

Number Systems

1.1 Introduction

Many number systems are used, such as decimal, binary, octal, hexadecimal, etc. All people are using the decimal system daily. So that, the most common used system is the decimal number system. The other number systems are used in digital systems applications. The feature which distinguishes one system from another is the number of digits which are used, and this is called the base (radix) of the system. These systems are classified according to the radix of the number system as shown below:

Base	name of number system	digits used in system
2	Binary	0, 1
8	Octal	0, 1, 2, 3, 4, 5, 6, 7
10	Decimal	0, 1, 2, 3, 4, 5, 6, 7, 8, 9
16	Hexadecimal	0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F

In general, quantities are represented as:

$$N = a_{-1} r^{-1} + a_{-2} r^{-2} + \dots + a_0 r^0 + a_1 r^1 + a_2 r^2 + \dots + a_n r^n$$

Where each coefficient a , can take any value of the number system digits and r is the base of the number system.

A **decimal number** system uses 10 digits to represent any quantity. The thousands, hundreds, etc., are powers of 10 implied by the position of the coefficients (symbols) in the number. The digit in the right is called Least Significant Digit (LSD), and the digit in the left is called Most Significant Digit (MSD).

....	10^4	10^3	10^2	10^1	10^0	.	10^{-1}	10^{-2}	10^{-3}
....	10000	1000	100	10	1	.	$\frac{1}{10}$	$\frac{1}{100}$	$\frac{1}{1000}$

Example: $(3752.46)_{10} = 3 \times 1000 + 7 \times 100 + 5 \times 10 + 2 \times 1 + 4 \times \frac{1}{10} + 6 \times \frac{1}{100}$

Binary Number: the decimal number can be represented in binary by arranging the 1 and 0 under weight of the binary system to get the decimal number. Each digit in binary number called a **Bit**. The bit in the right is called Least Significant Bit (LSB), and the bit in the left is called Most Significant Bit (MSB). The positional weight of each bit is a power of 2.

....	2^4	2^3	2^2	2^1	2^0	.	2^{-1}	2^{-2}	2^{-3}
....	16	8	4	2	1	.	$\frac{1}{2}$	$\frac{1}{4}$	$\frac{1}{8}$

Example: $(1101.11)_2 = 1 \times 8 + 1 \times 4 + 0 \times 2 + 1 \times 1 + 1 \times \frac{1}{2} + 1 \times \frac{1}{4}$

Octal Number: the decimal number can be present in Octal by arranging basic digits according to the octal system to get the decimal number where the system uses only 8 digits to represent any quantity. The positional weight of each digit is a power of 8.

....	8^4	8^3	8^2	8^1	8^0	.	8^{-1}	8^{-2}	8^{-3}
....	4096	512	64	8	1	.	$\frac{1}{8}$	$\frac{1}{64}$	$\frac{1}{512}$

Example: $(752.46)_8 = 7 \times 64 + 5 \times 8 + 2 \times 1 + 4 \times \frac{1}{8} + 6 \times \frac{1}{64}$

The **hexadecimal number system** is used commonly by designers to represent long strings of bits in the addresses, instructions, and data in digital systems. This system uses 16 digits to represent any quantity. The positional weight of each digit is a power of 16.

....	16^4	16^3	16^2	16^1	16^0	.	16^{-1}	16^{-2}	16^{-3}
....	65536	4096	256	16	1	.	$\frac{1}{16}$	$\frac{1}{256}$	$\frac{1}{4096}$

Example: $(7EA.8F)_{16} = 7 \times 256 + 14 \times 16 + 10 \times 1 + 8 \times \frac{1}{16} + 15 \times \frac{1}{256}$

1.2 Number Base Conversion

Representations of a number in a different radix are said to be equivalent if they have the same decimal representation. It is often required to convert a number in a particular number system to any other number system, e.g., it may be required to convert a decimal number to binary or octal or hexadecimal. The reverse is also true, i.e., a binary number may be converted into decimal and so on.

1.2.1 Decimal number-to-other number systems Conversion

The conversion process of a decimal number into any number system can be done according to the following steps:

- 1- Separate the integer part and the fraction part.
- 2- Divide the integer part by the required base until the quotient of zero is obtained.
- 3- The column of the remainder is read from bottom to top.
- 4- Multiplied the fraction part with the required base until zero fraction is obtained
- 5- The column of integer part of result is read from top to bottom.

1.2.1.1 Decimal to binary conversion

The above steps will be applied with the base of **2**.

Example. Convert $(34.25)_{10}$ into an equivalent binary number

Solution: the integer part is 34 and can be converted as follows:

Division	Quotient	Remainder	
$34 \div 2 =$	17	0	LSB
$17 \div 2 =$	8	1	
$8 \div 2 =$	4	0	
$4 \div 2 =$	2	0	
$2 \div 2 =$	1	0	
$1 \div 2 =$	0	1	MSB

The fraction part is 0.25 and it can be converted as follows:

Multiplication	result	integer part of result	
$0.25 \times 2 =$	0.5	0	MSB
$0.5 \times 2 =$	1.0	1	LSB

Hence the converted binary number is $(100010.01)_2$.

1.2.1.2 Decimal-to-octal Conversion

Similarly, the same steps are used with the base of **8**.

Example. Convert $(35.3125)_{10}$ into an octal number.

Solution: the integer part is 35 which can be converted as follows

Division	Quotient	Remainder	
$35 \div 8 =$	4	3	LSD
$4 \div 8 =$	0	4	MSD

The fraction part is 0.3125 and it can be converted as follows:

Multiplication	Result	integer part of result	
$0.3125 \times 8 =$	2.5	2	MSD
$0.5 \times 8 =$	4.0	4	LSD

Hence the converted octal number is $(43.24)_8$.

1.2.1.3 Decimal-to-hexadecimal Conversion

The same steps are repeated with the base of **16**.

Example. Convert $(34.3)_{10}$ into a hexadecimal number.

Solution: the integer part is 34 which can be converted as follows

Division	Quotient	Remainder	
$34 \div 16 =$	2	2	LSD
$2 \div 16 =$	0	2	MSD

The fraction part is 0.3 and it can be converted as follows:

Multiplication	Result	integer part of result	
$0.3 \times 16 =$	4.8	4	MSD
$0.8 \times 16 =$	12.8	12	
$0.8 \times 16 =$	12.8	12	LSD

This is cyclic number

Hence the converted hexadecimal number is $(22.4CC)_{16}$.

1.2.2 Conversion from any number system to decimal system

The conversion process from any number system to decimal system depends on the summation of the multiplied digits by the positional weight of that system.

1.2.2.1 Binary-to-decimal Conversion

Each of the digits in the number systems discussed above has a positional weight as in the case of the decimal system in which it is a power of 2 for binary system.

Example. Convert $(10101.01)_2$ into a decimal number.

Solution.

$$\begin{aligned} N &= 1 \times 2^4 + 0 \times 2^3 + 1 \times 2^2 + 0 \times 2^1 + 1 \times 2^0 + 0 \times 2^{-1} + 1 \times 2^{-2} \\ &= 1 \times 16 + 0 \times 8 + 1 \times 4 + 0 \times 2 + 1 \times 1 + 0 \times 0.5 + 1 \times 0.25 \\ &= 21.25 \end{aligned}$$

Hence the converted decimal number is $(21.25)_{10}$.

1.2.2.2 Octal-to-decimal Conversion

The positional weight of each digit in octal number is a power of 8.

Example. Convert $(162.35)_8$ into an equivalent decimal number.

Solution.

$$\begin{aligned} N &= 1 \times 8^2 + 6 \times 8^1 + 2 \times 8^0 + 3 \times 8^{-1} + 5 \times 8^{-2} \\ &= 1 \times 64 + 6 \times 8 + 2 \times 1 + 3 \times \frac{1}{8} + 5 \times \frac{1}{64} = 114.453125 \end{aligned}$$

Hence the converted decimal number is $(114.453125)_{10}$.

1.2.2.3 Hexadecimal-to-decimal Conversion

The positional weight of each digit in hexadecimal number is a power of 16.

Example. Convert $(3CD.F9)_{16}$ into an equivalent decimal number.

Solution.

$$\begin{aligned} N &= 3 \times 16^2 + 12 \times 16^1 + 13 \times 16^0 + 15 \times 16^{-1} + 9 \times 16^{-2} \\ &= 3 \times 256 + 12 \times 16 + 13 \times 1 + 15 \times \frac{1}{16} + 9 \times \frac{1}{256} \\ &= 973.97265625 \end{aligned}$$

Hence the converted decimal number is $(973.97265625)_{10}$.

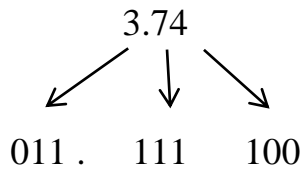
1.2.3 Conversion from Octal to Binary Number and Vice Versa

The conversion from octal to binary is performed by converting each octal digit to its three-bits binary equivalent. The eight possible digits are converted as indicated in this table.

Octal Digit	0	1	2	3	4	5	6	7
Binary Equivalent	000	001	010	011	100	101	110	111

Example. Convert $(3.74)_8$ into an equivalent binary number.

Solution: by converting each digit into binary of three bits group.

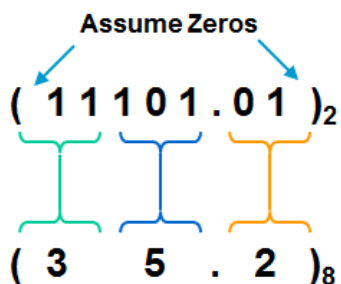


Hence the equivalent binary number is $(011.111100)_2$.

Converting from binary to octal is simply the reverse of the foregoing process. The bits of the binary number are grouped into groups of three bits starting from the LSB for integer part and starting from MSB for fraction part. Sometimes the binary number will not have even groups of three bits. For those cases, we can add one or two 0s to the left of the MSB for integer part and to the right of the LSB for fraction part.

Example: convert $(11101.01)_2$ into an equivalent octal number.

Solution:



Hence the equivalent octal number is $(35.2)_8$.

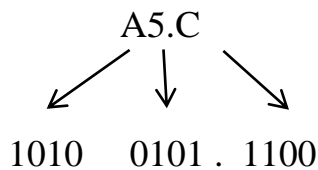
1.2.4 Conversion from Hexadecimal to Binary Number and Vice Versa

The conversion from hexadecimal to binary is performed by converting each hexa digit to its four-bits binary equivalent. The sixteen possible digits are converted as indicated in this table.

Hexa	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
Binary	0000	0001	0010	0011	0100	0101	0110	0111	1000	1001	1010	1011	1100	1101	1110	1111

Example. Convert $(A5.C)_{16}$ into an equivalent binary number.

Solution: by converting each digit into binary of four bits group.

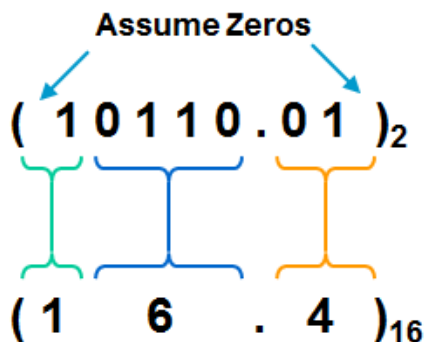


Hence, the equivalent binary number is $(10100101.1100)_2$.

Converting from binary to hexa is simply the reverse of the foregoing process. The bits of the binary number are grouped into groups of four bits starting from the LSB for integer part and starting from MSB for fraction part. Sometimes the binary number will not have even groups of four bits. For those cases, we can add one, two or three 0s to the left of the MSB for integer part and to the right of the LSB for fraction part.

Example: Convert $(10110.01)_2$ into an equivalent hexadecimal number.

Solution:



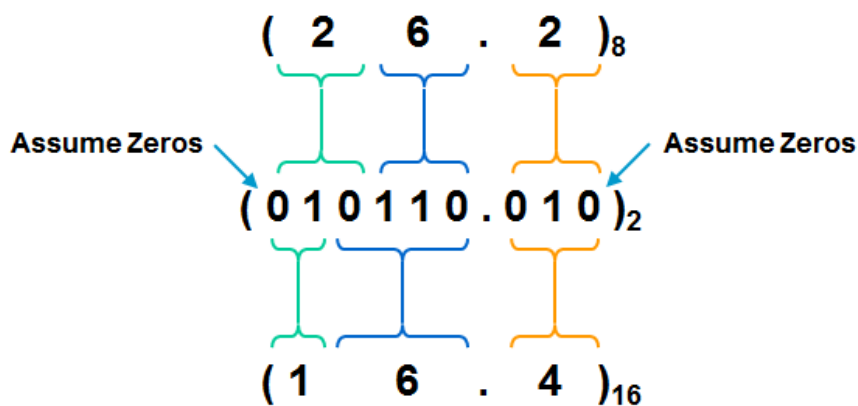
Hence the equivalent hexa number is $(16.4)_{16}$.

