Number System

When we type some letters or words, the computer translates them in numbers as computers can understand only numbers.

A computer can understand positional number system where there are only a few symbols called digits and these symbols represent different values depending on the position they occupy in the number.

A value of each digit in a number can be determined using:

- The digit
- The position of the digit in the number
- The base of the number system (where base is defined as the total number of digits available in the number system).

Decimal Number System

The number system that we use in our day-to-day life is the decimal number system. Decimal number system has base 10 as it uses 10 digits from 0 to 9. In decimal number system, the successive positions to the left of the decimal point represent units, tens, hundreds, thousands and so on.

Each position represents a specific power of the base (10). For example, the decimal number 1234 consists of the digit 4 in the units position, 3 in the tens position, 2 in the hundreds position, and 1 in the thousands position, and its value can be written as

(1x1000)+ (2x100)+ (3x10)+ (4xl)

$(1x10^3)+ (2x10^2)+ (3x10^1)+ (4xl0^0)$

1000 + 200 + 30 + 4

1234

As a computer programmer or an IT professional, you should understand the following number systems, which are frequently used in computers.

Binary Number System

- Uses two digits, 0 and 1.
- Also called base 2 number system.
- Each position represents a 0 power of the base (2). Example, 2^0.
- Last position represents a x power of the base (2). Example, 2^x where x represents the last position - 1.

EXAMPLE

Binary Number: 10101_2

Calculating Decimal Equivalent:

Step	Binary Number	Decimal Number
Step 1	10101_2	$((1 \times 2^4) + (0 \times 2^3) + (1 \times 2^2) + (0 \times 2^1) + (1 \times 2^0))_{10}$
Step 2	10101_2	$(16 + 0 + 4 + 0 + 1)_{10}$
Step 3	10101_2	21_{10}

Note: 10101_2 is normally written as 10101.

Octal Number System

- Uses eight digits: 0, 1, 2, 3, 4, 5, 6, 7.
- Also called base 8 number system.
- Each position in a octal number represents a 0 power of the base (8). Example, 8^0.
- Last position in a octal number represents a x power of the base (8). Example, 8^x where x represents the last position - 1.

EXAMPLE

Octal Number: 12570_8

Calculating Decimal Equivalent:

Step	Octal Number	Decimal Number
Step 1	12570_8	$((1 \times 8^4) + (2 \times 8^3) + (5 \times 8^2) + (7 \times 8^1) + (0 \times 8^0))_{10}$
Step 2	12570_8	$(4096 + 1024 + 320 + 56 + 0)_{10}$
Step 3	12570_8	5496_{10}

Note: 12570_8 is normally written as 12570.

Hexadecimal Number System

- Uses 10 digits and 6 letters: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F.
- Letters represent numbers starting from 10. A = 10. B = 11, C = 12, D = 13, E = 14, F = 15.
- Also called base 16 number system.
- Each position in a hexadecimal number represents a 0 power of the base (16). Example, 16^0.
- Last position in a hexadecimal number represents a x power of the base (16). Example, $16^x$where x represents the last position - 1.

EXAMPLE

Hexadecimal Number: $19FDE_{16}$

Calculating Decimal Equivalent:

Step	BINARY Number	Decimal Number
Step 1	$19FDE_{16}$	$((1 \times 16^4) + (9 \times 16^3) + (F \times 16^2) + (D \times 16^1) + (E \times 16^0))_{10}$
Step 2	$19FDE_{16}$	$((1 \times 16^4) + (9 \times 16^3) + (15 \times 16^2) + (13 \times 16^1) + (14 \times 16^0))_{10}$
Step 3	$19FDE_{16}$	$(65536 + 36864 + 3840 + 208 + 14)_{10}$
Step 4	$19FDE_{16}$	106462_{10}

Note: $19FDE_{16}$ is normally written as 19FDE.

Number System Conversion

There are many methods or techniques, which can be used to convert numbers from one base to another. Decimal to Other Base System

Decimal to Other Base System

Steps

- **Step 1** - Divide the decimal number to be converted by the value of the new base.
- **Step 2** - Get the remainder from Step 1 as the rightmost digit (least significant digit) of new base number.
- **Step 3** - Divide the quotient of the previous divide by the new base.

- **Step 4** - Record the remainder from Step 3 as the next digit (to the left) of the new base number.

  Repeat Steps 3 and 4, getting remainders from right to left, until the quotient becomes zero in Step 3.

  The last remainder thus obtained will be the most significant digit (MSD) of the new base number.

  EXAMPLE

  Decimal Number: 29_{10}

  Calculating Binary Equivalent:

Step	Operation	Result	Remainder
Step 1	29 / 2	14	1
Step 2	14 / 2	7	0
Step 3	7 / 2	3	1
Step 4	3 / 2	1	1
Step 5	1 / 2	0	1

  As mentioned in Steps 2 and 4, the remainders have to be arranged in the reverse order so that the first remainder becomes the least significant digit (LSD) and the last remainder becomes the most significant digit (MSD).

  Decimal Number: 29_{10} = Binary Number: 11101_2.

Other base system to Decimal System

Steps

- **Step 1** - Determine the column (positional) value of each digit (this depends on the position of the digit and the base of the number system).
- **Step 2** - Multiply the obtained column values (in Step 1) by the digits in the corresponding columns.
- **Step 3** - Sum the products calculated in Step 2. The total is the equivalent value in decimal.

  EXAMPLE

  Binary Number: 11101_2

  Calculating Decimal Equivalent:

Step	Binary Number	Decimal Number
Step 1	11101_2	$((1 \times 2^4) + (1 \times 2^3) + (1 \times 2^2) + (0 \times 2^1) + (1 \times 2^0))_{10}$
Step 2	11101_2	$(16 + 8 + 4 + 0 + 1)_{10}$
Step 3	11101_2	29_{10}

Binary Number: 11101_2 = Decimal Number: 29_{10}

Other Base System to Non-Decimal System

Steps

- **Step 1** - Convert the original number to a decimal number (base 10).
- **Step 2** - Convert the decimal number so obtained to the new base number.

EXAMPLE

Octal Number: 25_8

Calculating Binary Equivalent:

STEP 1: CONVERT TO DECIMAL

Step	Octal Number	Decimal Number
Step 1	25_8	$((2 \times 8^1) + (5 \times 8^0))_{10}$
Step 2	25_8	$(16 + 5)_{10}$
Step 3	25_8	21_{10}

Octal Number: 25_8 = Decimal Number: 21_{10}

STEP 2: CONVERT DECIMAL TO BINARY

Step	Operation	Result	Remainder
Step 1	21 / 2	10	1
Step 2	10 / 2	5	0

Step 3	5 / 2	2	1
Step 4	2 / 2	1	0
Step 5	1 / 2	0	1

Decimal Number: 21_{10} = Binary Number: 10101_2
Octal Number: 25_8 = Binary Number: 10101_2

Shortcut method - Binary to Octal

Steps

- **Step 1** - Divide the binary digits into groups of three (starting from the right).
- **Step 2** - Convert each group of three binary digits to one octal digit.

