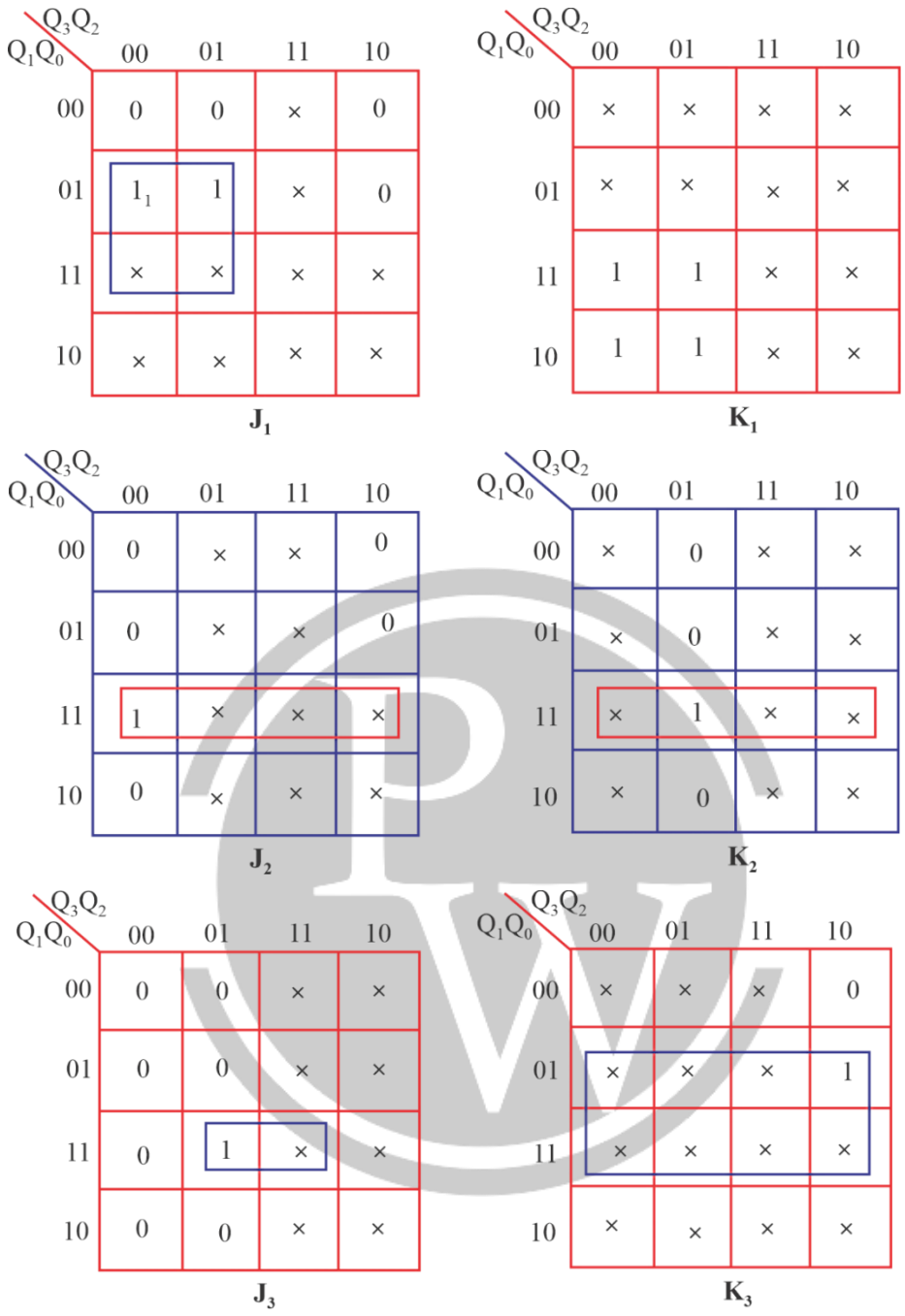


Fig. 4.38. K-Maps for 8-bit Synchronous counter

The K – maps for D_0, D_1 and D_2 are given as,

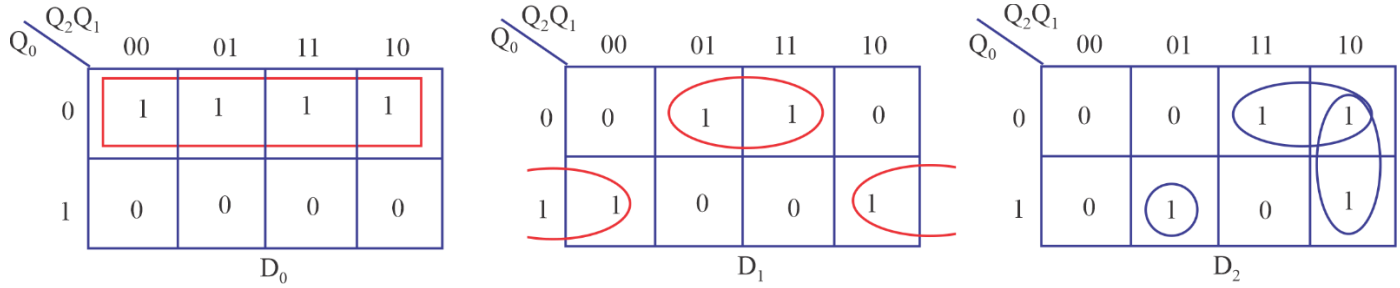


Fig. 4.39.

The minimised expressions for D0, D1 and D2 are:

$$D_0 = \bar{Q}_0$$

$$D_1 = Q_1\bar{Q}_0 + \bar{Q}_1 Q_0$$

$$D_2 = Q_2\bar{Q}_0 + Q_2\bar{Q}_1 + Q_2Q_1Q_0 = Q_2(\bar{Q}_0 + \bar{Q}_1) + \bar{Q}_2Q_1Q_0 = Q_2(\overline{Q_0.Q_1}) + \bar{Q}_2(Q_1Q_0)$$

$$= Q_2 \oplus Q_1.Q_0$$

The complete circuit of the synchronous counter using positive edge triggered D FLIP-FLOPs is shown in figure below as

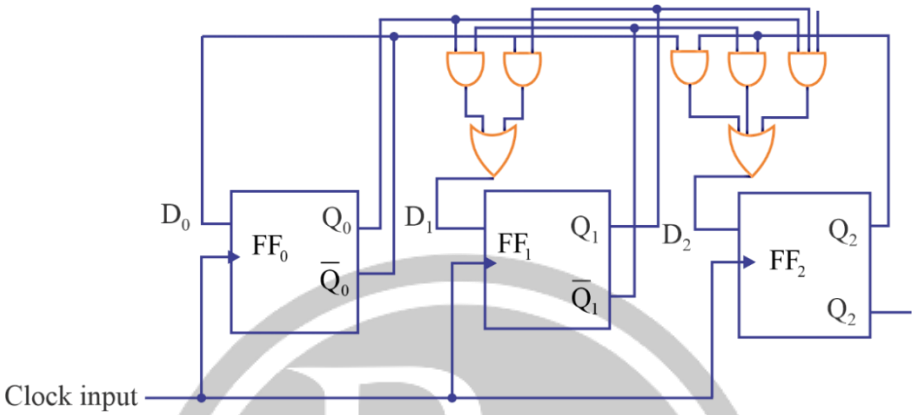


Fig. 4.40. 8-bit Synchronous Counter Circuit

4.12.2 Synchronous Sequential Circuit Models

A general block diagram of clocked sequential circuit is also known as finite stable machine (FSM). Depending upon the external outputs, there are two types of models of sequential circuits.

Mealy Model

In mealy model, the next of the function depends on present state as well as present inputs.

Moore Model

In more model, the next state depends on the present state. The block diagram of a Moore model is given as

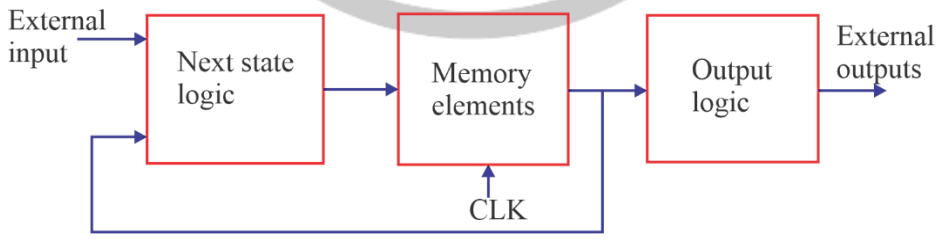


Fig. 4.41.

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 are 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 are represents 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

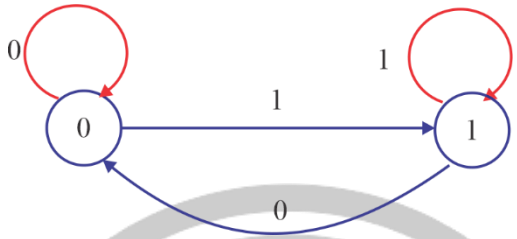


Fig. 4.42.

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

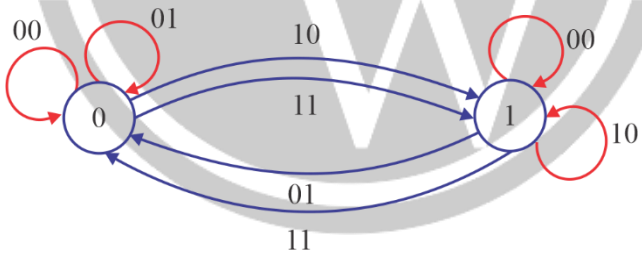


Fig. 4.43.

5 NUMBER SYSTEM

5.1. NUMBER SYSTEM

5.1.1. Base (Radix)

Total number of digit used in the system

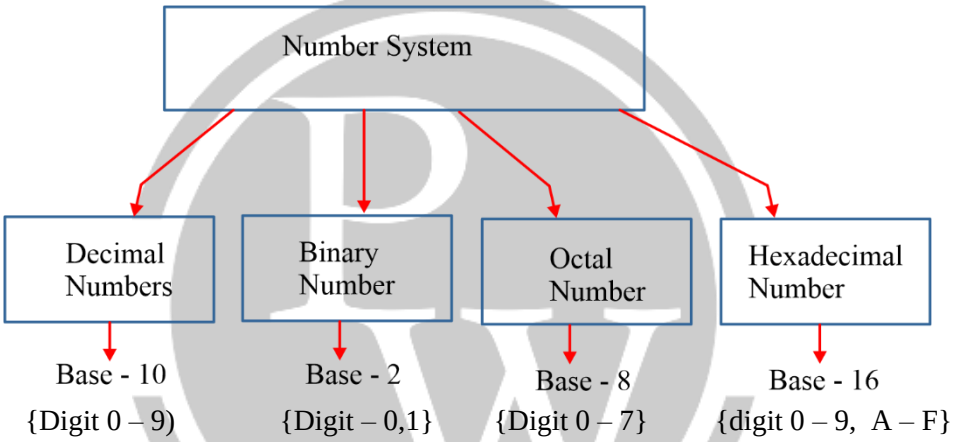


Fig. 5.1.

5.1.2. Decimal Number System

…	10^4	10^3	10^2	10^1	10^0	10^{-1}	10^{-2}	10^{-3}…
…	a_4	a_3	a_2	a_1	a_0	a_{-1}	a_{-2}	a_{-3} …

a_i → Coefficient of decimal number system

10^i → Weight of decimal number system

Example: - $(501.23)_{10}$

10^2	10^1	10^0	10^{-1}	10^{-2}
5	0	1	2	3

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} …

… a_4 a_3 a_2 a_1 a_0 a_{-1} a_{-2} a_{-3} …

2^i → Weight of Binary number system

a_i → 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}…

… a_3 a_2 a_1 a_0 a_{-1} a_{-2} a_{-3}…

8^i → Weight of Octal number system

a_i → 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}…

… a_3 a_2 a_1 a_0 a_{-1} a_{-2} a_{-3}…

16^i → Weight of Hexadecimal number system

a_i → 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:

$$(a_3\ a_2\ a_1\ a_0\ .\ a_{-1}\ a_{-2})=(\quad)_{10}$$

$$(a_3\times r^3+a_2\times r^2+a_1\times r^1+a_0\times r^0+a_{-1}\times r^{-1}+a_{-2}\times r^{-2})_{10}$$

Case (1) : Binary to Decimal conversion

Ex. $(1011.11)_2 = (\quad)_{10}$

$\Rightarrow \quad \left[(1\times 2^3)+(0\times 2^2)+(1\times 2^1)+(1\times 2^0)+(1\times 2^{-1})+(1\times 2^{-2})\right]$

$\Rightarrow \quad [8+0+2+1+0.5+0.25]_{10}$

$\Rightarrow \quad (11.75)_{10}$

Case (2) : Octal to Decimal conversion

Ex. $(721.4)_8 = (\quad)_{10}$

$\Rightarrow \quad \left[(7\times 8^2)+(2\times 8^1)+(1\times 8^0)+(4\times 8^{-1})\right]_{10}$

$\Rightarrow \quad [448+16+1+0.5]_{10}$

$\Rightarrow \quad (465.5)_{10}$

Case (3) : Hexadecimal to Decimal conversion

Ex. $(A2B.C)_{16} = (\quad)_{16}$

$\Rightarrow \quad \left[(A\times 16^2)+(2\times 16^1)+(B\times 16^0)+(C\times 16^{-1})\right]_{10}$

$\Rightarrow \quad \left[(10\times 256)+(2\times 16)+(11\times 1)+(12\times 16^{-1})\right]_{10}$

$\Rightarrow \quad [2560+32+11+0.75]_{10}$

$\Rightarrow \quad (2603.75)_{10}$

Case (4) : Base 5 to Decimal conversion

Ex. $(432.22)_5 = (\quad)_{10}$

$\Rightarrow \quad \left[(4\times 5^2)+(3\times 5^1)+(2\times 5^0)+(2\times 5^{-1})+(2\times 5^{-2})\right]_{10}$

$\Rightarrow \quad [100+15+2+0.4+0.08]_{10}$

$\Rightarrow \quad (117.48)_{10}$

(B) Decimal to any other Base conversion

$(a_3\ a_2\ a_1\ a_0\ .\ a_{-1}\ a_{-2}\ a_{-3})_{10} = (\quad)_r$

Before Decimal — After Decimal

r	a_3	a_2	a_1	a_0	Remainder
					b_0
					b_1
					b_2
					b_3

$$0 \cdot a_{-1}\, a_{-2}\, a_{-3} \times r = x_0 \cdot x_{-1}\, x_{-2}$$
$$0 \cdot x_{-1}\, x_{-2} \times r = x_1 \cdot x_{-3}\, x_{-4}$$
$$0 \cdot x_{-3}\, x_{-4} \times r = x_2 \cdot x_{-5}\, x_{-6}$$

$$\left(a_3 a_2 a_1 a_0 \cdot a_{-1} a_{-2} a_{-3}\right)_{10} = \left(b_3 b_2 b_1 b_0 \cdot x_0 x_1 x_2\right)_r$$

Case (1) : Decimal to Binary Base conversion.