1.2.5 Conversion from an Octal to Hexadecimal and Vice Versa

Conversion from octal to hexadecimal and vice versa is sometimes required. To convert an octal number into a hexadecimal number the following steps are to be followed:

- (i) First convert the octal number to its binary equivalent (as already discussed above).
- (ii) Then form groups of 4 bits, starting from the LSB.
- (iii) Then write the equivalent hexadecimal number for each group of 4 bits.

Example: convert $(26.2)_8$ into hexadecimal.



Example: convert $(16.4)_{16}$ into octal.

Solution:

$$\begin{aligned}
 & (\quad \underbrace{1} \quad \underbrace{6} \quad . \quad \underbrace{4} \quad)_{16} \\
 & (\quad \overbrace{0001} \quad \overbrace{0110} \quad . \quad \overbrace{0100} \quad)_2 \\
 & (00 \quad \underbrace{010} \quad \underbrace{110} \quad . \quad \underbrace{010} \quad 0)_2 \\
 & (\quad \overbrace{2} \quad \overbrace{6} \quad . \quad \overbrace{2} \quad)_8
 \end{aligned}$$

1.3 Complement:

Complements are used in digital computers to **simplify the subtraction operation** and for logical manipulation. There are two types of complements for each base- r system: the radix complement and the diminished radix complement. The first is referred to as the r 's complement and the second as the $(r - 1)$'s complement.

1.3.1 Binary numbers Complement:

1- One's (first) Complement:

$$1's \text{ complement} = r^n - N - 1$$

where n : number of bits

N : binary number

r : system base

Simply the 1's complement of binary number is the number we get by changing each bit (0 to 1) and (1 to 0).

Example: the first complement of $(101100)_2$

Solution:

binary number	101100
1's complement	010011

2- The Two's (second) Complement:

The equation is:

$$2's \text{ complement} = r^n - N$$

Simply the 2's complement is equal to 1's complement added by one.

Example: find the 2's complement of $(101101)_2$

Solution:

binary number	101101
1's complement	010010
2's complement	$010010 + 1 = 010011$

1.4 Binary Arithmetic Operations

1- Addition:-

$$\begin{array}{r}
 0 + 0 = 0 \\
 0 + 1 = 1 \\
 1 + 0 = 1 \\
 1 + 1 = 0 \quad \text{carry 1}
 \end{array}$$

Example: Add the two binary numbers (001) and (100)

$$\begin{array}{r} 001 \\ + 100 \\ \hline 101 \end{array}$$

Example: Add the two binary numbers (111) and (001)

$$\begin{array}{r} \textcircled{1} \quad \textcircled{1} \\ 1 \quad 1 \quad 1 \\ + 0 \quad 0 \quad 1 \\ \hline 1 \quad 0 \quad 0 \quad 0 \end{array}$$

2- Subtraction:-

$$\begin{array}{l} 0 - 0 = 0 \\ 0 - 1 = 1 \quad \text{borrow 1} \\ 1 - 0 = 1 \\ 1 - 1 = 0 \end{array}$$

Example: subtract the binary number (100) from (101)

Solution:

$$\begin{array}{r} 101 \\ - 100 \\ \hline 001 \end{array}$$

Example: subtract the binary number (1101) from (1110)

Solution:

$$\begin{array}{r} \\ \\ - \\ \hline \end{array}$$

Subtraction Using 1's Complement:

Add M to 1's complement of N (subtracted) and check the carry: If an end carry occur, add 1 to the least significant bit. And if an end carry does not occur, take the 1's complement of the number obtained in step 1 and place a negative sign in front.

Example: Given the two binary numbers $X = 1010100$ and $Y = 1000011$, perform the subtraction (a) $X - Y$ and (b) $Y - X$ by using 1's complements.

Solution:

$$(a) X - Y = 1010100 - 1000011$$

$$\begin{array}{r} X = \quad 1010100 \\ 1\text{'s complement of } Y = + \quad 0111100 \\ \text{Sum} = \quad 10010000 \\ \text{End-around carry} = + \quad \underline{\quad 1} \\ \text{Answer: } X - Y = \quad 0010001 \end{array}$$

$$(b) Y - X = 1000011 - 1010100$$

$$\begin{array}{r} Y = \quad 1000011 \\ 1\text{'s complement of } X = + \quad 0101011 \\ \text{Sum} = \quad 1101110 \end{array}$$

There is no end carry. Therefore, the answer is $Y - X = -(1\text{'s complement of } 1101110) = -0010001$.

Subtraction Using 2's Complement:

Apply the 2's complement to the subtracted N and then add it to M, if an end carry occur, discard it. If an end carry does not occur, apply 2's complement on the number that obtained in step 1.

Example: Given the two binary numbers $X = 1010100$ and $Y = 1000011$, perform the subtraction (a) $X - Y$ and (b) $Y - X$ by using 2's complements

Solution:

$$(a) \begin{array}{r} X = \quad 1010100 \\ 2\text{'s complement of } Y = + \quad 0111101 \\ \text{Sum} = \quad 10010001 \\ \text{Discard end carry } 2^7 = - \quad \underline{10000000} \\ \text{Answer: } X - Y = \quad 0010001 \end{array}$$

$$(b) \begin{array}{r} Y = \quad 1000011 \\ 2\text{'s complement of } X = + \quad 0101100 \\ \text{Sum} = \quad 1101111 \end{array}$$

There is no end carry. Therefore, the answer is $Y - X = -(2\text{'s complement of } 1101111) = -0010001$.

3- Multiplication:

$$\begin{array}{l} 0 \times 0 = 0 \\ 0 \times 1 = 0 \\ 1 \times 0 = 0 \\ 1 \times 1 = 1 \end{array}$$

Example: Multiply the two binary numbers $(111)_2$ and $(101)_2$.

$$\begin{array}{r} 111 \\ \times 101 \\ \hline 111 \\ + 0000 \\ \hline 11100 \\ 100011 \end{array}$$

4- Division

Binary division is again similar to its decimal counterpart:

Example: divide the number (11011) on (101)

$$\begin{array}{r} 101 \\ \overline{) 11001} \\ \underline{- 101} \\ 00101 \\ \underline{- 101} \\ 000 \end{array}$$

1.5 Binary Codes

The electronic digital systems like computers, microprocessors etc., are required to process data which may include numbers, alphabets or special characters. The binary system of representation is the most extensively used one in digital systems i.e, digital data is represented, stored and processed as group of binary digits (bits). Hence the numerals, alphabets, special characters and control functions are to be converted into binary format. The process of conversion into binary format is known as binary coding. Several binary codes have developed over the years. Some of them are discussed in this section.

1. Binary coded decimal (BCD).
2. Gray code.
3. ASCII code

1- Binary Coded Decimal (BCD)

Internally, digital computers operate on binary numbers. When interfacing to humans, digital processors, e.g. pocket calculators, communication is decimal-based.

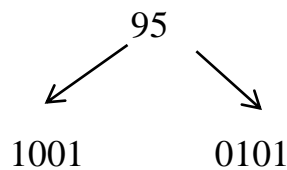
Input is done in decimal then converted to binary for internal processing. For output, the result has to be converted from its internal binary representation to a decimal form.

One commonly used code is the *Binary Coded Decimal* (BCD) code which corresponds to the first 10 binary representations of the decimal digits 0-9. The BCD code requires 4 bits to represent the 10 decimal digits. Since 4 bits may have up to 16 different binary combinations, a total of 6 combinations will be unused

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

Example: Convert $(95)_{10}$ into BCD code .

Solution:



2- Gray Code

The Gray code consists of 16 4-bit code words to represent the decimal Numbers 0 to 15. For Gray code, successive code words differ by only one bit from one to the next as shown in the table and further illustrated in the Figure.

Binary Number to Gray Code Conversion:

The procedures of conversion from binary to gray code are:

- 1- put down the MSB
- 2- start from the MSB, adding without carry each two adjacent bits

Example: convert the $(10110)_2$ into gray code.

Solution:

1	—	+	→	0	—	+	→	1	—	+	→	1	—	+	→	0	Binary
↓				↓				↓				↓					
1				1				0				1					Gray

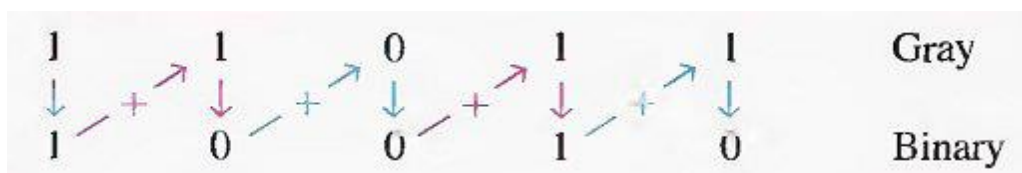
Gray Code to Binary Number Conversion:

The procedure of conversion from gray code to binary are:

- 1- put down the MSB
- 2- start from the MSB adding without carry each result binary bit with the lower gray code bit

Example: convert the $(11011)_{\text{gray}}$ into binary.

Solution:

**3- ASCII Code**

American Standard Codes for Information Interchanging (ASCII) is the most widely used alphanumeric code. It is pronounced as 'ASKEE'. This is basically a 7-bit code and so, it has $2^7 = 128$ possible code groups. The ASCII code can be used to encode both the lowercase and uppercase characters of the alphabet (52 symbols) and some special symbols as well, in addition to the 10 decimal digits. This code is used to exchange the information between input/output device and computers, and stored into the memory.

American Standard Code for Information Interchange (ASCII)

$b_4b_3b_2b_1$	$b_7b_6b_5$							
	000	001	010	011	100	101	110	111
0000	NUL	DLE	SP	0	@	P	`	p
0001	SOH	DC1	!	1	A	Q	a	q
0010	STX	DC2	"	2	B	R	b	r
0011	ETX	DC3	#	3	C	S	c	s
0100	EOT	DC4	\$	4	D	T	d	t
0101	ENQ	NAK	%	5	E	U	e	u
0110	ACK	SYN	&	6	F	V	f	v
0111	BEL	ETB	'	7	G	W	g	w
1000	BS	CAN	(8	H	X	h	x
1001	HT	EM)	9	I	Y	i	y
1010	LF	SUB	*	:	J	Z	j	z
1011	VT	ESC	+	;	K	[k	{
1100	FF	FS	,	<	L	\	l	
1101	CR	GS	-	=	M]	m	}
1110	SO	RS	.	>	N	^	n	~
1111	SI	US	/	?	O	_	o	DEL

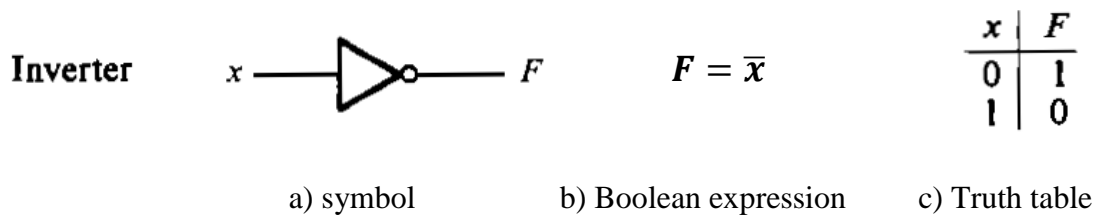
Logic Gates

Introduction:

The logic gate is the basic building block in digital systems. Logic gates operate with binary numbers. Gates are therefore referred to as binary logic gates. All voltages used with logic gates will be either HIGH or LOW. In this lecture, a HIGH voltage will mean a binary 1. A LOW voltage will mean a binary 0. Remember that logic gates are electronic circuits. These circuits will respond only to HIGH voltages (called 1s) or LOW (ground) voltages (called 0s). All digital systems are constructed by using only three basic logic gates. These basic gates are called the AND gate, the OR gate, and the NOT gate.

1- The NOT gate:

a NOT gate is also called an inverter. a NOT gate, or inverter, is an unusual gate. The NOT gate has only one input and one output. Many symbols can be used for NOT gate such as: $\bar{\quad}$, \prime . Fig(1) illustrates the logic symbol for the NOT gate, Boolean expression and the truth table. Boolean expression is a form of symbolic logic that shows how logic gates operate.