EXAMPLE

Binary Number: 10101_2

Calculating Octal Equivalent:

Step	**Binary Number**	**Octal Number**
Step 1	10101_2	010 101
Step 2	10101_2	2_8 5_8
Step 3	10101_2	25_8

Binary Number: 10101_2 = Octal Number: 25_8

Shortcut method - Octal to Binary

Steps

- **Step 1** - Convert each octal digit to a 3-digit Binary number (the octal digits may be treated as decimal for this conversion).
- **Step 2** - Combine all the resulting binary groups (of 3 digits each) into a single binary number.

EXAMPLE

Octal Number: 25_8

Calculating Binary Equivalent:

Step	Octal Number	Binary Number
Step 1	25_8	2_{10} 5_{10}
Step 2	25_8	010_2 101_2
Step 3	25_8	010101_2

Octal Number: 25_8 = Binary Number: 10101_2

Shortcut method - Binary to Hexadecimal

Steps

- **Step 1** - Divide the binary digits into groups of four (starting from the right).
- **Step 2** - Convert each group of four binary digits to one hexadecimal symbol.

EXAMPLE

Binary Number: 10101_2

Calculating hexadecimal Equivalent:

Step	Binary Number	Hexadecimal Number
Step 1	10101_2	0001 0101
Step 2	10101_2	1_{10} 5_{10}
Step 3	10101_2	15_{16}

Binary Number: 10101_2 = Hexadecimal Number: 15_{16}

Shortcut method - Hexadecimal to Binary

Steps

- **Step 1** - Convert each hexadecimal digit to a 4-digit Binary number (the hexadecimal digits may be treated as decimal for this conversion).
- **Step 2** - Combine all the resulting Binary groups (of 4 digits each) into a single binary number.

EXAMPLE

Hexadecimal Number: 15_{16}

Calculating Binary Equivalent:

Step	Hexadecimal Number	Binary Number
Step 1	15_{16}	$1_{10}\ 5_{10}$
Step 2	15_{16}	$0001_2\ 0101_2$
Step 3	15_{16}	00010101_2

Hexadecimal Number: 15_{16} = Binary Number: 10101_2

Binary Codes

In the coding, when numbers, letters or words are represented by a specific group of symbols, it is said that the number, letter or word is being encoded. The group of symbols is called as a code. 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 binary codes

The codes are broadly categorized into following four categories.

Weighted Codes

Non-Weighted Codes

Binary Coded Decimal Code

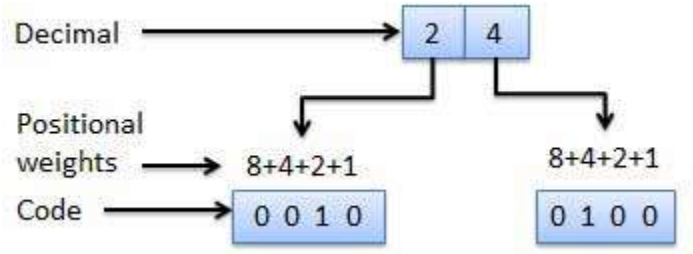
Alphanumeric Codes

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.

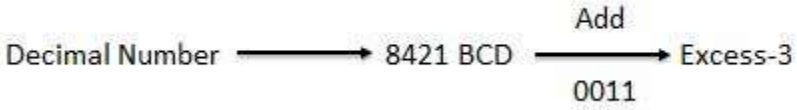


Non-Weighted Codes

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

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 has 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 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 Baudot 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.

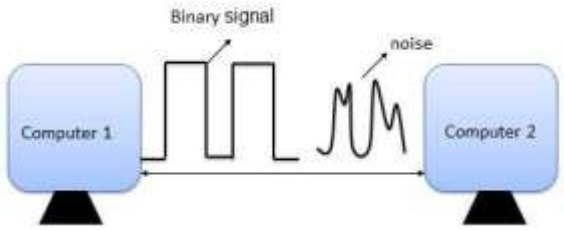
Error Codes

There are binary code techniques available to detect and correct data during data transmission.

Error Detection and Correction

What is Error?

Error means a condition when output information is not same as input information. When transmission of digital signals takes place between two systems such as a computer, the transmitted signal is combined with the "Noise". The noise can introduce an error in the binary bits travelling from one system to other. That means 0 may change to 1 or a 1 may change to 0.



Error Detecting codes

Whenever a message is transmitted then there are changes that it get scrambled by noise or data gets corrupted. When we add some additional data to a given digital message which can help us to detect if an error occurred during transmission of the message then adding such data is called an error-detecting code. A simple example of error-detecting code is parity check.

Error Correcting codes

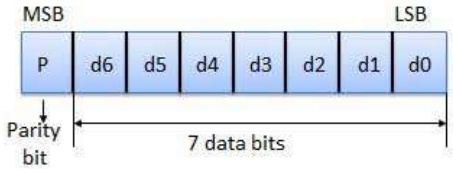
Along with Error detecting code, we can also pass some data which help to figure out the original message from the corrupt message that we received. This type of code is called an error-correcting code. Error correcting codes also deploys the same strategy as error detecting codes but additionally, such codes also detects the exact location of corrupt bit. In error correcting code, parity check has simple way to detect error along with a sophisticated mechanism to determine the corrupt bit location. Once 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?

- For the detection and correction of these errors, one or more than one extra bits are added to the data bits at the time transmitting.
- These extra bits are called as 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 a simplest technique for detecting and correcting error. In 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. Show in the 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. Show in the fig. (b).

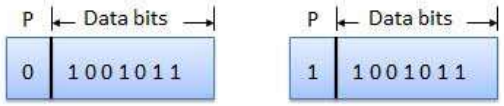


Fig. (a)

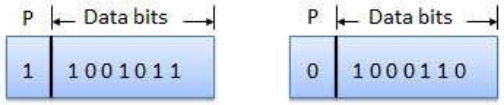
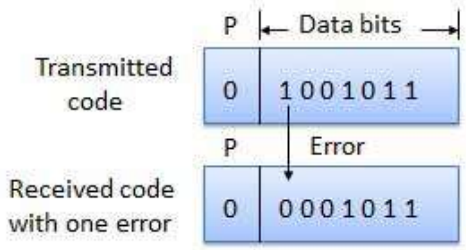


Fig. (b)

How does error detection take place?

The parity checking at the receiver can detect the presence of an error if the parity of the receiver signal is different from the expect parity. That means if 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 presence of error is detected then the receiver will ignore the received byte and request for retransmission of the same byte to the transmitter.



Codes Conversion

There are many methods or techniques which can be used to convert code from one format to another. We'll demonstrate here the following

- Binary to BCD Conversion
- BCD to Binary Conversion
- BCD to Excess-3
- Excess-3 to BCD

Binary to BCD Conversion

Steps

- Step 1 -- Convert the binary number to decimal.
- Step 2 -- Convert decimal number to BCD.

Example: convert (11101)2 to BCD.

Step 1 - Convert to Decimal

Binary Number: 111012

Calculating Decimal Equivalent:

Step	Binary Number	Decimal Number
Step 1	11101_2	$((1 \times 2^4) + (1 \times 2^3) + (1 \times 2^2) + (0 \times 2^1) + (1 \times 2^0))_{10}$
Step 2	11101_2	$(16 + 8 + 4 + 0 + 1)_{10}$
Step 3	11101_2	29_{10}

Binary Number: 111012 = Decimal Number: 2910

Step 2 - Convert to BCD

Decimal Number: 2910

Calculating BCD Equivalent. Convert each digit into groups of four binary digits equivalent.

Step	Decimal Number	Conversion
Step 1	29_{10}	$0010_2\ 1001_2$
Step 2	29_{10}	00101001_{BCD}

Result

$(11101)_2 = (00101001)_{BCD}$

BCD to Binary Conversion

Steps

- Step 1 -- Convert the BCD number to decimal.
- Step 2 -- Convert decimal to binary.

Example: convert (00101001)BCD to Binary.

Step 1 - Convert to BCD

BCD Number: (00101001)BCD

Calculating Decimal Equivalent. Convert each four digit into a group and get decimal equivalent or each group.

Step	BCD Number	Conversion
Step 1	$(00101001)_{BCD}$	$0010_2\ 1001_2$
Step 2	$(00101001)_{BCD}$	$2_{10}\ 9_{10}$
Step 3	$(00101001)_{BCD}$	29_{10}

BCD Number: (00101001)BCD = Decimal Number: 2910

Step 2 - Convert to Binary

Used long division method for decimal to binary conversion.

Decimal Number: 2910

Calculating Binary Equivalent:

Step	Operation	Result	Remainder
Step 1	29 / 2	14	1
Step 2	14 / 2	7	0
Step 3	7 / 2	3	1
Step 4	3 / 2	1	1
Step 5	1 / 2	0	1

As mentioned in Steps 2 and 4, the remainders have to be arranged in the reverse order so that the first remainder becomes the least significant digit (LSD) and the last remainder becomes the most significant digit (MSD).

Decimal Number: 2910 = Binary Number: 111012

Result

$$(00101001)_{BCD} = (11101)_2$$

BCD to Excess-3

Steps

- Step 1 -- Convert BCD to decimal.
- Step 2 -- Add (3)10 to this decimal number.
- Step 3 -- Convert into binary to get excess-3 code.

Example: convert (1001)BCD to Excess-3.

Step 1 - Convert to decimal

(1001)BCD = 910

Step 2 - Add 3 to decimal

(9)10 + (3)10 = (12)10

Step 3 - Convert to Excess-3

(12)10 = (1100)2

Result

$$(1001)_{BCD} = (1100)_{XS-3}$$

Excess-3 to BCD Conversion

Steps

- Step 1 -- Subtract (0011)2 from each 4 bit of excess-3 digit to obtain the corresponding BCD code.

Example: convert (10011010)XS-3 to BCD.

Given XS-3 number = 1 0 0 1 1 0 1 0

Subtract (0011)2 = 0 0 1 1 0 0 1 1

BCD = 0 1 1 0 0 1 1 1

Result

$$(10011010)_{XS-3} = (01100111)_{BCD}$$

Logic Gates

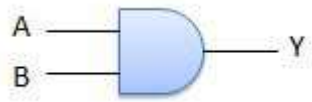
Logic gates are the basic building blocks of any digital system. It is an electronic circuit having one or more than one input and only one output. The relationship between the input and the output is based on a certain logic. Based on this logic gates are named as AND gate, OR gate, NOT gate etc.

AND Gate

A circuit which performs an AND operation is shown in figure. It has n input (n >= 2) and one output.

Y	=	A AND B AND C ..... N
Y	=	A.B.C ..... N
Y	=	ABC ..... N

Logic diagram



Truth Table

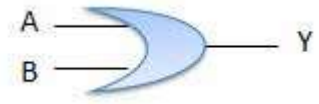
Inputs		Output
A	B	AB
0	0	0
0	1	0
1	0	0
1	1	1

OR Gate

A circuit which performs an OR operation is shown in figure. It has n input (n >= 2) and one output.

Y	=	A OR B OR C ..... N
Y	=	A + B + C ..... N

Logic diagram



Truth Table

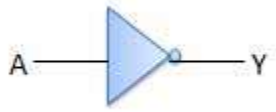
Inputs		Output
A	B	A + B
0	0	0
0	1	1
1	0	1
1	1	1

NOT Gate

NOT gate is also known as Inverter. It has one input A and one output Y.

$$Y = \text{NOT } A$$
$$Y = \overline{A}$$

Logic diagram



Truth Table

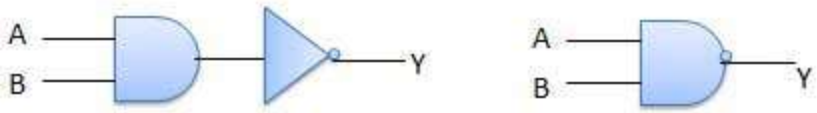
Inputs	Output
A	B
0	1
1	0

NAND Gate

A NOT-AND operation is known as NAND operation. It has n input (n >= 2) and one output.

Y = A NOT AND B NOT AND C ..... N
Y = A NAND B NAND C ..... N

Logic diagram



Truth Table

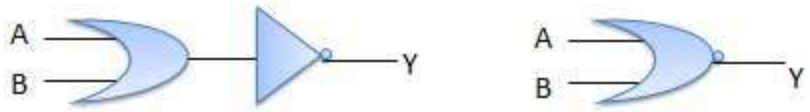
Inputs		Output
A	B	$\overline{AB}$
0	0	1
0	1	1
1	0	1
1	1	0

NOR Gate

A NOT-OR operation is known as NOR operation. It has n input (n >= 2) and one output.

Y = A NOT OR B NOT OR C ..... N
Y = A NOR B NOR C ..... N

Logic diagram



Truth Table

Inputs		Output
A	B	$\overline{A+B}$
0	0	1
0	1	0
1	0	0
1	1	0

XOR Gate

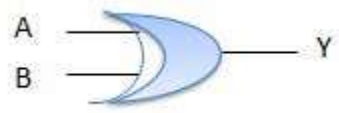
XOR or Ex-OR gate is a special type of gate. It can be used in the half adder, full adder and subtractor. The exclusive-OR gate is abbreviated as EX-OR gate or sometime as X-OR gate. It has n input (n >= 2) and one output.

Y = A XOR B XOR C ..... N

Y = $A \oplus B \oplus C$ ..... N

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

Logic diagram



Truth Table

Inputs		Output
A	B	$A \oplus B$
0	0	0
0	1	1
1	0	1
1	1	0

XNOR Gate

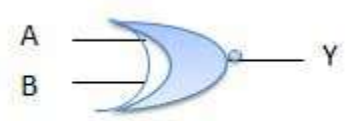
XNOR gate is a special type of gate. It can be used in the half adder, full adder and subtractor. The exclusive-NOR gate is abbreviated as EX-NOR gate or sometime as X-NOR gate. It has n input (n >= 2) and one output.

$$Y = A \text{ XOR } B \text{ XOR } C \ldots\ldots N$$

$$Y = A \odot B \odot C \ldots\ldots N$$

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

Logic diagram



Truth Table

Inputs		Output
A	B	$A \odot B$
0	0	1
0	1	0
1	0	0
1	1	1

Boolean Algebra

Boolean Algebra is used to analyze and simplify the digital (logic) circuits. It uses only the binary numbers i.e. 0 and 1. It is also called as Binary Algebra or logical Algebra. Boolean algebra was invented by George Boole in 1854.

Rule in boolean algebra

Following are the important rules used in boolean algebra.

- Variable used can have only two values. Binary 1 for HIGH and Binary 0 for LOW.

- Complement of a variable is represented by a overbar (-). Thus complement of variable B is represented as $\overline{B}$. Thus if B = 0 then $\overline{B}$ = 1 and B = 1 then $\overline{B}$ = 0.
- ORing of the variables is represented by a plus (+) sign between them. For example ORing of A, B, C is represented as A + B + C.
- Logical ANDing of the two or more variable is represented by writing a dot between them such as A.B.C. Sometime the dot may be omitted like ABC.

Boolean Laws

There are six types of Boolean Laws.

Commutative law

Any binary operation which satisfies the following expression is referred to as commutative operation

(i) $A.B = B.A$ (ii) $A + B = B + A$

Commutative law states that changing the sequence of the variables does not have any effect on the output of a logic circuit.

Associative law

This law states that the order in which the logic operations are performed is irrelevant as their effect is the same.

(i) $(A.B).C = A.(B.C)$ (ii) $(A + B) + C = A + (B + C)$

Distributive law

Distributive law states the following condition.

$A.(B + C) = A.B + A.C$

AND law

These laws use the AND operation. Therefore they are called as AND laws.

(i) $A.0 = 0$ (ii) $A.1 = A$

(iii) $A.A = A$ (iv) $A.\overline{A} = 0$

OR law

These laws use the OR operation. Therefore they are called as OR laws.

(i) $A + 0 = A$ (ii) $A + 1 = 1$

(iii) $A + A = A$ (iv) $A + \overline{A} = 1$

INVERSION law

This law uses the NOT operation. The inversion law states that double inversion of a variable result in the original variable itself.

$\overline{\overline{A}} = A$

Important Boolean Theorems

Following are few important boolean Theorems.

De Morgan's Theorems

The two theorems suggested by De-Morgan which are extremely useful in Boolean Algebra are as following.

Theorem 1

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

NAND = Bubbled OR

- The left hand side (LHS) of this theorem represents a NAND gate with input A and B where the right hand side (RHS) of the theorem represents an OR gate with inverted inputs.
- This OR gate is called as Bubbled OR.

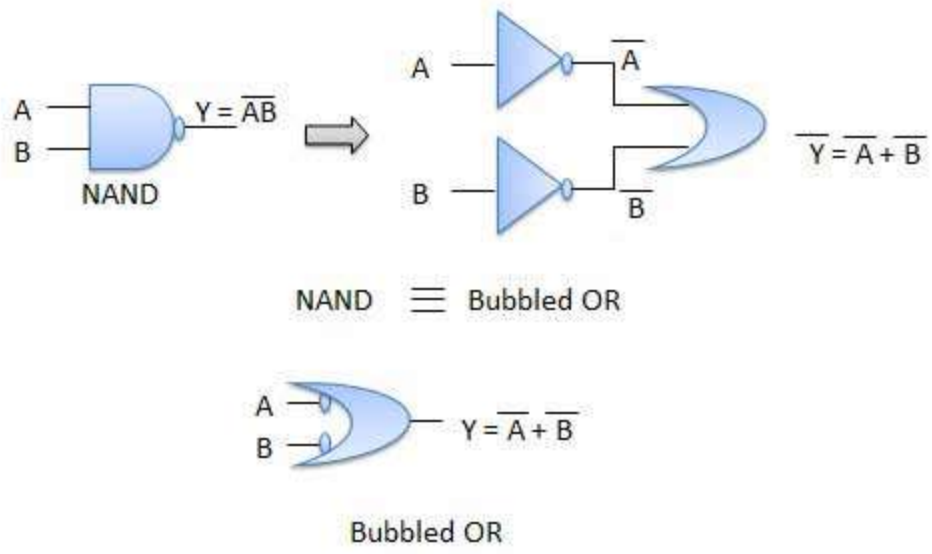


Table showing verification of the De-Morgans's first theorem

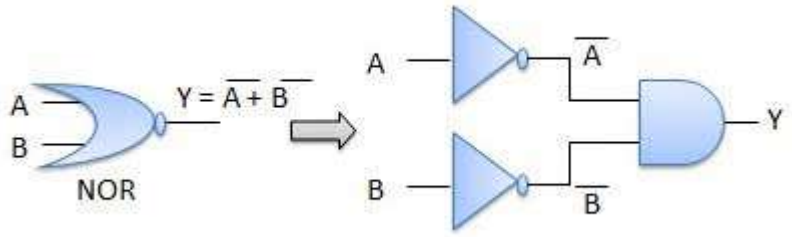
A	B	$\overline{AB}$	$\overline{A}$	$\overline{B}$	$\overline{A} + \overline{B}$
0	0	1	1	1	1
0	1	1	1	0	1
1	0	1	0	1	1
1	1	0	0	0	0

Theorem 2

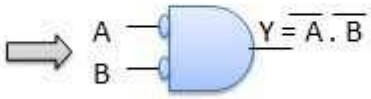
$\overline{A + B} = \overline{A}.\overline{B}$

NOR = Bubbled AND

- The LHS of this theorem represented a NOR gate with input A and B whereas the RHS represented an AND gate with inverted inputs.
- This AND gate is called as Bubbled AND.



NOR ☰ Bubbled AND



Bubbled AND

Table showing verification of the De-Morgans's second theorem

A	B	$\overline{A+B}$	$\overline{A}$	$\overline{B}$	$\overline{A} . \overline{B}$
0	0	1	1	1	1
0	1	0	1	0	0
1	0	0	0	1	0
1	1	0	0	0	0

Boolean Expression/Function

Boolean algebra deals with binary variables and logic operation. A Boolean Function is described by an algebraic expression called Boolean expression which consists of binary variables, the constants 0 and 1 and the logic operation symbols. Consider the following example

F (A, B, C, D)	=	$A + \overline{BC} + ADC$	Equation No. 1
Boolean Function		Boolean Expression	

Here the left side of the equation represents the output Y. So we can state equation no. 1

Y	=	$A + \overline{BC} + ADC$

Truth Table Formation

A truth table represents a table having all combinations of inputs and their corresponding result.

It is possible to convert the switching equation into a truth table. For example consider the following switching equation.

$$F(A, B, C) = A + BC$$

The output will be high (1) if A = 1 or BC = 1 or both are 1. The truth table for this equation is shown by Table (a). The number of rows in the truth table is 2n where n is the number of input variables (n=3 for the given equation). Hence there are 23 = 8 possible input combination of inputs.

Inputs			Output
A	B	C	F
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	1

Methods to simplify the boolean function

The methods used for simplifying the Boolean function are as follows.

- Karnaugh-map or K-map.
- NAND gate method

Karnaugh-map or K-map

The Boolean theorems and De-Morgan's theorems are useful in manipulating the logic expression. We can realize the logical expression using gates. The no. of logic gates required for the realization of a logical expression should be reduced to minimum possible value by K-map method.This method can be by two way

Sum of Products (SOP) Form

It is in the form of sum of three terms AB,AC,BC with each individual term is product of two variable. Say A.B or A.C etc. Therefore such expression are known as expression in SOP form. The sum and products in SOP form are not the actual additions or multiplications. In fact they are the OR and AND functions. In SOP form, 0 is represent for bar and 1 is represent for unbar. SOP form is represented by Σ.

Example of SOP is as follows.

In SOP form $\bar{A}\bar{B} + \bar{A}B + AB$

0 0 0 1 1 1

A \ B	0	1
0	1	1
1		1

Σ m (0,1,3)

Answer: $\bar{A}\bar{B} + \bar{A}B + AB = \bar{A} + B$

Product of Sums (POS) Form

It is in the form of product of three terms (A+B),(B+C) and (A+C) with each term is in the form of sum of two variables. Such expression are said to be in the product of sums (POS) form. In POS form, 0 is represent for unbar and 1 is represent for bar. POS form is represented by Π.

Example of POS is as follows.

In POS form $(B + \bar{C})(\bar{A} + \bar{B})(\bar{B} + C)$

0 1 1 1 1 0

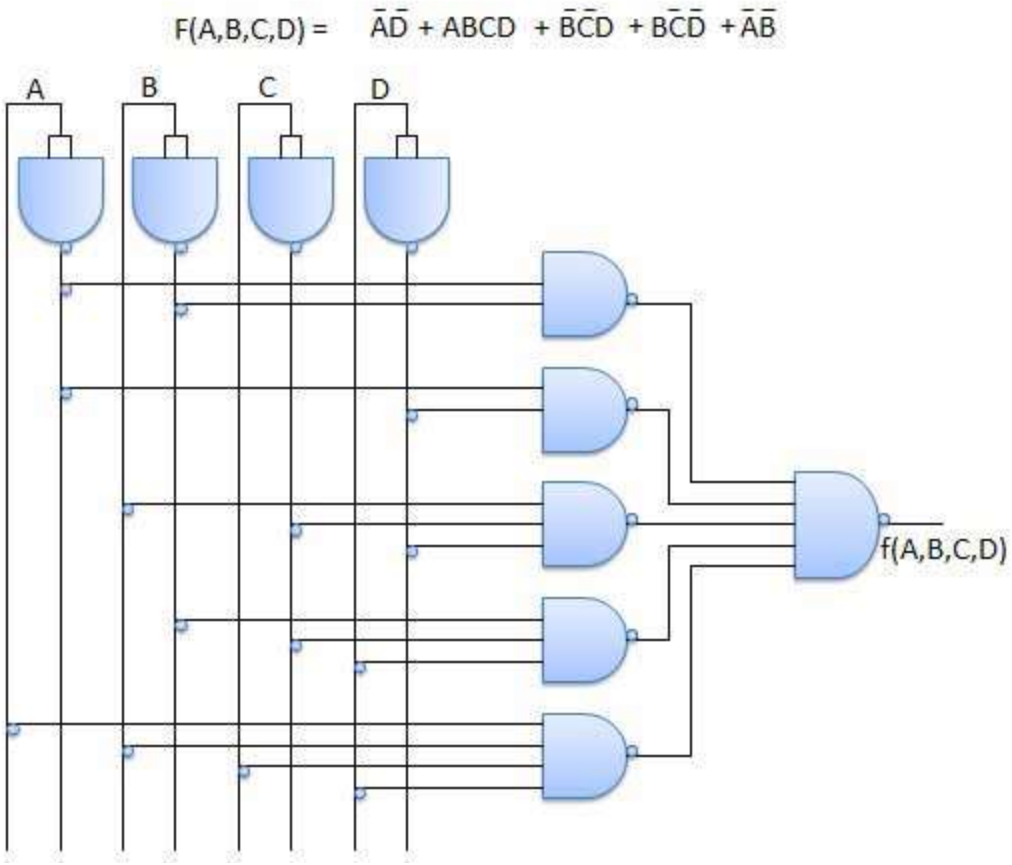
A \ BC	00	01	10	11
0		0	0	0
1				

Π m (1,3,2)

Answer : $(A + \bar{C})(A + \bar{B})$

NAND gates Realization

NAND gates can be used to simplify boolean functions as shown in example below.



Combinational Circuits

Combinational circuit is circuit in which we combine the different gates in the circuit for example encoder, decoder, multiplexer and demultiplexer. Some of the characteristics of combinational circuits are following.

- The output of combinational circuit at any instant of time, depends only on the levels present at input terminals.
- The combinational circuit do not use any memory. The previous state of input does not have any effect on the present state of the circuit.
- A combinational circuit can have a n number of inputs and m number of outputs.

Block diagram

We're going to elaborate few important combinational circuits as follows.

Half Adder

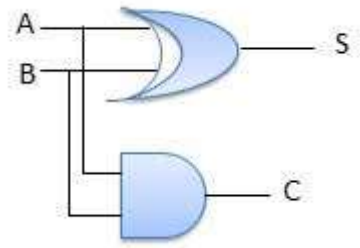
Half adder is a combinational logic circuit with two input and two output. The half adder circuit is designed to add two single bit binary number A and B. It is the basic building block for addition of two single bit numbers. This circuit has two outputs carry and sum.

Block diagram

Truth Table

Inputs		Output	
A	B	S	C
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

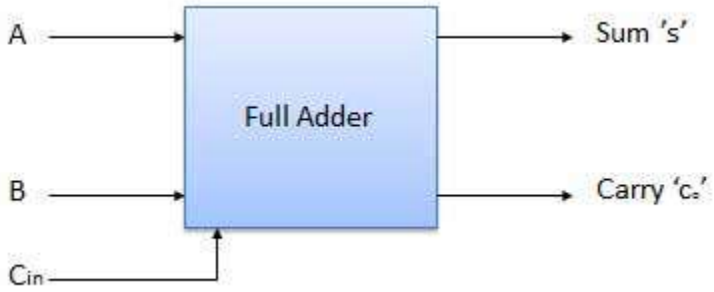
Circuit Diagram



Full Adder

Full adder is developed to overcome the drawback of Half Adder circuit. It can add two one-bit numbers A and B, and carry c. The full adder is a three input and two output combinational circuit.

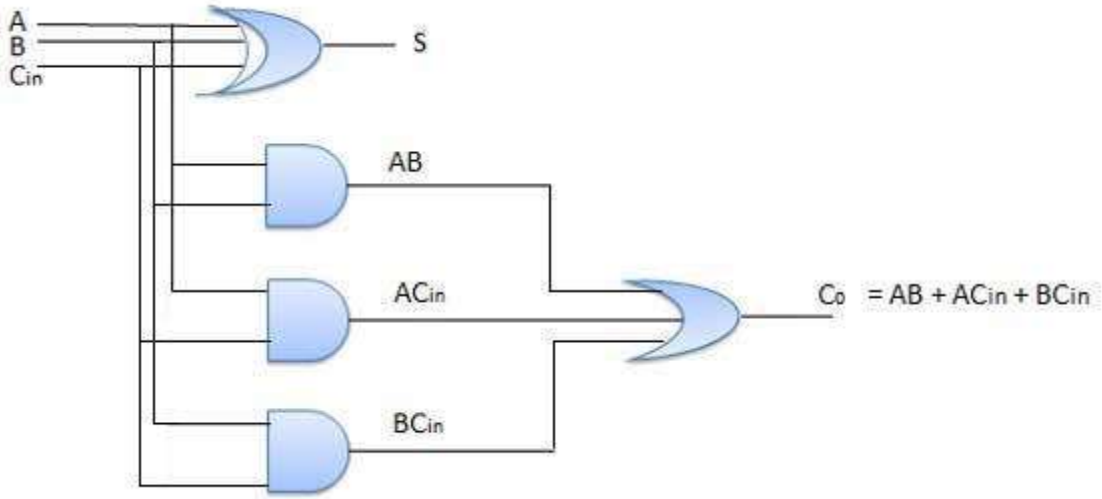
Block diagram



Truth Table

Inputs			Output	
A	B	Cin	S	Co
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

Circuit Diagram



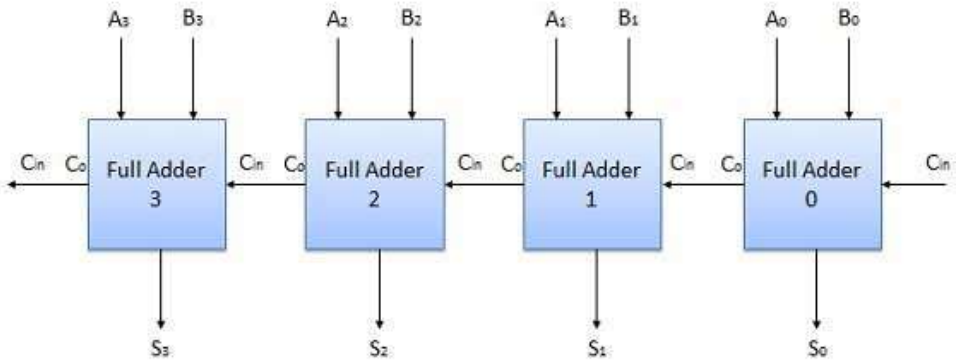
N-Bit Parallel Adder

The Full Adder is capable of adding only two single digit binary number along with a carry input. But in practical we need to add binary numbers which are much longer than just one bit. To add two n-bit binary numbers we need to use the n-bit parallel adder. It uses a number of full adders in cascade. The carry output of the previous full adder is connected to carry input of the next full adder.

4 Bit Parallel Adder

In the block diagram, A0 and B0 represent the LSB of the four bit words A and B. Hence Full Adder-0 is the lowest stage. Hence its Cin has been permanently made 0. The rest of the connections are exactly same as those of n-bit parallel adder is shown in fig. The four bit parallel adder is a very common logic circuit.

Block diagram



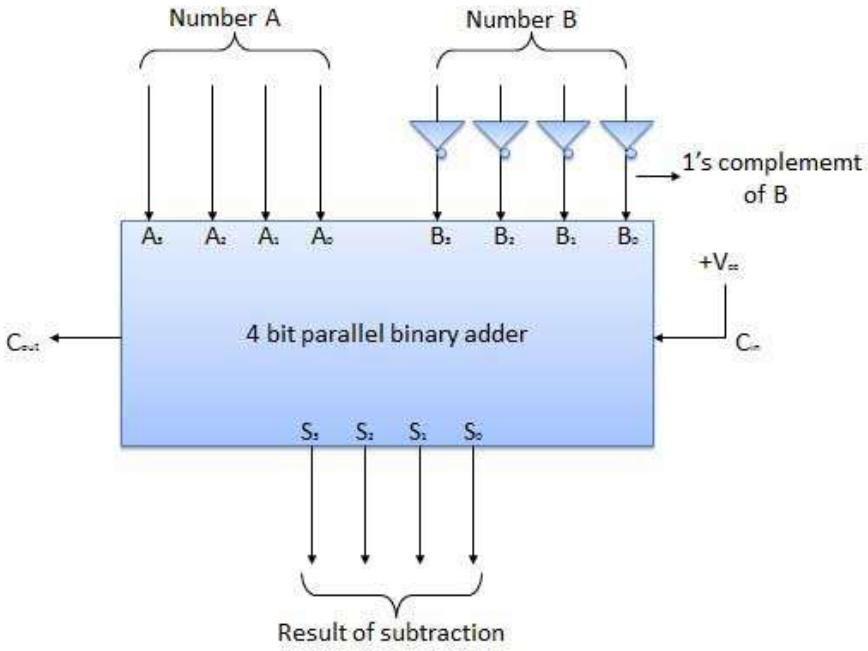
N-Bit Parallel Subtractor

The subtraction can be carried out by taking the 1's or 2's complement of the number to be subtracted. For example we can perform the subtraction (A-B) by adding either 1's or 2's complement of B to A. That means we can use a binary adder to perform the binary subtraction.

4 Bit Parallel Subtractor

The number to be subtracted (B) is first passed through inverters to obtain its 1's complement. The 4-bit adder then adds A and 2's complement of B to produce the subtraction. S3 S2 S1 S0 represent the result of binary subtraction (A-B) and carry output Cout represents the polarity of the result. If A > B then Cout =0 and the result of binary form (A-B) then Cout = 1 and the result is in the 2's complement form.

Block diagram



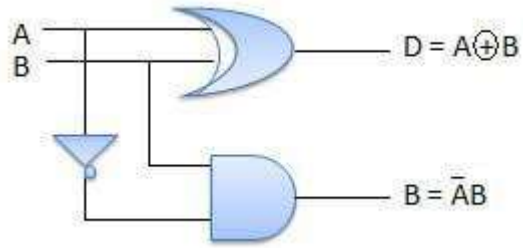
Half Subtractors

Half subtractor is a combination circuit with two inputs and two outputs (difference and borrow). It produces the difference between the two binary bits at the input and also produces a output (Borrow) to indicate if a 1 has been borrowed. In the subtraction (A-B), A is called as Minuend bit and B is called as Subtrahend bit.

Truth Table

Inputs		Output	
A	B	(A – B)	Borrow
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

Circuit Diagram



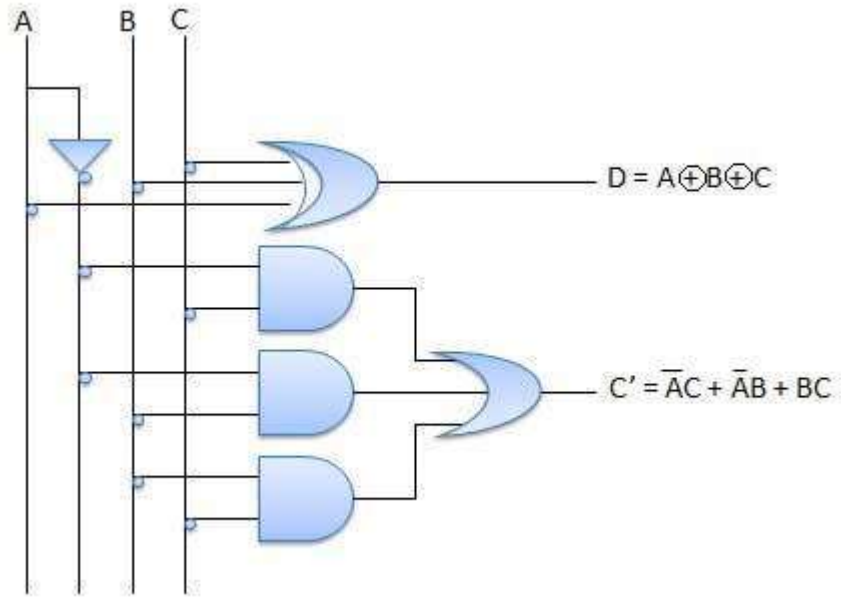
Full Subtractors

The disadvantage of a half subtractor is overcome by full subtractor. The full subtractor is a combinational circuit with three inputs A,B,C and two output D and C'. A is the minuend, B is subtrahend, C is the borrow produced by the previous stage, D is the difference output and C' is the borrow output.

Truth Table

Inputs			Output	
A	B	C	(A-B-C)	C'
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

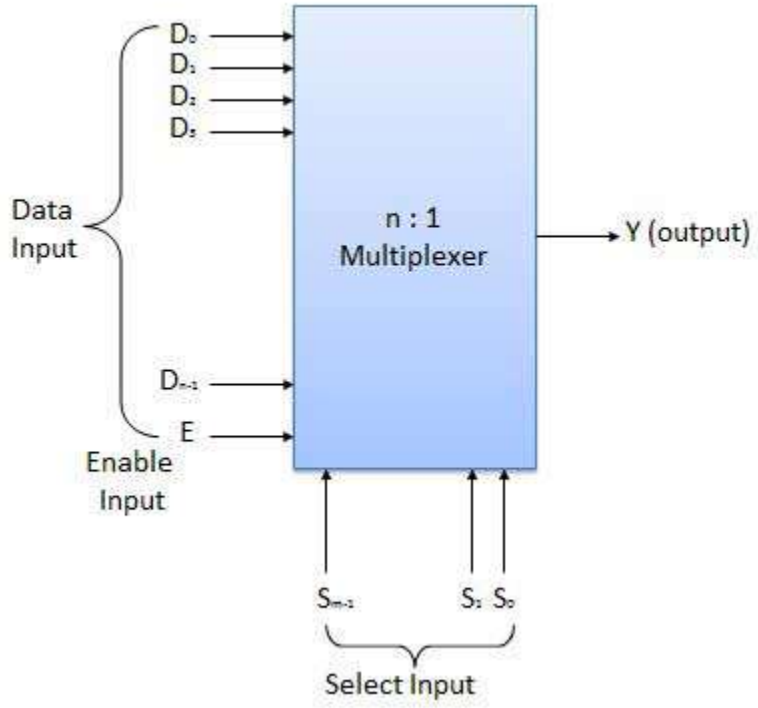
Circuit Diagram



Multiplexers

Multiplexer is a special type of combinational circuit. There are n-data inputs, one output and m select inputs with 2m = n. It is a digital circuit which selects one of the n data inputs and routes it to the output. The selection of one of the n inputs is done by the selected inputs. Depending on the digital code applied at the selected inputs, one out of n data sources is selected and transmitted to the single output Y. E is called the strobe or enable input which is useful for the cascading. It is generally an active low terminal, that means it will perform the required operation when it is low.

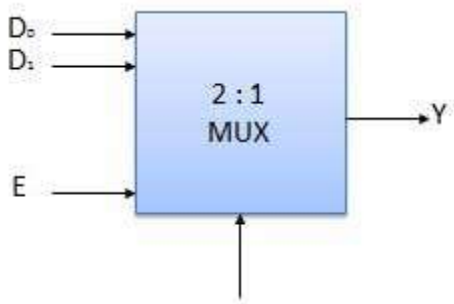
Block diagram



Multiplexers come in multiple variations

- 2 : 1 multiplexer
- 4 : 1 multiplexer
- 16 : 1 multiplexer
- 32 : 1 multiplexer

Block Diagram



Truth Table

Enable	Select	Output
E	S	Y
0	x	0
1	0	D_0
1	1	D_1

x = Don't care

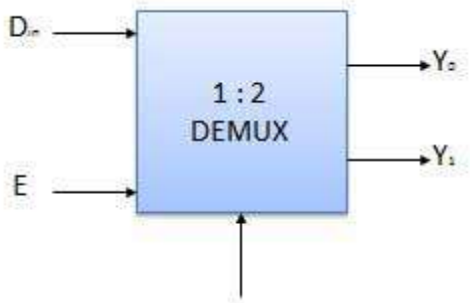
Demultiplexers

A demultiplexer performs the reverse operation of a multiplexer i.e. it receives one input and distributes it over several outputs. It has only one input, n outputs, m select input. At a time only one output line is selected by the select lines and the input is transmitted to the selected output line. A de-multiplexer is equivalent to a single pole multiple way switch as shown in fig.

Demultiplexers come in multiple variations

- 1 : 2 demultiplexer
- 1 : 4 demultiplexer
- 1 : 16 demultiplexer
- 1 : 32 demultiplexer

Block diagram



Truth Table

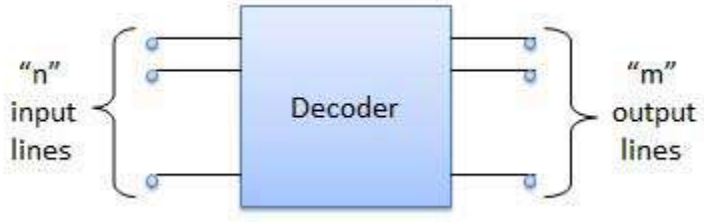
Enable	Select	Output	
E	S	Y0	Y1
0	x	0	0
1	0	0	D_{in}
1	1	D_{in}	0

x = Don't care

Decoder

A decoder is a combinational circuit. It has n input and to a maximum m = 2n outputs. Decoder is identical to a demultiplexer without any data input. It performs operations which are exactly opposite to those of an encoder.

Block diagram



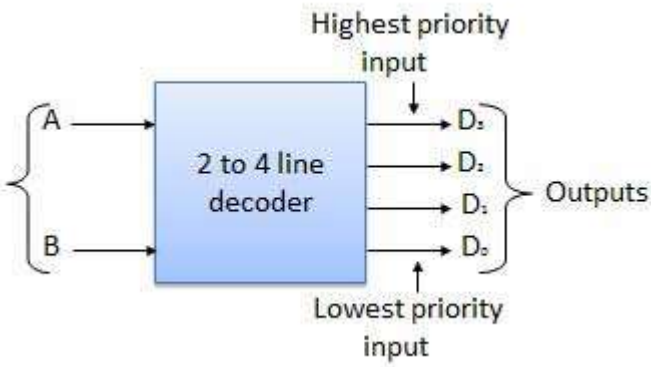
Examples of Decoders are following.

- Code converters
- BCD to seven segment decoders
- Nixie tube decoders
- Relay actuator

2 to 4 Line Decoder

The block diagram of 2 to 4 line decoder is shown in the fig. A and B are the two inputs where D through D are the four outputs. Truth table explains the operations of a decoder. It shows that each output is 1 for only a specific combination of inputs.

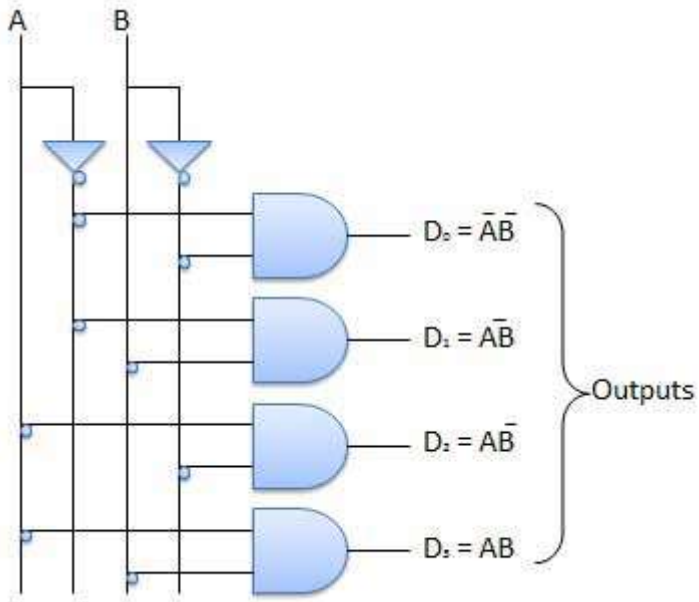
Block diagram



Truth Table

Inputs		Output			
A	B	D_0	D_1	D_2	D_3
0	0	1	0	0	0
0	1	0	1	0	0
0	1	0	0	1	0
1	1	0	0	0	1

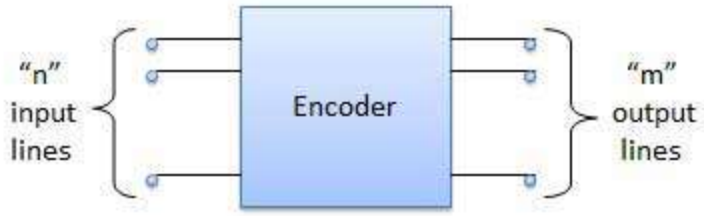
Logic Circuit



Encoder

Encoder is a combinational circuit which is designed to perform the inverse operation of the decoder. An encoder has n number of input lines and m number of output lines. An encoder produces an m bit binary code corresponding to the digital input number. The encoder accepts an n input digital word and converts it into an m bit another digital word.

Block diagram



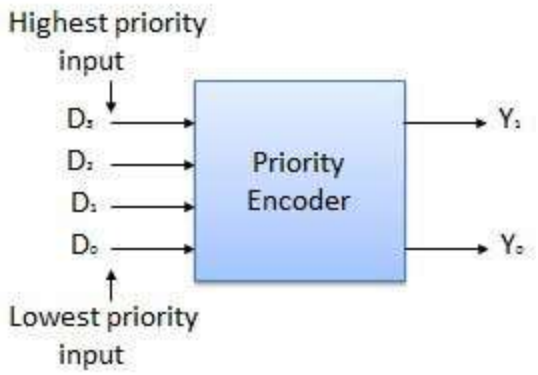
Examples of Encoders are following.

- Priority encoders
- Decimal to BCD encoder
- Octal to binary encoder
- Hexadecimal to binary encoder

Priority Encoder

This is a special type of encoder. Priority is given to the input lines. If two or more input line are 1 at the same time, then the input line with highest priority will be considered. There are four input D0, D1, D2, D3 and two output Y0, Y1. Out of the four input D3 has the highest priority and D0 has the lowest priority. That means if D3 = 1 then Y1 Y1 = 11 irrespective of the other inputs. Similarly if D3 = 0 and D2 = 1 then Y1 Y0 = 10 irrespective of the other inputs.

Block diagram



Truth Table

Highest	Inputs		Lowest	Outputs	
D_3	D_2	D_1	D_0	Y_0	Y_1
0	0	0	0	x	x
0	0	0	1	0	0
0	0	1	x	0	1
0	1	x	x	1	0
1	x	x	x	1	1