SWITCHING THEORY AND LOGIC DESIGN

II Year - I Semester **BRANCH: ECE**

UNIT – I: REVIEW OF NUMBER SYSTEMS & CODES:

i) Representation of numbers of different radix, conversation from one radix to another radix, r-1's compliments and r's compliments of signed members, problem solving.
ii) 4 bit codes, BCD, Excess-3, 2421, 84-2-1 9's compliment code etc.,
iii) Logic operations and error detection & correction codes; Basic logic operations -NOT, OR, AND, Universal building blocks, EX-OR, EX-NOR - Gates, Standard SOP and POS, Forms, Gray code, error detection, error correction codes (parity checking, even parity, odd parity, Hamming code) NAND-NAND and NOR-NOR realizations.

UNIT – II: MINIMIZATION TECHNIQUES

Boolean theorems, principle of complementation & duality, De-Morgan theorems, minimization of logic functions using Boolean theorems, minimization of switching functions using K-Map up to 6 variables, tabular minimization, problem solving (code-converters using K-Map etc..).

UNIT – III: COMBINATIONAL LOGIC CIRCUITS DESIGN

Design of Half adder, full adder, half subtractor, full subtractor, applications of full adders, 4-bit binary subtractor, adder-subtractor circuit, BCD adder circuit, Excess 3 adder circuit, look-a-head adder circuit, Design of decoder, demultiplexer, 7 segment decoder, higher order demultiplexing, encoder, multiplexer, higher order multiplexing, realization of Boolean functions using decoders and multiplexers, priority encoder, 4-bit digital comparator.

UNIT – IV: INTRODUCTION OF PLD's

PROM, PAL, PLA-Basics structures, realization of Boolean function with PLDs, programming tables of PLDs,merits & demerits of PROM, PAL, PLA comparison, realization of Boolean functions using PROM, PAL, PLA, programming tables of PROM, PAL, PLA.

UNIT – V: SEQUENTIAL CIRCUITS I

Classification of sequential circuits (synchronous and asynchronous); basic flip-flops, truth tables and excitation tables (nand RS latch, nor RS latch, RS flip-flop, JK flip-flop, T flip-flop, D flip-flop with reset and clear terminals). Conversion from one flip-flop to flip-flop. Design of ripple counters, design of synchronous counters, Johnson counter, ring counter. Design of registers - Buffer register, control buffer register, shift register, bi-directional shift register, universal shift register.

UNIT – VI: SEQUENTIAL CIRCUITS II

Finite state machine; Analysis of clocked sequential circuits, state diagrams, state tables, reduction of state tables and state assignment, design procedures. Realization of circuits using various flip-flops. Meelay to Moore conversion and vice-versa.

TEXT BOOKS:

1. Switching Theory and Logic Design by Hill and Peterson Mc-Graw Hill TMH edition.
2. Switching Theory and Logic Design by A. Anand Kumar
3. Digital Design by Mano PHI.

REFERENCE BOOKS:

1. Modern Digital Electronics by RP Jain, TMH
2. Fundamentals of Logic Design by Charles H. Roth Jr, Jaico Publishers
3. Micro electronics by Milliman MH edition.

Unit – I

Objectives:

- To familiarize with the concepts of different number systems and codes.

Syllabus:

REVIEW OF NUMBER SYSTEMS & CODES: Representation of numbers of different radix, conversion of numbers from one radix to another radix, r-1's complement and r's complement of signed numbers, problem solving.

- 4 bit codes, BCD, Excess-3, 2421, 84-2-1 9's compliment code etc., Logic operations and error detection & correction codes; Basic logic operations -NOT, OR, AND,Universal building blocks, EX-OR, EX-NOR - Gates, Standard SOP and POS, Forms, Gray code, error detection, error correction codes (parity checking, even parity, odd parity, Hamming code) NAND-NAND and NOR-NOR realizations

Outcomes:

Students will be able to

- understand various number systems.
- perform the arithmetic operations using complementary methods.
- distinguish the representation of signed and unsigned numbers.
- classify various numeric and alphanumeric codes.
- understand basic logic operations and gates.
- determine the parity of binary codes.
- perform the Two level NAND – NAND and NOR-NOR realizations of Boolean expressions.

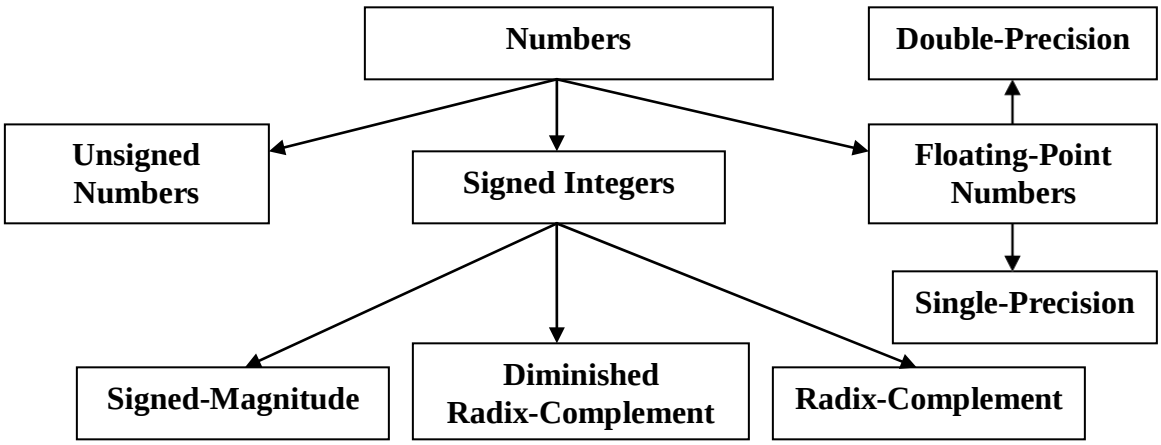
Learning Material

Number Systems

Purposes:

1. **To understand how does a digital computer work**. Binary digital computers only work with 1's and 0's, or high and low voltage, or true and false.
2. **To convert among different number systems**. We use **decimal** numbers everyday. Computers understand only **binary** numbers, which are lengthy and inconvenient to human beings. **Octal** and **Hexadecimal** numbers are introduced to make both happy: they are easier to be converted to binary numbers and also easier for us to handle.

Classification:

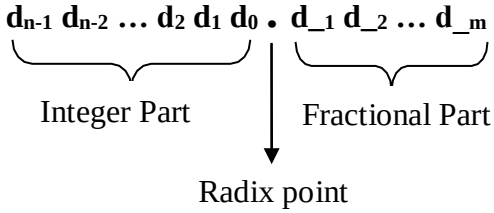


Unsigned Numbers

Radices and Characters:

- **Binary:** 0, 1
- **Decimal:** 0, 1, 2, 3, 4, 5, 6, 7, 8, 9
- **Octal:** 0, 1, 2, 3, 4, 5, 6, 7
- **Hexadecimal:** 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F

Structure of a number:



Note: If no fractional part, the radix point can be omitted!

Positional Notation or representation of numbers:

$N = d_{n-1}r^{n-1} + d_{n-2}r^{n-2} + \cdots + d_1 r^1 + d_0 r^0 + d_{-1}r^{-1} + d_{-2}r^{-2} + \cdots + d_{-m}r^{-m}$,

where $d_i \in \{0,1,2,\ r-1\}$, $i \in \{n-1,\ n-2, \cdots 2,1,0,-1,-2,\cdots -m\}$, and r is the radix.

The number of numerical values the system uses is called the **Base or Radix** of the system

System	Radix	Allowable Digits
Binary	2	0,1
Octal	8	0,1,2,3,4,5,6,7
Decimal	10	0,1,2,3,4,5,6,7,8,9
Hexadecimal	16	0,1,2,3,4,5,6,7,8,9,A, B, C, D, E, F

Conversion of numbers from one radix to another radix

- Conversion from given base to Decimal:

  write the number using the positional notation and then perform decimal arithmetic to compute the result, which is the decimal number.

Example: Given the positional notations of the following numbers: $(1101.1)_2$, $(724)_8$, and $(BCD)_{16}$.

- $(4021.2)_5 = 4 \times 5^3 + 0 \times 5^2 + 2 \times 5^1 + 1 \times 5^0 + 2 \times 5^{-1} = (511.4)_{10}$
  $4 \times 125 + 0 + 10 + 1 + 2 \times (1/5)$
  $500 + 11 + .4$

- $(B65F)_{16} = 11 \times 16^3 + 6 \times 16^2 + 5 \times 16^1 + 15 \times 16^0 = (46687)_{10}$
  $11 \times 4096 + 6 \times 256 + 5 \times 16 + 15$
  $45056 + 1536 + 80 + 15$

- $(1010.011)_2 = 2^3 + 2^1 + 2^{-2} + 2^{-3} = (10.375)_{10}$

- $(630.4)_8 = 6 \times 8^2 + 3 \times 8^1 + 0 \times 8^0 + 4 \times 8^{-1} = (408.5)_{10}$

- Conversion from Decimal to given base:

  Integer part: Divide the decimal number by the base to which we want to convert and cast out the reminders.
  Fractional part: Multiply the decimal number by the base to which we want to convert and cast out the integer part.

Rationale: based on the **positional notation.**

The conversion of decimal numbers with both integers and fraction parts is done by converting the integer and fraction separately and then combining the two answers.

Example: Convert $(210)_{10}$ to binary and to hexadecimal (Radix 16).