Fig(1). The NOT gate symbol, Boolean expression and truth table

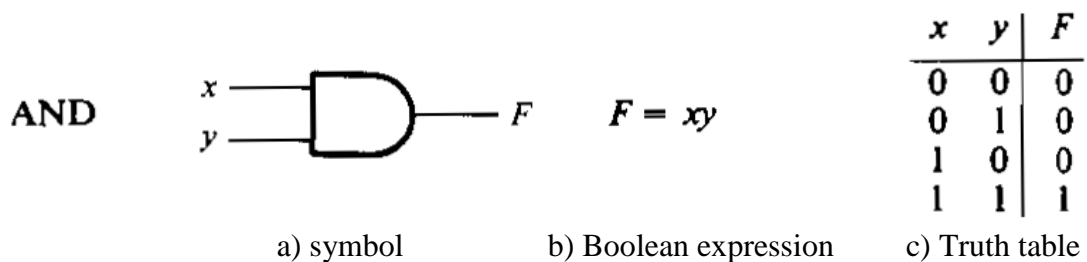
The input is always changed to its opposite. If the input is 0, the NOT gate will give its complement, or opposite, which is 1. If the input to the NOT gate is a 1, the circuit will complement it to give a 0. The double inverted x is equal to the original x .

The laws of Boolean algebra govern how NOT gates operate are:

$$\begin{array}{l}
 \text{If } \bar{0} = 1 \\
 \text{If } x = 0 \\
 \text{If } x = 1 \\
 \bar{\bar{x}} = x
 \end{array}
 \quad
 \begin{array}{l}
 \text{and} \\
 \text{then} \\
 \text{then}
 \end{array}
 \quad
 \begin{array}{l}
 \bar{1} = 0 \\
 \bar{x} = 1 \\
 \bar{x} = 0
 \end{array}$$

2- The AND gate:

The AND gate is called the “all or nothing” gate. The standard logic symbol for the AND gate is drawn in Fig.(2.a). This symbol shows the inputs as x and y . The output is shown as F . This is the symbol for a 2-input AND gate. The Boolean expression of this AND gate is shown in Fig.(2.b) . The truth table for the 2-input AND gate is shown in Fig. (2.c). The inputs are shown as binary digits (bits). Note that only when both inputs x and y are 1 will the output be 1.



Fig(2) The AND gate symbol , Boolean expression and truth table

The Boolean expression reads x AND y equals the output F . The formal laws for the AND function are:

$$x \cdot 1 = x$$

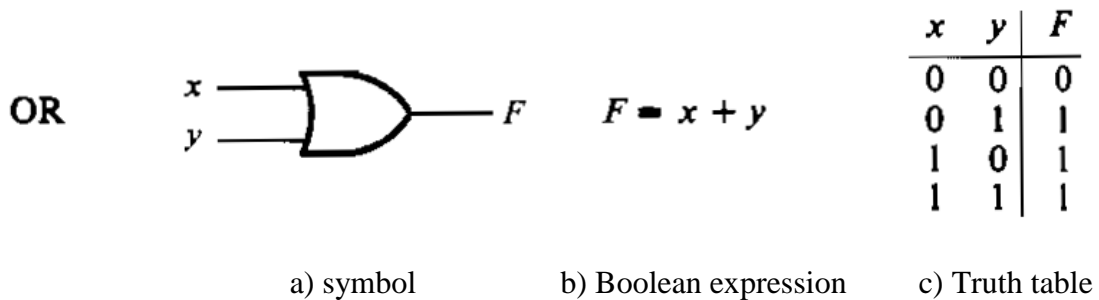
$$x \cdot 0 = 0$$

$$x \cdot x = x$$

$$x \cdot \bar{x} = 0$$

3- The OR gate:

The OR gate is called the “any or all” gate. The standard logic symbol for an OR gate is drawn in Fig (3). The OR gate has two inputs labeled x and y. The output is labeled F. The shorthand Boolean expression for this OR function is given as $x + y = F$. Note that the plus (+) symbol means OR in Boolean algebra. The expression ($x + y = F$) is read as x OR y equals output F. You will note that the plus sign does not mean to add as it does in regular algebra.



Fig(3) The OR gate symbol , Boolean expression and truth table

The formal laws for the OR function are:

$$x + 1 = 1$$

$$x + 0 = x$$

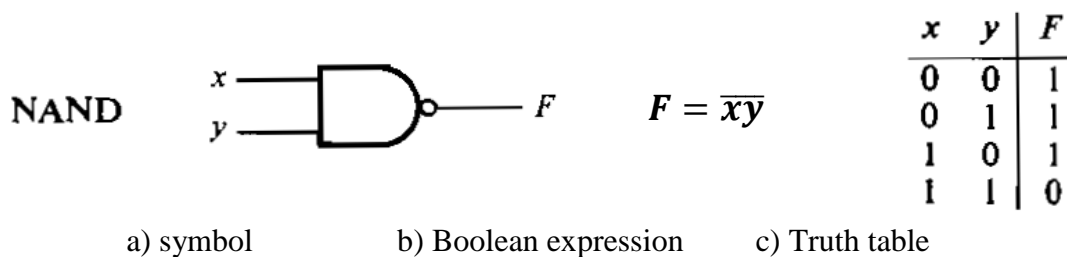
$$x + x = x$$

$$x + \bar{x} = 1$$

4-The NAND gate :

This is implemented from the AND gate with NOT gate , so that it is the complement of the AND gate.

The NAND gate symbol, Boolean expression and truth table are shown in fig(4).




Fig(4) The NAND gate symbol , Boolean expression and truth table

5- The NOR gate :

This is implemented from the OR gate with NOT gate , so that it is the complement of the OR gate.

The NOR gate symbol, Boolean expression and truth table are shown in fig(5).

NOR



$F = \overline{x + y}$

x	y	F
0	0	1
0	1	0
1	0	0
1	1	0


a) symbol b) Boolean expression c) Truth table

Fig(5) The NOR gate symbol , Boolean expression and truth table

6- The Exclusive OR (XOR) gate :

This is implemented from the (OR, NOT, AND) gates , as you can see it's Boolean expression. The XOR gate symbol, Boolean expression and truth table are shown in fig(6).

Exclusive-OR (XOR)



$F = x \oplus y$
 $= x\bar{y} + \bar{x}y$

x	y	F
0	0	0
0	1	1
1	0	1
1	1	0


a) symbol b) Boolean expression c) Truth table

Fig(6) The XOR gate symbol , Boolean expression and truth table

7- The Exclusive NOR (XNOR) gate :

This is the complement of the XOR gate . The XNOR gate symbol, Boolean expression and truth table are shown in fig(7).

Exclusive-NOR or equivalence



$F = x \odot y$
 $= xy + \bar{x}.\bar{y}$

x	y	F
0	0	1
0	1	0
1	0	0
1	1	1

a) symbol b) Boolean expression c) Truth table

Fig(7). The XNOR gate symbol , Boolean expression and truth table

The summary of all logic gates is shown in fig(8)




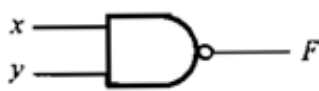


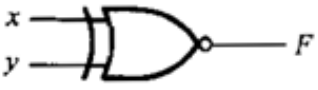
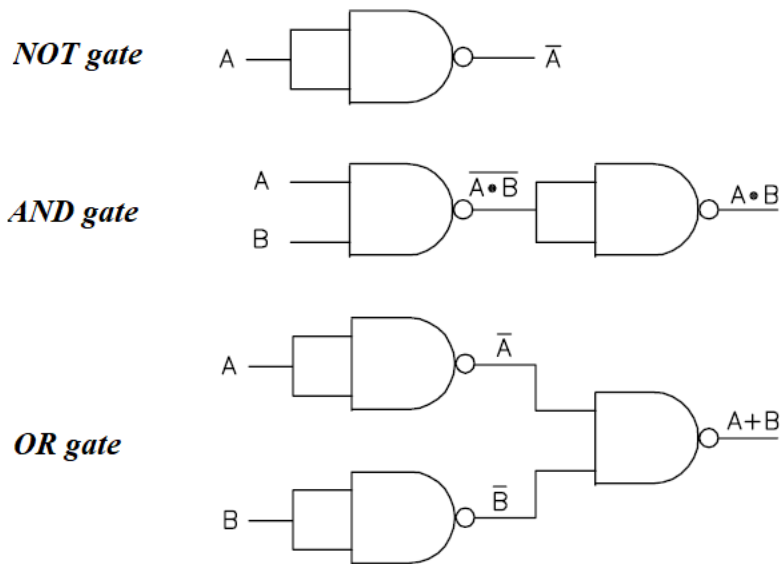
Name	Graphic symbol	Algebraic function	Truth table															
AND		$F = xy$	<table border="1"> <thead> <tr> <th>x</th> <th>y</th> <th>F</th> </tr> </thead> <tbody> <tr> <td>0</td> <td>0</td> <td>0</td> </tr> <tr> <td>0</td> <td>1</td> <td>0</td> </tr> <tr> <td>1</td> <td>0</td> <td>0</td> </tr> <tr> <td>1</td> <td>1</td> <td>1</td> </tr> </tbody> </table>	x	y	F	0	0	0	0	1	0	1	0	0	1	1	1
x	y	F																
0	0	0																
0	1	0																
1	0	0																
1	1	1																
OR		$F = x + y$	<table border="1"> <thead> <tr> <th>x</th> <th>y</th> <th>F</th> </tr> </thead> <tbody> <tr> <td>0</td> <td>0</td> <td>0</td> </tr> <tr> <td>0</td> <td>1</td> <td>1</td> </tr> <tr> <td>1</td> <td>0</td> <td>1</td> </tr> <tr> <td>1</td> <td>1</td> <td>1</td> </tr> </tbody> </table>	x	y	F	0	0	0	0	1	1	1	0	1	1	1	1
x	y	F																
0	0	0																
0	1	1																
1	0	1																
1	1	1																
NOT		$F = \bar{x}$	<table border="1"> <thead> <tr> <th>x</th> <th>F</th> </tr> </thead> <tbody> <tr> <td>0</td> <td>1</td> </tr> <tr> <td>1</td> <td>0</td> </tr> </tbody> </table>	x	F	0	1	1	0									
x	F																	
0	1																	
1	0																	
NAND		$F = \overline{xy}$	<table border="1"> <thead> <tr> <th>x</th> <th>y</th> <th>F</th> </tr> </thead> <tbody> <tr> <td>0</td> <td>0</td> <td>1</td> </tr> <tr> <td>0</td> <td>1</td> <td>1</td> </tr> <tr> <td>1</td> <td>0</td> <td>1</td> </tr> <tr> <td>1</td> <td>1</td> <td>0</td> </tr> </tbody> </table>	x	y	F	0	0	1	0	1	1	1	0	1	1	1	0
x	y	F																
0	0	1																
0	1	1																
1	0	1																
1	1	0																
NOR		$F = \overline{x + y}$	<table border="1"> <thead> <tr> <th>x</th> <th>y</th> <th>F</th> </tr> </thead> <tbody> <tr> <td>0</td> <td>0</td> <td>1</td> </tr> <tr> <td>0</td> <td>1</td> <td>0</td> </tr> <tr> <td>1</td> <td>0</td> <td>0</td> </tr> <tr> <td>1</td> <td>1</td> <td>0</td> </tr> </tbody> </table>	x	y	F	0	0	1	0	1	0	1	0	0	1	1	0
x	y	F																
0	0	1																
0	1	0																
1	0	0																
1	1	0																
Exclusive-OR (XOR)		$F = x \oplus y$ $= x\bar{y} + \bar{x}y$	<table border="1"> <thead> <tr> <th>x</th> <th>y</th> <th>F</th> </tr> </thead> <tbody> <tr> <td>0</td> <td>0</td> <td>0</td> </tr> <tr> <td>0</td> <td>1</td> <td>1</td> </tr> <tr> <td>1</td> <td>0</td> <td>1</td> </tr> <tr> <td>1</td> <td>1</td> <td>0</td> </tr> </tbody> </table>	x	y	F	0	0	0	0	1	1	1	0	1	1	1	0
x	y	F																
0	0	0																
0	1	1																
1	0	1																
1	1	0																
Exclusive-NOR		$F = x \odot y$ $= xy + \bar{x}.\bar{y}$	<table border="1"> <thead> <tr> <th>x</th> <th>y</th> <th>F</th> </tr> </thead> <tbody> <tr> <td>0</td> <td>0</td> <td>1</td> </tr> <tr> <td>0</td> <td>1</td> <td>0</td> </tr> <tr> <td>1</td> <td>0</td> <td>0</td> </tr> <tr> <td>1</td> <td>1</td> <td>1</td> </tr> </tbody> </table>	x	y	F	0	0	1	0	1	0	1	0	0	1	1	1
x	y	F																
0	0	1																
0	1	0																
1	0	0																
1	1	1																

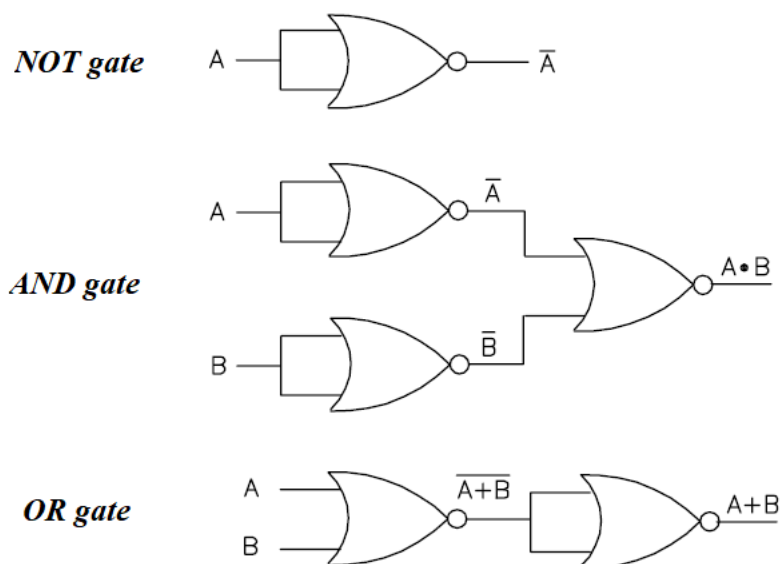
Fig (8) . The logic gates summary

Universality of NAND & NOR Gates

It is possible to implement any logic expression using only NAND gates and no other type of gate. This is because NAND gates, in the proper combination, can be used to perform each of the Boolean operations OR, AND, and NOT.