Ex.

$(19.75)_{10} = (\quad)_2$

Before Decimal R After Decimal

2	19	1
2	9	1
2	4	0
2	2	0
	1	1

$0.75 \times 2 = 1.5$ 1
$0.5 \times 2 = 1.0$ 1

$(19.75)_{10} = (10011.11)_2$

Case (2) : Decimal to Octal Base conversion.

Ex.

$(210.23)_{10} = (\quad)_8$

Before Decimal Remainder After Decimal

8	210	2
8	26	2
	3	3

$0.23 \times 8 = 1.84$ 1
$0.84 \times 8 = 6.72$ 6
$0.72 \times 8 = 5.76$ 5

$(210.23)_{10} = (322.165)_8$

Case (3): Decimal to Hexadecimal Base conversion.

Ex.

$(1228.55)_{10} = (\quad)_{16}$

Before Decimal Remainder After Decimal

16	1228	12 (C)
16	76	12 (C)
	4	4

$0.55 \times 16 = 8.8$ 8
$0.8 \times 16 = 12.8$ 12(C)

$(1228.56)_{10} = (4CC.8C)_{16}$

5.2. Some Special Case

Case (1): Binary to Octal base conversion

Example: $(10110111)_2 = (\quad)_8$

Octal → 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: $(1011011)_2 = (\quad)_{16}$

Hexadecimal → means base 16

$16 = 2^4$

Every four digits of binary represent one digit of Hexadecimal.

0101 1011

5 11(B)

Hence $(1011011)_2 = (5B)_{16}$

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.

Multiplicand:	1011
Multiplier:	X101
	1011
	0000
	1011
+	
Product:	110111

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.

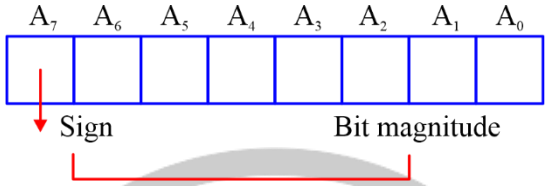


Fig. 5.2.

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 from, 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
+ 15 0001111	+3 (i) 0000010	– 3 1111100	– 15 (1) 1101111
	Carry + 1		Carry + 1
	+ 3 0000011		1110000
	carry		carry

The advantage of signed 2's complement representation over the signed 1's compliment from (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

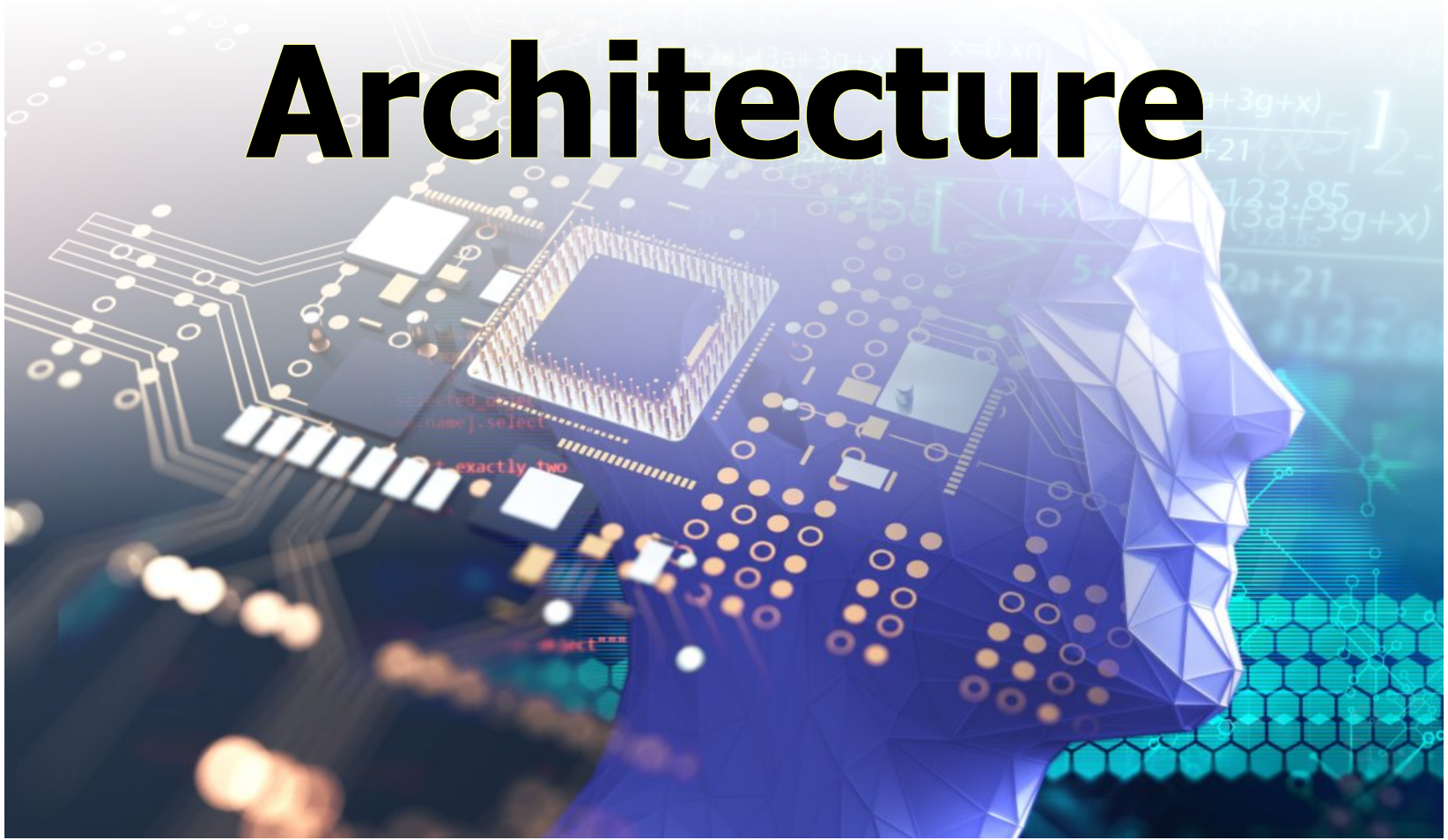
ME | CE | EE | EC | CS & IT

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

Weekday & Weekend Batches Available

JOIN NOW!

Computer Organization & Architecture



INDEX

1 MACHINE INSTRUCTIONS

1.1 Introduction

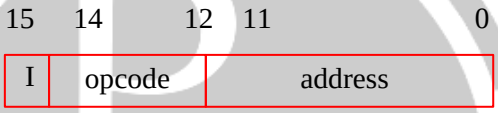
Opcode – Specifies the operation code. The number of bits in opcode depends on total operations.
Address – Specifies the operand address.

- The basic computer has three instruction code formats.

(i) **Memory – Reference instruction**

I = O ⇒ Direct addressing

I = 1 ⇒ Indirect addressing



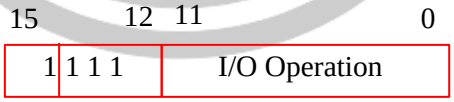
Instruction format

(opcode = 000 to 110)

(ii) **Register – Reference instruction**

(opcode = 111, I = 1)

(iii) **Input – output Instruction**



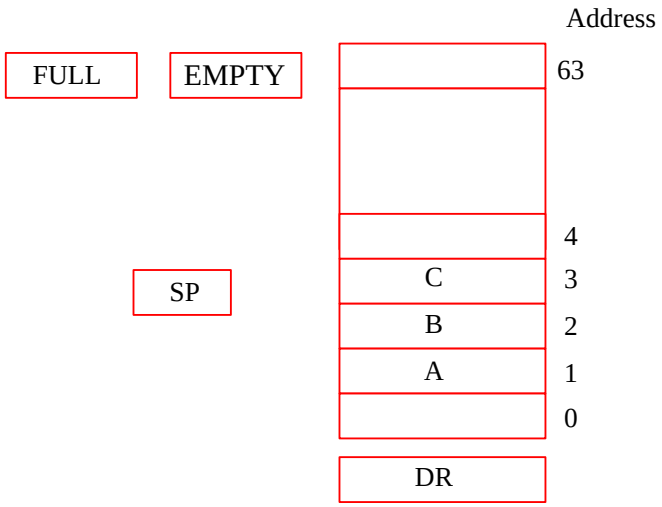
(op code = 111, I = 1)

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 ← SP + 1 [increment stack pointer]
M{SP} ← DR [write item on top of stack]
If (SP = 0) then FULL ←1 [check if stack is full]
EMTY ← 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 ← M{SP} [Read item from top of stack]
SP ← Sp – 1 [Decrement stack pointer]
If (SP = 0) then EMTY – 1

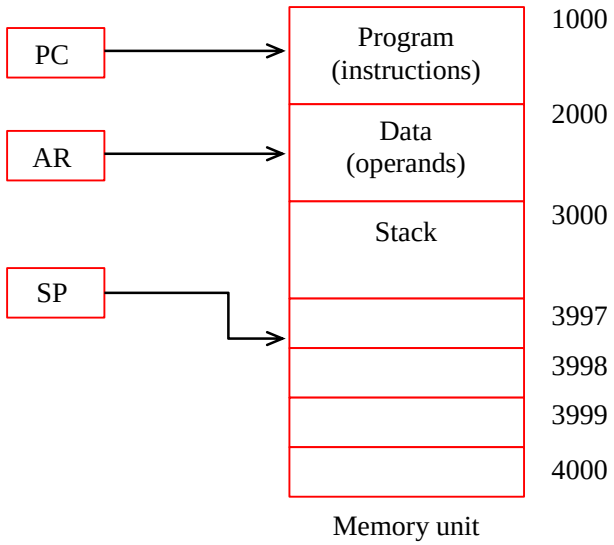
1.1.2. Memory Stack

- 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 ← SP – 1
M[SP] ← DR

- A new item is deleted with a POP operation as follows

DR ← M[SP]
SP ← 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 ← (A + B) * (C + D)
  With values A = 2, B = 3, C = 4, d = 2 is
  X ← (2 + 3) * (4 + 2) ← Infix expression
  x ← 23 + 42 + * ← 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

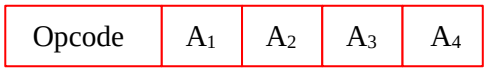
- Data transfer instructions – (LOAD, STOR, MOV, EXCT, IN, OUT, PUSH, POP)
- Arithmetic instructions – (INC, DEC, ADD, SUB, MUL, DIV)
- Logical Instructions & Shift instructions. (CLR, COM, AND, XOR, OR, SHR, SHL, SHRA, SHLA, RO, ROL….)
- Branching instructions. (ABR, JMP, SKP, CALL, RETURN)

1.4 Based on Number of References

Based on number of references, the instructions can be classified as

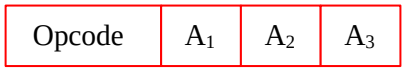
- 4 – address instructions
- 3- address instructions
- 2 – address instructions
- 1 – address instructions
- 0 – address instructions
- RISC Instructions

(i) 4 – Address Instructions:



- A_1, A_2 refers operands
- A_3 refers destination
- A_4 next instruction address.
- Since A_4 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



- 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 R_1, A, B $R_1 \leftarrow M[A] + M[B]$

Add R_2, C, D $R_2 \leftarrow M[C] + M[D]$

MUL X, R_1, R_2 $M[X] \leftarrow R_1 * R_2$

For these instructions program counter must be required.

