NAAC ACCREDITED

तेजस्वि नावधीतमस्तु

ISO 9001:2008 & 14001:2004

FAIRFIELD

Institute of Management & Technology

'A' Grade Institute by DHE, Govt. of NCT Delhi and Approved by the Bar Council of India and NCTE

Reference Material for Three Years

Bachelor of Computer Application

Code : 020

Semester – II

FIMT Campus, Kapashera, New Delhi-110037, Phones : 011-25063208/09/10/11, 25066256/ 57/58/59/60
Fax : 011-250 63212 Mob. : 09312352942, 09811568155 E-mail : fimtoffice@gmail.com Website : www.fimt-ggsipu.org

INDEX

Three Years

Bachelor of Computer Application

Code : 020

Semester – II

S.NO.	SUBJECTS	CODE	PG.NO.
1	***MATHEMATICS-II***	*102*	*5-32*
2	***PRINCIPLES OF MANAGEMENT***	*104*	*33-59*
3	***DIGITAL ELECTRONICS***	*106*	*60-206*
4	***DATA STRUCTURE USING C***	*108*	*207-215*
5	***DATABASE MANAGEMENT SYSTEM***	*110*	*216-227*

BCA102

(MATHEMATICS)

1) **Define sets, its types & operations**

ANS: **Definition OF SETS**

A set is a well defined collection of distinct objects. The objects that make up a set (also known as the elements or members of a set) can be anything: numbers, people, letters of the alphabet, other sets, and so on. Georg Cantor, the founder of set theory, gave the following definition of a set at the beginning of his *Beiträge zur Begründung der transfiniten Mengenlehre*

A set is a gathering together into a whole of definite, distinct objects of our perception or of our thought – which are called elements of the set.

Sets are conventionally denoted with capital letters. Sets *A* and *B* are equal if and only if they have precisely the same elements

As discussed below, the definition given above turned out to be inadequate for formal mathematics; instead, the notion of a "set" is taken as an undefined primitive in axiomatic set theory, and its properties are defined by the Zermelo–Fraenkel axioms. The most basic properties are that a set "has" elements, and that two sets are equal (one and the same) if and only if every element of one is an element of the other.

Describing sets

There are two ways of describing, or specifying the members of, a set. One way is by intensional definition, using a rule or semantic description:

A is the set whose members are the first four positive integers.
B is the set of colors of the French flag.

The second way is by extension – that is, listing each member of the set. An extensional definition is denoted by enclosing the list of members in curly brackets:

$C = \{4, 2, 1, 3\}$
$D = \{\text{blue, white, red}\}$.

Every element of a set must be unique; no two members may be identical. (A multi set is a generalized concept of a set that relaxes this criterion.) All set operations preserve this property. The order in which the elements of a set or multi set are listed is irrelevant (unlike for a sequence or tuple). Combining these two ideas into an example

$\{6, 11\} = \{11, 6\} = \{11, 6, 6, 11\}$

because the extensional specification means merely that each of the elements listed is a member of the set.

For sets with many elements, the enumeration of members can be abbreviated. For instance, the set of the first thousand positive integers may be specified extensionally as:

{1, 2, 3, ..., 1000},

where the ellipsis ("...") indicates that the list continues in the obvious way. Ellipses may also be used where sets have infinitely many members. Thus the set of positive even numbers can be written as {2, 4, 6, 8, ... }.

The notation with braces may also be used in an intentional specification of a set. In this usage, the braces have the meaning "the set of all ...". So, E = {playing card suits} is the set whose four members are ♠, ♦, ♥, and ♣. A more general form of this is set-builder notation, through which, for instance, the set F of the twenty smallest integers that are four less than perfect squares can be denoted:

$$F = \{n^2 - 4 : n \text{ is an integer; and } 0 \leq n \leq 19\}.$$

In this notation, the colon (":") means "such that", and the description can be interpreted as "F is the set of all numbers of the form $n^2 - 4$, such that n is a whole number in the range from 0 to 19 inclusive." Sometimes the vertical bar ("|") is used instead of the colon.

One often has the choice of specifying a set intensionally or extensionally. In the examples above, for instance, $A = C$ and $B = D$.

Membership

The key relation between sets is *membership* – when one set is an element of another. If a is a member of B, this is denoted $a \in B$, while if c is not a member of B then $c \notin B$. For example, with respect to the sets A = {1,2,3,4}, B = {blue, white, red}, and $F = \{n^2 - 4 : n$ is an integer; and $0 \leq n \leq 19\}$ defined above,

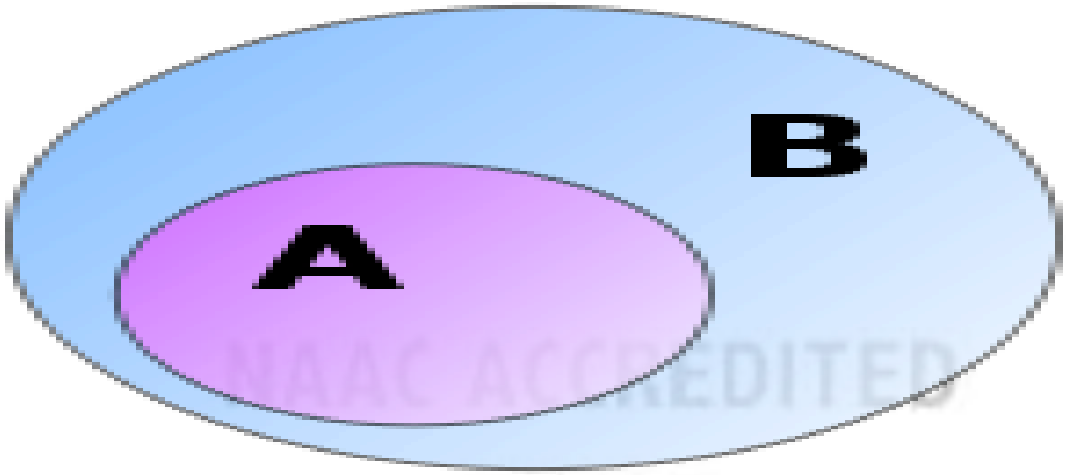
$4 \in A$ and $12 \in F$; but
$9 \notin F$ and green $\notin B$.

Subsets

If every member of set A is also a member of set B, then A is said to be a *subset* of B, written $A \subseteq B$ (also pronounced A *is contained in* B). Equivalently, we can write $B \supseteq A$, read as B *is a superset of* A, B *includes* A, or B *contains* A. The relationship between sets established by $\subseteq$ is called *inclusion* or *containment*.

If A is a subset of, but not equal to, B, then A is called a *proper subset* of B, written $A \subsetneq B$ (A *is a proper subset of* B) or $B \supsetneq A$ (B *is a proper superset of* A).

Note that the expressions $A \subset B$ and $B \supset A$ are used differently by different authors; some authors use them to mean the same as $A \subseteq B$ (respectively $B \supseteq A$), whereas other use them to mean the same as $A \subsetneq B$ (respectively $B \supsetneq A$).



A is a **subset** of *B*

Example:

- The set of all men is a proper subset of the set of all people.
- $\{1, 3\} \subsetneq \{1, 2, 3, 4\}$.
- $\{1, 2, 3, 4\} \subseteq \{1, 2, 3, 4\}$.

The empty set is a subset of every set and every set is a subset of itself:

- $\emptyset \subseteq A$.
- $A \subseteq A$.

An obvious but useful identity, which can often be used to show that two seemingly different sets are equal:

- $A = B$ if and only if $A \subseteq B$ and $B \subseteq A$.

A partition of a set *S* is a set of nonempty subsets of *S* such that every element *x* in *S* is in exactly one of these subsets.

Power sets

The power set of a set *S* is the set of all subsets of *S*, including *S* itself and the empty set. For example, the power set of the set {1, 2, 3} is {{1, 2, 3}, {1, 2}, {1, 3}, {2, 3}, {1}, {2}, {3}, ∅}. The power set of a set *S* is usually written as *P*(*S*).

The power set of a finite set with *n* elements has 2^n elements. This relationship is one of the reasons for the terminology *power set*. For example, the set {1, 2, 3} contains three elements, and the power set shown above contains $2^3 = 8$ elements.

The power set of an infinite (either countable or uncountable) set is always uncountable. Moreover, the power set of a set is always strictly "bigger" than the original set in the sense that there is no way to pair the elements of a set *S* with the elements of its power set *P*(*S*) such that every element of *S* set is paired with exactly one element of *P*(*S*), and every element of *P*(*S*) is paired with exactly one element of *S*. (There is never a bijection from *S* onto *P*(*S*).)

Every partition of a set S is a subset of the power set of S.

Cardinality

The cardinality $|S|$ of a set S is "the number of members of S." For example, if $B = \{blue, white, red\}$, $|B| = 3$.

There is a unique set with no members and zero cardinality, which is called the *empty set* (or the *null set*) and is denoted by the symbol $\emptyset$ (other notations are used; see empty set). For example, the set of all three-sided squares has zero members and thus is the empty set. Though it may seem trivial, the empty set, like the number zero, is important in mathematics; indeed, the existence of this set is one of the fundamental concepts of axiomatic set theory.

Some sets have infinite cardinality. The set **N** of natural numbers, for instance, is infinite. Some infinite cardinalities are greater than others. For instance, the set of real numbers has greater cardinality than the set of natural numbers. However, it can be shown that the cardinality of (which is to say, the number of points on) a straight line is the same as the cardinality of any segment of that line, of the entire plane, and indeed of any finite-dimensional Euclidean space.

Special sets

There are some sets that hold great mathematical importance and are referred to with such regularity that they have acquired special names and notational conventions to identify them. One of these is the empty set, denoted {} or $\emptyset$. Another is the unit set {x}, which contains exactly one element, namely x. Many of these sets are represented using blackboard bold or bold typeface. Special sets of numbers include:

- **P** or $\mathbb{P}$, denoting the set of all primes: **P** = {2, 3, 5, 7, 11, 13, 17, ...}.
- **N** or $\mathbb{N}$, denoting the set of all natural numbers: **N** = {1, 2, 3, . . .} (sometimes defined containing 0).
- **Z** or $\mathbb{Z}$, denoting the set of all integers (whether positive, negative or zero): **Z** = {..., −2, −1, 0, 1, 2, ...}.
- **Q** or $\mathbb{Q}$, denoting the set of all rational numbers (that is, the set of all proper and improper fractions): $\mathbf{Q} = \{a/b : a, b \in \mathbf{Z}, b \neq 0\}$. For example, $1/4 \in \mathbf{Q}$ and $11/6 \in \mathbf{Q}$. All integers are in this set since every integer a can be expressed as the fraction $a/1$ ($\mathbf{Z} \subsetneq \mathbf{Q}$).
- **R** or $\mathbb{R}$, denoting the set of all real numbers. This set includes all rational numbers, together with all irrational numbers (that is, numbers that cannot be rewritten as fractions, such as $\sqrt{2}$, as well as transcendental numbers such as π, e and numbers that cannot be defined).
- **C** or $\mathbb{C}$, denoting the set of all complex numbers: $\mathbf{C} = \{a + bi : a, b \in \mathbf{R}\}$. For example, $1 + 2i \in \mathbf{C}$.
- **H** or $\mathbb{H}$, denoting the set of all quaternions: $\mathbf{H} = \{a + bi + cj + dk : a, b, c, d \in \mathbf{R}\}$. For example, $1 + i + 2j - k \in \mathbf{H}$.

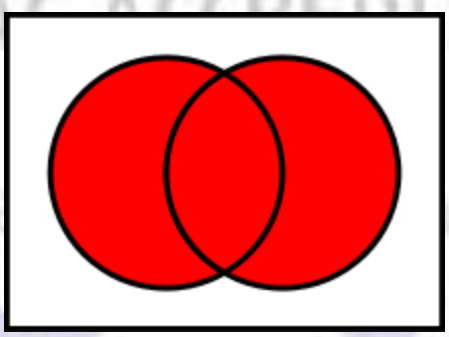
Positive and negative sets are denoted by a superscript - or +, for example: $\mathbb{Q}^+$ represents the set of positive rational numbers.

Each of the above sets of numbers has an infinite number of elements, and each can be considered to be a proper subset of the sets listed below it. The primes are used less frequently than the others outside of number theory and related fields.

Basic operations

There are several fundamental operations for constructing new sets from given sets.

Unions



The **union** of *A* and *B*, denoted $A \cup B$

Two sets can be "added" together. The *union* of *A* and *B*, denoted by $A \cup B$, is the set of all things that are members of either *A* or *B*.

Examples:

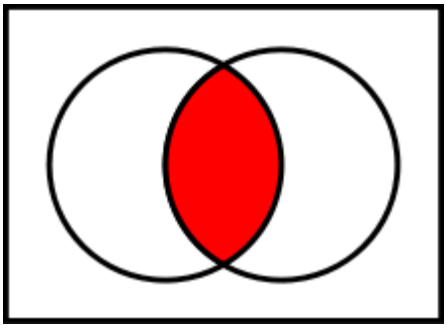
- $\{1, 2\} \cup \{1, 2\} = \{1, 2\}$.
- $\{1, 2\} \cup \{2, 3\} = \{1, 2, 3\}$.

Some basic properties of unions:

- $A \cup B = B \cup A$.
- $A \cup (B \cup C) = (A \cup B) \cup C$.
- $A \subseteq (A \cup B)$.
- $A \cup A = A$.
- $A \cup \emptyset = A$.
- $A \subseteq B$ if and only if $A \cup B = B$.

Intersections

A new set can also be constructed by determining which members two sets have "in common". The *intersection* of *A* and *B*, denoted by $A \cap B$, is the set of all things that are members of both *A* and *B*. If $A \cap B = \emptyset$, then *A* and *B* are said to be *disjoint*.



The **intersection** of A and B, denoted $A \cap B$.

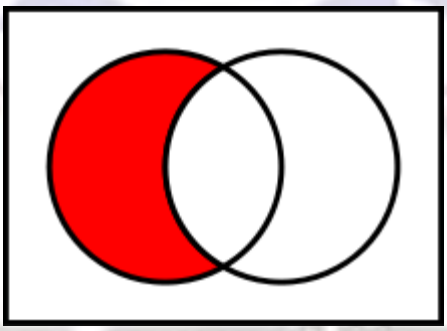
Examples:

- $\{1, 2\} \cap \{1, 2\} = \{1, 2\}$.
- $\{1, 2\} \cap \{2, 3\} = \{2\}$.

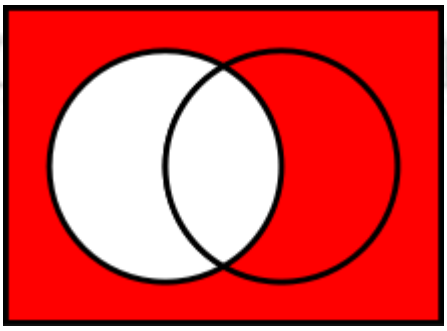
Some basic properties of intersections:

- $A \cap B = B \cap A$.
- $A \cap (B \cap C) = (A \cap B) \cap C$.
- $A \cap B \subseteq A$.
- $A \cap A = A$.
- $A \cap \emptyset = \emptyset$.
- $A \subseteq B$ if and only if $A \cap B = A$.

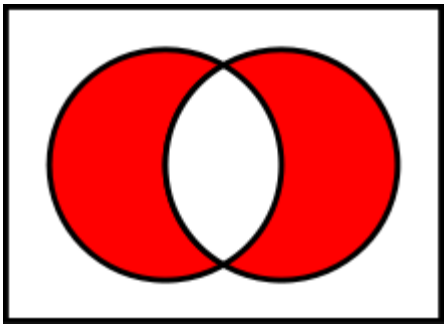
Complements



The **relative complement** of B in A



The **complement** of A in U



The **symmetric difference** of A and B

Two sets can also be "subtracted". The *relative complement* of B in A (also called the *set-theoretic difference* of A and B), denoted by $A \setminus B$ (or $A - B$), is the set of all elements that are members of A but not members of B. Note that it is valid to "subtract" members of a set that are not in the set, such as removing the element *green* from the set {1, 2, 3}; doing so has no effect.

In certain settings all sets under discussion are considered to be subsets of a given universal set U. In such cases, $U \setminus A$ is called the *absolute complement* or simply *complements* of A, and is denoted by A'.

Examples:

- $\{1, 2\} \setminus \{1, 2\} = \emptyset$.
- $\{1, 2, 3, 4\} \setminus \{1, 3\} = \{2, 4\}$.
- If U is the set of integers, E is the set of even integers, and O is the set of odd integers, then $U \setminus E = E' = O$.

Some basic properties of complements:

- $A \setminus B \neq B \setminus A$ for $A \neq B$.
- $A \cup A' = U$.
- $A \cap A' = \emptyset$.
- $(A')' = A$.
- $A \setminus A = \emptyset$.
- $U' = \emptyset$ and $\emptyset' = U$.
- $A \setminus B = A \cap B'$.

An extension of the complement is the symmetric difference, defined for sets A, B as

$$A \Delta B = (A \setminus B) \cup (B \setminus A).$$

For example, the symmetric difference of {7,8,9,10} and {9,10,11,12} is the set {7,8,11,12}.

Cartesian product

A new set can be constructed by associating every element of one set with every element of another set. The *Cartesian product* of two sets A and B, denoted by $A \times B$ is the set of all ordered pairs (a, b) such that a is a member of A and b is a member of B.

Examples:

- $\{1, 2\} \times \{\text{red, white}\} = \{(1, \text{red}), (1, \text{white}), (2, \text{red}), (2, \text{white})\}$.
- $\{1, 2\} \times \{\text{red, white, green}\} = \{(1, \text{red}), (1, \text{white}), (1, \text{green}), (2, \text{red}), (2, \text{white}), (2, \text{green})\}$.
- $\{1, 2\} \times \{1, 2\} = \{(1, 1), (1, 2), (2, 1), (2, 2)\}$.

Some basic properties of cartesian products:

- $A \times \emptyset = \emptyset$.
- $A \times (B \cup C) = (A \times B) \cup (A \times C)$.
- $(A \cup B) \times C = (A \times C) \cup (B \times C)$.

Let A and B be finite sets. Then

- $|A \times B| = |B \times A| = |A| \times |B|$.

2) What are the application of sets , Relations & properties of relations

Ans:- Applications

Set theory is seen as the foundation from which virtually all of mathematics can be derived. For example, structures in abstract algebra, such as groups, fields and rings, are sets closed under one or more operations.

One of the main applications of naive set theory is constructing relations. A relation from a domain A to a codomain B is a subset of the Cartesian product $A \times B$. Given this concept, we are quick to see that the set F of all ordered pairs (x, x^2), where x is real, is quite familiar. It has a domain set **R** and a codomain set that is also **R**, because the set of all squares is subset of the set of all reals. If placed in functional notation, this relation becomes $f(x) = x^2$. The reason these two are equivalent is for any given value, y that the function is defined for, its corresponding ordered pair, (y, y^2) is a member of the set F.

Definition of a relation.

We still have not given a formal definition of a relation between sets X and Y. In fact the above way of thinking about relations is easily formalized, as was suggested in class by Adam Osborne: namely, we canthink of a relation R as a function from X×Y to the two-element set {TRUE, FALSE}. In other words, for (x, y) 2 X×Y,

we say that xRy if and only if f((x, y)) = TRUE.

Properties of relations.

Let X be a set. We now consider various properties that a relation R on X – i.e., R _ X × X may or may not possess.

(R1) Reflexivity: for all x 2 X, (x, x) 2 R.In other words, each element of X bears relation R to itself. Another way tosay this is that the relation R contains the equality relation.Exercise X.X: Go back and decide which of the relations in Examples X.X above are reflexive. For instance, set membership is certainly not necessarily reflexive: 1 62 1 (and in more formal treatments of set theory, a set containing itself is usually explicitly prohibited), but _ is reflexive

.(R2) Symmetry: for all x, y 2 X, if (x, y) 2 R, then (y, x) 2 R.Again, this has a geometric interpretation in terms of symmetry across the diagonal y = x. For instance, the relation associated to the function y = 1/x is symmetricsince interchanging x and y changes nothing, whereas the relation associated to the function y = x2 is not. (Looking ahead a bit, a function y = f(x) is symmetric iff it coincides with its own inverse function.)Exercise X.X: Which of the relations in Examples X.X above are symmetric?

(R3) Anti-Symmetry: for all x, y 2 X, if (x, y) 2 R and (y, x) 2 R, then x = y.
For instance, _ satisfies anti-symmetry.
Exercise X.X: Which of the relations in Examples X.X above are anti-symmetric?

(R4) Transitivity: for all x, y, z 2 X, if (x, y) 2 R and (y, z) 2 R, then (x, z) 2 R.
For instance, “being a parent of” is not transitive, but “being an ancestor of”
is transitive.

Definition: An equivalence relation on a set X is a relation on X which is reflexive, symmetric and transitive.

Examples of equivalence relations.

Let n be a positive integer. Then there is a natural partition of Z into n parts which generalizes the partition into even and odd. Namely, we put Y1 = {. . . ,−2n,−n, 0, n, 2n, . . .} = {kn | k 2 Z} the set of all multiples of n,Y2 = {. . . ,−2n + 1,−n + 1, 1, n + 1, 2n + 1 . . .} = {kn + 1 | k 2 Z}, and similarly, for any 0 _ d _ n − 1, we put Yd = {. . . ,−2n + d,−n + d, d, n + d, 2n + d . . .} = {kn + d | kinZ}. That is, Yd is the set of all integers which, upon division by n, leave a remainder of d. Earlier we showed that the remainder upon division by n is a well-defined integer in the range 0 _ d < n. Here by “well-defined”, I mean that for 0 _ d1 6= d2 < n, the sets Yd1 and Yd2 are disjoint. Recall why this is true: if not, there exist k1, k2 such that k1n + d1 = k2n + d2, so d1 − d2 = (k2 − k1)n, so d1 − d2 is a multiple of n. But −n < d1 − d2 < n, so the only multiple of n it could possibly be is 0, i.e., d1 = d2. It is clear that each Yd is nonempty and that their union is all of Z, so {Yd}n−1 d=0 gives a partition of Z. The corresponding equivalence relation is called congruence modulo n, and written as follows: x _ y (mod n). What this means is that x and y leave the same remainder upon division by n.

3) Define Partial Order sets

Ans:- A (non-strict) **partial order** is a binary relation "≤" over a set P which is reflexive, anti symmetric, and transitive, i.e., which satisfies for all a, b, and c in P:

- $a \leq a$ (reflexivity);
- if $a \leq b$ and $b \leq a$ then $a = b$ (anti symmetry);
- if $a \leq b$ and $b \leq c$ then $a \leq c$ (transitivity).

In other words, a partial order is an anti symmetric preorder.

A set with a partial order is called a **partially ordered set** (also called a **poset**). The term *ordered set* is sometimes also used for posets, as long as it is clear from the context that no other kinds of orders are meant. In particular, totally ordered sets can also be referred to as "ordered sets", especially in areas where these structures are more common than posets.

For a, b, elements of a partially ordered set P, if $a \leq b$ or $b \leq a$, then a and b are **comparable**. Otherwise they are **incomparable**. In the figure on top-right, e.g. {x} and {x,y,z} are comparable, while {x} and {y} are not. A partial order under which every pair of elements is comparable is called a **total order** or **linear order**; a totally ordered set is also called a **chain** (e.g., the natural numbers with their standard order). A subset of a poset in which no two distinct elements are comparable is called an **antichain** (e.g. the set of singletons {{x}, {y}, {z}} in the top-right figure). An element a is said to be **covered** by another element b, written $a<:b$, if a is strictly less than b and no third element c fits between them; formally: if both $a \leq b$ and $a \neq b$ are true, and $a \leq c \leq b$ is false for each c with $a \neq c \neq b$. A more concise definition will be given below using the strict order corresponding to "≤". For example, {x} is covered by {x,z} in the top-right figure, but not by {x,y,z}.

Standard examples of posets arising in mathematics include:

- The real numbers ordered by the standard *less-than-or-equal* relation ≤ (a totally ordered set as well).
- The set of subsets of a given set (its power set) ordered by inclusion (see the figure on top-right). Similarly, the set of sequences ordered by subsequence, and the set of strings ordered by substring.
- The set of natural numbers equipped with the relation of divisibility.
- The vertex set of a directed acyclic graph ordered by reach ability.
- The set of subspaces of a vector space ordered by inclusion.
- For a partially ordered set P, the sequence space containing all sequences of elements from P, where sequence a precedes sequence b if every item in a precedes the corresponding item in b. Formally, $(a_n)_{n \in \mathbb{N}} \leq (b_n)_{n \in \mathbb{N}}$ if and only if $a_n \leq b_n$ for all n in $\mathbb{N}$.
- For a set X and a partially ordered set P, the function space containing all functions from X to P, where $f \leq g$ if and only if $f(x) \leq g(x)$ for all x in X.

- A fence, a partially ordered set defined by an alternating sequence of order relations $a < b > c < d$ ...

4) What is Functions , explain its types & domain & range

Ans:- Function

Consider the relation
f : {(a, 1), (b, 2), (c, 3), (d, 5)}
In this relation we see that each element of A has a unique image in *B* This relation *f* from set *A* to *B* where every element of *A* has a unique image in *B* is defined as a function from *A* to *B*. So we observe that ***in a function no two ordered pairs have the same first element.***

Domain and Range:-
We also see that $ an element Î B, i.e., 4 which does not have its preimage in A. Thus here:
(i) the set B will be termed as co-domain and
(ii) the set {1, 2, 3, 5} is called the range.
From the above we can conclude that ***range is a subset of co-domain.*** Symbolically, this function can be written as
f : A ® B or A ¾¾f ¾® B

Example

Which of the following relations are functions from A to B. Write their domain and range. If it is not a function give reason?
(a) { (1, -2),(3,7),(4, -6),(8,1) } , A = {1,3,4,8} , B = {-2,7, -6,1,2}
(b) { (1,0),(1 - 1),(2,3),(4,10) }, A = {1,2,4} , B = {0, -1,3,10}
(c) { (a,b),(b,c),(c,b),(d,c)} , A = { a,b,c,d,e} B = {b,c}
(d) { (2,4),(3,9),(4,16),(5,25),(6,36 }, A = { 2,3,4,5,6} , B = {4,9,16,25,36 }
(e) { (1, -1),(2, -2),(3, -3),(4, -4),(5, -5)} ,A = { 0,1,2,3,4,5} ,
B = {-1, -2, -3, -4, -5}

Solution :
(a) It is a function.
Domain= {1,3,4,8} , Range = {-2,7, -6,1}
(b) It is not a function. Because Ist two ordered pairs have same first elements.
(c) It is not a function.
Domain= {a,b,c,d} ¹ A, Range = { b, c}
(d) It is a function.
Domain = {2,3,4,5,6} , Range = {4,9,16,25,36 }
(e) It is not a function .
Domain = {1,2,3,4,5} ¹A , Range = {-1, -2, -3, -4, -5}

Types of functions :-

One-to-one:- Let *f* be a function from *A* to *B*. If every element of the set *B* is the image of at least one element of the set *A* i.e. if there is no unpaired element in the set *B* then we say that the function f maps the set *A* onto the set *B*. Otherwise we say that the function maps the set *A* into the set *B*. Functions for which each element of the set *A* is mapped to a different element of the set *B* are said to be ***one-to-one.***

Many-to-one.:- A function can map more than one element of the set A to the same element of the set B. Such a type of function is said to be ***many-to-one.***

Reciprocal Function/Inverse function:-
Functions of the type y = 1/x, x≠0, called a reciprocal function.

Composite function:- Consider the two functions given below:
y = 2x + 1, x Î{1,2,3}
z = y + 1, y Î{3,5,7}
Then *z* is the composition of two functions *x* and *y* because *z* is defined in terms of *y* and *y* in terms of *x*.
Graphically one can represent this as given below :

5) What is Hassing Function ? Explain its methods

Ans:- HASHING FUNCTION

To save space and time, each record stored in a computer is assigned an address (memory location) in the computer's memory. The task of assigning the address is performed by the Hashing function (or Hash function) $H : K \rightarrow L$, which maps the set K of keys to the set L of memory addresses. Thus a Hashing function provides means of identifying records stored in a table. The function H should be one-to-one. In fact, if $k_1 \neq k_2$ implies $H(k_1) = H(k_2)$, then two keys will have same address and we say that *collision* occurs. To resolve collisions, the following methods are used to define the hash function.

1. **Division Method.** In this method, we restrict the number of addresses to a fixed number (generally a prime) say m and define the hash function $H : K \rightarrow L$ by

$$H(k) = k \pmod{m},\ k \in K,$$

   where k (mod m) denotes the remainder when k is divided by m.
2. **Midsquare Method.** As the name suggest, we square the key in this method and define hash function $H : K \rightarrow L$ by $H(k) = l$, where l is obtained by deleting digits from both ends of k^2.
3. **Folding Method.** In this method the key k is partitioned into a number of parts, where each part, except possibly the last, has the same numbers of digits as the required memory address. Thus, if $k = k_1 + k_2 + \ldots + k_n$, then the hash function $H : K \rightarrow L$ is defined by

$H(k) = k_1 + k_2 + \ldots + k_n$, where the last carry has been ignored.

6) What is Partial Order Relations on a Lattice

Ans: Partial Order Relations on a Lattice:-

A partial order relation on a lattice (L) follows as a consequence of the axioms for the binary operations $\vee$ and $\wedge$.

PARTIALLY ORDERED SETS

A relation R on a set X is said to be anti symmetric if a R b and b R a imply a = b. Relation R on a set X is called a partial order relation if it is reflexive, anti-symmetric and transitive. A set X with the partial order R is called a partially ordered set or poset and is denoted by (X, R)

EXAMPLE

Let $\tilde{A}$ be a collection of subsets of a set S. The relation $\subseteq$ of set inclusion is a partial order relation on $\tilde{A}$. In fact, if A, B, C $\in \tilde{A}$, then,

$A \subseteq A$, that is, A is a subset of itself which is true.

If $A \subseteq B$, $B \subseteq A$, then A = B

If $A \subseteq B$, $B \subseteq C$, then A is a subset of C, that is, $A \subseteq C$.

7) What is hasse diagram ? explain with the help of example.

Ans:- **HASSE DIAGRAM**

Let A be a finite set. By the theorem proved above, the digraph of a partial order on A has only cycles of length 1. In fact, since a partial order is reflexive, every vertex in the digraph of the partial order is contained in a cycle of length 1. To simplify the matter, we shall delete all such cycles from the digraph.

We also eliminate all edges that are implied by transitivity property. Thus, if $a \leq b$, $b \leq c$, it follows that $a \leq c$. In this case, we omit the edge form a to c. We also agree to draw the digraph of partial order with all edges pointing upward, omit the arrows and to replace then the circles by dots

"The diagram of a partial order obtained from its digraph by omitting cycles of length 1, the edges implied by transitivity and the arrows (after arranging them pointing upward) is called Hasse diagram of the partial order of the poset".

EXAMPLE

Let A = {1, 2, 3, 4, 12}. Consider the partial order of divisibility on A. That is, if a and b are in A, $a \leq b$ if and only if a | b

Therefore, the Hasse diagram of the poset (A, $\leq$) is as shown in Figure 1.18.

EXAMPLE

Let S = {a, b, c} and Ã = P(S) (power set of S).

Consider the partial order of set inclusion (⊆). We note that

Ã = P(S) = { ϕ, {a}, {b}, {c}, {a, b}, {a, c}, {b, c}, {a, b, c}}

Then the Hasse diagram of the poset (Ã, ⊆)

Hasse diagram of a finite linearly ordered set is always of the form and thus consists of simply one path. Hence diagram of a totally order set is a chain.

Hasse Diagram of Dual Poset

If (A, ≤) is a poset and (A, ≥) is the dual poset, then the Hasse diagram of (A, ≥) is just the Hasse diagram of (A, ≤) turned upside down.

For example, let A = {a, b, c, d, e, f} and let be the Hasse diagram of poset (A, ≤). Then the Hasse diagram of dual poset (A, ≥) is which can be constructed by turning the Hasse diagram of (A, ≤) upside down.

EXAMPLE Let A = {a, b, c, d, e}. Then the Hasse diagram defines a partial order on B in the natural way. That is, d ≤ b, d ≤ a, e ≤ c and so on.

EXAMPLE Let n be a positive integer and Dn denote the set of all divisors of n. Considering the partial order of divisibility in Dn, draw Hasse diagram D24, D30 and D36.

Solution.

We know that

D24 = {1, 2, 3, 4, 6, 8, 12, 24},

D30 = {1, 2, 3, 5, 6, 10, 15, 30},

D36 = {1, 2, 3, 4, 6, 9, 12, 18, 36}.

Therefore, the Hasse diagram of D24, D30 and D36

{5}, {3, 2}, {2, 2, 1}, {1, 1, 1, 1, 1}, {4, 1}, {3, 1, 1}, {2, 1, 1, 1}.

We order the partitions of an integer m as follows:

A partition P1 precedes a partition P2 if the integers in P1 can be added to obtain integers in P2 or we can say that if the integers in P2 can be further subdivided to obtain the integers in P1. For example, {1, 1, 1, 1, 1} precedes {2, 1, 1, 1}. On the other hand, {3, 1, 1} and {2, 2, 1} are non-comparable.

The Hasse diagram of the partition of m = 5 is

Let A be a (non-empty) linearly ordered alphabet. Then Kleene closure of A consists of all words w on A and is denoted by A*.

Also then |w| denotes the length of w.

We have following two order relations on A*.

Alphabetical (Lexicographical) order: In this order we have

$\lambda < w$, where λ is empty word and w is any non-empty word.

Suppose $u = a\,u'$ and $v = b\,v'$ are distinct non-empty words where $a, b \in A$ and $u', v' \in A^*$. Then,

$u < v$ if $a < b$ or if $a = b$ but $u' < v'$ Short-lex order: Here A* is ordered first by length and then alphabetically, that is, for any distinct words u, v, in A*, $u < v$ if $|u| < |v|$ or if $|u| = |v|$ but u precedes v alphabetically. For example, “to” proceeds “and” since |to| = 2 but |and| = 3. Similarly, “an” precedes “to” since they have the same length but “an” precedes “to” alphabetically.

This order is also called free semi-group order.

Let A be a partially ordered set with respect to a relation ≤. An element a in A is called a maximal element of A if and only if for all b in A, either $b \leq a$ or b and a are not comparable.

An element a in A is called greatest element of A if and only if for all b in A, $b \leq a$.

An element a in A is called minimal element of A if and only if for all b in A, either $a \leq b$ or b and a are not comparable.

An element a in A is called a least element of A if and only if for all b in A, $a \leq b$.

A greatest element is maximal but a maximal element need not be greatest element. Similarly, a least element is minimal but a minimal element need not be a least element.

The elements a1, a2 and a3 are maximal elements of A, and the elements b1, b2 and b3 are minimal elements. Observe that since there is no line between b2 and b3 we can conclude that neither b3 ≤ b2 nor b2 ≤ b3 showing that b2 and b3 are not comparable.

Let A be the poset of non-negative real numbers with usual partial order ≤ (read as " less than or equal to "). Then 0 is the minimal element of A. There is no maximal element of A.

Let A be a finite non-empty poset with partial order ≤. Then A has at least one maximal element and at

.

Let (A, ≤) be a poset and B a subset of A. An element a ∈ A is called a least upper bound (supremum) of B if

a is an upper bound of B, that is, b ≤ a ∀ b ∈ B

a ≤ a′ whenever a′ is an upper bound of B.

An element a ∈ A is called a greatest lower bound (infimum) of B if

a is a lower bound of B, that is, a ≤ b ∀ b ∈ B

a′ ≤ a whenever a′ is a lower bound of B.

Further, upper bounds in the poset (A, ≤) correspond to lower bounds in the dual poset (A, ≥) and the lower bounds in (A, ≤) correspond to upper bound in (A, ≥).

Similar statements also hold for greatest lower bounds and least upper bounds.

Consider Example 1.69 above:

Since B1 = {a, b} has no lower bound, it has no greatest lower bound. However,

lub (B1) = c

Since the lower bounds of B2 = {c, d, e} are c, a and b, we have

glb (B2) = c

8) Discuss lattice & its properties.

Ans:- **Definition**

A **lattice** is a partially ordered set $(L, \leq)$ in which every subset $\{a, b\}$ consisting of two elements has a least upper bound and a greatest lower bound.

We denote LUB($\{a, b\}$) by $a \vee b$ and call it **join** or **sum of** a **and** b. Similarly, we denote GLB ($\{a, b\}$) by $a \wedge b$ and call it **meet** or **product of** a **and** b.

Other symbols used are

LUB: $\oplus$, +, $\cup$,
GLB: *, ·, $\cap$.

Thus **Lattice is** a mathematical structure with **two binary operations, join and meet.**

A totally ordered set is obviously a lattice but not all partially ordered sets are lattices.

EXAMPLE Let A be any set and $P(A)$ be its power set. The partially ordered set $(P(A), \subseteq)$ is a lattice in which the meet and join are the same as the operations $\cap$ and $\cup$, respectively. If A has single element, say a, then $P(A)=\{\phi, \{a\}\}$

PROPERTIES OF LATTICES

Let $(L, \leq)$ be a lattice and let $a, b, c \in L$. Then, from the definition of $\vee$ (join) and $\wedge$(meet) we have

i. $a \leq a \vee b$ and $b \leq a \vee b$; $a \vee b$ is an upper bound of a and b.
ii. If $a \leq c$ and $b \leq c$, then $a \vee b \leq c$; $a \vee b$ is the least upper bound of a and b.
iii. $a \wedge b \leq a$ and $a \wedge b \leq b$; $a \wedge b$ is a lower bound of a and b.
iv. If $c \leq a$ and $c \leq b$, then $c \leq a \wedge b$; $a \wedge b$ is the greatest lower bound of a and b.

Theorem

Let L be a lattice. Then for every a and b in L,

i. $a \vee b = b$ if and only if $a \leq b$,
ii. $a \vee b = a$ if and only if $a \leq b$,
iii. $a \wedge b = a$ if and only if $a \vee b = b$.

I. BOUNDED, COMPLEMENTED AND DISTRIBUTIVE LATTICES

ii. Let $(L, \vee, \wedge)$ be a lattice and let $S = \{a_1, a_2, \ldots, a_n\}$ be a finite subset of L. Then,
iii.
iv. LUB of S is represented by $a_1 \vee a_2 \vee \ldots \vee a_n$,
GLB of S is represented by $a_1 \wedge a_2 \wedge \ldots \wedge a_n$.
v. **Definition** A lattice is called **complete** if each of its non-empty subsets has a least upper bound and a greatest lower bound.
vi. Obviously, every finite lattice is complete.

vii. Also, every complete lattice must have a least element, denoted by 0 and a greatest element, denoted by I.

viii. The least and greatest elements if exist are called **bound (units, universal bounds)** of the lattice.

ix. **Definition** A lattice L is said to be **bounded** if it has a greatest element I and a least element 0.

x. For the lattice $(L, \vee, \wedge)$ with $L = \{a_1, a_2, \ldots, a_n\}$,

9) Discuss **DEFINITIONS AND BASIC CONCEPTS OF GRAPH**

ANS:- Definition

A **graph $G = (V, E)$** is a mathematical structure consisting of two finite sets V and E. The elements of V are called **vertices (or nodes)** and the elements of E are called **edges**. Each edge is associated with a set consisting of **either one** or **two vertices** called its **endpoints**.

The correspondence from edges to endpoints is called **edge-endpoint function**. This function is generally denoted by γ. Due to this function, some authors denote graph by $G = (V, E, \gamma)$.

Definition

A graph consisting of one vertex and no edges is called a **trivial graph**.

Definition

A graph whose vertex and edge sets are empty is called a **null graph**.

Definition

An edge with just one endpoint is called a **loop** or a **self-loop**.

SPECIAL GRAPHS

Definition

A graph G is said to **simple** if it has no parallel edges or loops. In a simple graph, an edge with endpoints v and w is denoted by $\{v, w\}$.

Definition

For each integer $n \geq 1$, let D_n denote the graph with n **vertices** and **no edges**. Then D_n is called the **discrete graph on n vertices**.

Definition

Let $n \geq 1$ be an integer. Then a simple graph with n vertices in which there is an edge between each pair of distinct vertices is called the **complete graph** on n vertices. It is denoted by K_n.

For example, the complete graphs K_2, K_3 and K_4 are shown in

10) Discuss sub graph, proper sub graph & isomorphic graph also

Ans:-

SUBGRAPHS

Definition

A graph H is said to be a subgraph of a graph G if and only if every vertex in H is also a vertex in G, every edge in H is also an edge in G and every edge in H has the same endpoints as in G.

We may also say that *G* is a super graph of *H*
Definition

A sub graph H is said to be a **proper sub graph** of a graph G if vertex set V_H of H is a proper subset of the vertex set V_G of G or edge set E_H is a proper subset of the edge set E_G.

For example, the sub graphs in the above examples are proper sub graphs of the given graphs.

ISOMORPHISMS OF GRAPHS

We know that shape or length of an edge and its position in space are not part of specification of a graph. For example, the represent the same graph.

Definition

Let G and H be graphs with vertex sets $V(G)$ and $V(H)$ and edge sets $E(G)$ and $E(H)$, respectively. Then G **is said to isomorphic to** H if there exist one-to-one correspondences g: $V(G) \rightarrow V(H)$ and h: $E(G) \rightarrow E(H)$ such that for all $v \in V(G)$ and $e \in E(G)$,

$$v \text{ is an endpoint of } e \Leftrightarrow g(v) \text{ is an endpoint of } h(e).$$

Definition

The property of mapping endpoints to endpoints is called **preserving incidence** or **the continuity rule** for graph mappings.

As a consequence of this property, a self-loop must map to a self-loop.

Thus, two isomorphic graphs are same except for the labelling of their vertices and edges.

11) DISCUSS WALKS, PATHS IN THE GRAPH

ANS:- **Definition**

In a graph G, **a walk from vertex** v_0 **to vertex** v_n is a finite alternating sequence $\{v_0, e_1, v_1, e_2, \ldots, v_{n-1}, e_n, v_n\}$ of vertices and edges such that v_{i-1} and v_i are the endpoints of e_i.

The **trivial walk** from a vertex v to v consists of the single vertex v.

Definition

In a graph G, a **path** from the vertex v_0 to the vertex v_n is a walk from v_0 to v_n that does not contain a repeated edge.

Thus a **path** from v_0 to v_n is a walk of the form

$$\{v_0, e_1, v_1, e_2, v_2, \ldots, v_{n-1}, e_n, v_n\},$$

where all the edges e_i are distinct.

Definition

In a graph, a **simple path** from v_0 to v_n is a path that does not contain a repeated vertex.

Thus a simple path is a walk of the form

$$\{v_0, e_1, v_1, e_2, v_2, \ldots, v_{i-1}, e_n, v_n\},$$

12) DISCUSS HAMILTONIAN CIRCUITS & MATRIX REPRESENTATION OF GRAPHS

ANS:- **Definition**

A **Hamiltonian path** for a graph G is a sequence of adjacent vertices and distinct edges in which every vertex of G appears exactly once.

Definition

A **Hamiltonian circuit** for a graph G is a sequence of adjacent vertices and distinct edges in which every vertex of G appears exactly once, except for the first and the last which are the same.

Definition

A graph is called **Hamiltonian** if it admits a Hamiltonian circuit.

MATRIX REPRESENTATION OF GRAPHS

A graph can be represented inside a computer by using the adjacency matrix or the incidence matrix of the graph.

Definition

Let G be a graph with n ordered vertices $v_1, v_2, \ldots, v_n$. Then the **adjacency matrix of** G is the $n \times n$ matrix $A(G) = (a_{ij})$ over the set of non-negative integers such that

$$a_{ij} = \text{the number of edges connecting } v_i \text{ and } v_j \text{ for all } i, j = 1, 2, \ldots, n.$$

We note that if G has no loop, then there is no edge joining v_i to v_i, $i = 1, 2, \ldots, n$.Therefore, in this case, all the entries on the main diagonal will be 0.

Further, if G has no parallel edge, then the entries of $A(G)$ are either 0 or 1.

It may be noted that adjacent matrix of a graph is symmetric.

Conversely, given a $n \times n$ symmetric matrix $A(G) = (a_{ij})$ over the set of non-negative integers, we can associate with it a graph G, whose adjacency matrix is $A(G)$, by letting G have n vertices and joining v_i to vertex v_j by a_{ij} edges

13) EXPLAIN COLOURING OF GRAPH

ANS:- Definition

Let G be a graph. The assignment of colours to the vertices of G, one colour to each vertex, so that the adjacent vertices are assigned different colours is called **vertex colouring** or **colouring of the graph** G.

Definition

A graph G is n-**colourable** if there exists a colouring of G which uses n colours.

Definition

The minimum number of colours required to paint (colour) a graph G is called the **chromatic number of** G and is denoted by $\chi(G)$.

14) Discuss Propositional Logic

Ans:- Logic is essentially the study of arguments. For example, someone may say, suppose A and B are true, can we conclude that C is true? Logic provides rules by which we can conclude that certain things are true given other things are true. Here is a simple example: A tells B that "if it rains, then the grass will get wet. It is raining"; B can then conclude that the grass is wet, if what A has told B is true. Logic provides a mechanism for showing arguments like this to be true or false.

The section starts by showing how to translate English sentences into a logical form, specifically into something called "propositions". In fact, our study of logic starts with *propositional calculus*.

Propositional calculus is the calculus of propositions and we plan to study propositions. Most of us may associate the term calculus with integrals and derivatives, but if we check out the definition of calculus in the dictionary, we will see that calculus just means "a way of calculating", so differential calculus, for instance, is how to calculate with derivatives and integral calculus is how to calculate with integrals.

However, before going into how to translate English sentences into propositions, we are going to introduce Boolean expressions (that is, propositions), and then discuss about translation. Hence, we will see the same ideas in two different forms.

15) EXPLAIN ABOUT BOOLEAN EXPRESSIONS

ANS:- Definition A Boolean variable is a variable that can either be assigned true or false. We have programmed in C++ and know about types such as integers, floats, and character pointers. However, C++ also has a Boolean type, as do Java and Pascal. We can declare variables to be of Boolean type, which means that they can only take on two values: true and false. Throughout the chapter, we shall generally use the letters, p, q, and r as Boolean variables. However, in some cases we will allow these letters to have subscripts. For example, p_0, q_{1492} and r_{1776} are all Boolean variables.

Definition A Boolean expression is either

1. a Boolean variable, or
2. it has the form $\neg\phi$, where ϕ is a Boolean expression, or
3. it has the form $(\phi * \psi)$, where ϕ and ψ are Boolean expressions and * is one of the following:

   $\wedge$, $\vee$, $\rightarrow$, or $\leftrightarrow$.

CONSTRUCTION OF BOOLEAN EXPRESSIONS

Suppose we are given a Boolean expression and asked to prove that it is a Boolean expression. How do we proceed? There are two different ways of doing it. The first is to build a Boolean expression from its constituent parts. Let us start off with an example. We want to show that $((p \wedge q) \vee \neg r)$ is a Boolean expression. To do so, we will take a bottom-up approach.

Expression	*Reason*
1. p	Boolean variable
2. q	Boolean variable
3. r	Boolean variable
4. $\neg r$	3, $\neg\phi$
5. $(p \wedge q)$	1, 2, $(\phi \wedge \psi)$

6. $((p \wedge q) \vee \neg r)$ 5, 4, $(\phi \wedge \psi)$

Notice that we start with the smallest Boolean expression (namely, Boolean variables) and work our way up. Look at line number 4. The reason is "3, $\neg\phi$". This means that we are using the rule $\neg\phi$ to create line 4, where ϕ is from line 3. This is just the second part of the definition being applied. And we use lines 1 and 2 and rule $(\phi \wedge \psi)$ to create the expression in line 5. Again, this rule comes from part 3 of the definition.

Definition A construction of a Boolean expression is a list of steps, where each line is either a Boolean variable or it uses a connective (e.g., $\neg$, $\wedge$, $\vee$, $\rightarrow$ or $\leftrightarrow$) to connect two other Boolean expressions (they may be the same), with line numbers that are less than itself. Each line is a valid Boolean expression.

For example, look at the construction above. If we have to add a 7^{th} step to the construction, we would have two choices. Either we could introduce a Boolean variable (we could always do this) or we could use a connective and find a Boolean expression that is already on the list and add it. For example, we could place a $\neg$ in front of $\neg r$ (from step 4) and produce $\neg\neg r$.

The point of this exercise was to explain how to convince someone else that $((p \wedge q) \vee \neg r)$ is a Boolean expression. It is a kind of proof and uses the definition of Boolean expressions as reasons or justifications for each step.

The only difficulty with using this (and it is a small one) is that it is sometimes easier seeing an expression top-down than bottom-up. That is, it is intuitively simpler to take a complicated expression like $((p \wedge q) \vee \neg r)$ and try to break it down to its two parts, $(p \wedge q)$ and $\neg r$.

MEANING OF BOOLEAN EXPRESSIONS

One use of logic is as a means of deciding what things are true, given that certain facts are already true. Logic provides us a framework for deducing new things that are true. However, this deduction is based on form. For example, we might say that either $x > 0$ or $x \leq 0$ and also that x is not greater than 0. Given these two facts, we should be able to conclude that $x \leq 0$.

Now both arguments actually have the same form that is, we basically said ϕ or ψ is true and then ϕ is not true, therefore we concluded ψ is true. In the first example, ϕ was $x > 0$ and ϕ was $x \leq 0$, while in the second example ϕ was "the capital of India is Mumbai" and ψ was "the capital of India is New Delhi". In both examples, there was a similar form of the argument and the conclusion that we drew was purely based on the form.

This is actually at the heart of logic. (Some) English arguments can be translated into Boolean expressions, and then we can apply rules of logic to determine whether the arguments make sense, atleast, based on their form.

We know that Boolean variables are the building blocks of Boolean expressions. Boolean variables like p generally stand for either English or mathematical propositions.

16) WHAT IS PREPOSITION OR STATEMENT

ANS:- Definition A proposition is something that is either true or false but not both.

Not all English sentences are propositions. For example, the sentence "Run away" cannot really be said to be true nor false. Not all mathematical "sentences" are propositions either. For example, $x > y$ is neither true nor false. We would need to know the values of x and y before we could draw the conclusion. Actually, we do not have to be this strict, $x > y$ is either true or false, so in some sense, we can consider it a proposition.

Once the translation has been made from English sentences or mathematical sentences into Boolean expressions, then we generally do not care what the original sentence means. We can make conclusions based on the Boolean expressions. We shall get into the details soon.

17) EXPLAIN CONJUNCTION & DISJUNCTION IN PREPOSITION

ANS:- Conjunctions

The most basic Boolean expression is a Boolean variable, which is either true or false. Throughout this section, we shall refer to two propositions: p and q.

p I own a cat.

q I own a dog.

One way to make a more complicated sentence is to connect two sentences with "and". For example, "I own a cat" AND "I own a dog". In propositional calculus (which is what we are studying now), our purpose is to determine when expressions or sentences are true. So, when is the entire expression "I own a cat AND I own a dog" true? Intuitively, we would say it is true if both parts are true.

Now let's look at the Boolean expression equivalent of that same sentence. It happens to be $(p \wedge q)$ (again, notice the use of parentheses). The symbol for AND is $\wedge$, which we can pronounce as AND. If it helps us to remember, the $\wedge$ symbol looks sort of like an "A", which is the first letter of AND. Sometimes, mathematicians say that $(p \wedge q)$ is a *conjunction* of p and q and that p and q are conjuncts of the conjunction. Despite the fancy name, the work "conjunction" does come up often enough and hence we ought to remember it.

So, when is $(p \wedge q)$ true? When both p and q are true. If either is false, then the whole expression is false. We can actually summarize this in a *truth table*. A truth table tells us the "truth" of a Boolean expression given that we assign either true or false to each of the Boolean variables.

Given two different Boolean variables, p and q, there are four different ways to assign truth values to them. Each of the four ways is listed below. T stands for true, while F stands for false. If we look at the column for variable q, we will see that it alternates T, then F, then T, then F, whereas the column with p to its immediate left alternates, T, T, then F, F. If we had another variable, r and placed it to the left of p, it would alternate T, T, T, T then F, F, F, F.

There is a pattern. Starting from the rightmost Boolean variable, we will alternate every turn T, F, T, F. The next column to its left will alternate T, T, F, F, T, T, F, F, etc. The next one to its left will alternate T, T, T, T, F, F, F, F. As we move to the left, we repeat the Ts

twice as many times as the previous column and twice as may Fs. This pattern actually covers all possible ways of combining truth values for *n* Boolean variables.

Now, look at line 1 in the truth table. Look at the last column. We will notice that the entry has the value T, which means that when *p* is assigned T and *q* is assigned T, then $(p \wedge q)$ is true as well. This is just a formalization of what we said before, $(p \wedge q)$ is only true when *p* and *q* are both true (that is, both assigned to true).

The key point is to notice that we can find out the truth value of a complicated expression by knowing the truth value of the parts that make it up. This is really no different from arithmetic expressions. For example, if we had the expression $(x + y) - z$, then we could tell that the value for this expression, provided if we knew the values for each of the variables. It is the same with Boolean expressions. If we know the truth values of the Boolean variables, then using truth tables, we can determine the truth value of a Boolean expression.

Now, we used *p* and *q* for the truth table above. However, there was nothing special about using those two Boolean variables. Any two different Boolean variables would have worked. In fact, any two Boolean *expressions* would have worked. We could have replaced *p* with ϕ and *q* with ψ and $(p \wedge q)$ with $(\phi \wedge \psi)$ and the truth table would still have been fine.

Disjunctions

Instead of saying "I own a cat" AND "I own a dog", we could connect the two statements with OR, as in, "I own a cat" OR "I own a dog". When would this statement be true? It would be true if I owned either a cat or a dog. That is, only one of the two statements has to be true. What happens if both are true? Then is the *entire* statement "I own a cat OR I own a dog" true? Going by propositional logic, we will say yes. That is, the entire OR statement is true if one or the other statement or both are true.

We use the symbol $\vee$ to represent OR. So $(p \vee q)$ (again, notice the parentheses) is the same as *p* OR *q* and the whole expression is true if *p* is true or *q* is true or both are true. It is false if both *p* and *q* are false. Sometimes the expression $(p \vee q)$ is called a *disjunction* (with a $\wedge$, it was a conjunction) and *p* and *q* are the disjuncts of the disjunction. Any Boolean expression (or subexpression) can be called a disjunction if it has the pattern $(\phi \vee \psi)$.

The use of "OR" in propositional calculus actually contrasts with the way we normally use it in English. For example, if A said, "I will go to the Cinema OR I will go to the Garden". Usually it means, I will go to one or the other, but NOT both. This kind of "OR" is called as *exclusive* OR, while the one we use in propositional calculus is called an *inclusive* OR. We will almost always use the inclusive OR (the exclusive OR can be defined using inclusive ORs and negations, which will be introduced in the next section).

Here is the truth table for OR.

Notice the rightmost column of this truth table and compare it to the rightmost column of the truth table of $(p \wedge q)$. In the case of conjunction (i.e., AND), $(p \wedge q)$ is true in only one case, namely, when *p* and *q* are both true. It is false in all other cases. However, for $(p \vee q)$ (read *p* OR *q*), it is true in all cases except when both *p* and *q* are false. In other words, only one of either *p* or *q* has to be true for the entire expression $(p \vee q)$ to be true.

The main point covered so far:

The symbols we have seen: {∧, ∨, →, ↔, ¬} are often called *connectives* because they connect two Boolean expressions (although in the case of ¬, it's only attached to a single Boolean expression).

18) EXPLAIN NEGATION

ANS:- 3 Negations

The symbol ¬, (pronounced "not") is like a negative sign in arithmetic. So, if we have *p* ("I own a cat"), the ¬*p* (notice there are *no* parentheses) can be read as "Not I own a cat", or "It is not the case that I own a cat" (in which case the ¬ could be read as "it is not the case that"). Both of these sound awkward, but the idea is to use the original sentence and attach something before it, just like the connective. In English, it sounds more correct to say "I do not own a cat".

Unlike ∧ and ∨, ¬ only attaches to a single Boolean expression. So, the truth table is actually smaller for ¬ since there is only one Boolean variable to worry about.

Line	*p*	¬*p*
1	T	F
2	F	T

This should be an easy truth table to understand. If *p* is true, then ¬*p* is false. The reverse holds as well. If *p* is false, then ¬*p* is true.

19) EXPLAIN ABOUT TRUTH TABLE

ANS:- CONSTRUCTION OF TRUTH TABLES

Suppose we are given a Boolean expression, say, ((*p* ∧ *q*) ∨ ¬*p*). We want to know whether this expression is true or false.

We introduce a function, *v*. This function will be called a *valuation*. If we were to write this function signature in pseudo-C++ code, it would look like

boolean v(boolExpr x);

In words, this function takes a Boolean expression as input (think of it as a class) and returns back a Boolean value, that is, it returns either true or false.

Let us formally define a valuation.