In a similar manner, it can be shown that NOR gates can be arranged to implement any of the Boolean operations.



Boolean Algebra

In 1854 George Boole introduced a systematic approach of logic and developed an algebraic system to treat the logic functions, which is now called Boolean algebra. In 1938 C.E. Shannon developed a two-valued Boolean algebra called Switching algebra, and demonstrated that the properties of two-valued or bistable electrical switching circuits can be represented by this algebra.

Boolean algebra is a mathematical system for the manipulation of variables that can have one of two values. In digital systems, these values are “on” and “off,” 1 and 0, or “high” and “low.”

The following Huntington postulates are satisfied for the definition of Boolean algebra on a set of elements S together with two binary operators (+) and (·).

1. (a) Closer with respect to the operator (+).
(b) Closer with respect to the operator (·).
2. (a) An identity element with respect to + is designated by 0 *i.e.*, $x + 0 = 0 + x = x$.
(b) An identity element with respect to · is designated by 1 *i.e.*, $x \cdot 1 = 1 \cdot x = x$.

Two-Valued Boolean Algebra

Two-valued Boolean algebra is defined on a set of only two elements, $S = \{0,1\}$, with rules for two binary operators (+) and (·) and inversion or complement as shown in the following operator tables, respectively.

A	B	A + B
0	0	0
0	1	1
1	0	1
1	1	1

A	B	A.B
0	0	0
0	1	0
1	0	0
1	1	1

A	A'
0	1
1	0

The basic rules of Boolean algebra are:-

Postulates and Theorems of Boolean Algebra

Postulate 2	(a)	$x + 0 = x$	(b)	$x \cdot 1 = x$
Postulate 5	(a)	$x + x' = 1$	(b)	$x \cdot x' = 0$
Theorem 1	(a)	$x + x = x$	(b)	$x \cdot x = x$
Theorem 2	(a)	$x + 1 = 1$	(b)	$x \cdot 0 = 0$
Theorem 3, involution		$(x')' = x$		
Postulate 3, commutative	(a)	$x + y = y + x$	(b)	$xy = yx$
Theorem 4, associative	(a)	$x + (y + z) = (x + y) + z$	(b)	$x(yz) = (xy)z$
Postulate 4, distributive	(a)	$x(y + z) = xy + xz$	(b)	$x + yz = (x + y)(x + z)$
Theorem 5, DeMorgan	(a)	$(x + y)' = x'y'$	(b)	$(xy)' = x' + y'$
Theorem 6, absorption	(a)	$x + xy = x$	(b)	$x(x + y) = x$

The proving of theorems can be done by using the **Postulates** or the **truth table** as illustrated in the following :

THEOREM 1(a): $x + x = x$.

Statement	Justification
$x + x = (x + x) \cdot 1$	postulate 2(b)
$= (x + x)(x + x')$	5(a)
$= x + xx'$	4(b)
$= x + 0$	5(b)
$= x$	2(a)

THEOREM 1(b): $x \cdot x = x$.

Statement	Justification
$x \cdot x = xx + 0$	postulate 2(a)
$= xx + xx'$	5(b)
$= x(x + x')$	4(a)
$= x \cdot 1$	5(a)
$= x$	2(b)

THEOREM 2(a): $x + 1 = 1$.

Statement	Justification
$x + 1 = 1 \cdot (x + 1)$	postulate 2(b)
$= (x + x')(x + 1)$	5(a)
$= x + x' \cdot 1$	4(b)
$= x + x'$	2(b)
$= 1$	5(a)

THEOREM 6(a): $x + xy = x$.

Statement	Justification
$x + xy = x \cdot 1 + xy$	postulate 2(b)
$= x(1 + y)$	4(a)
$= x(y + 1)$	3(a)
$= x \cdot 1$	2(a)
$= x$	2(b)

THEOREM 6(b): $x(x + y) = x$ by duality.

Example: prove that $x(y + z) = xy + xz$

Solution:

x	y	z	y + z	x(y + z)	xy	xz	xy + xz
0	0	0	0	0	0	0	0
0	0	1	1	0	0	0	0
0	1	0	1	0	0	0	0
0	1	1	1	0	0	0	0
1	0	0	0	0	0	0	0
1	0	1	1	0	0	1	1
1	1	0	1	1	1	0	1
1	1	1	1	1	1	1	1

Hence, it is proved because the left side is similar to the right side

Example: prove that $x + yz = (x+y)(x+z)$

Solution:

x	y	z	yz	x + yz	x + y	x + z	(x+y)(x+z)
0	0	0	0	0	0	0	0
0	0	1	0	0	0	1	0
0	1	0	0	1	1	0	0
0	1	1	1	1	1	1	1
1	0	0	0	1	1	1	1
1	0	1	0	1	1	1	1
1	1	0	0	1	1	1	1
1	1	1	1	1	1	1	1

Hence, it is proved because the left side is similar to the right side

Operator Precedence

The operator precedence for evaluating Boolean expressions is (1) parentheses, (2) NOT, (3) AND, and (4) OR. In other words, expressions inside parentheses must be evaluated before all other operations. The next operation that holds precedence is the complement, and then follows the AND and, finally, the OR.

Boolean Function

A Boolean function is a relation between the binary inputs and the binary outputs. The value of a function (output) may be 0 or 1, depending on the values of inputs present in the Boolean function. Boolean Function can be described by:

- 1- a truth table
- 2- Boolean equation,
- 3- a logic diagram

1- Truth table

Truth table for a function is a list of all combinations of 1's and 0's that can be assigned to the binary variables and a list that shows the value of the function for each binary combination.

For n variables, there are 2^n rows (states)

For example, when the number of variables (inputs) $n=3$, then the number of rows (states) = $2^3 = 8$ as shown in this table:

A	B	C	F
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	0
1	0	1	0
1	1	0	1
1	1	1	0

2- Boolean equation (Boolean function form):

Boolean equation consists of a binary variable identifying the function (output) followed by an equal sign and a Boolean expression formed with binary variables, the two binary operators AND and OR, one unary operator NOT, and parentheses. When a Boolean expression is implemented with logic gates, each literal in the function is designated as input to the gate. The literal may be a primed or unprimed variable. For example, the Boolean equation of the truth table above is:

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

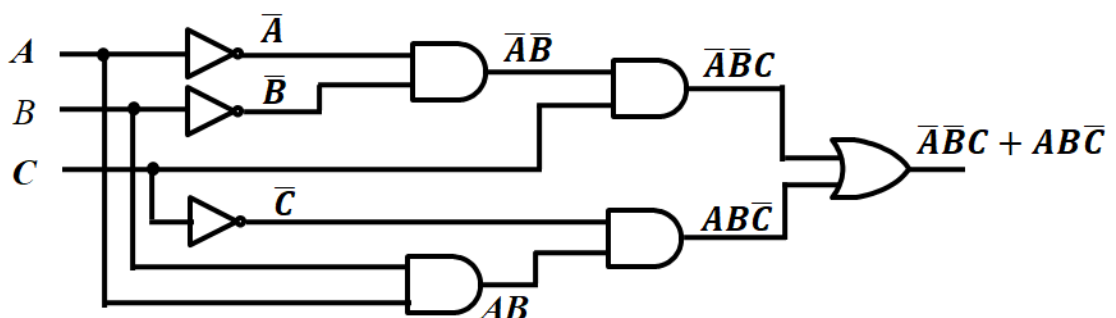
Where F is the function (output)

A, B, C are the input variables (literals)

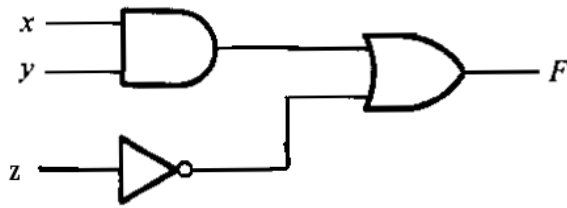
3- Logic diagram (circuit diagram):

The logic diagram composed of logic gates in which are interconnected by wires that carry logic signals. The figure below shows the logic diagram of the Boolean equation

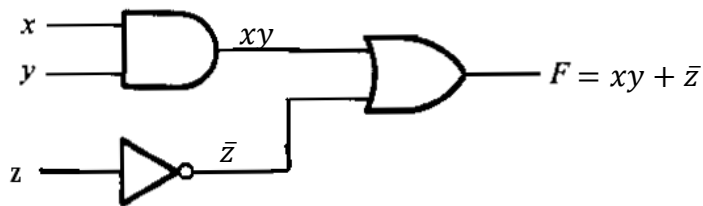
$$F = \bar{A}\bar{B}C + AB\bar{C}$$



Example: Write the Boolean expression for the logic diagram shown.



Solution:



The Boolean expression of this circuit is :

$$F = xy + \bar{z}$$

Simplification using Boolean algebra:

Minimization of the number of literals and the number of terms leads to less complex circuits as well as less number of gates, which should be a designer's aim. There are several methods to minimize the Boolean function such as Boolean algebra and Karnaugh map (K-Map). Here, simplification or minimization of complex algebraic expressions will be shown with the help of postulates and theorems of Boolean algebra.

Example : Simplify the following Boolean expression.

$$F = xyz + \bar{x}y + xy\bar{z}$$

Solution:

$$F = xy(z + \bar{z}) + \bar{x}y$$

$$F = xy + \bar{x}y$$

$$F = y(x + \bar{x})$$

$$F = y$$

Example : Simplify the following Boolean expression.

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

Solution:

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

$$F = A + B\bar{A} \quad \text{because } (B + \bar{B}) = 1$$

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

$$F = A + B \quad \text{because } (A + \bar{A}) = 1$$

Example: Simplify the following Boolean expression.

$$F = \overline{((\bar{x} + y) \cdot \bar{z})}$$

Solution: using De Morgan theorem

$$F = \overline{(\bar{x} + y)} + \bar{\bar{z}}$$

$$F = \bar{\bar{x}} \cdot \bar{y} + z$$

$$F = x \cdot \bar{y} + z$$

Example: simplify and draw the logic diagram of

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

Solution:

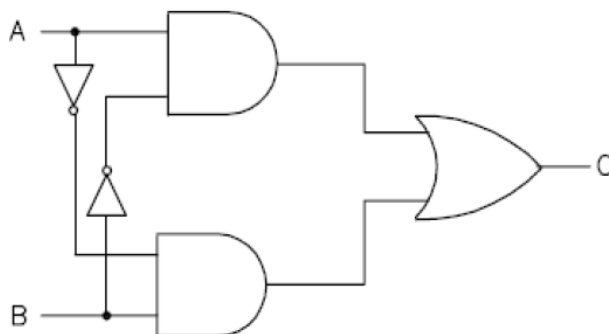
$$C = (A + B)\bar{A}\bar{B} \quad \text{applying DeMorgan theorem}$$

$$C = (A + B)(\bar{A} + \bar{B}) \quad \text{distributive}$$

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

$$C = 0 + A\bar{B} + B\bar{A} + 0 \quad \text{because } A\bar{A} = 0 \text{ and } B\bar{B} = 0$$

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



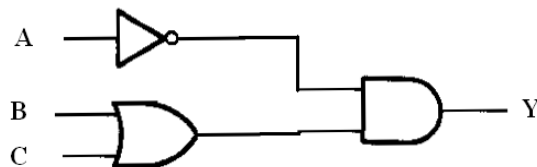
Example : Draw the Boolean expression

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

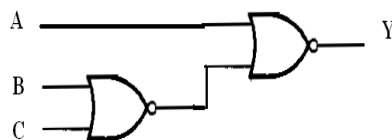
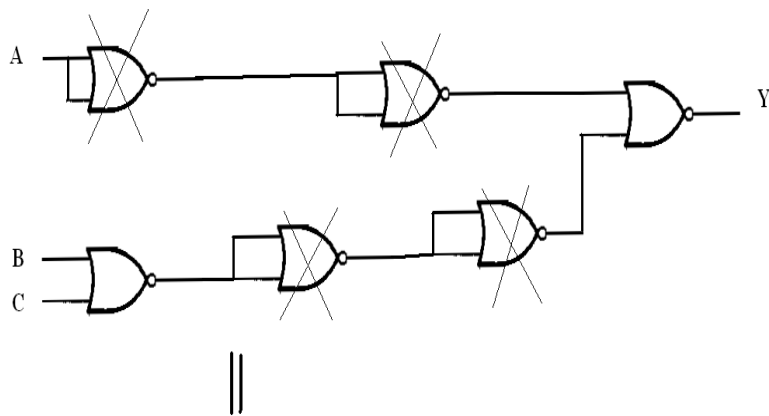
- a) using basic logic gates.
- b) using NOR gates only.