(iii) 2 – address Instructions:

- A_1 – first operand and destination
- A_2 – second operand
- Uses MOV instruction for data transfer

Example:

$X \leftarrow (A + B) * (C + D)$

MOV R_1, A

ADD R_1, B

MOV R_2, C

ADD R_2, D

MUL R_1, R_2

MOV X R_1

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    R1, A
LOAD    R2, B
LOAD    R3, C
LOAD    R4, D
ADD     R1, R1, R2
ADD     R3, R3, R4
MUL     R1, R1, R3
STOR    X, R1
```

❐❐❐

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 ← 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

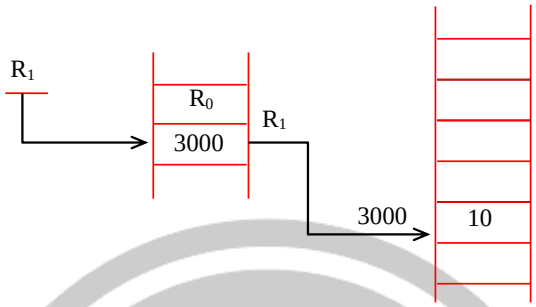
Register Indirect Mode:

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)

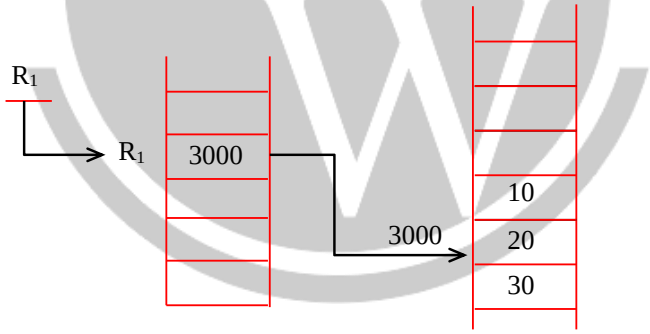


Autoincrement or Autodecrement Mode:

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 (R_1)

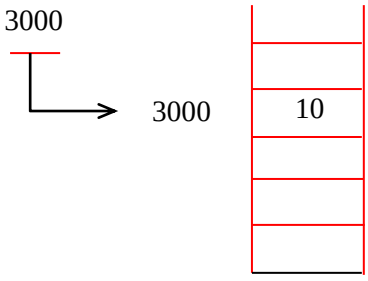


Direct Addressing Mode:

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".

Example. ADD 3000



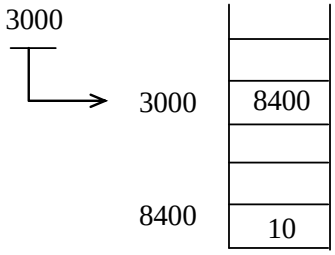
i.e. AC ← AC + M [3000]

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 = Address Part (off set) + 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 = Address Part (base address of data array) + 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 = Address Part (displacement/offset) + Base register value (Base address)

❐❐❐

3 ALU DATA PATH

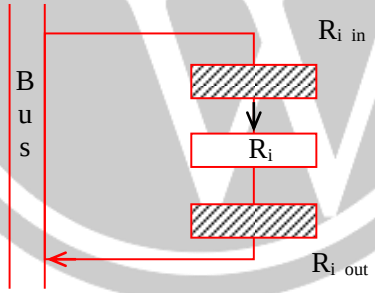
3.1 Introduction

- 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 switched controlled by the signals R_i in and $R_i$out



- When Ri in is set to 1, the data on the bus are loaded into Ri.
- When Ri out is set to 1, the contents of register Ri 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 R1 by setting R1 out to 1. This places the contents of R1 on the processor bus.
- Enable the input to register R_4 by setting $R_{4\,in}$ 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

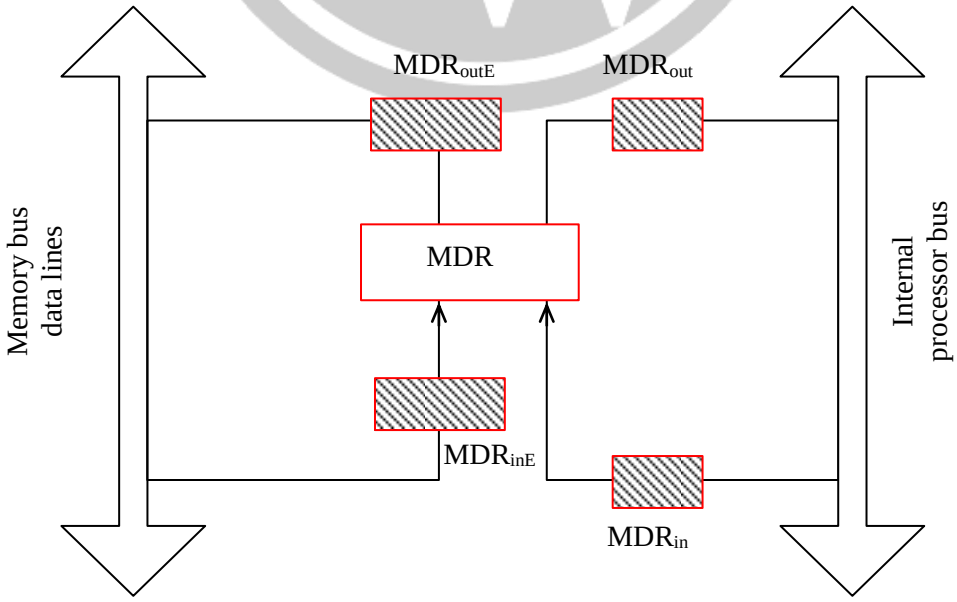
(5) $R_2 \leftarrow$ [MDR]

- 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 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) $R_{1\ out}$, MAR $_{in}$
  (2) R_{2out}, MDR_{in}, write
  (3) MDR_{outE}, WMFC

3.2.5 Branch Instructions

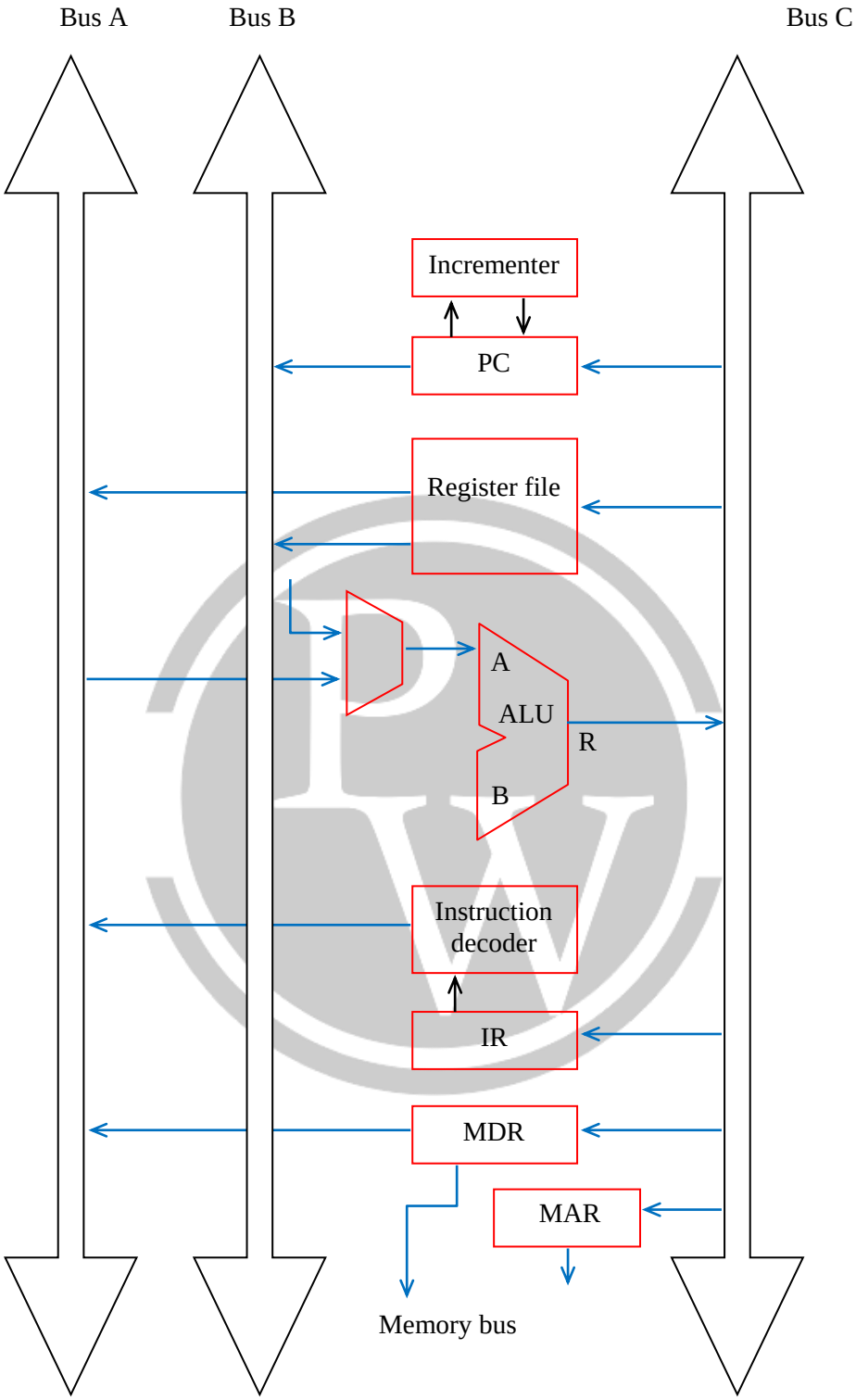
- A branch instruction replaces the content of PC with the branch target address. The address is obtained by adding an offset X, which is given in the branch instruction, to the updated value of PC.
- The control sequence for branching (unconditional is
  (1) PC_{out}, PC_{in}, Y_{in}, WMFC
  (2) Z_{out}, PC_{in}, Y_{in}, WMFC
  (3) MDR_{out}, IR_{in}
  (4) Offset – of – IR_{out}, Add, Z_{in}

3.2.6 Execution of complete Instruction

- Executing an instruction requires (Add (R_3), 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 ADD (R_3), R_1 in a single-bus organization is
  (1) PC_{out}, MAR_{in}, Read, Select, Add, Z_{in}
  (2) Z_{out}, PC_{in}, Y_{in}, WMFC
  (3) MDR_{out}, IR_{in}
  4) R_{3out}, MAR_{in}, Read
  (5) $R_{1\ out}$, Yin, WMFC
  (6) MDR_{out}, select Y, Add, Z_{in}
  (7) Z_{out}, $R_{1\ in}$, 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

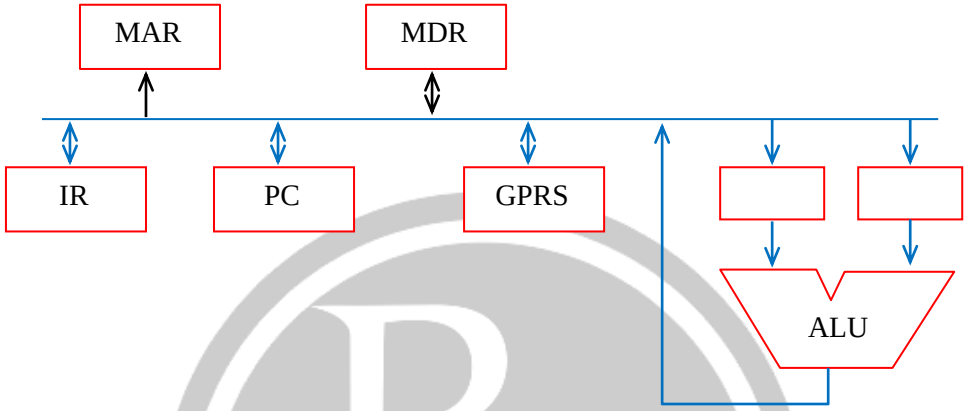


- All general – purpose registers are combined into a single block called register file. It has two – outputs, allowing the contents of two different registers to be accessed simultaneously and have their contents placed on buses A and B. The data on bus 'C' to be loaded into third register during same 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 R_4, R_5, R_6
  (1) P_{Cout}, R = B, MAR_{in}, Read, Inc PC
  (2) WMFC
  (3) MDT_{out} B, R = B, IRin
  (4) $R_{4\ outA}$, $R_{5\ outB}$, select A, Add, $R_{6\ in}$, 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

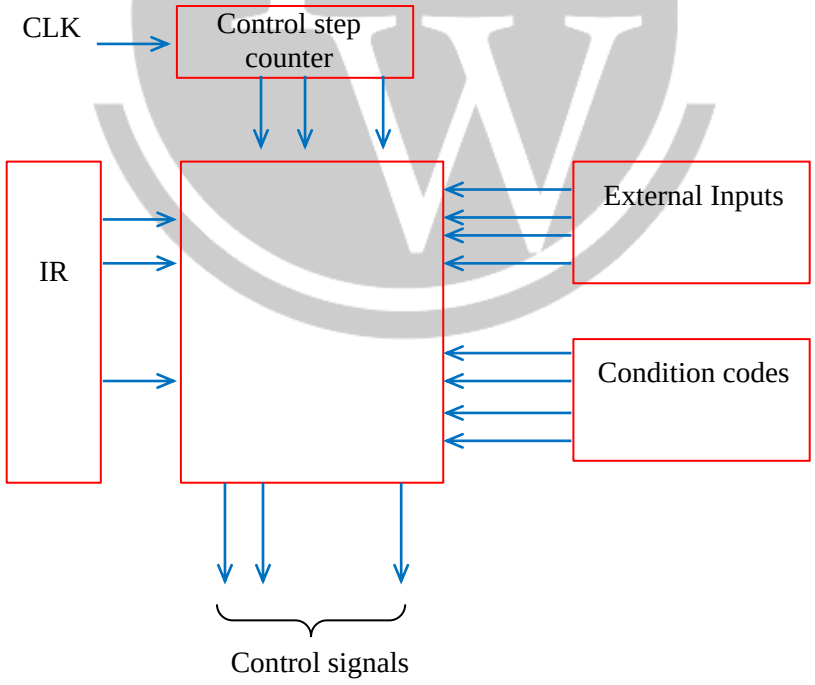
(1) Hardwired control

(2) Micro programmed control

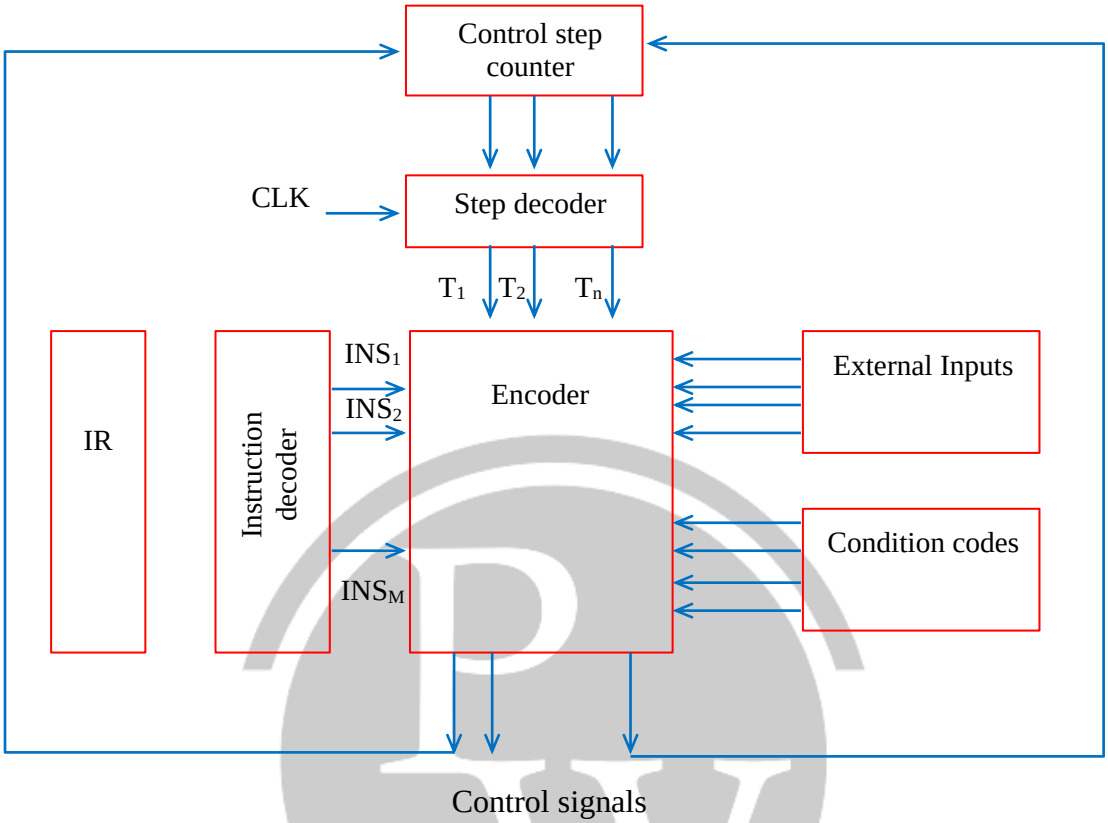
The purpose of control unit is to generate accurate timing & control signals.

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.

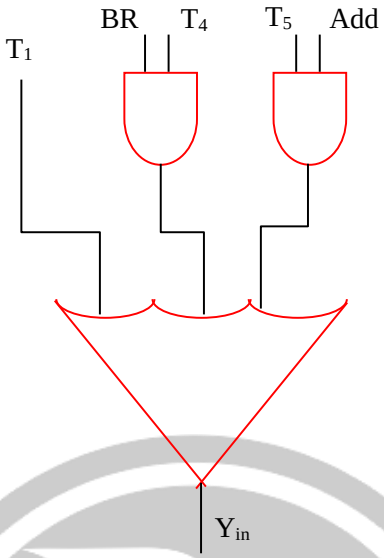
For example:

(1) The encoder circuit implements the following logic function to generate Y_{in}.

$Y_{in} = T_1 + T_5$ Add $+ T_4$ BR $+ \ldots..$

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.

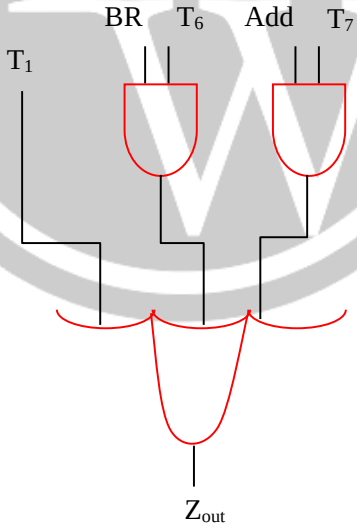
The generation of Y_{in} control signal is



(2) The logic function to generate Z_{out} signal can be given by

$Z_{out} = T_2 + T_7$ add $+ T_6$ BR + ……

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.

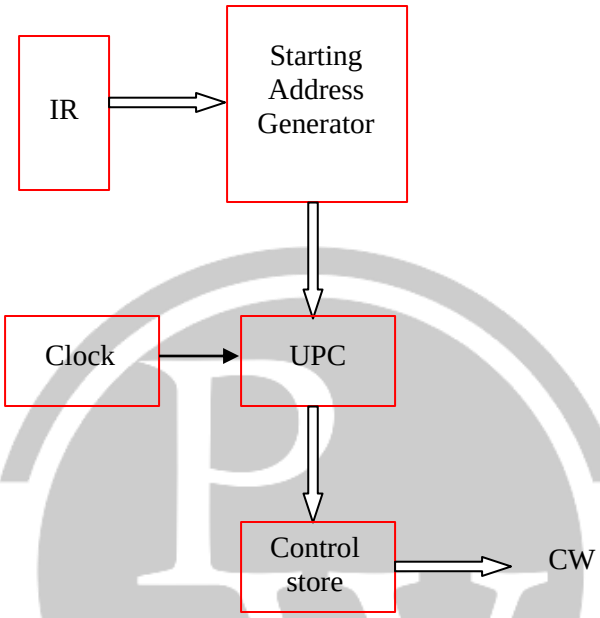


4.2 Microprogrammed Control

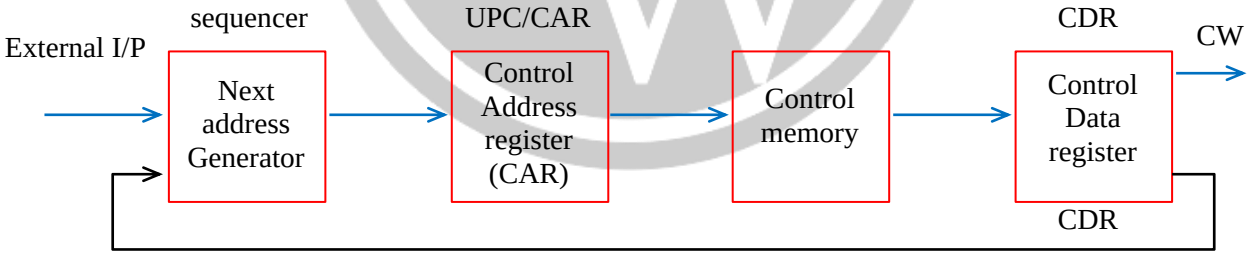
- Every instruction in a processor is implemented by a sequence of one or more sets of micro operations. Each micro operation is associated with a specific set of control lines which when activated, causes that micro operation to take place.
- Using micro programmed control, control signals are generated 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" (μPC) is used.



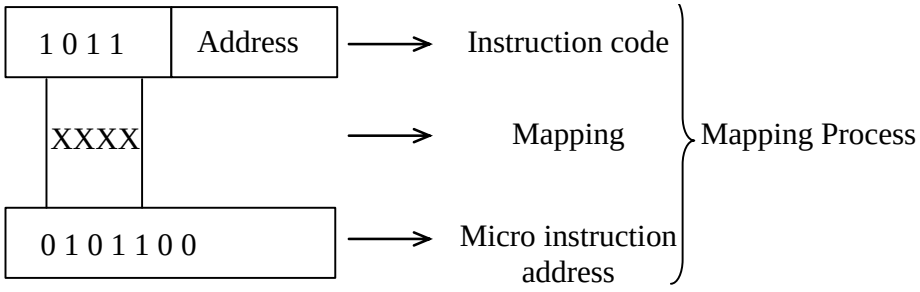
- The sequence of operations are:
  (1) CAR holds the address of next micro instruction to be read.
  (2) When address is available in UPC, the sequencer issues the READ command to CM.
  (3) The word from addressed location is read into CDR/UIR.
  (4) The UPC is incremented automatically by the clock, 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 μPC or CAR can be updated with various options using the address sequencer circuit as:

(1) When a new instruction is loaded in IR, the μPC is loaded with the starting address of the micro routine for that instruction (called mapping process).

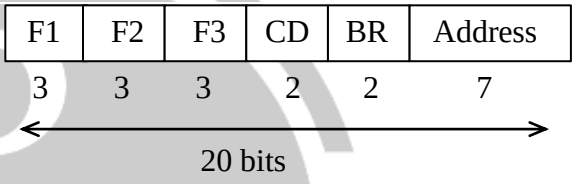
(2) When a branch instruction is (micro) encountered and branch condition satisfied, the μPC is loaded with the branch address.

(3) When END instruction is encountered, the μPC is loaded with address of first word.

(4) In any other situation, the μPC 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 & Microprogramed

	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.

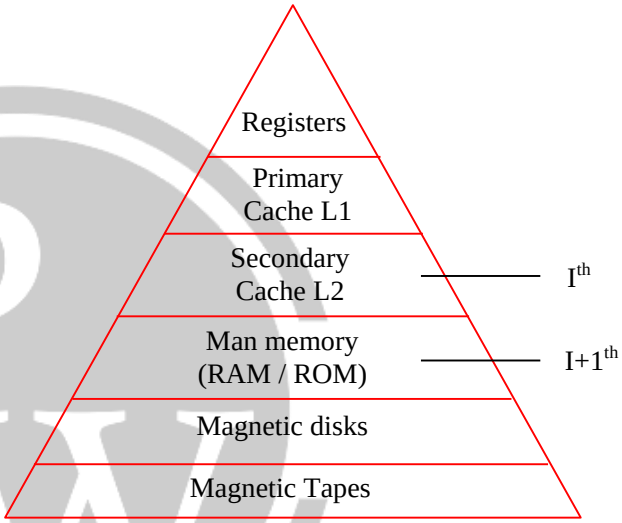
- To increase the degree of parallelism soft vertical microprogramming is used which divides the control signals into mutually exclusive groups. Each group is associated with associated number of control bits. (i.e., combination of both vertical and horizontal microprogramming).

❑❑❑

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 $(I + 1)^H$ level memory.
- Let Ti, Si, Ci, & Fi are the access time, size, cost per bit and frequency of references to the I^{th} level memory. Therefore

$$Ti < Ti + 1$$

$$Si < Si + 1$$

$$Ci > Ci + 1$$

$$fi > fi + 1$$

- Since same data presents at different levels. I^{th} level memory is the subset of $I\ i + 1^{th}$ 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

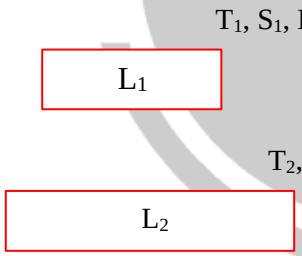
erasable / non erasable

	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)



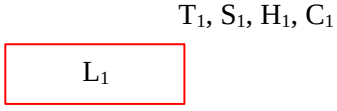
$T_{avg} = H_1 * T_1 + (1–H_1) * T_2$

Case – II: When fault occurs in L_1, must be brought from L2 to L1 memory. (**strict hierarchy**)

$T_{avg} = H_1 * T_1 + (1–H_1) * (T_2 + T_1)$

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

$$C_{Avg} = \frac{C_1S_1 + C_2S_2}{S_1 + S_2} \quad \text{(Two - level)}$$

$$C_{Avg} = \frac{C_1S_1 + C_2S_2 + C_3S_3}{S_1 + S_2 + S_3} \quad \text{(Three - level)}$$

❑❑❑

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 8^{th} 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

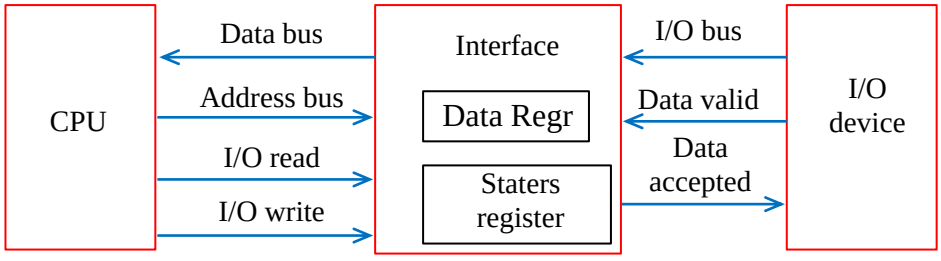
Data transfer b/n the central computer (main memory) and I/o device may be handled in a variety of modes like programmed I/o, Interrupt – Initiated I/o & Direct memory access.

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 extents several instruction.



F – flag bit

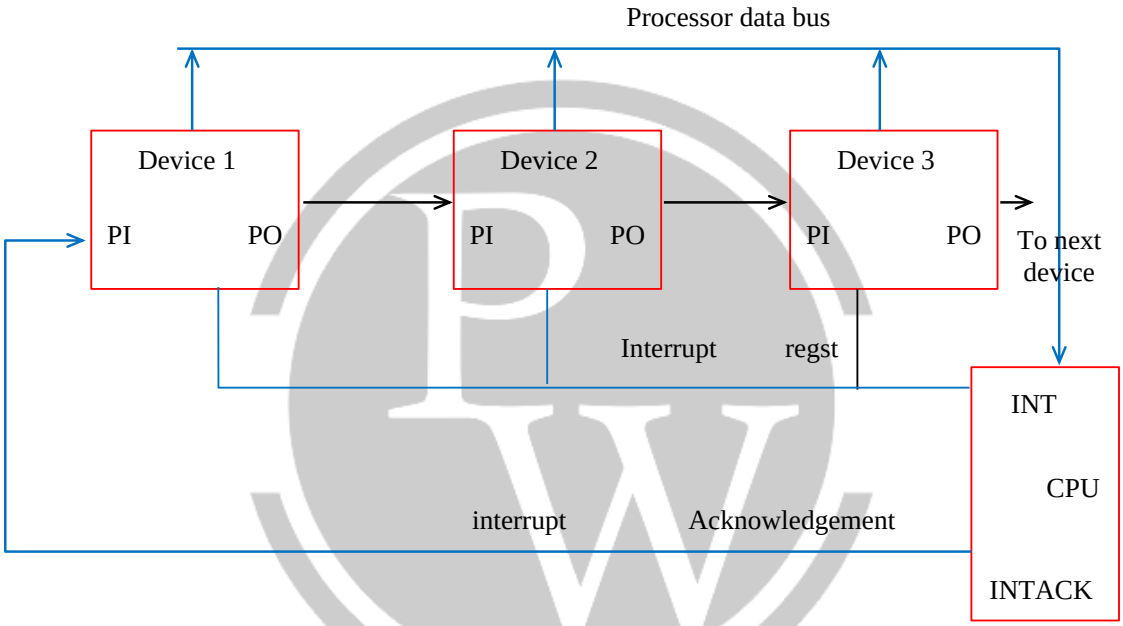
- (Data transfer from I/o device to CPU) The device transfers bytes of data one at a time as they are available. When a byte of data is available the device places it on I/o bus and enables its data valid line. The Interface accepts the byte into its data register and enables the data accepted line. 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

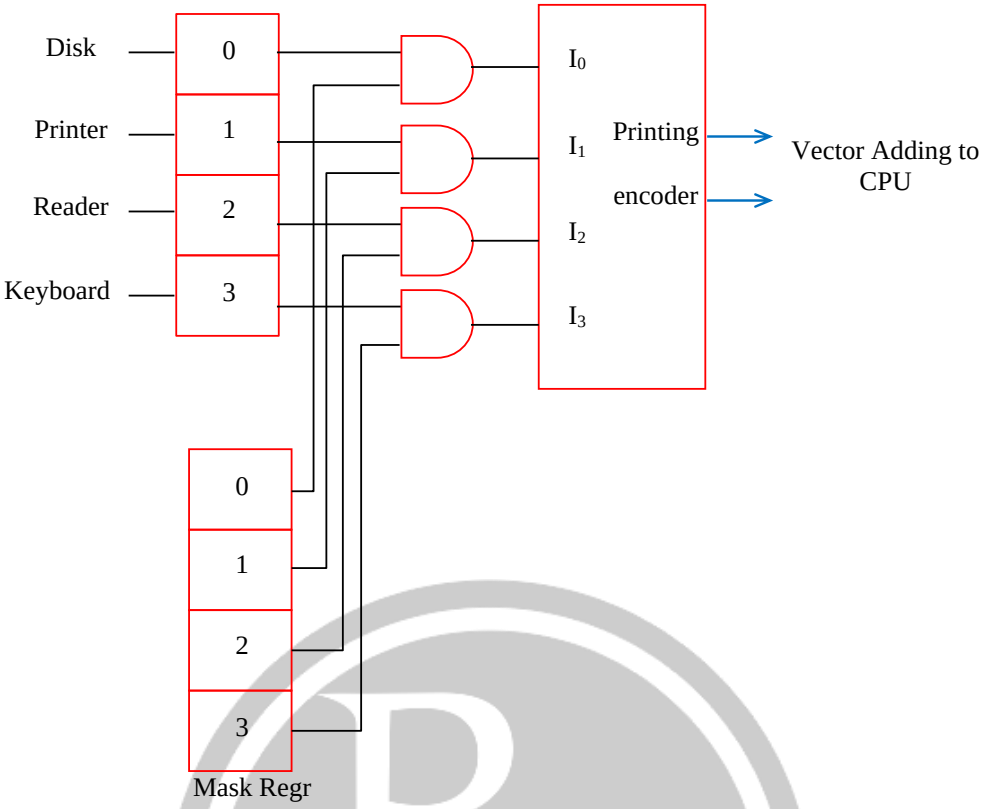
6.2.3. Daisy – Chaining Priority (Serial – Priority Interrupt)

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 revised. 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.

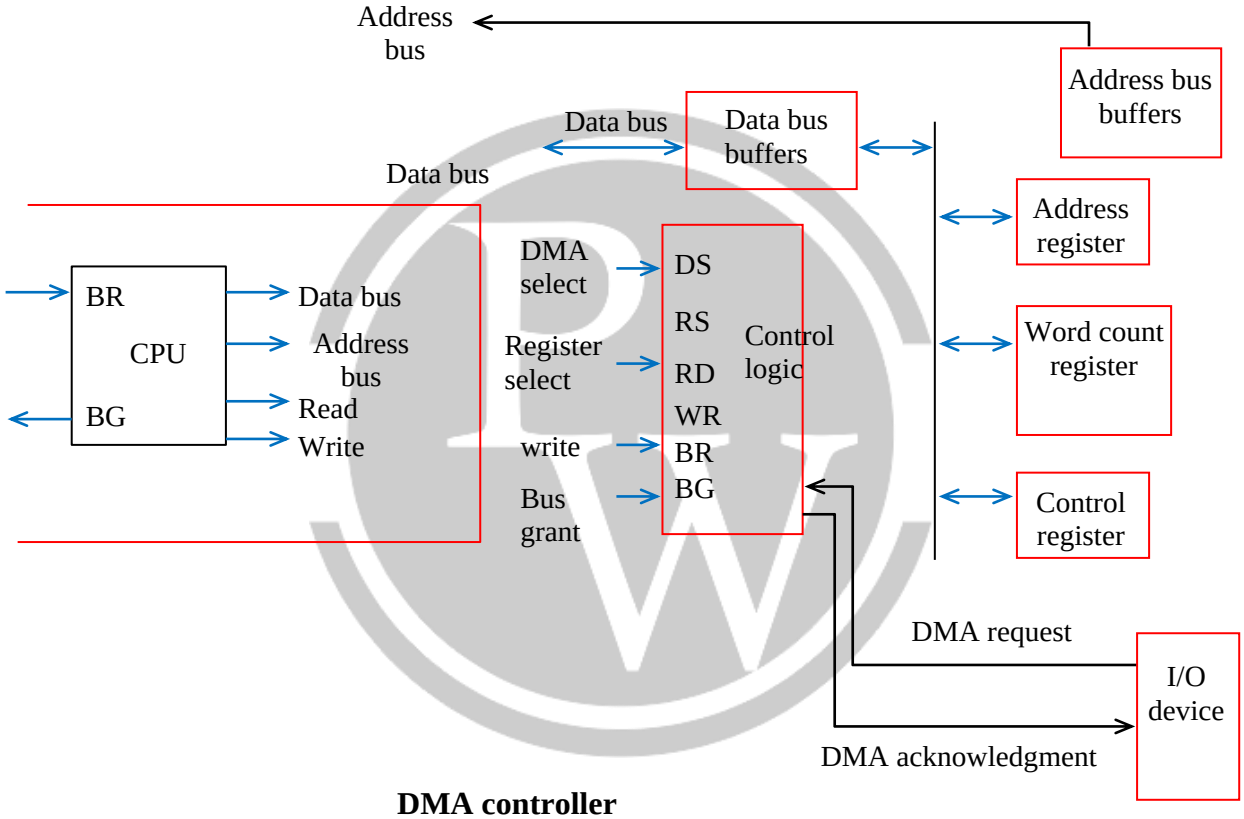
6.3. The DMA Transfer Take Place in Two Modes

(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.



DMA controller

- 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.

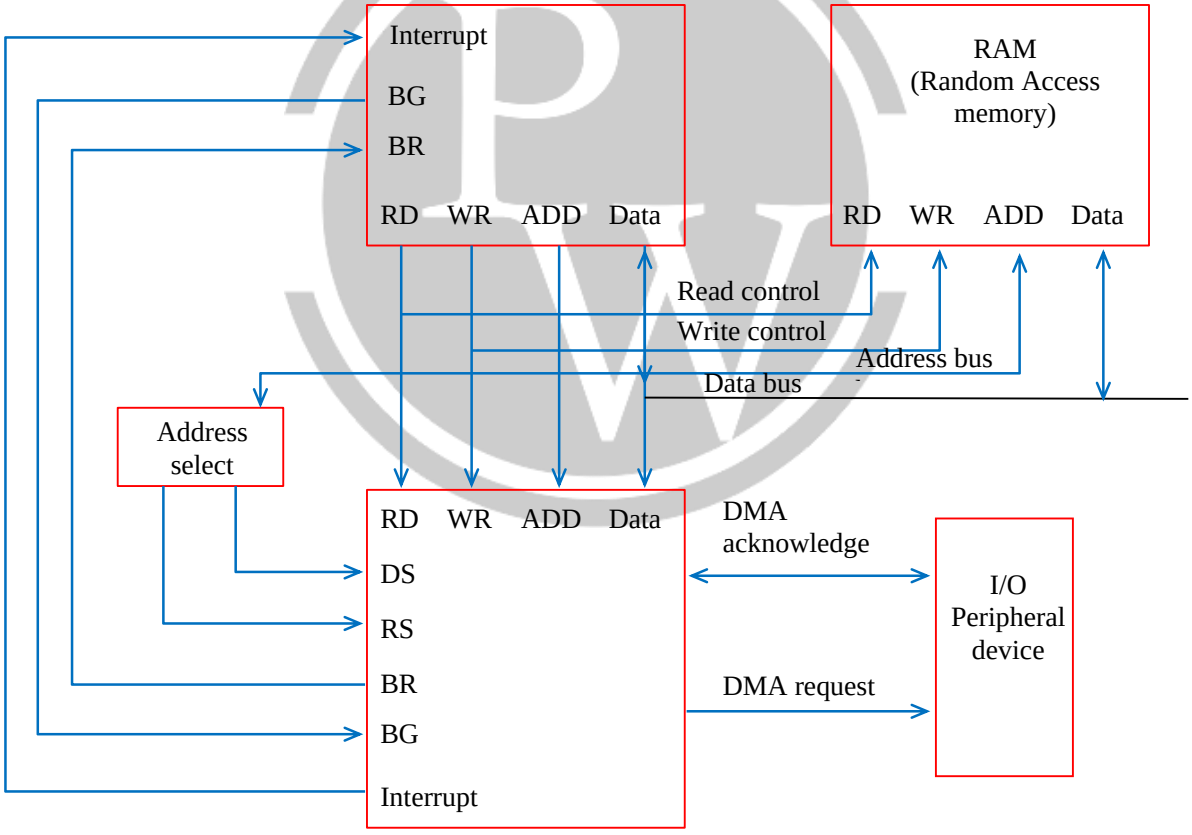
(3) Control of specify mode of transfer such as read or write.

(4) A control to start the DMA transfer.

- Once DMA is initialized, the CPU stops communicating with the DMA unless it receivers 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 on 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).

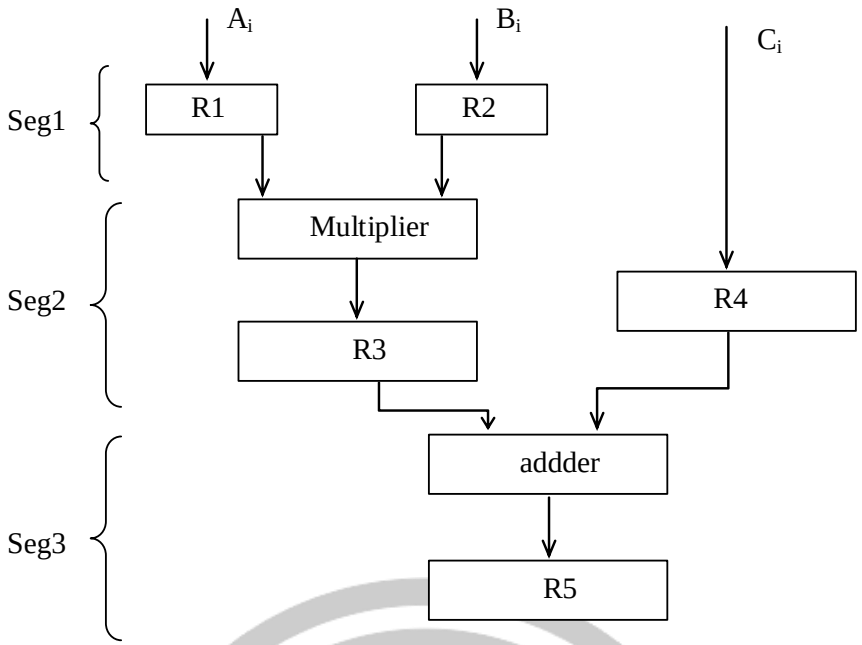
  (v) Using pipelining in unit processor systems.

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:

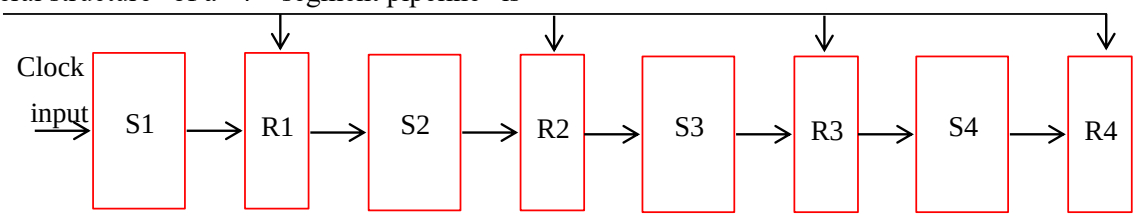
The perform Ai * Bi + Ci , for i = 1 to 6. Each sub operation multiply & add implemented in a segment with in a pipeline. Each segment has one or more registers.



- 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	A2 * B2	C2	A1 * B1 + C1
4	A4	B4	A3 * B3	C3	A2 * B2 + C2
5	A5	B5	A4 * B4	C4	A3 * B3 + C3
6	A6	B6	A5 * B5	C5	A4 * B4 + C4
7	---	---	A6 * B6	C6	A5 * B5 + C5
8	---	---	---	---	A6 * B6 + C6

- 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 Si that performs a sub operation.
- The segments are separated by registers Ri 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
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

Clock cycles

- "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 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" (td)
- 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.

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 $t_n \simeq K * t_p$

$$S = \frac{n * K * t_p}{(K + (n-1)) * t_p} \qquad \therefore t_n \simeq nt_p$$

$$S = \frac{nK}{K + (n-1)}$$

$$S = \frac{n\,t_n}{n * t_p} \qquad \therefore K + n - 1 \simeq n$$

$$\boxed{S = \frac{t_n}{t_p}}$$

For large number of tasks

$$S = \frac{nk}{K + n - 1} \qquad S = \frac{t_n}{t_p}$$

Or

$$= \frac{nK}{n} \qquad S = \frac{K \times t_p}{t_p}$$

$$\boxed{S = K} \qquad \boxed{S = K}$$

- 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]. (K).

$$S_{ideal} = S_{max} = K$$

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 influence by

  Number of instructions

  Uneven stage delays

  Buffer overhead (Interstage delay)

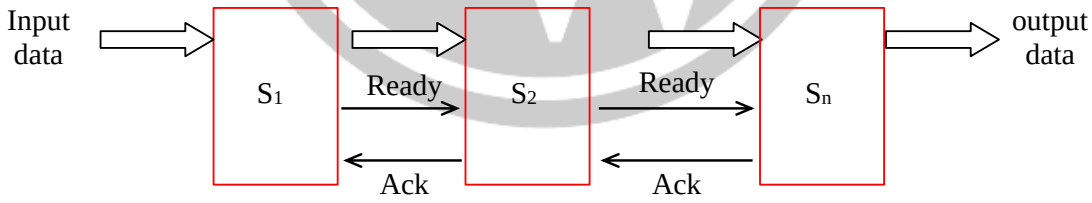
  Dependencies.
- The objective of pipelines is $CPI_{avg} = 1$.

  (i.e. Clocks per instruction – CPI)
- Depending on control mechanism used, the pipelines can be categorized as

7.2.6 I Asynchronous pipeline

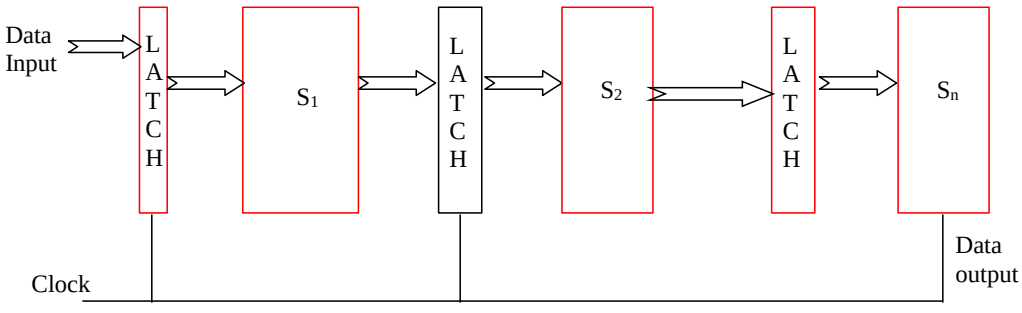
- Data flow along the pipeline stages is controlled by handshaking protector i.e. when S_i is ready to send/transmit, it sends ready signal to S_{i+1} and S_{i+1} send A.

  ck to Si 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.

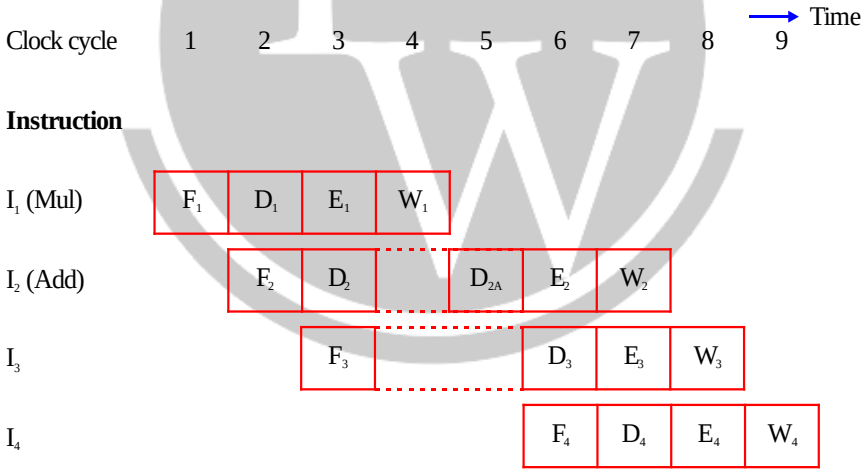


Fig: Pipeline stalled by data dependency between D_2 and W_1

- 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

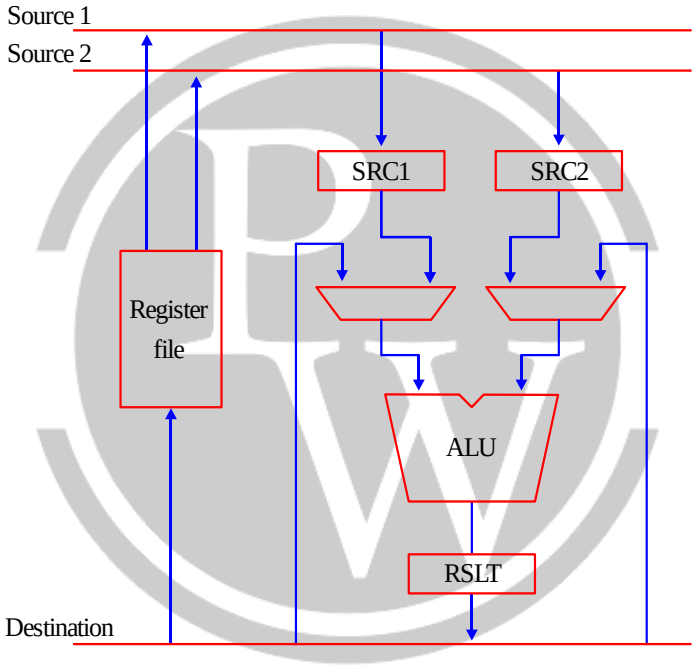
Mul R5.R4.R6

- 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 unit the W step of multiply instruction has been completed. Completion of step D_2 must be delayed to clock cycle 5, and is shown as step D_{2A} in figure. Instruction I_3 is fetched in cycle 3, but its decoding must be delayed because step D_3 cannot precede D_2. Hence, pipelined execution is stalled for two cycles.

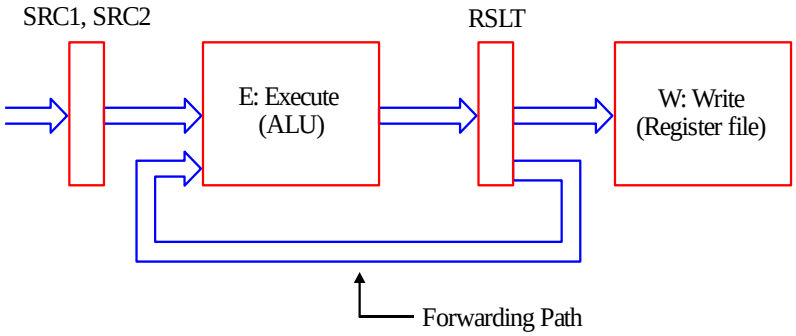
7.2.10 Operand Forwarding

- The data hazard just described arises because one instruction, instruction I_2 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 E_1. Hence, the delay can be reduced, or possibly eliminated, if we arrange for the result of instruction I_1 to be forwarded directly for use in step E_2.

Figure shows a part of the processor data path involving the ALU and the register file.



(a) Datapath



(b) Position of the source and result registers in the processor pipeline

- 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 bust 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 E_2. 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:

I_1 : Mul R2, R3, R4
NOP
NOP
I_2 : Add R5, R4, R6

- 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.

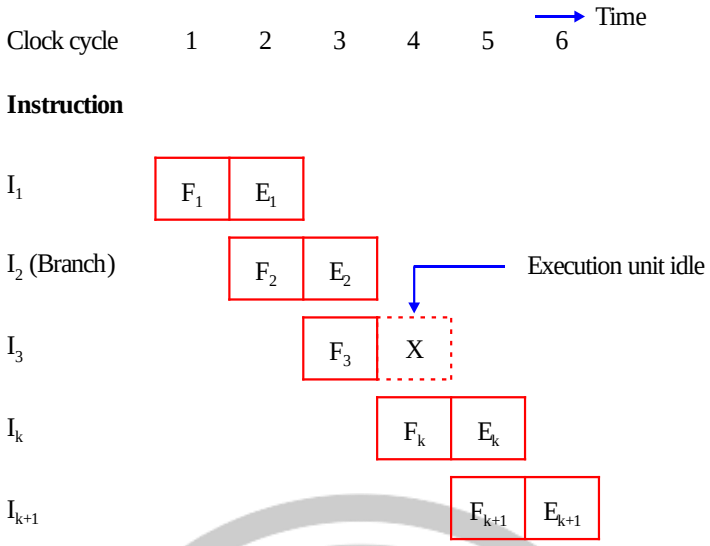
7.3 INSTRUCTION HAZARDS

- 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.

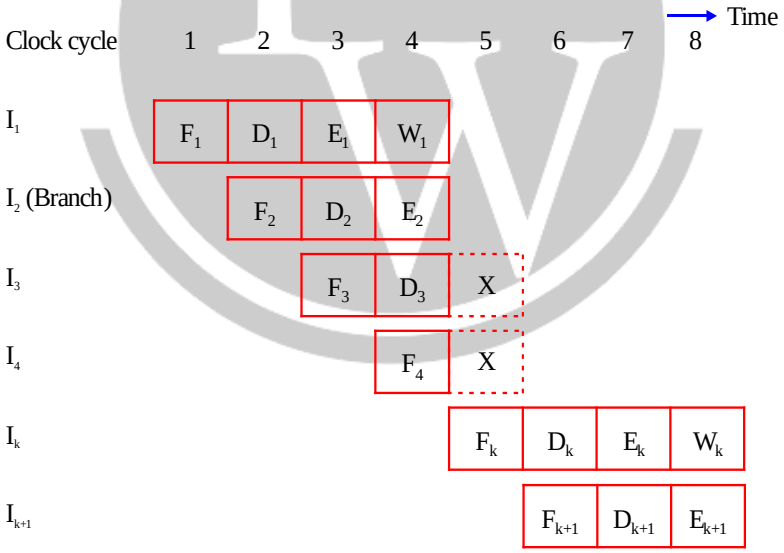
7.4 UNCONDITIONAL BRANCHES

- Figure shows a sequence of instructions being executed in a two-stage pipeline. Instructions I_1 to I_3 are stored at successive memory addresses, and I_2 is a branch instruction. Let the branch target be instruction I_h. In clock cycle 3, the fetch operation for instruction I_3 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 I_3, 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

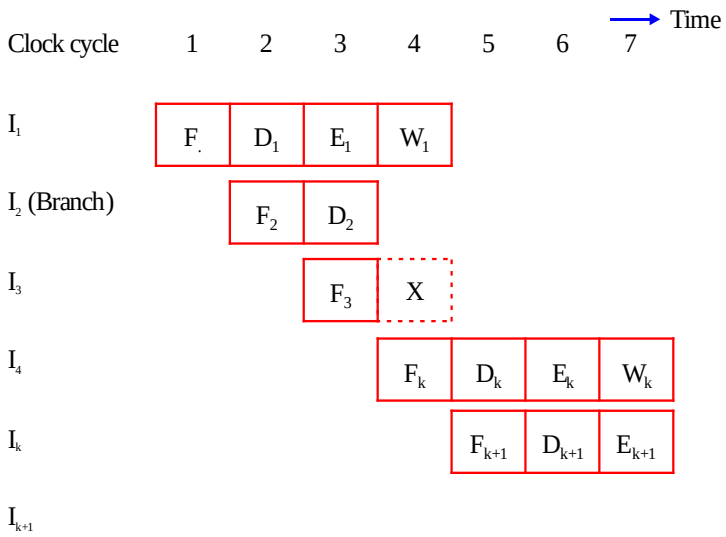
instruction on a four-stage pipeline. We have assumed that the branch address is computed in step E_2. Instructions I_3 and I_4 must be discarded, and the target instruction, I_h, is fetched in clock cycle 5. Thus, the branch penalty is two clock cycles.



- 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 D_2, 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

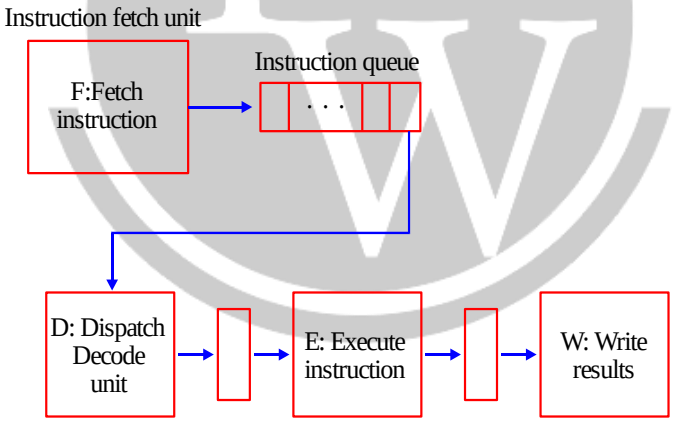


Fig: Use of an instruction queue in the hardware organization of fig (b)

- 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

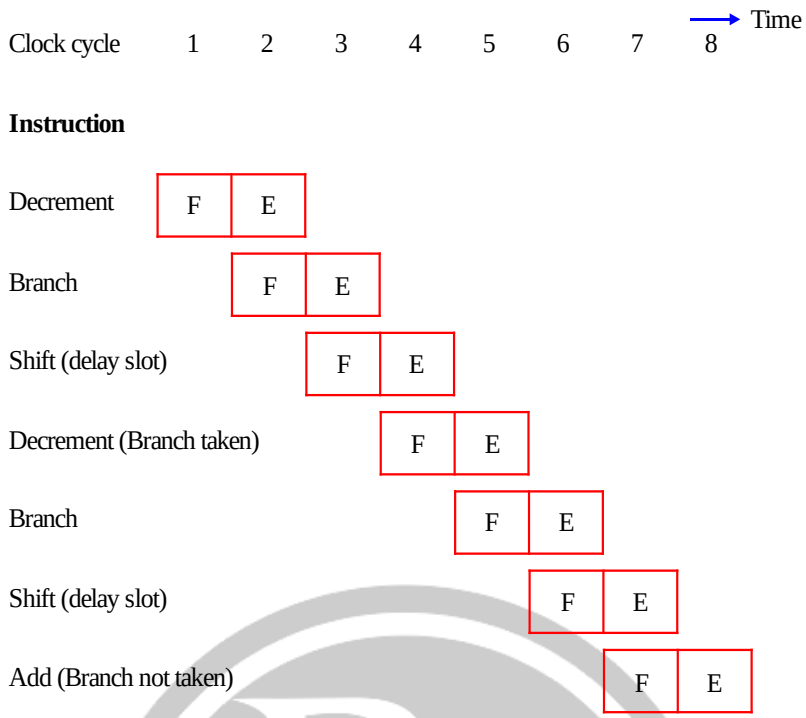
- In Figure the processor fetches instruction I_3 before it determines whether the current instruction, I_2, is a branch instruction. When execution of I_2 is completed and a branch is to be made, the processor must discard I_3 and fetch the instruction at the branch target. The location following a 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

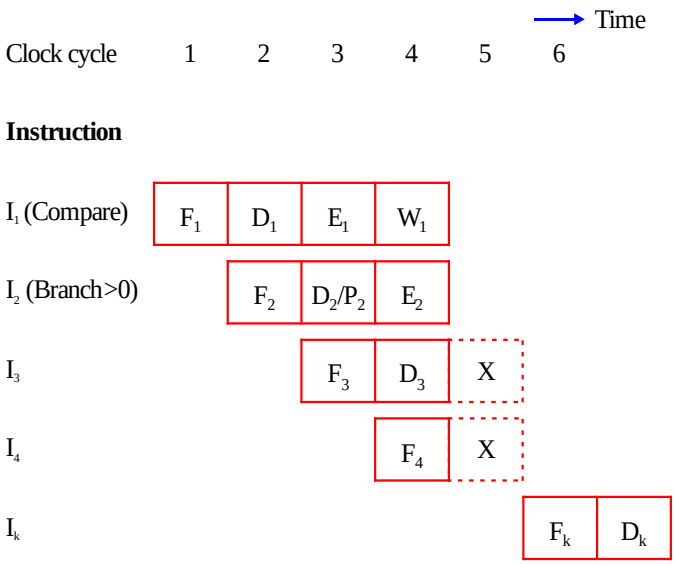
(b) Reordered instructions



- 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 o find two instructions preceding the branch instruction that it can move into the delay slots without introducing a logical error. The changes of finding two such instructions are considerably less than the changes 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 by 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 nest 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 E_2 rather than D_2 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   . . .
```

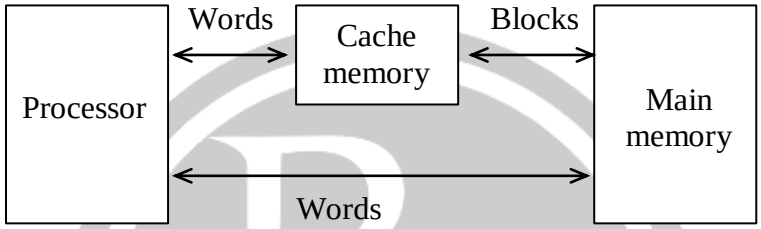
(b) Instructions reordered

❐❐❐

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
    1. **Temporal:** It means that recently executed instruction is likely to be executed again very soon.
    2. **Spatial:** It means that instructions in close proximity to Cl recently executed instruction,

       Example: loops, nested loops, procedure calls. Etc.

- The performance of cache is measured using Hit ratio

$$\text{Hit ratio} = \frac{\text{no. of hits}}{\text{Total CPU references}} \times 100$$

$$\text{Hit ratio} = \frac{\text{no. of hits}}{\text{no.of hits} + \text{no.of misses}} \times 100$$

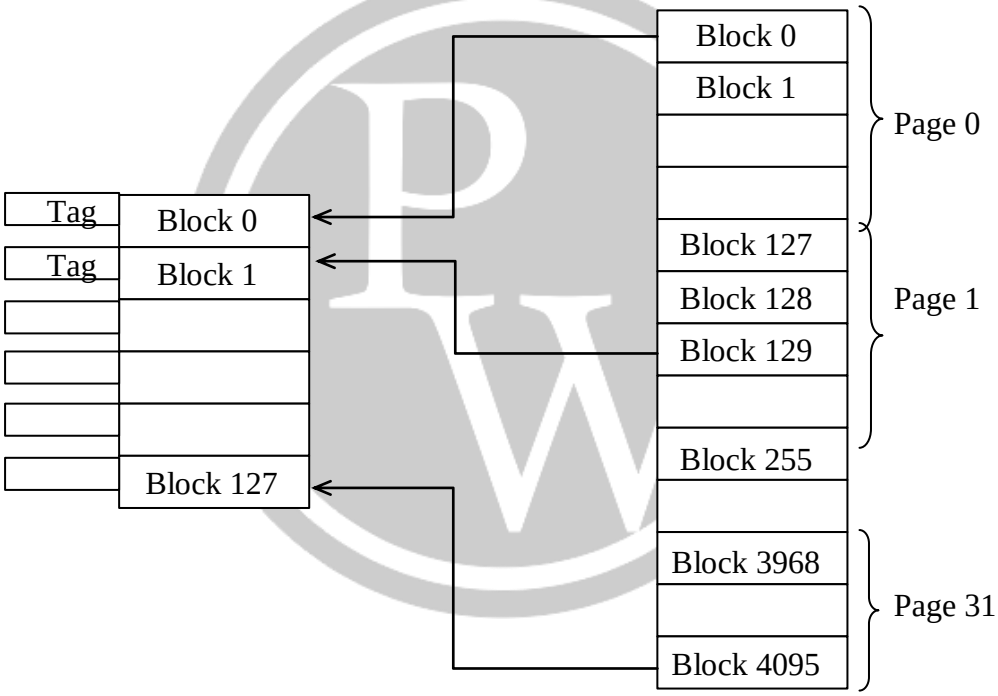
- The performance of cache can be analysed with the following characteristics.
  (1) Cache size (Small in KB's)
  (2) Block or line size
  (3) No. of levels of cache
  (4) Cache mapping process.
  (5) Cache replacement algorithm
  (6) Cache updating scheme.

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 mod n. Where n is the no. of blocks in the cache.



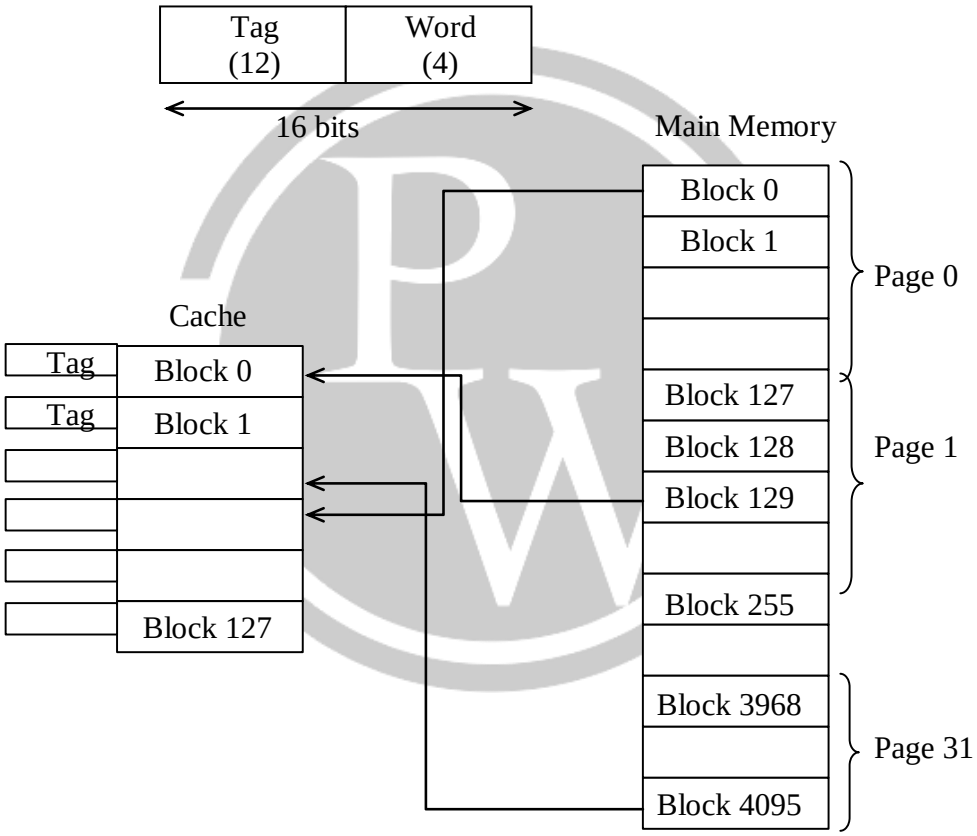
- If the processor needs to access same memory locations from two different pages of the main memory frequently, hen miss & will be more.
- Easy to implement but hit ratio is less.
- To implement this the address is divided into Three fields.
- Example :

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.

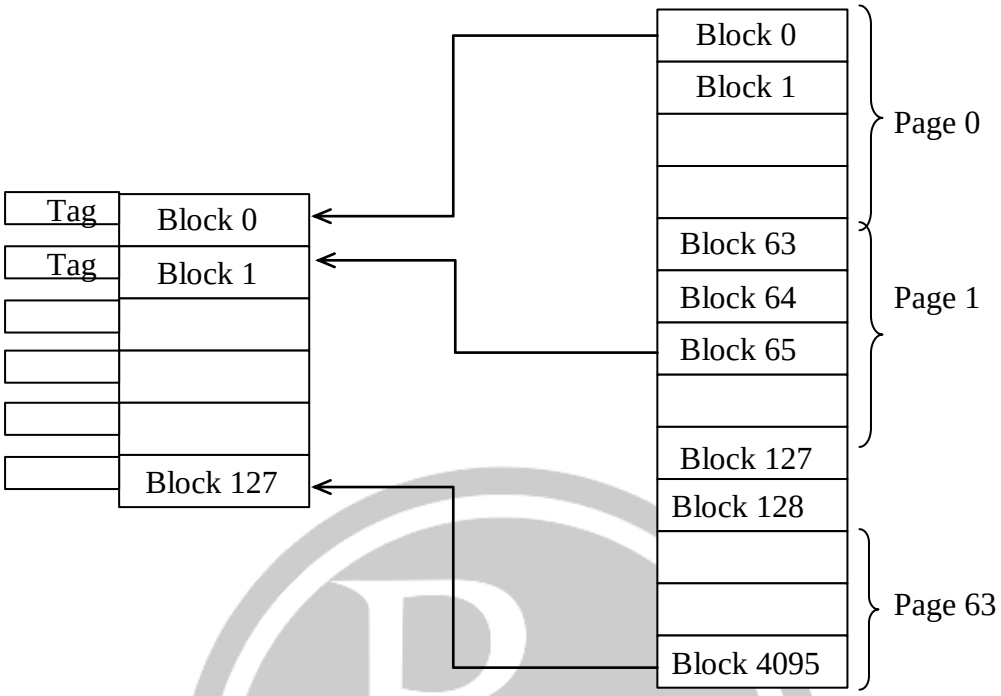


- We need an algorithm to select the block to be replaced.

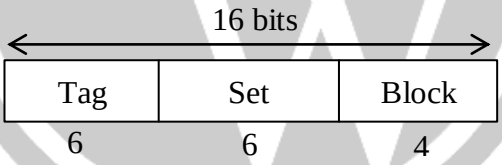
8.2.3 Set – Associative Mapping

- 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 se t 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).3

(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 ;

	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)

Ans.

(b) How many times block '0' is modified in the cache memory.

Ans. 0, 8, 16, - - - - - 80, 88, 96. ⇒ 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 case. 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 wit 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 present 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 x 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:

(a) The number of RAM ICs $= \frac{\text{Total'RAM size}}{\text{RAM IC size}}$

$= \frac{512\times 8}{128\times 8}$

$= 4$

(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

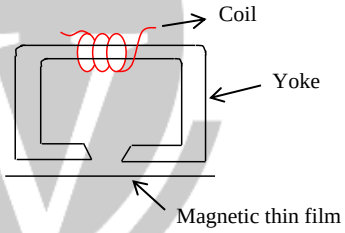
10.1 Introduction

- The cost per bit of stored information is high in c semi-conductor 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

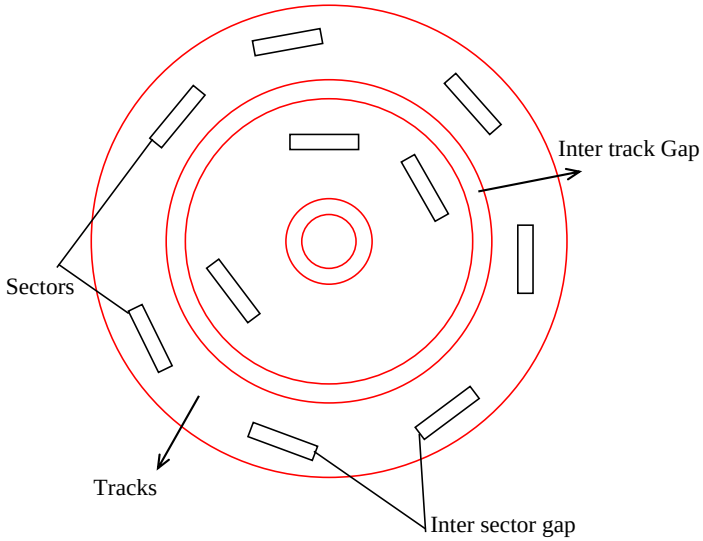
- A magnetic disk is a thin, circular metal plate. It is coated with a thin magnetic film usually on both sides.
- Digital information is stored on the magnetic disk by magnetizing the magnetic surface in a particular direction.
- Magnetic disks are semi random access memories.

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 concentration 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

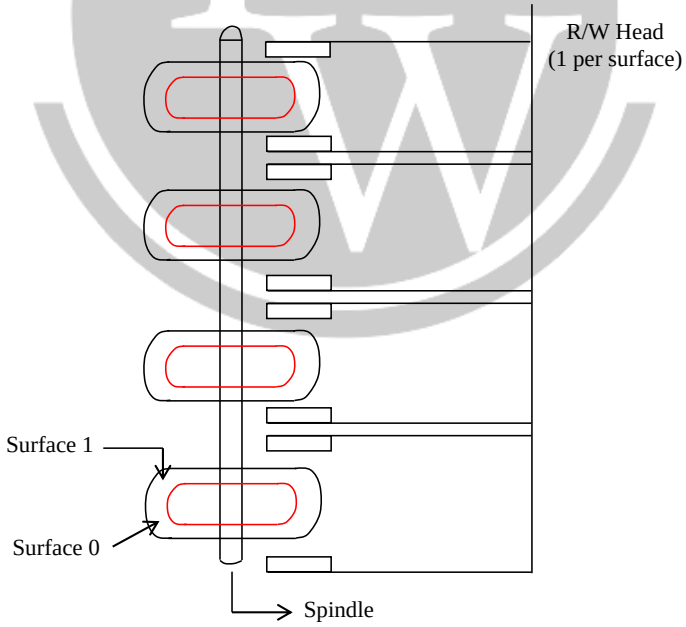
(i) Head Motion - Fixed head (one per track)
Movable head (one per surface)

(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 more the disk arm to the required track.

Rotational Delay: The time taken for the beginning of sector to reach. (latency)

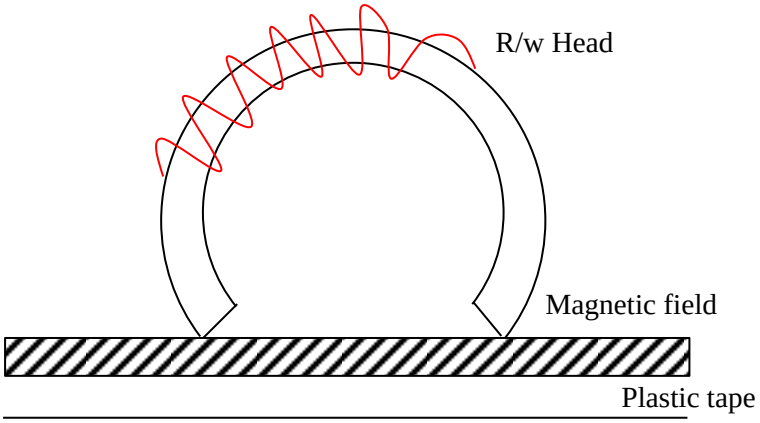
Access time: The sum of seek time and the rotational delay.

- 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.

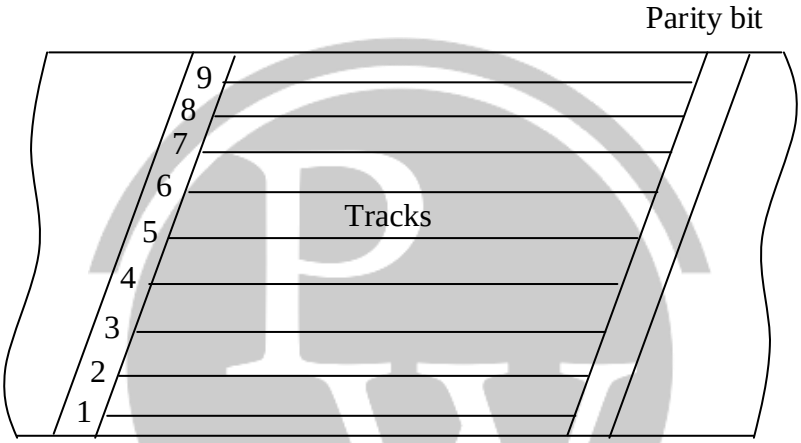
10.3 Secondary Storage Devices

10.3.1. Magnetic Tapes

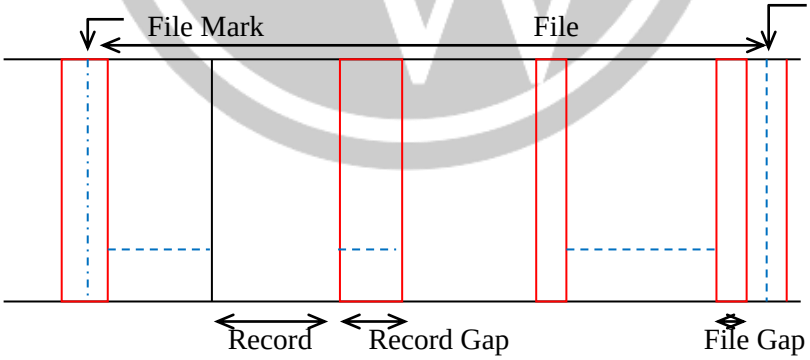
- Magnetic tape is one of the most popular storage medium for large data that are sequentially accessed and processed.
- The tape is formed by depositing magnetic film on a very thin and ½ or ¼ inch wide plastic tape.
- Usually, iron oxide is used as a magnetizing material.
- The information is recorded 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 a 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 9^{th} 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

$$D_{eff} = D * \frac{B_L}{B_L + G_L}$$

✓ 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.

$$d_{eff} = \frac{t_D \times d}{t_D + t_G + t_{ss}}$$

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

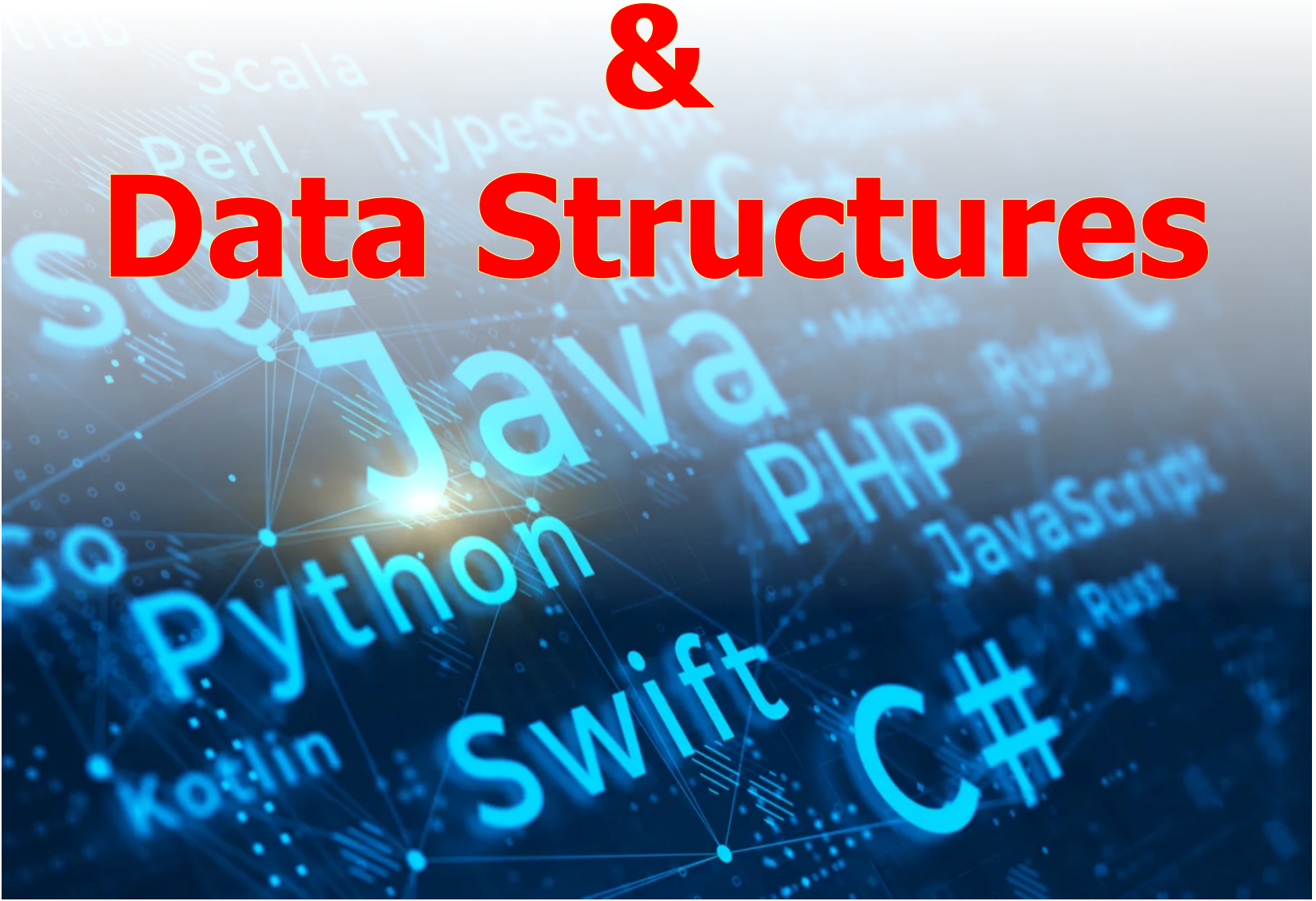
GATE Wallah

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

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 that 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.

- ~ 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 × 2 + 5 : 6! = 7

⇒ Left expression: 7 > 2 is true

⇒ So, the value returned would be the middle argument.

i.e., 3 × 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

L_1 M_1 R_1

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++)
statement.
```
represent an infinite loop

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)
statement

OR

while (expression)
{
statement-1
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
statement
while (expression);

OR

do {
statement-1
statement-2
⋮
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);
}
}
```

Scope S_1

Scope S_2

Output: 1

Compiler looks for 'a' in scope S_1, because S_1 does not have variable a, it will look into higher scope S_2 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:

$$\text{main ()} \xrightarrow{\text{calling}} \text{f()} \xrightarrow{\text{calling}} \text{g()}$$

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

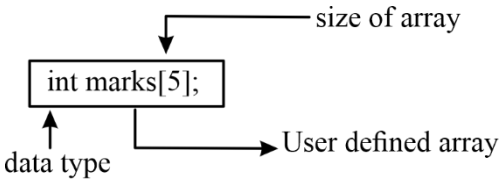
- **Fixed Size**: **1)** Whatever size, we define at the time of declaration of the array, we can't exceed the limit.
  2) It does not grow the size dynamically like linked list.

Declaration of C Array

Syntax:

```
data type array name [array_size]
```

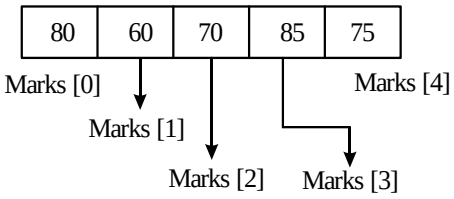
Example:



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;
}
```

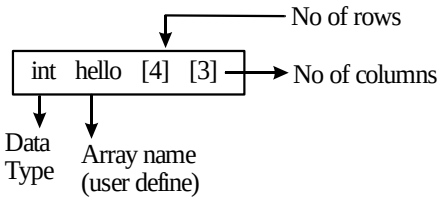
4.1.4 Two-Dimensional Array

- 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.

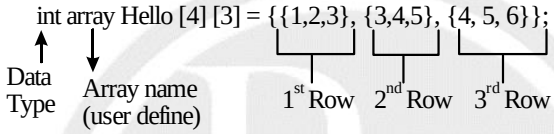
4.1.5 Declaration of two-dimensional array in C syntax

```
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;
}
```

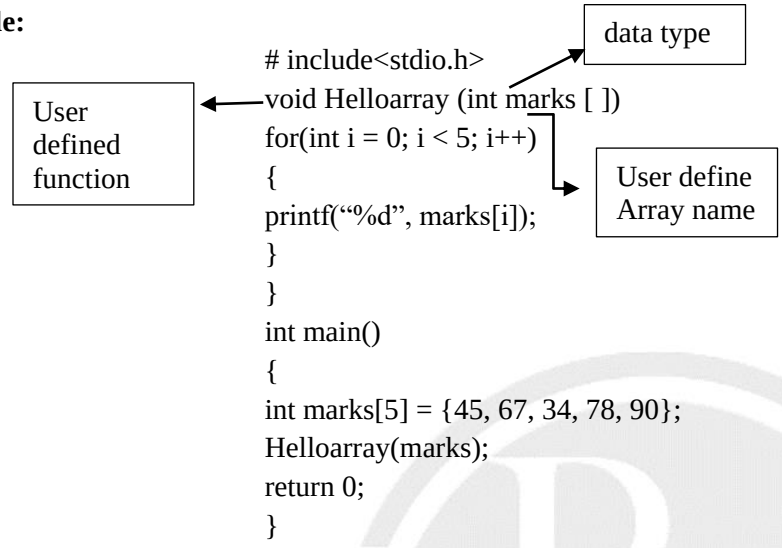
Output:

```
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>
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 fraud.
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

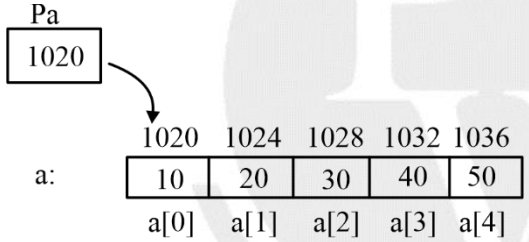
1. int A[ ] = {10, 20, 30}: valid
2. int A[3] = {10, 20, 30}: valid
3. int A[ ] ; invalid
4. int A[3] ; valid
5. int A[2][3]; valid
6. int A[ ] [3]; = {10, 20, 30}: invalid
7. int A[2] [3]; = {1, 2, 3, 4, 5, 6}: valid
8. int A[ ] [3]; = {1, 2, 3, 4, 5, 6}: valid
9. int A[2] [3] [2]; invalid
10. int A[ ] [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 pointes.
- 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 ≡ & a[i]
  $\Rightarrow$ *(Pa+i) ≡ *& a[i] ≡ a [i]
- Array name always represent address of the very first element
  i.e., Pa = &a[0];
  and

Pa = a ;

both are same.

- a[i] ≡ *(a+i) ≡ *(i + a) ≡ i [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];

i. a++, ++a, – –a , a – – are all invalid as we cannot apply increment/decrement operator on constants:

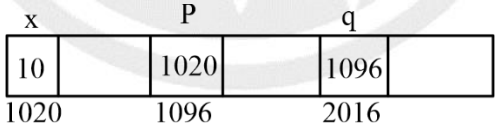
ii. Array name cannot be Lvalue of on assignment statement.
i.e.
array_name = any expression/address/value is invalid

- 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
```



p ≡ 1020 i, e address of x
*P ≡ value at (memory location 1020)
*p ≡ 10
q ≡ 1096 i,e address of variable p
*q ≡ value at (memory location 1096)
≡ 1020 which is again an address
*q ≡ value at (memory address 1020)
**q ≡ 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 variables 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 is 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 costing of void

(int) ptr dereferences the typecasted void pointer variable

- Pointer arithmetic in not possible on void pointer.
  i,e……. ptr+ +, + + ptr, – –ptr, ptr – –, ptr +1, ptr -1 is not allowed an void pointer.

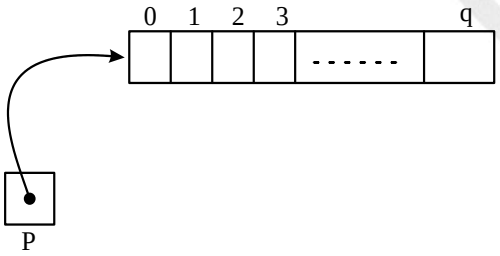
- An uninitialized pointer holds a garbage value while a NULL pointer holds a defined value but not a valid address.

4.1.10 Memory leakage problem

- Whenever a pragrammer allocated memory dynamically then it is the reponsibility 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 availbale memory.

4.1.11 Understanding Declartion

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 take 2 integer arguments and returns an integer value.