Definition A valuation (also called, a truth value function or a truth value assignment) is a function which assigns a truth value (that is, true or false) for a Boolean expression, under the following restrictions.

$$v((\phi \wedge \psi)) = \min(v(\phi), v(\psi))$$
$$v((\phi \vee \psi)) = \max(v(\phi), v(\psi))$$
$$v(\neg\phi) = 0, \text{ if } v(\phi) = 1$$

	= 1, if $v(\phi) = 0$
$v((\phi \rightarrow \psi))$	= 1, if $v(\phi) = F$ or $v(\psi) = T$
	= 0, otherwise
$v((\phi \leftarrow \psi))$	= 1, if $v(\phi) = v(\psi)$
	= 0, otherwise

The interesting thing is that because of these restrictions, once a truth value function has been defined for all the Boolean variables in a Boolean expression, the truth value for the Boolean expression (and all its subexpressions) are defined as well.

Let us take a closer look at the definition. We shall take it line by line. In the first line, we have

$$v((\phi \wedge \psi)) = \min(v(\phi), v(\psi))$$

This says that if we want to find the truth value of $(\phi \wedge \psi)$, then we have to find the truth value of ϕ (that is, $v(\phi)$) and the truth value of ψ (that is, $v(\psi)$). We take the "minimum" of $v(\phi)$ and $v(\psi)$. How does one take the minimum of the two? If we treat false as the number 0 and true as the number 1, then taking the minimum of two numbers makes sense. But is it an accurate translation of AND?

Let us think about this for a moment. When is $(\phi \wedge \psi)$ true? When ϕ is true AND when ψ is true. If we think of true as being the number 1, then we are asking what is the minimum of 1 and 1. And the minimum of those two numbers is 1. If we translate it back, we get true. That seems to work.

Now, when is $(\phi \wedge \psi)$ false? When either ϕ or ψ is false, that is, when $v(\psi) = F$ or (and this is an inclusive or) when $v(\psi) = F$. So, let's think about this. If one of the two is false, then it has a value of 0. The minimum of 0 and anything else is 0. Why? Well, since truth values are either 0 (for false) or 1 (for true), we can only take the minimum of 0 and some other number. That number could be 0, in which case the minimum is 0, or it could be 1, in which case the minimum is still 0. So, the minimum of 0 and any other number (restricted to 0 or 1) is 0. And that makes sense too because we want $(v(\phi \wedge \psi))$ to be false (i.e., 0) when either $v(\phi)$ or $v(\psi)$ is false.

If we treat false as 0 and true as 1, then we can show

$$v((\phi \vee \psi)) = \max(v(\phi), v(\psi))$$

makes sense too. *max* is the function that takes the maximum of two numbers (in this case, we need to treat the truth values like numbers).

The real point of this is to show that to find out the truth value of a Boolean expression (i.e., to find out the value of $v(\phi)$, we need to find out the value of the smaller subexpressions.) For example, to find $v(\neg\phi)$, we need to know the value of $v(\phi)$. And to find out $v(\phi)$ we need to see what pattern ϕ follows (is it a negation, conjunction, or disjunction?) and recursively apply the definition. At each step, we break down the equation into smaller and smaller subexpressions until we reach the smallest subexpression, which happens to be a Boolean variable.

To find the truth value of a Boolean expression, we just need to know the truth value of the Boolean variables in that expression.

Back to Derivations

Based on the insight of the previous section, we now return to our problem. We want to construct a truth table for $((p \wedge q) \vee \neg p)$. To do so, we need to find the Boolean variables in this expression. This is easy as there are only p and q.

This is how we will derive the Boolean expression. The reason will become clear, but we intend to use it to construct a truth table.

Here is the derivation.

Expression	*Reason*
1. p	Boolean variable
2. q	Boolean variable
3. $\neg p$	1, $\neg\phi$
4. $(p \wedge q)$	1, 2, $(\phi \wedge \psi)$
5. $((p \wedge q) \vee \neg p)$	4, 3, $(\phi \wedge \psi)$

We will create one column in the truth table for each line in the derivation. How many rows do we use? If n is the number of Boolean variables (in this case, 2), then 2^n is the number of rows (in this case, $2^2 = 4$ rows).

20) DISCUSS AOUT TAUTOLOGIES AND CONTRADICTIONS

ANS:-

Definition A tautology is a Boolean expression which always results in a true result, regardless of what the Boolean variables in the expression are assigned to.

We saw this earlier on, with the expression $(p \vee \neg p)$. If $v(p) = T$, then the whole expression is true. If $v(p) = F$, then also the whole expression is true. Basically, a tautology means that the result of the expression is independent of whatever Boolean variables have been assigned the result is always true.

Definition A contradiction is a Boolean expression which always results in a false result, regardless of what the Boolean variables in the expression are assigned to.

A contradiction is just the opposite of a tautology, in fact, given any Boolean expression that is a tautology, we just have to negate it to get a contradiction. For example, $(p \vee \neg p)$ is a tautology. The negation of that, $\neg(p \vee \neg p)$, is a contradiction. We can apply DeMorgan's law and get $(\neg p \wedge \neg\neg p)$ and by using the simplification for double negation result in $(\neg p \wedge p)$. So, since all of these are logically equivalent, then $(\neg p \wedge p)$ is also a contradiction.

CONTRADICTION RULE

$\neg p$ is true and then deduce a contradiction, then p is true. The idea runs something like this: one generally believes math is consistent that is, we do not derive contradictions using the

rules of logic (the most common contradiction is to derive $\neg q$ when we also know that q happens to be true). So, when we try to prove p, we try to assume $\neg p$ and if this leads to a contradiction, then we know that $\neg p$ cannot be true, since our system avoids contradiction and thus if $\neg p$ is not true, it is false, and if it is false, then p must be true. This is often the line of reasoning used in a proof by contradiction.

PRINCIPLE OF MANAGEMENT

Subject Code-104

1) Explain the Fayol Principle of Management.

Fayol's principles are listed below:

1. **Division of Work** – When employees are specialized, output can increase because they become increasingly skilled and efficient.
2. **Authority** – Managers must have the authority to give orders, but they must also keep in mind that with authority comes responsibility.
3. **Discipline** – Discipline must be upheld in organizations, but methods for doing so can vary.
4. **Unity of Command** – Employees should have only one direct supervisor.
5. **Unity of Direction** – Teams with the same objective should be working under the direction of one manager, using one plan. This will ensure that action is properly coordinated.
6. **Subordination of Individual Interests to the General Interest** – The interests of one employee should not be allowed to become more important than those of the group. This includes managers.
7. **Remuneration** – Employee satisfaction depends on fair remuneration for everyone. This includes financial and non-financial compensation.
8. **Centralization** – This principle refers to how close employees are to the decision-making process. It is important to aim for an appropriate balance.
9. **Scalar Chain** – Employees should be aware of where they stand in the organization's hierarchy, or chain of command.
10. **Order** – The workplace facilities must be clean, tidy and safe for employees. Everything should have its place.
11. **Equity** – Managers should be fair to staff at all times, both maintaining discipline as necessary and acting with kindness where appropriate.
12. **Stability of Tenure of Personnel** – Managers should strive to minimize employee turnover. Personnel planning should be a priority.
13. **Initiative** – Employees should be given the necessary level of freedom to create and carry out plans.
14. **Esprit de Corps** – Organizations should strive to promote team spirit and unity.

2) **Explain the meaning & concept of Management**

Management is essential for an organized life and necessary to run all types of organizations. Managing life means getting things done to achieve life's objectives and managing an organization means getting things done with and through other people to achieve its objectives.

There are basically five primary functions of management. These are:

1. **Planning**
2. **Organizing**
3. **Staffing**
4. **Directing**
5. **Controlling**

The controlling function comprises coordination, reporting, and budgeting, and hence the controlling function can be broken into these three separate functions. Based upon these seven functions, Luther Gulick coined the word **POSDCORB**, which generally represents the initials of these seven functions i.e. P stands for Planning, O for Organizing, S for Staffing, D for Directing, Co for Co-ordination, R for reporting & B for Budgeting.

"Planning is the continuous process of making present entrepreneurial decisions systematically and with best possible knowledge of their futurity, organizing systematically the efforts needed to carry out these decisions and measuring the results of these decisions against the expectations through organized and systematic feedback".

Organizing requires a formal structure of authority and the direction and flow of such authority through which work subdivisions are defined, arranged and coordinated so that each part relates to the other part in a united and coherent manner so as to attain the prescribed objectives.

Staffing is the function of hiring and retaining a suitable work-force for the enterprise both at managerial as well as non-managerial levels. It involves the process of recruiting, training, developing, compensating and evaluating employees and maintaining this workforce with proper incentives and motivations. Since the human element is the most vital factor in the process of management, it is important to recruit the right personnel.

The directing function is concerned with leadership, communication, motivation, and supervision so that the employees perform their activities in the most efficient manner possible, in order to achieve the desired goals.

The **leadership** element involves issuing of instructions and guiding the subordinates about procedures and methods.

3) Discuss the role & skills of Manager.

Certain **skills,** or abilities to translate knowledge into action that results in desired performance, are required to help other employees become more productive. These skills fall under the following categories:

- **Technical:** This skill requires the ability to use a special proficiency or expertise to perform particular tasks. Accountants, engineers, market researchers, and computer scientists, as examples, possess technical skills. Managers acquire these skills initially through formal education and then further develop them through training and job experience. Technical skills are most important at lower levels of management.
- **Human:** This skill demonstrates the ability to work well in cooperation with others. Human skills emerge in the workplace as a spirit of trust, enthusiasm, and genuine involvement in interpersonal relationships. A manager with good human skills has a high degree of self-awareness and a capacity to understand or empathize with the feelings of others. Some managers are naturally born with great human skills, while others improve their skills through classes or experience. No matter how human skills are acquired, they're critical for all managers because of the highly interpersonal nature of managerial work.
- **Conceptual:** This skill calls for the ability to think analytically. Analytical skills enable managers to break down problems into smaller parts, to see the relations among the parts, and to recognize the implications of any one problem for others. As managers assume ever-higher responsibilities in organizations, they must deal with more ambiguous problems that have long-term consequences. Again, managers may acquire these skills initially through formal education and then further develop them by training and job experience. The higher the management level, the more important conceptual skills become.

4) Discuss Various Managerial Levels

Most organizations have three management levels:

- Low-level managers;
- Middle-level managers; and
- Top-level managers.

These managers are classified in a hierarchy of authority, and perform different tasks. In many organizations, the number of managers in every level resembles a pyramid.
Below, you'll find the specifications of each level's different responsibilities and their likely job titles.

Top-level managers

The board of directors, president, vice-president, and CEO are all examples of top-level managers.
These managers are responsible for controlling and overseeing the entire organization. They develop goals, strategic plans, company policies, and make decisions on the direction of the business.
In addition, top-level managers play a significant role in the mobilization of outside resources.
Top-level managers are accountable to the shareholders and general public.

Middle-level managers

General managers, branch managers, and department managers are all examples of middle-level managers. They are accountable to the top management for their department's function.
Middle-level managers devote more time to organizational and directional functions than top-level managers. Their roles can be emphasized as:

- Executing organizational plans in conformance with the company's policies and the objectives of the top management;
- Defining and discussing information and policies from top management to lower management; and most importantly
- Inspiring and providing guidance to low-level managers towards better performance.

Some of their functions are as follows:

- Designing and implementing effective group and intergroup work and information systems;
- Defining and monitoring group-level performance indicators;
- Diagnosing and resolving problems within and among work groups;
- Designing and implementing reward systems supporting cooperative behavior.

Low-level managers

Supervisors, section leads, and foremen are examples of low-level management titles. These managers focus on controlling and directing.

Low-level managers usually have the responsibility of:

- Assigning employees tasks;
- Guiding and supervising employees on day-to-day activities;
- Ensuring the quality and quantity of production;
- Making recommendations and suggestions; and
- Upchanneling employee problems.

5) Explain the Classical & Neo-Classical Theories of Management

1. Classical Organisation Theory:

The classical writers viewed organisation as a machine and human beings as components of that machine. They were of the view that efficiency of the organisation can be increased by making human beings efficient. Their emphasis was on specialisation and co-ordination of activities. Most of the writers gave emphasis on efficiency at the top level and few at lower levels of organisation. That is why this theory has given streams; scientific management and administrative management. The scientific management group was mainly concerned with the tasks to be performed at operative levels.

Henry Fayol studied for the first time the principles and functions of management. Some authors like Gullick, Oliver Sheldon, Urwick viewed the problem where identification of activities is necessary for achieving organisation goals. Grouping or departmentation was also considered essential for making the functions effective. Since this theory revolves around structure it is also called 'structural theory of organisation."

2. Neo-Classical Organisation Theory:

The classical theory of organisation focussed main attention on physiological and mechanical variables of organisational functioning. The testing of these variables did not show positive results. The Hawthorne Studies conducted by George Elton Mayo and associates discovered

that real cause of human behaviour was somewhat more than mere physiological variables. These studies focussed attention on human beings in the organisation.

New-classical approach is contained in two points:

(i) Organisational situation should be viewed in social, economic and technical terms, and

(ii) The social process of group behaviour can be understood in terms of clinical method analogous to the doctor's diagnosis of human organism.

6) Explain the Process & Purpose of Planning

The following are some of the important purpose of planning in an organization.

1. **Facilitates Accomplishment of Objectives**: The aim of planning is to facilitate the attainment of objectives. It focuses its attention on the objectives of the organization. It states the objectives of each department in the organization and of the enterprise as a whole. This helps personnel to see the enterprise in its entirety and see how their actions contribute to its ultimate goals. Planning forces the managers to consider the future and revise its plans if necessary for achieving the objectives.

2. **Ensures Economy in Operations**: Since planning emphasizes efficient operation and consistency, it minimizes costs and gains economical operation. Coordinated group effort, even flow of work and deliberate decisions are due to planning.

3. **Precedes Control**: Control involves those activities which are carried out to force events to conform to plans. Plans serves as standards of performance. Control seeks to compare actual performance with set standards. So control cannot be exercised without plans.

4. **Provides for Future Contingency**: Planning is required because future is uncertain. Planning enables the management to look into the future and discover suitable alternative course of action. Planning helps the management to have a clear-cut idea about the future and to frame a suitable programme for action. Even when the future is highly certain, planning is essential to decide the best course of action.

5. **Facilitates Optimum Utilization of Resources**: Various resources that are relevant to an organization namely, funds, physical resources, manpower, technological know-how, etc., are by and large inadequate due to demand from competing organizations and have alternative uses. This necessitate the organization to make the best possible use of resources. Planning facilitates optimum use of available resources.

7) **Discuss the Decision making Process**

Step 1: Identify the decision

You realize that you need to make a decision. Try to clearly define the nature of the decision you must make. This first step is very important.

Step 2: Gather relevant information

Collect some pertinent information before you make your decision: what information is needed, the best sources of information, and how to get it. This step involves both internal and external "work." Some information is internal: you'll seek it through a process of self-assessment. Other information is external: you'll find it online, in books, from other people, and from other sources.

Step 3: Identify the alternatives

As you collect information, you will probably identify several possible paths of action, or alternatives. You can also use your imagination and additional information to construct new alternatives. In this step, you will list all possible and desirable alternatives.

Step 4: Weigh the evidence

Draw on your information and emotions to imagine what it would be like if you carried out each of the alternatives to the end. Evaluate whether the need identified in Step 1 would be met or resolved through the use of each alternative. As you go through this difficult internal process, you'll begin to favor certain alternatives: those that seem to have a higher potential for reaching your goal. Finally, place the alternatives in a priority order, based upon your own value system.

Step 5: Choose among alternatives

Once you have weighed all the evidence, you are ready to select the alternative that seems to be best one for you. You may even choose a combination of alternatives. Your choice in Step 5 may very likely be the same or similar to the alternative you placed at the top of your list at the end of Step 4.

Step 6: Take action

You're now ready to take some positive action by beginning to implement the alternative you chose in Step 5.

Step 7: Review your decision & its consequences

In this final step, consider the results of your decision and evaluate whether or not it has resolved the need you identified in Step 1. If the decision has *not* met the identified need, you may want to repeat certain steps of the process to make a new decision. For example, you

might want to gather more detailed or somewhat different information or explore additional alternatives.

8) Distinguish between Authority & Responsibility

Authority is the power to give orders and get it obeyed or in other words it is the power to take decisions.

Responsibility means state of being accountable or answerable for any obligation, trust, debt or something or in other words it means obligation to complete a job assigned on time and in best way.

Authority and responsibility are closely related and this principle states that these two must go hand in hand. It means that proper authority should be delegated to meet the responsibilities.

A match should be there between these two because of two main reasons:--

- ü Firstly, if a person is given some responsibility without sufficient authority he can't perform better, and also could not accomplish the desired goal.
- ü Secondly, if there is excess authority being delegated to an individual without matching responsibility then the delegated authority will be misused in one way or the other.

This is an important and useful principle of management because if adequate authority is not delegated to the employees they cannot discharge their duties with efficiency and this in turn will hamper the achievement of the organizational goal. Sometimes the relation between management and employees is also badly effected by non delegation of proper authority.

Positive impacts of this principle:

- Ø No misuse of authority.
- Ø Helps to complete job effectively and efficiently.
- Ø Individuals can be held accountable.
- Ø Systematized and effective achievement of organizational objectives.

Consequences of violation of this principle:

- Ø Misuse of authority.
- Ø Responsibility can't be discharged effectively.
- Ø No one can be held accountable.
- Ø Conflicts between management and employees.

9) Explain Decentralization

The process in which the power or authority present in the hands of the State and Central Government is taken back and is allocated to the local government it is called decentralization. In this mechanism the authoritative nature of people with power is eradicated and powers are handed over to the suitable person who can make good use of those powers for human welfare and in turn will pave way for development of their nation.

10) Discuss the Nature & Importance of Staffing

Staffing is the process of hiring eligible candidates in the organization or company for specific positions. In management, the meaning of staffing is an operation of recruiting the employees by evaluating their skills, knowledge and then offering them specific job roles accordingly.

Functions of Staffing

1. The first and foremost function of staffing is to obtain qualified personnel for different jobs position in the organization.
2. In staffing, the right person is recruited for the right jobs, therefore it leads to maximum productivity and higher performance.
3. It helps in promoting the optimum utilization of human resource through various aspects.
4. Job satisfaction and morale of the workers increases through the recruitment of the right person.
5. Staffing helps to ensure better utilization of human resources.
6. It ensures the continuity and growth of the organization, through development managers.

11) Explain the Process of Directing.

DIRECTING is said to be a process in which the managers instruct, guide and oversee the performance of the workers to achieve predetermined goals. Directing is said to be the heart of management process. Planning, organizing, staffing have got no importance if direction function does not take place.

Directing initiates action and it is from here actual work starts. Direction is said to be consisting of human factors. In simple words, it can be described as providing guidance to workers is doing work. In field of management, direction is said to be all

those activities which are designed to encourage the subordinates to work effectively and efficiently. According to Human, "Directing consists of process or technique by which instruction can be issued and operations can be carried out as originally planned" Therefore, Directing is the function of guiding, inspiring, overseeing and instructing people towards accomplishment of organizational goals.

Direction has got following characteristics:

1. **Pervasive Function -** Directing is required at all levels of organization. Every manager provides guidance and inspiration to his subordinates.
2. **Continuous Activity -** Direction is a continuous activity as it continuous throughout the life of organization.
3. **Human Factor -** Directing function is related to subordinates and therefore it is related to human factor. Since human factor is complex and behaviour is unpredictable, direction function becomes important.
4. **Creative Activity -** Direction function helps in converting plans into performance. Without this function, people become inactive and physical resources are meaningless.
5. **Executive Function -** Direction function is carried out by all managers and executives at all levels throughout the working of an enterprise, a subordinate receives instructions from his superior only.
6. **Delegate Function -** Direction is supposed to be a function dealing with human beings. Human behaviour is unpredictable by nature and conditioning the people's behaviour towards the goals of the enterprise is what the executive does in this function. Therefore, it is termed as having delicacy in it to tackle human behaviour.

12) Discuss Maslow Theory of Motivation.

One of the most popular needs theories is **Abraham Maslow's hierarchy of needs theory**. Maslow proposed that motivation is the result of a person's attempt at fulfilling five basic needs: physiological, safety, social, esteem and self-actualization. According to Maslow, these needs can create internal pressures that can influence a person's behavior.

Physiological needs are those needs required for human survival such as air, food, water, shelter, clothing and sleep. As a manager, you can account for the physiological needs of your employees by providing comfortable working conditions, reasonable work hours and the necessary breaks to use the bathroom and eat and/or drink.

Safety needs include those needs that provide a person with a sense of security and well-being. Personal security, financial security, good health and protection from accidents, harm and their adverse effects are all included in safety needs. As a manager, you can account for the safety needs of your employees by providing safe working conditions, secure compensation (such as a salary) and job security, which is especially important in a bad economy.

Social needs, also called **love and belonging**, refer to the need to feel a sense of belonging and acceptance. Social needs are important to humans so that they do not feel alone, isolated and depressed. Friendships, family and intimacy all work to fulfill social needs. As a manager, you can account for the social needs of your employees by making sure each of your employees know one another, encouraging cooperative teamwork, being an accessible and kind supervisor and promoting a good work-life balance.

Esteem needs refer to the need for self-esteem and respect, with self-respect being slightly more important than gaining respect and admiration from others. As a manager, you can account for the esteem needs of your employees by offering praise and recognition when the employee does well, and offering promotions and additional responsibility to reflect your belief that they are a valued employee.

Self-actualization needs describe a person's need to reach his or her full potential. The need to become what one is capable of is something that is highly personal. While I might have the need to be a good parent, you might have the need to hold an executive-level position within your organization. Because this need is individualized, as a manager, you can account for this need by providing challenging work, inviting employees to participate in decision-making and giving them flexibility and autonomy in their jobs.

13) Explain McGregor X & Y Theory of Leadership

Theory X

Theory X managers tend to take a pessimistic view of their people, and assume that they are naturally unmotivated and dislike work. As a result, they think that team members need to be prompted, **rewarded** or punished constantly to make sure that they complete their tasks.

Work in organizations that are managed like this can be repetitive, and people are often motivated with a "carrot and stick" approach. Performance **appraisals** and **remuneration** are usually based on tangible results, such as sales figures or product output, and are used to control staff and "keep tabs" on them.

This style of management assumes that workers:

- Dislike their work.
- Avoid responsibility and need constant direction.
- Have to be controlled, forced and threatened to deliver work.
- Need to be supervised at every step.
- Have no incentive to work or ambition, and therefore need to be enticed by rewards to achieve goals.

Theory Y

Theory Y managers have an optimistic, positive opinion of their people, and they use a decentralized, participative management style. This encourages a more **collaborative** , **trust-based** relationship between managers and their team members.

People have greater responsibility, and managers encourage them to develop their skills and suggest improvements. Appraisals are regular but, unlike in Theory X organizations, they are used to encourage open communication rather than control staff.

Theory Y organizations also give employees frequent opportunities for promotion.

This style of management assumes that workers are:

- Happy to work on their own initiative.
- More involved in decision making.
- Self-motivated to complete their tasks.
- Enjoy **taking ownership** of their work.
- Seek and accept responsibility, and need little direction.

14) Discuss Herzberg Two Factor Theory.

1. Motivating Factors

The presence of motivators causes employees to work harder. They are found within the actual job itself.

2. Hygiene Factors

The absence of hygiene factors will cause employees to work less hard. Hygiene factors are not present in the actual job itself but surround the job.

Motivating factors include:

- **Achievement**: A job must give an employee a sense of achievement. This will provide a proud feeling of having done something difficult but worthwhile.
- **Recognition**: A job must provide an employee with praise and recognition of their successes. This recognition should come from both their superiors and their peers.
- **The work itself**: The job itself must be interesting, varied, and provide enough of a challenge to keep employees motivated.
- **Responsibility**: Employees should "own" their work. They should hold themselves responsible for this completion and not feel as though they are being micromanaged.
- **Advancement**: Promotion opportunities should exist for the employee.
- **Growth**: The job should give employees the opportunity to learn new skills. This can happen either on the job or through more formal training.

Hygiene factors include:

- **Company policies**: These should be fair and clear to every employee. They must also be equivalent to those of competitors.
- **Supervision**: Supervision must be fair and appropriate. The employee should be given as much autonomy as is reasonable.
- **Relationships**: There should be no tolerance for bullying or cliques. A healthy, amiable, and appropriate relationship should exist between peers, superiors, and subordinates.
- **Work conditions**: Equipment and the working environment should be safe, fit for purpose, and hygienic.

- **Salary**: The pay structure should be fair and reasonable. It should also be competitive with other organizations in the same industry.
- **Status**: The organization should maintain the status of all employees within the organization. Performing meaningful work can provide a sense of status.

15) Elaborate on the Nature & Importance of Controlling.

One of the most essential qualities required in a manager is that he should command the respect of his team. This allows him to direct and control their actions. In fact controlling is one of his more important functions. Controlling is one of the important functions of a manager. In order to seek planned results from the subordinates, a manager needs to exercise effective control over the activities of the subordinates. In other words, the meaning of controlling function can be defined as ensuring that activities in an organization are performed as per the plans. Controlling also ensures that an organization's resources are being used effectively & efficiently for the achievement of predetermined goals.

- Controlling is a goal-oriented function.
- It is a primary function of every manager.
- Controlling the function of a manager is a pervasive function.

Managers at all levels of management Top, Middle & Lower – need to perform controlling function to keep control over activities in their areas. Therefore, controlling is very much important in an educational institution, military, hospital, & a club as in any business organization.

Therefore, controlling function should not be misunderstood as the last function of management. It is a function that brings back the management cycle back to the planning function. Thus, the controlling function act as a tool that helps in finding out that how actual performance deviates from standards and also finds the cause of deviations & attempts which are necessary to take corrective actions based upon the same.

This process helps in the formulation of future plans in light of the problems that were identified &, thus, helps in better planning in the future periods. So from the meaning of controlling we understand it not only completes the management process but also improves planning in the next cycle.

Importance of Controlling

After the meaning of control, let us see its importance. Control is an indispensable function of management without which the controlling function in an organization cannot be accomplished and the best of plans which can be executed can go away. A good control system helps an organization in the following ways:

1. Accomplishing Organizational Goals

The controlling function is an accomplishment of measures that further makes progress towards the organizational goals & brings to light the deviations, & indicates corrective action. Therefore it helps in guiding the organizational goals which can be achieved by performing a controlling function.

2. Judging Accuracy of Standards

A good control system enables management to verify whether the standards set are accurate & objective. The efficient control system also helps in keeping careful and progress check on the changes which help in taking the major place in the organization & in the environment and also helps to review & revise the standards in light of such changes.

3. Making Efficient use of Resources

Another important function of controlling is that in this, each activity is performed in such manner so an in accordance with predetermined standards & norms so as to ensure that the resources are used in the most effective & efficient manner for the further availability of resources.

16) Explain the need of Understanding Human Behaviour in Organization.

Organizational behavior deals with the study of human behavior within groups or organizations and how this behavior can be modeled through analysis to impact the organizations in a positive way. An organization in itself is composed of a group of people working individually or often within teams. The disposition of people towards each other in an organization remains the contributing factor towards shaping the organization.

Organizational Behavior is an interdisciplinary field, in that it draws greatly from other subjects such as psychology, sociology, anthropology, political science and economics, to mention a few.

The success of an organization is largely dependent on effective management of its people. Behavior of people within an organization is governed by their ideas, feelings and activities. For effective management of people, it is crucial to perceive their requirements. However, since human behavior can differ with each individual, it becomes almost impossible to come with a unique solution to the organizational problems. For this reason, it is important to consider psychological and social aspects to design solutions focused on solving organizational issues.

The behavior of individuals within an organization can either defile the organization or aid in its overall improvement. For instance, certain employees may be compassionate and helpful towards their co-workers which helps to create a supportive work culture. This selfless attitude can be a result of the employee's faith in the management and their satisfaction and commitment towards the organization.

In order to solve the organizational problems, it is necessary to first understand the reason for its occurrence. If the problems faced are due to damaging organizational behavior, it becomes crucial to recognize the purpose behind such a behavior. Only then can measures be taken to counter it and guide the organization in a progressive direction. Incorporation of new and encouraging behaviors in the company culture might prove rewarding to the employees and as such profit the company as a whole.

There are many factors that promote constructive organizational behavior, such as accomplishments, self-actualization, encouragement, affiliation etc. Management should try to figure out the driving force that stimulates such a behavior and try to integrate more such factors in the work culture.

This can be in the form of promotions, new incentives, plans or rewards. It does not make sense to hire capable people and expect complete dedication in the absence of pro-social and rewarding company culture. In fact, people are more likely to lose interest in their work if they don't feel recognized and rewarded.

17) Discuss the Various models of OB

Organizational Behavior – Our inherent power of generalization helps us to predict the behavior of other people, however sometimes our generalizations and predictions fail. This happens as we fail to analyze and go into the depth of the patterns that are affecting the behavior of people at that particular time or period. This calls for understanding and following the systematic approach to the study of the organizational behavior. The study helps in increasing our predictive ability to understand the behavior of the people particularly in the group or an organization, and how their behavior impacts the performance of an organization. Almost all organizations develop the models on the basis of which behavior of the people is determined. This model depends on the assumption that organizational behavior management carries about its people and mission and goals. It is noted that most of the organizations make the assumptions on the basis that people are not to be trusted even in the slightest matter. For instance McGregor theories X and Y is based on quite contradictory assumptions; Argyris focuses on the immaturity and maturity level of the people providing two opposing views. The Organizational Behavior models formulated would show many different variations and kind of continuum between the two opposite poles.

In management, the focus is on the study of the five organizational behavior models:

- Autocratic Model
- Custodial Model
- Supportive Model
- Collegial Model
- System Model

1. Autocratic model

This model has its roots in the historical past, and definitely became a most prominent model of the industrial revolution of 1800 and 1900s. It gives the owners and manager's power to dictate and form decisions while making employees obey their orders. The model asserts that employees need to be instructed and motivated to perform while managers do all the thinking. The whole process is formalized with the managers and authority power has the

right to give command to the people, "You do this or else…", is a general dictatorship command.

2. Custodial Model

Now the time came when managers began to think the security of the employees is imperative- it could be either social as well economic security. Now managers have begun to study about their employees needs, they found out that though in the autocratic setup employees does not talk back yet they have many things to say but incapability to speak result in frustrations, insecurity, and aggressive behavior towards their boss.

3. Supportive Model

Unlike the two previous approaches, the supportive model emphasis on motivated and aspiring leader. There is no space for any control or authoritative power in this model or on the incentives or reward schemes but it is simply based on motivating staff through the establishment of the manager and employee relationship and the treatment that is given to employees on daily basis.

4. The Collegial Model

In this scheme, the structure of an organization is developed in a way that there is no boss nor subordinates, but all are colleagues who have to work as a team. Each one of the employees has to participate and work in coordination with each other to achieve the target rate. No one is worried about his status or a job title. Manager's role is here like a coach whose function is to guide the team to perform and generate positive and motivating work environment, instead of focusing on his own personal growth. The team requires adopting new approaches, research and development and new technologies to better their performance.

5. The System Model

The most emerging model of the today's corporate era is the system model. This model emerged from the rigorous research to attain the higher level of meaning at work. Today's employees need more than salary and security from their job, they need the hours they are putting towards the organization is giving them some value and meaning. To add to it, they need the work that is ethical, respectful, integrated with trust and integrity and gives a space to develop a community feeling among the co-workers.

18) Explain the concept of Personality & Learning in OB

The four main personality theories are the following: Psychoanalytic, trait, humanistic, and social-cognitive. These theories all deal with the origin and

development of personality traits and identity, and they go about studying the personality in very different ways.

Psychoanalytic Views on Personality

Sigmund Freud believed that personality is made up of three components. The **id** is our impulse energy. It is responsible for all our needs (nourishment, appreciation) and urges (sexual instinct, hate, love and envy). According to Freud, the id seeks immediate satisfaction of our needs without referring to logic or morals. It is demanding, impulsive, blind, irrational, antisocial, selfish and lust oriented – our most primal instinct.

The **superego,** or conscience, represents morality as well as the norms of society. It contains all the ideals for which an individual strives and makes us feel guilty if we fall short of these standards. The superego essentially is our standard of perfection – the person we want to be. While the id strives for pleasure and the superego for perfection, the **ego** acts to moderate the two. It works on the reality principle, mediating the competing demands of the id and the superego and choosing the most realistic solution for the long term.

Trait Theory of Personality

According to the trait theory, personality is made up of a number of **stable characteristics, or traits,** that cause a person to act in a certain way. These traits are the blueprint for how we behave. Examples include introversion, sociability, aggressiveness, submissiveness, loyalty and ambition.

Perhaps the most scientific of all the trait theories, in the sense that an impressive body of research supports it, is the five-factor model, more typically known as the **Big Five**.

Humanistic Views on Personality

The key agent of the humanist movement is **Abraham Maslow.** Maslow believed that personality was not a matter of nature or nurture but of personal choice. Specifically, he suggested that people possess free will and are motivated to pursue the things that will help them reach their full potential as human beings.

Maslow developed a hierarchy of needs which typically is displayed as a pyramid. The bottom tier of the pyramid is made up of the most basic needs: **food, water, sleep** and **shelter.** These needs are so important that people act to meet them before doing anything else. Once those needs are met, people can move through the other levels of the pyramid, meeting the needs of **safety, belonging** and **self-esteem** until they reach the final level: **self-actualization**. Self-actualization is the process of developing and growing in order to reach your true potential. This, said Maslow, is a key motivator of human behavior.

Social Cognition Theory

The social cognition theory views personality through the lens of our social interactions, so instead of developing in a black box, our personality traits interact with our environment to influence behavior. This gives a much clearer view of the effect that other people have on our personalities.

The pioneer of the social cognition theory is a scientist named **Albert Bandura.** He argued that when people see someone gaining benefit from a certain behavior, they copy that behavior in order to earn a similar reward. His famous experiment saw a child being rewarded with a doll for punching a doll. When other children were shown the video, they acted in a similarly aggressive way to earn a reward. Thus, personality traits (in this case aggression) may be learned.

19) Define Perception & Attitude building.

Attitude is someone's reaction towards a certain situation or person according to their perception.

Personality is the quality of a person that makes their character. Someone's personality is what differentiates them from any other human being. I like to think that it gives everyone their aura.

Behaviour is the way a person treats other people, the way they carry themselves in a society. It comprises of manners, language.

Perception is the way a person views situations and how they make that person feel. It relates to positive or negative way of thinking.

1. Perception

Each of us has a particular way of perceiving and making sense of the world around us. It is tempting to assume that human behavior is a response to an objective reality but, as the comedian Lily Tomlin noted, "Reality is nothing more than a collective hunch." The same stimuli may be present in our environment, but what we do with that stimuli is affected by individual differences.

Perception is the selection and organization of environmental information to provide meaningful experiences to the perceiver. It is the process of making sense of sensory data. Perception serves as a filter or gatekeeper so that we are not overwhelmed by all the stimuli that bombard us. We need to pay attention to three aspects of perception: organizing data, selective attention, and perceptual bias. We organize information according to similarity, figure ground (what is in front compared to what is in the background), proximity, closure (filling in the gaps), continuity (continue things in a direction they seem to be heading), and simplicity (reducing things to their simplest shapes or patterns). We also have patterns of perception based in our life experience that become our schemas. (Schemas are mental frameworks that help us manage information by grouping individuals, objects and situations together). And we put together information into cause-and-effect patterns. All these together – organization, schemas, cause-and-effect patterns -- become our frame of reference. Once our frame of reference is established, it is usually efficient in managing environmental stimuli. It serves to focus our attention.

2. Selective Attention, Perceptual Distortion and Stereotypes

Selective attention means that we perceive only some of the stimuli that are actually present – usually information that fits into our existing frame of reference. Our ability to perceive information outside the frame or information that would eliminate the frame itself (discon organizationing data) is usually limited once this process is in use. We have a number of perceptual distortions that result from our particular way of organizing information and attentional focus. Some common distortions include halo / horn effects, projection, self-fulfilling prophecy and stereotyping. The halo effect occurs when one positive characteristic or skill a person has is used to develop an overall positive impression of that person, often in unrelated or irrelevant areas. The horn effect is when one negative characteristic or skill is made into a negative overall impression of a person. Projection is when an individual

attributes his/her attitudes or feelings to another person. It is a defense mechanism which serves to transfer blame and/or provide protection from our own unacceptable thoughts and feelings. Self-fulfilling prophecy occurs when our beliefs-expectations determine our behavior thereby making our expectations come true.

Stereotyping is the all too frequent result of rapid, automatic perception and attribution processes when we are dealing with people we consider to be different from us. A stereotype is an oversimplified evaluative opinion or judgement about a group of people applied to an individual. Stereotyping occurs when we attribute behavior, attitudes, motives, and/or attributes to a person on the basis of the group to which that person belongs. Just because stereotyping is so common in society does not mean we should accept stereotypical relating as inevitable. Stereotypes have negative consequences in relationships at work. Slowing down, describing rather than evaluating behavior, learning more about the individual or group with whom you are interacting, and consciously choosing behaviors that will enhance your relationship will all reduce, if not eliminate the negative impact of stereotyping.

Our perception processes have both advantages and drawbacks. The drawbacks are that selective attention and perceptual bias can prevent us from considering all the relevant information, thereby making our interpretations about the meaning of that information unreliable. The advantage is that our perceptual processes improve our decision making efficiency by preventing information overload and saving us time by organizing the information.

3. Attribution Process

Attribution refers to the specification of the perceived causes of events. It is our way to answer the question "Why did I/they do that?" We have learned through our study of attribution processes that:

- Different people often attribute different causes to the same event.
- When people try to understand their own or others' behavior they focus on the personal (internal) or situational (external) factors.

We have predictable attributional biases based on a combination of three factors:

- *Consensus.* How many others behaved in the same way as that individual? If that person's behavior is unique we attribute the cause of the behavior to that person's internal personality. If that person's behavior is like the behavior of others we attribute the cause of the behavior to the situation.

- *Distinctiveness.* How consistent or unusual is that person's behavior across situations? If that person's behavior is routine for them across situations we attribute the cause of the behavior to the personal factors. If that person's behavior is unusual when compared to their behavior in other situations, we attribute the cause of the behavior in this case to the situation.

- *Consistency.* How consistent is this person's behavior over time? If this person always acts this way and has done so all their life, we attribute the cause of the behavior to individual personality. If this person's behavior is different from their past or typical behavior we attribute the cause of the behavior to the situation or circumstances.

So in each case there is a decision made whether the cause of the behavior is due more to the personality or to the situation. We tend to be more generous with ourselves though, than with others.

4. The Fundamental Attribution Error and Self-Serving Bias

We also have a tendency to under estimate the influence of the situation and to over estimate the influence of personality when we are making judgements about others. We do the reverse for ourselves. This is called the Fundamental Attribution Error.

Moreover, we have a self-serving bias depending upon whether the behavior is considered good-positive or bad-negative. If it is good, it's because I am good. If it is bad, it is because the situation made me do it. Self-serving bias is the tendency to take credit and responsibility for positive outcomes of behavior and to deny credit and responsibility for negative outcomes.

Recent research supports the notion of a difference in these biases by gender. Women are more likely to attribute failure to themselves and success to external factors such as luck or task ease. Men are more likely to attribute success to their own efforts and failure to external factors such as time limitations or monetary constraints.

5. Attitudes

Attitudes are relatively lasting tendencies to consistently respond to various aspects of people, situations, or objects. Attitudes have three components: cognition (beliefs), affect (emotions), and behavior. These components of an attitude do not exist or function separately. Of the three, we can observe behavior, we infer beliefs, and we sense feelings. From these we attribute motives to people, including ourselves. Attitudes reflect how we feel, think and act. When I say "I am committed to my job" I am expressing my attitude about my work. When I attend work every day, I am expressing my attitude about my work.

Attitudes are the result of our learned experiences in life. We develop our attitudes through easily available information, personal experiences, and repeated expression. We learn them from our friends, family, media, culture, teachers, peers and role models. Attitudes are related to but different from values which we will discuss in the next module. It is important for individuals to have alignment between their cognition, affect and behavior. Festinger coined the term cognitive dissonance to refer to internal conflict between our beliefs. We can extend this idea of dissonance to include conflict between our personal beliefs, feelings and behaviors – attitudinal dissonance. Dissonance is an unpleasant state. When we experience cognitive or attitudinal dissonance, we are compelled to change one or the other component of our attitude to regain alignment. We tend to change either our beliefs, or our behaviors. Because behaviors can be seen and somewhat controlled, many people change attitudes through encouraging acceptable behaviors and constraining unacceptable behaviors. The person then feels compelled / motivated to change any beliefs or feelings aligned with the old behaviors, and to develop new beliefs that would be consistent with the new behaviors. Our attitudes influence our behavior -- when they are relevant and brought to mind. The reverse is also true: we are as likely to act ourselves into a way of thinking as to think ourselves into action. We are as likely to believe in what we have stood up for as to stand up for what we believe. Especially when we feel responsible for how we have acted, our beliefs and feelings follow our behavior. It is important to realize that inner feelings and thoughts and outer behaviors – all components of our attitudes -- like chickens and eggs generate one another.

20) How people are manage within the Organization. Elaborate

Managers who work in small corporations and companies must have people management skills to effectively perform their jobs. Most companies use a hierarchical organization structure that requires managers or supervisors to oversee the work of others. Hence, these managers must oversee and coordinate the work of others to complete various projects. People management skills can include communication, leadership, delegation, motivation, training and even performance feedback.

Communication

One important people management skill is communication. Managers of small companies must know how to effectively communicate with employees to let them know what they expect from them on the job. Often, managers conduct one-on-one or even group meetings to keep employees apprised of certain projects or management decisions. Managers also communicate with their employees through status reports. In addition, management employees should know how to listen to employees, as some workers may have suggestions on performing tasks more efficiently. Employees may also have personal problems, where a manager may need to give a worker time off when needed.

Training

Training is particularly important for new employees. Managers train employees directly, have experienced coworkers train them or coordinate an employee's offsite classroom training. Whatever the case, a manager needs to recognize an employee's skills and determine what training the worker needs to effectively perform his job.

Delegating

People management skills also include delegating. Managers usually have numerous projects to complete, and there are specific deadlines for these projects. Because managers in small companies cannot do all the work themselves, they need to delegate or assign tasks to other workers. The manager then holds each worker accountable for completing their tasks before the project deadline.

Managers must know the right employee to whom to delegate a specific task, according to the article titled "Free Basic Guide to Leadership and Supervision" on the Free Management

Library website. In other words, managers should assign tasks to employees based on their abilities and strengths. That way managers can be more confident that the work will be completed correctly.

Motivating

People management skills also include motivation, which is getting employees to have more excitement about performing their jobs. An effective manager knows that different things motivate employees. Some employees prefer closer supervision. Other employees prefer to be challenged in their jobs, desiring to take on more responsibility. Managers can often learn what best motivates employees by discussing their personal goals with them. Managers can then start helping them achieve these goals by assigning projects that best utilizes their creativity.

Performance Feedback

All managers must provide performance feedback to their workers. The formal way of conducting performance feedback is a performance appraisal. A performance appraisal is a half hour or hour session where the manager reviews an employee's work over the past six to 12 months. She may explain what tasks the worker performs well and areas where he needs improvement. Effective managers will not just evaluate an employee's work. They will write a development plan for the employee to help improve his performance.

Digital electronics

BCA 106

Unit 1

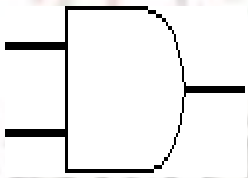
Q1. Explain Logic gate AND, OR, XOR, NOT, NAND, NOR and XNOR.

A logic gate is a building block of a digital circuit. Most logic gates have two inputs and one output and are based on Boolean algebra. At any given moment, every terminal is in one of the two binary conditions *false* (high) or *true* (low). False represents 0, and true represents 1. Depending on the type of logic gate being used and the combination of inputs, the binary output will differ. A logic gate can be thought of like a light switch, wherein one position the output is off—0, and in another, it is on—1. Logic gates are commonly used in integrated circuits (IC).

Basic logic gates

There are seven basic logic gates: AND, OR, XOR, NOT, NAND, NOR, and XNOR.

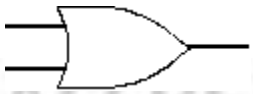
AND Gate: The *AND gate* is so named because, if 0 is called "false" and 1 is called "true," the gate acts in the same way as the logical "and" operator. The following illustration and table show the circuit symbol and logic combinations for an AND gate. (In the symbol, the input terminals are at left and the output terminal is at right.) The output is "true" when both inputs are "true." Otherwise, the output is "false." In other words, the output is 1 only when both inputs one AND two are 1.



AND gate

Input 1	Input 2	Output
	1	
1		
1	1	1

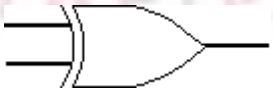
The *OR gate* gets its name from the fact that it behaves after the fashion of the logical inclusive "or." The output is "true" if either or both of the inputs are "true." If both inputs are "false," then the output is "false." In other words, for the output to be 1, at least input one OR two must be 1.



OR gate

Input 1	Input 2	Output
	1	1
1		1
1	1	1

The *XOR* (*exclusive-OR*) *gate* acts in the same way as the logical "either/or." The output is "true" if either, but not both, of the inputs are "true." The output is "false" if both inputs are "false" or if both inputs are "true." Another way of looking at this circuit is to observe that the output is 1 if the inputs are different, but 0 if the inputs are the same.



XOR gate

Input 1	Input 2	Output
	1	1
1		1
1	1	

A logical *inverter,* sometimes called a *NOT gate* to differentiate it from other types of electronic inverter devices, has only one input. It reverses the logic state. If the input is 1, then the output is 0. If the input is 0, then the output is 1.

Inverter or NOT gate

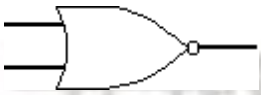
Input	Output
1	
	1

The *NAND gate* operates as an AND gate followed by a NOT gate. It acts in the manner of the logical operation "and" followed by negation. The output is "false" if both inputs are "true." Otherwise, the output is "true."

NAND gate

Input 1	Input 2	Output
		1
	1	1
1		1
1	1	

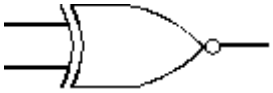
The *NOR gate* is a combination OR gate followed by an inverter. Its output is "true" if both inputs are "false." Otherwise, the output is "false."



NOR gate

Input 1	Input 2	Output
		1
	1	
1		
1	1	

The *XNOR (exclusive-NOR) gate* is a combination XOR gate followed by an inverter. Its output is "true" if the inputs are the same, and "false" if the inputs are different.



XNOR gate

Input 1	Input 2	Output
		1
	1	
1		
1	1	1

Using combinations of logic gates, complex operations can be performed. In theory, there is no limit to the number of gates that can be arrayed together in a single device. But in practice, there is a limit to the number of gates that can be packed into a given physical space. Arrays of logic gates are found in digital ICs. As IC technology advances, the required physical volume for each individual logic gate decreases and digital devices of the same or smaller size become capable of performing ever-more-complicated operations at ever-increasing speeds.

Composition of logic gates

High or low binary conditions are represented by different voltage levels. The logic state of a terminal can, and generally does, change often as the circuit processes data. In most logic gates, the low state is approximately zero volts (0 V), while the high state is approximately five volts positive (+5 V).

Logic gates can be made of resistors and transistors, or diodes. A resistor can commonly be used as a pull-up or pull-down resistor. Pull-up or pull-down resistors are used when there are any unused logic gate inputs to connect to either a logic level 1 or 0 respectively. This prevents any false switching of the gate. Pull-up resistors are connected to Vcc (+5V), and pull-down resistors are connected to ground (0 V).

Commonly used logic gates are TTL and CMOS. TTL, or Transistor-Transistor Logic, ICs will use NPN and PNP type Bipolar Junction Transistors. CMOS, or Complementary Metal-Oxide-Silicon, ICs are constructed from MOSFET or JFET type Field Effect Transistors. TTL IC's may commonly be labeled as the 7400 series of chips, while CMOS ICs may often be marked as a 4000 series of chips.

Q2. Explain Diode & Transistor as Switch.

Electronic Circuits - Diode as a Switch

Diode is a two terminal PN junction that can be used in various applications. One of such applications is an electrical switch. The PN junction, when forward biased acts as close circuited and when reverse biased acts as open circuited. Hence the change of forward and reverse biased states makes the diode work as a switch, the **forward** being **ON** and the **reverse** being **OFF** state.

Electrical Switches over Mechanical Switches

Electrical switches are a preferred choice over mechanical switches due to the following reasons −

Mechanical switches are prone to oxidation of metals whereas electrical switches don't.

Mechanical switches have movable contacts.

They are more prone to stress and strain than electrical switches.

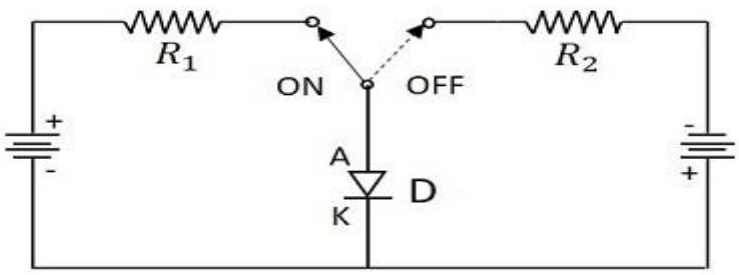
The worn and torn of mechanical switches often affect their working.

Hence an electrical switch is more useful than a Mechanical switch.

Working of Diode as a Switch

Whenever a specified voltage is exceeded, the diode resistance gets increased, making the diode reverse biased and it acts as an open switch. Whenever the voltage applied is below the reference voltage, the diode resistance gets decreased, making the diode forward biased, and it acts as a closed switch.

The following circuit explains the diode acting as a switch.



Switching circuit using Diode

A switching diode has a PN junction in which P-region is lightly doped and N-region is heavily doped. The above circuit symbolizes that the diode gets ON when positive voltage forward biases the diode and it gets OFF when negative voltage reverse biases the diode.

Ringing

As the forward current flows till then, with a sudden reverse voltage, the reverse current flows for an instance rather than getting switched OFF immediately. The higher the leakage current, the greater the loss. The flow of reverse current when diode is reverse biased suddenly, may sometimes create few oscillations, called as **RINGING**.

This ringing condition is a loss and hence should be minimized. To do this, the switching times of the diode should be understood.

Diode Switching Times

While changing the bias conditions, the diode undergoes a **transient response**. The response of a system to any sudden change from an equilibrium position is called as transient response. The sudden change from forward to reverse and from reverse to forward bias, affects the circuit. The time taken to respond to such sudden changes is the important criterion to define the effectiveness of an electrical switch.

The time taken before the diode recovers its steady state is called as **Recovery Time**.

The time interval taken by the diode to switch from reverse biased state to forward biased state is called as **Forward Recovery Time.**$tfr$$tfr$

The time interval taken by the diode to switch from forward biased state to reverse biased state is called as **Reverse Recovery Time.** $tfr$$tfr$

To understand this more clearly, let us try to analyze what happens once the voltage is applied to a switching PN diode.

Carrier Concentration

Minority charge carrier concentration reduces exponentially as seen away from the junction. When the voltage is applied, due to the forward biased condition, the majority carriers of one side move towards the other. They become minority carriers of the other side. This concentration will be more at the junction.

For example, if N-type is considered, the excess of holes that enter into N-type after applying forward bias, adds to the already present minority carriers of N-type material.

Let us consider few notations.

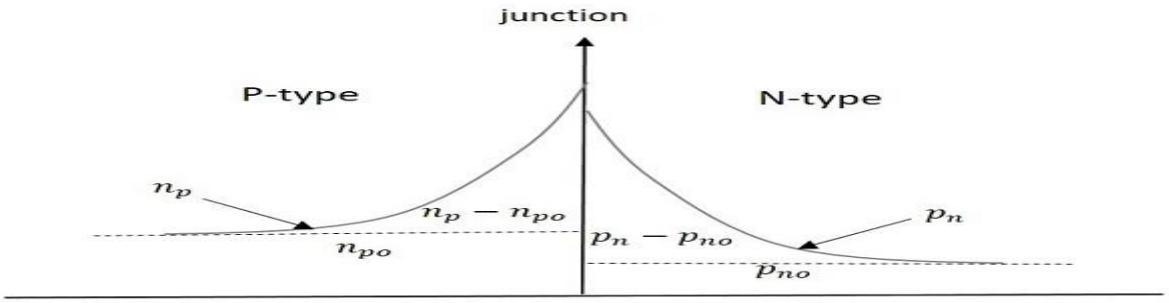
The majority carriers in P-type holesholes = PpoPpo

The majority carriers in N-type electronselectrons = NnoNno

The minority carriers in P-type electronselectrons = NpoNpo

The majority carriers in N-type holesholes = PnoPno

During Forward biased Condition − The minority carriers are more near junction and less far away from the junction. The graph below explains this.

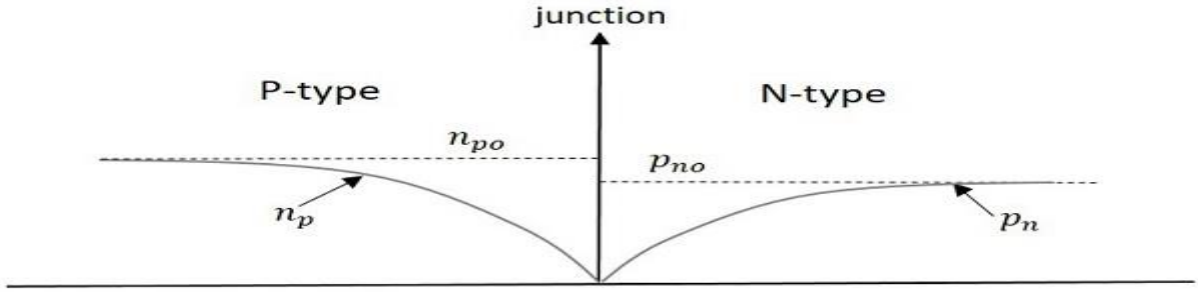


Excess minority carrier charge in P-type = Pn−PnoPn−Pno with pnopno steadystatevaluesteadystatevalue

Excess minority carrier charge in N-type = Np−NpoNp−Npo with NpoNpo steadystatevaluesteadystatevalue

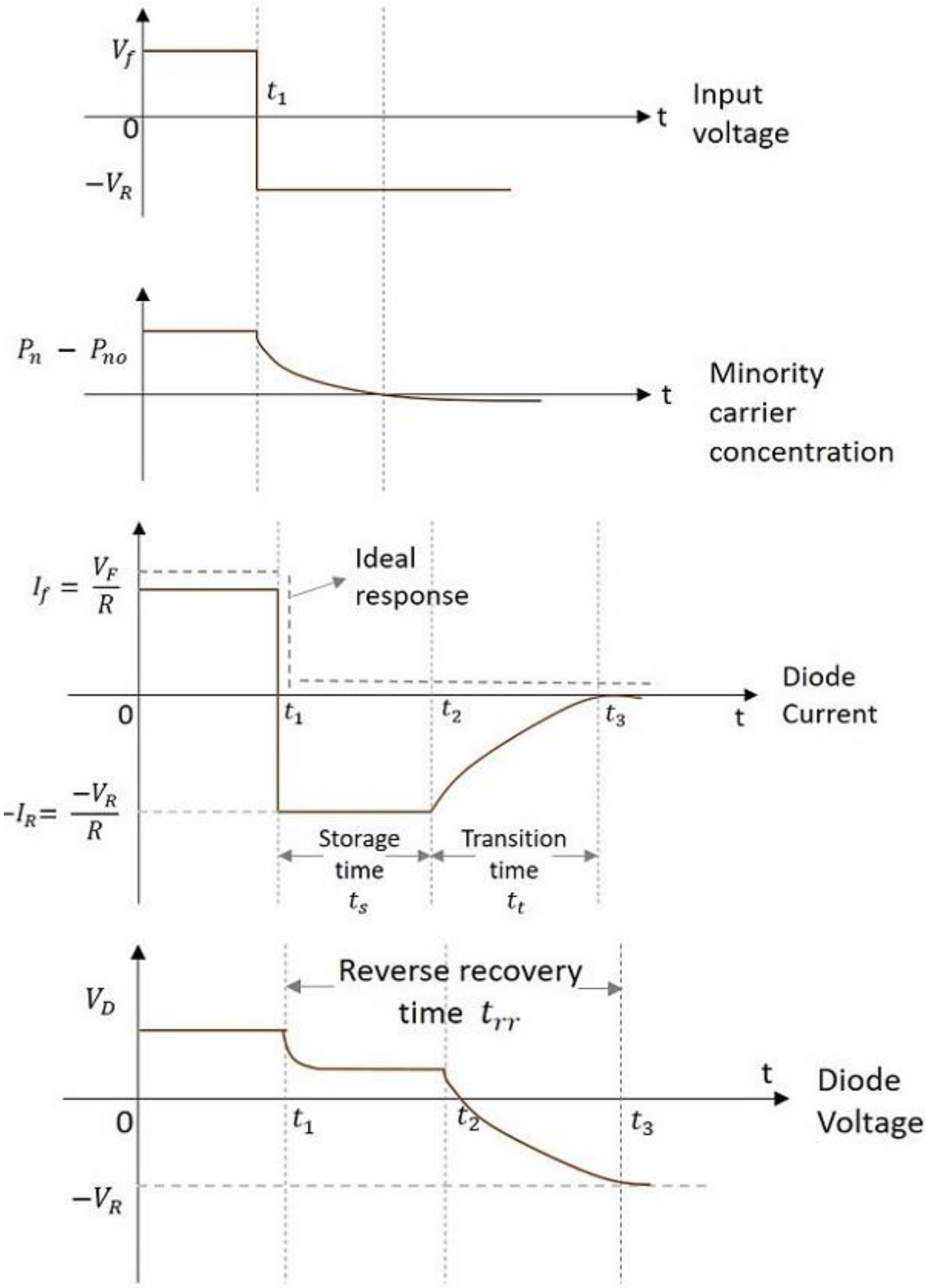
During reverse bias condition − Majority carriers doesn't conduct the current through the junction and hence don't participate in current condition. The switching diode behaves as a short circuited for an instance in reverse direction.

The minority carriers will cross the junction and conduct the current, which is called as **Reverse Saturation Current**. The following graph represents the condition during reverse bias.



In the above figure, the dotted line represents equilibrium values and solid lines represent actual values. As the current due to minority charge carriers is large enough to conduct, the circuit will be ON until this excess charge is removed.

The time required for the diode to change from forward bias to reverse bias is called **Reverse recovery time** $trr$$trr$. The following graphs explain the diode switching times in detail.



From the above figure, let us consider the diode current graph.

At t1t1 the diode is suddenly brought to OFF state from ON state; it is known as Storage time. **Storage time** is the time required to remove the excess minority carrier charge. The

negative current flowing from N to P type material is of a considerable amount during the Storage time. This negative current is,

−IR=−VRR−IR=−VRR

The next time period is the **transition time**" from$t2$to$t3$from$t2$to$t3$

Transition time is the time taken for the diode to get completely to open circuit condition. After t3t3 diode will be in steady state reverse bias condition. Before t1t1 diode is under steady state forward bias condition.

So, the time taken to get completely to open circuit condition is

Reverserecoverytime(trr)=Storagetime(Ts)+Transitiontime(Tt)

Whereas to get to ON condition from OFF, it takes less time called as **Forward recovery time**. Reverse recovery time is greater than Forward recovery time. A diode works as a better switch if this Reverse recovery time is made less.

Definitions

Let us just go through the definitions of the time periods discussed.

Storage time − The time period for which the diode remains in the conduction state even in the reverse biased state, is called as **Storage time**.

Transition time − The time elapsed in returning back to the state of non-conduction, i.e. steady state reverse bias, is called **Transition time**.

Reverse recovery time − The time required for the diode to change from forward bias to reverse bias is called as **Reverse recovery time**.

Forward recovery time − The time required for the diode to change from reverse bias to forward bias is called as **Forward recovery time**.

Factors that affect diode switching times

There are few factors that affect the diode switching times, such as

Diode Capacitance − The PN junction capacitance changes depending upon the bias conditions.

Diode Resistance − The resistance offered by the diode to change its state.

Doping Concentration − The level of doping of the diode, affects the diode switching times.

Depletion Width − The narrower the width of the depletion layer, the faster the switching will be. A Zener diode has narrow depletion region than an avalanche diode, which makes the former a better switch.

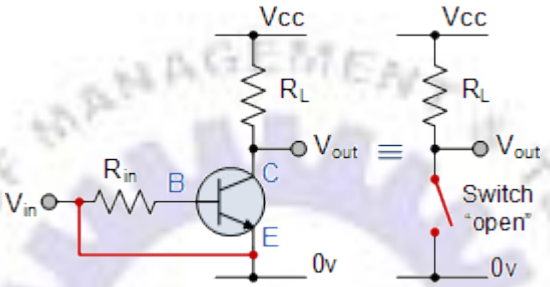
Applications

There are many applications in which diode switching circuits are used, such as −

1. High speed rectifying circuits

2. High speed switching circuits
3. RF receivers
4. General purpose applications
5. Consumer applications
6. Automotive applications
7. Telecom applications etc.

Transistor as a Switch.



Transistor switches can be used to switch a low voltage DC device (e.g. LED's) ON or OFF by using a transistor in its saturated or cut-off state.

When used as an AC signal amplifier, the transistors Base biasing voltage is applied in such a way that it always operates within its "active" region, that is the linear part of the output characteristics curves are used.

However, both the NPN & PNP type bipolar transistors can be made to operate as "ON/OFF" type solid state switch by biasing the transistors Base terminal differently to that for a signal amplifier.

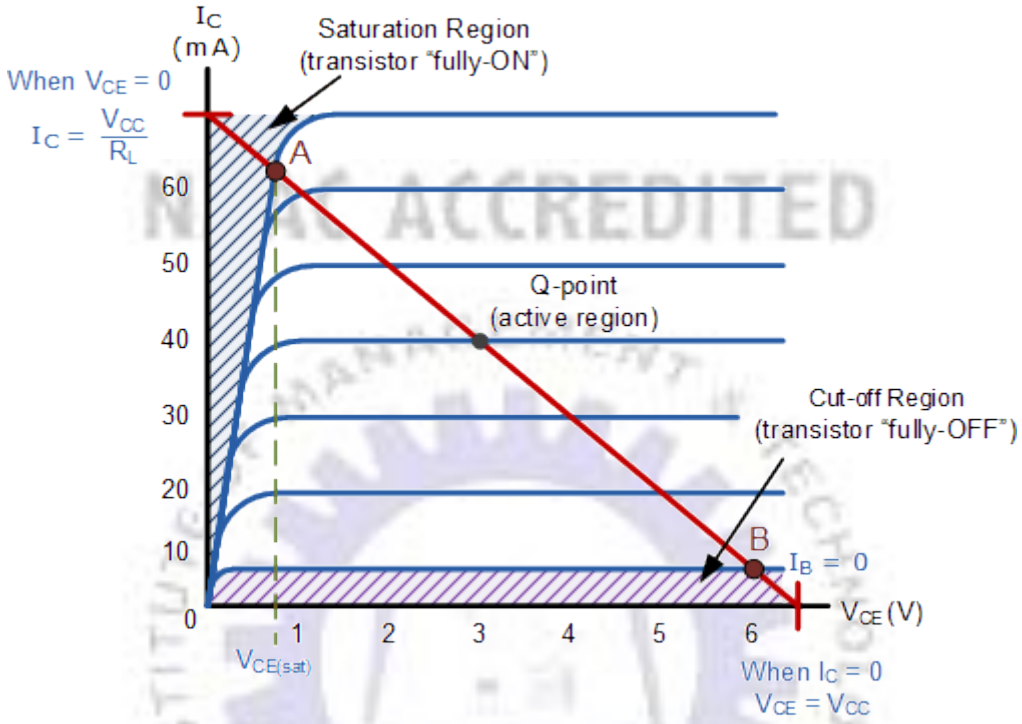
Solid state switches are one of the main applications for the use of transistor to switch a DC output "ON" or "OFF". Some output devices, such as LED's only require a few milliamps at logic level DC voltages and can therefore be driven directly by the output of a logic gate. However, high power devices such as motors, solenoids or lamps, often require more power than that supplied by an ordinary logic gate so transistor switches are used.

If the circuit uses the **Bipolar Transistor as a Switch**, then the biasing of the transistor, either NPN or PNP is arranged to operate the transistor at both sides of the " I-V " characteristics curves we have seen previously.

The areas of operation for a transistor switch are known as the **Saturation Region** and the **Cut-off Region**. This means then that we can ignore the operating Q-point biasing and voltage divider circuitry required for amplification, and use the transistor as a switch by

driving it back and forth between its "fully-OFF" (cut-off) and "fully-ON" (saturation) regions as shown below.

Operating Regions

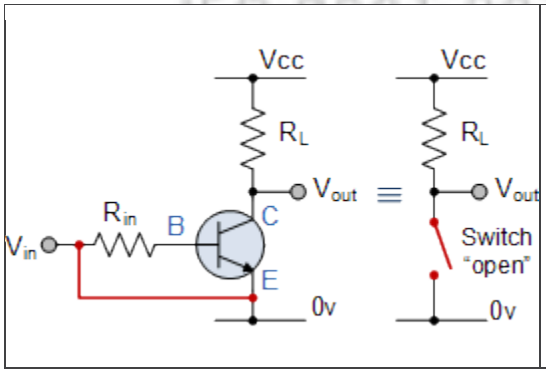


The pink shaded area at the bottom of the curves represents the "Cut-off" region while the blue area to the left represents the "Saturation" region of the transistor. Both these transistor regions are defined as:

1. Cut-off Region

Here the operating conditions of the transistor are zero input base current (I_B), zero output collector current (I_C) and maximum collector voltage (V_{CE}) which results in a large depletion layer and no current flowing through the device. Therefore the transistor is switched "Fully-OFF".

Cut-off Characteristics



- The input and Base are grounded (0v)
- Base-Emitter voltage $V_{BE} < 0.7v$
- Base-Emitter junction is reverse biased
- Base-Collector junction is reverse biased
- Transistor is "fully-OFF" (Cut-off region)
- No Collector current flows ($I_C = 0$)

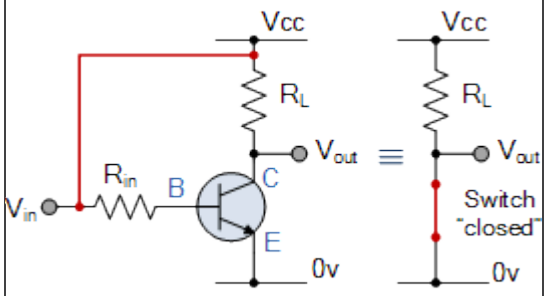
	• $V_{OUT} = V_{CE} = V_{CC} = ”1″$ • Transistor operates as an “open switch”

Then we can define the “cut-off region” or “OFF mode” when using a bipolar transistor as a switch as being, both junctions reverse biased, $V_B < 0.7v$ and $I_C = 0$. For a PNP transistor, the Emitter potential must be negative with respect to the Base.

2. Saturation Region

Here the transistor will be biased so that the maximum amount of base current is applied, resulting in maximum collector current resulting in the minimum collector emitter voltage drop which results in the depletion layer being as small as possible and maximum current flowing through the transistor. Therefore the transistor is switched “Fully-ON”.

Saturation Characteristics

	• The input and Base are connected to V_{CC} • Base-Emitter voltage $V_{BE} > 0.7v$ • Base-Emitter junction is forward biased • Base-Collector junction is forward biased • Transistor is “fully-ON” (saturation region) • Max Collector current flows ($I_C = Vcc/R_L$) • $V_{CE} = 0$ (ideal saturation) • $V_{OUT} = V_{CE} = ”0″$ • Transistor operates as a “closed switch”

Then we can define the “saturation region” or “ON mode” when using a bipolar transistor as a switch as being, both junctions forward biased, $V_B > 0.7v$ and I_C = Maximum. For a PNP transistor, the Emitter potential must be positive with respect to the Base.

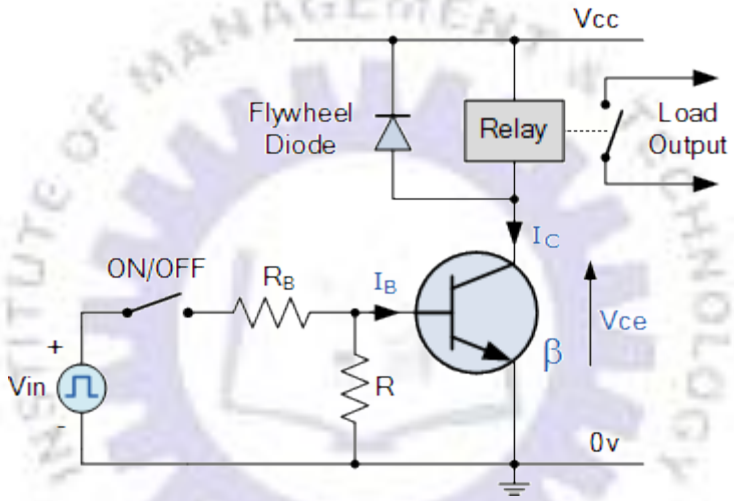
Then the transistor operates as a “single-pole single-throw” (SPST) solid state switch. With a zero signal applied to the Base of the transistor it turns “OFF” acting like an open switch and zero collector current flows. With a positive signal applied to the Base of the transistor it turns “ON” acting like a closed switch and maximum circuit current flows through the device.

The simplest way to switch moderate to high amounts of power is to use the transistor with an open-collector output and the transistors Emitter terminal connected directly to ground.

When used in this way, the transistors open collector output can thus “sink” an externally supplied voltage to ground thereby controlling any connected load.

An example of an NPN Transistor as a switch being used to operate a relay is given below. With inductive loads such as relays or solenoids a flywheel diode is placed across the load to dissipate the back EMF generated by the inductive load when the transistor switches “OFF” and so protect the transistor from damage. If the load is of a very high current or voltage nature, such as motors, heaters etc, then the load current can be controlled via a suitable relay as shown.

Basic NPN Transistor Switching Circuit



The circuit resembles that of the *Common Emitter* circuit we looked at in the previous tutorials. The difference this time is that to operate the transistor as a switch the transistor needs to be turned either fully “OFF” (cut-off) or fully “ON” (saturated). An ideal transistor switch would have infinite circuit resistance between the Collector and Emitter when turned “fully-OFF” resulting in zero current flowing through it and zero resistance between the Collector and Emitter when turned “fully-ON”, resulting in maximum current flow.

In practice when the transistor is turned “OFF”, small leakage currents flow through the transistor and when fully “ON” the device has a low resistance value causing a small saturation voltage (V_{CE}) across it. Even though the transistor is not a perfect switch, in both the cut-off and saturation regions the power dissipated by the transistor is at its minimum.

In order for the Base current to flow, the Base input terminal must be made more positive than the Emitter by increasing it above the 0.7 volts needed for a silicon device. By varying this Base-Emitter voltage V_{BE}, the Base current is also altered and which in turn controls the amount of Collector current flowing through the transistor as previously discussed.

When maximum Collector current flows the transistor is said to be **Saturated**. The value of the Base resistor determines how much input voltage is required and corresponding Base current to switch the transistor fully "ON".

Transistor as a Switch Example No1

Using the transistor values from the previous tutorials of: β = 200, Ic = 4mA and Ib = 20uA, find the value of the Base resistor (Rb) required to switch the load fully "ON" when the input terminal voltage exceeds 2.5v.

$$R_B = \frac{V_{in} - V_{BE}}{I_B} = \frac{2.5v - 0.7v}{20x10^{-6}} = 90k\Omega$$

The next lowest preferred value is: 82kΩ, this guarantees the transistor switch is always saturated.

Transistor as a Switch Example No2

Again using the same values, find the minimum Base current required to turn the transistor "fully-ON" (saturated) for a load that requires 200mA of current when the input voltage is increased to 5.0V. Also calculate the new value of Rb.

Transistor Base current:

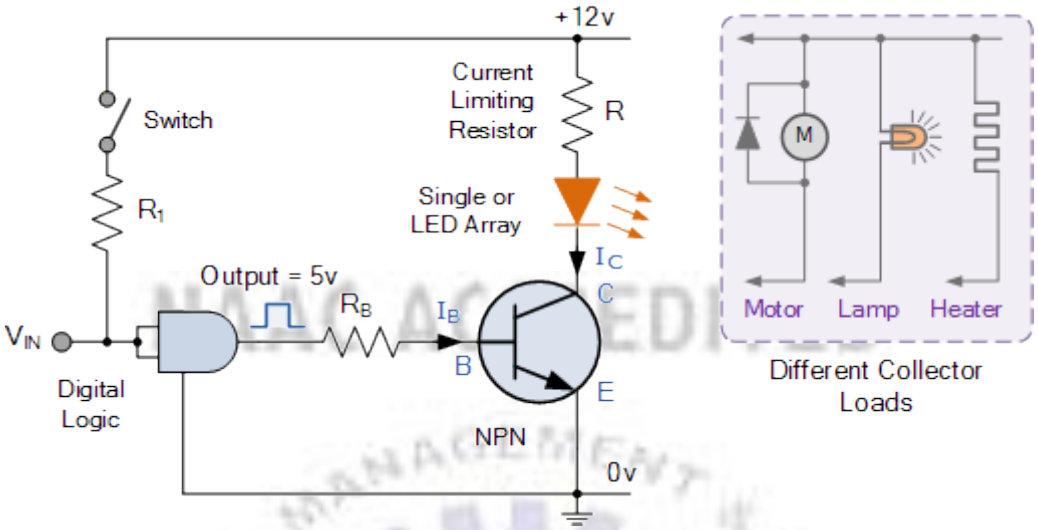
$$I_B = \frac{I_C}{\beta} = \frac{200mA}{200} = 1mA$$

Transistor Base resistance:

$$R_B = \frac{V_{in} - V_{BE}}{I_B} = \frac{5.0v - 0.7v}{1\times10^{-3}} = 4.3k\Omega$$

Transistor switches are used for a wide variety of applications such as interfacing large current or high voltage devices like motors, relays or lamps to low voltage digital IC's or logic gates like AND gates or OR gates. Here, the output from a digital logic gate is only +5v but the device to be controlled may require a 12 or even 24 volts supply. Or the load such as a DC Motor may need to have its speed controlled using a series of pulses (Pulse Width Modulation). transistor switches will allow us to do this faster and more easily than with conventional mechanical switches.

Digital Logic Transistor Switch

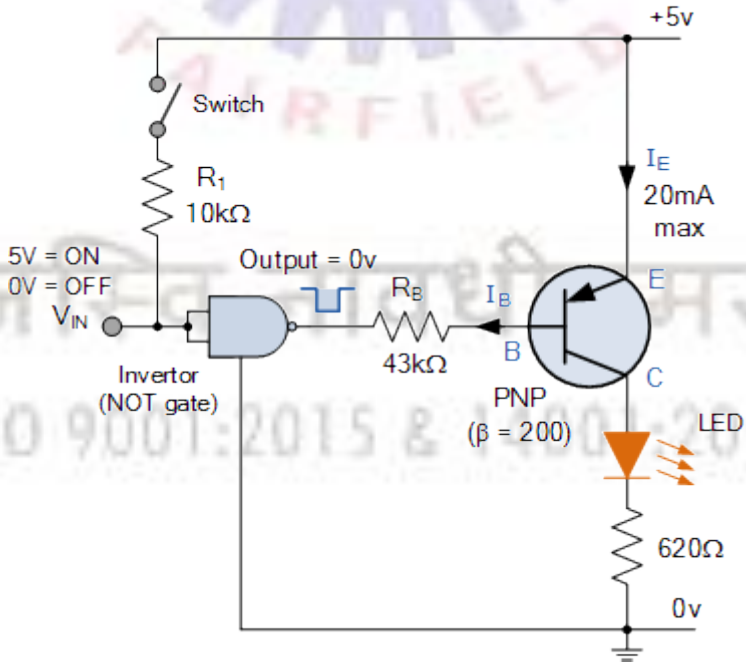


The base resistor, Rb is required to limit the output current from the logic gate.

PNP Transistor Switch

We can also use the PNP Transistors as a switch, the difference this time is that the load is connected to ground (0v) and the PNP transistor switches the power to it. To turn the PNP transistor operating as a switch "ON", the Base terminal is connected to ground or zero volts (LOW) as shown.

PNP Transistor Switching Circuit



The equations for calculating the Base resistance, Collector current and voltages are exactly the same as for the previous NPN transistor switch. The difference this time is that we are switching power with a PNP transistor (sourcing current) instead of switching ground with an NPN transistor (sinking current).

Darlington Transistor Switch

Sometimes the DC current gain of the bipolar transistor is too low to directly switch the load current or voltage, so multiple switching transistors are used. Here, one small input transistor is used to switch "ON" or "OFF" a much larger current handling output transistor. To maximise the signal gain, the two transistors are connected in a "Complementary Gain Compounding Configuration" or what is more commonly called a "**Darlington Configuration**" were the amplification factor is the product of the two individual transistors.

Darlington Transistors simply contain two individual bipolar NPN or PNP type transistors connected together so that the current gain of the first transistor is multiplied with that of the current gain of the second transistor to produce a device which acts like a single transistor with a very high current gain for a much smaller Base current. The overall current gain Beta (β) or hfe value of a Darlington device is the product of the two individual gains of the transistors and is given as:

$$\beta_{TOTAL} = \beta_1 \times \beta_2$$

So Darlington Transistors with very high β values and high Collector currents are possible compared to a single transistor switch. For example, if the first input transistor has a current gain of 100 and the second switching transistor has a current gain of 50 then the total current gain will be 100 * 50 = 5000. So for example, if our load current from above is 200mA, then the darlington base current is only 200mA/5000 = 40uA. A huge reduction from the previous 1mA for a single transistor.

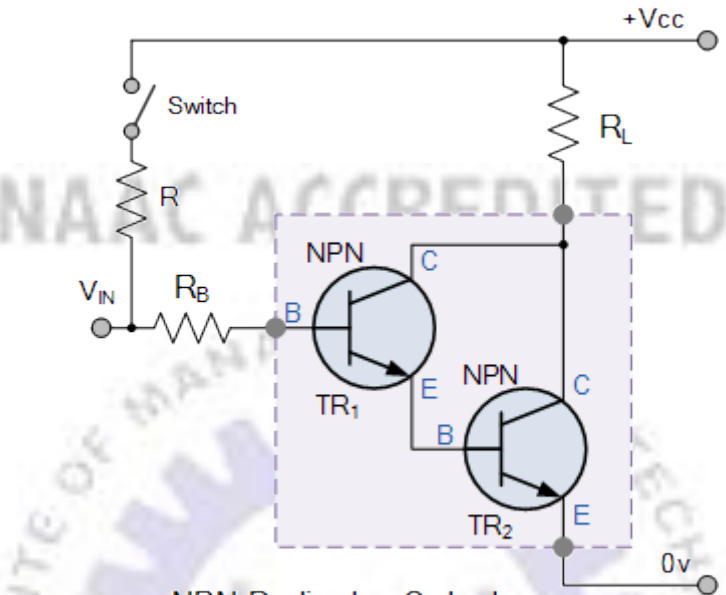
An example of the two basic types of Darlington transistor configurations are given below.

Darlington Transistor Configurations

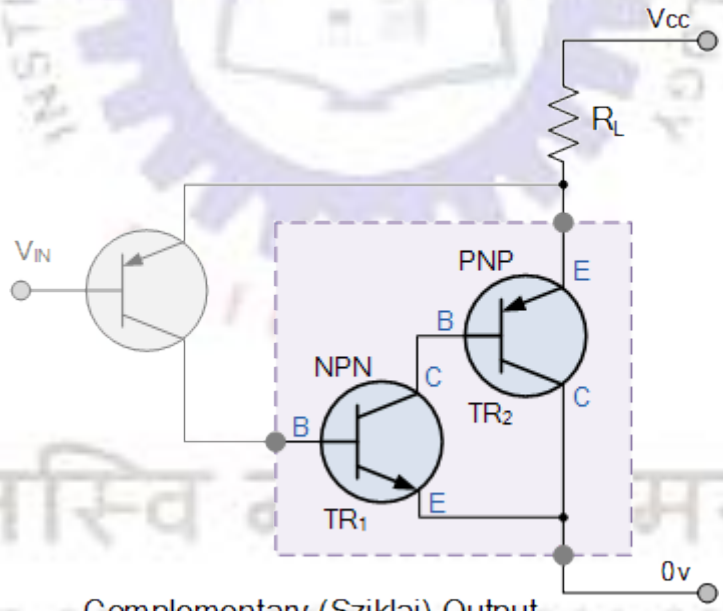
The above NPN Darlington transistor switch configuration shows the Collectors of the two transistors connected together with the Emitter of the first transistor connected to the Base terminal of the second transistor therefore, the Emitter current of the first transistor becomes the Base current of the second transistor switching it "ON".

The first or "input" transistor receives the input signal to its Base. This transistor amplifies it in the usual way and uses it to drive the second larger "output" transistors. The second transistor amplifies the signal again resulting in a very high current gain. One of the main

characteristics of **Darlington Transistors** is their high current gains compared to single bipolar transistors.



NPN Darlington Output



Complementary (Sziklai) Output

As well as its high increased current and voltage switching capabilities, another advantage of a "Darlington Transistor Switch" is in its high switching speeds making them ideal for use in inverter circuits, lighting circuits and DC motor or stepper motor control applications.

One difference to consider when using Darlington transistors over the conventional single bipolar types when using the transistor as a switch is that the Base-Emitter input voltage (V_{BE}) needs to be higher at approx 1.4v for silicon devices, due to the series connection of the two PN junctions.

Transistor as a Switch Summary

Then to summaries when using a **Transistor as a Switch** the following conditions apply:

Transistor switches can be used to switch and control lamps, relays or even motors.

When using the bipolar transistor as a switch they must be either “fully-OFF” or “fully-ON”.

Transistors that are fully “ON” are said to be in their **Saturation** region.

Transistors that are fully “OFF” are said to be in their **Cut-off** region.

When using the transistor as a switch, a small Base current controls a much larger Collector load current.

When using transistors to switch inductive loads such as relays and solenoids, a “Flywheel Diode” is used.

When large currents or voltages need to be controlled, **Darlington Transistors** can be used.

Q3. Explain the logic families in digital electronics

In computer engineering, a logic family may refer to one of two related concepts. A logic family of monolithic digital integrated circuit devices is a group of electronic logic gates constructed using one of several different designs, usually with compatible logic levels and power supply characteristics within a family. Many logic families were produced as individual components, each containing one or a few related basic logical functions, which could be used as "building-blocks" to create systems or as so-called "glue" to interconnect more complex integrated circuits. A "logic family" may also refer to a set of techniques used to implement logic within VLSI integrated circuits such as central processors, memories, or other complex functions. Some such logic families use static techniques to minimize design complexity. Other such logic families, such as domino logic, use clocked dynamic techniques to minimize size, power consumption and delay.

Before the widespread use of integrated circuits, various solid-state and vacuum-tube logic systems were used but these were never as standardized and interoperable as the integrated-circuit devices. The most common logic family in modern semiconductor devices is metal–oxide–semiconductor (MOS) logic, due to low power consumption, small transistor sizes, and high transistor density.

Different circuit configurations and production technologies are used during the production of digital integrated circuits. Each of these approaches is called a specific **Logic Families**. Now the idea of having different approaches or different logic families is that each ICs of same family when fabricated will have identical electrical characteristics. The characteristics which are bound to be identical are supply voltage range, speed of response, dissipation of power, input and output logic levels, current sinking capability, current sourcing capability, noise margin, fan-out etc.

Significance of Logic Families

When we talk about digital systems actually the digital ICs are the ones which make up the whole system. And if all the ICs are of same logic family then they are compatible to each other and the intended logic functions are performed and the goal is achieved. But in case ICs belonging to **different logic families** are used in a digital system then to ensure compatibility interfacing techniques must be used. And that is the reason why we must understand different logic families and use the best combination of ICs during the design of a digital system. Now the question arises what might be the consequence of choosing wrong combinations of ICs. The answer is that it may not match the necessary capability needed.

Types of Logic Family

The digital ICs are designed using any of either bipolar devices or MOS or a combination of both. The logic families which fall under the first kind are called bipolar families, this include diode logic (DL), emitted coupled logic (ECL), resistor transistor logic (RTL), diode transistor logic (DTL), transistor transistor logic (TTL). The members of other logic family i.e. MOS family are PMOS, NMOS family, CMOS family. Now the Bi-MOS logic family is the one that uses both bipolar and MOS devices.

Of the above mentioned families DL, RTL and DTL are not used these days they have become obsolete. TTL, CMOS, ECL, NMOS and Bi-CMOS are the families which are still used. We will discuss about few of them in this article.

TTL subfamilies.

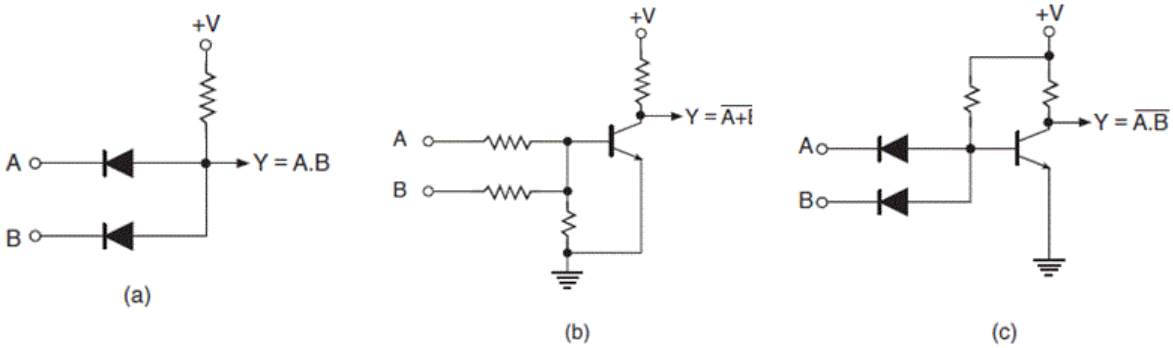
The TTL family consists of various subfamilies such as standard TTL, low-power TTL, high power TTL, low power Schottky TTL, Schottky TTL, advanced low-power Schottky TTL, advanced Schottky TTL and fast TTL. The ICs which belong to TTL family are designated as follows – 74 or 54 for standard TTL, 74L or 54L for low-power TTL, 74H or 54H for high power TTL, 74LS or 54LS for Low power schottky TTL and so on.

CMOS subfamilies

This is a popular logic family which includes 4000A, 4000B, 4000UB, 54/74C, 54/74HC, 54/74HCT, 54/74AC and 54/74ACT families. The subfamilies are divided on the basis of voltage difference and other parameters.

ECL Subfamilies

ECL stands for Emitter Coupled Logic family and it was introduced by ON semiconductor in 1962. The first product launched of this family was MECL-1 series. Later MECL-II, MECL-III, MECL-10K, MECL-10H series came into existence.



(a) Diode logic (b) resistor transistor logic and (c) diode transistor logic.

Resistor-Transistor Logic (RTL):

Resistor-transistor logic gates use Transistors to combine multiple input signals, which also amplify and invert the resulting combined signal. Often an additional transistor is included to

re-invert the output signal. This combination provides clean output signals and either inversion or non-inversion as needed.

RTL gates are almost as simple as DL gates, and remain inexpensive. They also are handy because both normal and inverted signals are often available. However, they do draw a significant amount of current from the power supply for each gate. Another limitation is that RTL gates cannot switch at the high speeds used by today's computers, although they are still useful in slower applications.

Although they are not designed for linear operation, RTL integrated circuits are sometimes used as inexpensive small-signal amplifiers, or as interface devices between linear and digital circuits.

RTL Logic Circuit:

Resistor-transistor logic (RTL) is a class of digital circuits built using resistors as the input network and bipolar junction transistors (BJTs) as switching devices. RTL is the earliest class of transistorized digital logic circuit used; other classes include diode-transistor logic (DTL) and transistor-transistor logic (TTL).

Advantages of RTL Logic circuit:

The primary advantage of RTL technology was that it involved a minimum number of transistors, which was an important consideration before integrated circuit technology (that is, in circuits using discrete components), as transistors were the most expensive component to produce. Early IC logic production (such as Fairchild's in 1961) used the same approach briefly, but quickly transitioned to higher-performance circuits such as diode-transistor logic and then transistor-transistor logic (starting 1963 at Sylvania), since diodes and transistors were no more expensive than resistors in the IC.

Limitations:

The obvious disadvantage of RTL is its high current dissipation when the transistor conducts to overdrive the output biasing resistor. This requires that more current be supplied to and heat be removed from RTL circuits. In contrast, TTL circuits minimize both of these requirements.

Lancaster says that integrated circuit RTL NOR gates (which have one transistor per input) may be constructed with "any reasonable number" of logic inputs, and gives an example of an 8-input NOR gate.

A standard integrated circuit RTL NOR gate can drive up to 3 other similar gates. Alternatively, it has enough output to drive up to 2 standard integrated circuit RTL "buffers", each of which can drive up to 25 other standard RTL NOR gates.

Diode-Transistor Logic (DTL):

By letting diodes perform the logical AND or OR function and then amplifying the result with a transistor, we can avoid some of the limitations of RTL. DTL takes diode logic gates and adds a transistor to the output, in order to provide logic inversion and to restore the signal to full logic levels.

Diode-transistor logic

Diode-Transistor Logic (DTL) is a class of digital circuits built from bipolar junction transistors (BJT), diodes and resistors; it is the direct ancestor of transistor-transistor logic. It is called diode-transistor logic because the logic gating function (e.g., AND) is performed by a diode network and the amplifying function is performed by a transistor (contrast this with RTL and TTL).

Operation:

With the simplified circuit shown in the picture the negative bias voltage at the base is required to prevent unstable or invalid operation. In an integrated circuit version of the gate, two diodes replace R3 to prevent any base current when one or more inputs are at low logic level. Alternatively to increase fan-out of the gate an additional transistor and diode may be used. The IBM 1401 used DTL circuits almost identical to this simplified circuit, but solved the base bias level problem mentioned above by alternating NPN and PNP based gates operating on different power supply voltages instead of adding extra diodes.

Advantages of DTL:

One advantage of digital circuits when compared to analog circuits is that signals represented digitally can be transmitted without degradation due to noise. For example, a continuous audio signal, transmitted as a sequence of 1s and 0s, can be reconstructed without error provided the noise picked up in transmission is not enough to prevent identification of the 1s and 0s. An hour of music can be stored on a compact disc as about 6 billion binary digits.

In a digital system, a more precise representation of a signal can be obtained by using more binary digits to represent it. While this requires more digital circuits to process the signals, each digit is handled by the same kind of hardware. In an analog system, additional resolution requires fundamental improvements in the linearity and noise charactersitics of each step of the signal chain.

Computer-controlled digital systems can be controlled by software, allowing new functions to be added without changing hardware. Often this can be done outside of the factory by updating the product's software. So, the product's design errors can be corrected after the product is in a customer's hands.

Information storage can be easier in digital systems than in analog ones. The noise-immunity of digital systems permits data to be stored and retrieved without degradation. In an analog system, noise from aging and wear degrade the information stored. In a digital system, as long as the total noise is below a certain level, the information can be recovered perfectly.

Disadvantages:

In some cases, digital circuits use more energy than analog circuits to accomplish the same tasks, thus producing more heat. In portable or battery-powered systems this can limit use of digital systems.

For example, battery-powered cellular telephones often use a low-power analog front-end to amplify and tune in the radio signals from the base station. However, a base station has grid power and can use power-hungry, but very flexible software radios. Such base stations can be easily reprogrammed to process the signals used in new cellular standards.

Digital circuits are sometimes more expensive, especially in small quantities.

The sensed world is analog, and signals from this world are analog quantities. For example, light, temperature, sound, electrical conductivity, electric and magnetic fields are analog. Most useful digital systems must translate from continuous analog signals to discrete digital signals. This causes quantization errors. Quantization error can be reduced if the system stores enough digital data to represent the signal to the desired degree of fidelity. The Nyquist-Shannon sampling theorem provides an important guideline as to how much digital data is needed to accurately portray a given analog signal.

In some systems, if a single piece of digital data is lost or misinterpreted, the meaning of large blocks of related data can completely change. Because of the cliff effect, it can be difficult for users to tell if a particular system is right on the edge of failure, or if it can tolerate much more noise before failing.

Digital fragility can be reduced by designing a digital system for robustness. For example, a parity bit or other error management method can be inserted into the signal path. These schemes help the system detect errors, and then either correct the errors, or at least ask for a new copy of the data. In a state-machine, the state transition logic can be designed to catch unused states and trigger a reset sequence or other error recovery routine.

Embedded software designs that employ Immunity Aware Programming, such as the practice of filling unused program memory with interrupt instructions that point to an error recovery routine. This helps guard against failures that corrupt the microcontroller's instruction pointer which could otherwise cause random code to be executed. Digital memory and transmission

systems can use techniques such as error detection and correction to use additional data to correct any errors in transmission and storage.

On the other hand, some techniques used in digital systems make those systems more vulnerable to single-bit errors. These techniques are acceptable when the underlying bits are reliable enough that such errors are highly unlikely.

TTL Logic Circuit:

Transistor-transistor logic (TTL) is a class of digital circuits built from bipolar junction transistors (BJT) and resistors. It is called transistor-transistor logic because both the logic gating function (e.g., AND) and the amplifying function are performed by transistors (contrast this with RTL and DTL).

TTL is notable for being a widespread integrated circuit (IC) family used in many applications such as computers, industrial controls, test equipment and instrumentation, consumer electronics, synthesizers, etc. The designation TTL is sometimes used to mean TTL-compatible logic levels, even when not associated directly with TTL integrated circuits, for example as a label on the inputs and outputs of electronic instruments.

*TTL contrasts with the preceding resistor-transistor logic (RTL) and diode-transistor logic (DTL) generations by using transistors not only to amplify the output but also to isolate the inputs. The p-n junction of a diode has considerable capacitance, so changing the logic level of an input connected to a diode, as in DTL, requires considerable time and energy.

As shown in the top schematic at right, the fundamental concept of TTL is to isolate the inputs by using a common-base connection, and amplify the function using a common emitter connection. Note that the base of the output transistor is driven high only by the forward-biased base-collector junction of the input transistor. The second schematic adds to this a "totem-pole output". When V2 is off (output equals 1), the resistors turn V3 on and V4 off, resulting in a stronger 1 output. When V2 is on, it activates V4, driving 0 to the output. The diode forces the emitter of V3 to ~0.7 V, while V4 base-emitter junction and V2 collector-emitter junction pull its base to a voltage ~0.7, turning it off. By removing pull-up and pull-down resistors from the output stage, this allows the strength of the gate to be increased without proportionally affecting power consumption.

TTL is particularly well suited to integrated circuits because the inputs of a gate may all be integrated into a single base region to form a multiple-emitter transistor. Such a highly customized part might increase the cost of a circuit where each transistor is in a separate package, but, by combining several small on-chip components into one larger device, it conversely reduces the cost of implementation on an IC.

As with all bipolar logic, a small current must be drawn from a TTL input to ensure proper logic levels. The total current drawn must be within the capacities of the preceding stage, which limits the number of nodes that can be connected (the fanout).

All standardized common TTL circuits operate with a 5-volt power supply. A TTL input signal is defined as "low" when between 0V and 0.8V with respect to the ground terminal, and "high" when between 2.2V and 5V (precise logic levels vary slightly between sub-types). TTL outputs are typically restricted to narrower limits of between 0V and 0.4V for a "low" and between 2.6V and 5V for a "high", providing 0.4V of noise immunity. Standardization of the TTL levels was so ubiquitous that complex circuit boards often contained TTL chips made by many different manufacturers selected for availability and cost, compatibility being assured; two circuit board units off the same assembly line on different successive days or weeks might have a different mix of brands of chips in the same positions on the board; repair was possible with chips manufactured years (sometimes over a decade) later than original components. Within usefully broad limits, logic gates could be treated as ideal Boolean devices without concern for electrical limitations.

Advantages of TTL Logic circuit:

Advantages of TTL logic family, one should have a basic idea about RTL, DTL etc. Diode logic (DL) uses diodes to implement logical functions like AND and OR. But the disadvantage is that it can not perform NOT operation. As AND and OR are not complete functions by themselves, they can not perform several logic functions without NOT. Hence, there was a need for some device which can perform a NOT function as diodes can not. That device is a transistor. Then came the DTL which uses a transistor along with diodes. As a transistor can act as an inverter, NAND (NOT-AND) & NOR (NOT-OR) operations can be performed. But this logic uses several diodes which will slow down its operation. Due to the delay offered by them, the logic levels may sometimes change i. e. 0 t0 1 or 1 to 0. Then came TTL. This logic uses a multi emitter transistor, a transistor with many emitter terminals. As every emitter is nothing but a diode, this logic eliminates the use of all diodes. This is the major advantage.

As transistor becomes ON and OFF much rapidly than a diode, switching time will be faster.

TTL, or Transistor-transistor logic replaced resistor-transistor logic, and used much less power. The TTL family is very fast and reliable, and newer faster, less power-consuming, etc. types are always being developed.

In TTL (Transistor-Transistor Logic), think that the device using this technology is made from several transistors.

Q4. Explain Current and voltage parameters of logic families, FANIN FANOUT, Noise Margins

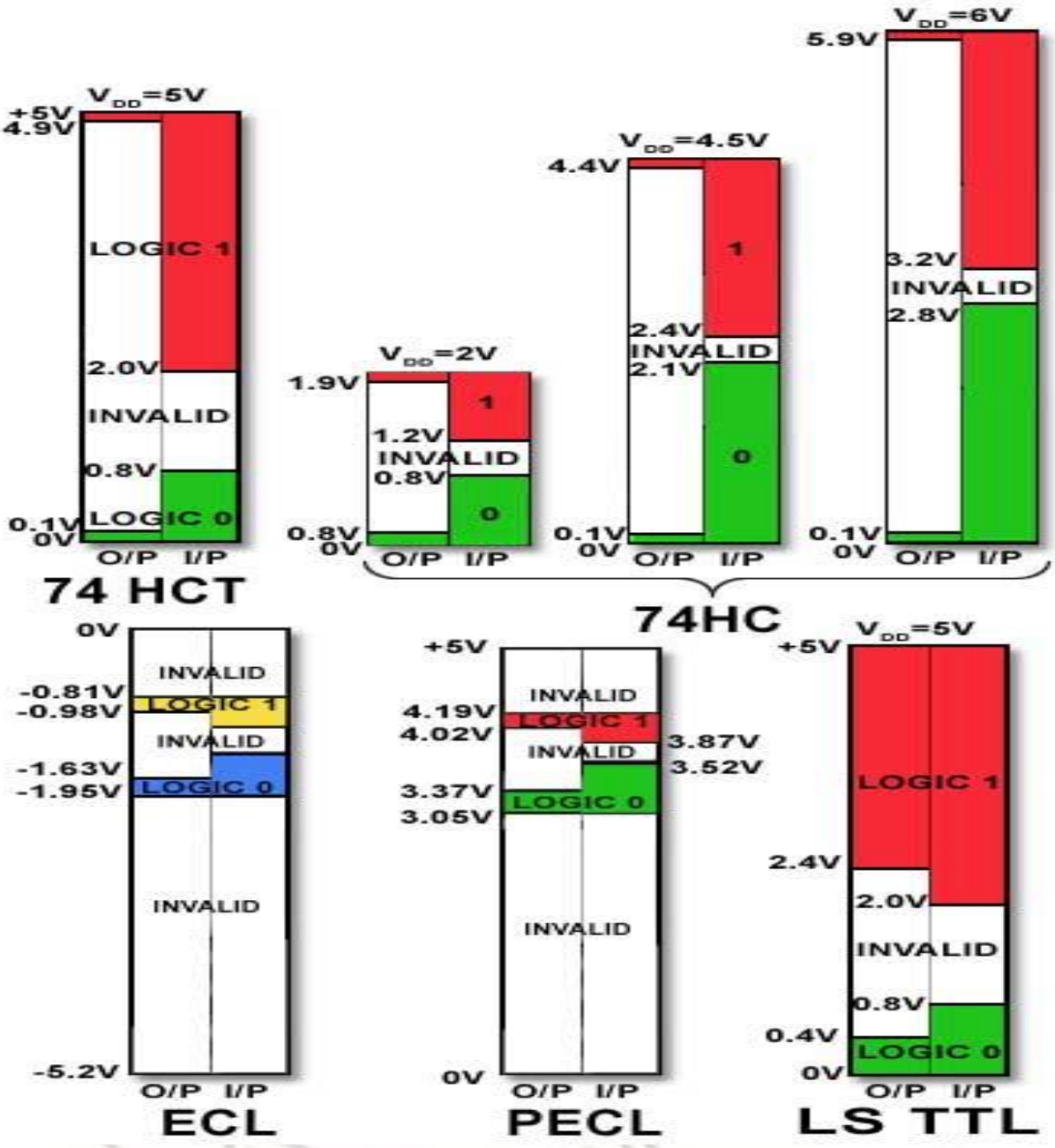
Current and voltage parameters of logic families

Logic 1 and Logic 0

Logic 1 and Logic 0 are not simply 5V and 0V or even Vcc and Ground. Within any family of ICs the voltages and currents indicating 1 and 0 cover defined ranges unique to that logic family. The range of voltages allowed for a particular logic level depends on the amount of current flowing into or out of the logic gate inputs or output, the larger the current the output is supplying, the lower the output voltage will be.

Each output will supply a certain amount of current before the output voltage falls too far to be called logic 1, and each gate input will need to be supplied with a certain amount of current to raise the input voltage sufficiently to be recognized as logic 1.

Examples of typical logic levels at inputs and outputs in a range of logic families are illustrated in Fig. 3.3.1. These levels are fairly standard throughout a particular family, although there can be minor differences in these and other parameters, between products from different manufacturers. In addition there are sub families within these families that may have different defined levels. When designing digital circuits, or replacing ICs in critical equipment, it is therefore essential to consult the appropriate manufacturer's data sheets.



Logic 1 levels for inputs and outputs are shown in red and logic 0 in green. To highlight the fact that true ECL gates, have negative logic levels, these colours have been changed to yellow and blue respectively.

Notice that the logic levels for outputs (left column) and inputs (right column) in all of the families are different. This ensures that provided that the output voltage of a gate is within its defined logic limits for 1 or 0, any compatible gate input connected to that output will recognise the correct 1 or 0 levels. The difference between levels at the output and input in any particular family is called the 'Noise Margin'.

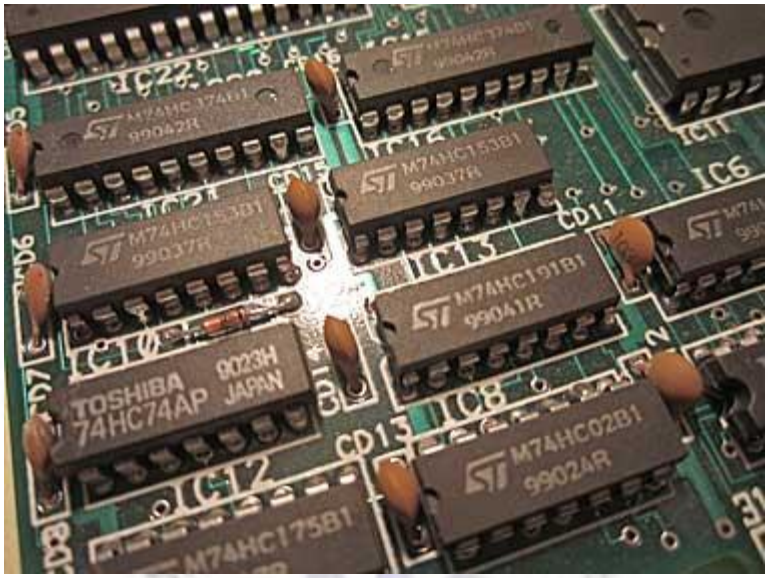


Fig. 3.3.2 Logic IC Decoupling

Noise Margin

Because voltages in digital circuits can be continually changing very rapidly between logic 1 and logic 0, (virtually between supply voltage and ground), they have the potential to produce a lot of noise, in the form of high frequency voltage spikes on the IC power supply lines.

To counteract this it is important to include effective decoupling, not only at the power supply unit, but also by connecting decoupling capacitors across the VDD and 0V connections at each IC. These capacitors are normally connected as physically close to the IC as possible, as shown in Fig. 3.3.2.

Fan In and Fan Out in Digital Electronics

Fan In and Fan Out are characteristics of Digital ICs. Digital ICs are complete functioning logic networks. Typically, a Digital IC requires only a power supply, I/P (input) and O/P (output). Here are the definitions of Fan In and Fan Out.

Fan In: The fan-in defined as the maximum number of inputs that a logic gate can accept. If number of input exceeds, the output will be undefined or incorrect. It is specified by manufacturer and is provided in the data sheet.

Fan Out: The fan-out is defined as the maximum number of inputs (load) that can be connected to the output of a gate without degrading the normal operation. Fan Out is calculated from the amount of current available in the output of a gate and the amount of current needed in each input of the connecting gate. It is specified by manufacturer and is provided in the data sheet. Exceeding the specified maximum load may cause a malfunction because the circuit will not be able supply the demanded power.

Q5. Explain Simplifying Boolean Expression using K Map

Simplifying Boolean Expression using K Map

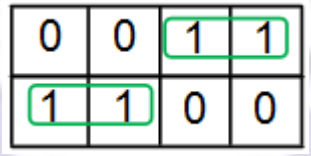
Minterm Solution of K Map

The following are the steps to obtain simplified minterm solution using K-map.

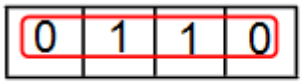
Step 1: Initiate Express the given expression in its canonical form

Step 2: Populate the K-map Enter the value of 'one' for each product-term into the K-map cell, while filling others with zeros.

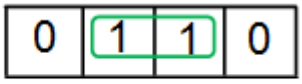
Step 3: Form Groups Consider the consecutive 'ones' in the K-map cells and group them (green boxes).



Each group should contain the largest number of 'ones' and no blank cell.



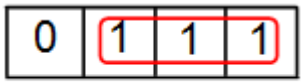
Incorrect



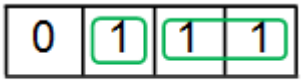
Correct

The number of 'ones' in a group must be a power of 2 i.e. a group can contain

$$16(=2^4)\ or\ 8(=2^3)\ or\ 4(=2^2)\ or\ 2(=2^1)\ or\ 1(=2^0)\ cells$$

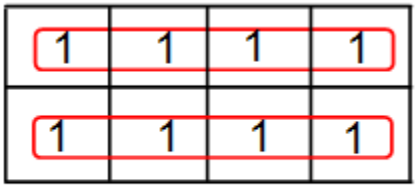


Incorrect

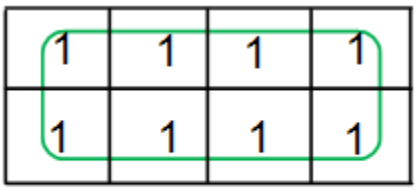


Correct

Grouping has to be carried-on in decreasing order meaning, one has to try to group for 8 (octet) first, then for 4 (quad), followed by 2 and lastly for 1 (isolated 'ones').

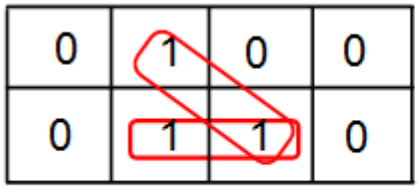


Incorrect

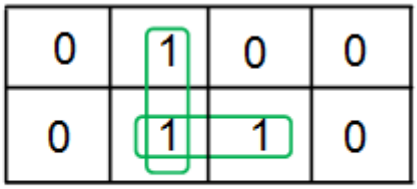


Correct

Grouping is to done either horizontally or vertically or interms of squares or rectangles. Diagonal grouping of 'ones' is not permitted.

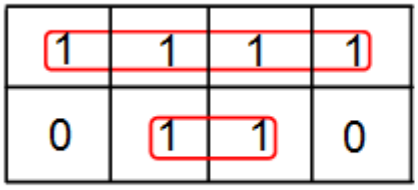


Incorrect

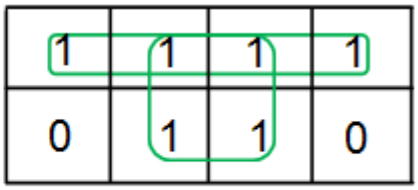


Correct

The same element(s) may repeat in multiple groups only if this increases the size of the group.

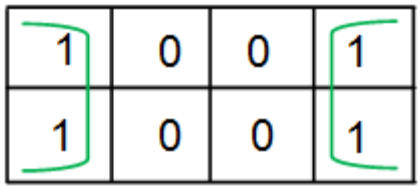


Incorrect

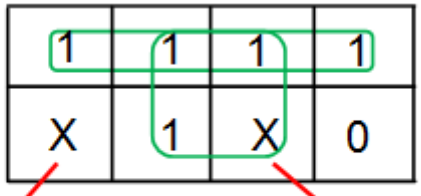


Correct

The elements around the edges of the table are considered to be adjacent and can be grouped together.



Don't care conditions are to be considered only if they aid in increasing the group-size (else neglected).



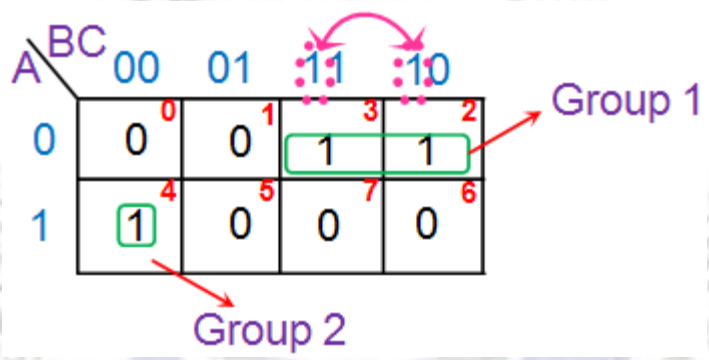
Neglect Consider

Step 4: Obtain Boolean Expression for Each Group

Express each group interms of input variables by looking at the common variables seen in

cell-labelling. For example in the figure shown below there are two groups with two and one number of 'ones' in them (Group 1 and Group 2, respectively). All the 'ones' in the Group 1 of the K-map are present in the row for which A = 0. Thus they contain the variable $\overline{A}$. Further these two 'ones' are present in adjacent columns which have only B term in common as indicated by the pink arrow in the figure.

Hence the next term is B. This yields the product term corresponding to this group as $\overline{A}B$. Similarly the 'one' in the Group 2 of the K-map is present in the row for which A = 1. Further the variables corresponding to its column are $\overline{B}\overline{C}$. Thus one gets the overall product-term for this group as $A\overline{B}\overline{C}$.

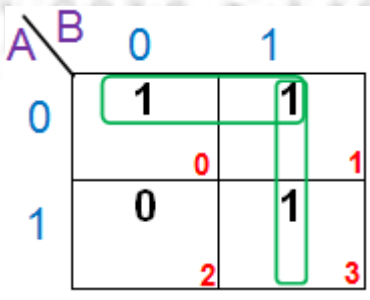


Step 5: Obtain Boolean Expression for the Output The product-terms obtained for individual groups are to be combined to form sum-of-product (SOP) form which yields the overall **simplified Boolean expression**. This means that for the K-map shown in Step 4, the overall simplified output expression is

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

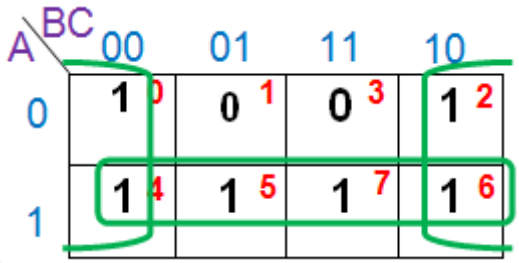
A few more examples elaborating K-map simplification process are shown below.

$$Example\ 1 : Y = \overline{A}\ \overline{B} + \overline{A}B + AB$$



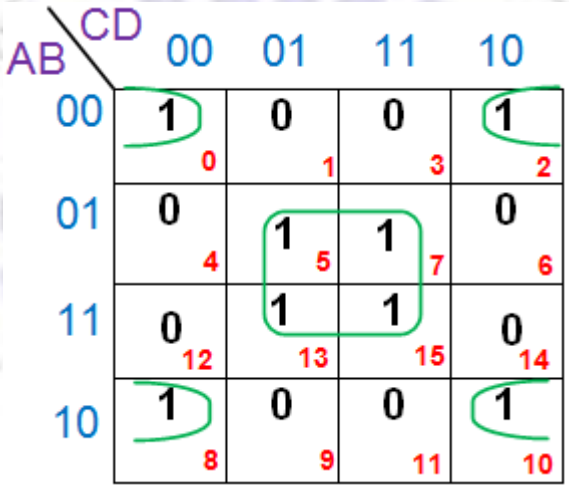
$$Simplified\ Expression : Y = \overline{A} + B$$

$$Example\ 2 : Y = \overline{A}\ \overline{B}\ \overline{C} + \overline{A}B\ \overline{C} + A\overline{B}\ \overline{C} + A\overline{B}C + AB\overline{C} + ABC$$



$$Simplified\ Expression : Y = A + \overline{C}$$

$$Example\ 3 : Y = \overline{A}\ \overline{B}\ \overline{C}\ \overline{D} + \overline{A}\ \overline{B}C\ \overline{D} + \overline{A}BC\ \overline{D} + \overline{A}BCD + A\overline{B}\ \overline{C}\ \overline{D} + A\overline{B}C\overline{D} + ABC\overline{D} + ABCD$$



$$Simplified\ Expression : Y = BD + \overline{B}\ \overline{D}$$

Maxterm Solution of K Map

The method to be followed in order to obtain simplified **maxterm solution** using K-map is similar to that for **minterm solution** except minor changes listed below.

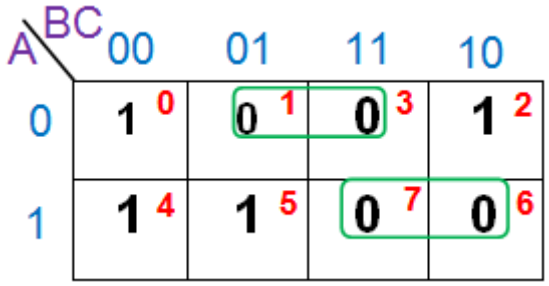
K-map cells are to be populated by 'zeros' for each sum-term of the expression instead of 'ones'.

Grouping is to be carried-on for 'zeros' and not for 'ones'.

Boolean expressions for each group are to be expressed as sum-terms and not as product-terms.

Sum-terms of all individual groups are to be combined to obtain the overall simplified Boolean expression in product-of-sums (POS) form.

$$Example : Y = (A + B + \overline{C}) + (A + \overline{B} + \overline{C}) + (\overline{A} + \overline{B} + C) + (\overline{A} + \overline{B} + \overline{C})$$



$$Simplified\ Expression: Y = (A + \overline{C})(\overline{A} + \overline{B})$$

Unit 2

Q.1 Explain Parity Generator and Parity Check

What is Parity Bit?

The parity generating technique is one of the most widely used error detection techniques for the data transmission. In digital systems, when binary data is transmitted and processed , data may be subjected to noise so that such noise can alter 0s (of data bits) to 1s and 1s to 0s.

Hence, **parity bit** is added to the word containing data in order to make number of 1s either even or odd. Thus it is used to detect errors , during the transmission of binary data .The message containing the data bits along with parity bit is transmitted from transmitter node to receiver node.

At the receiving end, the number of 1s in the message is counted and if it doesn't match with the transmitted one, then it means there is an error in the data.

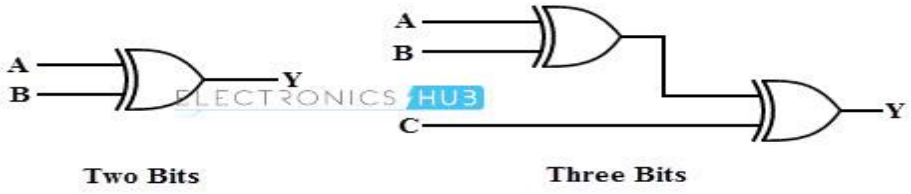
Parity generator and checker

A parity generator is a combinational logic circuit that generates the parity bit in the transmitter. On the other hand, a circuit that checks the parity in the receiver is called parity checker. A combined circuit or devices of parity generators and parity checkers are commonly used in digital systems to detect the single bit errors in the transmitted data word.

The sum of the data bits and parity bits can be even or odd . In even parity, the added parity bit will make the total number of 1s an even amount whereas in odd parity the added parity bit will make the total number of 1s odd amount.

The basic principle involved in the implementation of parity circuits is that sum of odd number of 1s is always 1 and sum of even number of 1s is always zero. Such error detecting and correction can be implemented by using Ex-OR gates (since Ex-OR gate produce zero output when there are even number of inputs).

To produce two bits sum, one Ex-OR gate is sufficient whereas for adding three bits two Ex-OR gates are required as shown in below figure.



Parity Generator

It is combinational circuit that accepts an n-1 bit stream data and generates the additional bit that is to be transmitted with the bit stream. This additional or extra bit is termed as a parity bit.

In **even parity** bit scheme, the parity bit is '**0**' if there are **even number of 1s** in the data stream and the parity bit is '**1**' if there are **odd number of 1s** in the data stream.

In **odd parity** bit scheme, the parity bit is '**1**' if there are **even number of 1**s in the data stream and the parity bit is '**0**' if there are **odd number of 1s** in the data stream. Let us discuss both even and odd parity generators.

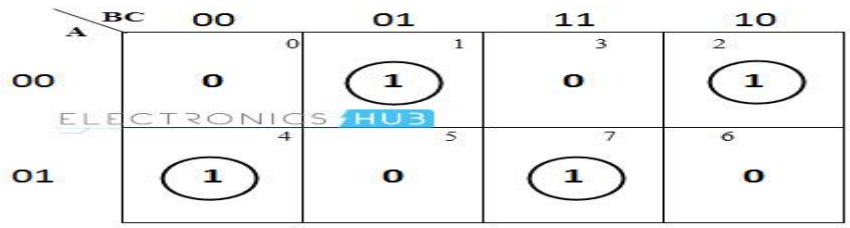
Even Parity Generator

Let us assume that a 3-bit message is to be transmitted with an even parity bit. Let the three inputs A, B and C are applied to the circuits and output bit is the parity bit P. The total number of 1s must be even, to generate the even parity bit P.

The figure below shows the truth table of even parity generator in which 1 is placed as parity bit in order to make all 1s as even when the number of 1s in the truth table is odd.

3-bit message			Even parity bit generator (P)
A	B	C	Y
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	1

The K-map simplification for 3-bit message even parity generator is



From the above truth table, the simplified expression of the parity bit can be written as

$$P = \bar{A}\,\bar{B}\,C + \bar{A}\,B\,\bar{C} + A\,\bar{B}\,\bar{C} + A\,B\,C$$

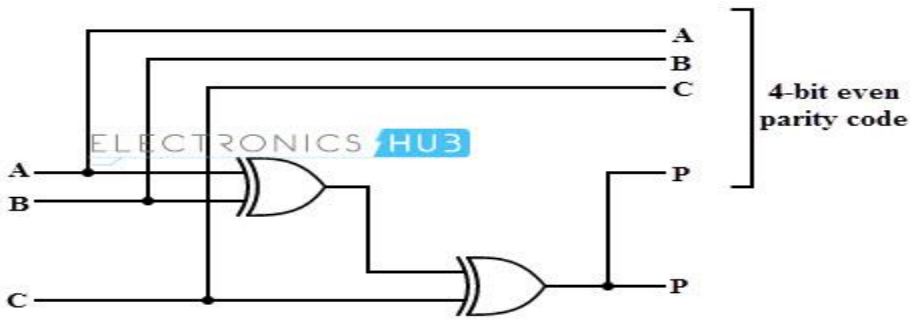
$$= \bar{A}\,(\bar{B}\,C + B\,\bar{C}) + A\,(\bar{B}\,\bar{C} + B\,C)$$

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

$$P = A \oplus B \oplus C$$

The above expression can be implemented by using two Ex-OR gates. The logic diagram of even parity generator with two Ex – OR gates is shown below. The three bit message along with the parity generated by this circuit which is transmitted to the receiving end where parity checker circuit checks whether any error is present or not.

To generate the even parity bit for a 4-bit data, three Ex-OR gates are required to add the 4-bits and their sum will be the parity bit.



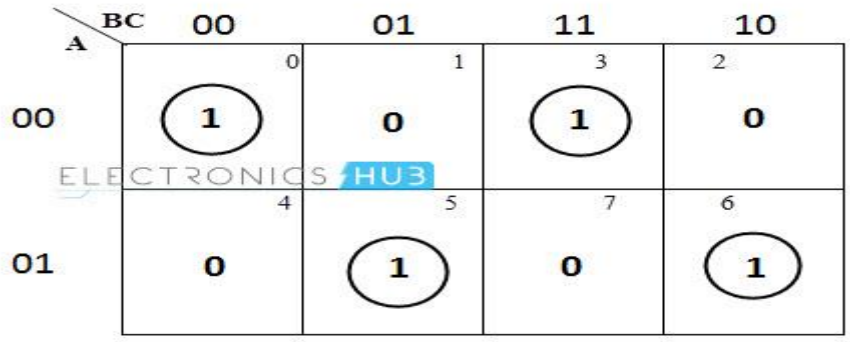
Odd Parity Generator

Let us consider that the 3-bit data is to be transmitted with an odd parity bit. The three inputs are A, B and C and P is the output parity bit. The total number of bits must be odd in order to generate the odd parity bit.

In the given truth table below, 1 is placed in the parity bit in order to make the total number of bits odd when the total number of 1s in the truth table is even.

3-bit message			Odd parity bit generator (P)
A	B	C	Y
0	0	0	1
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	0

The truth table of the odd parity generator can be simplified by using K-map as

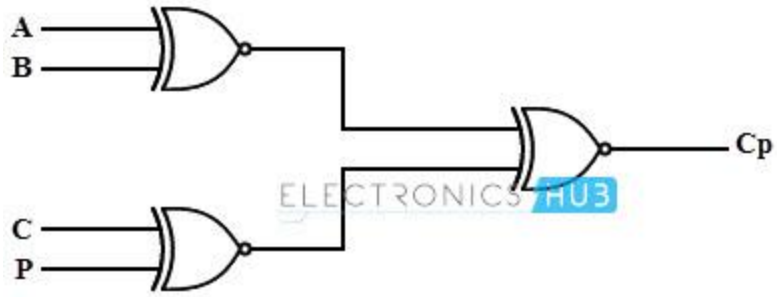


The output parity bit expression for this generator circuit is obtained as

P = A ⊕ B Ex-NOR C

The above Boolean expression can be implemented by using one Ex-OR gate and one Ex-NOR gate in order to design a 3-bit odd parity generator.

The logic circuit of this generator is shown in below figure , in which . two inputs are applied at one Ex-OR gate, and this Ex-OR output and third input is applied to the Ex-NOR gate , to produce the odd parity bit. It is also possible to design this circuit by using two Ex-OR gates and one NOT gate.



Parity Check

It is a logic circuit that checks for possible errors in the transmission. This circuit can be an even parity checker or odd parity checker depending on the type of parity generated at the transmission end. When this circuit is used as even parity checker, the number of input bits must always be even.

When a parity error occurs, the ‘sum even’ output goes low and ‘sum odd’ output goes high. If this logic circuit is used as an odd parity checker, the number of input bits should be odd, but if an error occurs the ‘sum odd’ output goes low and ‘sum even’ output goes high.

Even Parity Checker

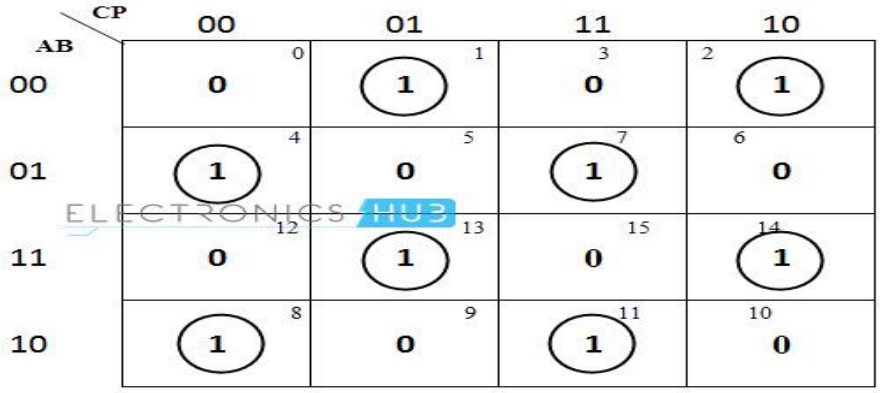
Consider that three input message along with even parity bit is generated at the transmitting end. These 4 bits are applied as input to the parity checker circuit which checks the possibility of error on the data. Since the data is transmitted with even parity, four bits received at circuit must have an even number of 1s.

If any error occurs, the received message consists of odd number of 1s. The output of the parity checker is denoted by PEC (parity error check).

The below table shows the truth table for the even parity checker in which PEC = 1 if the error occurs, i.e., the four bits received have odd number of 1s and PEC = 0 if no error occurs, i.e., if the 4-bit message has even number of 1s.

4-bit received message				Parity error check C_p
A	B	C	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 above truth table can be simplified using K-map as shown below.



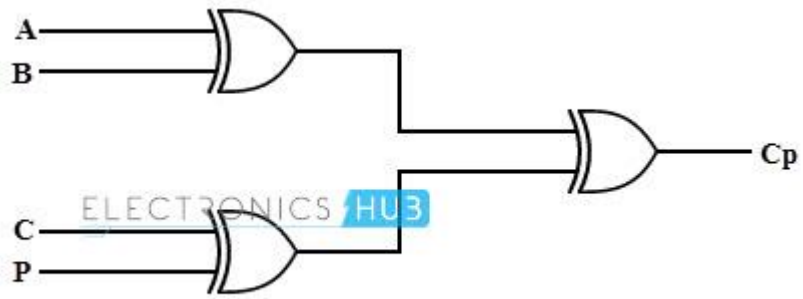
$$PEC = \overline{A}\ \overline{B}\ (\overline{C}\ D + C\ \overline{D}) + \overline{A}\ B\ (\overline{C}\ \overline{D} + C\ D) + A\ B\ (\overline{C}\ D + C\ \overline{D}) + A\ \overline{B}\ (\overline{C}\ \overline{D} + C\ D)$$

$$= \overline{A}\ \overline{B}\ (C \oplus D) + \overline{A}\ B\ (\overline{C \oplus D}) + A\ B\ (C \oplus D) + A\ \overline{B}\ (\overline{C \oplus D})$$

$$= (\overline{A}\ \overline{B} + A\ B)\ (C \oplus D) + (\overline{A}\ B + A\ \overline{B})\ (\overline{C \oplus D})$$

$$= (A \oplus B) \oplus (C \oplus D)$$

The above logic expression for the even parity checker can be implemented by using three Ex-OR gates as shown in figure. If the received message consists of five bits, then one more Ex-OR gate is required for the even parity checking.



Odd Parity Checker

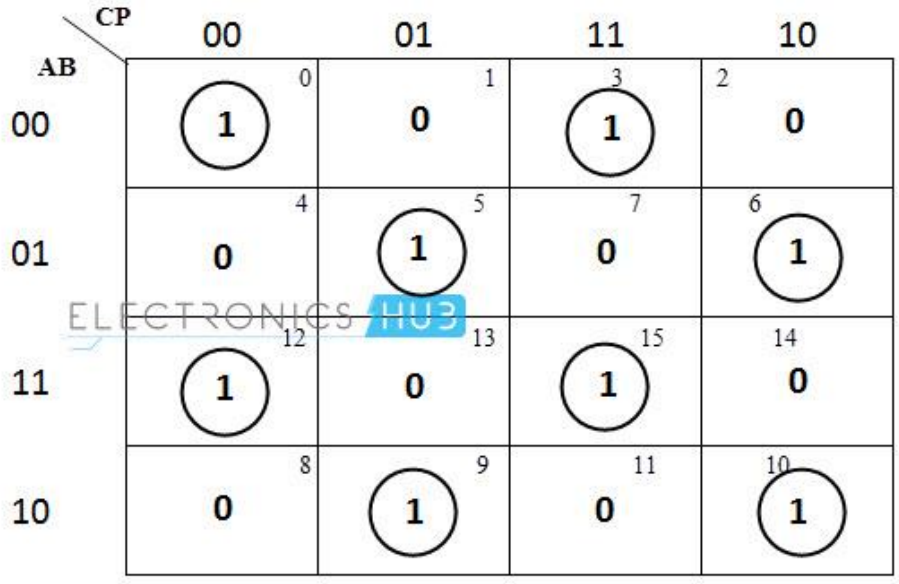
Consider that a three bit message along with odd parity bit is transmitted at the transmitting end. Odd parity checker circuit receives these 4 bits and checks whether any error are present in the data.

If the total number of 1s in the data is odd, then it indicates no error, whereas if the total number of 1s is even then it indicates the error since the data is transmitted with odd parity at transmitting end.

The below figure shows the truth table for odd parity generator where **PEC =1** if the 4-bit message received consists of **even number of 1s** (hence the error occurred) and **PEC= 0** if the message contains **odd number of 1s** (that means no error).

4-bit received message				Parity error check C_p
A	B	C	P	
0	0	0	0	1
0	0	0	1	0
0	0	1	0	0
0	0	1	1	1
0	1	0	0	0
0	1	0	1	1
0	1	1	0	1
0	1	1	1	0
1	0	0	0	0
1	0	0	1	1
1	0	1	0	1
1	0	1	1	0
1	1	0	0	1
1	1	0	1	0
1	1	1	0	0
1	1	1	1	1

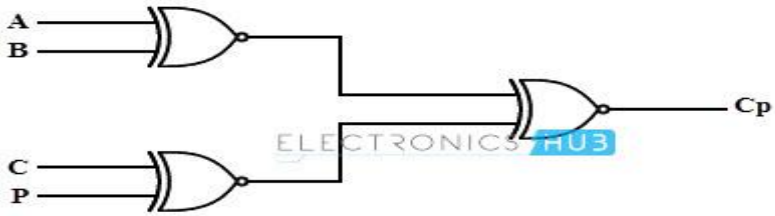
The expression for the PEC in the above truth table can be simplified by K-map as shown below.



After simplification, the final expression for the PEC is obtained as

PEC = (A Ex-NOR B) Ex-NOR (C Ex-NOR D)

The expression for the odd parity checker can be designed by using three Ex-NOR gates as shown below.

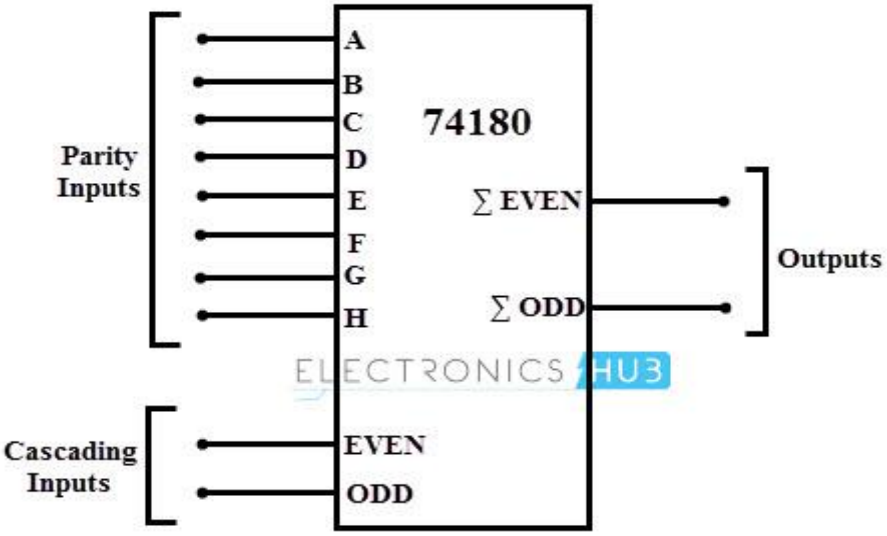


Parity Generator/Checker ICs

There are different types of parity generator /checker ICs are available with different input configurations such as 5-bit, 4-bit, 9-bit, 12-bit, etc. A most commonly used and standard type of parity generator/checker IC is 74180.

It is a 9-bit parity generator or checker used to detect errors in high speed data transmission or data retrieval systems. The figure below shows the pin diagram of 74180 IC.

This IC can be used to generate a 9-bit odd or even parity code or it can be used to check for odd or even parity in a 9-bit code (8 data bits and one parity bit).



This IC consists of eight parity inputs from A through H and two cascading inputs. There are two outputs even sum and odd sum. In implementing generator or checker circuits, unused parity bits must be tied to logic zero and the cascading inputs must not be equal.

Q2. Write a Short note on Half Adder & Full Adder

Binary Adder

The most basic arithmetic operation is addition. The circuit, which performs the addition of two binary numbers is known as **Binary adder**. First, let us implement an adder, which performs the addition of two bits.

Half Adder

Half adder is a combinational circuit, which performs the addition of two binary numbers A and B are of **single bit**. It produces two outputs sum, S & carry, C.

The **Truth table** of Half adder is shown below.

Inputs		**Outputs**	
A	B	C	S
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

When we do the addition of two bits, the resultant sum can have the values ranging from 0 to 2 in decimal. We can represent the decimal digits 0 and 1 with single bit in binary. But, we can't represent decimal digit 2 with single bit in binary. So, we require two bits for representing it in binary.

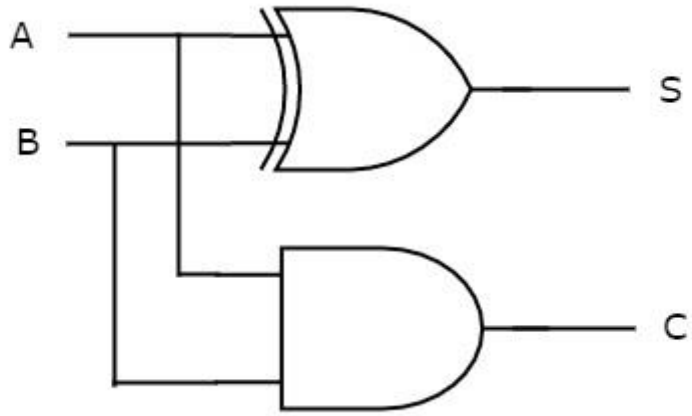
Let, sum, S is the Least significant bit and carry, C is the Most significant bit of the resultant sum. For first three combinations of inputs, carry, C is zero and the value of S will be either zero or one based on the **number of ones** present at the inputs. But, for last combination of inputs, carry, C is one and sum, S is zero, since the resultant sum is two.

From Truth table, we can directly write the **Boolean functions** for each output as

S=A⊕BS=A⊕B

C=ABC=AB

We can implement the above functions with 2-input Ex-OR gate & 2-input AND gate. The **circuit diagram** of Half adder is shown in the following figure.



In the above circuit, a two input Ex-OR gate & two input AND gate produces sum, S & carry, C respectively. Therefore, Half-adder performs the addition of two bits.

Full Adder

Full adder is a combinational circuit, which performs the **addition of three bits** A, B and C_{in}. Where, A & B are the two parallel significant bits and C_{in} is the carry bit, which is generated from previous stage. This Full adder also produces two outputs sum, S & carry, C_{out}, which are similar to Half adder.

The **Truth table** of Full adder is shown below.

Inputs			**Outputs**	
A	**B**	**C_{in}**	**C_{out}**	**S**
0	0	0	0	0
0	0	1	0	1

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

When we do the addition of three bits, the resultant sum can have the values ranging from 0 to 3 in decimal. We can represent the decimal digits 0 and 1 with single bit in binary. But, we can't represent the decimal digits 2 and 3 with single bit in binary. So, we require two bits for representing those two decimal digits in binary.

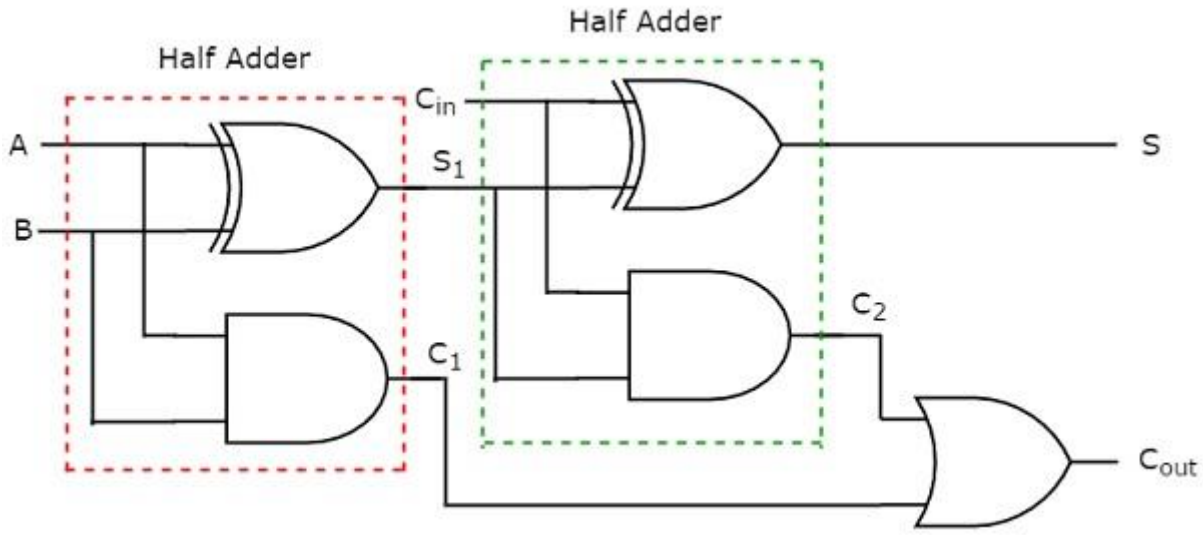
Let, sum, S is the Least significant bit and carry, C_{out} is the Most significant bit of resultant sum. It is easy to fill the values of outputs for all combinations of inputs in the truth table. Just count the **number of ones** present at the inputs and write the equivalent binary number at outputs. If C_{in} is equal to zero, then Full adder truth table is same as that of Half adder truth table.

We will get the following **Boolean functions** for each output after simplification.

$$S=A \oplus B \oplus CinS=A \oplus B \oplus Cin$$

$$cout=AB+(A \oplus B)cincout=AB+(A \oplus B)cin$$

The sum, S is equal to one, when odd number of ones present at the inputs. We know that Ex-OR gate produces an output, which is an odd function. So, we can use either two 2input Ex-OR gates or one 3-input Ex-OR gate in order to produce sum, S. We can implement carry, C_{out} using two 2-input AND gates & one OR gate. The **circuit diagram** of Full adder is shown in the following figure.



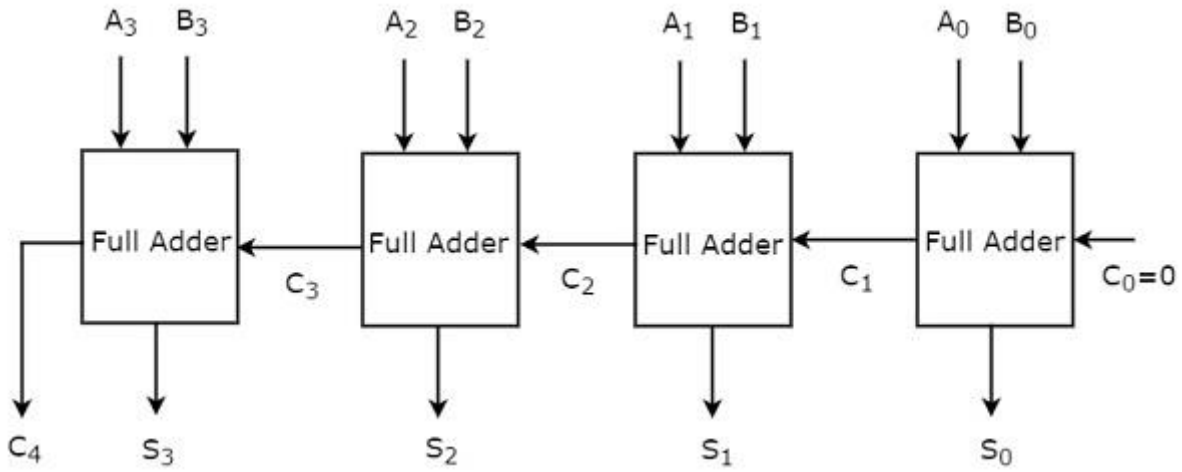
This adder is called as **Full adder** because for implementing one Full adder, we require two Half adders and one OR gate. If C_{in} is zero, then Full adder becomes Half adder. We can verify it easily from the above circuit diagram or from the Boolean functions of outputs of Full adder.

4-bit Binary Adder

The 4-bit binary adder performs the **addition of two 4-bit numbers**. Let the 4-bit binary numbers, A=A3A2A1A0A=A3A2A1A0 and B=B3B2B1B0B=B3B2B1B0. We can implement 4-bit binary adder in one of the two following ways.

- Use one Half adder for doing the addition of two Least significant bits and three Full adders for doing the addition of three higher significant bits.
- Use four Full adders for uniformity. Since, initial carry C_{in} is zero, the Full adder which is used for adding the least significant bits becomes Half adder.

For the time being, we considered second approach. The **block diagram** of 4-bit binary adder is shown in the following figure.



Here, the 4 Full adders are cascaded. Each Full adder is getting the respective bits of two parallel inputs A & B. The carry output of one Full adder will be the carry input of subsequent higher order Full adder. This 4-bit binary adder produces the resultant sum having at most 5 bits. So, carry out of last stage Full adder will be the MSB.

In this way, we can implement any higher order binary adder just by cascading the required number of Full adders. This binary adder is also called as **ripple carry** binarybinary **adder** because the carry propagates ripplesripples from one stage to the next stage.

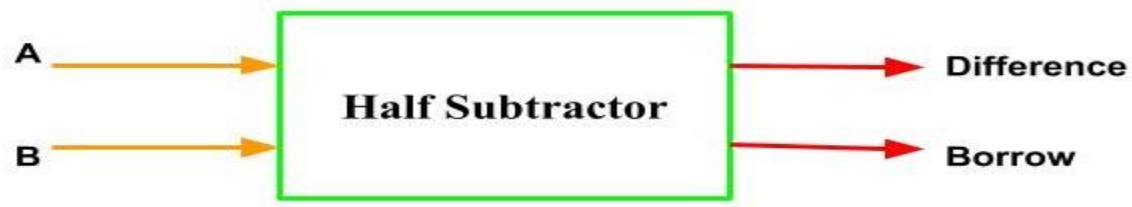
Q3. Explain Half Subtarctor & Full Subtarctor.

Half Subtractor Circuit Construction using Logic Gates

Half subtractor is the most essential combinational logic circuit which is used in digital electronics. Basically, this is an electronic device or in other terms, we can say it as a logic circuit. Half subtractor is used to perform two binary digits subtraction. In the previous article, we have already discussed the concepts of half adder and a full adder circuit which uses the binary numbers for the calculation. Similarly, the subtractor circuit uses binary numbers (0,1) for the subtraction. The circuit of the half subtractor can be built with two logic gates namely NAND and EX-OR gates. This circuit gives two elements such as the difference as well as the borrow. This article gives half subtractor theory concept which includes theories like what is a subtractor, half subtractor with the truth table, etc.

What is a Half Subtractor?

Before going to discuss the half subtractor, we have to know the binary subtraction. In binary subtraction, the process of subtraction is similar to arithmetic subtraction. In arithmetic subtraction the base 2 number system is used whereas in binary subtraction, binary numbers are used for subtraction. The resultant terms can be denoted with the difference and borrow.



Half Subtractor Block Diagram

As in binary subtraction, the major digit is 1, we can generate borrow while the subtrahend 1 is superior to minuend 0 and due to this, borrow will need. The following example gives the binary subtraction of two binary bits.

First Digit	Second Digit	Difference	Borrow
0	0	0	0

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

In the above subtraction, the two digits can be represented with A and B. These two digits can be subtracted and gives the resultant bits as difference and borrow.

When we observe the first two and fourth rows, the difference of these rows, then the difference and borrow are similar because subtrahend is lesser than the minuend. Similarly, when we observe the third row, the minuend value is subtracted from the subtrahend. So the difference and borrow bits are 1 because the subtrahend digit is superior to the minuend digit.

Half-Subtractor Block Diagram

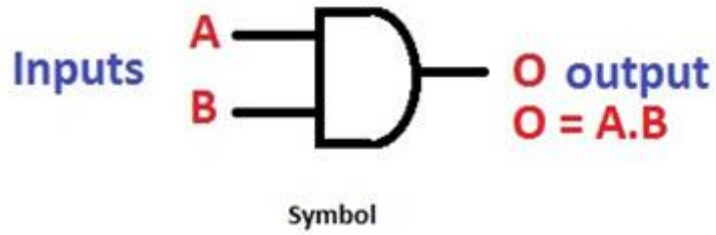
The block diagram of the half subtractor is shown above. It requires two inputs as well as gives two outputs. Here inputs are represented with A&B, and outputs are Difference and Borrow.

The above circuit can be designed with EX-OR & NAND gates. Here, NAND gate can be build by using AND and NOT gates. So we require three logic gates for making half subtractor circuit namely EX-OR gate, NOT gate, and NAND gate.

Combination of AND and NOT gate produce a different combined gate named as NAND Gate. The Ex-OR gate output will be the Diff bit and the NAND Gate output will be the Borrow bit for the same inputs A&B.

AND-Gate

The AND-gate is one type of digital logic gate with multiple inputs and a single output and based on the inputs combinations it will perform the logical conjunction. When all the inputs of this gate are high, then the output will be high otherwise the output will be low. The logic diagram of AND gate with truth table is shown below.



Symbol

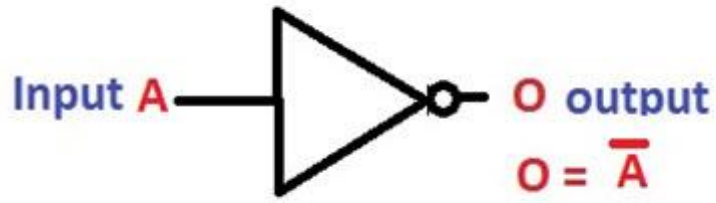
Inputs		Output
A	B	O
0	0	0
0	1	0
1	0	0
1	1	1

Truth table

AND Gate and its Truth Table

NOT Gate

The NOT-gate is one type of digital logic gate with a single input and based on the input the output will be reversed. For instance, when the input of the NOT gate is high then the output will be low. The logic diagram of NOT-gate with truth table is shown below. By using this type of logic gate, we can execute NAND and NOR gates.



Symbol

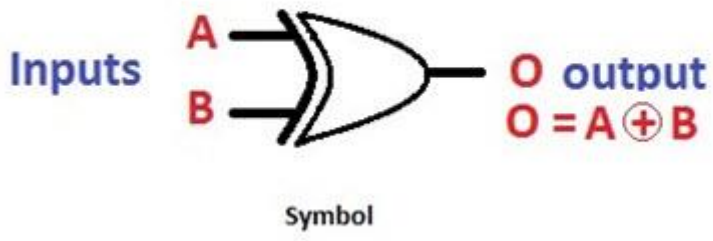
Inputs	Output
A	O
0	1
1	0

Truth table

NOT Gate and its Truth Table

Ex-OR Gate

The Exclusive-OR or EX-OR gate is one type of digital logic gate with 2-inputs & single output. The working of this logic gate depends on OR gate. If any one of the inputs of this gate is high, then the output of the EX-OR gate will be high. The symbol and truth table of the EX-OR are shown below.



Symbol

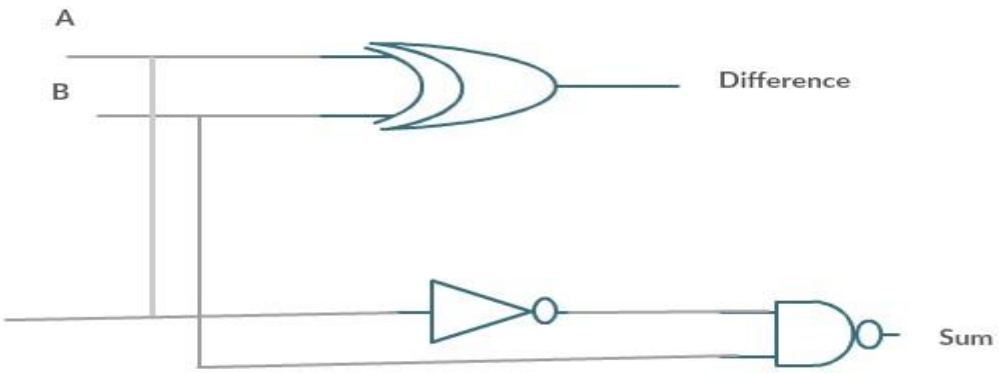
Inputs		Output
A	B	O
0	0	0
0	1	1
1	0	1
1	1	0

Truth table

EXOR Gate and its Truth Table

Half Subtractor Circuit using Nand Gate

The designing of half subtractor can be done by using logic gates like NAND gate & Ex-OR gate. In order to design this half subtractor circuit, we have to know the two concepts namely difference and borrow.



Half Subtractor Circuit using Nand Gate

If we monitor cautiously, it is fairly clear that the variety of operation executed by this circuit which is accurately related to the EX-OR gate operation. Therefore, we can simply use the EX-OR gate for making difference. In the same way, the borrow produced by half adder circuit can be simply attained by using the blend of logic gates like AND- gate and NOT-gate.

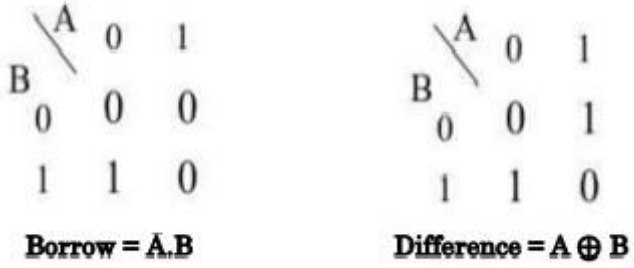
Truth Table

The truth table of the half adder circuit is shown below. It is an essential tool for any kind of digital circuit to know the possible combinations of inputs and outputs. For instance, if the subtractor has two inputs then the resultant outputs will be four. The o/p of the half subtractor is mentioned in the below table that will signify the difference bit as well as borrow bit. The

half subtractor truth table explanation can be done by using the logic gates like EX-OR logic gate and AND gate operation followed by NOT gate.

First Bit	Second Bit	Difference (EX-OR Out)	Borrow (NAND Out)
0	0	0	0
1	0	1	0
0	1	1	1
1	1	0	0

Solving the truth table using **K-Map** is shown below.



half subtractor k map

The Boolean expression of the half subtractor using truth table and K-map can be derived as

Difference **(D) = (**x'y + xy**')**

= x ⊕ y

Borrow (B) = x'y

Application of Half Subtractor

The applications of half subtractor include the following.

- Half subtractor is used to reduce the force of audio or radio signals
- It can be used in amplifiers to reduce the sound distortion
- Half subtractor is used in ALU of processor
- It can be used to increase and decrease operators and also calculates the addresses

- Half subtractor is used to subtract the least significant column numbers. For subtraction of multi-digit numbers, it can be used for the LSB.

Therefore, from the above half subtractor theory, at last, we can close that by using this circuit we can subtract from one binary bit from another to provide the outputs like Difference and Borrow. Similarly, we can design half subtractor using NAND gates circuit as well as NOR gates.

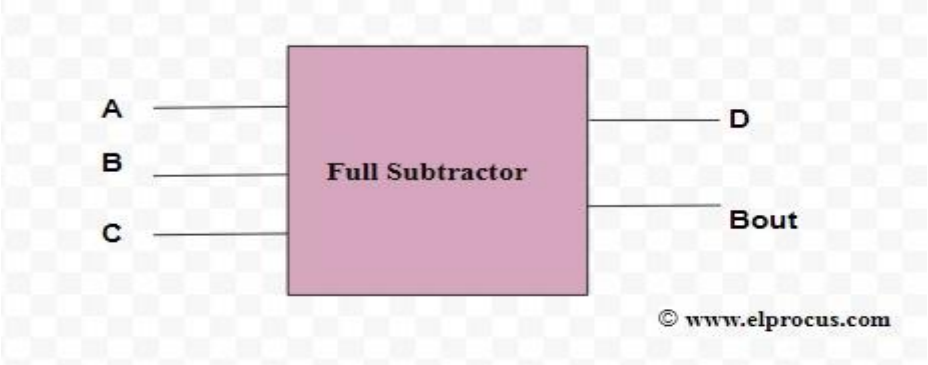
Full Subtractor Circuit Construction using Logic Gates

Generally, the full subtractor is one of the most used and essential combinational logic circuits. It is a basic electronic device, used to perform subtraction of two binary numbers. In the earlier article, already we have given the basic theory of half adder & a full adder which uses the binary digits for the computation. Likewise, the full-subtractor uses binary digits like 0,1 for the subtraction. The circuit of full subtractor can be built with logic gates such as OR, Ex-OR, NAND gate. The inputs of this subtractor are A, B, Bin and outputs are D, Bout.

This article gives full-subtractor theory idea which comprises the premises like what is a subtractor, full subtractor design with logic gates, truth table, etc. This article is useful for engineering students who can go through these topics in HDL Practical lab.

What is Full Subtractor?

Full subtractor is an electronic device or logic circuit which performs subtraction of two binary digits. It is a combinational logic circuit used in digital electronics. Many combinational circuits are available in integrated circuit technology namely adders, encoders, decoders and multiplexers. In this article, we are going to discuss full subtractor construction using half subtractor and also the terms like truth table.

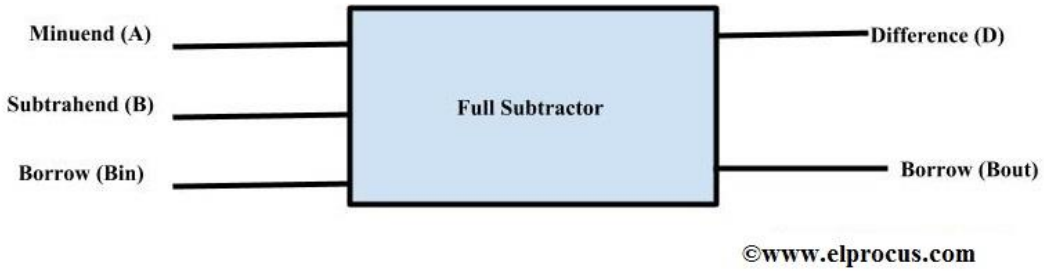


A full subtractor is formed by two half subtractors, which involves three inputs such as minuend, subtrahend and borrow, borrow bit among the inputs is obtained from subtraction

of two binary digits and is subtracted from next higher order pair of bits, outputs as difference and borrow.

Full Subtractor Block Diagram

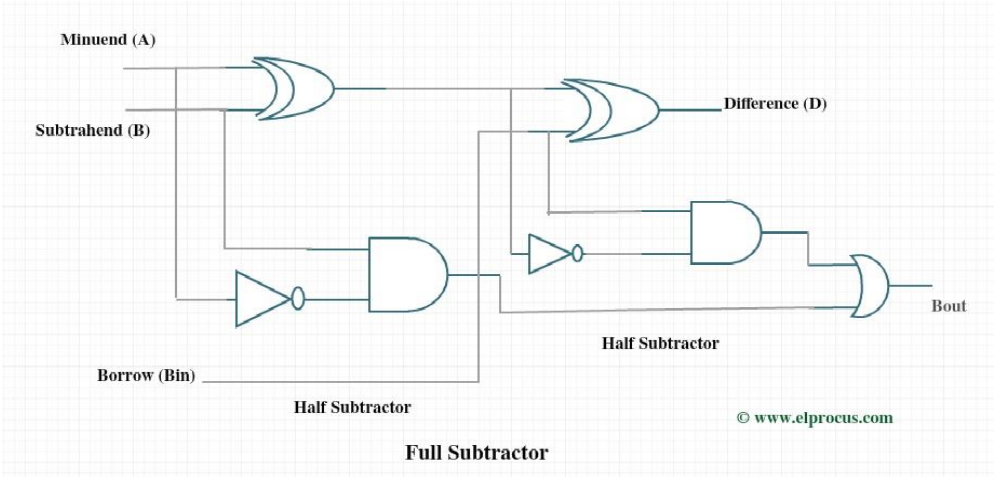
The foremost disadvantage of the half subtractor is, we cannot make a Borrow bit in this subtractor. Whereas in full subtractor design, actually we can make a Borrow bit in the circuit & can subtract with remaining two i/ps. Here A is minuend, B is subtrahend & Bin is borrow in. The outputs are Difference (Diff) & Bout (Borrow out). The complete subtractor circuit can obtain by using two half subtractors with an extra OR gate.



Full Subtractor Circuit Diagram with Logic Gates

The circuit diagram of full subtractor using basic gates is shown in the following block diagram. This circuit can be done with two half-Subtractor circuits.

In the initial half-Subtractor circuit, the binary inputs are A and B. As we have discussed in the previous half-Subtractor article, it will generate two outputs namely difference (Diff) & Borrow.



Full Subtractor using Logic Gates

The difference o/p of the left subtractor is given to the Left half-Subtractor circuit's. Diff output is further provided to the input of the right half Subtractor circuit. We offered the Borrow in bit across the other i/p of next half subtractor circuit. Once more it will give Diff out as well as Borrow out the bit. The final output of this subtractor is Diff output.
On the other hand, the Borrow out of both the half Subtractor circuits is connected to OR logic gate. Later than giving out OR logic for two output bits of the subtractor, we acquire the final Borrow out of the subtractor. The last Borrow out to signify the MSB (a most significant bit).

If we observe the internal circuit of the full Subtractor, we can see two Half Subtractors with NAND gate and XOR gate with an extra OR gate.

Full Subtractor Truth Table

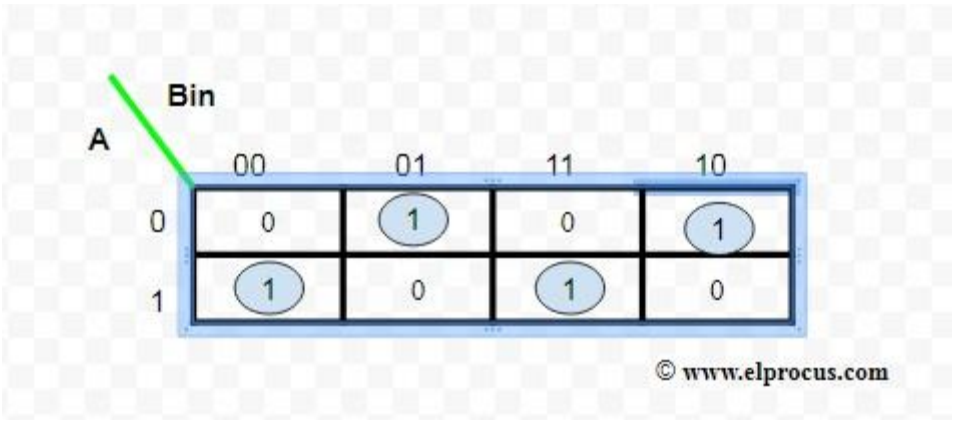
This subtractor circuit executes a subtraction between two bits, which has 3- inputs (A, B and Bin) and two outputs (D and Bout). Here the inputs indicate minuend, subtrahend, & previous borrow, whereas the two outputs are denoted as borrow o/p and difference. The following image shows the truth table of full-subtractor.

Inputs			**Outputs**	
Minuend (A)	**Subtrahend (B)**	**Borrow (Bin)**	**Difference (D)**	**Borrow (Bout)**
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

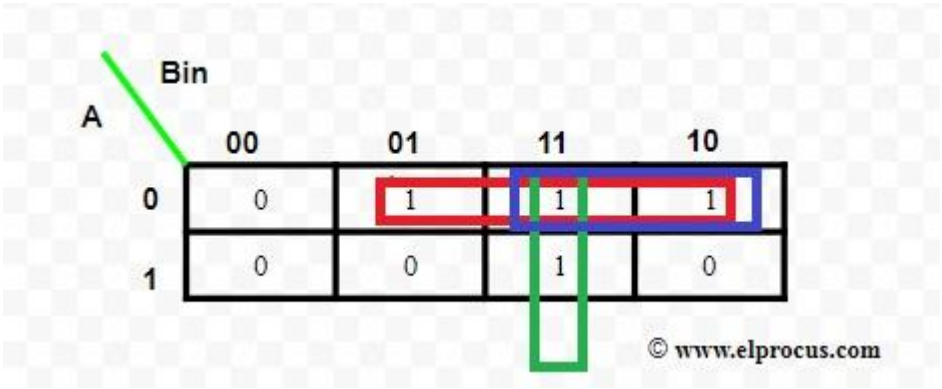
1	1	0	0	0
1	1	1	1	1

Full Subtractor K-Map

The simplification of the K-map for the above difference and borrow is shown below.



The full subtractor equations for the difference as well as Bin are mentioned below.



The full subtractor expression for Difference is,

$$D = A'B'Bin + AB'Bin' + A'BBin' + ABBin$$

The full-subtractor expression for Borrow is,

$$Bout = A'Bin + A'B + BBin$$

Applications of Full Subtractor

Some of the applications of full-subtractor include the following

- These are generally employed for ALU (Arithmetic logic unit) in computers to subtract as CPU & GPU for the applications of graphics to decrease the circuit difficulty.
- Subtractors are mostly used for performing arithmetical functions like subtraction, in electronic calculators as well as digital devices.
- These are also applicable for different microcontrollers for arithmetic subtraction, timers, and program counter (PC)
- Subtractors are used in processors to compute tables, address, etc.
- It is also useful for DSP and networking based systems.

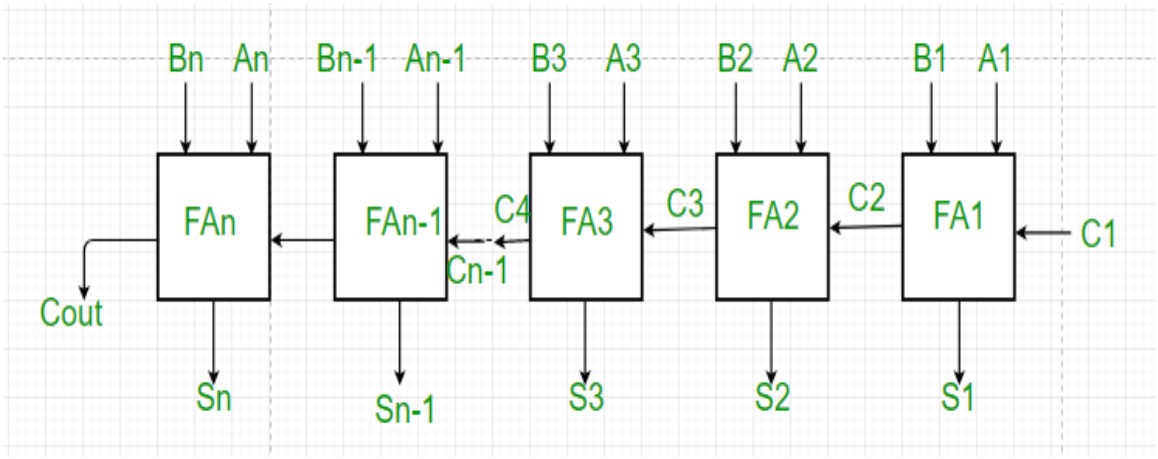
From the above information, by evaluating the adder, full subtractor using two half subtractor circuits, and its tabular forms, one can notice that Dout in the full-subtractor is accurately similar to the Sout of the full-adder. The only variation is that A (input variable) is complemented in the full-subtractor. Thus, it is achievable to change the full-adder circuit into full-subtractor by just complementing the i/p A before it is given to the logic gates to generate the last borrow-bit output (Bout).

Q4. Explain Parallel Adder and Parallel Subtarctor

Parallel Adder and Parallel Subtarctor

Parallel Adder –

A single full adder performs the addition of two one bit numbers and an input carry. But a **Parallel Adder** is a digital circuit capable of finding the arithmetic **sum** of two binary numbers that is **greater than one bit** in length by operating on corresponding pairs of bits in parallel. It consists of **full adders connected in a chain** where the output carry from each full adder is connected to the carry input of the next higher order full adder in the chain. **A n bit parallel adder requires n full adders to perform the operation.** So for the two-bit number, two adders are needed while for four bit number, four adders are needed and so on. Parallel adders normally incorporate carry lookahead logic to ensure that carry propagation between subsequent stages of addition does not limit addition speed.



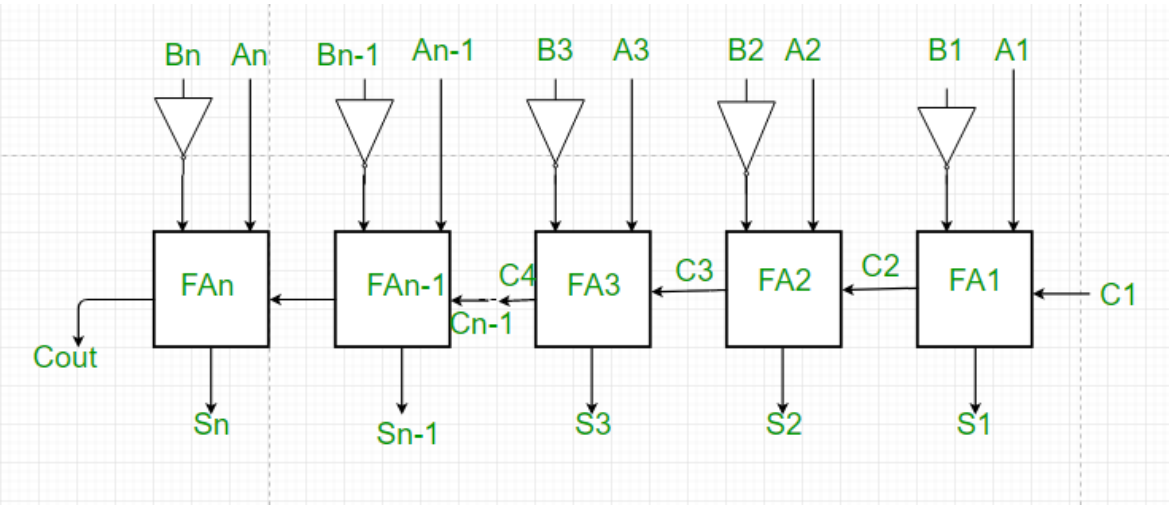
Working of parallel Adder –

1. As shown in the figure, firstly the full adder FA1 adds A1 and B1 along with the carry C1 to generate the sum S1 (the first bit of the output sum) and the carry C2 which is connected to the next adder in chain.
2. Next, the full adder FA2 uses this carry bit C2 to add with the input bits A2 and B2 to generate the sum S2(the second bit of the output sum) and the carry C3 which is again further connected to the next adder in chain and so on.

3. The process continues till the last full adder FAn uses the carry bit Cn to add with its input An and Bn to generate the last bit of the output along last carry bit Cout.

Parallel Subtractor –

A Parallel Subtractor is a digital circuit capable of finding the arithmetic difference of two binary numbers that is greater than one bit in length by operating on corresponding pairs of bits in parallel. The parallel subtractor can be designed in several ways including combination of half and full subtractors, all full subtractors or all full adders with subtrahend complement input.



Working of Parallel Subtractor –

1. As shown in the figure, the parallel binary subtractor is formed by combination of all full adders with subtrahend complement input.
2. This operation considers that the addition of minuend along with the 2's complement of the subtrahend is equal to their subtraction.
3. Firstly the 1's complement of B is obtained by the NOT gate and 1 can be added through the carry to find out the 2's complement of B. This is further added to A to carry out the arithmetic subtraction.
4. The process continues till the last full adder FAn uses the carry bit Cn to add with its input An and 2's complement of Bn to generate the last bit of the output along last carry bit Cout.

Advantages of parallel Adder/Subtractor –

1. The parallel adder/subtractor performs the addition operation faster as compared to serial adder/subtractor.
2. Time required for addition does not depend on the number of bits.
3. The output is in parallel form i.e all the bits are added/subtracted at the same time.
4. It is less costly.

Disadvantages of parallel Adder/Subtractor –

1. Each adder has to wait for the carry which is to be generated from the previous adder in chain.
2. The propagation delay(delay associated with the travelling of carry bit) is found to increase with the increase in the number of bits to be added.

Q5. What is Digital Binary Multiplier?

A **binary multiplier** is a combinational logic circuit or digital device used for **multiplying two binary numbers**. The two numbers are more specifically known as **multiplicand** and **multiplier** and the result is known as a **product**.

The multiplicand & multiplier can be of various bit size. The product's bit size depends on the bit size of the multiplicand & multiplier. The bit size of the product is equal to the sum of the bit size of multiplier & multiplicand.

Binary multiplication method is same as decimal multiplication. Binary multiplication of more than 1-bit numbers contains 2 steps. The 1st step is single bit-wise multiplication known as partial product and the 2nd step is adding all partial products into a single product.

Partial products or single bit products can be obtained by using AND gates. However, to add these partial products we need full adders & half adders.

The schematic design of a digital multiplier differs with bit size. The design becomes complex with the increase in bit size of the multiplier.

Types of Binary Multipliers

- **2×2 Bit Multiplier**

lets discuss one by one as follow:

2×2 Bit Multiplier

This multiplier can multiply two numbers having bit size = 2 i.e. the multiplier and multiplicand can be of 2 bits. The product bit size will be the sum of the bit size of the input i.e. **2+2=4**. The maximum range of its output is **3 x 3 = 9**. So we can accommodate decimal 9 in 4 bits. It is another way of finding the bit size of the product.

Suppose multiplicand $\mathbf{A_1\ A_0}$ & multiplier $\mathbf{B_1\ B_0}$ & $\mathbf{P_3\ P_2\ P_1\ P_0}$ as a product of the 2×2 multiplier.

First, multiplicand A_1A_0 is multiplied with LSB B_0 of the multiplier to obtain the partial product. This is obtained using AND gates. Then the same multiplicand is multiplied (AND) with the 2nd LSB to get the 2nd partial product. The multiplicand is multiplied with each bit of the multiplier (from LSB to MSB) to obtain partial products.

The number of partial products is equal to the number of bit size of the multiplier. In 2×2 multiplier, multiplier size is 2 bits so we get 2 partial products.

Now we need to add these partial products. There are two ways of adding;

- **Using 2-bit full adder**

- **Using individual single bit adders.**

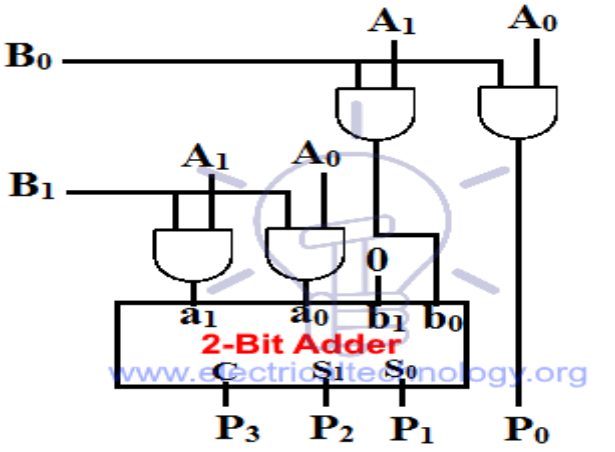
2×2 Bit Multiplier using 2-Bit Full Adder

if we use 2-bit full adder all we have to do is to know which term should be added.

The partial product of LSBs of inputs is the LSB of the product. So it should remain untouched.

The other terms of each partial product should be considered and added using 2-bit full adder.

Construction and design schematic of 2×2 bit multiplier is given in the figure below;



Schematic of 2x2 Bit Multiplier Using 2-Bit Full Adder

The single bit from LSB partial product, 2 bits from the Sum & a carry bit makes the 4 bits of the products.

Truth Table for 2 Bit Multiplier

Multiplier Bits		Multiple of Multiplicand	
Yi+1	**Y1**	**Multiples**	**Implementation**
0	0	0	0
0	1	1	x
1	0	2	Shift left X by 1

1	1	3	(Shift left X by 1) + X

Unit 3

Q1. Explain Digital Circuits – Multiplexers & Demultiplexer.

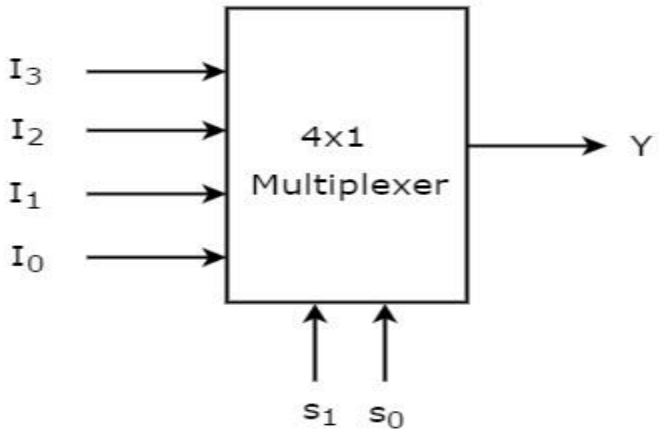
Multiplexers

Multiplexer is a combinational circuit that has maximum of 2^n data inputs, 'n' selection lines and single output line. One of these data inputs will be connected to the output based on the values of selection lines.

Since there are 'n' selection lines, there will be 2^n possible combinations of zeros and ones. So, each combination will select only one data input. Multiplexer is also called as **Mux**.

4x1 Multiplexer

4x1 Multiplexer has four data inputs I_3, I_2, I_1 & I_0, two selection lines s_1 & s_0 and one output Y. The **block diagram** of 4x1 Multiplexer is shown in the following figure.



One of these 4 inputs will be connected to the output based on the combination of inputs present at these two selection lines. **Truth table** of 4x1 Multiplexer is shown below.

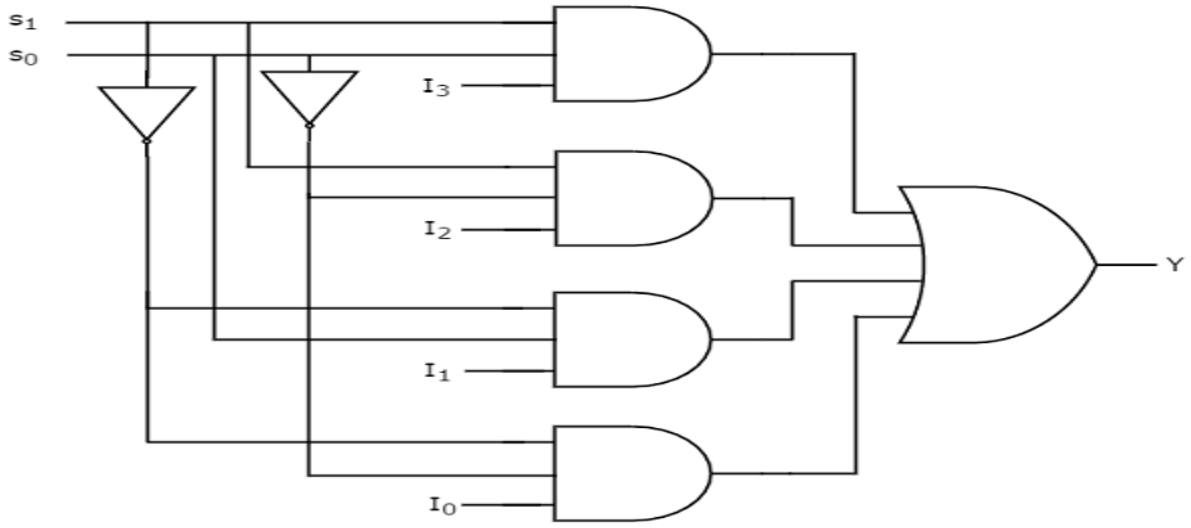
Selection Lines		Output
S_1	**S_0**	**Y**
0	0	I_0
0	1	I_1

1	0	I_2
1	1	I_3

From Truth table, we can directly write the **Boolean function** for output, Y as

$$Y=S1'S0'I0+S1'S0I1+S1S0'I2+S1S0I3Y=S1'S0'I0+S1'S0I1+S1S0'I2+S1S0I3$$

We can implement this Boolean function using Inverters, AND gates & OR gate. The **circuit diagram** of 4x1 multiplexer is shown in the following figure.



We can easily understand the operation of the above circuit. Similarly, you can implement 8x1 Multiplexer and 16x1 multiplexer by following the same procedure.

Implementation of Higher-order Multiplexers.

Now, let us implement the following two higher-order Multiplexers using lower-order Multiplexers.

- 8x1 Multiplexer
- 16x1 Multiplexer

8x1 Multiplexer

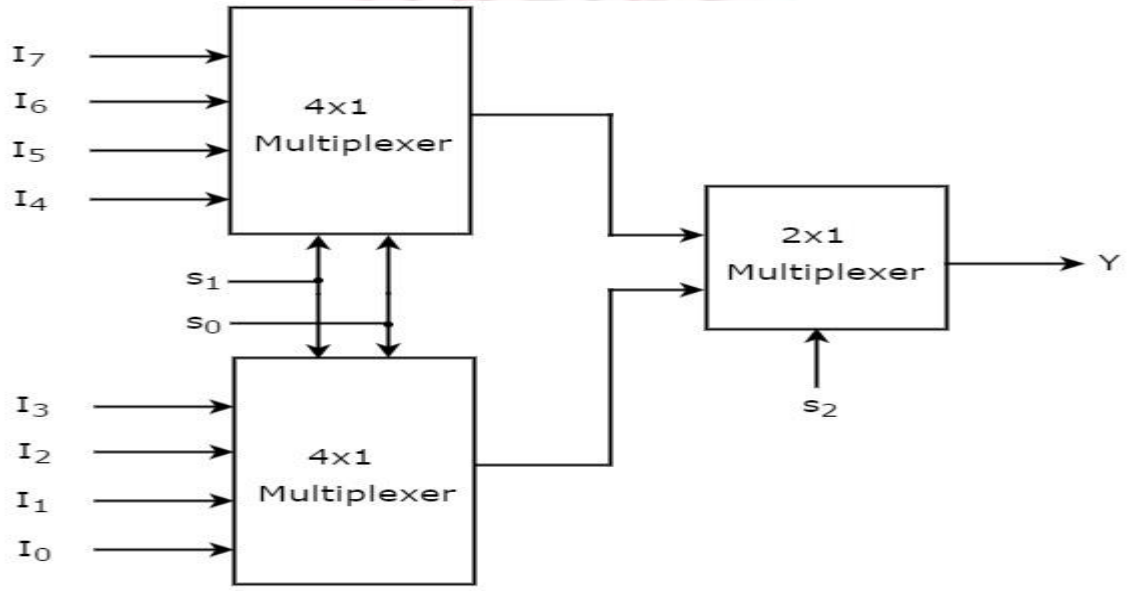
In this section, let us implement 8x1 Multiplexer using 4x1 Multiplexers and 2x1 Multiplexer. We know that 4x1 Multiplexer has 4 data inputs, 2 selection lines and one output. Whereas, 8x1 Multiplexer has 8 data inputs, 3 selection lines and one output.

So, we require two **4x1 Multiplexers** in first stage in order to get the 8 data inputs. Since, each 4x1 Multiplexer produces one output, we require a **2x1 Multiplexer** in second stage by considering the outputs of first stage as inputs and to produce the final output.

Let the 8x1 Multiplexer has eight data inputs I_7 to I_0, three selection lines s_2, s_1 & s0 and one output Y. The **Truth table** of 8x1 Multiplexer is shown below.

Selection Inputs			Output
S_2	S_1	S_0	Y
0	0	0	I_0
0	0	1	I_1
0	1	0	I_2
0	1	1	I_3
1	0	0	I_4
1	0	1	I_5
1	1	0	I_6
1	1	1	I_7

We can implement 8x1 Multiplexer using lower order Multiplexers easily by considering the above Truth table. The **block diagram** of 8x1 Multiplexer is shown in the following figure.



The same **selection lines, s_1 & s_0** are applied to both 4x1 Multiplexers. The data inputs of upper 4x1 Multiplexer are I_7 to I_4 and the data inputs of lower 4x1 Multiplexer are I_3 to I_0. Therefore, each 4x1 Multiplexer produces an output based on the values of selection lines, s_1 & s_0.

The outputs of first stage 4x1 Multiplexers are applied as inputs of 2x1 Multiplexer that is present in second stage. The other **selection line, s_2** is applied to 2x1 Multiplexer.

- If s_2 is zero, then the output of 2x1 Multiplexer will be one of the 4 inputs I_3 to I_0 based on the values of selection lines s_1 & s_0.
- If s_2 is one, then the output of 2x1 Multiplexer will be one of the 4 inputs I_7 to I_4 based on the values of selection lines s_1 & s_0.

Therefore, the overall combination of two 4x1 Multiplexers and one 2x1 Multiplexer performs as one 8x1 Multiplexer.

16x1 Multiplexer

In this section, let us implement 16x1 Multiplexer using 8x1 Multiplexers and 2x1 Multiplexer. We know that 8x1 Multiplexer has 8 data inputs, 3 selection lines and one output. Whereas, 16x1 Multiplexer has 16 data inputs, 4 selection lines and one output.

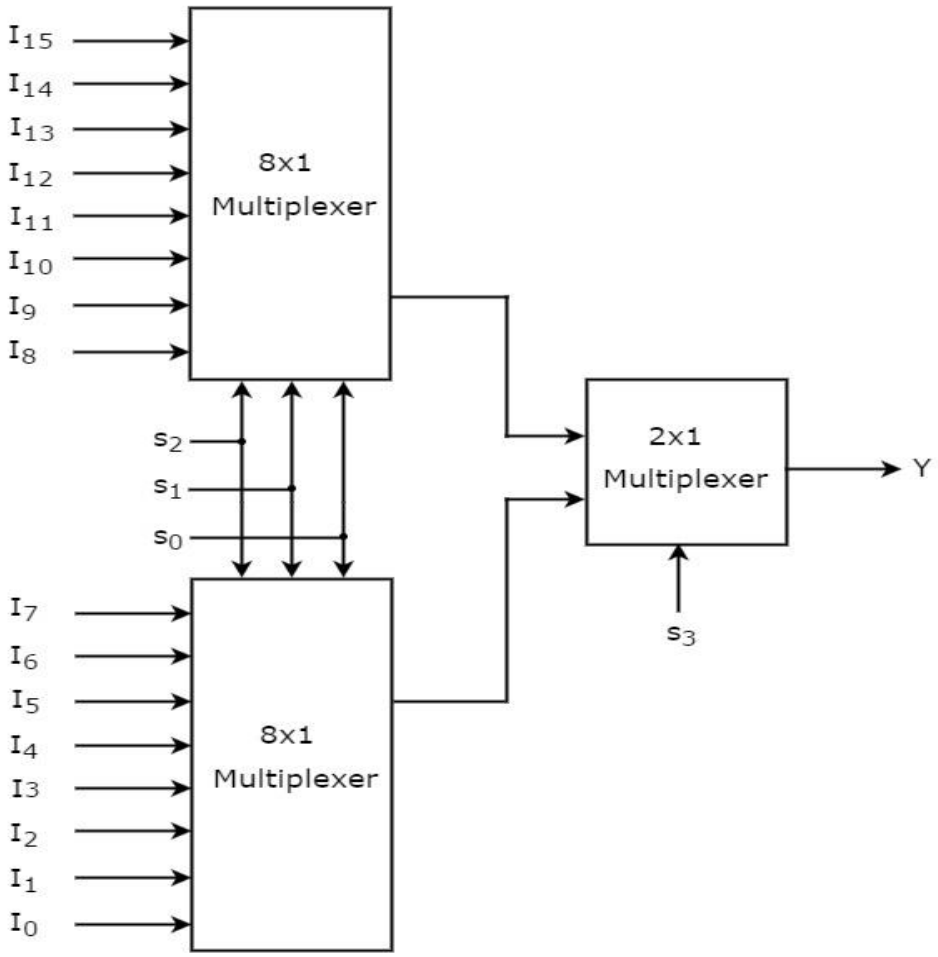
So, we require two **8x1 Multiplexers** in first stage in order to get the 16 data inputs. Since, each 8x1 Multiplexer produces one output, we require a 2x1 Multiplexer in second stage by considering the outputs of first stage as inputs and to produce the final output.

Let the 16x1 Multiplexer has sixteen data inputs I_{15} to I_0, four selection lines s_3 to s_0 and one output Y. The **Truth table** of 16x1 Multiplexer is shown below.

Selection Inputs				Output
S_3	S_2	S_1	S_0	Y
0	0	0	0	I_0
0	0	0	1	I_1
0	0	1	0	I_2
0	0	1	1	I_3
0	1	0	0	I_4

0	1	0	1	I_5
0	1	1	0	I_6
0	1	1	1	I_7
1	0	0	0	I_8
1	0	0	1	I_9
1	0	1	0	I_{10}
1	0	1	1	I_{11}
1	1	0	0	I_{12}
1	1	0	1	I_{13}
1	1	1	0	I_{14}
1	1	1	1	I_{15}

We can implement 16x1 Multiplexer using lower order Multiplexers easily by considering the above Truth table. The **block diagram** of 16x1 Multiplexer is shown in the following figure.



The **same selection lines, s_2, s_1 & s_0** are applied to both 8x1 Multiplexers. The data inputs of upper 8x1 Multiplexer are I_{15} to I_8 and the data inputs of lower 8x1 Multiplexer are I_7 to I_0. Therefore, each 8x1 Multiplexer produces an output based on the values of selection lines, s_2, s_1 & s_0.

The outputs of first stage 8x1 Multiplexers are applied as inputs of 2x1 Multiplexer that is present in second stage. The other **selection line, s_3** is applied to 2x1 Multiplexer.

- If s_3 is zero, then the output of 2x1 Multiplexer will be one of the 8 inputs Is_7 to I_0 based on the values of selection lines s_2, s_1 & s_0.
- If s_3 is one, then the output of 2x1 Multiplexer will be one of the 8 inputs I_{15} to I_8 based on the values of selection lines s_2, s_1 & s_0.

Therefore, the overall combination of two 8x1 Multiplexers and one 2x1 Multiplexer performs as one 16x1 Multiplexer.

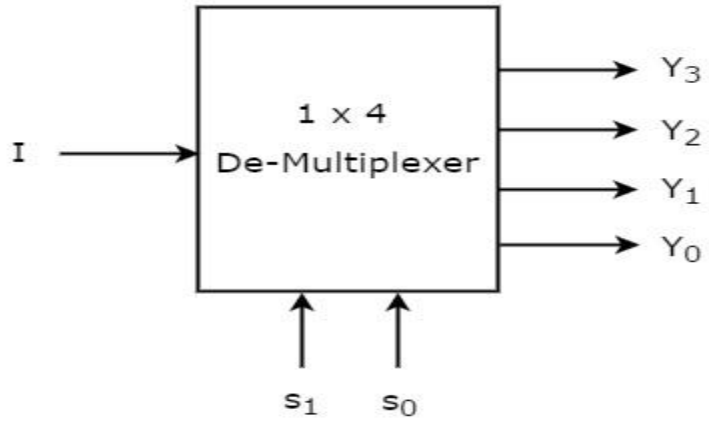
Demultiplexer.

De-Multiplexer is a combinational circuit that performs the reverse operation of Multiplexer. It has single input, 'n' selection lines and maximum of 2^n outputs. The input will be connected to one of these outputs based on the values of selection lines.

Since there are 'n' selection lines, there will be 2^n possible combinations of zeros and ones. So, each combination can select only one output. De-Multiplexer is also called as **De-Mux**.

1x4 De-Multiplexer

1x4 De-Multiplexer has one input I, two selection lines, s_1 & s_0 and four outputs Y_3, Y_2, Y_1 &Y_0. The **block diagram** of 1x4 De-Multiplexer is shown in the following figure.



The single input 'I' will be connected to one of the four outputs, Y_3 to Y_0 based on the values of selection lines s_1 & s0. The **Truth table** of 1x4 De-Multiplexer is shown below.

Selection Inputs		**Outputs**			
S_1	**S_0**	**Y_3**	**Y_2**	**Y_1**	**Y_0**
0	0	0	0	0	**I**
0	1	0	0	**I**	0
1	0	0	**I**	0	0
1	1	**I**	0	0	0

From the above Truth table, we can directly write the **Boolean functions** for each output as

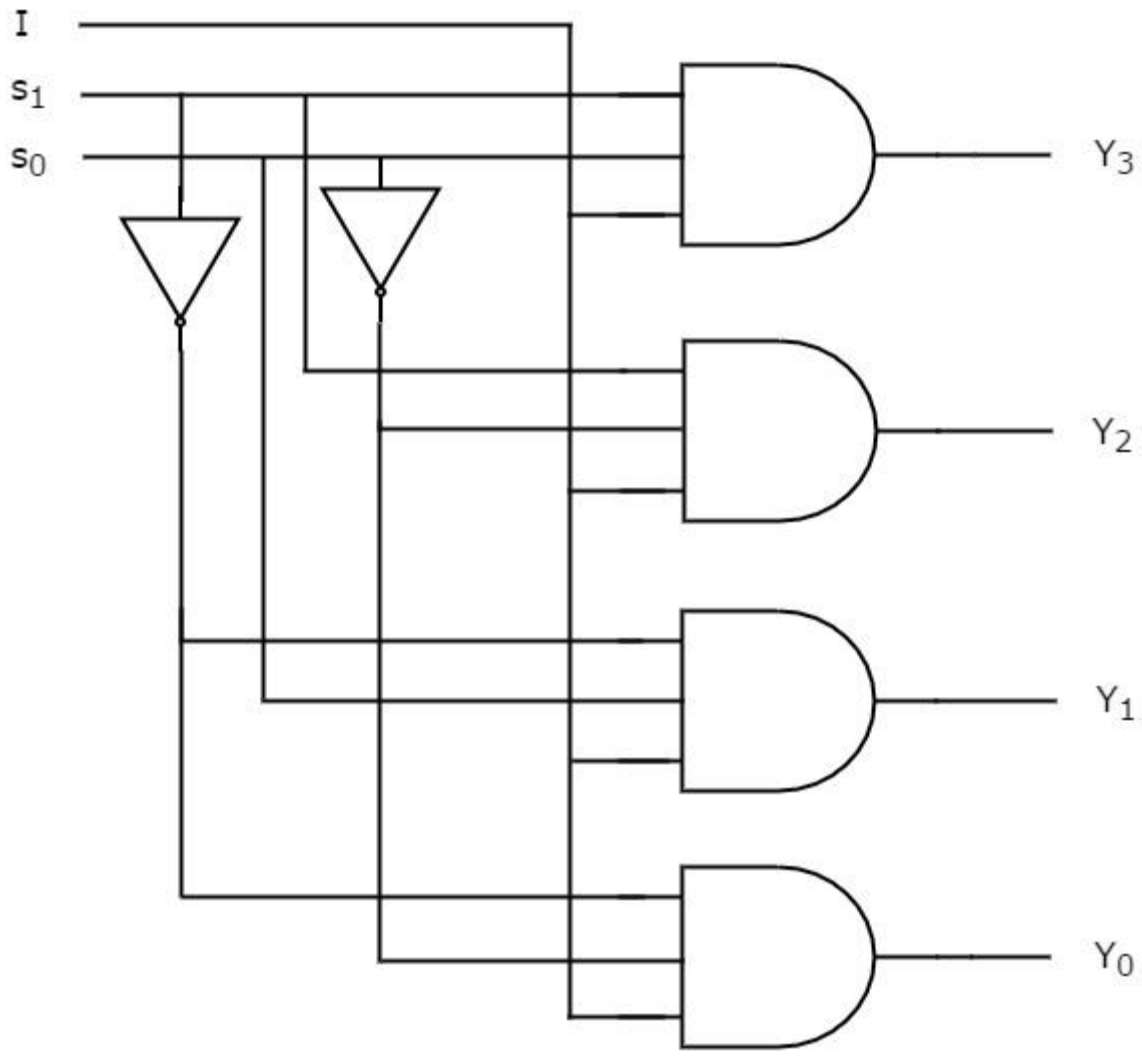
Y3=s1s0IY3=s1s0I

Y2=s1s0′IY2=s1s0′I

Y1=s1′s0IY1=s1′s0I

Y0=s1′s0′IY0=s1′s0′I

We can implement these Boolean functions using Inverters & 3-input AND gates. The **circuit diagram** of 1x4 De-Multiplexer is shown in the following figure.



We can easily understand the operation of the above circuit. Similarly, you can implement 1x8 De-Multiplexer and 1x16 De-Multiplexer by following the same procedure.

Implementation of Higher-order De-Multiplexers

Now, let us implement the following two higher-order De-Multiplexers using lower-order De-Multiplexers.

- 1x8 De-Multiplexer
- 1x16 De-Multiplexer

1x8 De-Multiplexer

In this section, let us implement 1x8 De-Multiplexer using 1x4 De-Multiplexers and 1x2 De-Multiplexer. We know that 1x4 De-Multiplexer has single input, two selection lines and four outputs. Whereas, 1x8 De-Multiplexer has single input, three selection lines and eight outputs.

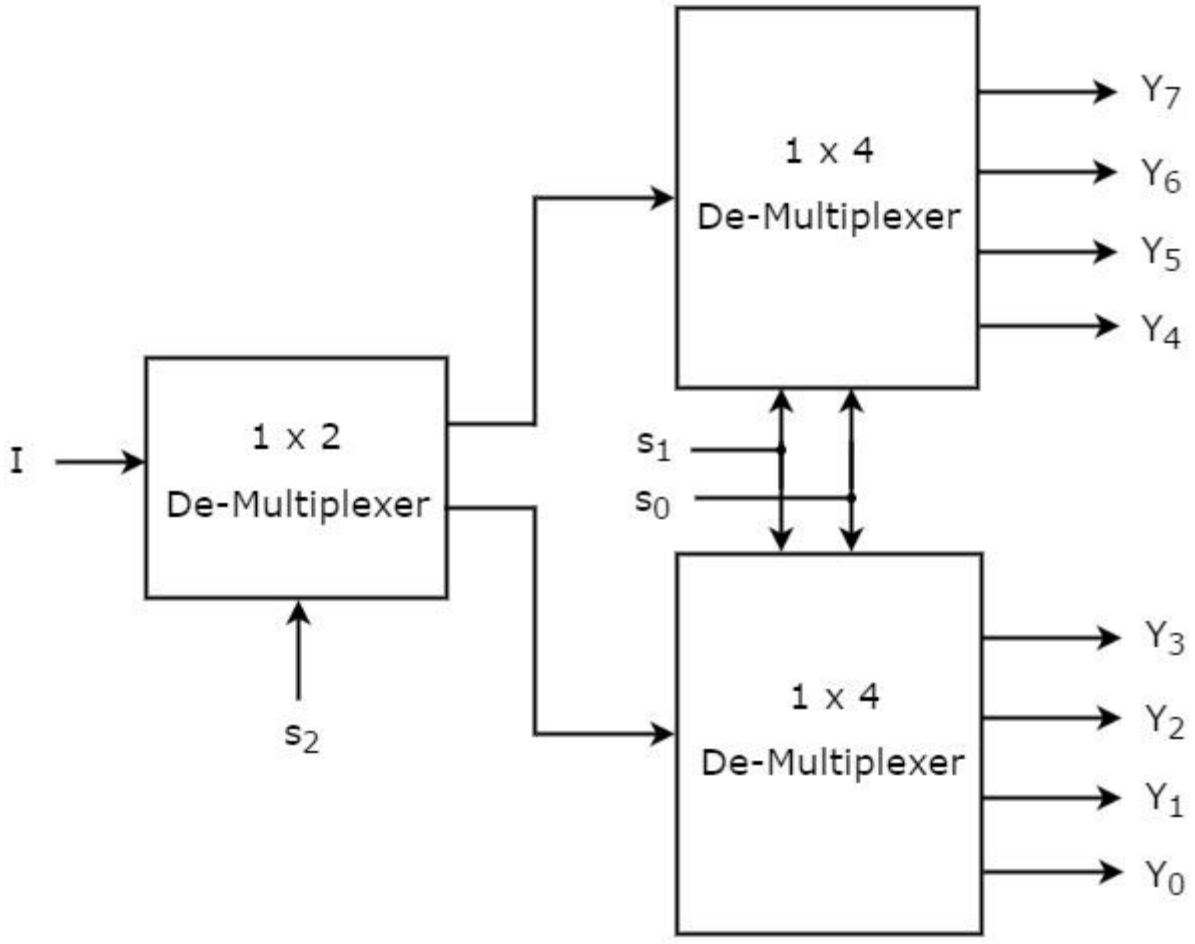
So, we require two **1x4 De-Multiplexers** in second stage in order to get the final eight outputs. Since, the number of inputs in second stage is two, we require **1x2 DeMultiplexer** in first stage so that the outputs of first stage will be the inputs of second stage. Input of this 1x2 De-Multiplexer will be the overall input of 1x8 De-Multiplexer.

Let the 1x8 De-Multiplexer has one input I, three selection lines s_2, s_1 & s_0 and outputs Y_7 to Y_0. The **Truth table** of 1x8 De-Multiplexer is shown below.

Selection Inputs			**Outputs**							
s_2	**s_1**	**s_0**	**Y_7**	**Y_6**	**Y_5**	**Y_4**	**Y_3**	**Y_2**	**Y_1**	**Y_0**
0	0	0	0	0	0	0	0	0	0	**I**
0	0	1	0	0	0	0	0	0	**I**	0
0	1	0	0	0	0	0	0	**I**	0	0
0	1	1	0	0	0	0	**I**	0	0	0
1	0	0	0	0	0	**I**	0	0	0	0

1	0	1	0	0	**I**	0	0	0	0	0
1	1	0	0	**I**	0	0	0	0	0	0
1	1	1	**I**	0	0	0	0	0	0	0

We can implement 1x8 De-Multiplexer using lower order Multiplexers easily by considering the above Truth table. The **block diagram** of 1x8 De-Multiplexer is shown in the following figure.



The common **selection lines, s_1 & s_0** are applied to both 1x4 De-Multiplexers. The outputs of upper 1x4 De-Multiplexer are Y_7 to Y_4 and the outputs of lower 1x4 De-Multiplexer are Y_3 to Y_0.

The other **selection line, s_2** is applied to 1x2 De-Multiplexer. If s_2 is zero, then one of the four outputs of lower 1x4 De-Multiplexer will be equal to input, I based on the values of

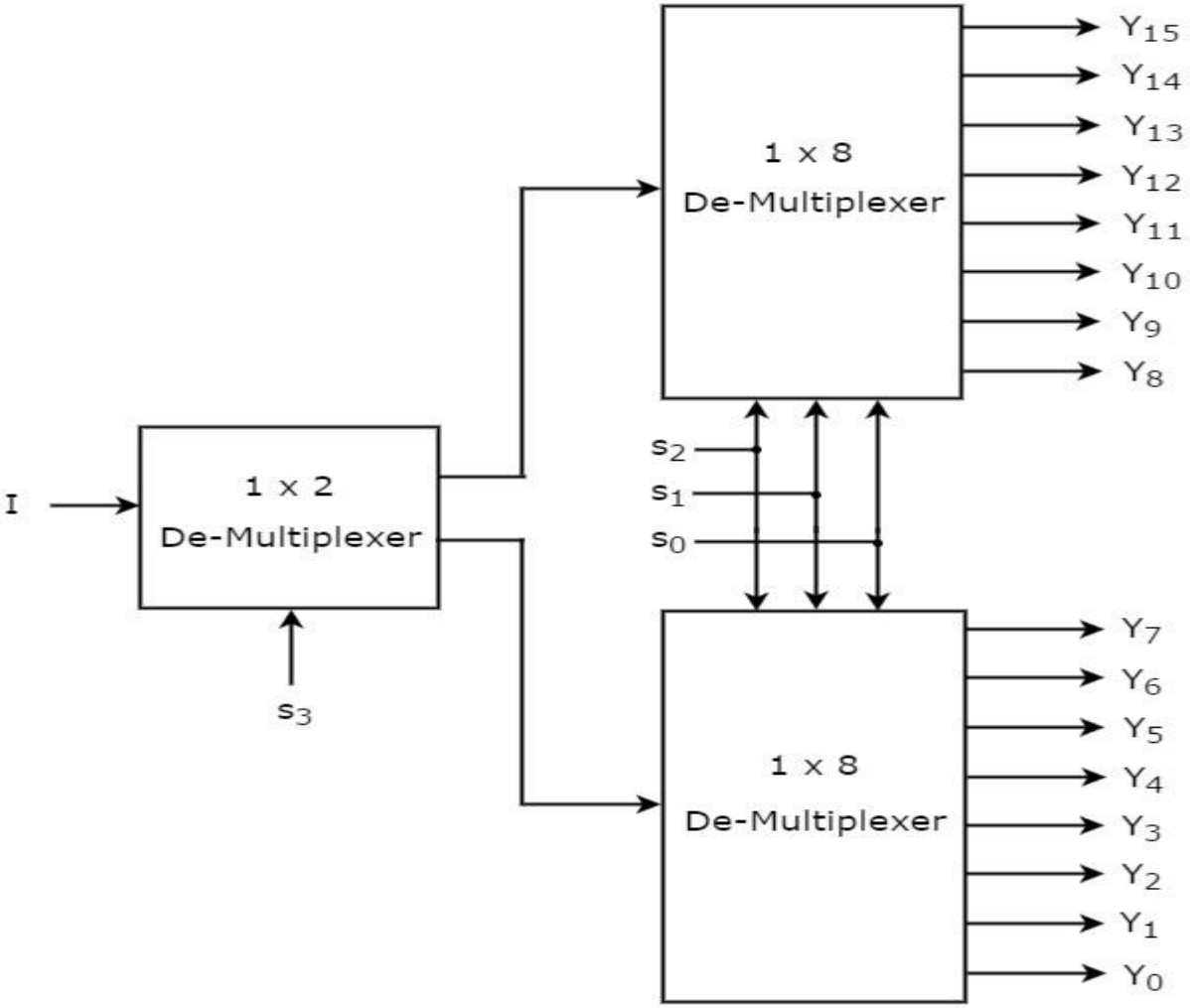
selection lines s_1 & s_0. Similarly, if s_2 is one, then one of the four outputs of upper 1x4 DeMultiplexer will be equal to input, I based on the values of selection lines s_1 & s_0.

1x16 De-Multiplexer

In this section, let us implement 1x16 De-Multiplexer using 1x8 De-Multiplexers and 1x2 De-Multiplexer. We know that 1x8 De-Multiplexer has single input, three selection lines and eight outputs. Whereas, 1x16 De-Multiplexer has single input, four selection lines and sixteen outputs.

So, we require two **1x8 De-Multiplexers** in second stage in order to get the final sixteen outputs. Since, the number of inputs in second stage is two, we require **1x2 DeMultiplexer** in first stage so that the outputs of first stage will be the inputs of second stage. Input of this 1x2 De-Multiplexer will be the overall input of 1x16 De-Multiplexer.

Let the 1x16 De-Multiplexer has one input I, four selection lines s_3, s_2, s_1 & s_0 and outputs Y_{15} to Y_0. The **block diagram** of 1x16 De-Multiplexer using lower order Multiplexers is shown in the following figure.



The common **selection lines s_2, s_1 & s_0** are applied to both 1x8 De-Multiplexers. The outputs of upper 1x8 De-Multiplexer are Y_{15} to Y_8 and the outputs of lower 1x8 DeMultiplexer are Y_7 to Y_0.

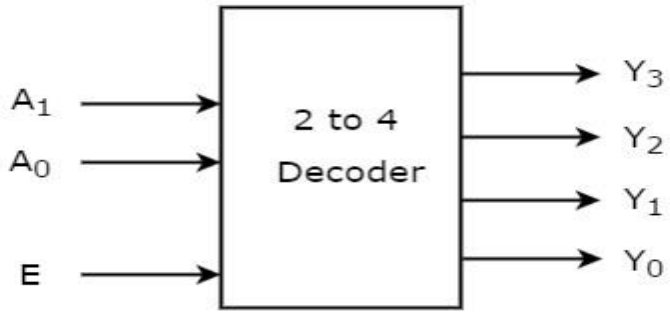
The other **selection line, s_3** is applied to 1x2 De-Multiplexer. If s_3 is zero, then one of the eight outputs of lower 1x8 De-Multiplexer will be equal to input, I based on the values of selection lines s_2, s_1 & s_0. Similarly, if s3 is one, then one of the 8 outputs of upper 1x8 De-Multiplexer will be equal to input, I based on the values of selection lines s_2, s_1 & s_0.

Q2. Explain Decoders

Decoder is a combinational circuit that has 'n' input lines and maximum of 2^n output lines. One of these outputs will be active High based on the combination of inputs present, when the decoder is enabled. That means decoder detects a particular code. The outputs of the decoder are nothing but the **min terms** of 'n' input variables lineslines, when it is enabled.

2 to 4 Decoder

Let 2 to 4 Decoder has two inputs A_1 & A_0 and four outputs Y_3, Y_2, Y_1 & Y_0. The **block diagram** of 2 to 4 decoder is shown in the following figure.



One of these four outputs will be '1' for each combination of inputs when enable, E is '1'. The **Truth table** of 2 to 4 decoder is shown below.

Enable	Inputs		Outputs			
E	A_1	A_0	Y_3	Y_2	Y_1	Y_0
0	x	x	0	0	0	0
1	0	0	0	0	0	1
1	0	1	0	0	1	0
1	1	0	0	1	0	0
1	1	1	1	0	0	0

From Truth table, we can write the **Boolean functions** for each output as

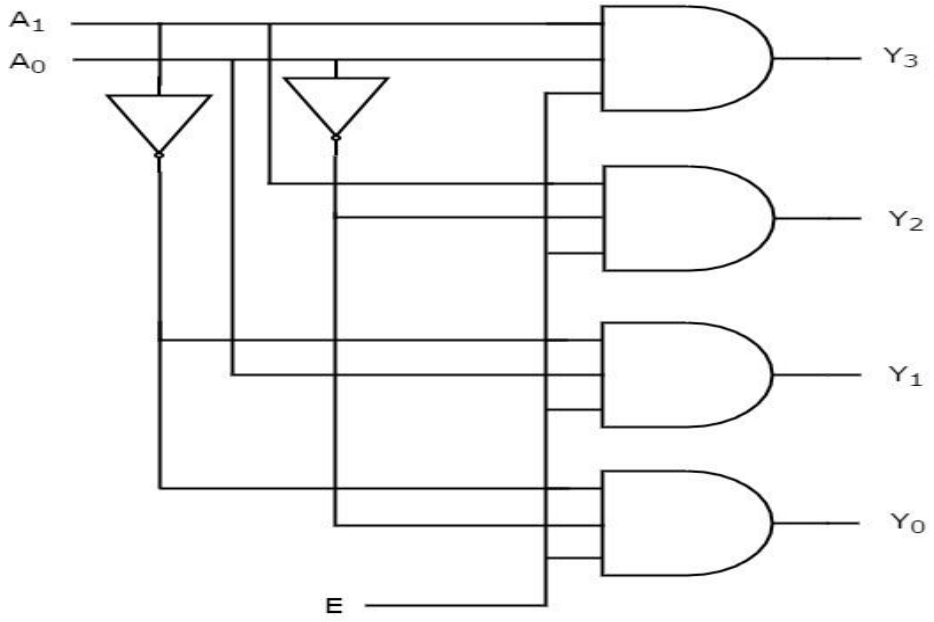
Y3=E.A1.A0Y3=E.A1.A0

Y2=E.A1.A0′Y2=E.A1.A0′

Y1=E.A1′.A0Y1=E.A1′.A0

$$Y0=E.A1'.A0'Y0=E.A1'.A0'$$

Each output is having one product term. So, there are four product terms in total. We can implement these four product terms by using four AND gates having three inputs each & two inverters. The **circuit diagram** of 2 to 4 decoder is shown in the following figure.



Therefore, the outputs of 2 to 4 decoder are nothing but the **min terms** of two input variables A_1 & A_0, when enable, E is equal to one. If enable, E is zero, then all the outputs of decoder will be equal to zero.

Similarly, 3 to 8 decoder produces eight min terms of three input variables A_2, A_1 & A_0 and 4 to 16 decoder produces sixteen min terms of four input variables A_3, A_2, A_1 & A_0.

Implementation of Higher-order Decoders

Now, let us implement the following two higher-order decoders using lower-order decoders.

- 3 to 8 decoder
- 4 to 16 decoder

3 to 8 Decoder

In this section, let us implement **3 to 8 decoder using 2 to 4 decoders**. We know that 2 to 4 Decoder has two inputs, A_1 & A_0 and four outputs, Y_3 to Y_0. Whereas, 3 to 8 Decoder has three inputs A_2, A_1 & A_0 and eight outputs, Y_7 to Y_0.

We can find the number of lower order decoders required for implementing higher order decoder using the following formula.

Requirednumberoflowerorderdecoders=m2m1Requirednumberoflowerorderdecoders=m2m1

Where,

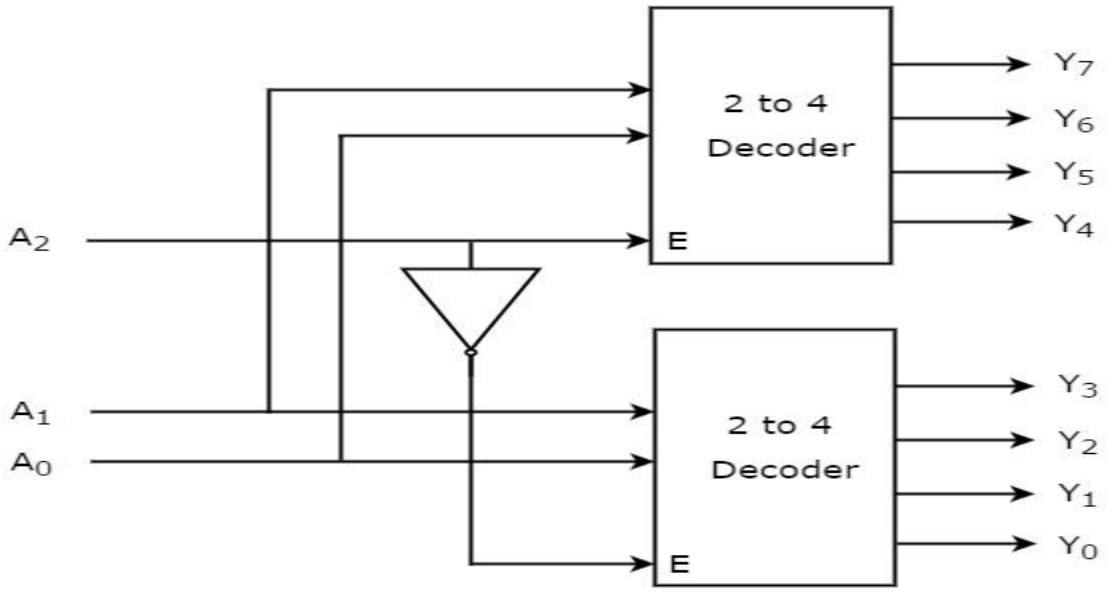
m1m1 is the number of outputs of lower order decoder.

m2m2 is the number of outputs of higher order decoder.

Here, m1m1 = 4 and m2m2 = 8. Substitute, these two values in the above formula.

Requirednumberof2to4decoders=84=2Requirednumberof2to4decoders=84=2

Therefore, we require two 2 to 4 decoders for implementing one 3 to 8 decoder. The **block diagram** of 3 to 8 decoder using 2 to 4 decoders is shown in the following figure.



The parallel inputs A_1 & A_0 are applied to each 2 to 4 decoder. The complement of input A_2 is connected to Enable, E of lower 2 to 4 decoder in order to get the outputs, Y_3 to Y_0. These are the **lower four min terms**. The input, A_2 is directly connected to Enable, E of upper 2 to 4 decoder in order to get the outputs, Y_7 to Y_4. These are the **higher four min terms**.

4 to 16 Decoder

In this section, let us implement **4 to 16 decoder using 3 to 8 decoders**. We know that 3 to 8 Decoder has three inputs A_2, A_1 & A_0 and eight outputs, Y_7 to Y_0. Whereas, 4 to 16 Decoder has four inputs A_3, A_2, A_1 & A_0 and sixteen outputs, Y_{15} to Y_0

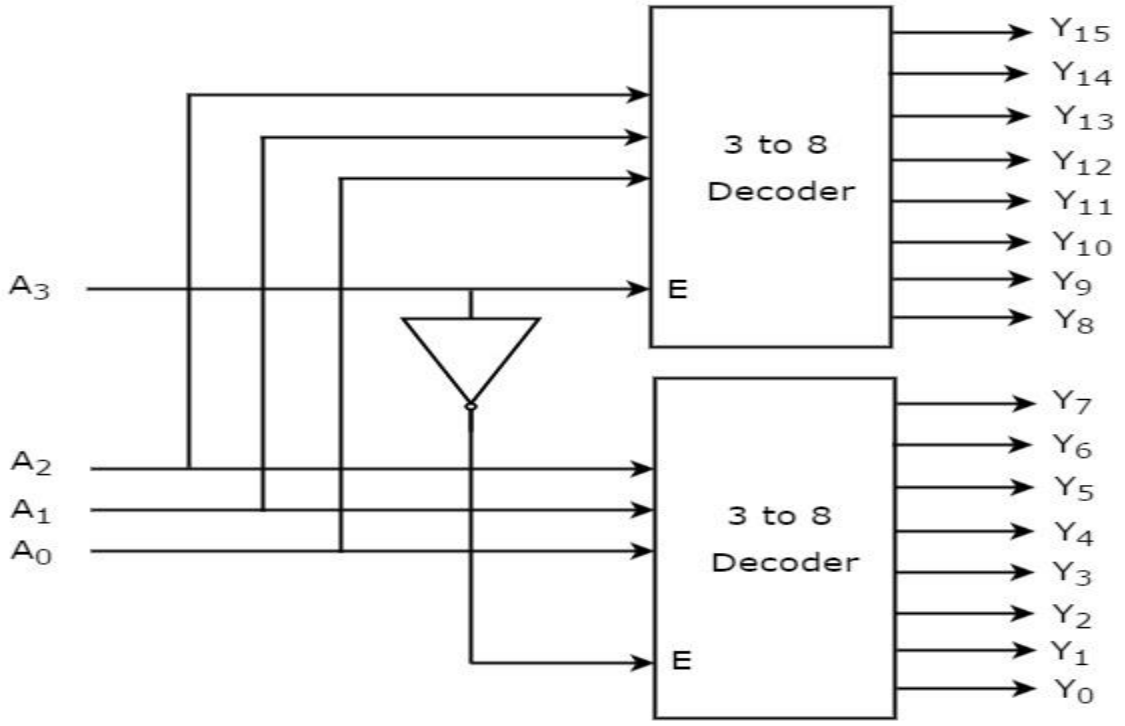
We know the following formula for finding the number of lower order decoders required.

Requirednumberoflowerorderdecoders=m2m1Requirednumberoflowerorderdecoders=m2m1

Substitute, m1m1 = 8 and m2m2 = 16 in the above formula.

Requirednumberof3to8decoders=168=2Requirednumberof3to8decoders=168=2

Therefore, we require two 3 to 8 decoders for implementing one 4 to 16 decoder. The **block diagram** of 4 to 16 decoder using 3 to 8 decoders is shown in the following figure.



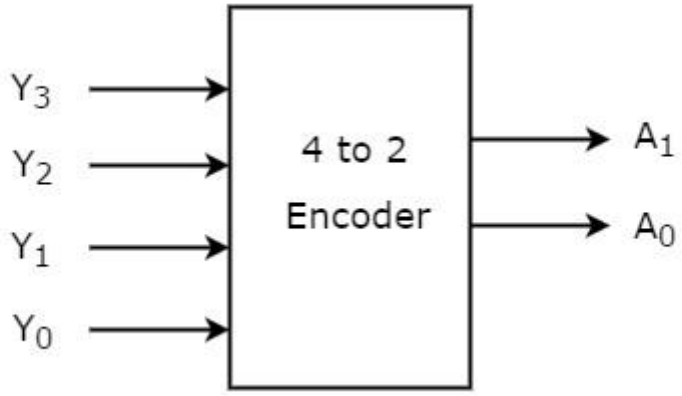
The parallel inputs A_2, A_1 & A_0 are applied to each 3 to 8 decoder. The complement of input, A3 is connected to Enable, E of lower 3 to 8 decoder in order to get the outputs, Y_7 to Y_0. These are the **lower eight min terms**. The input, A_3 is directly connected to Enable, E of upper 3 to 8 decoder in order to get the outputs, Y_{15} to Y_8. These are the **higher eight min terms**.

Q.3. Explain Encoders.

An **Encoder** is a combinational circuit that performs the reverse operation of Decoder. It has maximum of 2^n input lines and 'n' output lines. It will produce a binary code equivalent to the input, which is active High. Therefore, the encoder encodes 2^n input lines with 'n' bits. It is optional to represent the enable signal in encoders.

4 to 2 Encoder

Let 4 to 2 Encoder has four inputs Y_3, Y_2, Y_1 & Y_0 and two outputs A_1 & A_0. The **block diagram** of 4 to 2 Encoder is shown in the following figure.



At any time, only one of these 4 inputs can be '1' in order to get the respective binary code at the output. The **Truth table** of 4 to 2 encoder is shown below.

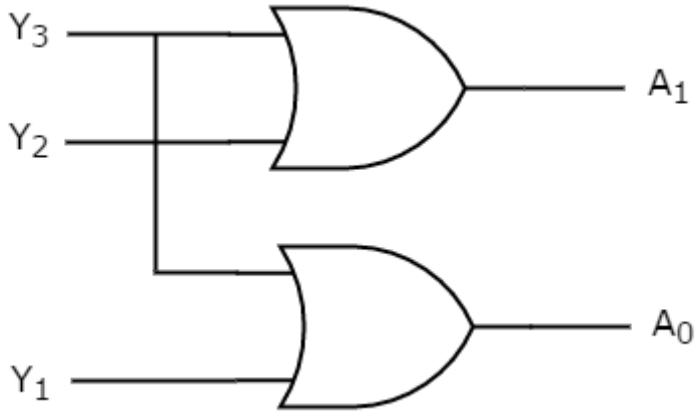
Inputs				**Outputs**	
Y_3	**Y_2**	**Y_1**	**Y_0**	**A_1**	**A_0**
0	0	0	1	0	0
0	0	1	0	0	1
0	1	0	0	1	0
1	0	0	0	1	1

From Truth table, we can write the **Boolean functions** for each output as

$$A1=Y3+Y2A1=Y3+Y2$$

$$A0=Y3+Y1A0=Y3+Y1$$

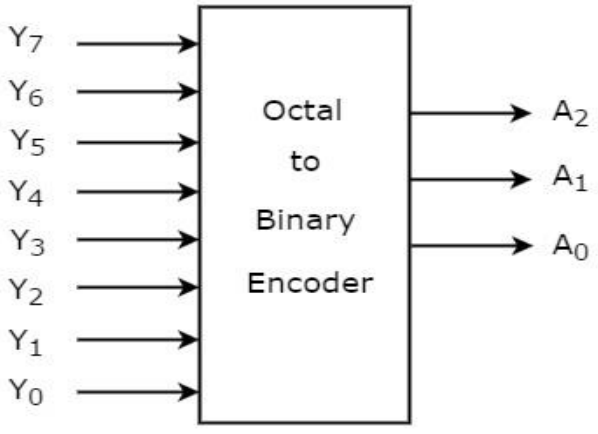
We can implement the above two Boolean functions by using two input OR gates. The **circuit diagram** of 4 to 2 encoder is shown in the following figure.



The above circuit diagram contains two OR gates. These OR gates encode the four inputs with two bits

Octal to Binary Encoder

Octal to binary Encoder has eight inputs, Y_7 to Y_0 and three outputs A_2, A_1 & A_0. Octal to binary encoder is nothing but 8 to 3 encoder. The **block diagram** of octal to binary Encoder is shown in the following figure.



At any time, only one of these eight inputs can be '1' in order to get the respective binary code. The **Truth table** of octal to binary encoder is shown below.

Inputs								Outputs		
Y_7	Y_6	Y_5	Y_4	Y_3	Y_2	Y_1	Y_0	A_2	A_1	A_0
0	0	0	0	0	0	0	1	0	0	0
0	0	0	0	0	0	1	0	0	0	1
0	0	0	0	0	1	0	0	0	1	0
0	0	0	0	1	0	0	0	0	1	1
0	0	0	1	0	0	0	0	1	0	0
0	0	1	0	0	0	0	0	1	0	1
0	1	0	0	0	0	0	0	1	1	0
1	0	0	0	0	0	0	0	1	1	1

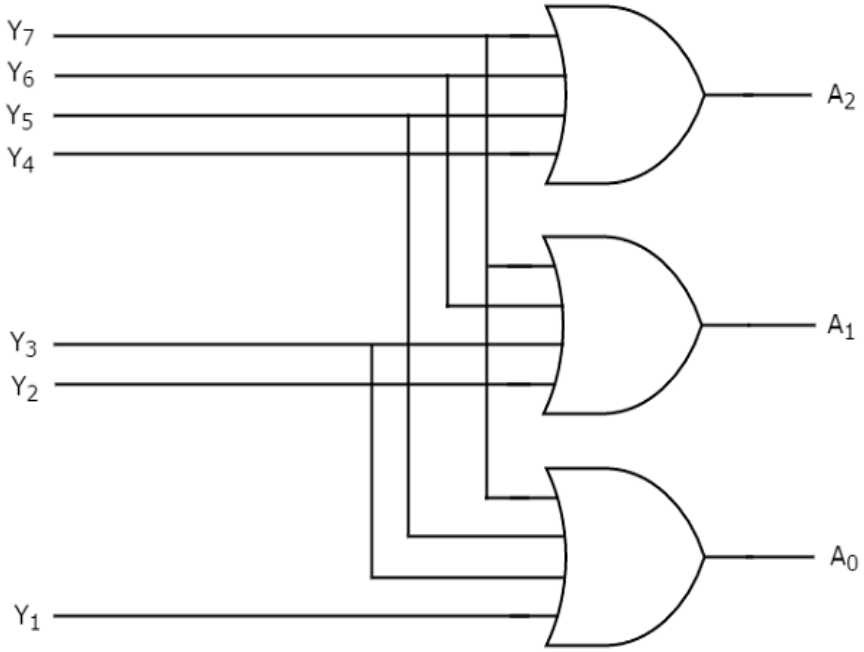
From Truth table, we can write the **Boolean functions** for each output as

$$A2=Y7+Y6+Y5+Y4A2=Y7+Y6+Y5+Y4$$

$$A1=Y7+Y6+Y3+Y2A1=Y7+Y6+Y3+Y2$$

$$A0=Y7+Y5+Y3+Y1A0=Y7+Y5+Y3+Y1$$

We can implement the above Boolean functions by using four input OR gates. The **circuit diagram** of octal to binary encoder is shown in the following figure.



The above circuit diagram contains three 4-input OR gates. These OR gates encode the eight inputs with three bits.

Drawbacks of Encoder

Following are the drawbacks of normal encoder.

- There is an ambiguity, when all outputs of encoder are equal to zero. Because, it could be the code corresponding to the inputs, when only least significant input is one or when all inputs are zero.
- If more than one input is active High, then the encoder produces an output, which may not be the correct code. For **example**, if both Y_3 and Y_6 are '1', then the encoder produces 111 at the output. This is neither equivalent code corresponding to Y_3, when it is '1' nor the equivalent code corresponding to Y_6, when it is '1'.

So, to overcome these difficulties, we should assign priorities to each input of encoder. Then, the output of encoder will be the binarybinary code corresponding to the active High inputss, which has higher priority. This encoder is called as **priority encoder**.

Q4.Explain Priority Encoders.

Priority Encoder

A 4 to 2 priority encoder has four inputs Y_3, Y_2, Y_1 & Y_0 and two outputs A_1 & A_0. Here, the input, Y_3 has the highest priority, whereas the input, Y_0 has the lowest priority. In this case, even if more than one input is '1' at the same time, the output will be the binarybinary code corresponding to the input, which is having **higher priority**.

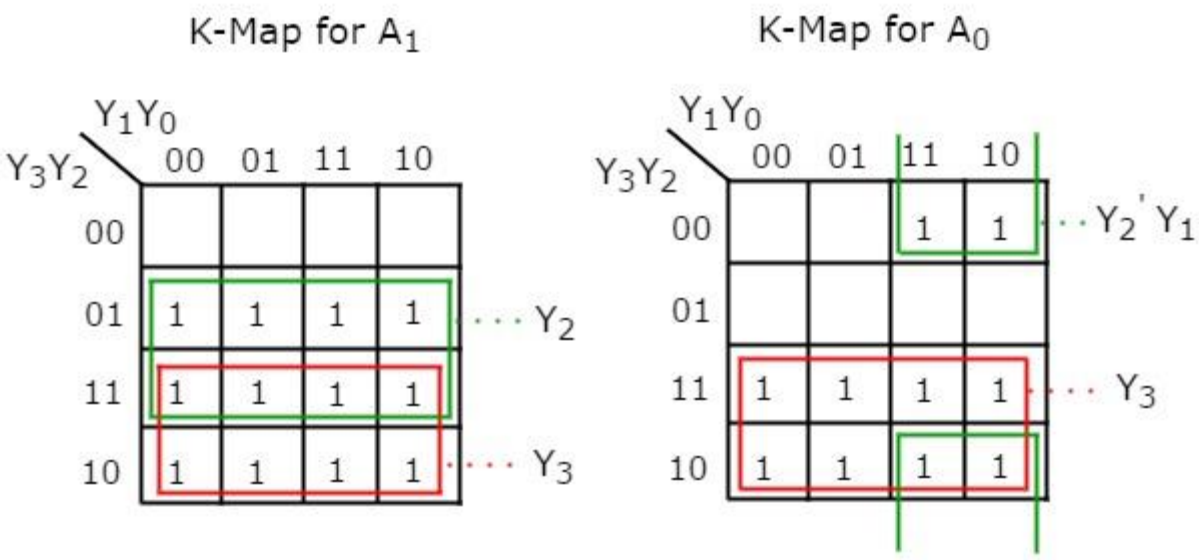
We considered one more **output, V** in order to know, whether the code available at outputs is valid or not.

- If at least one input of the encoder is '1', then the code available at outputs is a valid one. In this case, the output, V will be equal to 1.
- If all the inputs of encoder are '0', then the code available at outputs is not a valid one. In this case, the output, V will be equal to 0.

The **Truth table** of 4 to 2 priority encoder is shown below.

Inputs				**Outputs**		
Y_3	**Y_2**	**Y_1**	**Y_0**	**A_1**	**A_0**	**V**
0	0	0	0	0	0	0
0	0	0	1	0	0	1
0	0	1	x	0	1	1
0	1	x	x	1	0	1
1	x	x	x	1	1	1

Use **4 variable K-maps** for getting simplified expressions for each output.



The simplified **Boolean functions** are

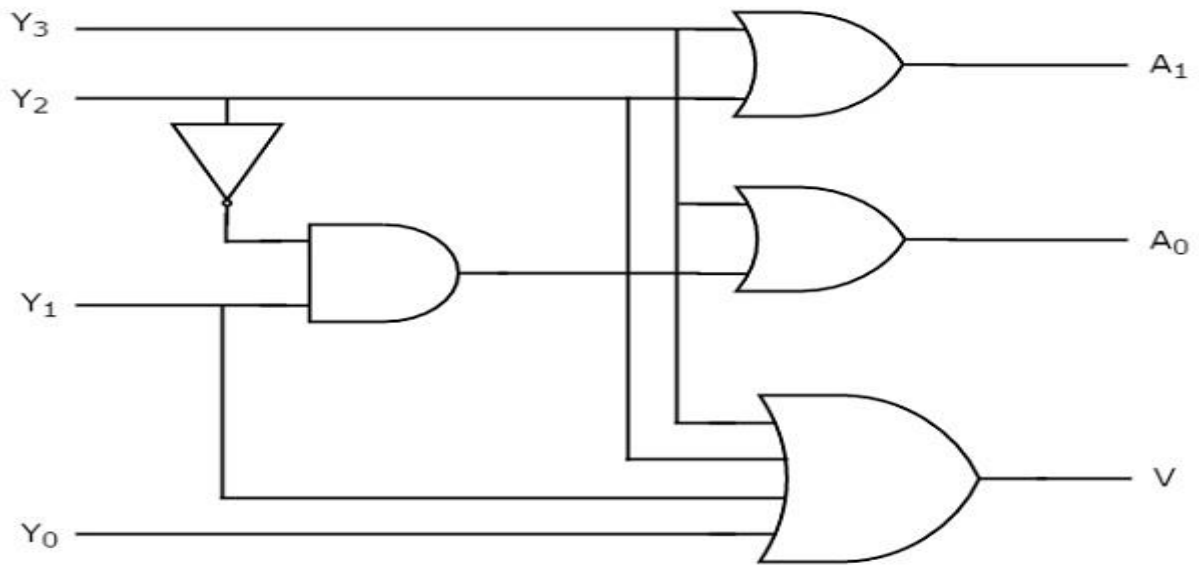
$$A1=Y3+Y2A1=Y3+Y2$$

$$A0=Y3+Y2′Y1A0=Y3+Y2′Y1$$

Similarly, we will get the Boolean function of output, V as

$$V=Y3+Y2+Y1+Y0V=Y3+Y2+Y1+Y0$$

We can implement the above Boolean functions using logic gates. The **circuit diagram** of 4 to 2 priority encoder is shown in the following figure.

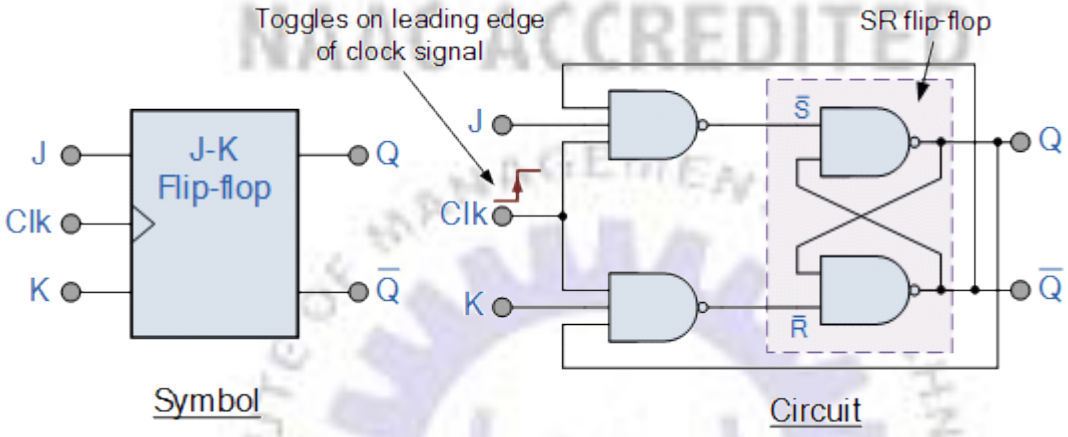


The above circuit diagram contains two 2-input OR gates, one 4-input OR gate, one 2input AND gate & an inverter. Here AND gate & inverter combination are used for producing a valid code at the outputs, even when multiple inputs are equal to '1' at the same time. Hence, this circuit encodes the four inputs with two bits based on the **priority** assigned to each input.

Q5. Explain Flip Flops

The JK Flip Flop

The JK Flip-flop is similar to the SR Flip-flop but there is no change in state when the J and K inputs are both LOW



The basic S-R NAND flip-flop circuit has many advantages and uses in sequential logic circuits but it suffers from two basic switching problems.

- 1. the Set = 0 and Reset = 0 condition (S = R = 0) must always be avoided
- 2. if Set or Reset change state while the enable (EN) input is high the correct latching action may not occur

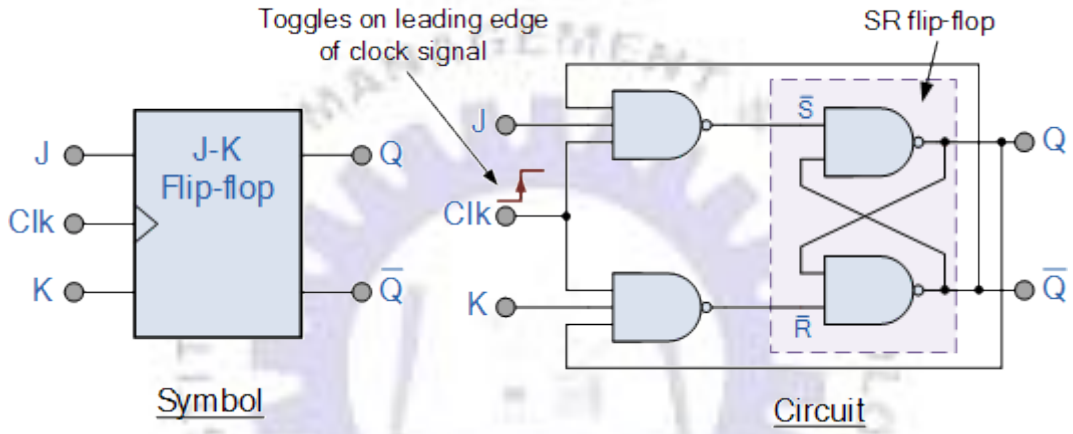
Then to overcome these two fundamental design problems with the SR flip-flop design, the **JK flip Flop** was developed.

This simple **JK flip Flop** is the most widely used of all the flip-flop designs and is considered to be a universal flip-flop circuit. The two inputs labelled "J" and "K" are not shortened abbreviated letters of other words, such as "S" for Set and "R" for Reset, but are themselves autonomous letters chosen by its inventor Jack Kilby to distinguish the flip-flop design from other types.

The sequential operation of the JK flip flop is exactly the same as for the previous SR flip-flop with the same "Set" and "Reset" inputs. The difference this time is that the "JK flip flop" has no invalid or forbidden input states of the SR Latch even when S and R are both at logic "1".

The **JK flip flop** is basically a gated SR flip-flop with the addition of a clock input circuitry that prevents the illegal or invalid output condition that can occur when both inputs S and R are equal to logic level "1". Due to this additional clocked input, a JK flip-flop has four possible input combinations, "logic 1", "logic 0", "no change" and "toggle". The symbol for a JK flip flop is similar to that of an *SR Bistable Latch* as seen in the previous tutorial except for the addition of a clock input.

The Basic JK Flip-flop



Both the S and the R inputs of the previous SR bistable have now been replaced by two inputs called the J and K inputs, respectively after its inventor Jack Kilby. Then this equates to: J = S and K = R.

The two 2-input AND gates of the gated SR bistable have now been replaced by two 3-input NAND gates with the third input of each gate connected to the outputs at Q and Q. This cross coupling of the SR flip-flop allows the previously invalid condition of S = "1" and R = "1" state to be used to produce a "toggle action" as the two inputs are now interlocked.

If the circuit is now "SET" the J input is inhibited by the "0" status of Q through the lower NAND gate. If the circuit is "RESET" the K input is inhibited by the "0" status of Q through the upper NAND gate. As Q and Q are always different we can use them to control the input. When both inputs J and K are equal to logic "1", the JK flip flop toggles as shown in the following truth table.

The Truth Table for the JK Function

	Clock	Input		Output		Description
	Clk	J	K	Q	Q	
same as for the SR Latch	X	0	0	1	0	Memory no change
	X	0	0	0	1	
	⁻↓_	0	1	1	0	Reset Q » 0
	X	0	1	0	1	
	⁻↓_	1	0	0	1	Set Q » 1
	X	1	0	1	0	
toggle action	⁻↓_	1	1	0	1	Toggle
	⁻↓_	1	1	1	0	

Then the JK flip-flop is basically an SR flip flop with feedback which enables only one of its two input terminals, either SET or RESET to be active at any one time thereby eliminating the invalid condition seen previously in the SR flip flop circuit.

Also when both the J and the K inputs are at logic level “1” at the same time, and the clock input is pulsed “HIGH”, the circuit will “toggle” from its SET state to a RESET state, or visa-versa. This results in the JK flip flop acting more like a T-type toggle flip-flop when both terminals are “HIGH”.

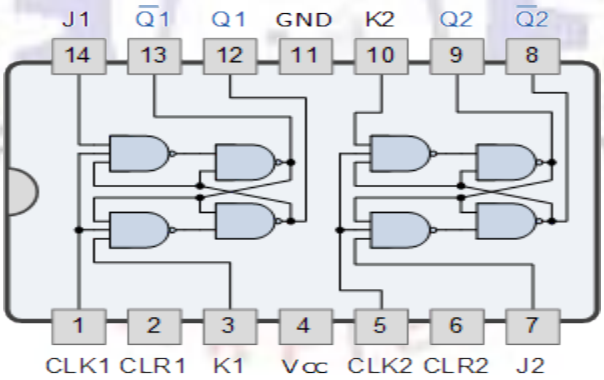
Although this circuit is an improvement on the clocked SR flip-flop it still suffers from timing problems called “race” if the output Q changes state before the timing pulse of the clock input has time to go “OFF”. To avoid this the timing pulse period (T) must be kept as short as possible (high frequency). As this is sometimes not possible with modern TTL IC’s the much improved **Master-Slave JK Flip-flop** was developed.

Master-Slave JK Flip-flop

The master-slave flip-flop eliminates all the timing problems by using two SR flip-flops connected together in a series configuration. One flip-flop acts as the "Master" circuit, which triggers on the leading edge of the clock pulse while the other acts as the "Slave" circuit, which triggers on the falling edge of the clock pulse. This results in the two sections, the master section and the slave section being enabled during opposite half-cycles of the clock signal.

The TTL 74LS73 is a Dual JK flip-flop IC, which contains two individual JK type bistable's within a single chip enabling single or master-slave toggle flip-flops to be made. Other JK flip flop IC's include the 74LS107 Dual JK flip-flop with clear, the 74LS109 Dual positive-edge triggered JK flip flop and the 74LS112 Dual negative-edge triggered flip-flop with both preset and clear inputs.

Dual JK Flip-flop 74LS73



Other Popular JK Flip-flop ICs

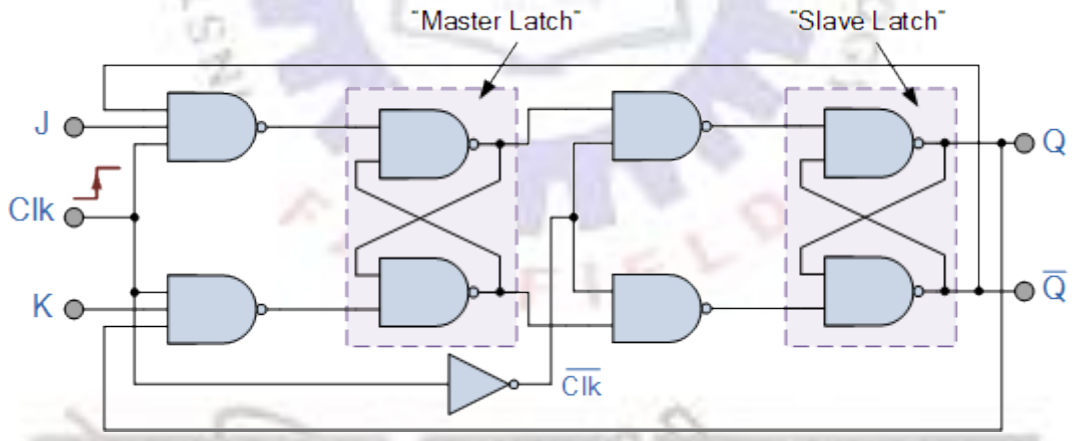
Device Number	Subfamily	Device Description
74LS73	LS TTL	Dual JK-type Flip Flops with Clear
74LS76	LS TTL	Dual JK-type Flip Flops with Preset and Clear

74LS107	LS TTL	Dual JK-type Flip Flops with Clear
4027B	Standard CMOS	Dual JK-type Flip Flop

The Master-Slave JK Flip-flop

The **Master-Slave Flip-Flop** is basically two gated SR flip-flops connected together in a series configuration with the slave having an inverted clock pulse. The outputs from Q and Q from the "Slave" flip-flop are fed back to the inputs of the "Master" with the outputs of the "Master" flip flop being connected to the two inputs of the "Slave" flip flop. This feedback configuration from the slave's output to the master's input gives the characteristic toggle of the JK flip flop as shown below.

The Master-Slave JK Flip Flop



The input signals J and K are connected to the gated "master" SR flip flop which "locks" the input condition while the clock (Clk) input is "HIGH" at logic level "1". As the clock input of the "slave" flip flop is the inverse (complement) of the "master" clock input, the "slave" SR flip flop does not toggle. The outputs from the "master" flip flop are only "seen" by the gated "slave" flip flop when the clock input goes "LOW" to logic level "0".

When the clock is "LOW", the outputs from the "master" flip flop are latched and any additional changes to its inputs are ignored. The gated "slave" flip flop now responds to the state of its inputs passed over by the "master" section.

Then on the “Low-to-High” transition of the clock pulse the inputs of the “master” flip flop are fed through to the gated inputs of the “slave” flip flop and on the “High-to-Low” transition the same inputs are reflected on the output of the “slave” making this type of flip flop edge or pulse-triggered.

Then, the circuit accepts input data when the clock signal is “HIGH”, and passes the data to the output on the falling-edge of the clock signal. In other words, the **Master-Slave JK Flip flop** is a “Synchronous” device as it only passes data with the timing of the clock signal.

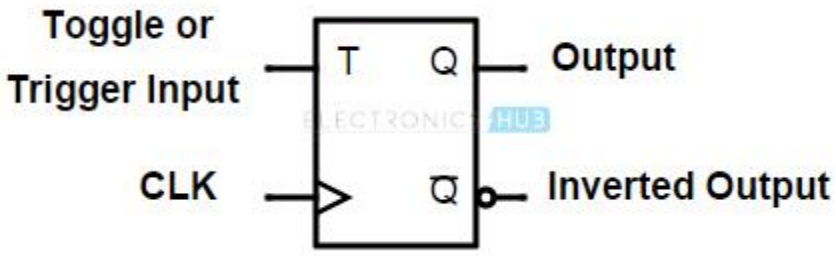
In the next tutorial about **Sequential Logic Circuits**, we will look at *Multivibrators* that are used as waveform generators to produce the clock signals to switch sequential circuits.

Designing of T Flip Flop

T flip – flop is also known as “Toggle Flip – flop”. To avoid the occurrence of intermediate state in SR flip – flop, we should provide only one input to the flip – flop called Trigger input or Toggle input (T). Then the flip – flop acts as a Toggle switch. Toggling means ‘Changing the next state output to complement of the present state output’.

We can design the T flip – flop by making simple modifications to the JK flip – flop. The T flip – flop is a single input device and hence by connecting J and K inputs together and giving them with single input called T we can convert a JK flip – flop into T flip – flop. So a T flip – flop is sometimes called as single input JK flip – flop.

The logic symbol of T flip – flop is shown below. It has one Toggle input (T) & one clock signal input (CLK).



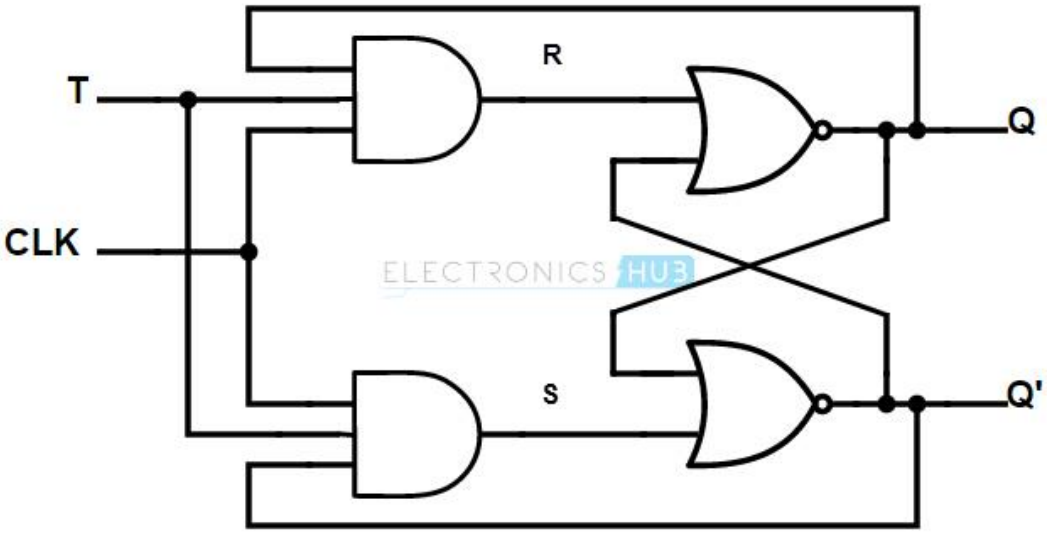
T Flip – flop Circuit

We can construct a T flip – flop by any of the following methods.

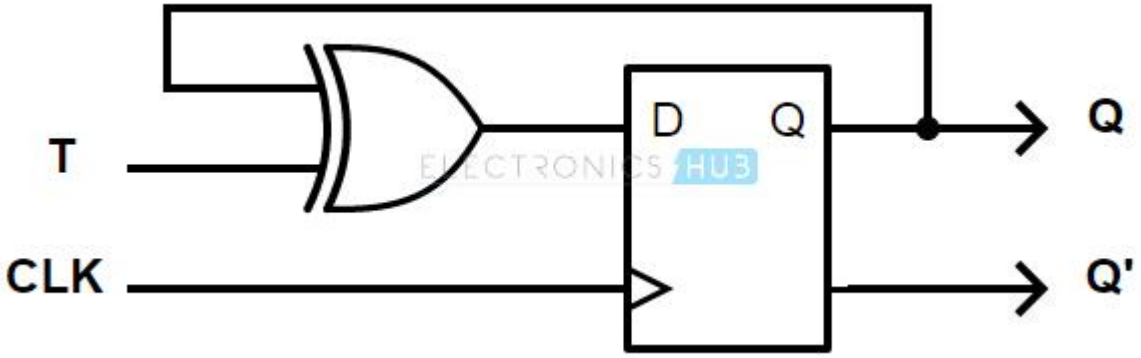
Connecting the output feedback to the input, in SR flip – flop.
Connecting the XOR of T input and Q PREVIOUS output to the Data input, in D flip – flop.
Hard – wiring the J and K inputs together and connecting it to T input, in JK flip – flop.

Construction

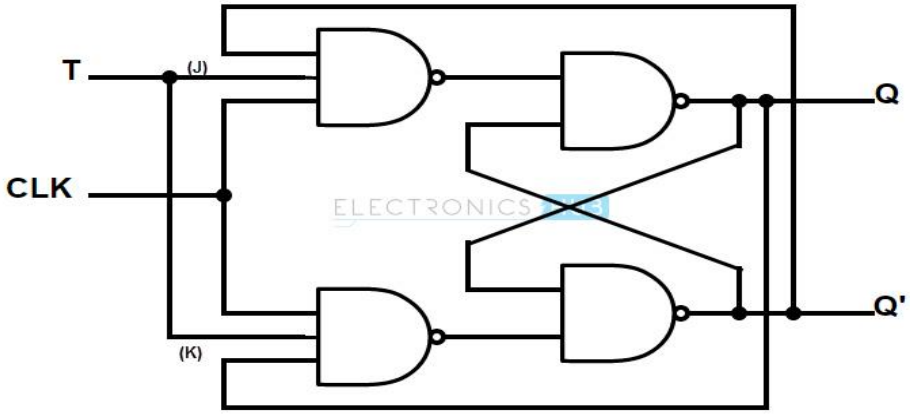
We can construct a T flip – flop by connecting AND gates as input to the NOR gate SR latch. And these AND gate inputs are fed back with the present state output Q and its complement Q' to each AND gate. A toggle input (T) is connected in common to both the AND gates as an input. The AND gates are also connected with common Clock (CLK) signal. In the T flip – flop, a pulse train of narrow triggers are provided as input (T) which will cause the change in output state of flip – flop. So these flip – flops are also called Toggle flip – flops. The circuit diagram of a T flip – flop constructed from SR latch is shown below



Similarly, a T flip – flop can be constructed by modifying D flip – flop. In D flip – flop, the output QPREV is XORed with the T input and given at the D input. The circuit of a T flip – flop constructed from a D flip – flop is shown below.



The simplest of the constructions of a D flip – flop is with JK flip – flop. The J input and K input of the JK flip – flop are connected together and provided with the T input. The logic circuit of a T flip – flop constructed from a JK flip – flop is shown below.



Working

T flip – flop is an edge triggered device i.e. the low to high or high to low transitions on a clock signal of narrow triggers that is provided as input will cause the change in output state of flip – flop.

Truth Table of T flip – flop

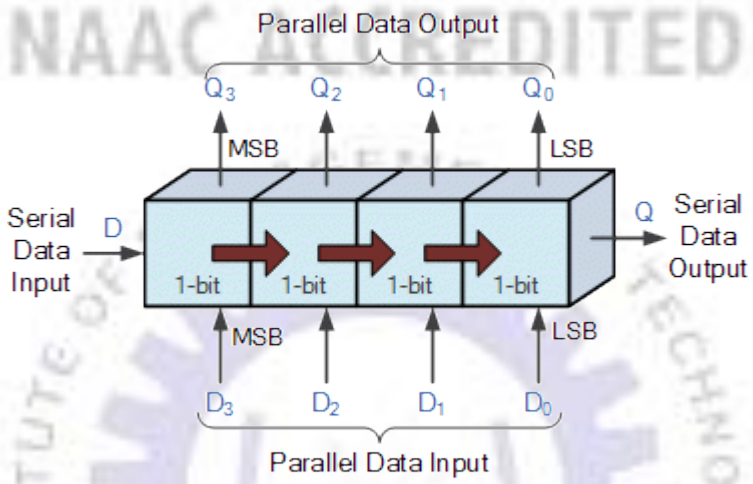
The truth table of a T flip – flop is shown below.

	Previous		Next	
T	Q_{Prev}	Q'_{Prev}	Q_{Next}	Q'_{Next}
0	0	1	0	1
0	1	0	1	0
1	0	1	1	0
1	1	0	0	1

Q6. Explain Shift Registers.

The Shift Register

The Shift Register is another type of sequential logic circuit that can be used for the storage or the transfer of binary data



This sequential device loads the data present on its inputs and then moves or "shifts" it to its output once every clock cycle, hence the name **Shift Register**.

A *shift register* basically consists of several single bit "D-Type Data Latches", one for each data bit, either a logic "0" or a "1", connected together in a serial type daisy-chain arrangement so that the output from one data latch becomes the input of the next latch and so on.

Data bits may be fed in or out of a shift register serially, that is one after the other from either the left or the right direction, or all together at the same time in a parallel configuration.

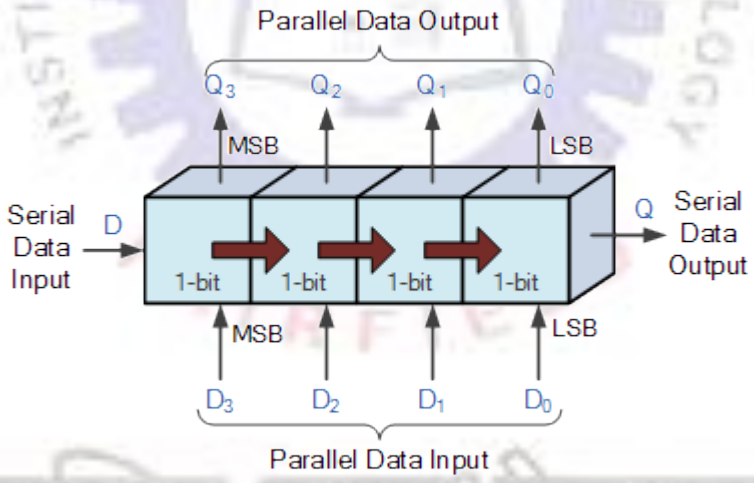
The number of individual data latches required to make up a single **Shift Register** device is usually determined by the number of bits to be stored with the most common being 8-bits (one byte) wide constructed from eight individual data latches.

Shift Registers are used for data storage or for the movement of data and are therefore commonly used inside calculators or computers to store data such as two binary numbers before they are added together, or to convert the data from either a serial to parallel or parallel to serial format. The individual data latches that make up a single shift register are all driven by a common clock (Clk) signal making them synchronous devices.

Shift register IC's are generally provided with a *clear* or *reset* connection so that they can be "SET" or "RESET" as required. Generally, shift registers operate in one of four different modes with the basic movement of data through a shift register being:

- Serial-in to Parallel-out (SIPO) - the register is loaded with serial data, one bit at a time, with the stored data being available at the output in parallel form.
- Serial-in to Serial-out (SISO) - the data is shifted serially "IN" and "OUT" of the register, one bit at a time in either a left or right direction under clock control.
- Parallel-in to Serial-out (PISO) - the parallel data is loaded into the register simultaneously and is shifted out of the register serially one bit at a time under clock control.
- Parallel-in to Parallel-out (PIPO) - the parallel data is loaded simultaneously into the register, and transferred together to their respective outputs by the same clock pulse.

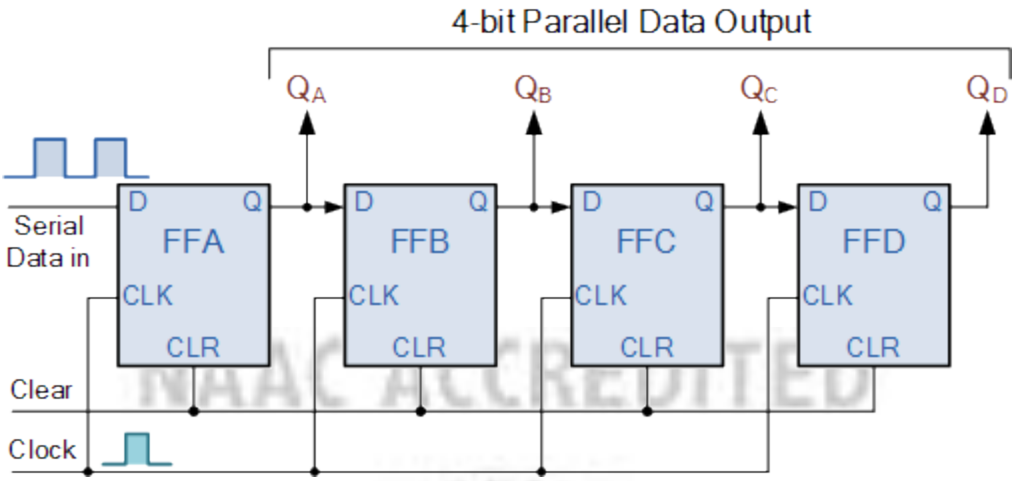
The effect of data movement from left to right through a shift register can be presented graphically as:



Also, the directional movement of the data through a shift register can be either to the left, (left shifting) to the right, (right shifting) left-in but right-out, (rotation) or both left and right shifting within the same register thereby making it *bidirectional*. In this tutorial it is assumed that all the data shifts to the right, (right shifting).

Serial-in to Parallel-out (SIPO) Shift Register

4-bit Serial-in to Parallel-out Shift Register



The operation is as follows. Lets assume that all the flip-flops (FFA to FFD) have just been RESET (CLEAR input) and that all the outputs Q_A to Q_D are at logic level "0" ie, no parallel data output.

If a logic "1" is connected to the DATA input pin of FFA then on the first clock pulse the output of FFA and therefore the resulting Q_A will be set HIGH to logic "1" with all the other outputs still remaining LOW at logic "0". Assume now that the DATA input pin of FFA has returned LOW again to logic "0" giving us one data pulse or 0-1-0.

The second clock pulse will change the output of FFA to logic "0" and the output of FFB and Q_B HIGH to logic "1" as its input D has the logic "1" level on it from Q_A. The logic "1" has now moved or been "shifted" one place along the register to the right as it is now at Q_A.

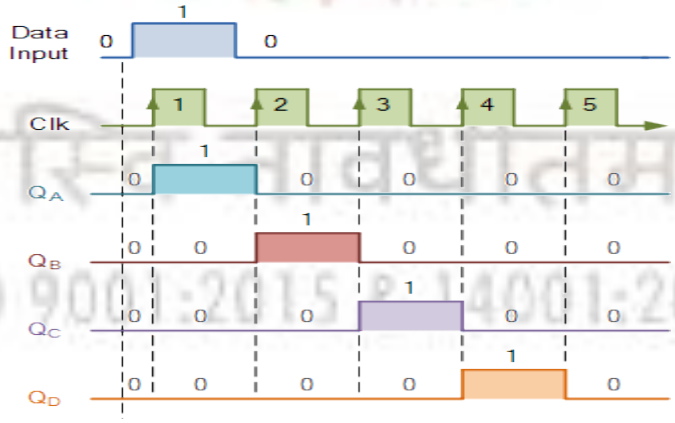
When the third clock pulse arrives this logic "1" value moves to the output of FFC (Q_C) and so on until the arrival of the fifth clock pulse which sets all the outputs Q_A to Q_D back again to logic level "0" because the input to FFA has remained constant at logic level "0".

The effect of each clock pulse is to shift the data contents of each stage one place to the right, and this is shown in the following table until the complete data value of 0-0-0-1 is stored in the register. This data value can now be read directly from the outputs of Q_A to Q_D.

Then the data has been converted from a serial data input signal to a parallel data output. The truth table and following waveforms show the propagation of the logic "1" through the register from left to right as follows.

Basic Data Movement Through A Shift Register

Clock Pulse No	QA	QB	QC	QD
0	0	0	0	0
1	1	0	0	0
2	0	1	0	0
3	0	0	1	0
4	0	0	0	1
5	0	0	0	0



Note that after the fourth clock pulse has ended the 4-bits of data (0-0-0-1) are stored in the register and will remain there provided clocking of the register has stopped. In practice the

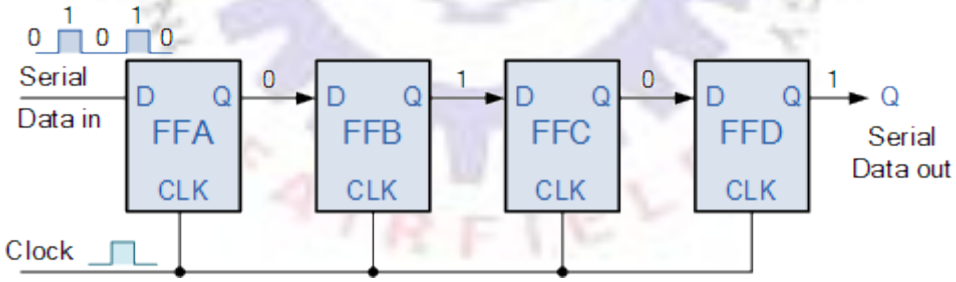
input data to the register may consist of various combinations of logic "1" and "0". Commonly available SIPO IC's include the standard 8-bit 74LS164 or the 74LS594.

Serial-in to Serial-out (SISO) Shift Register

This **shift register** is very similar to the SIPO above, except were before the data was read directly in a parallel form from the outputs Q_A to Q_D, this time the data is allowed to flow straight through the register and out of the other end. Since there is only one output, the DATA leaves the shift register one bit at a time in a serial pattern, hence the name **Serial-in to Serial-Out Shift Register** or **SISO**.

The SISO shift register is one of the simplest of the four configurations as it has only three connections, the serial input (SI) which determines what enters the left hand flip-flop, the serial output (SO) which is taken from the output of the right hand flip-flop and the sequencing clock signal (Clk). The logic circuit diagram below shows a generalized serial-in serial-out shift register.

4-bit Serial-in to Serial-out Shift Register



You may think what's the point of a SISO shift register if the output data is exactly the same as the input data. Well this type of **Shift Register** also acts as a temporary storage device or it can act as a time delay device for the data, with the amount of time delay being controlled by the number of stages in the register, 4, 8, 16 etc or by varying the application of the clock pulses. Commonly available IC's include the 74HC595 8-bit Serial-in to Serial-out Shift Register all with 3-state outputs.

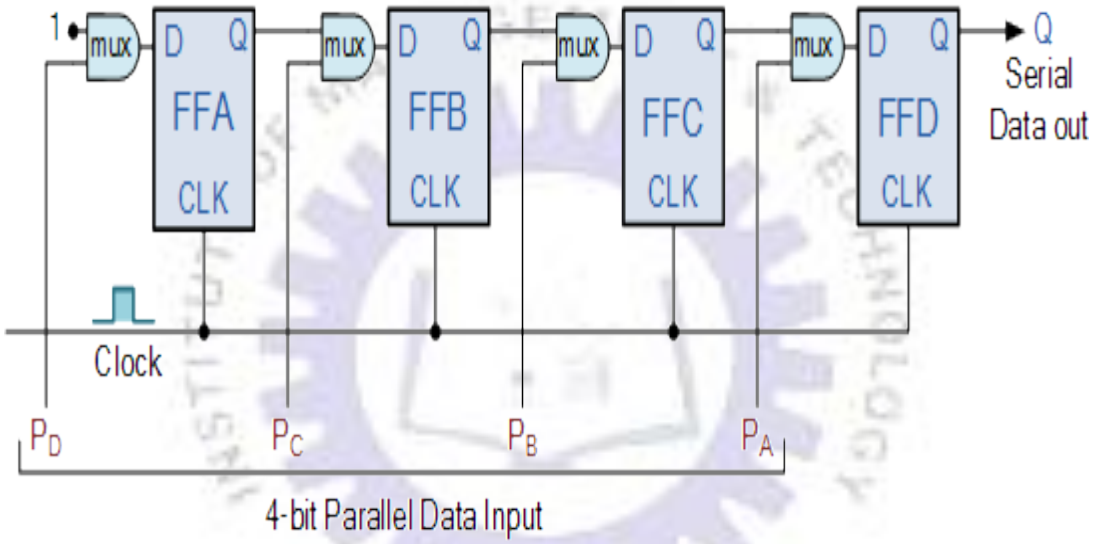
Parallel-in to Serial-out (PISO) Shift Register

The Parallel-in to Serial-out shift register acts in the opposite way to the serial-in to parallel-out one above. The data is loaded into the register in a parallel format in which all the data

bits enter their inputs simultaneously, to the parallel input pins P_A to P_D of the register. The data is then read out sequentially in the normal shift-right mode from the register at Q representing the data present at P_A to P_D.

This data is outputted one bit at a time on each clock cycle in a serial format. It is important to note that with this type of data register a clock pulse is not required to parallel load the register as it is already present, but four clock pulses are required to unload the data.

4-bit Parallel-in to Serial-out Shift Register

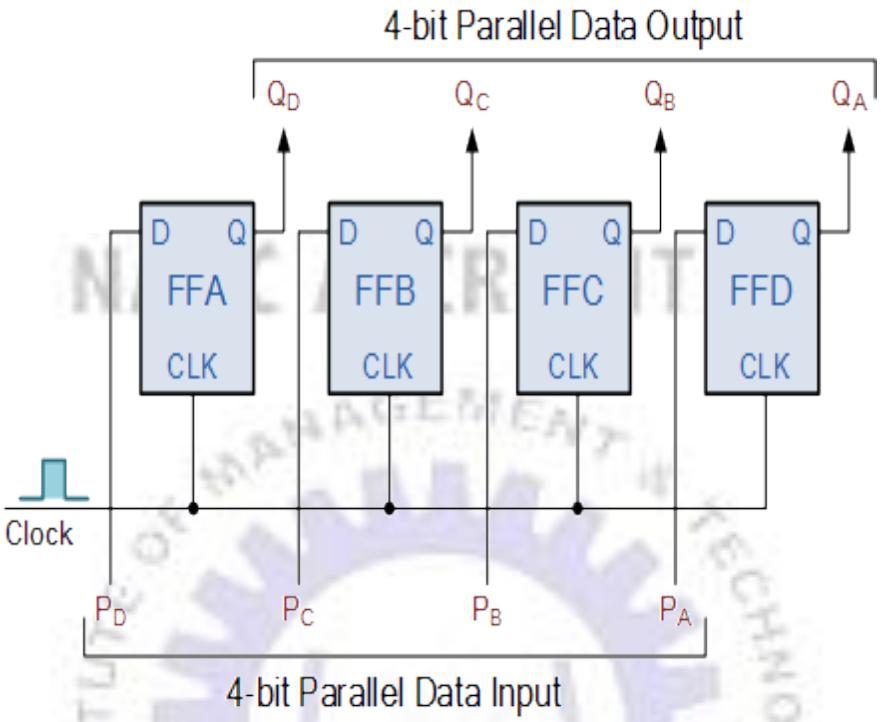


As this type of shift register converts parallel data, such as an 8-bit data word into serial format, it can be used to multiplex many different input lines into a single serial DATA stream which can be sent directly to a computer or transmitted over a communications line. Commonly available IC's include the 74HC166 8-bit Parallel-in/Serial-out Shift Registers.

Parallel-in to Parallel-out (PIPO) Shift Register

The final mode of operation is the Parallel-in to Parallel-out Shift Register. This type of shift register also acts as a temporary storage device or as a time delay device similar to the SISO configuration above. The data is presented in a parallel format to the parallel input pins P_A to P_D and then transferred together directly to their respective output pins Q_A to Q_D by the same clock pulse. Then one clock pulse loads and unloads the register. This arrangement for parallel loading and unloading is shown below.

4-bit Parallel-in to Parallel-out Shift Register



The PIPO shift register is the simplest of the four configurations as it has only three connections, the parallel input (PI) which determines what enters the flip-flop, the parallel output (PO) and the sequencing clock signal (Clk).

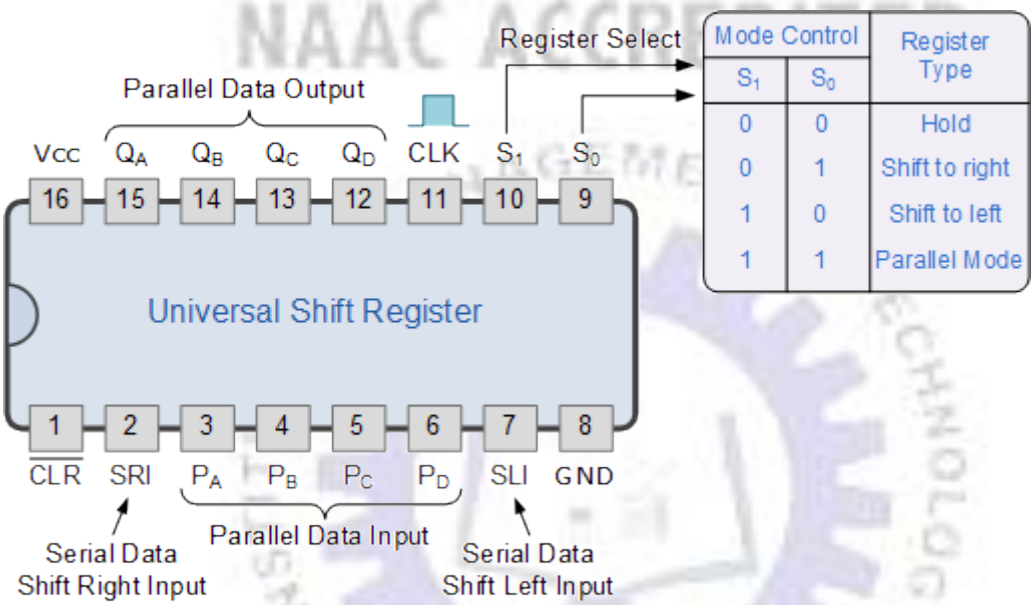
Similar to the Serial-in to Serial-out shift register, this type of register also acts as a temporary storage device or as a time delay device, with the amount of time delay being varied by the frequency of the clock pulses. Also, in this type of register there are no interconnections between the individual flip-flops since no serial shifting of the data is required.

Universal Shift Register

Today, there are many high speed bi-directional "universal" type **Shift Registers** available such as the TTL 74LS194, 74LS195 or the CMOS 4035 which are available as 4-bit multi-function devices that can be used in either serial-to-serial, left shifting, right shifting, serial-to-parallel, parallel-to-serial, or as a parallel-to-parallel multifunction data register, hence their name "Universal".

These universal shift registers can perform any combination of parallel and serial input to output operations but require additional inputs to specify desired function and to pre-load and reset the device. A commonly used universal shift register is the TTL 74LS194 as shown below.

4-bit Universal Shift Register 74LS194



Universal shift registers are very useful digital devices. They can be configured to respond to operations that require some form of temporary memory storage or for the delay of information such as the SISO or PIPO configuration modes or transfer data from one point to another in either a serial or parallel format. Universal shift registers are frequently used in arithmetic operations to shift data to the left or right for multiplication or division.

Shift Register Tutorial Summary

Then to summarise a little about **Shift Registers**

- A simple **Shift Register** can be made using only D-type flip-Flops, one flip-Flop for each data bit.
- The output from each flip-Flop is connected to the D input of the flip-flop at its right.
- Shift registers hold the data in their memory which is moved or "shifted" to their required positions on each clock pulse.

- Each clock pulse shifts the contents of the register one bit position to either the left or the right.
- The data bits can be loaded one bit at a time in a series input (SI) configuration or be loaded simultaneously in a parallel configuration (PI).
- Data may be removed from the register one bit at a time for a series output (SO) or removed all at the same time from a parallel output (PO).
- One application of shift registers is in the conversion of data between serial and parallel, or parallel to serial.
- Shift registers are identified individually as SIPO, SISO, PISO, PIPO, or as a Universal Shift Register with all the functions combined within a single device.

Serial in Serial out (SISO) Shift Register

Serial In Serial Out (SISO) shift registers are a kind of shift registers where both data loading as well as data retrieval to/from the shift register occurs in serial-mode. Figure 1 shows a n-bit synchronous **SISO shift register** sensitive to positive edge of the clock pulse. Here the data word which is to be stored is fed bit-by-bit at the input of the first flip-flop. Further it is seen that the inputs of all other flip-flops (except the first flip-flop FF_1) are driven by the outputs of the preceding ones say for example, the input of FF_2 is driven by the output of FF_1. At last the data stored within the register is obtained at the output pin of the n^{th} flip-flop in serial-fashion.

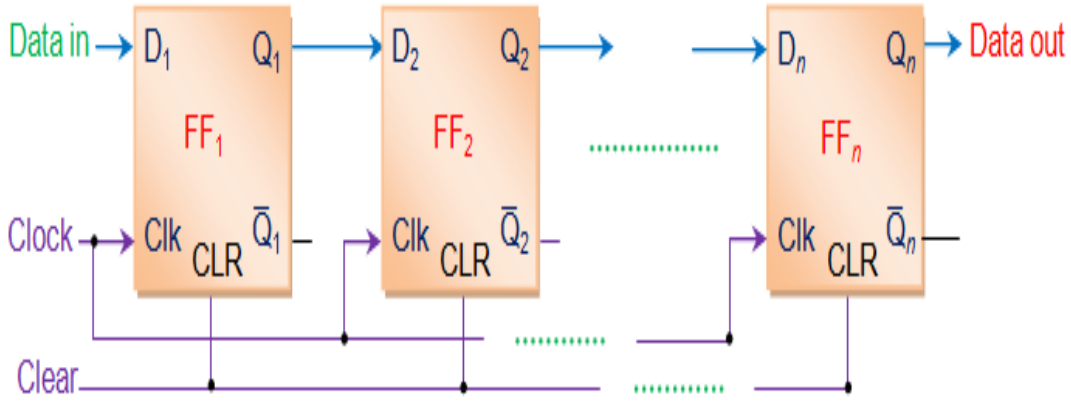


Figure 1 *n*-bit Right-Shift Serial-in Serial-Out Shift Register

Initially all the flip-flops in the register are cleared by applying high on their clear pins. Next the input data word is fed serially to FF_1.

This causes the bit appearing at the D_1 pin (B_1) to be stored into FF_1 as soon as the first

leading edge of the clock appears. Further at the second clock tick, B_1 gets stored into FF_2 while a new bit enters into FF_1 (B_2).

This kind of shift in data bits continues for every rising edge of the clock pulse. This indicates that for every single clock pulse the data within the register moves towards right by a single bit. Thus the design shown in Figure 1 is regarded as a right-shift **SISO shift register**. Following the data transmission as explained, one can note that the first bit of an input word appears at the output of n^{th} flip-flop for the n^{th} clock tick. On applying further clock cycles, one gets the next successive bits of the input data word as the serial output (Table I). The waveforms pertaining to the same are shown by Figure 2.

Clock Cycle	Data in	Q_1	Q_2	...	Q_n = Data out
1	B_1 →	B_1	0	...	0
2	B_2 →	B_2	B_1	...	0
3	B_3 →	B_3	B_2	...	0
4	B_4 →	B_4	B_3	...	0
5	B_5 →	B_5	B_4	...	0
6	B_6 →	B_6	B_5	...	0
.	.	.	.	...	.
.	.	.	.	...	.
.	.	.	.	...	.
n	B_n →	B_n	B_{n-1}	...	B_1
n+1	B_{n+1} →	B_{n+1}	B_n	...	B_2
.	.	.	.	...	.
.	.	.	.	...	.
.	.	.	.	...	.

Serial Output Bits of SISO (Right-Shift) Shift Register

Table I **Data Movement in Right-Shift SISO Shift**

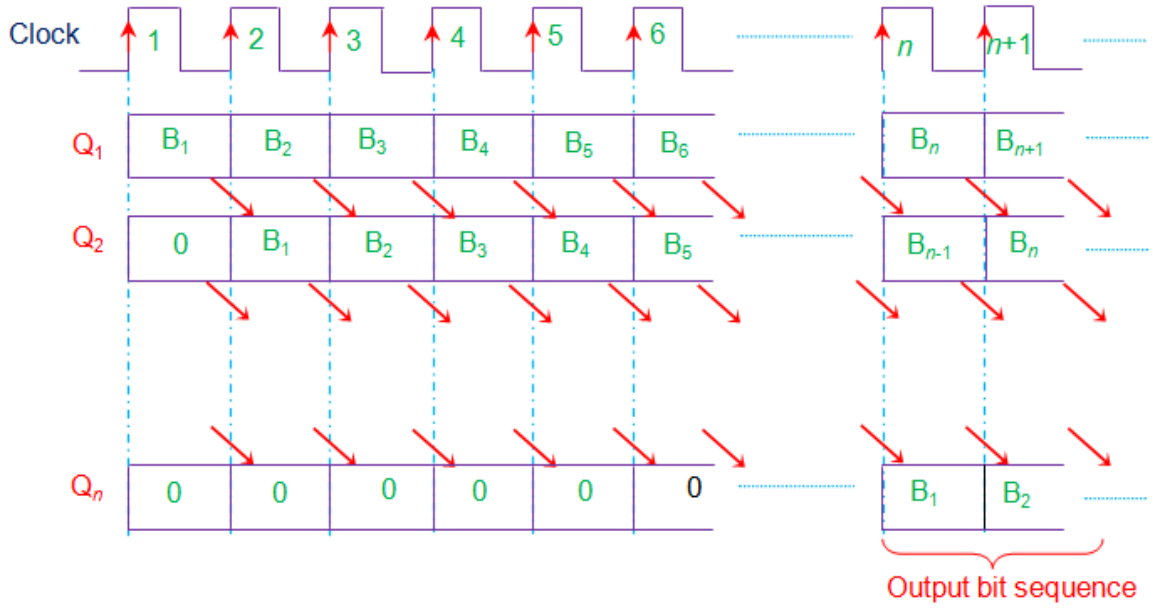


Figure 2 **Output Waveform of *n*-bit Right-Shift SISO Shift Register**

Similar to the right-shift SISO shift-register shown, there can exist a left-shift **SISO shift-register** also (Figure 3). However the working principle remains the same except the fact that

the data movement will be from right to left.

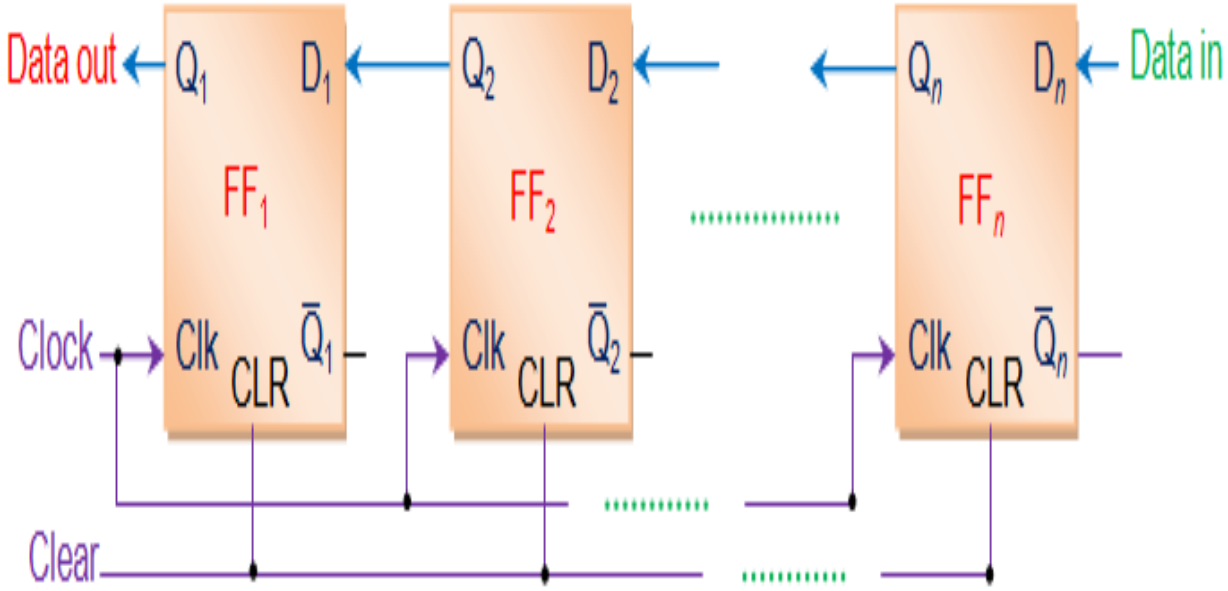


Figure 3 *n*-bit Left-Shift Serial-in Serial-Out Shift Register

Serial in Parallel Out (SIPO) Shift Register

In **Serial In Parallel Out (SIPO) shift registers**, the data is stored into the register serially while it is retrieved from it in parallel-fashion. Figure 1 shows an n-bit synchronous **SIPO shift register** sensitive to positive edge of the clock pulse. Here the data word which is to be stored (Data in) is fed serially at the input of the first flip-flop (D_1 of FF_1). It is also seen that the inputs of all other flip-flops (except the first flip-flop FF_1) are driven by the outputs of the preceding ones say for example, the input of FF_2 is driven by the output of FF_1. In this kind of shift register, the data stored within the register is obtained as a parallel-output data word (Data out) at the individual output pins of the flip-flops (Q_1 to Q_n).

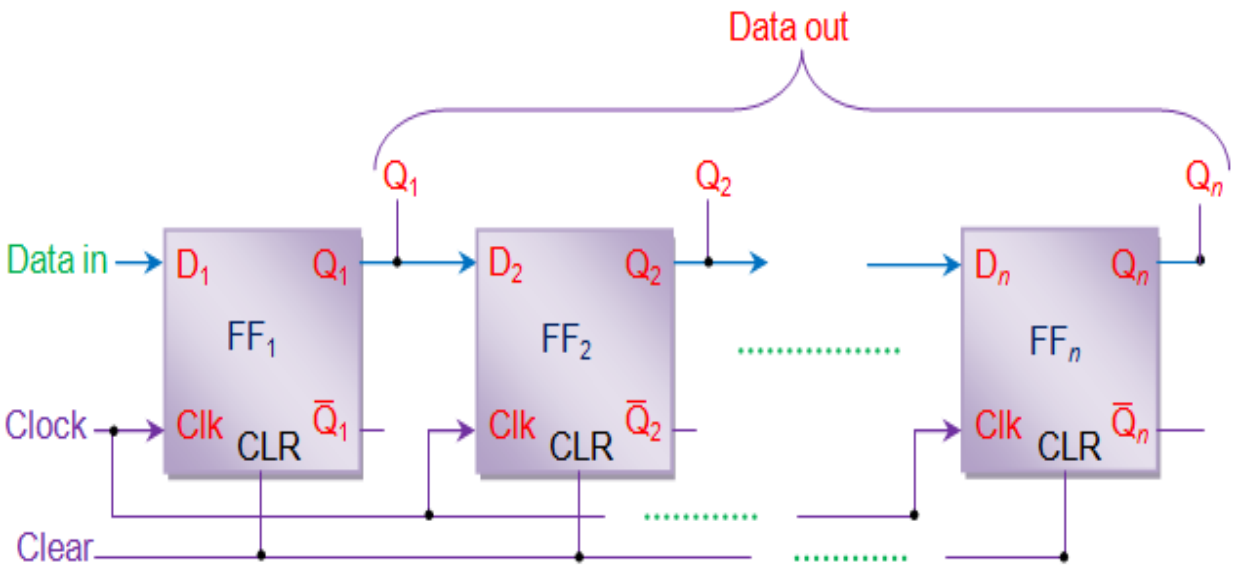


Figure 1 *n*-bit Serial-In Parallel-Out Right-Shift Shift Register

In general, the register contents are cleared by applying high on the clear pins of all the flip-flops at the initial stage. After this, the first bit, B_1 of the input data word is fed at the D_1 pin of FF_1.

This bit (B_1) will enter into FF_1, get stored and thereby appears at its output Q_1 on the appearance of first leading edge of the clock. Further at the second clock tick, the bit B_1 right-shifts and gets stored into FF_2 while appearing at its output pin Q_2 while a new bit, B_2 enters into FF_1. Similarly at each clock tick the data within the register moves towards right by a single bit while a new bit of the input word enters into the register. Meanwhile one can extract the bits stored within the register in parallel-fashion at the individual flip-flop outputs. Analyzing on the same grounds, one can note that the n-bit input data word is obtained as an n-bit output data word from the shift register at the rising edge of the n^{th} clock pulse. This working of the shift-register can be summarized as in Table I and the corresponding wave forms are given by Figure 2.

Table I Data Movement in Right-Shift SIPO Shift Register

Clock Cycle	Data in	Q_1	Q_2	...	Q_n
1	B_1	B_1	0	...	0
2	B_2	B_2	B_1	...	0
3	B_3	B_3	B_2	...	0
4	B_4	B_4	B_3	...	0
5	B_5	B_5	B_4	...	0
6	B_6	B_6	B_5	...	0
.	.	.	.	...	.
.	.	.	.		.
.	.	.	.		.
n	B_n	B_n	B_{n-1}	...	B_1

Output of SIPO (right-shift) Shift Register

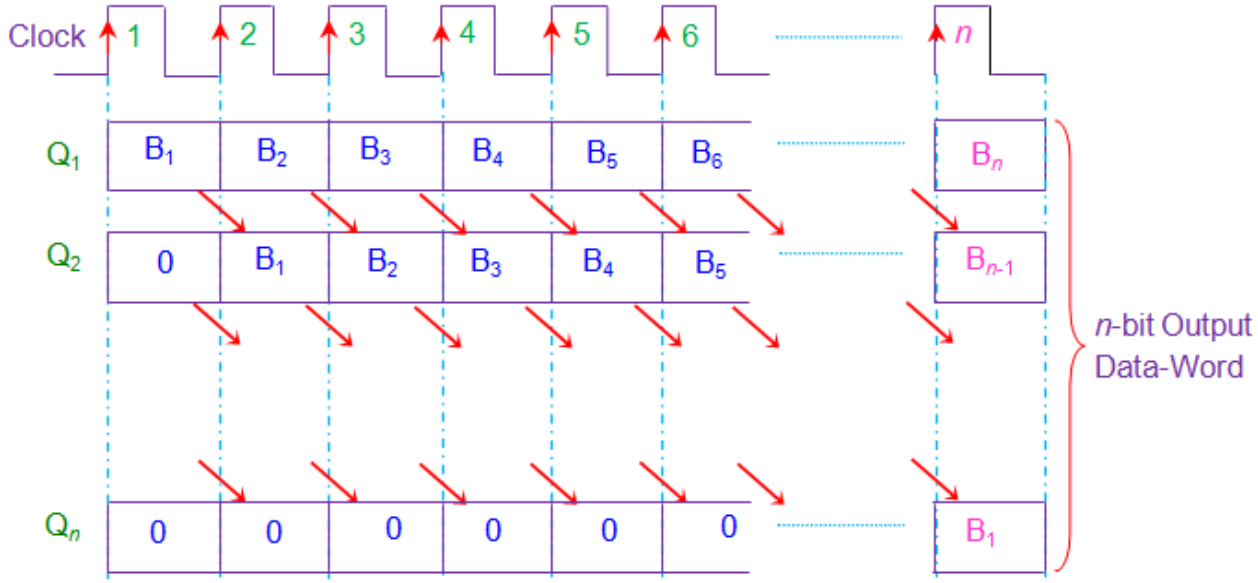


Figure 2 Output Waveform of *n*-bit Right-Shift SIPO Shift Register

In the right-shift SIPO shift-register, data bits shift from left to right for each clock tick. However if the data bits are made to shift from right to left in the same design, one gets a left-shift SIPO shift-register as shown by Figure 3. Nevertheless the basic working principle remains the same except the fact that now B_n down to B_1 is stored in Q_n down to Q_1 i.e. Q_1 =

B_1, Q_2 = B_2 … Q_n = B_n at the n^{th} clock tick.

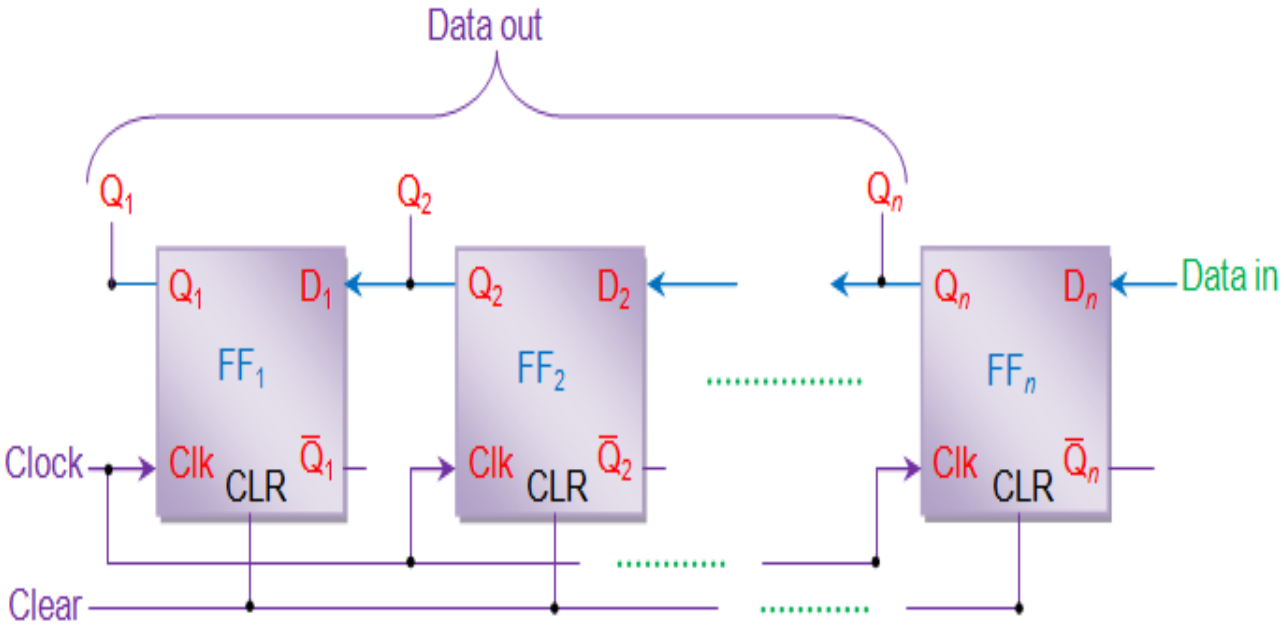


Figure 3 *n*-bit Serial-In Parallel-Out Left-Shift Shift Register

Parallel in Serial Out (PISO) Shift Register

In **Parallel In Serial Out (PISO) shift registers**, the data is loaded onto the register in parallel format while it is retrieved from it serially. Figure 1 shows a **PISO shift register** which has a control-line $(SH/\overline{LD})$and combinational circuit (AND and OR gates) in addition to the basic register components (flip-flops) fed with clock and clear pins.

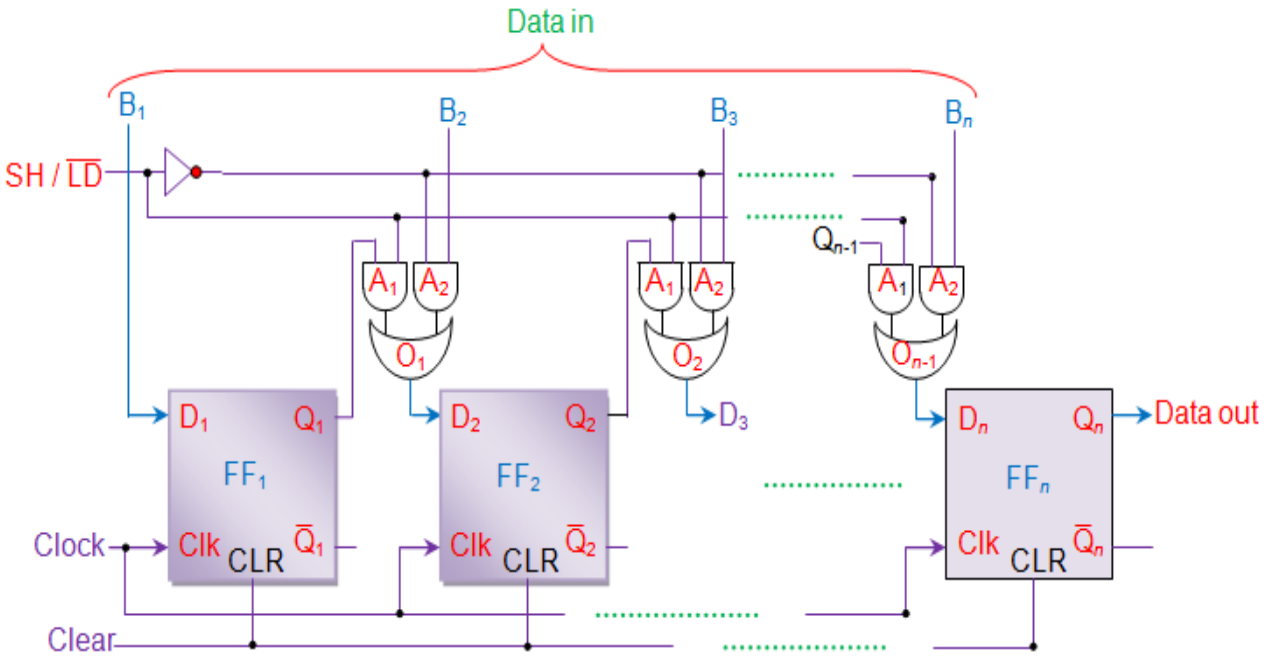


Figure 1 *n*-bit Parallel-In Serial-Out Right-Shift Shift Register

Here $SH/\overline{LD}$ control line is used to select the functionality of the shift register amongst shift or load at a given instant of time. This is because when the $SH/\overline{LD}$ line is made low, A_2 AND gates of all the combinational circuits become active while A_1 gates become inactive. Thus the bits of the input data word (Data in) appearing as inputs to the gates A_2 are passed on as the outputs of OR gates at each individual combinational circuit. This causes the individual bits of the Data in to be loaded/stored into respective flip-flops at the appearance of first leading edge of the clock (except the bit B_1 which gets directly stored into FF_1 at the first clock tick). This indicates that all the bits of the input data word are stored into the register components at the same clock tick.

Next, $SH/\overline{LD}$ line is driven high to activate the gates A_1 of the combinational circuits which inturn disables the gates A_2. This causes output bit of each flip-flop to appear at the output of the OR gate driving the very-next flip-flop (except the last flip-flop FF_n) i.e. output bit of FF_1 (Q_1) appears as the output of OR gate 1 (O_1) connected to D_2; Q_2 = output of O_2 = D_3 and so on. At this stage, if the rising edge of the clock pulse appears, then Q_1 appears at Q_2, Q_2 appears at Q_3, … and Q_{n-1} appears at Q_n.

This is nothing but right-shift of the data stored within the register by one-bit. Similarly it is seen that for each of the further clock pulses applied, one bit exits the PISO shift register through the output pin of n^{th} flip-flop (Data out = Q_n of FF_n), which is nothing but the serial

output. Thus one requires n clock cycles to obtain the entire n-bit input data word as a serial output of PISO shift register.

The truth table of the **PISO shift register** emphasizing the loading and retrieval processes is shown by Table I, while the corresponding wave forms are shown by Figure 2.

Table I Data Movement in Right-Shift PISO Shift

Clock Cycle	SH / $\overline{LD}$	Q_1	Q_2	Q_3	...	Q_{n-1}	Q_n = Data out
1	0	B_1	B_2	B_3		B_{n-1}	B_n
2	1	0	B_1	B_2	...	B_{n-2}	B_{n-1}
3	1	0	0	B_1	...	B_{n-3}	B_{n-2}
.		.	.	.		.	.
.		.	.	.	...	.	.
.		.	.	.		.	.
n-1	1	0	0	0	...	B_1	B_2
n	1	0	0	0		0	B_1

Parallel data Loading Operation

Serial data retrieval

By slightly modifying the design of Figure 1, one can make the data bits within the register to shift from right to left, thus obtaining a left-shift PISO shift-register (Figure 3). However the

basic working principle remains unaltered.

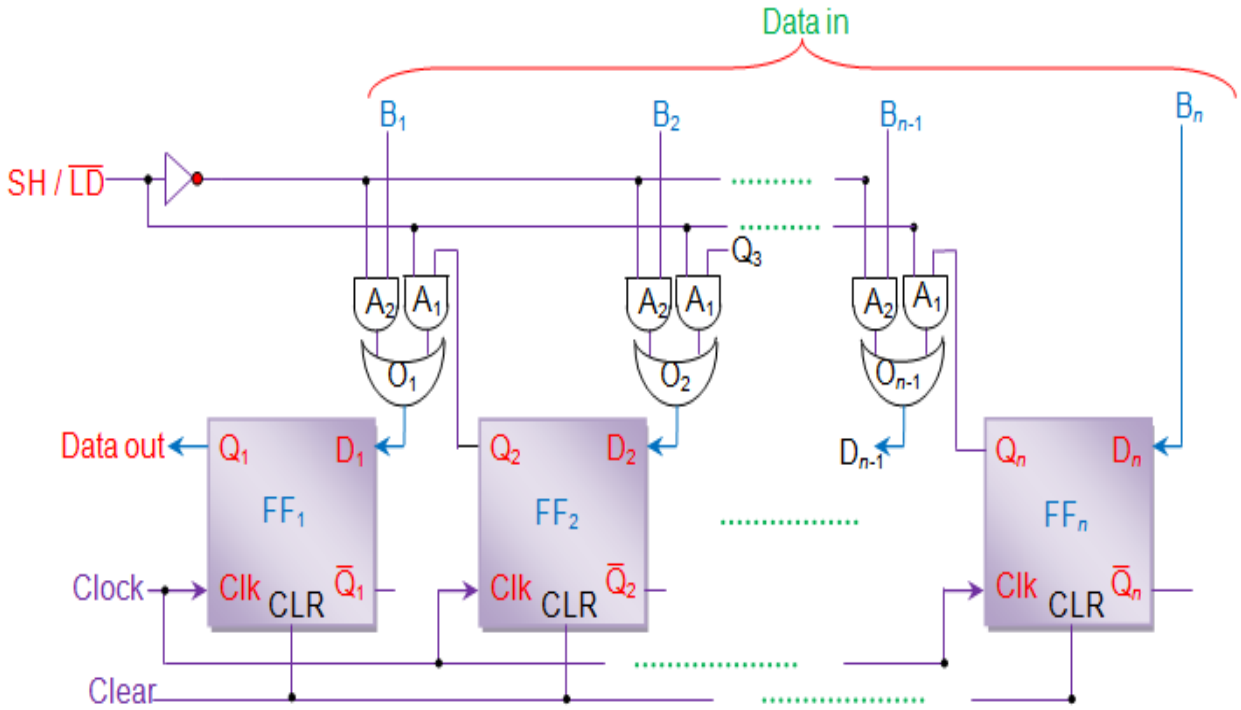


Figure 3 *n*-bit Parallel-In Serial-Out Left-Shift Shift Register

Parallel in Parallel Out (PIPO) Shift Register

Parallel In Parallel Out (PIPO) shift registers are the type of storage devices in which both data loading as well as data retrieval processes occur in parallel mode. Figure 1 shows a PIPO register capable of storing n-bit input data word (Data in). Here each flip-flop stores an individual bit of the data in appearing as its input (FF_1 stores B_1 appearing at D_1; FF_2 stores B_2 appearing at D_2 … FF_n stores B_n appearing at D_n) at the instant of first clock pulse. Further, at the same instant, the bit stored in each individual flip-flop also appears at their respective output pins ($Q_1 = D_1$; $Q_2 = D_2$ … $Q_n = B_n$). This indicates that both data storage as well as data recovery occur at a single (and at the same) clock pulse in PIPO registers.

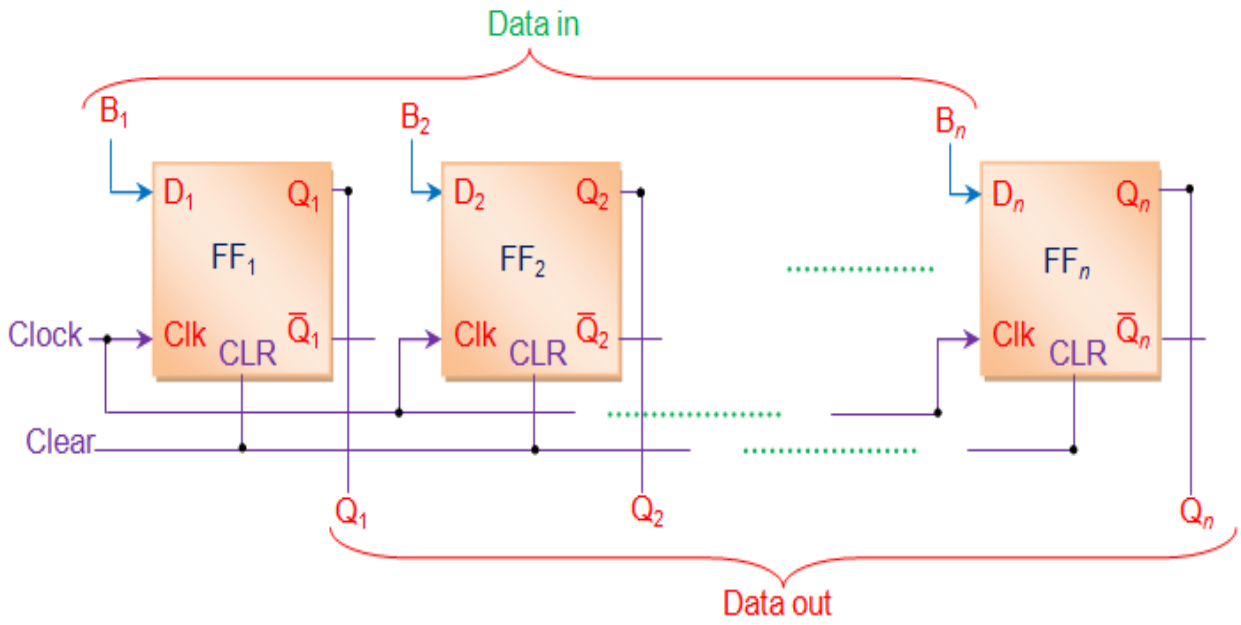


Figure 1 *n*-bit Parallel-In Parallel-Out Register

However one has to note that the PIPO register shown in Figure 1 is not capable of shifting the data bits. In order to convert PIPO register of Figure 1 into PIPO shift register, one has to modify its circuit by adding combinational circuit and control line $SH/\overline{LD}$ as shown by Figure 2.

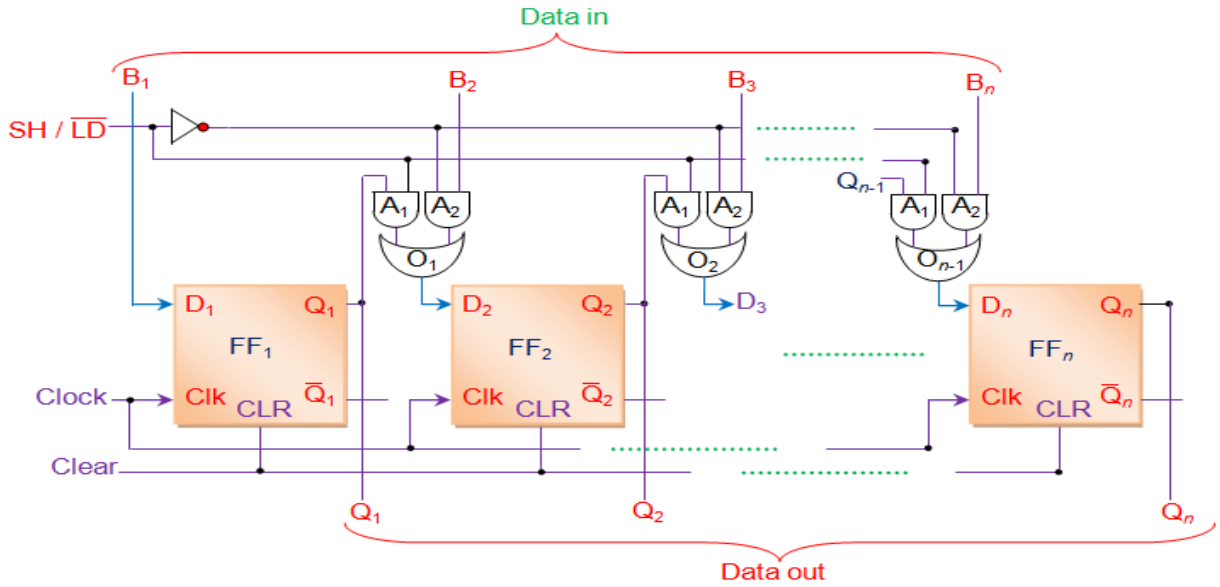


Figure 2 *n*-bit Parallel-In Parallel-Out Right-Shift Shift Register

Here if $SH/\overline{LD}$ line goes low, A_2 AND gates of all the combinational circuits become active while A_1 gates become inactive.

Thus the bits of the input data word (Data in) appearing as inputs to the gates A_2 are passed on as the OR gate outputs which are further loaded/stored into respective flip-flops at the appearance of first leading edge of the clock (except the bit B_1 which gets directly stored into FF_1 at the first clock tick). This indicates that all the bits of the input data word are stored into the register components at the same clock tick. At the same time, these bits also appear at the output pins of the respective flip-flops thus yielding parallel-output data word at the same clock tick.

Further when $SH/\overline{LD}$ line is made high, A_1 gates of all the combinational circuits enable while A_2 gates get disabled. This causes the output bit of each flip-flop to appear at the output of the OR gate driving the very-next flip-flop (except the last flip-flop FF_n) i.e. output bit of FF_1 (Q_1) appears as the output of OR gate 1 (O_1) connected to D_2; Q_2 = output of O_2 = D_3 and so on. At this stage, if the rising edge of the clock pulse appears, then Q_1 appears at Q_2, Q_2 appears at Q_3, … and Q_{n-1} appears at Q_n. This is nothing but right-shift of the data stored within the register by one-bit. This working is further emphasized in the Table I and Figure 3.

Table I Data Movement in Right-Shift PIPO Shift Register

Clock Cycle	SH / $\overline{LD}$	Q_1	Q_2	Q_3	…	Q_{n-1}	Q_n
1	0	B_1	B_2	B_3		B_{n-1}	B_n
2	1	0	B_1	B_2	…	B_{n-2}	B_{n-1}
3	1	0	0	B_1	…	B_{n-3}	B_{n-2}
.		.	.	.		.	.
.		.	.	.	…	.	.
.		.	.	.		.	.

Parallel Data Loading

Parallel Data Retrieval

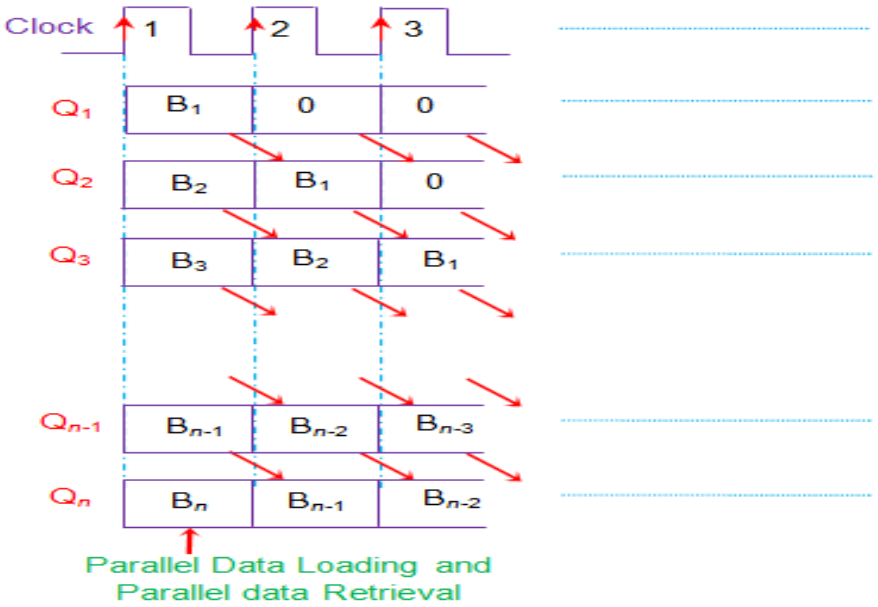


Figure 3 Output Waveform of *n*-bit Right-Shift PIPO Shift Register

Similar to the right-shift PIPO shift register, there can also be a left-shift PIPO shift register as shown by Figure 4. Nevertheless the mode of working remains the same.

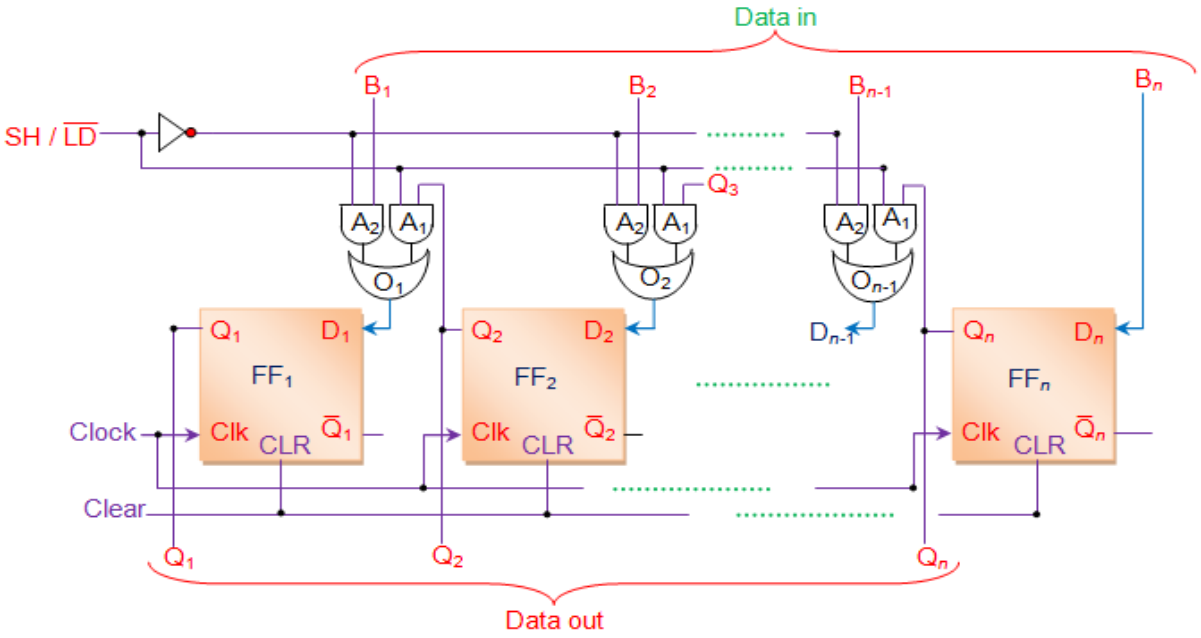


Figure 4 *n*-bit Parallel-In Parallel-Out Left-Shift Shift Register

Unit4

Q1. Explain Digital counters.

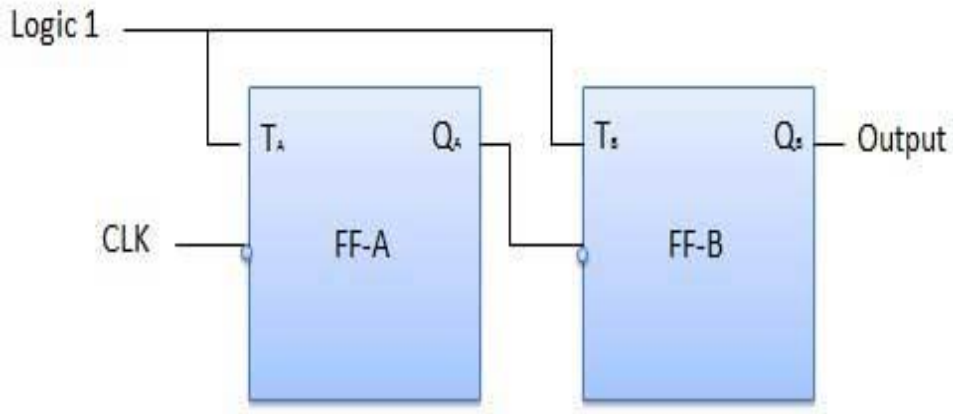
Counter is a sequential circuit. A digital circuit which is used for a counting pulses is known counter. Counter is the widest application of flip-flops. It is a group of flip-flops with a clock signal applied. Counters are of two types.

- Asynchronous or ripple counters.
- Synchronous counters.

Asynchronous or ripple counters

The logic diagram of a 2-bit ripple up counter is shown in figure. The toggle (T) flip-flop are being used. But we can use the JK flip-flop also with J and K connected permanently to logic 1. External clock is applied to the clock input of flip-flop A and Q_A output is applied to the clock input of the next flip-flop i.e. FF-B.

Logical Diagram



Operation

S.N.	Condition	Operation
1	**Initially let both the FFs be in the reset state**	Q_BQ_A = 00 initially

2	**After 1st negative clock edge**	As soon as the first negative clock edge is applied, FF-A will toggle and Q_A will be equal to 1. Q_A is connected to clock input of FF-B. Since Q_A has changed from 0 to 1, it is treated as the positive clock edge by FF-B. There is no change in Q_B because FF-B is a negative edge triggered FF. Q_BQ_A = 01 after the first clock pulse.
3	**After 2nd negative clock edge**	On the arrival of second negative clock edge, FF-A toggles again and Q_A = 0. The change in Q_A acts as a negative clock edge for FF-B. So it will also toggle, and Q_B will be 1. Q_BQ_A = 10 after the second clock pulse.
4	**After 3rd negative clock edge**	On the arrival of 3rd negative clock edge, FF-A toggles again and Q_A become 1 from 0. Since this is a positive going change, FF-B does not respond to it and remains inactive. So Q_B does not change and continues to be equal to 1. Q_BQ_A = 11 after the third clock pulse.
5	**After 4th negative clock edge**	On the arrival of 4th negative clock edge, FF-A toggles again and Q_A becomes 1 from 0. This negative change in Q_A acts as clock pulse for FF-B. Hence it toggles to change Q_B from 1 to 0. Q_BQ_A = 00 after the fourth clock pulse.

Truth Table

Clock	Counter output		State number	Deciimal Counter output
	Q_B	Q_A		
Initially	0	0	—	0
1st	0	1	1	1
2nd	1	0	2	2
3rd	1	1	3	3
4th	0	0	4	0

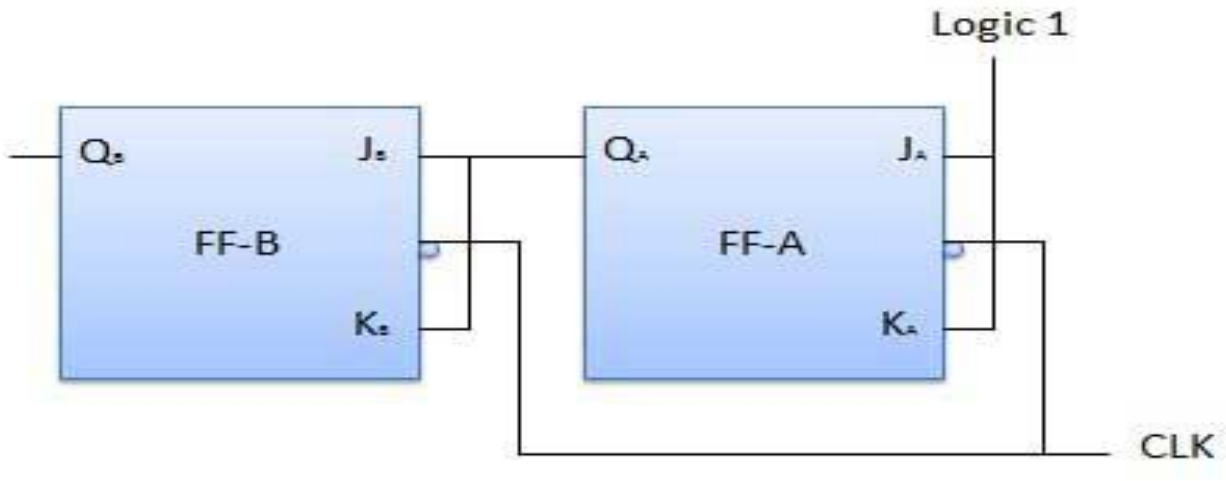
Synchronous counters

If the "clock" pulses are applied to all the flip-flops in a counter simultaneously, then such a counter is called as synchronous counter.

2-bit Synchronous up counter

The J_A and K_A inputs of FF-A are tied to logic 1. So FF-A will work as a toggle flip-flop. The J_B and K_B inputs are connected to Q_A.

Logical Diagram



Operation

S.N.	Condition	Operation
1	**Initially let both the FFs be in the reset state**	Q_BQ_A = 00 initially.
2	**After 1st negative clock edge**	As soon as the first negative clock edge is applied, FF-A will toggle and Q_A will change from 0 to 1. But at the instant of application of negative clock edge, Q_A , J_B = K_B = 0. Hence FF-B will not change its state. So Q_B will remain 0. Q_BQ_A = 01 after the first clock pulse.
3	**After 2nd negative clock edge**	On the arrival of second negative clock edge, FF-A toggles again and Q_A changes from 1 to 0. But at this instant Q_A was 1. So J_B = K_B= 1 and FF-B will toggle. Hence Q_B changes from 0 to 1. Q_BQ_A = 10 after the second clock pulse.
4	**After 3rd negative clock edge**	On application of the third falling clock edge, FF-A will toggle from 0 to 1 but there is no change of state for FF-B. Q_BQ_A = 11 after the third clock pulse.
5	**After 4th negative clock edge**	On application of the next clock pulse, Q_A will change from 1 to 0 as Q_B will also change from 1 to 0. Q_BQ_A = 00 after the fourth clock pulse.

Classification of counters

Depending on the way in which the counting progresses, the synchronous or asynchronous counters are classified as follows −

- Up counters
- Down counters
- Up/Down counters

UP/DOWN Counter

Up counter and down counter is combined together to obtain an UP/DOWN counter. A mode control (M) input is also provided to select either up or down mode. A combinational circuit is required to be designed and used between each pair of flip-flop in order to achieve the up/down operation.

- Type of up/down counters
- UP/DOWN ripple counters
- UP/DOWN synchronous counter

UP/DOWN Ripple Counters

In the UP/DOWN ripple counter all the FFs operate in the toggle mode. So either T flip-flops or JK flip-flops are to be used. The LSB flip-flop receives clock directly. But the clock to every other FF is obtained from (Q = Q bar) output of the previous FF.

- **UP counting mode (M=0)** − The Q output of the preceding FF is connected to the clock of the next stage if up counting is to be achieved. For this mode, the mode select input M is at logic 0 (M=0).
- **DOWN counting mode (M=1)** − If M = 1, then the Q bar output of the preceding FF is connected to the next FF. This will operate the counter in the counting mode.

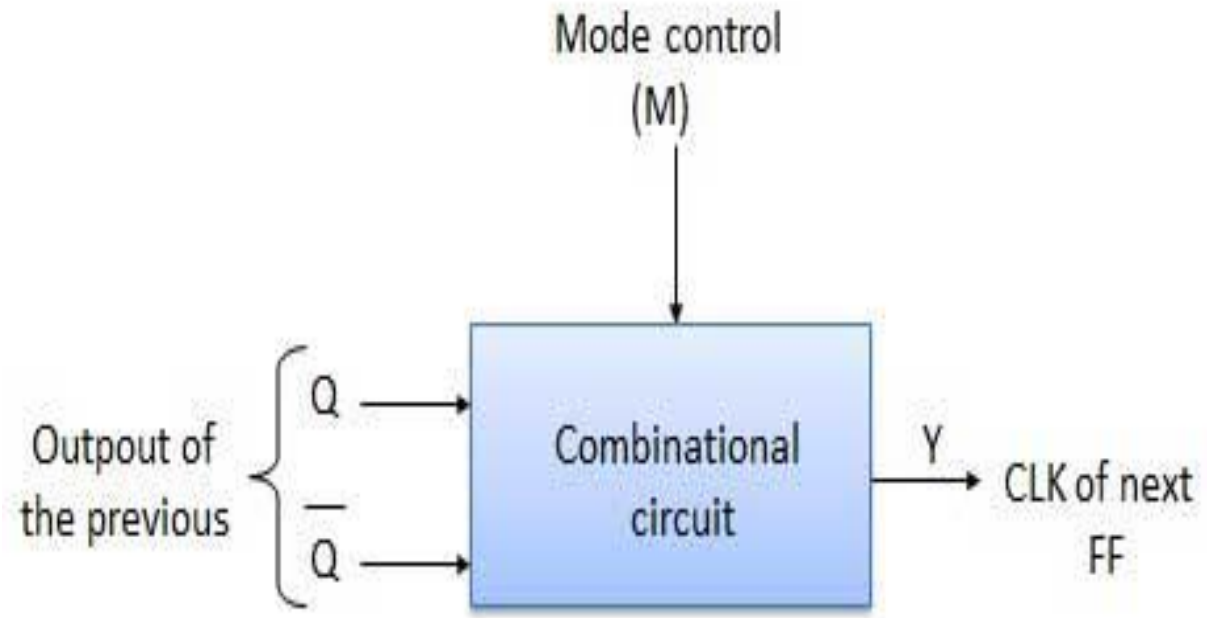
Example

3-bit binary up/down ripple counter.

- 3-bit − hence three FFs are required.
- UP/DOWN − So a mode control input is essential.
- For a ripple up counter, the Q output of preceding FF is connected to the clock input of the next one.

- For a ripple up counter, the Q output of preceding FF is connected to the clock input of the next one.
- For a ripple down counter, the Q bar output of preceding FF is connected to the clock input of the next one.
- Let the selection of Q and Q bar output of the preceding FF be controlled by the mode control input M such that, If M = 0, UP counting. So connect Q to CLK. If M = 1, DOWN counting. So connect Q bar to CLK.

Block Diagram



Truth Table

Inputs			Outputs	
M	Q	$\overline{Q}$	Y	
0	0	0	0	Y = Q for up counter
0	0	1	0	
0	1	0	1	
0	1	1	1	
1	0	0	0	$Y = \overline{Q}$ for up counter
1	0	1	1	
1	1	0	0	
1	1	1	1	

Operation

S.N.	Condition	Operation
1	**Case 1 – With M = 0 (Up counting mode)**	If M = 0 and M bar = 1, then the AND gates 1 and 3 in fig. will be enabled whereas the AND gates 2 and 4 will be disabled. Hence Q_A gets connected to the clock input of FF-B and Q_B gets connected to the clock input of FF-C. These connections are same as those for the normal up

		counter. Thus with M = 0 the circuit work as an up counter.
2	**Case 2: With M = 1 (Down counting mode)**	If M = 1, then AND gates 2 and 4 in fig. are enabled whereas the AND gates 1 and 3 are disabled. Hence Q_A bar gets connected to the clock input of FF-B and Q_B bar gets connected to the clock input of FF-C. These connections will produce a down counter. Thus with M = 1 the circuit works as a down counter.

Modulus Counter (MOD-N Counter)

The 2-bit ripple counter is called as MOD-4 counter and 3-bit ripple counter is called as MOD-8 counter. So in general, an n-bit ripple counter is called as modulo-N counter. Where, MOD number = 2^n.

Type of modulus

- 2-bit up or down (MOD-4)
- 3-bit up or down (MOD-8)
- 4-bit up or down (MOD-16)

Application of counters

- Frequency counters
- Digital clock
- Time measurement
- A to D converter
- Frequency divider circuits
- Digital triangular wave generator.

Counters in Digital Logic

According to Wikipedia, in digital logic and computing, a **C**ounter is a device which stores (and sometimes displays) the number of times a particular event or process has occurred,

often in relationship to a clock signal. Counters are used in digital electronics for counting purpose, they can count specific event happening in the circuit. For example, in UP counter a counter increases count for every rising edge of clock. Not only counting, a counter can follow the certain sequence based on our design like any random sequence 0,1,3,2… .They can also be designed with the help of flip flops.

Counter Classification

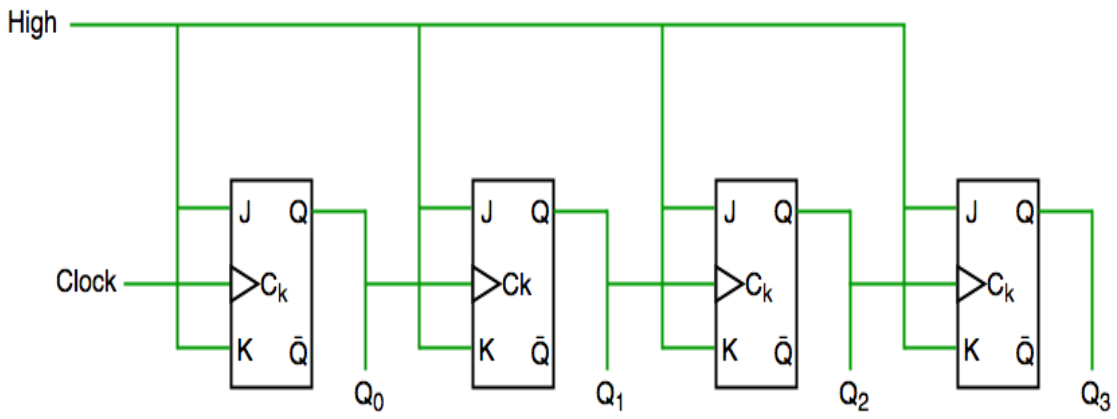
Counters are broadly divided into two categories

1. Asynchronous counter
2. Synchronous counter

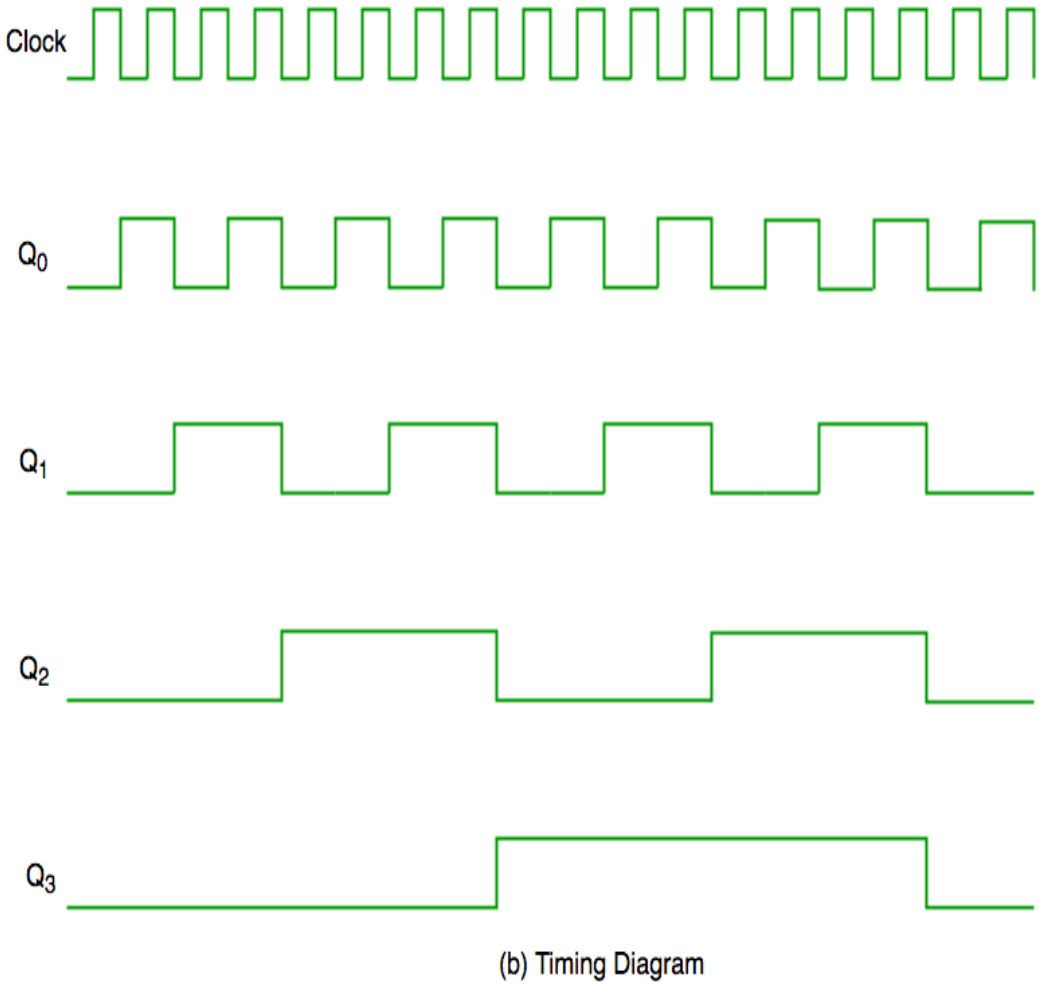
1. **Asynchronous Counter**

In asynchronous counter we don't use universal clock, only first flip flop is driven by main clock and the clock input of rest of the following counters is driven by output of previous flip

flops. We can understand it by following diagram-



(a) Asynchronous counter



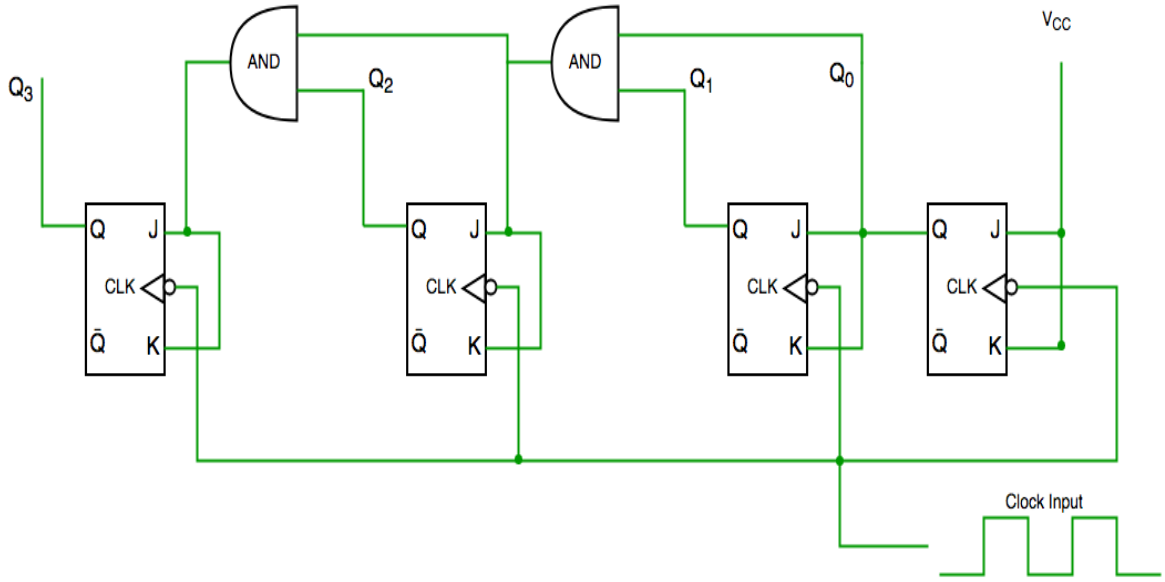
(b) Timing Diagram

It is evident from timing diagram that Q0 is changing as soon as the rising edge of clock pulse is encountered, Q1 is changing when rising edge of Q0 is encountered(because Q0 is

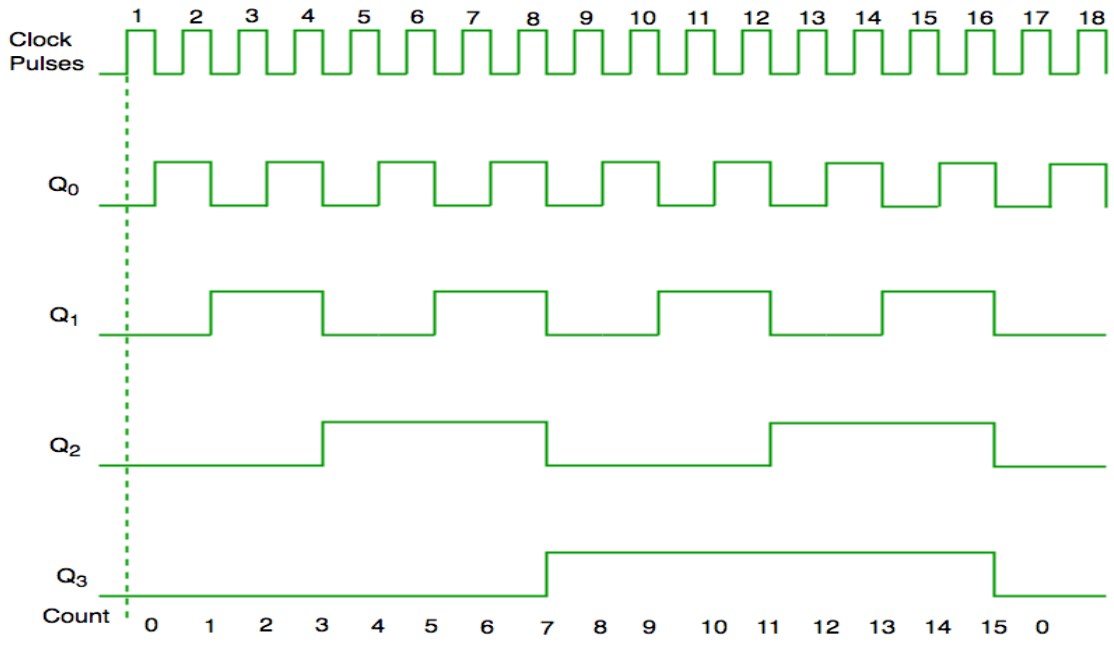
like clock pulse for second flip flop) and so on. In this way ripples are generated through Q0,Q1,Q2,Q3 hence it is also called **RIPPLE counter.**

2. **Synchronous Counter**

Unlike the asynchronous counter, synchronous counter has one global clock which drives each flip flop so output changes in parallel. The one advantage of synchronous counter over asynchronous counter is, it can operate on higher frequency than asynchronous counter as it does not have cumulative delay because of same clock is given to each flip flop.



Synchronous counter circuit



Timing diagram synchronous counter

From circuit diagram we see that Q0 bit gives response to each falling edge of clock while Q1 is dependent on Q0, Q2 is dependent on Q1 and Q0 , Q3 is dependent on Q2,Q1 and Q0.

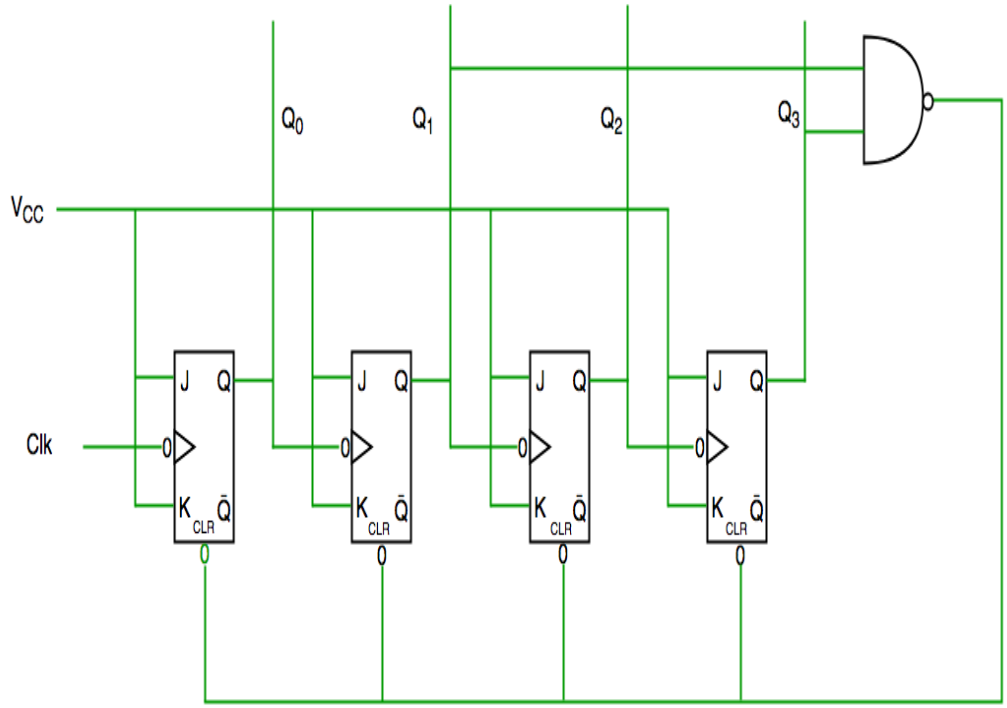
Decade Counter

A decade counter counts ten different states and then reset to its initial states. A simple decade counter will count from 0 to 9 but we can also make the decade counters which can go through any ten states between 0 to 15(for 4 bit counter).

Clock pulse	Q3	Q2	Q1	Q0
0	0	0	0	0
1	0	0	0	1
2	0	0	1	0
3	0	0	1	1
4	0	1	0	0
5	0	1	0	1
6	0	1	1	0
7	0	1	1	1
8	1	0	0	0

9	1	0	0	1
10	0	0	0	0

Truth table for simple decade counter



Decade counter circuit diagram

We see from circuit diagram that we have used nand gate for Q3 and Q1 and feeding this to clear input line because binary representation of 10 is—

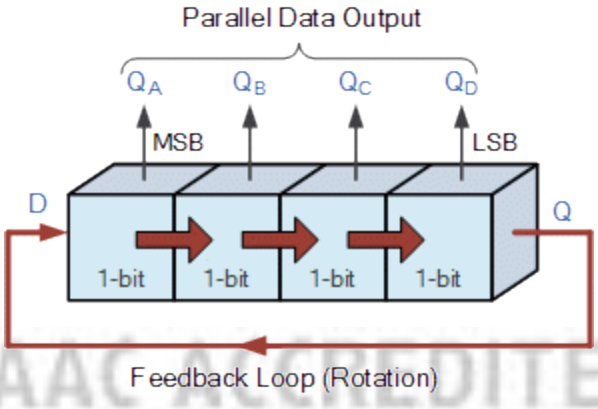
1010

And we see Q3 and Q1 are 1 here, if we give NAND of these two bits to clear input then counter will be clear at 10 and again start from beginning.

Q2. Explain Johnson Ring Counter OR TWISTED RING COUNTER

Johnson Ring Counter OR TWISTED RING COUNTER

The Johnson Ring Counter consists of a number of counters connected together with the output fed back to the input



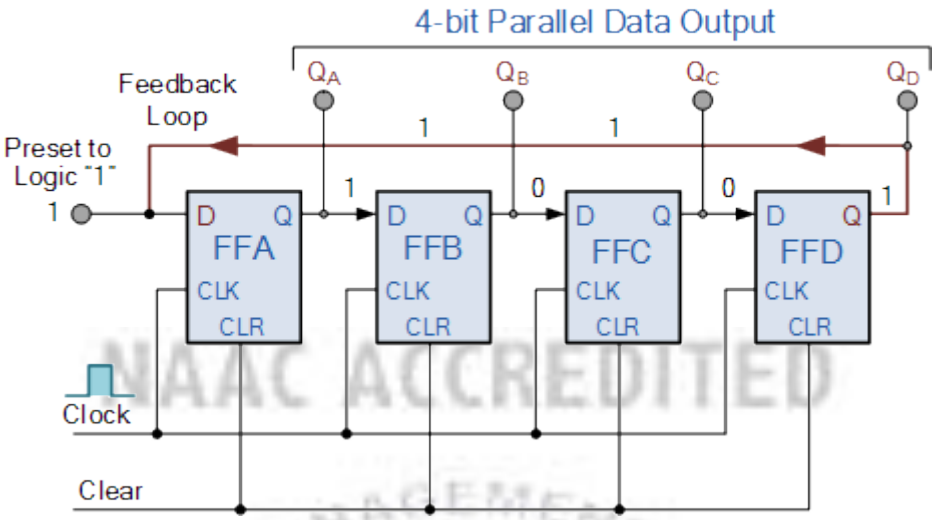
In the previous Shift Register tutorial we saw that if we apply a serial data signal to the input of a *Serial-in to Serial-out Shift Register*, the same sequence of data will exit from the last flip flip in the register chain.

This serial movement of data through the resister occurs after a preset number of clock cycles thereby allowing the SISO register to act as a sort of time delay circuit to the original input data signal.

But what if we were to connect the output of this shift register back to its input so that the output from the last flip-flop, Q_D becomes the input of the first flip-flop, Q_A. We would then have a closed loop circuit that "recirculates" the same bit of DATA around a continuous loop for every state of its sequence, and this is the principal operation of a **Ring Counter**.

Then by looping the output back to the input, (feedback) we can convert a standard shift register circuit into a ring counter. Consider the circuit below.

4-bit Ring Counter

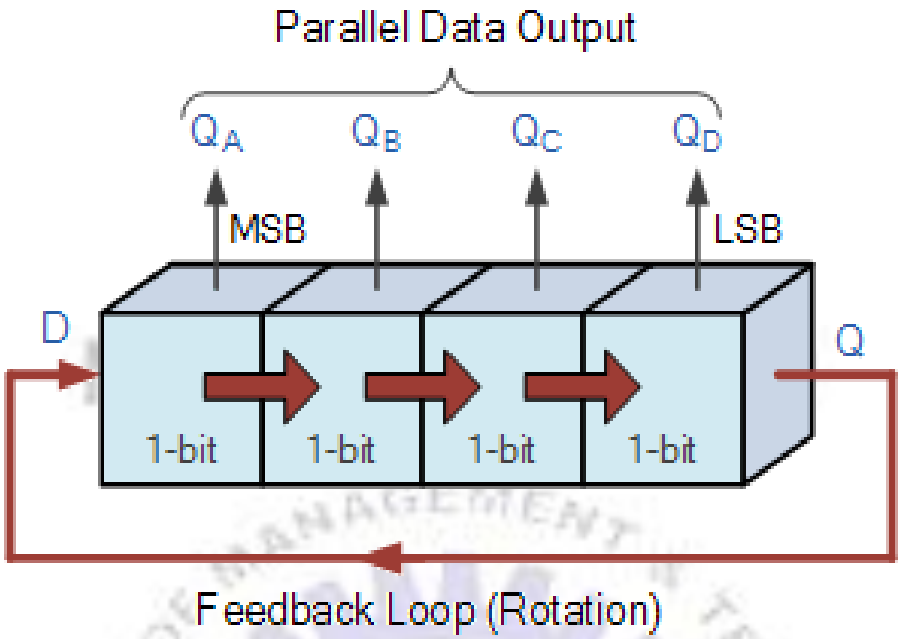


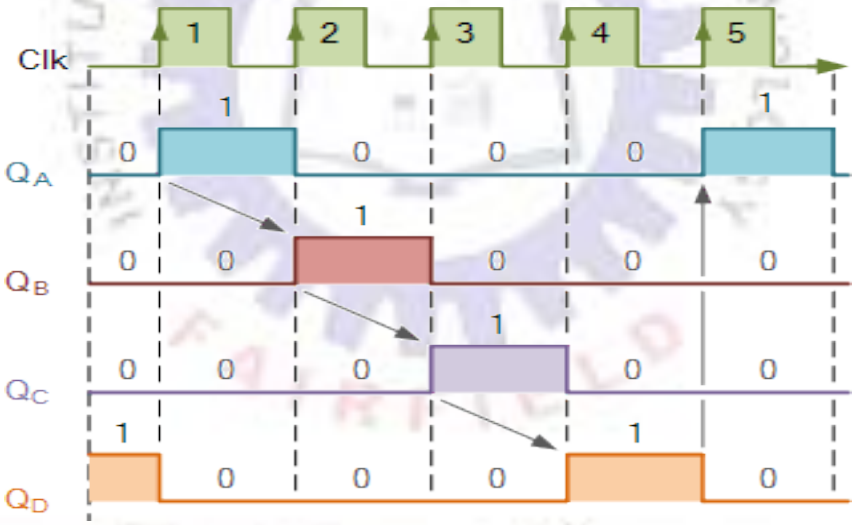
The synchronous **Ring Counter** example above, is preset so that exactly one data bit in the register is set to logic "1" with all the other bits reset to "0". To achieve this, a "CLEAR" signal is firstly applied to all the flip-flops together in order to "RESET" their outputs to a logic "0" level and then a "PRESET" pulse is applied to the input of the first flip-flop (FFA) before the clock pulses are applied. This then places a single logic "1" value into the circuit of the ring counter.

So on each successive clock pulse, the counter circulates the same data bit between the four flip-flops over and over again around the "ring" every fourth clock cycle. But in order to cycle the data correctly around the counter we must first "load" the counter with a suitable data pattern as all logic "0's" or all logic "1's" outputted at each clock cycle would make the ring counter invalid.

This type of data movement is called "rotation", and like the previous shift register, the effect of the movement of the data bit from left to right through a ring counter can be presented graphically as follows along with its timing diagram:

Rotational Movement of a Ring Counter





Since the ring counter example shown above has four distinct states, it is also known as a "modulo-4" or "mod-4" counter with each flip-flop output having a frequency value equal to one-fourth or a quarter (1/4) that of the main clock frequency.

The "MODULO" or "MODULUS" of a counter is the number of states the counter counts or sequences through before repeating itself and a ring counter can be made to output any modulo number. A "mod-n" ring counter will require "n" number of flip-flops connected together to circulate a single data bit providing "n" different output states.

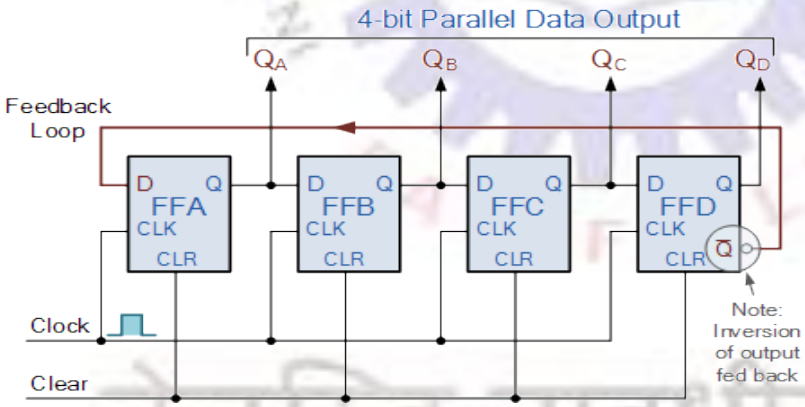
For example, a mod-8 ring counter requires eight flip-flops and a mod-16 ring counter would require sixteen flip-flops. However, as in our example above, only four of the possible sixteen states are used, making ring counters very inefficient in terms of their output state usage.

Johnson Ring Counter

The **Johnson Ring Counter** or "Twisted Ring Counters", is another shift register with feedback exactly the same as the standard *Ring Counter* above, except that this time the inverted output Q of the last flip-flop is now connected back to the input D of the first flip-flop as shown below.

The main advantage of this type of ring counter is that it only needs half the number of flip-flops compared to the standard ring counter then its modulo number is halved. So a "n-stage" Johnson counter will circulate a single data bit giving sequence of 2n different states and can therefore be considered as a "mod-2n counter".

4-bit Johnson Ring Counter



This inversion of Q before it is fed back to input D causes the counter to "count" in a different way. Instead of counting through a fixed set of patterns like the normal ring counter such as for a 4-bit counter, "0001"(1), "0010"(2), "0100"(4), "1000"(8) and repeat, the Johnson counter counts up and then down as the initial logic "1" passes through it to the right replacing the preceding logic "0".

A 4-bit Johnson ring counter passes blocks of four logic "0" and then four logic "1" thereby producing an 8-bit pattern. As the inverted output Q is connected to the input D this 8-bit

pattern continually repeats. For example, “1000”, “1100”, “1110”, “1111”, “0111”, “0011”, “0001”, “0000” and this is demonstrated in the following table below.

Truth Table for a 4-bit Johnson Ring Counter

Clock Pulse No	FFA	FFB	FFC	FFD
0	0	0	0	0
1	1	0	0	0
2	1	1	0	0
3	1	1	1	0
4	1	1	1	1
5	0	1	1	1
6	0	0	1	1
7	0	0	0	1

As well as counting or rotating data around a continuous loop, ring counters can also be used to detect or recognise various patterns or number values within a set of data. By connecting simple logic gates such as the *AND* or the *OR* gates to the outputs of the flip-flops the circuit can be made to detect a set number or value.

Standard 2, 3 or 4-stage **Johnson Ring Counters** can also be used to divide the frequency of the clock signal by varying their feedback connections and divide-by-3 or divide-by-5 outputs are also available.

For example, a 3-stage Johnson Ring Counter could be used as a 3-phase, 120 degree phase shift square wave generator by connecting to the data outputs at A, B and NOT-B.

The standard 5-stage Johnson counter such as the commonly available CD4017 is generally used as a synchronous decade counter/divider circuit.

Q3. Explain Memory Devices

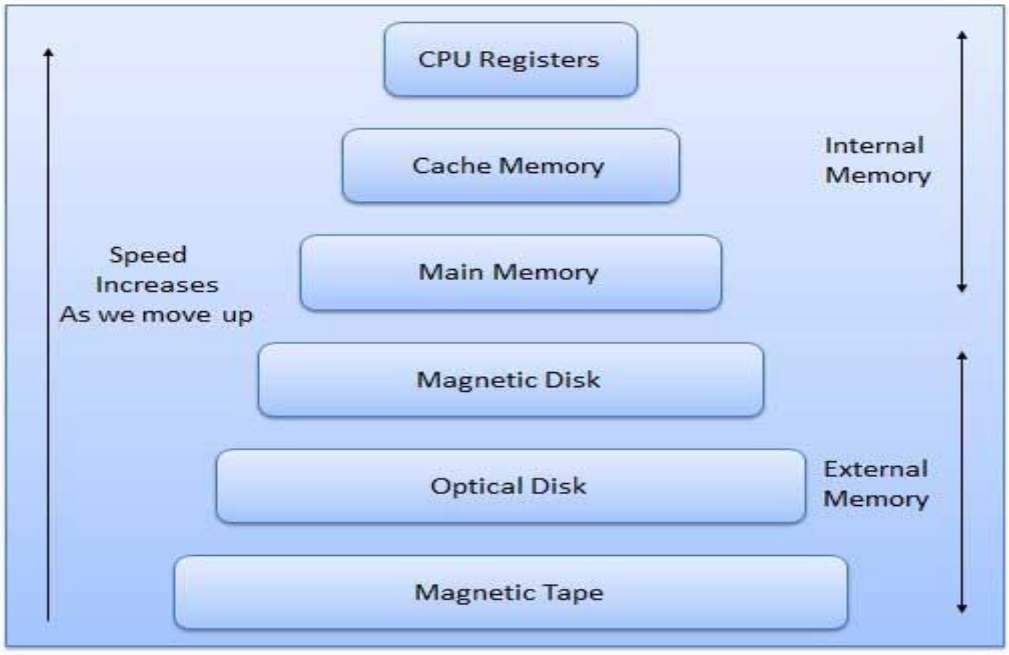
A memory is just like a human brain. It is used to store data and instruction. Computer memory is the storage space in computer where data is to be processed and instructions required for processing are stored.

The memory is divided into large number of small parts. Each part is called a cell. Each location or cell has a unique address which varies from zero to memory size minus one.

For example if computer has 64k words, then this memory unit has 64 * 1024 = 65536 memory location. The address of these locations varies from 0 to 65535.

Memory is primarily of two types

- **Internal Memory** − cache memory and primary/main memory
- **External Memory** − magnetic disk / optical disk etc.



Characteristics of Memory Hierarchy are following when we go from top to bottom.

- Capacity in terms of storage increases.
- Cost per bit of storage decreases.
- Frequency of access of the memory by the CPU decreases.
- Access time by the CPU increases.

RAM

A RAM constitutes the internal memory of the CPU for storing data, program and program result. It is read/write memory. It is called random access memory (RAM).

Since access time in RAM is independent of the address to the word that is, each storage location inside the memory is as easy to reach as other location & takes the same amount of time. We can reach into the memory at random & extremely fast but can also be quite expensive.

RAM is volatile, i.e. data stored in it is lost when we switch off the computer or if there is a power failure. Hence, a backup uninterruptible power system (UPS) is often used with computers. RAM is small, both in terms of its physical size and in the amount of data it can hold.

RAM is of two types

- Static RAM (SRAM)
- Dynamic RAM (DRAM)

Static RAM (SRAM)

The word **static** indicates that the memory retains its contents as long as power remains applied. However, data is lost when the power gets down due to volatile nature. SRAM chips use a matrix of 6-transistors and no capacitors. Transistors do not require power to prevent leakage, so SRAM need not have to be refreshed on a regular basis.

Because of the extra space in the matrix, SRAM uses more chips than DRAM for the same amount of storage space, thus making the manufacturing costs higher.

Static RAM is used as cache memory needs to be very fast and small.

Dynamic RAM (DRAM)

DRAM, unlike SRAM, must be continually **refreshed** in order for it to maintain the data. This is done by placing the memory on a refresh circuit that rewrites the data several hundred times per second. DRAM is used for most system memory because it is cheap and small. All DRAMs are made up of memory cells. These cells are composed of one capacitor and one transistor.

ROM

ROM stands for Read Only Memory. The memory from which we can only read but cannot write on it. This type of memory is non-volatile. The information is stored permanently in such memories during manufacture.

A ROM, stores such instruction as are required to start computer when electricity is first turned on, this operation is referred to as bootstrap. ROM chip are not only used in the computer but also in other electronic items like washing machine and microwave oven.

Following are the various types of ROM −

MROM (Masked ROM)

The very first ROMs were hard-wired devices that contained a pre-programmed set of data or instructions. These kind of ROMs are known as masked ROMs. It is inexpensive ROM.

PROM (Programmable Read Only Memory)

PROM is read-only memory that can be modified only once by a user. The user buys a blank PROM and enters the desired contents using a PROM programmer. Inside the PROM chip there are small fuses which are burnt open during programming. It can be programmed only once and is not erasable.

EPROM (Erasable and Programmable Read Only Memory)

The EPROM can be erased by exposing it to ultra-violet light for a duration of upto 40 minutes. Usually, an EPROM eraser achieves this function. During programming an electrical charge is trapped in an insulated gate region. The charge is retained for more than ten years because the charge has no leakage path. For erasing this charge, ultra-violet light is passed through a quartz crystal window (lid). This exposure to ultra-violet light dissipates the charge. During normal use the quartz lid is sealed with a sticker.

EEPROM (Electrically Erasable and Programmable Read Only Memory)

The EEPROM is programmed and erased electrically. It can be erased and reprogrammed about ten thousand times. Both erasing and programming take about 4 to 10 ms (millisecond). In EEPROM, any location can be selectively erased and programmed.

EEPROMs can be erased one byte at a time, rather than erasing the entire chip. Hence, the process of re-programming is flexible but slow.

Serial Access Memory

Sequential access means the system must search the storage device from the beginning of the memory address until it finds the required piece of data. Memory device which supports such access is called a Sequential Access Memory or Serial Access Memory. Magnetic tape is an example of serial access memory.

Direct Access Memory

Direct access memory or Random Access Memory, refers to conditions in which a system can go directly to the information that the user wants. Memory device which supports such access is called a Direct Access Memory. Magnetic disks, optical disks are examples of direct access memory.

Cache Memory

Cache memory is a very high speed semiconductor memory which can speed up CPU. It acts as a buffer between the CPU and main memory. It is used to hold those parts of data and program which are most frequently used by CPU. The parts of data and programs, are transferred from disk to cache memory by operating system, from where CPU can access them.

Advantages

- Cache memory is faster than main memory.
- It consumes less access time as compared to main memory.
- It stores the program that can be executed within a short period of time.
- It stores data for temporary use.

Disadvantages

- Cache memory has limited capacity.
- It is very expensive.

Virtual memory is a technique that allows the execution of processes which are not completely available in memory. The main visible advantage of this scheme is that programs

can be larger than physical memory. Virtual memory is the separation of user logical memory from physical memory.

This separation allows an extremely large virtual memory to be provided for programmers when only a smaller physical memory is available. Following are the situations, when entire program is not required to be loaded fully in main memory.

- User written error handling routines are used only when an error occurred in the data or computation.
- Certain options and features of a program may be used rarely.
- Many tables are assigned a fixed amount of address space even though only a small amount of the table is actually used.
- The ability to execute a program that is only partially in memory would counter many benefits.
- Less number of I/O would be needed to load or swap each user program into memory.
- A program would no longer be constrained by the amount of physical memory that is available.
- Each user program could take less physical memory, more programs could be run the same time, with a corresponding increase in CPU utilization and throughput.

Auxiliary Memory

Auxiliary memory is much larger in size than main memory but is slower. It normally stores system programs, instruction and data files. It is also known as secondary memory. It can also be used as an overflow/virtual memory in case the main memory capacity has been exceeded. Secondary memories cannot be accessed directly by a processor. First the data/information of auxiliary memory is transferred to the main memory and then that information can be accessed by the CPU. Characteristics of Auxiliary Memory are following –

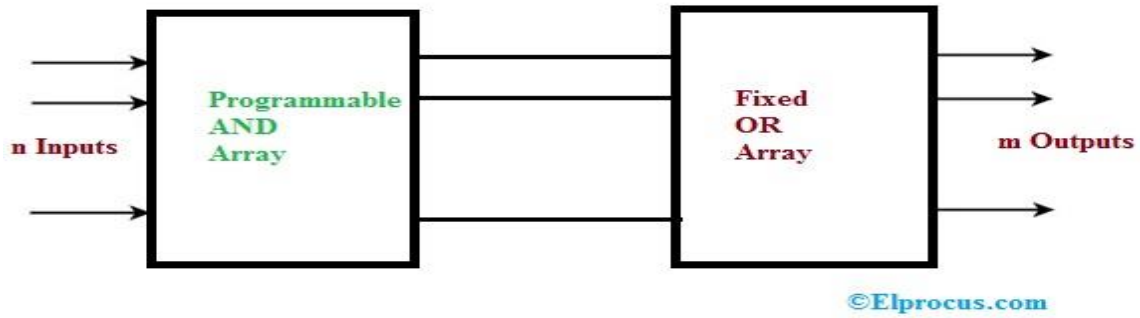
- **Non-volatile memory** − Data is not lost when power is cut off.
- **Reusable** − The data stays in the secondary storage on permanent basis until it is not overwritten or deleted by the user.

- **Reliable** − Data in secondary storage is safe because of high physical stability of secondary storage device.
- **Convenience** − With the help of a computer software, authorised people can locate and access the data quickly.
- **Capacity** − Secondary storage can store large volumes of data in sets of multiple disks.
- **Cost** − It is much lesser expensive to store data on a tape or disk than primary memory.

Q4. Explain Programmable Array Logic

Programmable Array Logic

Both **Programmable Array Logic** and **Programmable Logic Array** are types of PLDs (programmable logic devices), and these are mainly used for designing combination logic mutually by sequential logic. The main difference among these two is that PAL can be designed with a collection of AND gates and fixed collection of OR gates whereas PLA can be designed with a programmable array of AND although a fixed collection of OR gate. A programmable logic device offers a simple as well as flexible logic circuit designing.



Programmable Array Logic

Design of Programmable Array Logic (PAL)

The **definition of term PAL or Programmable Array Logic** is one type of PLD which is known as Programmable Logic Device circuit, and working of this PAL is the same as the PLA. The designing of the programmable array logic can be done with fixed OR gates as well as programmable AND gates. By using this we can implement two easy functions wherever the associates AND gates with each OR gate denote the highest number of product conditions that can be produced in the form of **SOP (sum of product)** of an exact function.

As the logic gates like AND is connected continually toward the OR gates, and that indicates that the produced product term is not distributed with the output functions. The major notion behind PLD development is to fabricate a compound Boolean logic onto a single chip by removing the defective wiring, avoiding the logic design, as well as decreasing the consumption of power.

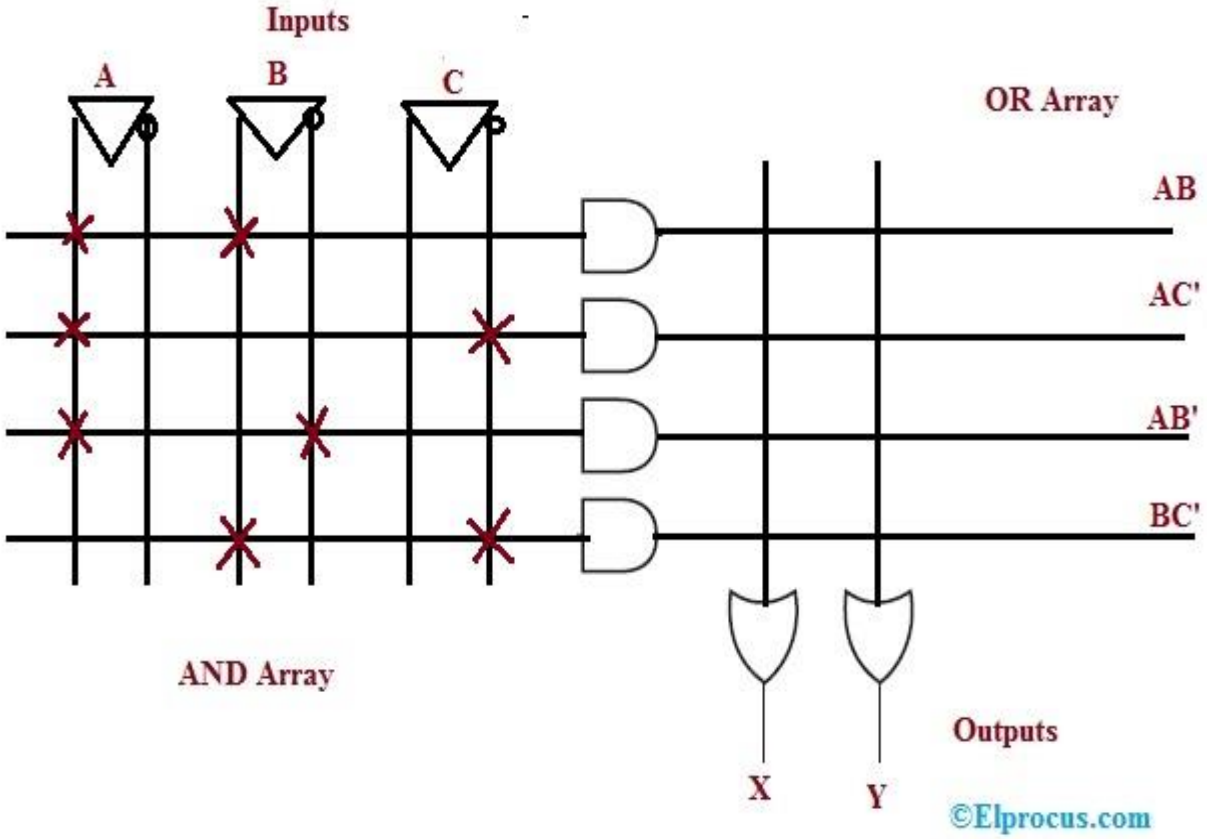
Example of PAL

Implement the following **Boolean expression** with the help of **programmable array logic (PAL)**

X =AB + AC'

Y= AB' + BC'

The above given two Boolean functions are in the form of **SOP (sum of products)**. The product terms present in the Boolean expressions are X & Y, and one product term that is AC' is common in every equation. So, the total required logic gates for generating the above two equations is AND gates-4 OR programmable gates-2. The equivalent PAL logic diagram is shown below.



PAL Logic Circuit

The AND gates which are programmable have the right of entry for normal as well as complemented variable inputs. In the above logic diagram, the available inputs for each AND gate are A, A', B, B', C, C'. So, in order to generate a single product term with every AND gate, the program is required.

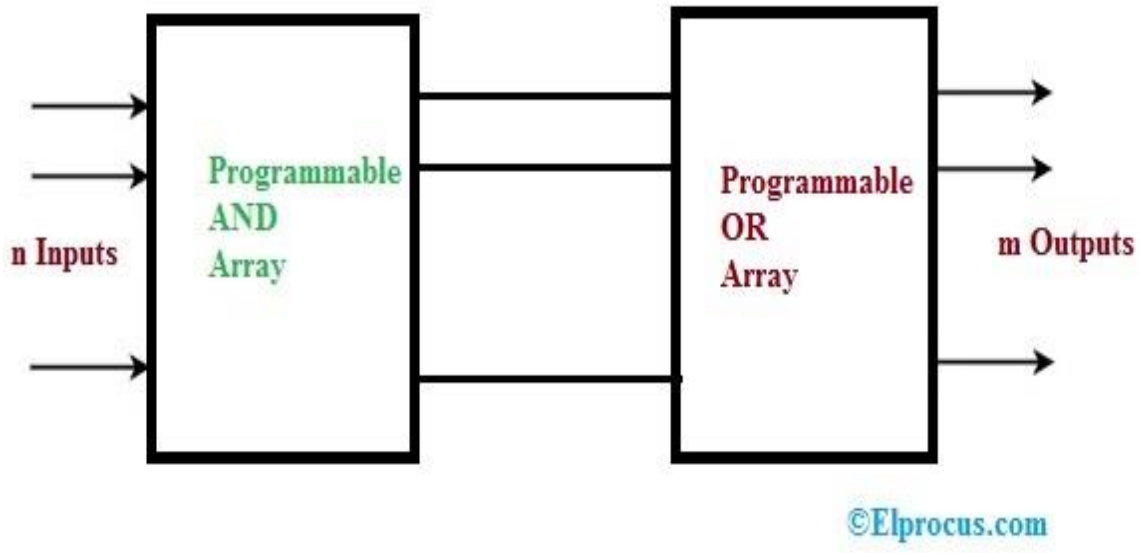
All the product terms are obtainable at the inputs of an each OR gate. Here, the programmable connections on the logic gate can be denoted with the symbol 'X'.

Here, the OR gate inputs are fixed. Thus, the required product terms are associated with each OR gate inputs. As a result, these gates will generate particular Boolean equations. The **'.'** The symbol represents permanent connections.

Q5. Explain Programmable Logic Array (PLA)

Both **Programmable Array Logic** and **Programmable Logic Array** are types of PLDs (programmable logic devices), and these are mainly used for designing combination logic mutually by sequential logic. The main difference among these two is that PAL can be designed with a collection of AND gates and fixed collection of OR gates whereas PLA can be designed with a programmable array of AND although a fixed collection of OR gate. A programmable logic device offers a simple as well as flexible logic circuit designing.

Previous to programmable logic devices, the **combinational logic circuits** can be designed with multiplexers, and these circuits were rigid as well as compound, then PLDs are developed. The initial programmable logic device was ROM, but it was not successful due to the hardware wastage issues as well as exponential growth enhancement in the every hardware application. To overcome this issue, PAL and PLA were used. These two are programmable, and efficiently uses the hardware.



Programmable Logic Array

Design of Programmable Logic Array (PLA)

The definition of term PLA presents the Boolean function in the form of a sum of product (SOP). The designing of this programmable logic array can be done using the logic gates like AND, OR, and NOT by fabricating on the chip, that makes every input as well as its compliment obtainable toward every AND gate.

An every AND gate's output is connected to the every OR gate. Finally, the output of the OR gate generates the output of the chip. Thus, this is how an appropriate association is finished to use the expressions of the sum of the product. In the programmable logic array, the connections of logic gates like AND & OR are programmable. PLA is expensive and difficult to compare with PAL. The PAL uses two dissimilar developed methods can be used for a programmable logic array for enhancing the effortlessness of programming. In this kind of method, every connection can be done using a fuse on each intersection point wherever the unnecessary connections can be detached by the fuse blowing. The final technique engages the making of connection while the process of the fabrication using the suitable cover offered for the precise interconnection model.

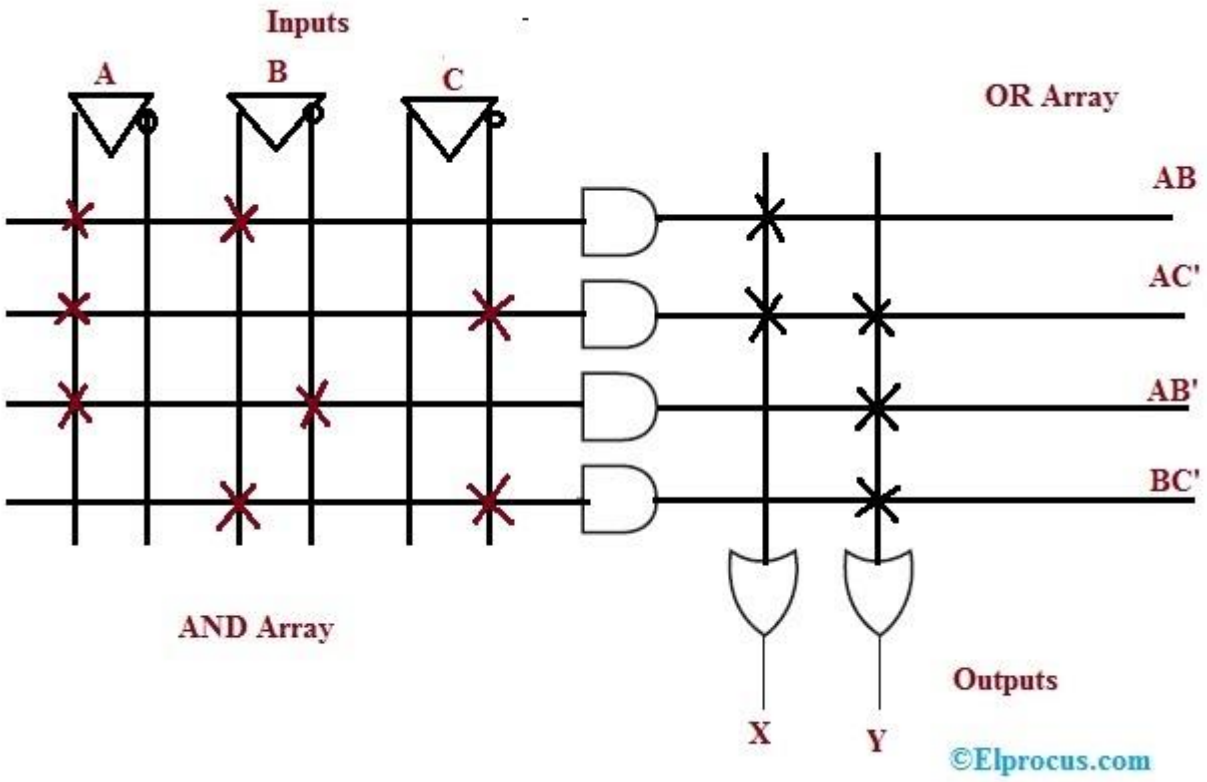
Example of PLA

Implement the following Boolean expression with the help of programmable logic array (PLA)

X = AB + AC'

Y = AB' +BC + AC'

The above given two Boolean functions are in the form of SOP (sum of products). The product terms present in the Boolean expressions are X & Y, and one product term that is AC' is common in every equation. So, the total required logic gates for generating the above two equations is AND gates-4, OR programmable OR gates-2. The equivalent PLA logic diagram is shown below.



PLA Logic Circuit

The AND gates which are programmable have the right of entry for normal as well as complemented variable inputs. In the above logic diagram, the available inputs for each AND gate are A, A', B, B', C, C'. So, in order to generate a single product term with every AND gate, the program is required.

All the product terms are obtainable at the inputs of each OR gate. Here, the programmable connections on the logic gate can be denoted with the symbol 'X'

DATA STRUCTURE

BCA 108

Unit-1

Question1. Write the routine for insertion operation of singly linked list.

Answer: Void insert (Element Type X, List L, Position P)

```
{Position T mpCell;
T mpCell=malloc(sizeof(struct Node));
If(TmpCell=NULL)
FatalError("Out Of space!!!");
 TmpCell->Element=X;TmpCell->Next=P->Next;
P->Next=TmpCell;
}
};
```

Question2 How the queue is implemented by linked list?

• It is based on the dynamic memory management techniques which allow allocation and

De-allocation of memory space at runtime.

Insert operation

It involves the following subtasks:

1. Reserving memory space of the size of a queue element in memory
2. Storing the added value at the new location
3. Linking the new element with existing queue
4. Updating the *rear* pointer

Delete operation

It involves the following subtasks:

1. Checking whether queue is empty

2. Retrieving the front most element of the queue

3. Updating the front pointer

Returning the retrieved value

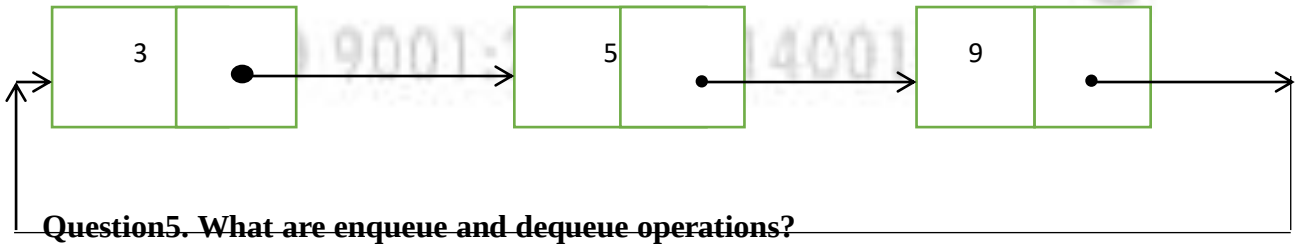
Question3. Differentiate linear and non- linear data structure.

Answer:

Linear data structure	**Non- linear data structure**
Data are arranged in linear or sequential manner	Data are not arranged in linear manner.
Every item is related to its previous and next item.	Every item is attached with many other items.
Data items can be trans versed in a single run.	Data items cannot be trans versed in a single run.
Implementation is easy.	Implementation is difficult.
Example: array, stack, queue, linked list.	Example: tree, graph.

Question4. When singly linked list can be represented as circular linked list?

Answer: In a singly linked list, all the nodes are connected with forward links to the next nodes in the list. The last node has a next field, NULL. In order to implement the circularly linked list from singly linked lists, the last node's next field to the first node.



Question5. What are enqueue and dequeue operations?

• **Answer: Enqueue** - adding an element to the queue at the rear end

If the queue is not full, this function adds an element to the back of the queue, else it prints "**Overflow**".

```
void enqueue(int queue[], int element, int& rear, int arraySize) { if(rear == arraySize)      //
   Queue is full
  printf("Overflow\n"); else{

       queue[rear] = element;    // Add the element to the back rear++;
   }
}
```


- **Dequeue** – removing or deleting an element from the queue at the front end

If the queue is not empty, this function removes the element from the front of the queue, else it prints "**Underflow**". void dequeue(int queue[], int& front, int rear) {

```
   if(front == rear)          // Queue is empty printf("Underflow\n");

   else {

      queue[front] = 0;       // Delete the front element front++;
   }
}
```


Unit-2

Question1- What are the two methods of binary tree implementation?

Answer Two methods to implement a binary tree are

- Linear representation.
- Linked representation

Question2. Write the advantages of threaded binary tree.

Answer: The difference between a binary tree and the threaded binary tree is that in the binary trees the nodes are null if there is no child associated with it and so there is no way to traverse back.

But in a threaded binary tree we have threads associated with the nodes i.e. they either are linked to the predecessor or successor in the in order traversal of the nodes. This helps us to traverse further or backward in the in order traversal fashion.

There can be two types of threaded binary tree:-

1) Single Threaded: - i.e. nodes are threaded either towards its in order predecessor or successor.

2) Double threaded: - i.e. nodes are threaded towards both the in order predecessor and successor.

Question3. List out the steps involved in deleting a node from a binary search tree.

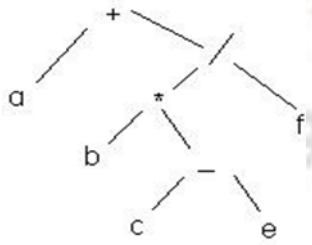
Answer: (1.) t has no right hand child node t->r == z

(2.) t has a right hand child but its right hand child node has no left sub tree t->r->l == z

(3.) t has a right hand child node and the right hand child node has a left hand child node t->r->l != z

Question4.Give the pre & postfix form of the expression (a + ((b*(c-e))/f).

Answer:



Question5. Define a heap. How can it be used to represent a priority queue?

Answer: A priority queue is a different kind of queue, in which the next element to be

removed is defined by (possibly) some other criterion. The most common way to implement a priority queue is to use a different kind of binary tree, called a heap. A heap avoids the long paths that can arise with binary search trees.

Unit-4

Question1. What is binary search?

Answer: For binary search, the array should be arranged in ascending or descending order.In each step, the algorithm compares the search key value with the middle element of the array. If the key match, then a matching element has been found and its index, or Position, is returned.

Otherwise, if the search key is less than the middle element, then the algorithm repeats its action on the sub-array to the left of the middle element or, if the search key is greater, on the sub-array to the right.

Question2. Write the function in c for shell sort?

Answer: Void Shellsort(Elementtype A[ ],int N)

```
{
int i , j , increment ; elementtype tmp ;
for(elementtype=N / 2;increment > 0;increment / = 2) For( i= increment ; i
<N ; i ++)
{
tmp=A[ ];
for( j=I; j>=increment; j - =increment) if(tmp< A[ ]=A[j –
increment];
A[ j ]=A[ j – increment]; Else
Break;
A[ j ]=tmp;
}}
```

Question3. How the insertion sort is done with the array?

Answer: It sorts a list of elements by inserting each successive element in the previously Sorted sub listed.

Consider an array to be sorted A[1],A[2],….A[n]

(a). Pass 1: A[2] is compared with A[1] and placed them in sorted order.

(b). Pass 2: A[3] is compared with both A[1] and A[2] and inserted at an appropriate place. This makes A[1], A[2],A[3] as a sorted sub array.

(c). Pass n-1 : A[n] is compared with each element in the sub array A [1], A [2] …A [n-1] and inserted at an appropriate position.

Question4. What is open addressing?

Answer: Open addressing is also called closed hashing, which is an alternative to resolve the Collisions with linked lists. In this hashing system, if a collision occurs, alternative cells are tired until an empty cell is found.

There are three strategies in open addressing:

- Linear probing
- Quadratic probing
- Double hashing

Question5. Define searching?

Answer: Searching refers to determining whether an element is present in a given list of elements or not. If the element is present, the search is considered as successful, otherwise it is considered as an unsuccessful search. The choice of a searching technique is based on the following factors

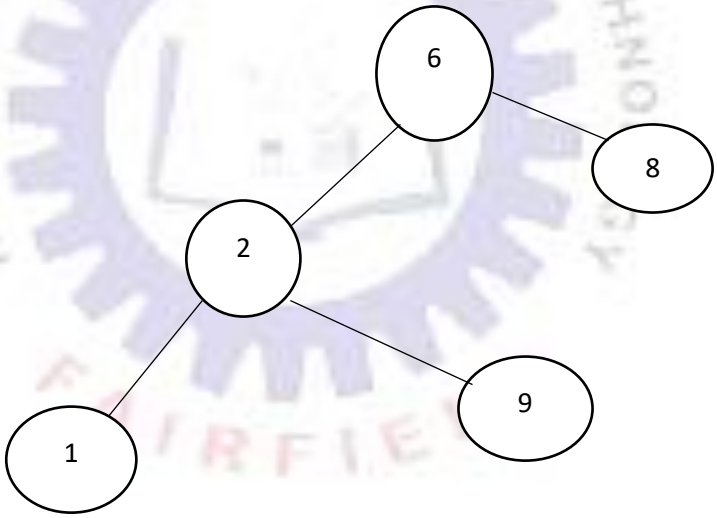
a. Order of elements in the list i.e., random or sorted

b. Size of the list

Unit-3

Question1. Give an algorithm to check whether the given binary tree is a BST or not.

Answer: Consider the following simple program. For each node, check if left node of it is smaller than the node and right node of it is greater than the node. This approach is wrong as this will return true for below binary tree. Checking only at current node is not enough.




```
Int IsBST(struct BinaryTreeNode*root){
    If(root=NULL) return 1:
   /*false if left is > than root */
  If(root=left !-NULL && root –left –data> root – data)
        Return0;
```

```
  /*false if right is < than root*/
If (root-right! =NULL && root-right-data<root-data)
        Return0;
 /*false if, recursively, the left or right is not a BST*/
If(!IsBST(root-left)||!IsBST(root-right))
        Return0;
/* passing all that, it's a BST*/
Return  1;
```

Question2. Give an algorithm for finding the size of binary tree.

Answer: calculate the size of left and right subtree recursively, add 1 (currently node) and return to its parent,// compute the number of nodes in a tree.

```
Int sizeOfBinary treeNode*root){
    If(root=NULL)
        Return0;
 Else     return(sizeOfBinaryTree(root-left ) + 1+SizeOfBinaryTree(root-right));
}
```

Time complexity: O(n). Space Complexity:O(n).

Question3- What are the steps to convert a general tree into binary tree?

Answer- The steps to convert a general tree into binary tree are ;

i. use the root of the general tree as the root of the binary tree

ii. Determine the first child of the root. This is the leftmost node in the general tree at the next level

iii. Insert this node. The child reference of the parent node refers to this node.

iv. Continue finding the first child of each parent node and insert it below the parent node with the child reference of the parent to this node.

v. When no more first children exist in the path just used, move back to the parent of the last node entered and repeat the above process. In other words, determine the first sibling of the last node entered.

vi. Complete the tree for all nodes. In order to locate where the node fits you must search for the first child at that level and then follow the sibling references to a nil where the next sibling can be inserted. The children of any sibling node can be inserted by locating the parent and then inserting the first child. Then the above process is repeated.

Question4. Define AVL tree. What is a balance factor in AVL trees?

Answer

AVL -AVL tree is a binary search tree except that for every node in the tree, the height of the left and right sub trees can differ by at most 1.

Balancing factor in AVL trees- Balance factor of a node is defined to be the difference between the height of the node's left subtree and the height of the node's right subtree.

Question5. What is meant by pivot node?

Answer: The node to be inserted travel down the appropriate branch track along the way of the deepest level node on the branch that has a balance factor of +1 or -1 is called pivot node.

Database Management System

Subject Code:BCA-110

1. **What is SQL (Structured Query Language)?**

Ans. Structured Query Language, is a database computer language designed for managing data in relational database management systems (RDBMS), and originally based upon relational algebra and calculus. Its scope includes data insert, query, update and delete, schema creation and modification, and data access control. SQL was one of the first commercial languages for Edgar F. Codd's relational model.

SQL is a DSL (Data Sub Language), which is really a combination of two languages. These are the Data Definition Language (DDL) and the Data Manipulation Language (DML). Schema changes are part of the DDL, while data changes are part of the DML.

2. **Explain Role and Resposibilities of DBA?**

Ans: The success of a database environment depends on central control of database design, implementation, and use. This central control and coordination is the role of the database administrator (DBA). The DBA is a single person; however, large organizations may divide DBA responsibilities among a team of personnel, each with specific skills and areas of responsibility such as database design, tuning, or problem resolution.

A **database administrator** (**DBA**) is a person responsible for the design, implementation, maintenance and repair of an organization's database. They are also known by the

titles *Database Coordinator* or *Database Programmer*, and is closely related to the *Database Analyst*, *Database Modeler*, *Programmer Analyst*, and *Systems Manager*. The role includes the development and design of database strategies, monitoring and improving database performance and capacity, and planning for future expansion requirements. They may also plan, co-ordinate and implement security measures to safeguard the database. Employing organizations may require that a database administrator have a certification or degree for database systems (for example, the Microsoft Certified Database Administrator). Some organizations have a hierarchical level of database administrators, generally:

- Data Analysts/Query designers
- Junior DBAs
- Midlevel DBAs
- DBA consultants

Personal Characteristics/Skills

1. Strong organizational skills
2. Strong logical and analytical thinker
3. Ability to concentrate and pay close attention to detail
4. Strong written and verbal communication skills
5. Willing to pursue education throughout your career.

Role of DBA:

Schema Definition:

The Database Administrator creates the database schema by executing DDL statements. Schema includes the logical structure of database table(Relation) like data types of attributes,length of attributes,integrity constraints etc.

Storage structure and access method definition

Database tables or indexes are stored in the following ways: Flat files,Heaps,B+ Tree etc..

Schema and physical organization modification

The DBA carries out changes to the existing schema and physical organization.

Granting authorization for data access

The DBA provides different access rights to the users according to their level. Ordinary users might have highly restricted access to data, while you go up in the hierarchy to the administrator ,you will get more access rights.

Routine Maintenance:

Some of the routine maintenance activities of a DBA is given below.

- Taking backup of database periodically
- Ensuring enough disk space is available all the time.
- Monitoring jobs running on the database.
- Ensure that performance is not degraded by some expensive task submitted by some users.
- Performance Tuning.

3. What is DBMS (Database Management System) ?

Ans: A Database Management System (DBMS) is a set of computer programs that controls the creation, maintenance, and the use of a database. It allows organizations to place control of database development in the hands of database administrators (DBAs) and other specialists. A DBMS is a system software package that helps the use of integrated collection of data records and files known as databases. It allows different user application programs to easily access the same database. DBMSs may use any of a variety of database models, such as the network model or relational model. In large systems, a DBMS allows users and other software to store and retrieve data in a structured way. Instead of having to write computer programs to extract information, user can ask simple questions in a query language. Thus, many DBMS packages provide Fourth-generation programming language (4GLs) and other application development features. It helps to specify the logical organization for a database and access and use the information within a database. It provides facilities for controlling data access, enforcing data integrity, managing concurrency, and restoring the database from backups. A DBMS also provides the ability to logically present database information to users.

4. What is the need of DBMS ?

Ans: A database management system (DBMS) can help address the employee count scenario and a range of even more complex situations related to cost, order status or inventory management by presenting the same data to everyone in the business at the same time. A DBMS also eliminates the frustrating hunt for the right version of the right spreadsheet on a vast and disorganized network drive.

- As businesses grow, the volume of data they accumulate grows exponentially. Managing this data deluge becomes increasingly difficult just at the moment when superior data

management becomes more important to business success.
• As businesses expand, more sophisticated tools are needed to manage data. Tools that serve start-ups well are overwhelmed by the demands faced by larger businesses.
• A database management system (DBMS) is a powerful tool used to store data, secure it, protect it and make it quickly available to people who need it..
• A DBMS enables a business to squeeze more value from the data it collects for improved decision-making.

5. What are Advantages and Disadvantages of DBMS?

Ans: The advantages and disadvantages of DBMS are as follows:

Advantages:

- Reduced data redundancy
- Reduced updating errors and increased consistency
- Greater data integrity and independence from applications programs
- Improved data acces s to users through use of host and query languages
- Improved data security
- Reduced data entry, storage, and retrieval costs
- Facilitated development of new applications program

Disadvantages:

Database systems are complex, difficult, and time-consuming to design

- Substantial hardware and software start-up costs
- Damage to database affects virtually all applications programs
- Extensive conversion costs in moving form a file-based system to a database system
- Initial training required for all programmers and users.

6. What is an Attribute?

Ans: An attribute in a table is a named column or they are the set of important properties which describes the particular entity. An attribute may consist of name, roll number, age etc. Relations are used to hold information about the object. The attributes in a relation can appear in any order and the relation is the same relation and hence conveys the same meaning. Attributes are set of qualities of qualities which are used to identify a particular entity.

7. **What are Derived Attributes?**

Ans: Derived attributes are those attributes which are based on and are derived from the attributes of another table or a relation. The derived attributes may contain new values or the values from the base table from which it was derived. Derived attributes are effectively read-only since there is no place to write them back to. Also, because derived attributes don't directly point to anything in the database, they cannot be used as primary keys. For example: a derived attribute person's full name may be derived from attribute person's first name and the last name.

8. **Explain the various keys in DBMS.**

Ans: **Primary Key:** The primary key of a relational table uniquely identifies each record in the table. It can either be a normal attribute that is guaranteed to be unique (such as Social Security Number in a table with no more than one record per person) or it can be generated by the DBMS . Primary keys may consist of a single attribute or multiple attributes in combination.

Examples: Imagine we have a STUDENTS table that contains a record for each student at a university. The student's unique student ID number would be a good choice for a primary key in the STUDENTS table. The student's first and last name would not be a good choice, as there is always the chance that more than one student might have the same name.

Super Key: A superkey is a combination of attributes that can be uniquely used to identify a database record. A table might have many superkeys. Candidate keys are a special subset of superkeys that do not have any extraneous information in them.

Examples: Imagine a table with the fields <Name>, <Age>, <SSN> and <Phone Extension>. This table has many possible superkeys. Three of these are <SSN>, <Phone Extension,

Name> and <SSN, Name>. Of those listed, only <SSN> is a candidate key, as the others contain information not necessary to uniquely identify records.

Candidate Key: A candidate key is a combination of attributes that can be uniquely used to identify a database record without any extraneous data. Each table may have one or more candidate keys. One of these candidate keys is selected as the table primary key.

In the relational model of databases, a **candidate key** of a relation is a minimal superkey for that relation; that is, a set of attributes such that

1. the relation does not have two distinct tuples with the same values for these attributes (which means that the set of attributes is a superkey)
2. there is no proper subset of these attributes for which (1) holds (which means that the set is minimal).

Since a relation contains no duplicate tuples, the set of all its attributes is a superkey if NULL values are not used. It follows that every relation will have at least one candidate key.

The candidate keys of a relation tell us all the possible ways we can identify its tuples. As such they are an important concept for the design database schema.

Foreign Key: A foreign key is a field (or fields) that points to the primary key of another table. The purpose of the foreign key is to ensure referential integrity of the data. In other words, only values that are supposed to appear in the database are permitted.

For example, say we have two tables, a CUSTOMER table that includes all customer data, and an ORDERS table that includes all customer orders. The constraint here is that all orders must be associated with a customer that is already in the CUSTOMER table. In this case, we will place a foreign key on the ORDERS table and have it relate to the primary key of the CUSTOMER table. This way, we can ensure that all orders in the ORDERS table are related to a customer in the CUSTOMER table. In other words, the ORDERS table cannot contain information on a customer that is not in the CUSTOMER table.

The structure of these two tables will be as follows:

Table ***CUSTOMER***

column name	characteristic
SID	Primary Key
Last_Name	

First_Name	

Table ***ORDERS***

column name	Characteristic
Order_ID	Primary Key
Order_Date	
Customer_SID	Foreign Key
Amount	

In the above example, the Customer_SID column in the ORDERS table is a foreign key pointing to the SID column in the CUSTOMER table.

Composite Key : A composite key is a key that uses more than one column to identify the data as opposed to a single column. This can sometimes be more useful than assigning each row an arbitrary value to use as a key such as an auto number field.

9. What is the domain of an Attribute?

Ans: Attribute domains are rules that describe the legal values of a field type, providing a method for enforcing data integrity. Attribute domains are used to constrain the values allowed in any particular attribute for a table or feature class. If the features in a feature class or nonspatial objects in a table have been grouped into subtypes, different attribute domains can be assigned to each of the subtypes. A domain is a declaration of acceptable attribute values. Whenever a domain is associated with an attribute field, only the values within that domain are valid for the field. In other words, the field will not accept a value that is not in that domain. Using domains helps ensure data integrity by limiting the choice of values for a particular field.

For example: Rooms in hotel (1-300)

Age (1-99)

Married (yes or no)

Nationality (Sri Lankan, Indian, American, or British)

10. What do you mean by Entity and Entity set?

Ans: An entity is a thing in the real world with an independent existance. and entity set is collection or set all entities of a particular entity type at any point of time.

* An entity is an object that exists and is distinguishable from other objects. For instance, John Harris with S.I.N. 890-12-3456 is an entity, as he can be uniquely identified as one particular person in the universe.

* An entity may be concrete (a person or a book, for example) or abstract (like a holiday or a concept).

* An entity set is a set of entities of the same type (e.g., all persons having an account at a bank).

* Entity sets need not be disjoint. For example, the entity set employee (all employees of a bank) and the entity set customer (all customers of the bank) may have members in common.

* An entity is represented by a set of attributes.
o E.g. name, S.I.N., street, city for "customer" entity.

11. What is a Log File?

Ans: A **log file** is a recording of everything that goes in and out of a particular server. It is a concept much like the black box of an airplane that records everything going on with the plane in the event of a problem. The information is frequently recorded chronologically, and is located in the root directory, or occasionally in a secondary folder, depending on how it is set up with the server. The only person who has regular access to the log files of a server is the server administrator, and a log file is generally password protected, so that the server administrator has a record of everyone and everything that wants to look at the log files for a specific server.

The point of a **log file** is to keep track of what is happening with the server. If something should malfunction within a complex system, there may be no other way of identifying the problem. Log files are also used to keep track of complex systems, so that when a problem does occur, it is easy to pinpoint and fix. Log files are also important to keeping track of applications that have little to no human interaction, such as server applications. There are times when log files are too difficult to read or make sense of, and it is then that log file analysis is necessary. Log file analysis is generally performed by some kind of computer program that makes the log file information more concise and readable format. Log files can also be used to correlate data between servers, and find common problems between different systems that might need one major solution to repair them all.

12. What is Normalization?

Ans: Normalization is the process of efficiently organizing data in a database. There are two goals of the normalization process: eliminating redundant data (for example, storing the same data in more than one table) and ensuring data dependencies make sense (only storing related data in a table). Both of these are worthy goals as they reduce the amount of space a database consumes and ensure that data is logically stored.

Why it is requried?

Normalization reduces redundancy. Redundancy is the unnecessary repetition of data. It can cause problems with storage, reterival and updation of data. Redundancy can lead to:

- **Inconsistencies:-**errors are more likely to occur when facts are repeated.
- **Update anomalies:-**inserting, modifying and deleting data may cause inconsistencies. Inconsistency occurs when we perform updation or deletion of data in one relation, while forgetting to make corresponding changes in other relations.

During the process of normalization, you can identify dependencies, which can cause problems when deleting or updating. Normalization also helps to simplify the structure of the tables. A fully normalized record consist of:

- A primary key that identifies that entity.
- A set of attributes that describe that entity.

13. What is the Difference between primary key and unique key?

Ans: The column holding the primary key constraint cannot accept null values.whereas colum holding the unique constraint can accept null values assume that t3 is a table with two columns t1 having primary key constraint and t2 having unique constraint if u try to insert null into t2 it will accept that values whereas column t1 will not accept null. Each table having only one PRIMARY KEY.And my contain many UNIQUE KEYS.

14. What is Data Redundancy?

Ans: Data redundancy occurs in database systems which have a field that is repeated in two or more tables. For instance, in case when customer data is duplicated and attached with each product bought then redundancy of data is a known source of inconsistency, since customer might appear with different values for given attribute.Data redundancy leads to data anomalies and corruption and generally should be avoided by design.Database normalization prevents redundancy and makes the best possible usage of storage.Proper use of foreign keys can minimize data redundancy and chance of destructive anomalies. However

sometimes concerns of efficiency and convenience can result redundant data design despite the risk of corrupting the data.

15. What is the Difference between Single valued and multi valued attributes ?

Ans: A single valued attribute can have only a single value. For example a person can have only one 'date of birth', 'age' etc. That is a single valued attributes can have only single value. But it can be simple or composite attribute.That is 'date of birth' is a composite attribute , 'age' is a simple attribute. But both are single valued attributes.

Multivalued attributes can have multiple values. For instance a person may have multiple phone numbers,multiple degrees etc.Multivalued attributes are shown by a double line connecting to the entity in the ER diagram.

Single Valued Attribute: Attribute that hold a single value

Example1: Age

Exampe2: City

Example3:Customer id

Multi Valued Attribute: Attribute that hold multiple values.

Example1: A customer can have multiple phone numbers, email id's etc

Example2: A person may have several college degrees.

16 What is the difference between Strong and weak entity?

Ans: An entity set that does not have sufficient attributes to form a primary key is termed as a weak entity set. An entity set that has a primary key is termed as strong entity set.

A weak entity is existence dependent. That is the existence of a weak entity depends on the existence of a identifying entity set. The discriminator (or partial key) is used to identify other attributes of a weak entity set.The primary key of a weak entity set is formed by primary key of identifying entity set and the discriminator of weak entity set. The existence of a weak entity is indicated by a double rectangle in the ER diagram. We underline the discriminator of a weak entity set with a dashed line in the ER diagram.

17. What do you mean by Integrity Constraints?

Ans: Integrity constraints are used to ensure accuracy and consistency of data in a relational database. Data integrity is handled in a relational database through the concept of referential integrity. There are many types of integrity constraints that play a role in referential integrity

Entity integrity:-The entity integrity constraint states that no primary key value can be null. This is because the primary key value is used to identify individual tuples in a relation . Having null value for the primary key implies that we cannot identify some tuples.This also specifies that there may not be any duplicate entries in primary key column.

Referential Integrity:-The referential integrity constraint is specified between two relations and is used to maintain the consistency among tuples in the two relations. Informally, the referential integrity constraint states that a tuple in one relation that refers to another relation must refer to an existing tuple in that relation.

Domain Integrity:-The domain integrity states that every element from a relation should respect the type and restrictions of its corresponding attribute. A type can have a variable length which needs to be respected. Restrictions could be the range of values that the element can have, the default value if none is provided, and if the element can be NULL.

18. What is a Data Warehouse?

Ans: A data warehouse (DW) is a database used for reporting . The data is offloaded from the operational systems for reporting. The data may pass through an operational data store for additional operations before it is used in the DW for reporting.

A data warehouse maintains its functions in three layers: staging, integration, and access. *Staging* is used to store raw data for use by developers (analysis and support). The *integration* layer is used to integrate data and to have a level of abstraction from users. The *access* layer is for getting data out for users.

This definition of the data warehouse focuses on data storage. The main source of the data is cleaned, transformed, catalogued and made available for use by managers and other business professionals for data mining, online analytical processing, market research and decision support (Marakas & OBrien 2009). However, the means to retrieve and analyze data, to extract, transform and load data, and to manage the data dictionary are also considered essential components of a data warehousing system. Many references to data warehousing use this broader context. Thus, an expanded definition for data warehousing includes business intelligence tools, tools to extract, transform and load data into the repository, and tools to manage and retrieve metadata.

19. Explain Referential Integrity?

Ans: Referential integrity is a database concept that ensures that relationships between tables remain consistent. When one table has a foreign key to another table, the concept of referential integrity states that you may not add a record to the table that contains the foreign key unless there is a corresponding record in the linked table. It also includes the techniques known as cascading update and cascading delete, which ensure that changes made to the linked table are reflected in the primary table.

Consider the situation where we have two tables: Employees and Managers. The Employees table has a foreign key attribute entitled ManagedBy which points to the record for that

employee's manager in the Managers table. Referential integrity enforces the following three rules:

1. We may not add a record to the Employees table unless the Managed By attribute points to a valid record in the Managers table.
2. If the primary key for a record in the Managers table changes, all corresponding records in the Employees table must be modified using a cascading update.
3. If a record in the Managers table is deleted, all corresponding records in the Employees table must be deleted using a cascading delete.

20. What is the use of DROP command and what are the differences between DROP, TRUNCATE and DELETE commands?

Answer: DROP command is a DDL command which is used to drop/delete the existing table, database, index or view from the database.

The major difference between DROP, TRUNCATE and DELETE commands are:

DROP and TRUNCATE commands are the DDL commands which are used to delete tables from the database and once the table gets deleted, all the privileges and indexes that are related to the table also get deleted. These 2 operations cannot be rolled back and so should be used only when necessary.

DELETE command, on the other hand, is a DML Command which is also used to delete rows from the table and this can be rolled back.

Note: It is recommended to use the 'WHERE' clause along with the DELETE command else the complete table will get deleted from the database.