Solution :

a)



b)



Boolean Function Forms

We will get four Boolean product terms by combining two variables (literals) x and y with logical AND operation. These Boolean product terms are called as **Minterms** or **product terms**. The minterms are $\bar{x}\bar{y}$, $\bar{x}y$, $x\bar{y}$, xy .

Similarly, we will get four Boolean sum terms by combining two variables x and y with logical OR operation. These Boolean sumterms are called as **Maxterms** or **sum terms**. The Maxterms are $x + y$, $x + \bar{y}$, $\bar{x} + y$, $\bar{x} + \bar{y}$.

The following table shows the representation of Minterms and Maxterms for 2 variables.

x	y	Minterms	Maxterms
0	0	$m_0 = \bar{x}\bar{y}$	$M_0 = x + y$
0	1	$m_1 = \bar{x}y$	$M_1 = x + \bar{y}$
1	0	$m_2 = x\bar{y}$	$M_2 = \bar{x} + y$
1	1	$m_3 = xy$	$M_3 = \bar{x} + \bar{y}$

If the binary variable is '0', then it is represented as complement of variable in Minterm and as the variable itself in Maxterm. Similarly, if the binary variable is '1', then it is represented as complement of variable in Maxterm and as the variable itself in Minterm.

a) Canonical forms

A truth table consists of a set of inputs and outputs. If there are 'n' input variables, then there will be 2^n possible combinations with zeros and ones. So the value of each output variable depends on the combination of input variables. So, each output variable will have '1' for some combination of input variables and '0' for some other combination of input variables. Therefore, we can express each output variable in following two ways.

- 1- Canonical sum of product (SoP) form
- 2- Canonical product of sum (PoS) form

1- Canonical SoP form

Canonical SoP form means Canonical Sum of Products form. In this form, each product term **contains all literals**. So, these product terms are nothing but the minterms. Hence, canonical SoP form is also called as **sum of Minterms** form.

- a- identify the minterms for which, the output variable is one
- b- do the logical OR of those minterms in order to get the Boolean expression function corresponding to that output variable.

Example: Drive the canonical SoP form from the following truth table

Inputs			Output	
A	B	C	F	
0	0	0	1	$\bar{A}\bar{B}\bar{C}$
0	0	1	0	
0	1	0	0	
0	1	1	1	$\bar{A}BC$
1	0	0	1	$A\bar{B}\bar{C}$
1	0	1	0	
1	1	0	1	$AB\bar{C}$
1	1	1	1	ABC

Here, the output F is '1' for five combinations of inputs. The corresponding Minterms are $\bar{A}\bar{B}\bar{C}$, $\bar{A}BC$, $A\bar{B}\bar{C}$, $AB\bar{C}$, ABC . By doing logical OR of these five minterms, we will get the Boolean function of output F.

Therefore, the Boolean function of output is,

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

This is the **canonical SoP form** of output, F. We can also represent this function in following two notations.

$$F(A,B,C) = m_0 + m_3 + m_4 + m_6 + m_7$$

$$F(A,B,C) = \sum (0, 3, 4, 6, 7)$$

2- Canonical PoS form

In this form, **each sum term contains all literals**. So, these sum terms are nothing but the Maxterms. Hence, canonical PoS form is also called as **product of Maxterms** form.

- identify the Maxterms for which, the output variable is zero
- do the logical AND of those Maxterms in order to get the Boolean expression function corresponding to that output variable.

Example: Drive the canonical PoS form from the following truth table

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

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

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

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

Here, the output F is '0' for three combinations of inputs. The corresponding maxterms are $A + B + \bar{C}$, $A + \bar{B} + C$, $\bar{A} + B + \bar{C}$. By doing logical AND of these three maxterms, we will get the Boolean function of output F.

Therefore, the Boolean function of output is,

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

This is the **canonical PoS form** of output, F. We can also represent this function in following two notations.

$$F(A, B, C) = M_1 \cdot M_2 \cdot M_5$$

$$F(A, B, C) = \prod (1, 2, 5)$$

Note: the sequence of literals(letters) must be the same in truth table and in Boolean equation.

b) Standard forms

We discussed two canonical forms of representing the Boolean outputs. Similarly, there are two standard forms of representing the Boolean outputs. These are the simplified version of canonical forms.

- Standard SoP form: such as $F = AB + \bar{A}C + \bar{B}\bar{C}$
- Standard PoS form: such as $F = (A + C)(A + B + C)(\bar{A} + B)$

The main **advantage** of standard forms is that the number of inputs applied to logic gates can be minimized. Sometimes, there will be reduction in the total number of logic gates required.

Conversion from Canonical SoP form to Standard SoP form

In this form, each product term need not contain all literals. So, the product terms **may** or **may not** be the Minterms. Therefore, the Standard SoP form is the simplified form of canonical SoP form.

We will get Standard SoP form of output variable in two steps.

- Get the canonical SoP form of output variable
- Simplify the above Boolean function, which is in canonical SoP form.

Sometimes, it may not possible to simplify the canonical SoP form. In that case, both canonical and standard SoP forms are same.

Example: convert the canonical SoP expression to standard SoP form

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

Solution:

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

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

$$F = \bar{A}\bar{B}\bar{C} + AB \quad \text{because } \bar{C} + C = 1$$

Hence the standard SoP expression of the given function is

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

Conversion from standard SoP form to canonical SoP form

The canonical SoP form of a logic function can be obtained by using the following procedure:

- 1) Check each term in the given logic function. Retain if it is a minterm, continue to examine the next term in the same manner.
- 2) Examine for the variables that are missing in each product which is not a min term. If the missing variable in the minterm is X, multiply that minterm with $(X+X')$.
- 3) Multiply all the products and discard the redundant terms.

Example. Obtain the canonical SoP form of the following function:

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

Solution. The given function contains two variables A and B. The variable B is missing from the first term of the expression and the variable A is missing from the second term of the expression. Therefore, the first term is to be multiplied by $(B + \bar{B})$ and the second term is to be multiplied by $(A + \bar{A})$ as demonstrated below.

$$\begin{aligned} F(A, B) &= A + B \\ &= A \cdot 1 + B \cdot 1 \\ &= A(B + \bar{B}) + B(A + \bar{A}) \\ &= AB + A\bar{B} + BA + B\bar{A} \\ &= AB + A\bar{B} + B\bar{A} \quad \text{where } AB + BA = AB \end{aligned}$$

Hence the canonical SoP expression of the given function is

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

Example. Obtain the canonical sum of product form of the following function.

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

Solution. Here neither the first term nor the second term is Minterm. The given function contains three variables A, B, and C. The variables B and C are missing from the first term of the expression and the variable A is missing from the second term of the expression. Therefore, the first term is to be multiplied by $(B + \bar{B})$ and $(C + \bar{C})$. The second term is to be multiplied by $(A + \bar{A})$. This is demonstrated below.

$$\begin{aligned}
F(A, B, C) &= A + BC \\
&= A.1.1 + BC.1 \\
&= A(B + \bar{B})(C + \bar{C}) + BC(A + \bar{A}) \\
&= (AB + A\bar{B})(C + \bar{C}) + BCA + BC\bar{A} \\
&= ABC + AB\bar{C} + A\bar{B}C + A\bar{B}\bar{C} + BCA + BC\bar{A} \\
&= ABC + AB\bar{C} + A\bar{B}C + A\bar{B}\bar{C} + BC\bar{A} \quad \text{where } ABC + BCA = ABC
\end{aligned}$$

Hence the canonical SoP expression of the given function is

$$F(A, B, C) = ABC + AB\bar{C} + A\bar{B}C + A\bar{B}\bar{C} + BC\bar{A}$$

Conversion from standard PoS form to canonical PoS form

The canonical product of sums form of a logic function can be obtained by using the following procedure.

- 1) Check each term in the given logic function. Retain it if it is a maxterm, continue to examine the next term in the same manner.
- 2) Examine for the variables that are missing in each sum term that is not a maxterm. If the missing variable in the maxterm is X, add that maxterm with (X \bar{X}).
- 3) Expand the expression using the properties and postulates as described earlier and discard the redundant terms.

Example. Obtain the canonical product of the sum form of the following function.

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

Solution. Now, in the above expression, C is missing from the first term and B is missing from the second term. Hence $C\bar{C}$ is to be added with the first term and $B\bar{B}$ is to be added with the second term as shown below.

$$\begin{aligned}
F(A, B, C) &= (A + \bar{B} + 0)(A + C + 0) \\
&= (A + \bar{B} + C\bar{C})(A + C + B\bar{B}) \\
&= (A + \bar{B} + C)(A + \bar{B} + \bar{C})(A + C + B)(A + C + \bar{B}) \\
&= (A + \bar{B} + C)(A + \bar{B} + \bar{C})(A + C + B)
\end{aligned}$$

Hence the canonical PoS expression for the given function is

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

Simplify the Boolean Function using Karnaugh Map (K-Map)

The second method that used to simplify the Boolean function is the Karnaugh map. K-map basically deals with the technique of inserting the values of the output variable in cells within a rectangle or square grid according to a definite pattern. The number of cells in the K-map is determined by the number of input variables and is mathematically expressed as two raised to the power of the number of input variables, i.e., 2^n , where the number of input variables is n .

Thus, to simplify a logical expression with **two inputs**, we require a K-map with ($2^2 = 4$) cells. A **four-input** logical expression would lead to a ($2^4 = 16$) celled-K-map, and so on.

Advantages of K-Maps

- 1- The K-map simplification technique is simpler and less error-prone compared to the method of solving the logical expressions using Boolean laws.
- 2- It prevents the need to remember each and every Boolean algebraic theorem.
- 3- It involves fewer steps than the algebraic minimization technique to arrive at a simplified expression.
- 4- K-map simplification technique always results in minimum expression if carried out properly.

Disadvantages of K-Maps

- 1- As the number of variables in the logical expression increases, the K-map simplification process becomes complicated.
- 2- The minimum logical expression arrived by using the K-map simplification procedure may or may not be unique depending on the choices made while forming the groups

K-mapping & Minimization Steps

Step 1: generate K-map based on the number of input variables n

- Put a 1 in all specified minterms
- Put a 0 in all other boxes (optional)

Step 2: group all adjacent 1s without including any 0s. All groups must be rectangular and contain a “power-of-2” number of 1s 1, 2, 4, 8, 16, 32, ...

Step 3: define product terms using variables common to all minterms in group

Step 4: sum all essential groups plus a minimal set of remaining groups to obtain a minimum SOP.