- $(210)_{10} = 1 \times 2^7 + 1 \times 2^6 + 0 \times 2^5 + 1 \times 2^4 + 0 \times 2^3 + 0 \times 2^2 + 1 \times 2^1 + 0 \times 2^0$

$= 128 + 64 + 0 + 16 + 0 + 0 + 1 + 0$

$= (11010010)_2$

- $(210)_{10} = 13 \times 16^1 + 2 \times 16^0$

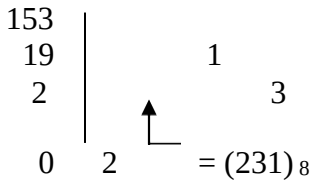
$= 208 + 2 = 210 = (D2)_{16}$

- Conversion from Decimal 41 to Binary:

		Integer quotient		**Remainder**	**Coefficient**
41/2	=	20	+	½	$a_0 = 1$
20/2	=	10	+	0	$a_1 = 0$
10/2	=	5	+	0	$a_2 = 0$
5/2	=	2	+	½	$a_3 = 1$
2/2	=	1	+	0	$a_4 = 0$
1/2	=	0	+	½	$a_5 = 1$

- The conversion from decimal integers to any base-*r* system is similar to the example, except that division is done by *r* instead of 2.

- Conversion from Decimal 153 to Octal:



- Conversion from Decimal fraction $(0.6875)_{10}$ to Binary:

	Integer		**Fraction**	**Coefficient**
0.6875 x 2 =	1	+	0.3750	$a_{-1} = 1$
0.3750 x 2 =	0	+	0.7500	$a_{-2} = 0$
0.7500 x 2 =	1	+	0.5000	$a_{-3} = 1$
0.5000 x 2 =	1	+	0.0000	$a_{-4} = 1$

- The conversion from decimal fraction to any base-*r* system is similar to the example. Multiplication is by *r* instead of **2**, and the coefficients found from the integers may range in value from 0 to *r*-1 instead of **0** and **1**.

- Conversion from Decimal fraction $(0.513)_{10}$ to Octal:

$$0.513 \times 8 = 4.104$$
$$0.104 \times 8 = 0.832$$
$$0.832 \times 8 = 6.656$$
$$0.656 \times 8 = 5.248$$
$$0.248 \times 8 = 1.984$$
$$0.984 \times 8 = 7.872$$

$$(0.513)_{10} = (0.406517\ldots)_{8}$$

Binary to/from Octal and Hexadecimal: Starting at the binary point, cast off three (four) bits at a time and convert each group to its octal (hexadecimal) equivalent. Padding 0's to the left for the integer part and to the right for the fractional part when necessary.

The conversion from and to binary, octal and hexadecimal plays an important part in digital computers. Since $2^3 = 8$ and $2^4 = 16$, each octal digit corresponds to three binary digits and each hexadecimal digit corresponds to four binary digits.

- Conversion from binary to Octal:

  $(10\ 110\ 001\ 101\ 011.\ 111\ 100\ 000\ 110)_{2} = (26153.7406)_{8}$

- Conversion from binary to Hexadecimal:

  $(10\ 1100\ 0110\ 1011.\ 1111\ 0000\ 0110)_{2} = (2C6B.F06)_{16}$

- Conversion from Octal to binary:

  $(673.124)_{8} = (110\ 111\ 011.\ 001\ 010\ 100)_{2}$

- Conversion from Hexadecimal to binary:

  $(306.D)_{16} = (0011\ 0000\ 0110.\ 1101)_{2}$

- Conversion from Hexadecimal to Decimal:

  $(37B)_{16}$
  $3 \times 16^2 + 7 \times 16^1 + 11 \times 16^0$
  $= 3 \times 256 + 7 \times 16 + 11 \times 1$
  $= 768 + 112 + 11$
  $= (891)_{10}$

r-1's complement and r's complement of unsigned numbers subtraction:

9's & 10's Complements for decimal numbers:

- The Subtraction of decimal numbers can be accomplished by the 9's & 10's compliment methods similar to the 1's & 2's compliment methods of binary numbers.
- The 9's compliment (*diminished radix complement*) of a decimal number is obtained by subtracting each digit of that decimal number from 9.
- The 10's compliment (*radix complement*) of a decimal number is obtained by adding a 1 to its 9's compliment.

Example:
9's compliment of 3465 and 782.54 is

```
 9999                    999.99
-3465                   -782.54
----------              -----------
6534                    217.45
------------------      --------------------
```

10's complement of 4069 is

```
  9999 -
4069
----------
 5930
   +1
----------
 5931
-----------
```

9's compliment method of subtraction:

To perform this, obtain the 9's compliment of the subtrahend and to it, add the minuend, now call this number as intermediate result. If there is a carry to the LSD of this result to get the answer called **end around carry**. If there is no carry, it indicates that the answer is negative & the intermediate result is its 9's compliment.

Example: Subtract using 9's complement

(1) 745.81- 436.62

```
 745.81     (normal subtraction)
-436.62
----------
309.19
-----------
 745.81
+563.37     9's compliment of 436.62
```

(2) 436.62 - 745.82

```
 436.62
-745.81
----------
-309.19
---------
 436.62
+254.18
```

1309.18	(end around carry)	690.80 (no carry)
+1		9‘s complement of 690.80
+309.19		= - 309.19

- If there is no carry indicating that answer is negative. so take 9‘s complement of intermediate result & put minus sign (-) then the result should be -309.19.
- If there is a carry indicates that the answer is positive +309.19. Then there is no need of taking 9‘s complement.

10’s compliment method of subtraction:

- To perform this, obtain the 10‘s compliment of the subtrahend & add it to the minuend. If there is a carry ignore it.
- The presence of the carry indicates that the answer is positive, the result is the answer.
- If there is no carry, it indicates that the answer is negative & the result is its 10‘s compliment.
- Obtain the 10‘s compliment of the result & place negative sign infront to get the answer.

Example:

(a) 2928.54 - 416.73

2928.54	(normal subtraction)
-0416.73	
2511.81	
2928.54	
+9583.27	10‘s compliment of 416.73
12511.81	ignore the carry
+2511.81	

(b) 416.73 - 2928.54

0416.73
-2928.54
-2511.81
0416.73
+7071.46
7488.19
(10’s complement)
-2511.81

1‘s & 2’s compliment form for binary numbers:

- The **1‘s complement** of a binary number is defined as the value obtained by inverting all the bits in the binary representation of the number (swapping 0s for 1s and vice versa).

Example:
For X = 1010, the 1's complement is given by 0101.

- The **2's complement** of a binary number X is obtained by following three methods

1. The expression $2^n - X$, where n is the number of bits of X.
2. All the bits are inverted (1's complement) and a 1 is added in the least significant place.
3. The lowest order 1 in X is sensed, and all succeeding higher digits are inverted.

Example:
For X = 1010, the 2's complement is given by:

1. $2^4 - 1010 = 10000 - 1010 = 0110$.
2. 1's complement of 1010 is 0101 and 0101 + 1 = 0110.
3. The low order 1 in 1010 is at 1st bit position and after that the higher digits are inverted and the result is 1010.

$X = 1010$
Invert Sense
$[X]_2 = 0110$

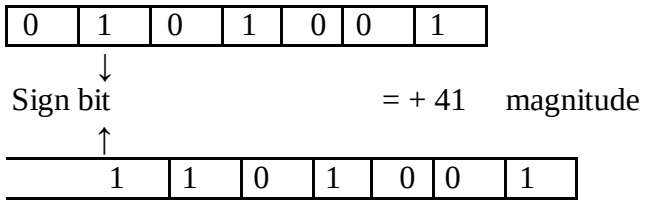
Signed binary numbers:

Two ways of representation of signed numbers
1. Sign Magnitude form
2. Complemented form

Sign Magnitude form:

- In sign magnitude form, an additional bit called the sign bit is placed in front of the number.
- If the sign bit is 0, the number is positive, and if it is a 1, then the number is negative.

Example:



Representation of signed numbers using 2's or 1's complement method:

- If the number is positive, the magnitude is represented in its true binary form & a sign bit 0 is placed in front of the MSB.
- If the no is negative, the magnitude is represented in its 2's or 1's compliment form & a sign bit 1 is placed in front of the MSB.

Example:

Sign bit magnitude

↓

0	1	1	0	0	1	1

In any form

= +51

1	1	1	0	0	1	1

In sign magnitude form

= -51

1	0	0	1	1	0	1

In sign 2‘s compliment form

= -51

1	0	0	1	1	0	0

In sign 1‘s compliment form

= -51

Given no.	Sign magnitude form	2‘s complement form	1‘s complement form
01101	+13	+13	+13
010111	+23	+23	+23
10111	-7	-9	-8
1101010	-42	-22	-21

Special case in 2’s complement representation:

Whenever a signed no. has a 1 in the sign bit & all 0‘s for the magnitude bits, the decimal equivalent is -2^n, where n is the no of bits in the magnitude.

Example:

1000 = -8 & 10000 = -16

2’s compliment Arithmetic:

- The 2‘s complement system is used to represent positive numbers using modulus arithmetic.
- The word length of a computer is fixed. i.e., if a 4-bit number is added to another 4-bit number, the result will be only of 4 bits.
- Carry if any, from the fourth bit will overflow called the Modulus arithmetic.
  Example: 1100+1111=1011

- In the 2‘s complement subtraction, add the 2‘s complement of the subtrahend to the minuend.
- If there is a carry out, ignore it and look at the sign bit i.e., MSB of the sum term.
- If the MSB is a 0, the result is positive and it is in true binary form.
- If the MSB is a 1 (carry in or no carry at all) the result is negative and is in its 2‘s complement form. Take its 2‘s complement to find its magnitude in binary.

Example:

Subtract 14 from 46 using 8-bit 2‘s complement arithmetic:

+14 = 00001110
-14 = 11110010 2‘s complement of 14

+46 = 00101110
-14 = +11110010 2‘s complement form of 14

-32 (1)00100000 ignore carry

Ignore carry and the MSB is 0. So, the result is positive and is in normal binary form. So the result is +00100000 = +32.

Example: Add -75 to +26 using 8-bit 2‘s complement arithmetic

+75 = 01001011
-75 = 10110101 2‘s complement of 75

+26 = 00011010
-75 = +10110101

-49 11001111 No carry

No carry and MSB is 1. So the result is negative and is in 2‘s complement form. The magnitude is 2‘s complement of 11001111. i.e., 00110001 = 49. So result is -49

1’s compliment arithmetic:

- In 1‘s complement subtraction, add the 1‘s complement of the subtrahend to the minuend.
- If there is a carryout, bring the carry around & add it to the LSB called the **end around carry.**
- Look at the sign bit (MSB). If this is a 0, the result is positive and a true binary number.
- If the MSB is a 1 (carry or no carry), the result is negative and in complement form. Take its 1‘s complement to get the magnitude in binary.

Example: Using 8-bit 1‘s complement

Subtract 14 from 25

25 = 00011001
-14 = 11110001

+11 (1)00001010
+1

00001011

MSB is a 0 so result is positive (true binary)

ADD -25 to +14

+14 = 00001110
-25 = +11100110

-11 11110100

No carry and MSB = 1

Result is negative and in 1’s complement form

Compliment Arithmetic Advantage:

Subtraction is also performed by addition. Instead of subtracting one number from other the compliment of the subtrahend is added to minuend.

Codes

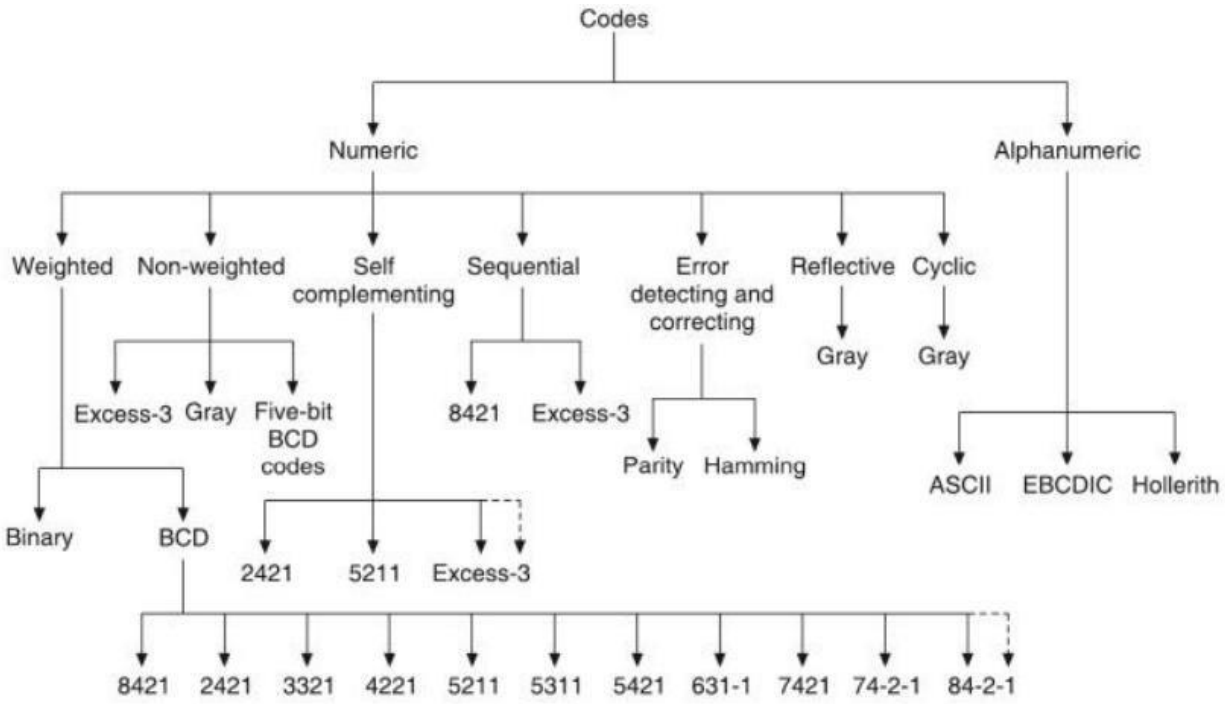
The digital data is represented, stored and transmitted as group of binary bits. This group is also called as **binary code**. The binary code is represented by the number as well as alphanumeric letter.

Advantages of Binary Code

Following is the list of advantages that binary code offers.

- Binary codes are suitable for the computer applications.
- Binary codes are suitable for the digital communications.
- Binary codes make the analysis and designing of digital circuits if we use the binary codes.
- Since only 0 & 1 are being used, implementation becomes easy.

Classification of codes



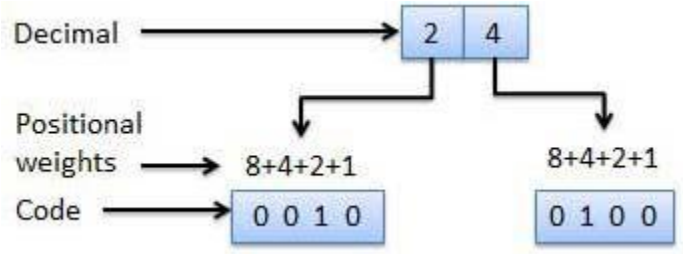
The codes are broadly categorized into following four categories.

- Weighted Codes
- Non-Weighted Codes

- Binary Coded Decimal Code
- Alphanumeric Codes
- Error Detecting Codes
- Error Correcting Codes

Weighted Codes

Weighted binary codes are those binary codes which obey the positional weight principle. Each position of the number represents a specific weight. Several systems of the codes are used to express the decimal digits 0 through 9. In these codes each decimal digit is represented by a group of four bits.

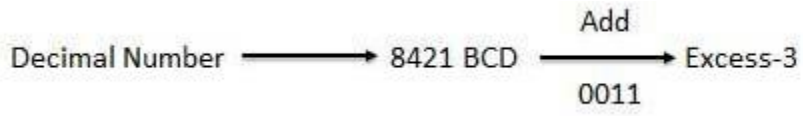


Non-Weighted Codes

In this type of binary codes, the positional weights are not assigned. The examples of non-weighted codes are Excess-3 code and Gray code.

Excess-3 code

The Excess-3 code is also called as XS-3 code. It is non-weighted code used to express decimal numbers. The Excess-3 code words are derived from the 8421 BCD code words adding (0011)2 or (3)10 to each code word in 8421. The excess-3 codes are obtained as follows −



Example:

Decimal	BCD 8 4 2 1	Excess-3 BCD + 0011
0	0 0 0 0	0 0 1 1
1	0 0 0 1	0 1 0 0
2	0 0 1 0	0 1 0 1
3	0 0 1 1	0 1 1 0
4	0 1 0 0	0 1 1 1
5	0 1 0 1	1 0 0 0
6	0 1 1 0	1 0 0 1
7	0 1 1 1	1 0 1 0
8	1 0 0 0	1 0 1 1
9	1 0 0 1	1 1 0 0

Binary Coded Decimal (BCD) code

In this code each decimal digit is represented by a 4-bit binary number. BCD is a way to express each of the decimal digits with a binary code. In the BCD, with four bits we can represent sixteen numbers (0000 to 1111). But in BCD code only first ten of these are used (0000 to 1001). The remaining six code combinations i.e. 1010 to 1111 are invalid in BCD.

Decimal	0	1	2	3	4	5	6	7	8	9
BCD	0000	0001	0010	0011	0100	0101	0110	0111	1000	1001

Advantages of BCD Codes

- It is very similar to decimal system.
- We need to remember binary equivalent of decimal numbers 0 to 9 only.

Disadvantages of BCD Codes

- The addition and subtraction of BCD have different rules.
- The BCD arithmetic is little more complicated.
- BCD needs more number of bits than binary to represent the decimal number. So BCD is less efficient than binary.

Alphanumeric codes

A binary digit or bit can represent only two symbols as it has only two states '0' or '1'. But this is

not enough for communication between two computers because there we need many more symbols for communication. These symbols are required to represent 26 alphabets with capital and small letters, numbers from 0 to 9, punctuation marks and other symbols.

The alphanumeric codes are the codes that represent numbers and alphabetic characters. Mostly such codes also represent other characters such as symbol and various instructions necessary for conveying information. An alphanumeric code should at least represent 10 digits and 26 letters of alphabet i.e. total 36 items. The following three alphanumeric codes are very commonly used for the data representation.

- American Standard Code for Information Interchange (ASCII).
- Extended Binary Coded Decimal Interchange Code (EBCDIC).
- Five bit BCD Code.

ASCII code: ASCII code is a 7-bit code whereas EBCDIC is an 8-bit code. ASCII code is more commonly used worldwide while EBCDIC is used primarily in large IBM computers.

Sequential Code: These are those codes in which each succeeding code is 1 binary number greater than the preceding code. This property is used for mathematical manipulation of data. For ex:- BCD And Excess-3 Code.

Self-complementary Code: A code is said to be self-complementary if the code for 9's complement of N i.e. 9-N can be obtained by interchanging all 0s and 1s.

- Decimal 9 is the complement of code for 0, 8 for 1, 7 for 2 and so on.
- For a code to be self complementing, the sum of all its weights must be 9. Digit. 8421 and 5421 codes are not self complementing codes whereas 5211, 2421, 3321, 4321 are self complementing.
- In general, a code is self-complementary if we produce a code by taking the first complement of the digit which is same as 9's complement of the number.

Cyclic codes:

- Cyclic codes are those in which each successive code word differs from the preceding one in only one bit position.
- They are also called unit distance codes
- Example: gray code Reflective Code: Example : Gray code

Binary–Gray Code Conversion A given binary number can be converted into its Gray code equivalent by going through the following steps:

- Begin with the most significant bit (MSB) of the binary number. The MSB of the Gray

code equivalent is the same as the MSB of the given binary number.

- The second most significant bit, adjacent to the MSB, in the Gray code number is obtained by adding the MSB and the second MSB of the binary number and ignoring the carry, if any. That is, if the MSB and the bit adjacent to it are both '1', then the corresponding Gray code bit would be a '0'.
- The third most significant bit, adjacent to the second MSB, in the Gray code number is obtained by adding the second MSB and the third MSB in the binary number and ignoring the carry, if any.
- The process continues until we obtain the LSB of the Gray code number by the addition of the LSB and the next higher adjacent bit of the binary number.

The conversion process is further illustrated with the help of an example showing step-by-step conversion of binary code 1011 into its Gray code equivalent:

Gray code 1- - - Binary 1011

Gray code 11- - Binary 1011

Gray code 111- Binary 1011

Gray code 1110

Basic logic operations NOT, OR, AND*:*

Binary logic consists of binary variables and logic operations. Each binary variable consists of two states called logic '0' and logic '1'. There are 3 basic logical operations: AND,OR,NOT and derived operations are NAND, NOR,X-OR, X-NOR.

AXIOMS:

Axioms or Postulates are a set of logical expressions without proof. Each axiom can be interpreted as the outcome of an operation performed by a logic gate.

AND(A.B=C)	OR(A+B=C)	NOT(A'=B)
0.0=0	0+0=0	1‟= 0
0.1=0	0+1=1	0‟ = 1
1.0=0	1+0=1	
1.1=1	1+1=1	

LOGIC GATES:

Logic gates are fundamental building blocks of digital systems. Logic gateproduces one output level when some combinations of input levels are present and a different output level when other combination of input levels is present. Based on the axioms there 3 basic types of logic gates were available which are indicated by AND, OR, NOT.

The interconnection of gates to perform a variety of logical operation is called *Logic Design*. Inputs & outputs of logic gates can occur only in two levels i.e., 1,0 or High, Low or True ,False or On , Off.

A table which lists all the possible combinations of input variables & the corresponding outputs is called a Truth Table. It shows how the logic circuits output responds to various combinations of logic levels at the inputs.

Level Logic, a logic in which the voltage levels represent logic 1 & logic 0.Level logic may be Positive Logic or Negative Logic.

In *Positive Logic* the higher of two voltage levels represent logic 1 & Lower of two voltage levels represent logic 0.In *Negative Logic* the lower of two voltage levels represent logic 1 & higher of two voltage levels represent logic 0.

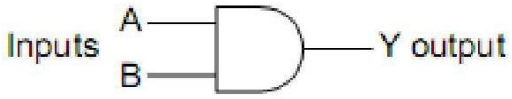
Ex:

In TTL (Transistor-Transistor Logic) Logic family voltage levels are +5V and 0V.Logic 1 represent +5Vand Logic 0 represent 0V.

AND Gate:

It is represented by "."(dot) It has two or more inputs but only one output. The output assume the logic 1 state only when each one of its inputs is at logic 1 state. The output assumes the logic 0 state even if one of its inputs is at logic 0 state. The AND gate is also called an All or Nothing gate.

Boolean Expression: A AND B, Y=A.B

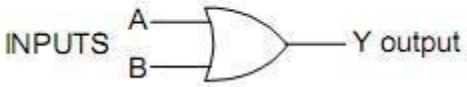


A	B	Y
0	0	0
0	1	0
1	0	0
1	1	1

Logic Symbol Truth Table

OR Gate:

It is represented by "+"(plus). It has two or more inputs but only one output. The output assumes the logic 1 state only when one of its inputs is at logic 1 state. The output assumes the logic 0 state even if each one of its inputs is at logic 0 state. The OR gate is also called an any or All gate. Also called an inclusive OR gate because it includes the condition both the inputs can be present.



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

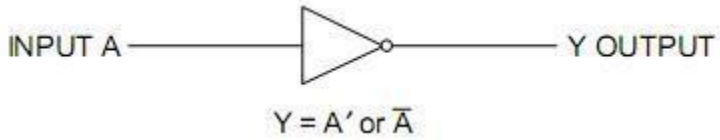
Logic Symbol Truth Table

Boolean Expression: A OR B, A+B=Y

NOT Gate:

It is represented by "-"(bar).It is also called an *Inverter or Buffer*. It has only one input and one output. Whose output always the compliment of its input. The output assumes logic 1 when input is logic 0 & output assume logic 0 when input is logic 1.

Logic Symbol



Truth table:

A	X
1	0
0	1

- Logic circuits of any complexity can be realized using only AND, OR , NOT gates. Using these 3 called AND-OR-INVERT i.e, AOI Logic circuits.

The Universal Gates:

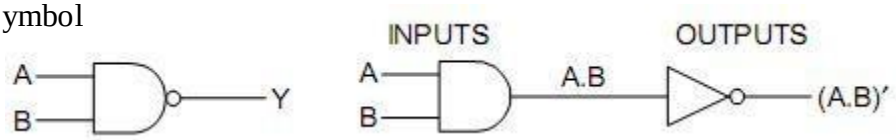
The universal gates are NAND, NOR. These gates are called universal gates because any Boolean logic function including basic operations(AND, OR, INVERT) can be implemented using NAND and NOR gates. More over AOI logic can be easily converted to NAND logic or NOR logic.

NAND Gate: It is combination of AND gate followed by NOT gate

Boolean Expression: $Y = (\ .B)$'

NAND assumes Logic 0 when each of inputs assumes logic 1.

Logic Symbol



Truth table

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

Bubbled OR gate: The output of this is same as NAND gate.

Bubbled OR gate is OR gate with inverted inputs.

$$Y = A' + B' = (AB)$$

NAND gate as an Inverter:

All its input terminals together & applying the signal to be inverted to the common terminal by connecting all input terminals except one to logic 1 & applying the signal to be inverted to the remaining terminal. It is also called Controlled Inverter.

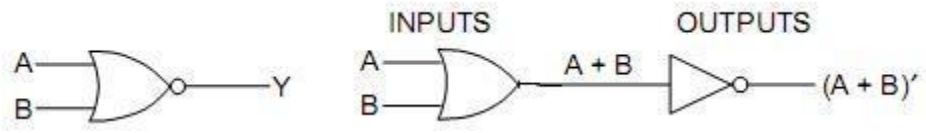
Bubbled NAND Gate: The output of bubbled NAND gate is same as OR gate

NOR Gate:

NOR gate is NOT gate with OR gate. i.e, OR gate is NOTed.

Boolean expression:$= (A + B)$'

Logic Symbol Logic symbol with OR and NOT



Truth Table:

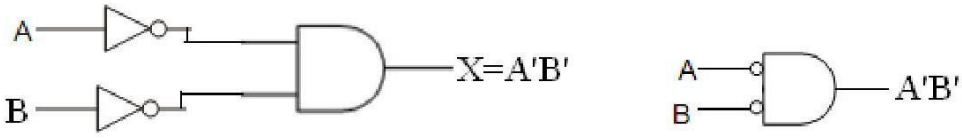
A	B	Y
0	0	1
0	1	0
1	0	0
1	1	0

Bubbled AND gate:

It is AND gate with inverted inputs. The AND gate with inverted inputs is called a bubbled AND gate. So a NOR gate is equivalent to a bubbled and gate. A bubbled AND gate is also called a negative AND gate. Since its output assumes the HIGH state only when all its inputs are in LOW state, a NOR gate is also called active-LOW AND gate.

Output Y is 1 only when both A & B are equal to 0.i.e, only when both A,, and B,, are equal to 1.NOR can also realized by first inverting the inputs and performing AND operation those inverted inputs.

Logic Symbol

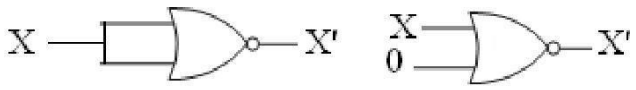


Truth table:

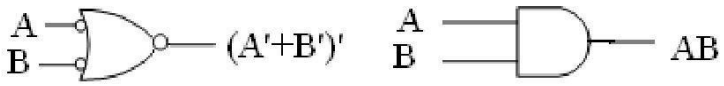
Inputs A B	Inverted Inputs A,, B,,	Output Y
0 0	1 1	1
0 1	1 0	0
1 0	0 1	0
1 1	0 0	0

NOR gate as an inverter:

is tying all input terminals together & applying the signal to be inverted to the common terminals or all inputs set as logic 0 except one & applying signal to be inverted to the remaining terminal.



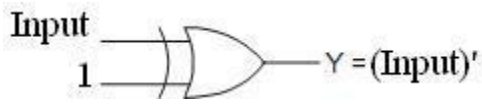
Neither bubbled NOR Gate: is AND gate.



The Exclusive OR (X-OR) gate:

It has 2 inputs& only 1 output. It assumes output as 1 when input is not equal called anti-coincidence gate or inequality detector.

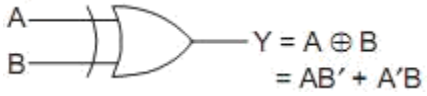
Logic Symbol



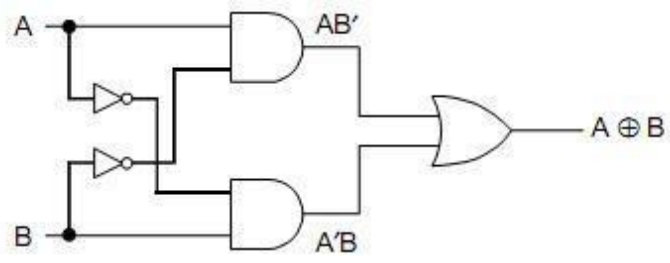
Truth table:

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

The high outputs are generated only when odd number of high inputs is present. This is why x-or function also known as odd *function.*

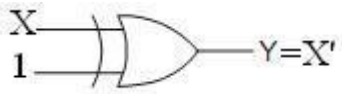


The X-OR gate using AND-OR-NOT gates:



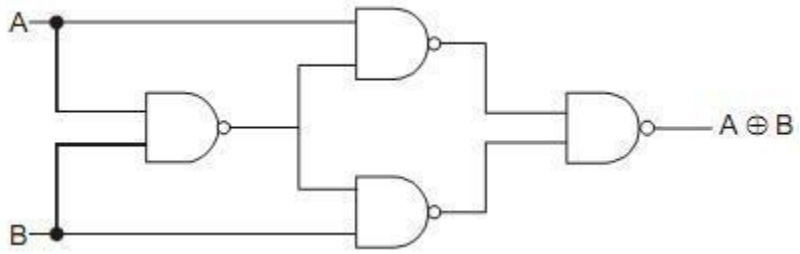
X-OR gate as an Inverter:

By connecting one of two input terminals to logic 1 & feeding the sequence to be inverted to other terminal

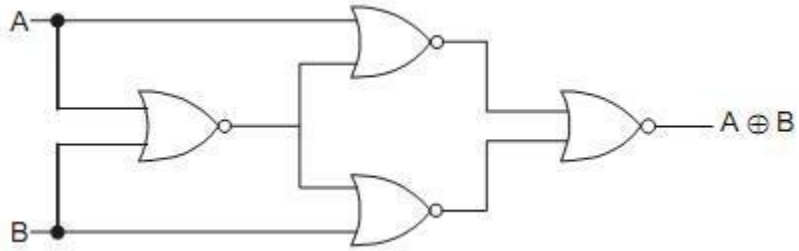


Logic Symbol

X-OR gate using NAND gates only:



X-OR gate using NOR gates only:



The EX-NOR Gate:

It is X-OR gate with a NOT gate. It has two inputs & one output logic circuit. It assumes output as 0 when one if inputs are 0 and other 1.It can be used as an equality detector because it outputs a 1 only when its inputs are equal.

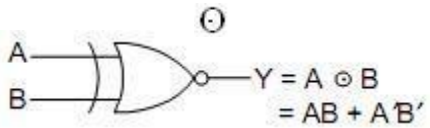
Proof: A Θ B = (A⊕B)'

= (AB'+A'B)'

= (A'+B).(A+B')

= AA'+A'B'+AB+BB'

= AB+A'B'



Logic Symbol.

Truth table:	
Inputs A B	Output X= A B
0 0	1
0 1	0
1 0	0
1 0	1

Standard SOP and POS forms

Reducing Boolean Expressions:

Procedure:

1. Multiply all variables necessary to remove parenthesis
2. Look for identical terms. Only one of those terms to be retained & other dropped.

   Ex: AB+AB+AB+AB=AB

3. Look for a variable & its negation in the same term. This term can be

   dropped 1 Ex: AB +AB = AB (+1)=AB .1=AB

4. Look for pairs of terms which have the same variables, with one or more variables complemented. If a variable in one term of such a pair is complemented while in the second term it is not then such terms can be combined into a single term with variable dropped.

   Ex: AB +AB D= AB (+D)=AB .1=AB

Boolean functions & their representation:

A function of n Boolean variables denoted by $f(x_1,x_2,x_3 \text{----- } x_n)$ is another variable denoted by & takes one of the two possible values 0 & 1. The various ways of representing the given function is

1. Sum of Product(SOP) form: It is called the Disjunctive Normal Form(DNF) Ex: f(A,B,C)= A.B'+ C'

2. Product of Sums (POS) form:It is called the Conjunctive Normal Form(CNF).This is implemented using Consensus theorem.
   Ex:f(A,B,C)=(A+B)(B+C)

3. Truth Table form:The function is specified by listing all possible combinations of values assumed by the variables & the corresponding values of the function.
   Ex: Truth table for f(A,B,C) = (B+ C)

Decimal Code	A	B	C	F(A,B,C)
0	0	0	0	0
1	0	0	1	1
2	0	1	0	1
3	0	1	1	1
4	1	0	0	0
5	1	0	1	1
6	1	1	0	0
7	1	1	1	0

4. Standard Sum of Products form called Disjunctive Canonical form (DCF) & also called Expanded SOP form or Canonical SOP form.
   Ex: f(A,B,C)= A.B.C' +A.B.C

A product term contains all the variables of the function either in complemented or uncomplemented form is called a minterm. A minterm assumes the value 1 only for one combination of the variables. An n variable function can have in all 2^n minterms to 1 is the

standard sum of products form of the function. Minterms are denoted as m_0, m_1,m_2 .....Here suffixes are denoted by the decimal codes.

Ex: F(A,B,C)=m_1+m_2+m_3 then m_1=A‟B‟C , m_2=AB‟C, m_3=A‟BC

The function in DCF is listing the decimal codes of the minterms for

which F=1 F(A,B,C)=∑m(1,2,3,).

5. Standard Product of Sums form: It is called as Conjunctive Canonical form (CCF). It is also called Expanded POS or Canonical POS.

Ex: If A=0,B=0, C=0 and the term=0
Thus function f (A, B, C) = (A'+B'+C').(A+B'+C').(A+B+C')

A sum term which contains each of the n variables in either complemented form is called a *Maxterm*. A maxterm assumes the value „0„only for one combination of the variables. The most there are 2^n maxterms. It is represented as M_0,M_1,M_2 Here the suffixes are decimal codes.

The CCF of f(A,B,C)=M_0.M_4.M_6

f(A,B,C)=πM(0,4,6,7) where π or ^ represents the product of all maxterms.

Expansion of a Boolean expression in SOP form to the standard SOP form:

1. Write down all the terms.
2. If one or more variables are missing in any term. Expand that term by multiplying it with the sum of each one of the missing variable and its complement.
3. Drop out redundant terms.

Expansion of a Boolean expression in POS form to standard POS form:

1. Write down all the terms.
2. If one or more variables are missing in any sum term. Expand that term by adding the product of each of the missing variable and its complement.
3. Drop out redundant terms.

Conversion between Canonical forms:
The complement of a function expressed as the sum of minterms equals the sum of minterms missing from the original function is expressed by those minterms that make the function equal to 1 for those minterms that make the function equal to 0.

Ex: f(A,B,C)=∑m(0,2,4,6,7)
Complement is f‟(A,B ,C) =∑m(1,3,5).=m_1+m_3+m_5
Complement of deMorgan's theorem: f= (m1 + m3 + m5) then f‟= M1.M3.M5

1=M_j, the maxterm with subscript j is a complement of the minterm with the same subscript j and vice versa. To convert one canonical form to another, interchange the symbol $\sum$ and π, and list those numbers missing from the original form.

Gray Code

- It is the non-weighted code and it is not arithmetic codes. That means there are no specific weights assigned to the bit position.
- It has a very special feature that, only one bit will change each time the decimal number is incremented as shown in fig. As only one bit changes at a time, the gray code is called as a unit distance code. The gray code is a cyclic code. Gray code cannot be used for arithmetic operation.

Decimal	BCD	Gray
0	0 0 0 0	0 0 0 0
1	0 0 0 1	0 0 0 1
2	0 0 1 0	0 0 1 1
3	0 0 1 1	0 0 1 0
4	0 1 0 0	0 1 1 0
5	0 1 0 1	0 1 1 1
6	0 1 1 0	0 1 0 1
7	0 1 1 1	0 1 0 0
8	1 0 0 0	1 1 0 0
9	1 0 0 1	1 1 0 1

Application of Gray code

- Gray code is popularly used in the shaft position encoders.
- A shaft position encoder produces a code word which represents the angular position of the shaft.

Binary–Gray Code Conversion A given binary number can be converted into its Gray code equivalent by going through the following steps:

- Begin with the most significant bit (MSB) of the binary number. The MSB of the Gray code equivalent is the same as the MSB of the given binary number.
- The second most significant bit, adjacent to the MSB, in the Gray code number is obtained by adding the MSB and the second MSB of the binary number and ignoring the carry, if any. That is, if the MSB and the bit adjacent to it are both '1', then the corresponding Gray code bit would be a '0'.

- The third most significant bit, adjacent to the second MSB, in the Gray code number is obtained by adding the second MSB and the third MSB in the binary number and ignoring the carry, if any.
- The process continues until we obtain the LSB of the Gray code number by the addition of the LSB and the next higher adjacent bit of the binary number.

The conversion process is further illustrated with the help of an example showing step-by-step conversion of binary code 1011 into its Gray code equivalent:

Gray code 1- - - Binary 1011

Gray code 11- - Binary 1011

Gray code 111- Binary 1011

Gray code 1110

Error detection and correction codes

- Error detection is the detection of errors caused by noise or other impairments during transmission from the transmitter to the receiver.
- Error correction is the detection of errors and reconstruction of the original, error-free data.

What is Error?

Error is a condition when the output information does not match with the input information. During transmission, digital signals suffer from noise that can introduce errors in the Binary bits travelling from one system to other. That means a 0 bit may change to 1 or a 1 bit may change to 0.

Error-Detecting codes

Whenever a message is transmitted, it may get scrambled by noise or data may get corrupted. To avoid this, we use error-detecting codes which are additional data added to a given digital message to help us detect if an error occurred during transmission of the message. A simple example of error-detecting code is parity check.

Error-Correcting codes

Along with error-detecting code, we can also pass some data to figure out the original message from the corrupt message that we received. This type of code is called an error-correcting code. Error-correcting codes also deploy the same strategy as error-detecting codes but additionally, such codes also detect the exact location of the corrupt bit.

In error-correcting codes, parity check has a simple way to detect errors along with a sophisticated mechanism to determine the corrupt bit location. Once the corrupt bit is located, its value is reverted (from 0 to 1 or 1 to 0) to get the original message.

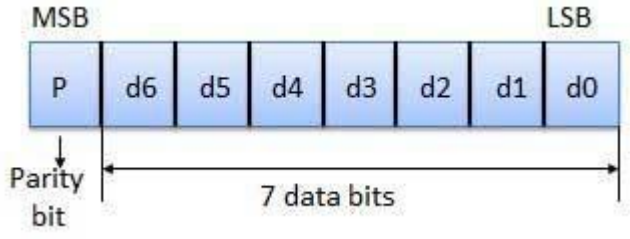
How to Detect and Correct Errors?

To detect and correct the errors, additional bits are added to the data bits at the time of transmission.

- The additional bits are called parity bits. They allow detection or correction of the errors.
- The data bits along with the parity bits form a code word.

Parity Checking of Error Detection

- It is the simplest technique for detecting and correcting errors. The MSB of an 8-bits word is used as the parity bit and the remaining 7 bits are used as data or message bits. The parity of 8-bits transmitted word can be either even parity or odd parity.



- Even parity -- Even parity means the number of 1's in the given word including the parity bit should be even (2,4,6, ...).
- Odd parity -- Odd parity means the number of 1's in the given word including the parity bit should be odd (1,3,5,...).
- Use of Parity Bit
- The parity bit can be set to 0 and 1 depending on the type of the parity required.
- For even parity, this bit is set to 1 or 0 such that the no. of "1 bits" in the entire word is even. Shown in fig. (a).

- For odd parity, this bit is set to 1 or 0 such that the no. of "1 bits" in the entire word is odd. Shown in fig. (b).

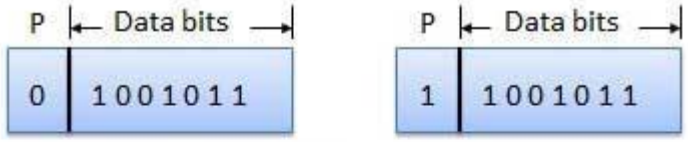


Fig. (a)

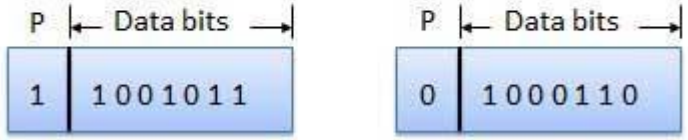
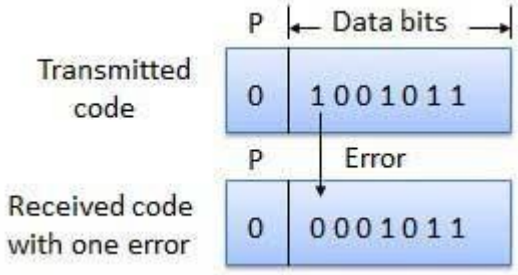


Fig. (b)

How Does Error Detection Take Place?

Parity checking at the receiver can detect the presence of an error if the parity of the receiver signal is different from the expected parity. That means, if it is known that the parity of the transmitted signal is always going to be "even" and if the received signal has an odd parity, then the receiver can conclude that the received signal is not correct. If an error is detected, then the receiver will ignore the received byte and request for retransmission of the same byte to the transmitter.



Error Detecting Codes

Basic approach used for error detection is the use of redundancy, where additional bits are added to facilitate detection and correction of errors. Popular techniques are:

• Simple Parity check

• Two-dimensional Parity check

• Checksum

• Cyclic redundancy check

Error Correcting Codes

The techniques that we have discussed so far can detect errors, but do not correct them. Error Correction can be handled in two ways.

One is when an error is discovered; the receiver can have the sender retransmit the entire data unit. This is known as backward error correction.

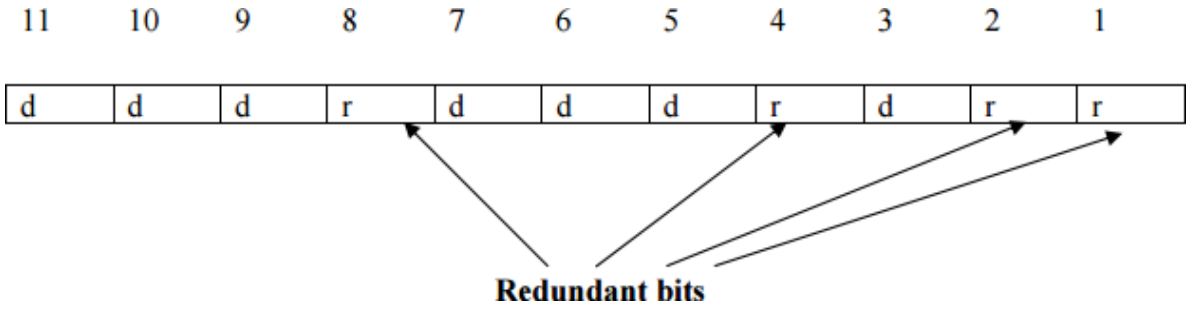
In the other, receiver can use an error-correcting code, which automatically corrects certain errors. This is known as forward error correction.

Single-bit error correction (Hamming Code)

For example, to correct a single-bit error in an ASCII character, the error correction must determine which one of the seven bits is in error. To this, we have to add some additional redundant bits. To calculate the numbers of redundant bits (r) required to correct d data bits, let us find out the relationship between the two. So we have (d+r) as the total number of bits, which are to be transmitted; then r must be able to indicate at least d+r+1 different value. Of these, one value means no error, and remaining d+r values indicate error location of error in each of d+r locations. So, d+r+1 states must be distinguishable by r bits, and r bits can indicates 2r states. Hence, 2r must be greater than d+r+1.

$$2r >= d+r+1$$

The value of r must be determined by putting in the value of d in the relation. For example, if d is 7, then the smallest value of r that satisfies the above relation is 4. So the total bits, which are to be transmitted is 11 bits (d+r = 7+4 =11). Now let us examine how we can manipulate these bits to discover which bit is in error. A technique developed by R.W. Hamming provides a practical solution. The solution or coding scheme he developed is commonly known as Hamming Code. Hamming code can be applied to data units of any length and uses the relationship between the data bits and redundant bits as discussed.



Positions of redundancy bits in hamming code

Basic approach for error detection by using Hamming code is as follows:

- To each group of m information bits k parity bits are added to form (m+k) bit code as shown in the figure above.
- Location of each of the (m+k) digits is assigned a decimal value.
- The k parity bits are placed in positions 1, 2, …, 2k-1 positions. –K parity checks are performed on selected digits of each codeword.
- At the receiving end the parity bits are recalculated. The decimal value of the k parity bits provides the bit-position in error, if any.

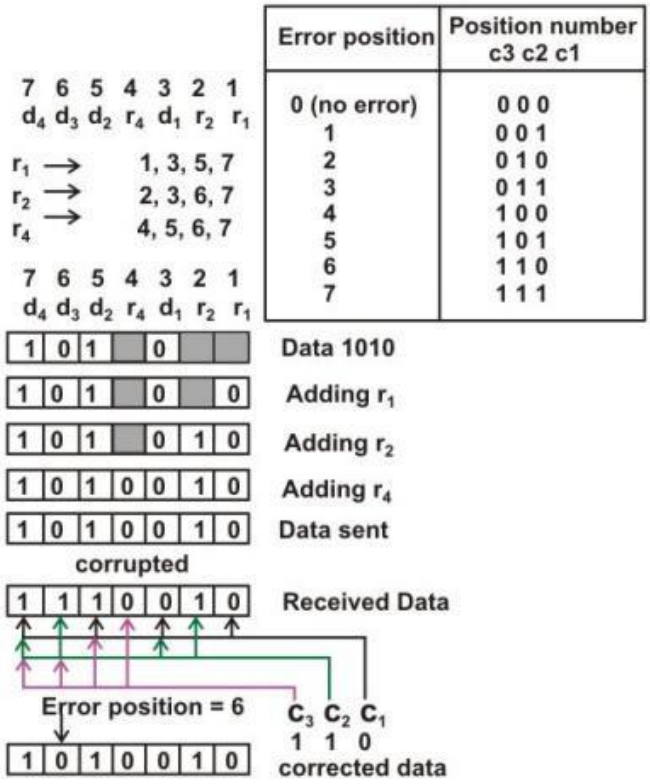


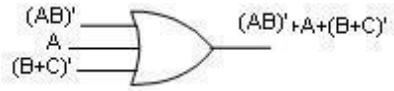
Figure: Use of hamming code for error correction for a 4-bit data

The figure above shows how hamming code is used for correction for 4-bit numbers (d4d3d2d1) with the help of three redundant bits (r3 r2 r1). For the example data 1010, first r1 (0) is calculated considering the parity of the bit positions, 1, 3, 5 and 7. Then the parity bits r2 is calculated considering bit positions 2, 3, 6 and 7. Finally, the parity bits r4 is calculated considering bit positions 4, 5, 6 and 7 as shown. If any corruption occurs in any of the transmitted code 1010010, the bit position in error can be found out by calculating r3 r2 r1 at the receiving end. For example, if the received code word is 1110010, the recalculated value of r3 r2 r1 is 110, which indicates that bit position in error is 6, the decimal value of 110.

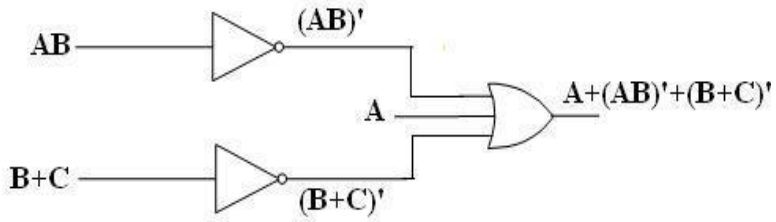
Two Level NAND – NAND and NOR-NOR realizations:

Boolean expressions can be realized as hardware using logic gates. Conversely, hardware can be translated into Boolean expressions for the analysis of existing circuits.

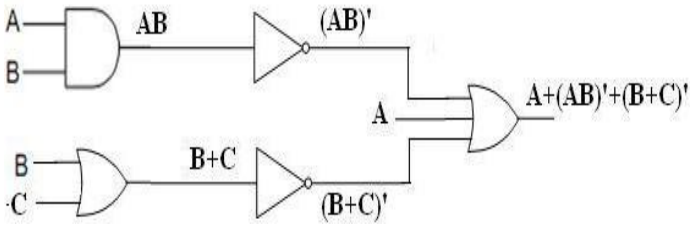
1. *Converting Boolean Expressions to Logic:* To convert, start with the output & work towards the input. Assume the expression (AB)'+A+(B+C)' is to be realized using AOI logic. Start with this expression. Since it is three terms, it must be the output of a three-input OR gates. So, draw an OR gate with three inputs as



(AB)' is the output of an inverter whose inputs is AB and (B+C)' must be the output of an inverter whose input is B+C. so, those two inverters are as



Now AB must be output of a two-input AND gate whose inputs are A and B. And B+C must be the output of a two-input OR gate whose inputs are B and C. so, an AND gate and an OR gate are as



2. *Converting Logic to Boolean Expressions:*

To convert logic to algebra, start with the input signals and develop the terms of the Boolean expression until the output is reached.

Since NAND logic and NOR logic are universal logic circuits which are first computed and converted to AOI logic may then be converted to either NAND logic or NOR logic depending on the choice. The procedure is

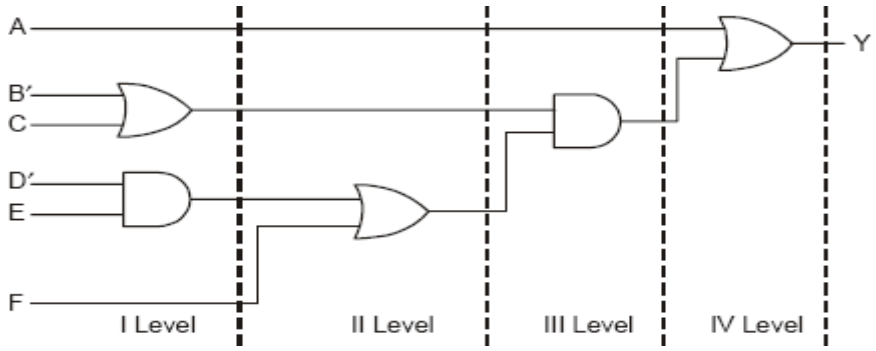
1. Draw the circuit in AOI logic
2. If NAND hardware is chosen, add a circle at the output of each AND gate and at the inputs to all the AND gates.
3. If NOR hardware is chosen, add a circle at the output of each OR gate and at the inputs to all the AND gates

4. Add or subtract an inverter on each line that received a circle in steps 2 or 3 so that the polarity of signals on those lines remains unchanged from that of the original diagram
5. Replace bubbled OR by NAND and bubbled AND by NOR Eliminate double inversions.

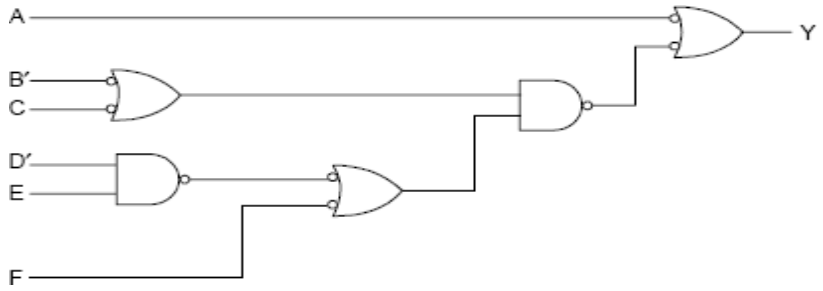
Ex: Now consider a Boolean function to demonstrate the procedure for converting into NAND gates:

$$Y = A + (B' + C)(D'E + F)$$

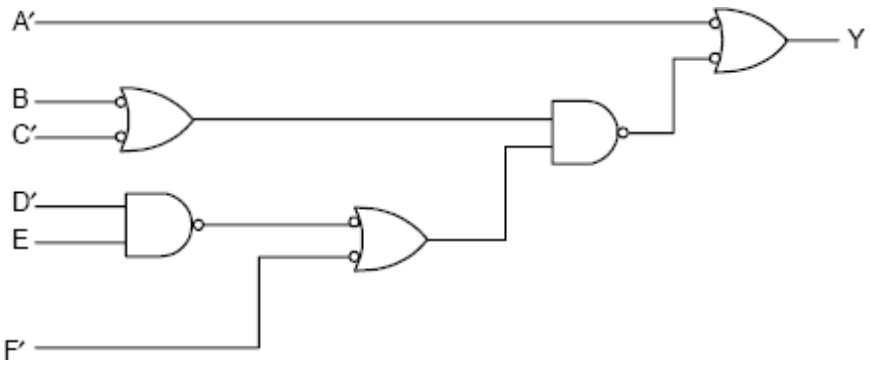
Step 1:We first draw the logic diagram using basic gates as shown in figure before



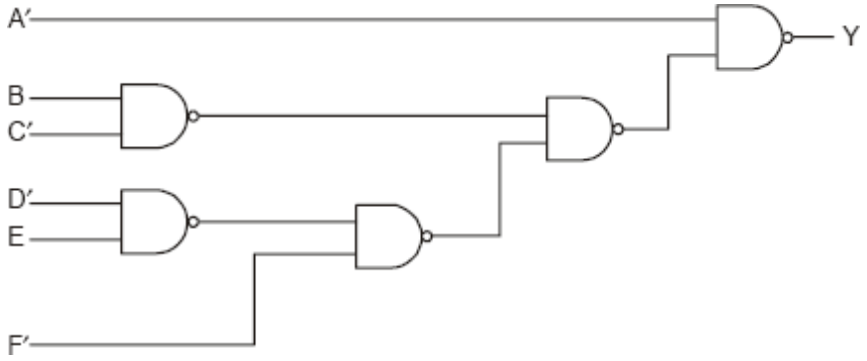
Step 2 and 3: Convert all AND gates to NAND using AND-invert symbol and all OR gates to NAND using Invert-OR symbol.



Step 4: It is very clear that only two inputs D′ and E are emerging in the original forms at the output. Rest inputs A, B′, C and F are emerging as the complement of their original form.

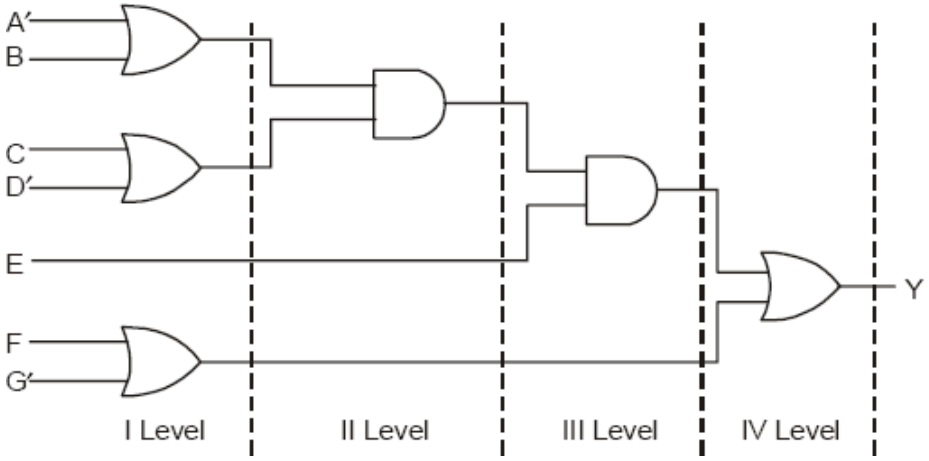


Step 5: Now because both the symbols AND-invert and invert-OR represent a NAND gate.

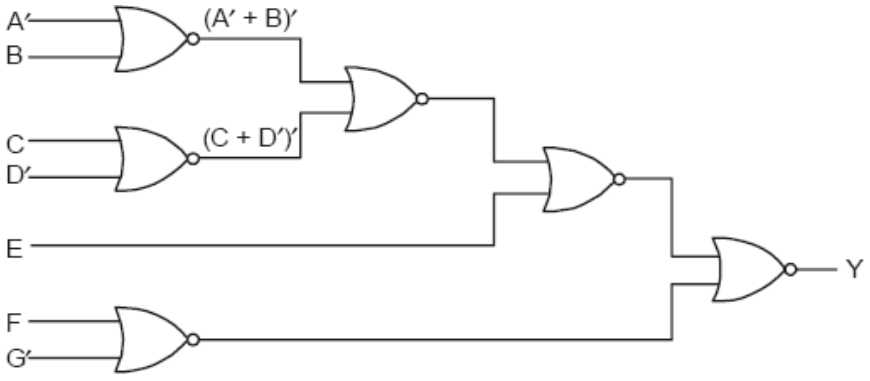


Ex:

Now consider a Boolean function to for converting into NOR gates: Y = ((A+B).(C+D))E+(F+G')



Convert all OR gates to NOR using OR-invert and all AND gates to NOR using invert AND symbol. Convert both symbols OR-invert and invent-AND represent a NOR gate



Minimization of logic functions using Boolean theorems

The keys to Boolean function minimization lie in the theorems introduced for Boolean algebra. Particularly the theorems shown below are useful.

(*a*) A + AB = A (*b*) A (A + B) = A

(*c*) A + A′B = A + B (*d*) A (A′ + B) = AB

(e) AB + AB′ = A (*f*) (A + B) (A + B′) = A

Ex: Minimize F =

CD + AB′C + ABC′ + BCD

A+AB=A

F = CD + AB′C + ABC′

Assignment-Cum-Tutorial Questions

A. Questions testing the remembering / understanding level of students

I) Objective Questions

1) The_____or_____of a number system indicates the number of unique symbols used in that system.

2) The highest decimal number that can be represented with 10 binary digits is
(A) 512 (B) 1023 (C) 1024 (D) 2^{11}-1

3) Let $(A2C)_{16} = (X)_8$. Then X is
(A) 7054 (B) 6054 (C) 5154 (D) 5054

4) The eight bit 2's complement form of $(-23)_{10}$ is_____.

5) 4-bit 2's complement representation of a decimal number is 1000. The number is
(A) +8 (B) 0 (C) -7 (D) -8

6) The 2's complement representation of -17 is
(A) 101110 (B) 101111 (C) 111110 (D) 110001

7) How many bits are in an ASCII character?
(A) 16 (B) 8 (C) 7 (D) 14

8) A binary number's value changes most drastically when the _____is changed.

9) Decimal 11 in BCD is_____.
(A) 00001011 (B) 00001100 (C) 00010001 (D) 00010010

10) The two types of parity are_____and_____.

11) For a code to be an error detecting code, the minimum hamming distance between two code words must be_____.

12) The parity of the binary number 11011001 is
(A) Even (B) odd (C) same as the number of zeros (D) none

II) Descriptive Questions

1) Explain the classification of different number systems.
2) Describe the conversion of decimal number to hexadecimal number with an example.
3) Convert the following decimal numbers to octal numbers
(a) 4796 (b) 8957.75

4) Explain the process of decimal number subtraction using 9's complement and 10's complement with an example.

5) Discuss by an example, the operation of binary number subtraction using the method of 1's and 2's complement.

6) Describe how signed number is represented in two's complement form.

7) Write the equivalent $(743)_{10}$ in BCD, 2421 and 6421 codes.

8) What are the different types of non weighted codes and explain them with examples.

9) Explain how errors are detected using
i) Parity ii) Check sums iii) Block parity iv) five bit codes

10) Write notes on error correcting codes

11) What is hamming code? How is the hamming code word tested and corrected?

B. Question testing the ability of students in applying the concepts.

I) Objective Questions

1) The $(128)_{10} = (1003)_b$, the possible base b is

(A) 3 (B) 4 (C) 5 (D) 6

2) Decimal 43 in Hexadecimal and BCD number system is respectively

(A) B2, 0100 011 (B) 2B, 0100 0011 (C) 2B, 0011 0100 (D) B2, 0100 0100

3) An equivalent 2's complement representation of the 2's complement number 1101 is

(A) 110100 (B) 01101 (C) 110111 (D) 111101

4) 11001, 1001, 111001 correspond to the 2's complement representation of which one of the following sets of number

(A) 25, 9, and 57 (B) -6, -6, and -6 (C) -7, -7 and -7 (D) -25, -9 and -57

5) Consider the signed binary number A = 01000110 and B = 11010011, where B is in 2's complement and MSB is the sign bit. In list-I operation is given and is List-II resultant binary number is given in the table below:

List - I	List - II
P. $A+B$	1. 1 0 0 0 1 1 0 1 2. 1 1 1 0 0 1 1 1
Q. $A-B$	3. 0 1 1 1 0 0 1 1 4. 1 0 0 0 1 1 1 0
R. $B-A$	5. 0 0 0 1 1 0 1 0 6. 0 0 0 1 1 0 0 1
S. $-A-B$	7. 0 0 0 1 1 0 0 1 8. 0 1 0 1 1 0 1 1

The correct match of P Q R S is

(A) 5 7 4 2 (B) 6 3 1 2 (C) 6 7 1 3 (D) 5 3 4 2

6) The range of signed decimal numbers that can be represented by 6-bit 1's complement number is

(A) -31 to +31 (B) -63 to +63 (C) -64 to +63 (D) -32 to +31

7) Which of the following is an invalid BCD code?

(A) 0011 (B) 1101 (C) 0101 (D) 1001

8) Convert the 127decimal number to BCD.
(A) 011100100001 (B) 111010001 (C) 001010111 (D) 000100100111

9) The binary-coded decimal (BCD) system can be used to represent each of the 10 decimal digits as an:

(A) 4-bit binary code (B) 8-bit binary code (C) 16-bit binary code (D) ASCII code

10) For 2-bit error detecting, the minimum hamming distance must be

(A) 1 (B) 2 (C) 3 (D) 4

11) The number of parity bits in a 12-bit hamming code is
(A) 4 (B) 5 (C) 6 (D) 8

II) Descriptive Questions

1) Convert the following numbers with the given radix to decimal.
i) $(334)_5$ ii) $(12345)_7$ iii) $(768)_9$

2) Solve
i) $(AD012)_{16} = (X)_5$ ii) $(5.204)_{10} = (X)_3$

3) Perform the following
a) $(137.64)_{10} = (\ )_6 = (\ )_2$
b) $(1111.1011)_2 = (\ )_8 = (\ )_{16}$

4) Subtract $(0001.1110)_2$ from $(0011.1001)_2$ using 2's complement method.

5) If A = -57 and B = +38, then represent A and B in 8-bit 2's complement.
Find (i) A + B (ii) A - B using 2's complement method.

6) Perform the following operations using r-1's complement arithmetic:
i) $(+43)_{10} - (-53)_{10}$. ii) $(+346.56)_{10} - (+456.78)_{10}$.

7) Convert (1101) binary code to gray code and reflective code with detailed steps?

8) Perform the following subtraction in XS-3 code using 10's complement method.
$(597)_{10} - (239)_{10}$

9) Represent the unsigned decimal numbers 351 and 986 in BCD, and then show the steps

necessary to form their sum.

10) Generate Hamming code for the given 11 bit message 10001110101 and rewrite the entire message with Hamming code.

11) Given the 8-bit data word 01011011, generate the 12 bit composite word for the hamming code that corrects and detects single errors.

C. Questions testing the analyzing / evaluating ability of students

1) Assume an arbitrary number system having a radix of 5 and 0, 1, 2, L and M as its independent digits. Determine:
   i) The decimal equivalent of (2LM.L1)
   ii) The octal equivalent of (21L.M2)
   iii) The hexagonal equivalent of (LM1.L2)

2) Test the following hamming code sequence for 11 bit message and correct if necessary (100, 1101, 1110, 1011)

3) Generate the weighted codes for the decimal digits using the weights
   (a) 3, 3, 2, 1 (b) 4, 4, 3, -2

4) Represent numeric digits 0 to 9 atleast in any two self complementing codes

D. GATE/IES Questions

1) The number of bytes required to represent the decimal number 1856357 in packed BCD (Binary Coded Decimal) form is_____. **GATE-2014**

2) The two numbers represented in signed 2's complement form are P = 11101101 and Q = 11100110. If Q is subtracted from P, the value obtained in signed 2's complement form is
A. 100000111 B. 00000111 C. 11111001 D. 111111001 **GATE-2008**

3) X = 01110 and Y = 11001 are two 5 bit binary numbers represented in 2's complement format. The sum of X and Y represented in 2's compliment format using 6 bits is
A. 100111 B. 001000 C. 000111 D. 101001 **GATE-2007**

4) A new Binary Coded Pentary (BCP) number system is proposed in which every digit of a base-5 number is represented by its corresponding 3-bit binary code. For example, the base-5 number 24 will be represented by its BCP code 010100. In this number system, the BCP code 100010011001 corresponds to the following number in base-5 system
A. 423 B. 1324 C. 2201 D. 4231 **GATE-2006**

5) Decimal 43 in Hexadecimal and BCD number system is respectively **GATE-2005**
A. B2, 01000011 B. 2B, 01000011
C. 2B, 00110100 D. B2, 00110100

6) 11001, 1001 and 111001 correspond to the 2's complement representation of which one of the following sets of number? **GATE-2004**

A. 25, 9 and 57 respectively B. -6, -6 and -6 respectively
C. -7, -7 and -7 respectively D. -25, -9 and -57 respectively

7) The range of .signed decimal numbers that can be represented by 6 bit 1's complement form is
A. -31 to +31 B. -63 to +64 C. -64 to +63 D. -32 to+31 **GATE-2004**

8) 4-bit 2's complement representation of a decimal number is 1000. The number is
A. +8 B. 0 C. -7 D. -8 **GATE-2002**

9) The 2's complement representation of- 17 is **GATE-2001**
A. 01110 B. 101111 C.11110 D. 10001

10) An equivalent 2's complement representation of the 2's complement number is 1101 is
A. 110100 B. 001101 C. 110111 D. 111101 **GATE-1998**

11) 2's complement representation of a 16 bit number (one sign bit and 15 magnitude bits) is FFFF, Its magnitude in decimal representation is **GATE-1993**
A. 0 B. 1 C.32,767 D.65,535

12) The subtraction of a binary number Y from another binary number X, done by adding 2's compliment of Y to X, results in a binary number without overflow. This implies that the result is

A. Negative and is in normal form **GATE-1987**
B. Negative an is in 2's compliment form
C. Positive and is in normal form
D. Positive and is in 2's compliment form

13) The BDC code for a decimal number 874 is: **IES-2013**
A. $(100001110100)_{BCD}$ B. $(010001111000)_{BCD}$
C. $(100001000111)_{BCD}$ D. $(011110000100)_{BCD}$

14) The decimal equivalent of binary number 10110.11 is: **IES-2013**
A. 16.75 B. 20.75 C. 16.50 D. 22.75

15) A seven -bit Hamming code is received as 1111101. What is the correct code? **IES-2013**
A. 1101111 B. 1011111 C. 1111111 D. 1111011

16) Hexadecimal conversion of decimal number 227 will be: **IES-2013**
A. A3 B. E3 C. CC D. C3