1- Two variables K-Map

Number of input variables are 2

Hence the number of squares = $2^n = 2^2 = 4$

Inputs A B		Decimal equivalent	Minterms	Output F
0	0	0	m_0 $\overline{A}\overline{B}$	
0	1	1	m_1 $\overline{A}B$	
1	0	2	m_2 $A\overline{B}$	
1	1	3	m_3 AB	

And K-Map of two variables is:

	\overline{B}	B
	0	1
\overline{A}	0	1
A	2	3

Example: simplify the Boolean expression by using K-Map

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

Solution:

Number of input variables are 2

Hence the number of squares = $2^n = 2^2 = 4$

	\bar{B}	B
	0	1
\bar{A}	0	1
A	0	1

A Karnaugh map for the expression $F = \bar{A}B + AB$. The map is a 2x2 grid. The columns are labeled \bar{B} (0) and B (1). The rows are labeled \bar{A} (0) and A (1). The cells contain values: top-left (0), top-right (1), bottom-left (0), and bottom-right (1). A blue circle groups the two '1's in the right column (minterms 1 and 3). Red numbers 0, 1, 2, and 3 are placed in the top-right, bottom-right, bottom-left, and top-left cells respectively.

$$F = B$$

Example: simplify the Boolean expression by using K-Map

$$F(A, B) = \sum m(2, 0, 3)$$

Solution:

Number of input variables are 2

Hence the number of squares = $2^n = 2^2 = 4$

	\bar{B}	B
	0	1
\bar{A}	1	0
A	1	1

A Karnaugh map for the expression $F(A, B) = \sum m(2, 0, 3)$. The map is a 2x2 grid. The columns are labeled \bar{B} (0) and B (1). The rows are labeled \bar{A} (0) and A (1). The cells contain values: top-left (1), top-right (0), bottom-left (1), and bottom-right (1). A blue circle groups the two '1's in the left column (minterms 0 and 2). An orange circle groups the two '1's in the bottom row (minterms 2 and 3). Red numbers 0, 1, 2, and 3 are placed in the top-right, bottom-right, bottom-left, and top-left cells respectively.

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

Example: simplify the Boolean expression by using K-Map

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

Solution:

Number of input variables are 2

Hence the number of squares = $2^n = 2^2 = 4$

	\bar{B}	B
	0	1
\bar{A}	0 1	1 1
A	2 0	3 0

$$F = \bar{A}$$

Example: simplify the Boolean expression by using K-Map

$$F(A, B) = \sum m(0, 3)$$

Solution:

Number of input variables are 2

Hence the number of squares = $2^n = 2^2 = 4$

	\bar{B}	B
	0	1
\bar{A}	0 1	1 0
A	2 0	3 1

$$F(A, B) = \sum m(0, 3) = \bar{A}\bar{B} + AB$$

2- Three Variables K-Map

Number of input variables are 3

Hence the number of squares = $2^n = 2^3 = 8$

The truth table is

Inputs			Decimal equivalent	Minterms		Output F
A	B	C				
0	0	0	0	m ₀	$\overline{A}\overline{B}\overline{C}$	
0	0	1	1	m ₁	$\overline{A}\overline{B}C$	
0	1	0	2	m ₂	$\overline{A}B\overline{C}$	
0	1	1	3	m ₃	$\overline{A}BC$	
1	0	0	4	m ₄	$A\overline{B}\overline{C}$	
1	0	1	5	m ₅	$A\overline{B}C$	
1	1	0	6	m ₆	$AB\overline{C}$	
1	1	1	7	m ₇	ABC	

And the K-Map of three variables is:

	$\overline{B}\overline{C}$ 00	$\overline{B}C$ 01	BC 11	$B\overline{C}$ 10
\overline{A} 0	0	1	3	2
A 1	4	5	7	6

Example: simplify the Boolean expression by using K-Map

$$F(A, B, C) = \overline{A}\overline{B}\overline{C} + \overline{A}BC + \overline{A}B\overline{C}$$

Solution:

Number of input variables are 3

Hence the number of squares = $2^n = 2^3 = 8$

	$\overline{B}\overline{C}$ 00	$\overline{B}C$ 01	BC 11	$B\overline{C}$ 10
\overline{A} 0	1 ⁰	0 ¹	1 ³	1 ²
A 1	0 ⁴	0 ⁵	0 ⁷	0 ⁶

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

Example: simplify the Boolean expression by using K-Map

$$F(A, B, C) = \sum m(0, 3, 7, 6)$$

Solution:

Number of input variables are 3

Hence the number of squares = $2^n = 2^3 = 8$

	$\overline{B}\overline{C}$ 00	$\overline{B}C$ 01	BC 11	$B\overline{C}$ 10
\overline{A} 0	1 ⁰	0 ¹	1 ³	0 ²
A 1	0 ⁴	0 ⁵	1 ⁷	1 ⁶

$$F(A, B, C) = \overline{A}\overline{B}\overline{C} + BC + AB$$

3- Four Variables K-map

Number of input variables are 4

Hence the number of squares = $2^n = 2^4 = 16$

The truth table is

Inputs				Decimal equivalent	Minterms		Output F
A	B	C	D				
0	0	0	0	0	m_0	$\overline{A}\overline{B}\overline{C}\overline{D}$	
0	0	0	1	1	m_1	$\overline{A}\overline{B}\overline{C}D$	
0	0	1	0	2	m_2	$\overline{A}\overline{B}C\overline{D}$	
0	0	1	1	3	m_3	$\overline{A}\overline{B}CD$	
0	1	0	0	4	m_4	$\overline{A}B\overline{C}\overline{D}$	
0	1	0	1	5	m_5	$\overline{A}B\overline{C}D$	
0	1	1	0	6	m_6	$\overline{A}BC\overline{D}$	
0	1	1	1	7	m_7	$\overline{A}BCD$	
1	0	0	0	8	m_8	$A\overline{B}\overline{C}\overline{D}$	
1	0	0	1	9	m_9	$A\overline{B}\overline{C}D$	
1	0	1	0	10	m_{10}	$A\overline{B}C\overline{D}$	
1	0	1	1	11	m_{11}	$A\overline{B}CD$	
1	1	0	0	12	m_{12}	$AB\overline{C}\overline{D}$	
1	1	0	1	13	m_{13}	$AB\overline{C}D$	
1	1	1	0	14	m_{14}	$ABC\overline{D}$	
1	1	1	1	15	m_{15}	$ABCD$	

And the K-Map of four variables is:

	$\overline{C}\overline{D}$ 00	$\overline{C}D$ 01	CD 11	$C\overline{D}$ 10
$\overline{A}\overline{B}$ 00	0	1	3	2
$\overline{A}B$ 01	4	5	7	6
AB 11	12	13	15	14
$A\overline{B}$ 10	8	9	11	10

Example: simplify the Boolean expression by using K-Map

$$F(A, B, C, D) = \bar{A}\bar{B}\bar{C}\bar{D} + \bar{A}\bar{B}C\bar{D} + A\bar{B}\bar{C}\bar{D} + A\bar{B}C\bar{D} + \bar{A}BCD + AB\bar{C}\bar{D}$$

Solution: Number of input variables are 4
 Hence the number of squares = $2^n = 2^4 = 16$

	$\bar{C}\bar{D}$ 00	$\bar{C}D$ 01	CD 11	$C\bar{D}$ 10
$\bar{A}\bar{B}$ 00	0 1	1 0	3 0	2 1
$\bar{A}B$ 01	4 0	5 0	7 1	6 0
AB 11	12 0	13 1	15 0	14 0
$A\bar{B}$ 10	8 1	9 0	11 0	10 1

$$F(A, B, C, D) = \bar{B}\bar{D} + \bar{A}BCD + AB\bar{C}\bar{D}$$

Example: simplify the Boolean expression by using K-Map

$$F(A, B, C, D) = \sum m(0, 2, 4, 6, 12, 14, 15, 8, 10)$$

Solution: Number of input variables are 4
 Hence the number of squares = $2^n = 2^4 = 16$

	$\bar{C}\bar{D}$ 00	$\bar{C}D$ 01	CD 11	$C\bar{D}$ 10
$\bar{A}\bar{B}$ 00	0 1	1 0	3 0	2 1
$\bar{A}B$ 01	4 1	5 0	7 0	6 1
AB 11	12 1	13 0	15 1	14 1
$A\bar{B}$ 10	8 1	9 0	11 0	10 1

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

K-Map with Don't Care Conditions

In certain cases some of the minterms may never occur or it may not matter what happens if they do

- In such cases we fill in the Karnaugh map with an X that meaning don't care
- When minimizing an X is like a "joker"
- X can be 0 or 1 - whatever helps best with the minimization

Example: simplify the Boolean expression by using K-Map

$$F(A, B, C, D) = \sum m(3, 7, 9, 11) + \sum d(1, 5, 12, 14)$$

Solution: Number of input variables are 4

Hence the number of squares = $2^n = 2^4 = 16$

	$\bar{C}\bar{D}$ 00	$\bar{C}D$ 01	CD 11	$C\bar{D}$ 10
$\bar{A}\bar{B}$ 00	0	X	1	0
$\bar{A}B$ 01	0	X	1	0
AB 11	X	0	0	X
$A\bar{B}$ 10	0	1	1	0

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

Logic Circuits

The digital system consists of two types of circuits, namely

- 1- Combinational Logic Circuits.
- 2- Sequential Logic Circuits.

1- Combinational Logic Circuits:

A combinational circuit consists of logic gates whose outputs at any time are determined from only the present combination of inputs without regard to previous inputs or previous state of outputs. The design of Combinational logic circuit depending on the derivation of the Boolean expression from the truth table base on **sum of product** or **product of sum**.

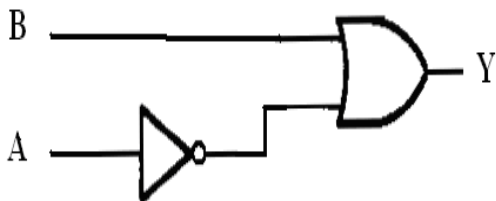
Design Procedures using POS are :-

- 1- Write the truth table for all input states which equal to (2^n) where n is the number of inputs.
- 2- Write the expression for each (Logic 0) output with OR gate
- 3- Write the overall output expression by AND ing the terms in step 2 and if it is possible Simplify this expression.
- 4- Implement this expression using logic gates

Example : Derive the Boolean expression from the following truth table using POS and draw the logic diagram.

Inputs		Output
B	A	Y
0	0	1
0	1	0
1	0	1
1	1	1

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



Design Procedures using SOP are :-

- 1- Write the truth table for all input states which equal to (2^n) where n is the number of inputs.
- 2- Write the expression for each (Logic 1) output with AND gate
- 3- Write the overall output expression by OR ing the terms in step 2 and if it is possible Simplify this expression.
- 4- Implement this expression using logic gates

Example: Design a logic circuit that has 3 inputs and gives a (logic 1) output when the binary input value less than or equal 2.

Sol: number of inputs =3 ; number of states = $2^3 = 8$

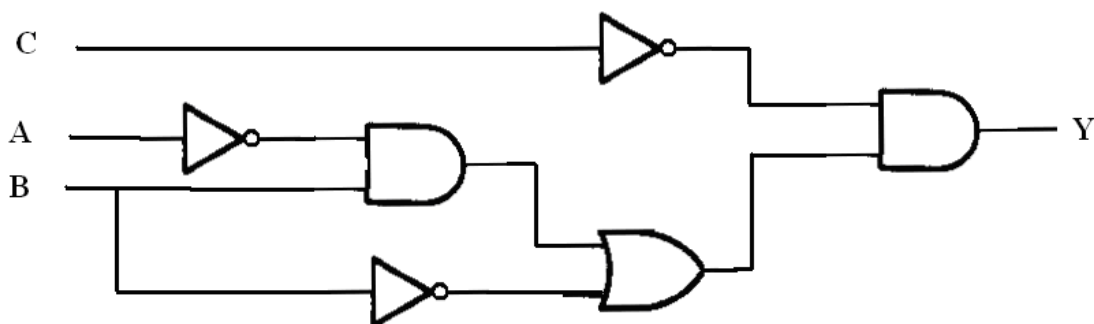
Inputs			Output
C	B	A	Y
0	0	0	1
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	0
1	0	1	0
1	1	0	0
1	1	1	0

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

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

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

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



Common Functions of Combinational Logic

There are many combinational logic circuits or diagrams commonly used in all logic systems such as:

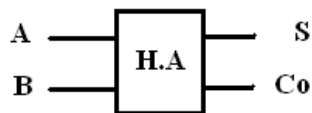
- 1- Adders & Subtractors
- 2- Magnitude Comparators
- 3- Multiplexer and demultiplexers
- 4- Decoders and Encoders

Adders

Arithmetic operations are among the basic functions of a digital computer. Addition of two binary digits is the most basic arithmetic operation. The simple addition consists of four possible elementary operations, which are $0+0 = 0$, $0+1 = 1$, $1+0 = 1$, and $1+1 = 0$ with carry one. The two types of adders are :-

a) Half adder (HA) :

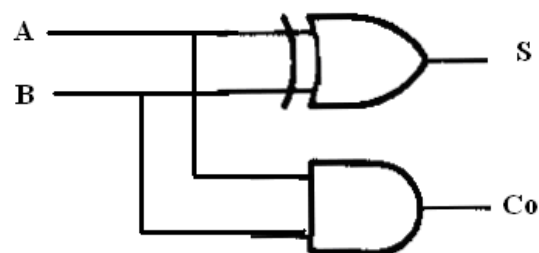
A half adder is a multiple outputs combinational logic circuit which add two bits of binary data **without carry**, producing a sum (S) and a carry out (Co).



Symbol of half adder (H.A)

B	A	S	Co
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

Truth Table of half adder (H.A)



Circuit diagram of half adder (H.A)

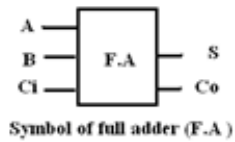
$$S = \bar{B}A + B\bar{A}$$

$$S = A \oplus B$$

$$C_o = AB$$

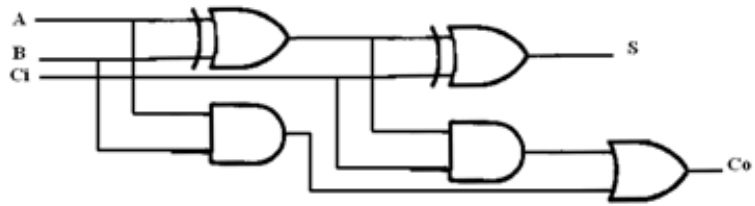
b) Full adder (FA) :

A full adder is a multiple outputs combinational logic circuit which add two bits of binary data **with carry input (C_i)**, producing a sum (S) and a carry out (C_o).



C _i	B	A	S	C _o
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

Truth Table of full adder (F.A)



Circuit diagram of full adder (F.A)

$$S = A\bar{B}\bar{C}_i + \bar{A}B\bar{C}_i + \bar{A}\bar{B}C_i + ABC_i$$

$$S = \bar{C}_i(A\bar{B} + \bar{A}B) + C_i(\bar{A}\bar{B} + AB)$$

$$S = \bar{C}_i(A \oplus B) + C_i(\overline{A \oplus B})$$

$$S = A \oplus B \oplus C_i$$

$$C_o = A\bar{B}\bar{C}_i + \bar{A}B\bar{C}_i + \bar{A}\bar{B}C_i + ABC_i$$

$$C_o = AB(\bar{C}_i + C_i) + (\bar{A}\bar{B} + \bar{A}B)C_i$$

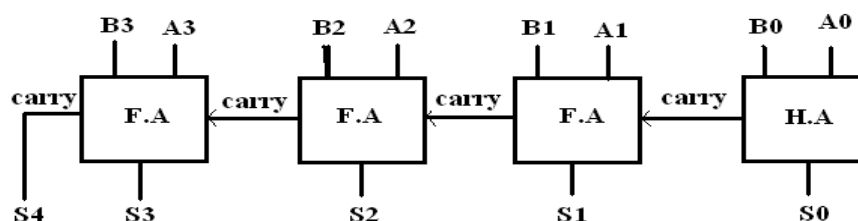
$$C_o = AB + (A \oplus B)C_i$$

Parallel adder :

How can we add two binary numbers of 4 bits

N	A3	A2	A1	A0	
M	B3	B2	B1	B0	+
S	S4	S3	S2	S1	S0

We need 3 full adders and 1 half adder



Subtractors

Subtraction is the other basic function of arithmetic operations of information-processing tasks of digital computers. Similar to the addition function, subtraction of two binary digits consists of four possible elementary operations, which are $0-0 = 0$, $0-1 = 1$ with borrow of 1, $1-0 = 1$, and $1-1 = 0$. There are two types of subtractor

a) Half Subtractor

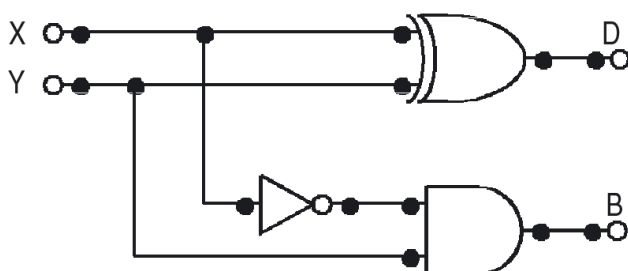
A half-subtractor has two inputs and two outputs. Let the input be designated as X and Y respectively, and output functions be designated as D for difference and B for borrow. The truth table of the function $X-Y$ is as follows

Input variables		Output variables	
X	Y	D	B
0	0	0	0
0	1	1	1
1	0	1	0
1	1	0	0

$$D = \bar{X}Y + X\bar{Y}$$

$$D = X \oplus Y$$

$$B = \bar{X}Y$$



b) Full Subtractor

A combinational circuit of full-subtractor performs the operation of subtraction of three bits—the minuend, subtrahend, and borrow generated from the subtraction operation of previous significant digits and produces the outputs difference and borrow. Let us designate the input variables minuend as X, subtrahend as Y, and previous borrow as Z, and outputs difference as D and borrow as B. Eight different input combinations are possible for three input variables. The truth table of X- Y is shown below

Input variables			Outputs	
X	Y	Z	D	B
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

$$D = \bar{X}\bar{Y}Z + \bar{X}Y\bar{Z} + X\bar{Y}\bar{Z} + XYZ$$

$$D = \bar{Z}(\bar{X}Y + X\bar{Y}) + Z(\bar{X}\bar{Y} + XY)$$

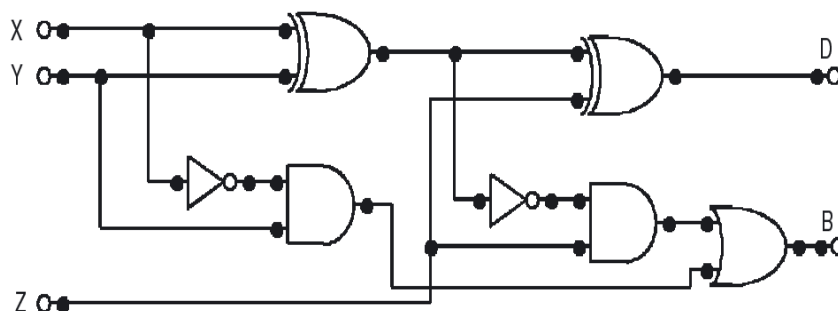
$$D = \bar{Z}(X \oplus Y) + Z\overline{(X \oplus Y)}$$

$$D = X \oplus Y \oplus Z$$

$$B = \bar{X}\bar{Y}Z + \bar{X}Y\bar{Z} + \bar{X}YZ + XYZ$$

$$B = \bar{X}Y(\bar{Z} + Z) + Z(\bar{X}\bar{Y} + XY)$$

$$B = \bar{X}Y + Z\overline{(X \oplus Y)}$$



Magnitude Comparator

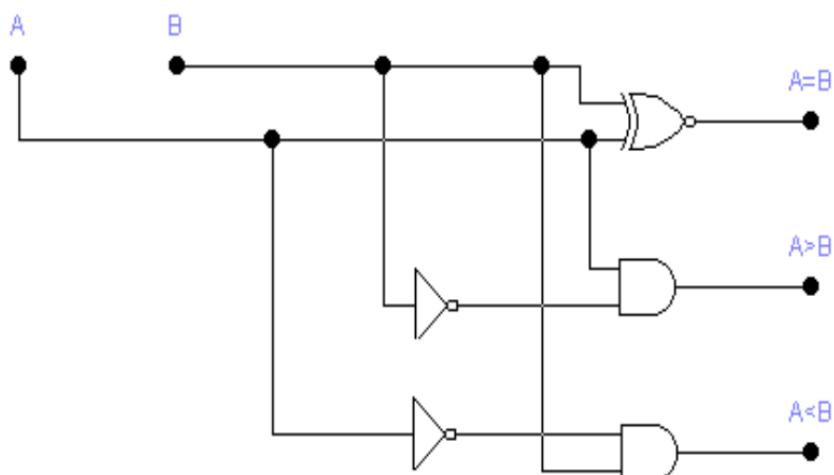
A *magnitude comparator* is one of the useful combinational logic networks and has wide applications. It compares two binary numbers and determines if one number is greater than, less than, or equal to the other number. It is a multiple output combinational logic circuit. If two binary numbers are considered as A and B, the magnitude comparator gives three outputs for $A > B$, $A < B$, and $A = B$.

Input variables		Output Variables		
A	B	A=B	A>B	A<B
0	0	1	0	0
0	1	0	0	1
1	0	0	1	0
1	1	1	0	0

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

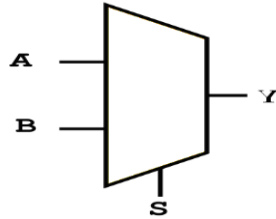
$$(A > B) = A\overline{B}$$

$$(A < B) = \overline{A}B$$



Multiplexer

A multiplexer (mux) is a digital system that selects one out of possible 2^n inputs depending on n select bits. For instance, the truth table and schematic symbol for a 2-to-1 mux are shown below.



symbol of a 2-to-1 mux

And the truth table of (2-to-1) mux is :

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

OR

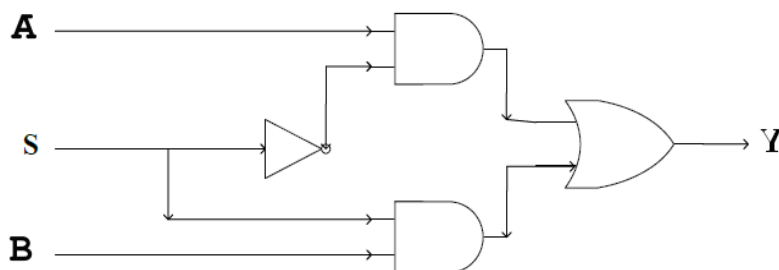
S	Y
0	A
1	B

The Boolean expression for the output (Y) in terms of inputs A, B and S is:

$$Y = \bar{S}\bar{B}A + \bar{S}BA + SB\bar{A} + SBA$$

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

$$Y = \bar{S}A + SB$$



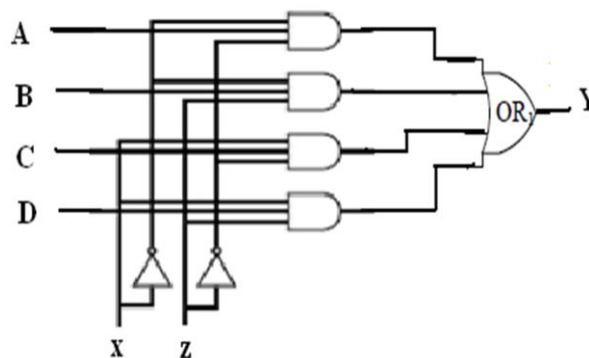
2 - to - 1 multiplexer

Larger multiplexers are also common, if you have 4 inputs then you need 2 select bits. This is the reason for the n-select bits mapping 2^n inputs to one output.

Selectors		Inputs				Output
X	Z	D	C	B	A	Y
0	0	0	0	0	0	0
0	0	0	0	0	1	1
0	0	0	0	1	0	0
0	0	0	0	1	1	1
0	0	0	1	0	0	0
0	0	0	1	0	1	1
0	0	0	1	1	0	0
0	0	0	1	1	1	1
0	0	1	0	0	0	0
0	0	1	0	0	1	1
0	0	1	0	1	0	0
0	0	1	0	1	1	1
0	0	1	1	0	0	0
0	0	1	1	0	1	1
0	0	1	1	1	0	0
0	0	1	1	1	1	1
0	1	0	0	0	0	0
0	1	0	0	0	1	1
0	1	0	0	1	0	0
0	1	0	0	1	1	1
0	1	0	1	0	0	0
0	1	0	1	0	1	1
0	1	0	1	1	0	0
0	1	0	1	1	1	1
0	1	1	0	0	0	0
0	1	1	0	0	1	1
0	1	1	0	1	0	0
0	1	1	0	1	1	1
0	1	1	1	0	0	0
0	1	1	1	0	1	1
0	1	1	1	1	0	0
0	1	1	1	1	1	1
1	0	0	0	0	0	0
1	0	0	0	0	1	1
1	0	0	0	1	0	0
1	0	0	0	1	1	1
1	0	0	1	0	0	0
1	0	0	1	0	1	1
1	0	0	1	1	0	0
1	0	0	1	1	1	1
1	0	1	0	0	0	0
1	0	1	0	0	1	1
1	0	1	0	1	0	0
1	0	1	0	1	1	1
1	0	1	1	0	0	0
1	0	1	1	0	1	1
1	0	1	1	1	0	0
1	0	1	1	1	1	1
1	1	0	0	0	0	0
1	1	0	0	0	1	1
1	1	0	0	1	0	0
1	1	0	0	1	1	1
1	1	0	1	0	0	0
1	1	0	1	0	1	1
1	1	0	1	1	0	0
1	1	0	1	1	1	1
1	1	1	0	0	0	0
1	1	1	0	0	1	1
1	1	1	0	1	0	0
1	1	1	0	1	1	1
1	1	1	1	0	0	0
1	1	1	1	0	1	1
1	1	1	1	1	0	0
1	1	1	1	1	1	1

x	z	Y
0	0	A
0	1	B
1	0	C
1	1	D

$$Y = \bar{X}\bar{Z}A + \bar{X}ZB + X\bar{Z}C + XZD$$



4 - to - 1 multiplexer

Demultiplexer

A demultiplexer basically reverses the multiplexing function. It is take data from one line and distribute them to given number of output lines.

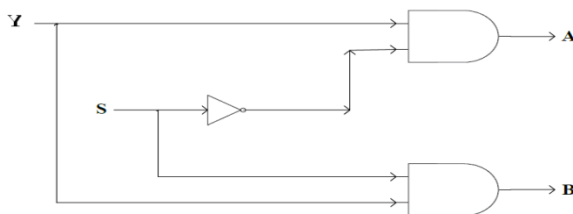
The simplest type of demultiplexer is the 1- to- 2 lines DMUX. as shown in Figure below.

Selector	Input	Outputs	
S	Y	B	A
0	0	0	0
0	1	0	1
1	0	0	0
1	1	1	0

S	A	B
0	Y	0
1	0	Y

$$A = \bar{S}Y$$

$$B = SY$$



1 to 2 Demultiplexer

Figure below shows a one to four line demultiplexer circuit. The input data line goes to all of the AND gates. The two select lines enable only one gate at a time and the data appearing on the input line will pass through the selected gate to the associated output line.

Truth table of 1- to – 4 demultiplexer

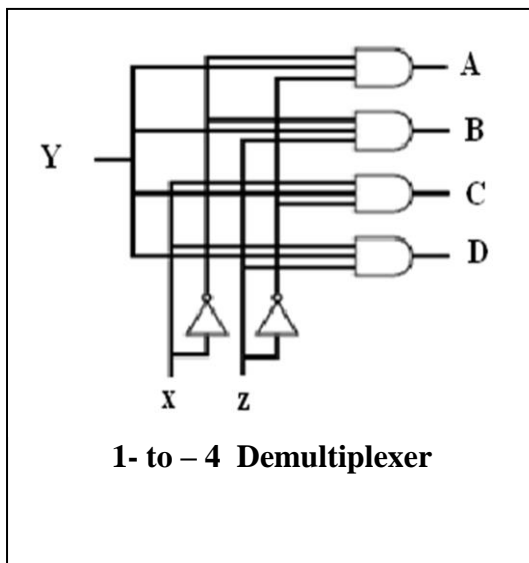
x	z	A	B	C	D
0	0	Y	0	0	0
0	1	0	Y	0	0
1	0	0	0	Y	0
1	1	0	0	0	Y

$$A = \bar{X}\bar{Z}Y$$

$$B = \bar{X}ZY$$

$$C = X\bar{Z}Y$$

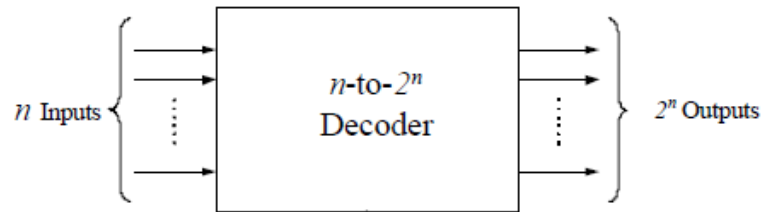
$$D = XZY$$



1- to – 4 Demultiplexer

Decoder

As its name indicates, a decoder is a circuit component that decodes an input code. Given a binary code of n -bits, a decoder will tell which code is this out of the 2^n possible codes (See Figure). Thus, a decoder has n - inputs and 2^n outputs. Each of the 2^n outputs corresponds to one of the possible 2^n input combinations.



In general, output i equals 1 if and only if the input binary code has a value of i .

Example: 2-to-4 decoder

Let us discuss the operation and combinational circuit design of a decoder by taking the specific example of a 2-to-4 decoder. It contains two inputs denoted by A and B and four outputs denoted by D_0 , D_1 , D_2 , and D_3 as shown in figure.

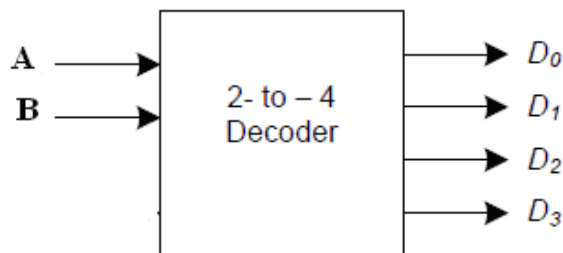


Figure Block Diagram of 2-to-4 Decoder

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

Truth table of 2-to-4 Decoder

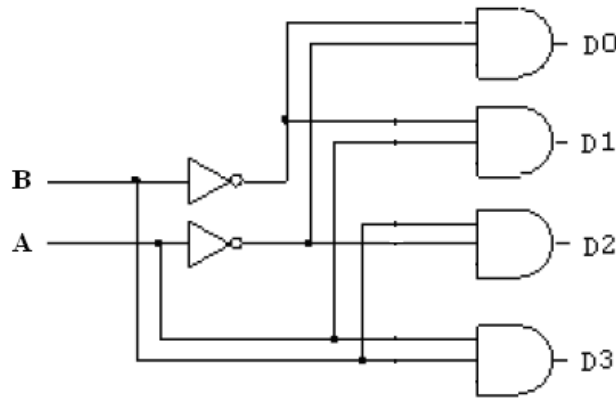
As we see in the truth table , for each input combination, one output line is activated, that is, the output line corresponding to the input combination becomes 1, while other lines remain inactive. For example, an input of 00 at the input will activate line D_0 . and 01 at the input will activate line D_1 , and so on.

$$D_0 = \overline{A}\overline{B}$$

$$D_1 = A\overline{B}$$

$$D_2 = \overline{A}B$$

$$D_3 = AB$$



The logic diagram of 2-to-4 Decoder

Example: 3-to-8 decoder

It contains three inputs denoted by a, b and c, with eight outputs denoted by D0, D1, D2, D3, D4, D5, D6 and D7 as shown in figure

c	b	a	D0	D1	D2	D3	D4	D5	D6	D7
0	0	0	1	0	0	0	0	0	0	0
0	0	1	0	1	0	0	0	0	0	0
0	1	0	0	0	1	0	0	0	0	0
0	1	1	0	0	0	1	0	0	0	0
1	0	0	0	0	0	0	1	0	0	0
1	0	1	0	0	0	0	0	1	0	0
1	1	0	0	0	0	0	0	0	1	0
1	1	1	0	0	0	0	0	0	0	1

$$D_0 = \overline{c}\overline{b}\overline{a}$$

$$D_1 = \overline{c}\overline{b}a$$

$$D_2 = \overline{c}b\overline{a}$$

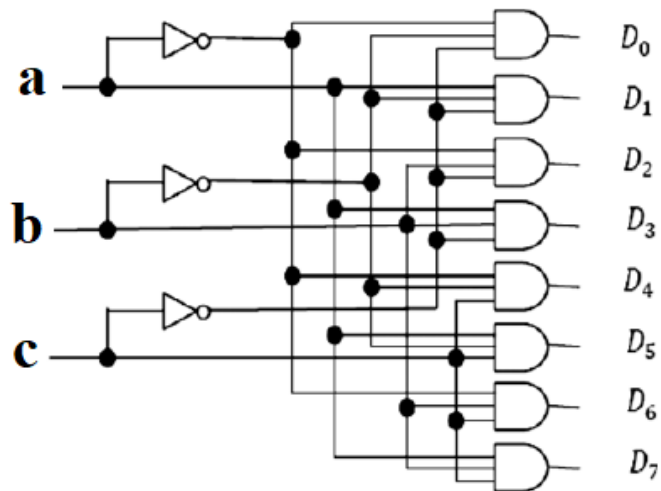
$$D_3 = \overline{c}ba$$

$$D_4 = c\overline{b}\overline{a}$$

$$D_5 = c\overline{b}a$$

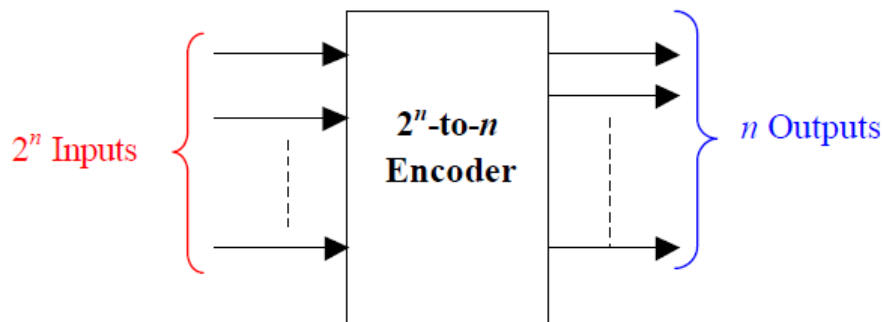
$$D_6 = cb\overline{a}$$

$$D_7 = cba$$



Encoder

The encoder is a combinational circuit that performs the reverse operation of the decoder. The encoder has a maximum of 2^n inputs and n outputs. Only one input can be logic 1 at any given time (active input). All other inputs must be 0's and the Output lines generate the binary code corresponding to the active input. The block diagram of 2^n -to- n encoder as shown below .



Example: 4-to-2 Encoder

The inputs are 4 and the outputs are 2
The block diagram and the truth table of a 4-to-2 encoder are shown below .

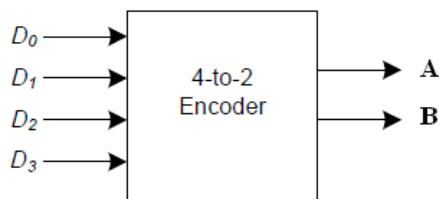


Figure Block Diagram of 4-to-2 Encoder

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

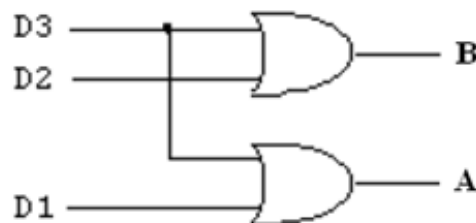
truth table For the 4-to-2 encoder

So that the logic expression of outputs are:

$$\mathbf{A = D1 + D3}$$

$$\mathbf{B = D2 + D3}$$

And the logic diagram of 4-to-2 encoder as shown below:



Logic Diagram of the 4-to-2 Encoder

Example: 8-to-3 Encoder

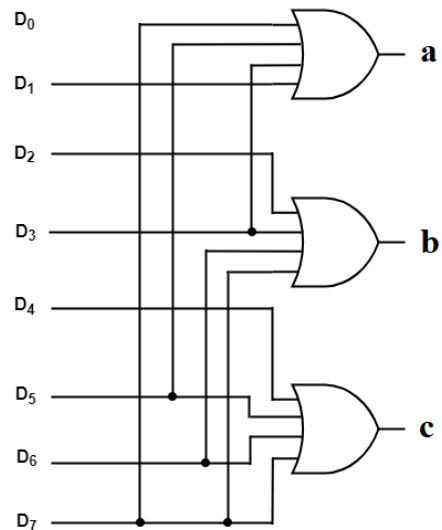
The inputs are 8 and the outputs are 3. The truth table and the logic diagram of a 8-to-3 encoder are shown below.

INPUTS								OUTPUTS		
D0	D1	D2	D3	D4	D5	D6	D7	c	b	a
1	0	0	0	0	0	0	0	0	0	0
0	1	0	0	0	0	0	0	0	0	1
0	0	1	0	0	0	0	0	0	1	0
0	0	0	1	0	0	0	0	0	1	1
0	0	0	0	1	0	0	0	1	0	0
0	0	0	0	0	1	0	0	1	0	1
0	0	0	0	0	0	1	0	1	1	0
0	0	0	0	0	0	0	1	1	1	1

$$a = D_1 + D_3 + D_5 + D_7$$

$$b = D_2 + D_3 + D_6 + D_7$$

$$c = D_4 + D_5 + D_6 + D_7$$



Binary to Gray / Gray to Binary Conversion:

The gray code is widely used in many digital systems specially in shaft encoders and analog to digital conversion, but it is difficult to use the gray-code in arithmetic operations, since there are only one bit change between two consecutive gray code number, and it is unweighted code, and the XOR gate is the most suitable gate for this purpose as shown in Figure below:

