1- Number Systems

In number system modern method of representing numbers symbolically is based on positional notations.

In this method, each number is represented by a string of symbols where each symbol is associated with a specific weight depending upon its positions. The total number of different symbols which are used in a particular number system is called the base or radix of the system and the weight of each position of a particular number is expressed as a power of the base. When a number is formed with the combination of the symbols, each symbol is then called a digit and the position of each symbol is referred to as the digit position.

Thus if a number system has symbols starting from 0, and the digits of the system are 0, 1, 2, ….. (r - 1) then the base or radix is r. If a number D of this system be represented by:
$D = d_{n-1}\ d_{n-2} \ldots\ldots\ d_i \ldots\ldots\ d_1\ d_0$

then the magnitude of this number is given by

$|D| = d_{n-1}\ r^{n-1} + d_{n-2}\ r^{n-2} + \ldots\ldots\ d_i\ r^i + \ldots\ldots\ d_1\ r^1 + d_0\ r^0$

$$\sum_{i=0}^{n-1} d_i\, r^i$$

Where each d_i ranges from 0 to r - 1, such that
$0 \le d_i \le r - 1$, i = 0, 1, 2 …… (n - 1).

The digit at the extreme left has the highest positional value and is generally called the**Most Significant Digit**, or in short **MSD**.

similarly, the digit occupying the extreme right position has the least positional value and is referred to as the **Least Significant Digit** or**LSD**.

1- Decimal number

Decimal number system is the most common example of positional notational number system and all the arithmetical calculations undertaken by human being are carried out on the basis of this number system. In this system, the symbols used are 0, 1, 2, 3, 4, 5, 6, 7, 8, 9 and the base is 10. Thus the number

$d_{n-1}\ d_{n-2}.....d_1\ d_0$ means $d_{n-1}\ 10^{n-1} + d_{n-2}\ 10^{n-2} + ... + d_1\ 10^1 + d_0\ 10^0$

and this is an n-digit number. If the number be extended to the right of the decimal point, then the powers of the base will be negative starting from -1.

For example, the number 3528 has the magnitude

$3528 = 3 \times 10^3 + 5 \times 10^2 + 2 \times 10^1 + 8 \times 10^0$

and the number 26.57 has the magnitude

$26.57 = 2 \times 10 + 6 \times 10^0 + 5 \times 10^{-1} + 7 \times 10^{-2}$

2- Binary Number System

Binary number system uses two symbols 0 and 1 and its radix is 2. The symbols 0 and 1 are generally called **BITS** which is a contraction of the two words Binary digits.

An n-bit binary number of the form **$a_{n-1}\ a_{n-2}\ .....\ a_1\ a_0$** where each a_i (i = 0, 1, …. n - 1) is either 0 or 1 has the magnitude.

$a_{n-1}\ 2^{n-1} + a_{n-2}\ 2^{n-2} + ...+ a_1\ 2^1 + a_0 2^0$.

For fractional binary numbers, the base has negative integral powers starting with -1 for the bit position just after the binary point.

The bit at the extreme left of a binary number has the highest positional value and is usually called the **Most Significant Bit** or **MSB**. Similarly, the bit occupying the extreme right position of a given binary number has the least positional value and is referred to as the **Least Significant Bit** or **LSB**.

To facilitate the distinction between different number systems, we generally use the respective radix as a subscript of the number. However the subscript will not be used when there is no scope of confusion.

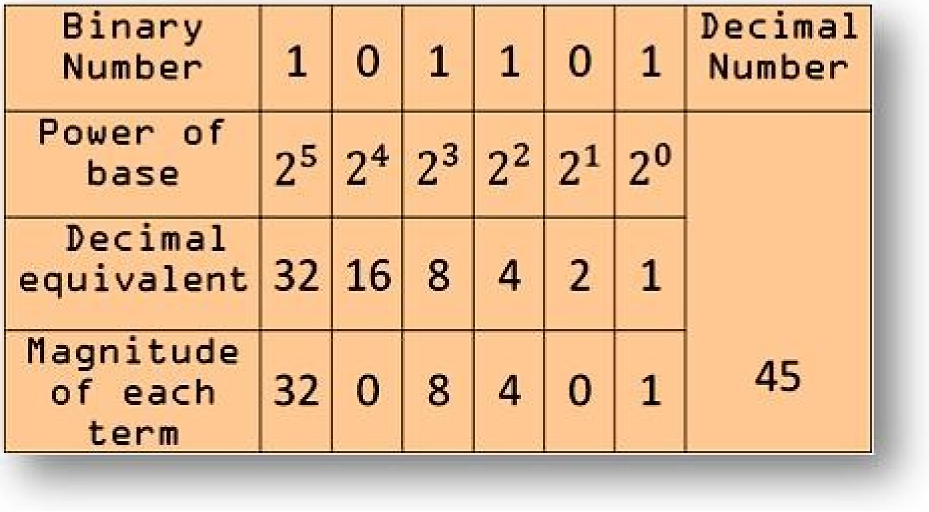
a few examples on binary numbers and their decimal equivalents are given below:

examp1: $101101_2 = 1 \times 2^5 + 0 \times 2^4 + 1 \times 2^3 + 1 \times 2^2 + 0 \times 2^1 + 1 \times 2^0$

= 32 + 0 + 8 + 4 + 0 + 1

= 45_{10}

The above results can be more clearly expressed in the following manner:



Binary Number	1	0	1	1	0	1	Decimal Number
Power of base	2^5	2^4	2^3	2^2	2^1	2^0	45
Decimal equivalent	32	16	8	4	2	1	
Magnitude of each term	32	0	8	4	0	1	

Exmple2:

Binary point

↓

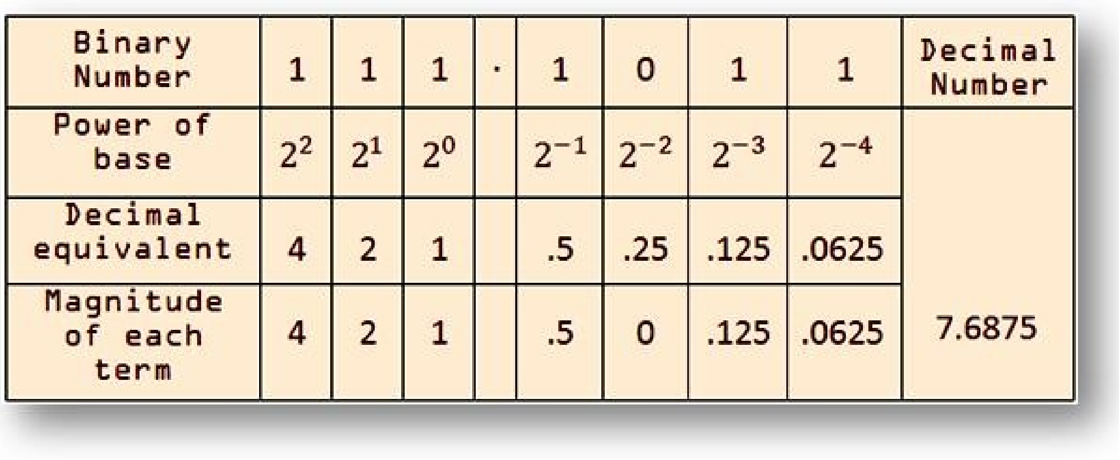
111.1011_2

$= 1 \times 2^2 + 1 \times 2^1 + 1 \times 2^0 + 1 \times 2^{-1} + 0 \times 2^{-2} + 1 \times 2^{-3} + 1 \times 2^{-4}$

$= 4 + 2 + 1 + .5 + 0 + .125 + .0625$

$= 7.6875_{10}$

The above results can be more clearly expressed in the following manner:



Binary Number	1	1	1	.	1	0	1	1	Decimal Number
Power of base	2^2	2^1	2^0		2^{-1}	2^{-2}	2^{-3}	2^{-4}	7.6875
Decimal equivalent	4	2	1		.5	.25	.125	.0625	
Magnitude of each term	4	2	1		.5	0	.125	.0625	

# Why binary numbers are used?

It may be observed from the discussions of the preceding section that the use of a base smaller than 10 requires more positions to represent a given decimal number. As for example, the binary number 10101 requires 5 bit positions to represent the decimal number 21 which requires two positions for its decimal representation. This is a major disadvantage of the binary number system. In spite of this fact, all the modern digital computers have been basically designed on the basis of binary number system.

Why this bias to binary number?

There are several reasons for this.

The first and foremost reason is that electronic components, as a natural coincidence, operate in a binary mode. A switch is either open/off (called 0 state) or closed/on (called 1 state); a transistor is either not conducting (0 state) or is conducting (1 state).

- ❖ **This two-state nature of the electronic components can be easily expresses with the help of binary numbers.**
- ❖ **The second reason is that computer circuits have to handle only two bits instead of 10 digits of the decimal system. This simplifies the design of the machine, reduces the cost and improves the reliability.**
- ❖ Lastly, binary number system is used because all the operations that can be done in the decimal system can also be done with a binary number of radix 2.

3- Octal Number System

Octal number system has a base or radix 8. Eight different symbols, namely **0, 1, 2, 3, 4, 5, 6, 7** are used to represent octal numbers. Conversion of octal numbers to their decimal equivalents can be accomplished by using the same rule which was followed to convert binary numbers to decimal numbers, except that we now have a radix 8 instead of 2. Thus the octal number 273 has the decimal equivalent.

$$273_8$$

$$= 2 \times 8^2 + 7 \times 8^1 + 3 \times 8^0$$

$$= 128 + 56 + 3$$

$$= 187_{10}$$

4- Hexa -decimal Number System

The hexa-decimal number system has a radix or base 16. It requires 16 symbols to represent a number in this system. The symbols are **0 to 9, A, B, C, D, E, F** where the symbols **A, B, C, D, E, F** represent the decimal numbers **10, 11, 12, 13, 14, 15** respectively.

Example : Convert $B6A_{16}$ to its decimal equivalent.

Solution:

$B6A_{16}$

$= 11 \times 16^2 + 6 \times 16^1 + 10 \times 16^0$

$= 2816 + 96 + 10$

$= 2922_{10}$ **Therefore, $B6A_{16} = 2922_{10}$**

# Conversion Of Numbers

Conversion of numbers from one system to another becomes necessary to understand the process and the logic of the operations of a computer system. It is not very difficult to convert numbers from one base to another.

There are several traditional methods of converting the numbers from binary to decimal conversion. We shall discuss here the two most commonly used methods, namely: "**Expansion or value box Method**" **and** "**Multiplication and Division Method**"

Note **:We will first discuss about the conversion of binary numbers to their decimal equivalents , after that applying these methods to octal and hexa-decimal number respectively.**

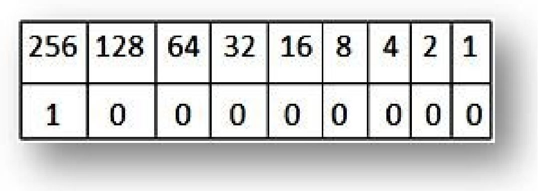
(i) Expansion Method:

In expansion method the conversion of binary numbers to their decimal equivalents are shown with the help of the examples.

1. Convert the decimal numbers to their binary equivalents:

Exp1: 256

Solution: 256



256	128	64	32	16	8	4	2	1
1	0	0	0	0	0	0	0	0

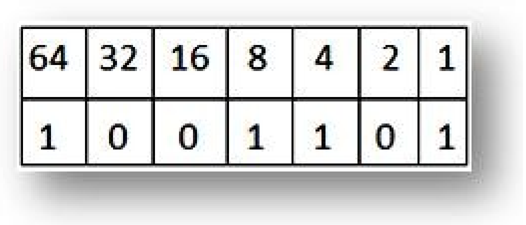
Since the given number 256 appears in the first row, we put 1 in the slot below 256 and fill all the other slots to the right of this slot with zeros.

Thus, 256_{10} = 100000000_2

exp2: 77

Solution:

77



64	32	16	8	4	2	1
1	0	0	1	1	0	1

The given number is less than 128 but greater than 64. We therefore put 1 in the slot corresponding to 64 in the first row. Next, we subtract 64 from 77 and get 13 as remainder.

This remainder is less than 16 and greater than 8. So we put 1 in the slot corresponding to 8 and subtract 8 from 13. This gives 13 - 8 = 5. This remainder is greater than 4 and less than 8.

Hence we put 1 in the slot corresponding to 4 and subtracting 4 from 5 we get 1. Now, 1 is present in the right hand most slot of the first row. We, therefore, put 1 in the corresponding slot and fill all other slots with zeros.

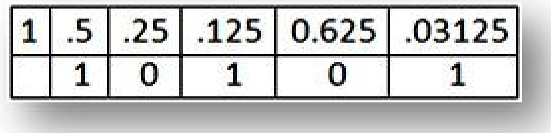
Thus, 77_{10} = 1001101_2.

2. Conversion of decimal fractions to binary fractions

Conversion of decimal fractions to binary fractions may also be accomplished by using similar method. Let us observe the procedure with the help of the following example:

Exp1 : Convert 0.675_{10} to its binary equivalent.

Solution:



1	.5	.25	.125	0.625	.03125
	1	0	1	0	1

Subtract .5 from the given number to get .675 - .5 = .175 and place 1 in the slot corresponding to .5 of the first row.

Now the number .175 is less than .25 and greater than .125. So, we put 1 in the slot corresponding to the number .125 of the first row and subtract .125 from .175 to get .175 - .125 = .05. The remainder .05 is less than .0625 but greater than . 03125.

Hence we put 1 in the slot corresponding to 0.3125 and the subtraction given .05 - .03125 = .01875 and continue the process. The other slots are then filled with zeros.

Thus, $.675_{10}$ = $(.10101\ldots)_2$

Note:

It should be noted that the conversion of decimal fractions to binary fractions may not be exact and the process is to be continued until there is no remainder or the remainder is less than the order of accuracy desired.

(ii) Multiplication and Division Method

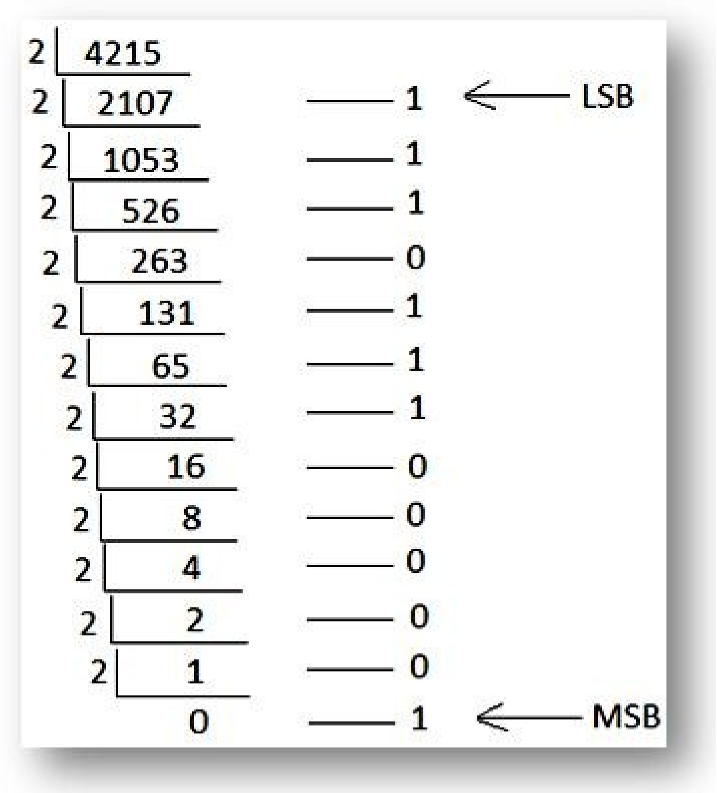
The expiation or value box method of converting numbers from decimal to binary is laborious and time consuming and is suitable for small numbers when it can be performed mentally. It is advisable not to use it for large numbers. The conversion of large numbers may be conveniently done by multiplication and division method which is described below.

To effect the conversion of positive integers of the decimal system to binary numbers the decimal number is repeatedly divided by the base of the binary number system, i.e., by 2. The division is to be carried until the quotient is zero and the remainder of each division is recorded on the right. The binary equivalent of the decimal number is then obtained by writing down the successive remainders. The first remainder is the least significant bit and the last one is the most significant bit of the binary number. Thus the binary equivalent is written from the bottom upwards.

We explain conversion of numbers using multiplication and division method with the help of following example.

Exp1: Convert 4215_{10} to its binary equivalent

Solution:



Therefore, 4215_{10} =1000001110111_2

The conversion of decimal fractions to binary fractions is accomplished by multiplying repeatedly the decimal fraction by the base 2 of the binary number. The integral part after each multiplication is either 0 or 1. The equivalent binary fraction is obtained by writing the integral parts of each product to the right of the binary point in the same sequence. If the fractional part of the product becomes exactly zero at a certain stage, then the binary fraction is finite, otherwise, the fraction is non-terminating and then we find the binary fraction upto the desired degree of accuracy. We explain the process with the help of the following examples.

Exp2. Convert the following decimal numbers to their binary equivalents:

(a) 0.375

Solution:

Decimal Numbers to Binary Number Conversion Table		
Multiplication	**Integer**	**Fraction**
0.375 × 2 = 0.75	**0**	**.75**
0.75 × 2 = 1.5	**1**	**.5**
.5 × 2 = 1.0	**1**	**0**

Therefore, $0.375_{10} = 0.011_2$

(b) 0.435

Solution:

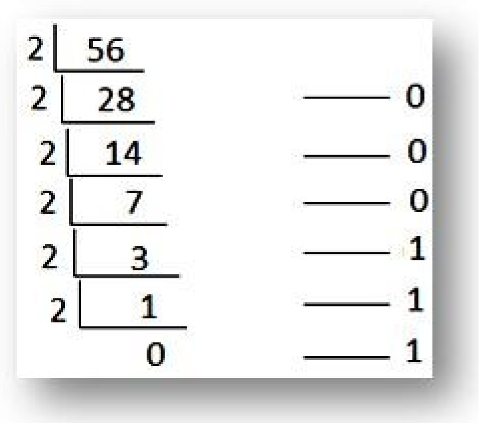
Decimal Numbers to Binary Number Conversion Table		
Multiplication	**Integer**	**Fraction**
0.435 × 2 = 0.87	**0**	**.87**
0.87 × 2 = 1.74	**1**	**.74**
.74 × 2 = 1.48	**1**	**.48**
.48 × 2 = 0.96	**0**	**.96**
.96 × 2 = 1.92	**1**	**.92**

Therefore, $0.435_{10} = (0.01101...)_2$

o **Note :** Fox mixed number, we will have to separate the number into its integral and fractional parts and find the binary equivalent of each part independently. Finally, we add the two parts to get the binary equivalent of the given number.

Exp3.: Convert $(56.75)_{10}$ to its binary equivalent.

Solution:
At first we find the binary equivalent of 56.



Therefore, $56_{10} = 111000_2$
The binary equivalent of 0.75 is obtained below:

Decimal Numbers to Binary Number Conversion Table		
Multiplication	**Integer**	**Fraction**
0.75 × 2 = 1.5	**1**	**.5**
0.5 × 2 = 1.0	**1**	**0**

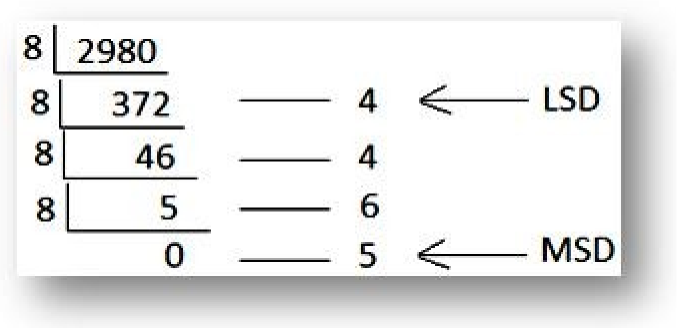
Therefore, $0.75_{10} = 0.11_{10}$

Hence $56.75_{10} = 111000.11_{10}$

Exp3 : Convert the decimal numbers to their octal equivalents:

(a) 2980

Solution:



8 | 2980
8 | 372 —— 4 ⟵ LSD
8 | 46 —— 4
8 | 5 —— 6
0 —— 5 ⟵ MSD

Hence $2980_{10} = 5644_{8}$

(b) 0.685

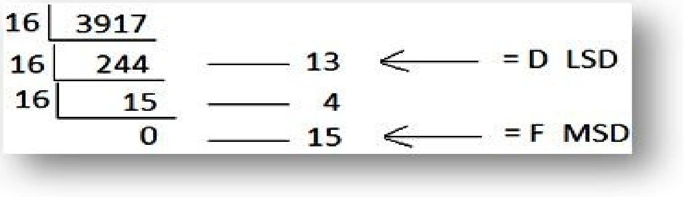
Solution:

Decimal Numbers to Binary Number Conversion Table		
Multiplication	**Integer**	**Fraction**
0.685 × 8 = 5.480	**5**	**.48**
0.48 × 8 = 3.84	**3**	**.84**
.84 × 8 = 6.72	**6**	**.72**
.72 × 8 = 5.76	**5**	**.76**

4- conversion of hexa-decimal numbers to their decimal equivalents

The conversion of hexa-decimal numbers to their decimal equivalents is straight-forward and follows the same rules as that of octal or binary to decimal. Similarly, conversion of decimal to hexa-decimal may be worked out with the help of division or multiplication, as the case may be, by the radix 16.

Exp 4: Convert 3917_{10} to its hexa-decimal equivalent
Solution:



Therefore, 3917_{10} = $F4D_{16}$

# Conversion of Binary Numbers to Octal or Hexa - decimal Numbers

Conversion of binary numbers to octal or hexa-decimal numbers and vice-versa may be accomplished very easily.

Since a string of 3 bits can have 8 different permutations, it follows that each 3-bit string is uniquely represented by one octal digit. Similarly, since a string of 4 bits has 16 different permutations each 4 bit string represents a hexa-decimal digit uniquely. The table below gives the decimal numbers 0 to 15 and their binary, octal and hexa-decimal equivalents and also the corresponding 3-bit and 4-bit strings.

Conversion of binary numbers to octal or hexa-decimal numbers and vice versa:

Decimal	Binary	Octal	3-bit String	Hexa-decimal	4-bit String
0	0	0	000	0	0000
1	1	1	001	1	0001
2	10	2	010	2	0010
3	11	3	011	3	0011
4	100	4	100	4	0100
5	101	5	101	5	0101
6	110	6	110	6	0110
7	111	7	111	7	0111
8	1000	10	-	8	1000
9	1001	11	-	9	1001
10	1010	12	-	A	1010
11	1011	13	-	B	1011
12	1100	14	-	C	1100
13	1101	15	-	D	1101
14	1110	16	-	E	1110
15	1111	17	-	F	1111

Thus to convert a binary number to its octal equivalent we arrange the bits into groups of 3 starting at the binary point and move towards the MSB. We then replace each group by the corresponding octal digit. If the number of bits is not a multiple of 3, we add necessary number of zeros to the left of MSB. For binary fractions, we have to work towards the right of the binary point and follow the same procedure. Similarly, for conversion of octal numbers to binary numbers, we have to replace each octal digit by its 3-bit binary equivalent.

The same procedure is to be adopted in the case of hexa-decimal numbers and vice versa by converting the given numbers to binary numbers first with the help of above procedure and then converting these binary numbers to hexa-decimal numbers. Conversion to decimal may also be accomplished by the same procedure.

The following examples explain the converting method :

1. Convert the following to octal numbers:

(a) 1110101110_2

Solution: $001\underline{110}101\underline{110} = 001\ 110\ 101\ 110 = 1656_8$

Hence the required octal equivalent is 1656.

(b) 111101.01101_2

Solution: $111\underline{101}.011\underline{010}_2 = 75.32_8$

Hence the required octal equivalent is 75.32.

2. Convert the following to their binary equivalents:

(a) 1573_8

Solution: $1573_8 = 001\ 101\ 111\ 011 = 1101111011_2$

Hence the required binary number is 1101111011.

(b) 64.175_8

Solution: $64.175_8 = 110\ 100\ .\ 001\ 111\ 101 = 110100.001111101_2$

Hence the required binary number is 110100.001111101.

3.Convert the following to hexa-decimal numbers:

(a) 1111101101_2

Solution: $\underline{0011}1110\underline{1101} = 0011\ 1110\ 1101 = 3ED_{16}$

Therefore, $11\ 1110\ 1101_2 = 3ED_{16}$

(b) 11110.01011_2

Solution:

$1\underline{1110}.01011_2$ = 0001 1110 . 0101 1000 = $1E.58_{16}$

Therefore, $11110.01011_2 = 1E.58_{16}$

4. Convert the following to binary equivalents:

(a) $A748_{16}$

Solution: $A748_{16}$= 1010 0111 0100 1000= 1010011101001000_2

Hence the required binary equivalent is 1010011101001000.

(b) $BA2.23C_{16}$

Solution: $BA2.23C_{16}$ = 1011 1010 0010 . 0010 0011 1100_2

= 101110100010.0010001111

Hence the required binary equivalent is 101110100010 . 0010001111

5. Convert 1573_8 to hexa-decimal

Solution: 1573_8 = 001101111011 = 0011 0111 1011 $37B_{16}$

Hence $1573_8 = 37B_{16}$

6. Convert $A748_{16}$ to octal equivalents.

Solution: $A748_{16}$ = 1010 0111 0100 1000= 001 010 011 101 001 000

= 123510_8

Therefore, $A748_{16} = 123510_8$

7. Convert the following to decimal numbers:

(a) 725_8

Solution: 725_8 = 111010101= 256 + 128 + 64 + 16 + 4 + 1= 469_{10}

Therefore, $725_8 = 469_{10}$

(b) $D9F_{16}$

Solution: $D9F_{16}$ = 1101 1001 1111= 110110011111

= 2048 + 1024 + 256 + 128 + 16 + 8 + 4 + 2 + 1 = 3487_{10}

Therefore, $D9F_{16} = 3487_{10}$

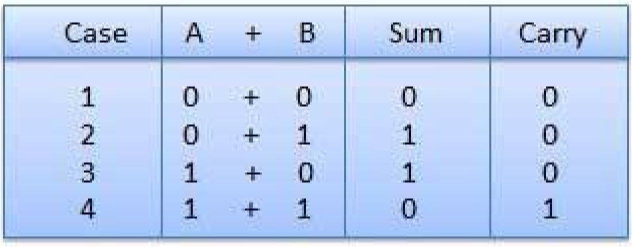
2-Arithmatic Operations

# Binary Arithmetic

We are very familiar with different arithmetic operations, *viz*. addition, subtraction, multiplication, and division in a decimal system. Now we want to find out how those same operations may be performed in a binary system, where only two digits, *viz.* 0 and 1 exist.

-Binary Addition

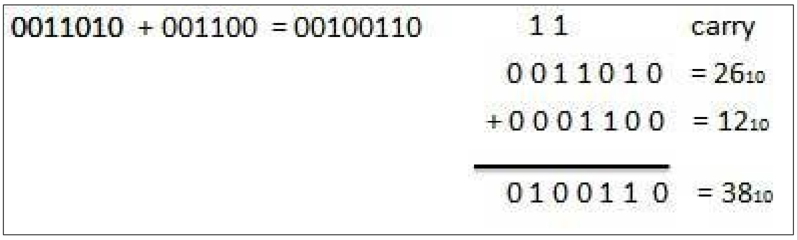
it is a key for binary subtraction, multiplication, division. There four rules of the binary addition.



Case	A + B	Sum	Carry
1	0 + 0	0	0
2	0 + 1	1	0
3	1 + 0	1	0
4	1 + 1	0	1

In fourth case, a binary addition is creating a sum of (1+1=10) i.e. 0 is write in the given column and a carry of 1 over to the next column.

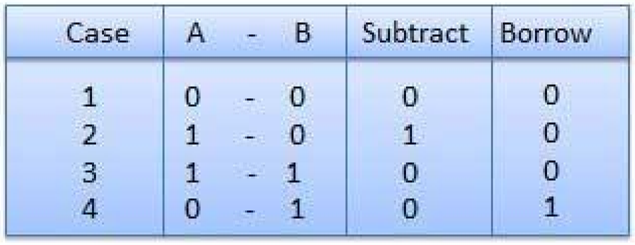
Exp1:



```
0011010 + 001100 = 00100110        1 1          carry
                                 0 0 1 1 0 1 0   = 26₁₀
                                +0 0 0 1 1 0 0   = 12₁₀
                                ______________
                                 0 1 0 0 1 1 0   = 38₁₀
```

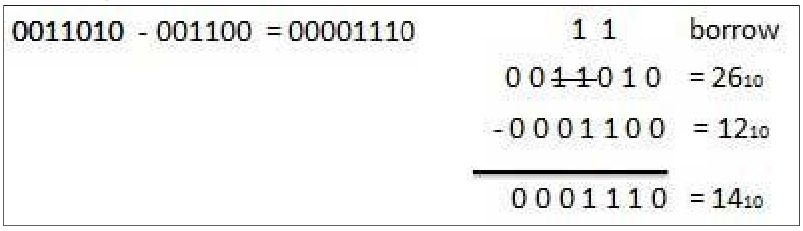
- Binary Subtraction

Subtraction and Borrow, these two words will be used very frequently for the binary subtraction. There four rules of the binary subtraction. There four rules of the binary subtraction.



Case	A - B	Subtract	Borrow
1	0 - 0	0	0
2	1 - 0	1	0
3	1 - 1	0	0
4	0 - 1	0	1

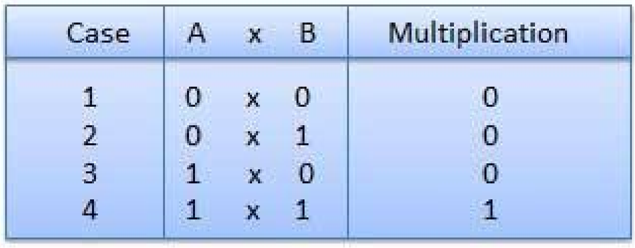
Exp:



```
0011010 - 001100 = 00001110              1 1      borrow
                                    0 0 ~~1 1~~ 0 1 0   = 26₁₀
                                   - 0 0 0 1 1 0 0   = 12₁₀
                                   _______________
                                     0 0 0 1 1 1 0   = 14₁₀
```

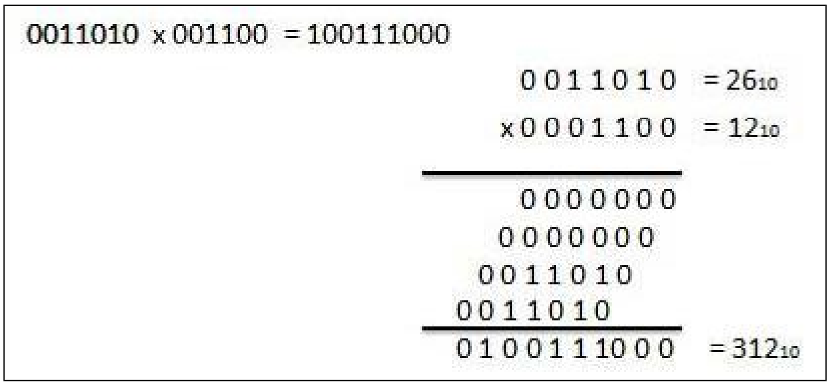
- Binary Multiplication

Binary multiplication is similar to decimal multiplication. It is simpler than decimal multiplication because only 0s and 1s are involved. There four rules of the binary multiplication.



Case	A x B	Multiplication
1	0 x 0	0
2	0 x 1	0
3	1 x 0	0
4	1 x 1	1

Exp:



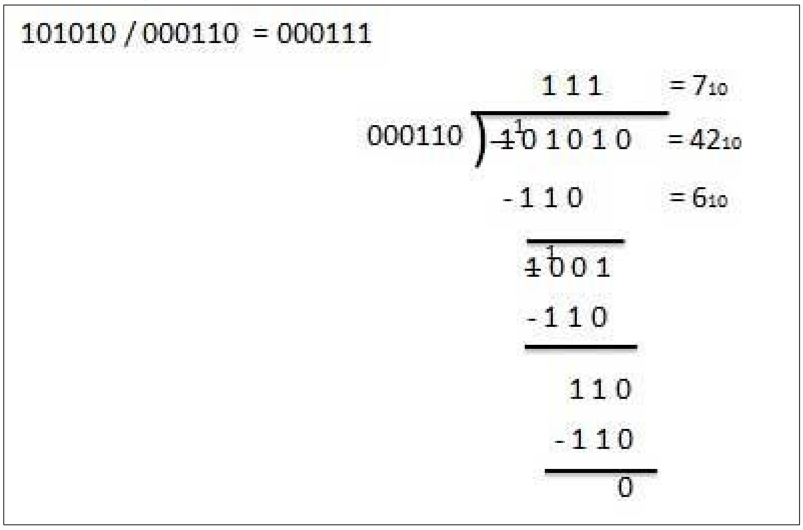
```
0011010 x 001100 = 100111000
                            0 0 1 1 0 1 0   = 26₁₀
                          x 0 0 0 1 1 0 0   = 12₁₀
                    ___________________
                            0 0 0 0 0 0 0
                          0 0 0 0 0 0 0
                        0 0 1 1 0 1 0
                      0 0 1 1 0 1 0
                    ___________________
                      0 1 0 0 1 1 1 0 0 0   = 312₁₀
```

- Binary Division

Binary division is similar to decimal division.where, the division of two digits is as follows:

case	A ÷ B	Division
1	1 ÷ 1	1
2	0 ÷ 1	0

It is called as the long division procedure.



101010 / 000110 = 000111

$$
\begin{array}{r|l l}
 & 111 & =7_{10} \\
000110 & \cancel{1}\overset{1}{0}1010 & =42_{10} \\
 & -110 & =6_{10} \\
 & \cancel{1}\overset{1}{0}01 & \\
 & -110 & \\
 & 110 & \\
 & -110 & \\
 & 0 &
\end{array}
$$

Home work :

1- $(11110)_2 + (1100)_2 = (?)_2$, 2-$(1011101)_2 + (1001011)_2 = (?)_2$, 3-$(110)_2 + (101)_2 + (10)_2 = (?)_2$

4-$(10011)_2 - (1001)_2 = (?)_2$, 5- $(110.1)_2 - (100.10)_2 = (?)_2$, 6- $(111)_2 - (100)_2 - (01)_2 = (?)_2$

7- $(101101)_2 * (10101)_2 = (?)_2$, 8- $(101.01)_2 * (11.01)_2 = (?)_2$, 9- $(110)_2 * (11)_2 * (01)_2 = (?)_2$

10- $(1100)_2 \div (11)_2 = (?)_2$, 11- $(1011010)_2 \div (11)_2 = (?)_2$, 12- $(1100101)_2 \div (100)_2 = (?)_2$

#Octal Arithmetic

This section describes octal arithmetic operations addition and subtraction.

Quick Preview :the following are the characteristics of an 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.

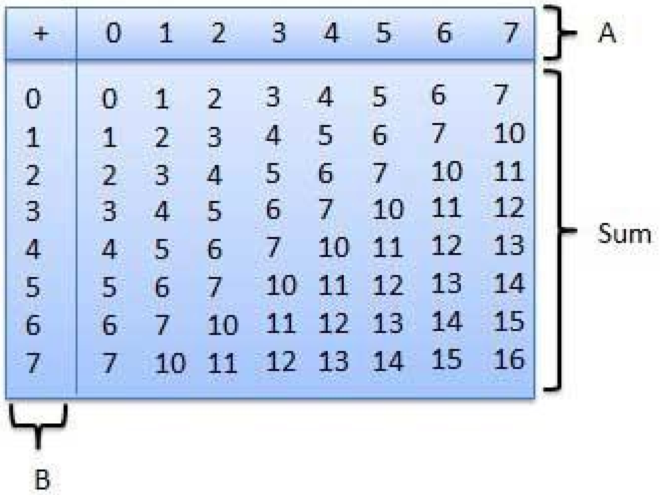
Example: Octal Number: 12570_8 , Calculating Decimal Equivalent:

Step1: $12570_8 = ((1 \times 8^4) + (2 \times 8^3) + (5 \times 8^2) + (7 \times 8^1) + (0 \times 8^0))_{10}$
Step2: $12570_8 = (4096 + 1024 + 320 + 56 + 0)_{10}$
Step3: $12570_8 = 5496_{10}$

Note: 12570_8 is normally written as 12570.

- Octal Addition

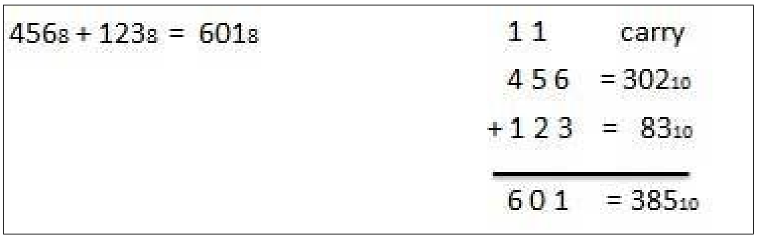
Following octal addition table will help you greatly to handle Octal addition.



+	0	1	2	3	4	5	6	7
0	0	1	2	3	4	5	6	7
1	1	2	3	4	5	6	7	10
2	2	3	4	5	6	7	10	11
3	3	4	5	6	7	10	11	12
4	4	5	6	7	10	11	12	13
5	5	6	7	10	11	12	13	14
6	6	7	10	11	12	13	14	15
7	7	10	11	12	13	14	15	16

To use this table, simply follow the directions used in this example: Add: 6_8 and 5_8.Locate 6 in the A column then locate the 5 in the B column. The point in sum area where these two columns intersect is the sum of two numbers. $6_8 + 5_8 = 13_8$.

Example:

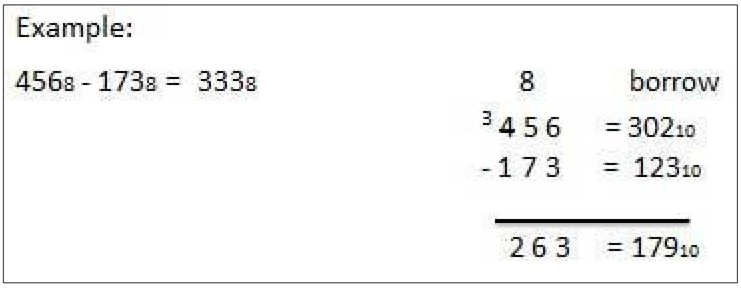


$456_8 + 123_8 = 601_8$

```
 1 1      carry
 4 5 6    = 302₁₀
+1 2 3    =  83₁₀
______
 6 0 1    = 385₁₀
```

- Octal Subtraction

The subtraction of octal numbers follows the same rules as the subtraction of numbers in any other number system. The only variation is in borrowed number. In the decimal system, you borrow a group of 10_{10}. In the binary system, you borrow a group of 2_{10}. In the octal system you borrow a group of $8_{10.}$



Example:

$456_8 - 173_8 = 333_8$

```
   8          borrow
 ³4 5 6    = 302₁₀
 -1 7 3    = 123₁₀
 ______
  2 6 3    = 179₁₀
```

Home work :

1- $(123)_8 + (65)_8 = (?)_8$, 2-$(1740)_8 + (1234)_8 = (?)_8$, 3-$(16)_8 + (23)_8 + (10)_8 = (?)_8$

4-$(356)_8 - (43)_8 = (?)_8$, 5- $(4457)_8 - (3210)_8 = (?)_8$, 6-$(123)_8 - (65)_8 - (12)_8 = (?)_8$

# Hexadecimal Arithmetic

This section describes hexadecimal arithmetic operations addition and subtraction.

Quick Preview :the following are the characteristics of a 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 represents 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:

Step1: $19FDE_{16} = ((1 \times 16^4) + (9 \times 16^3) + (F \times 16^2) + (D \times 16^1) + (E \times 16^0))_{10}$

Step2: $19FDE_{16} =((1 \times 16^4) + (9 \times 16^3) + (15 \times 16^2) + (13 \times 16^1) + (14 \times 16^0))_{10}$

Step3: $19FDE_{16} =(65536+ 36864 + 3840 + 208 + 14)_{10}$

Step4: $19FDE_{16} =106462_{10}$

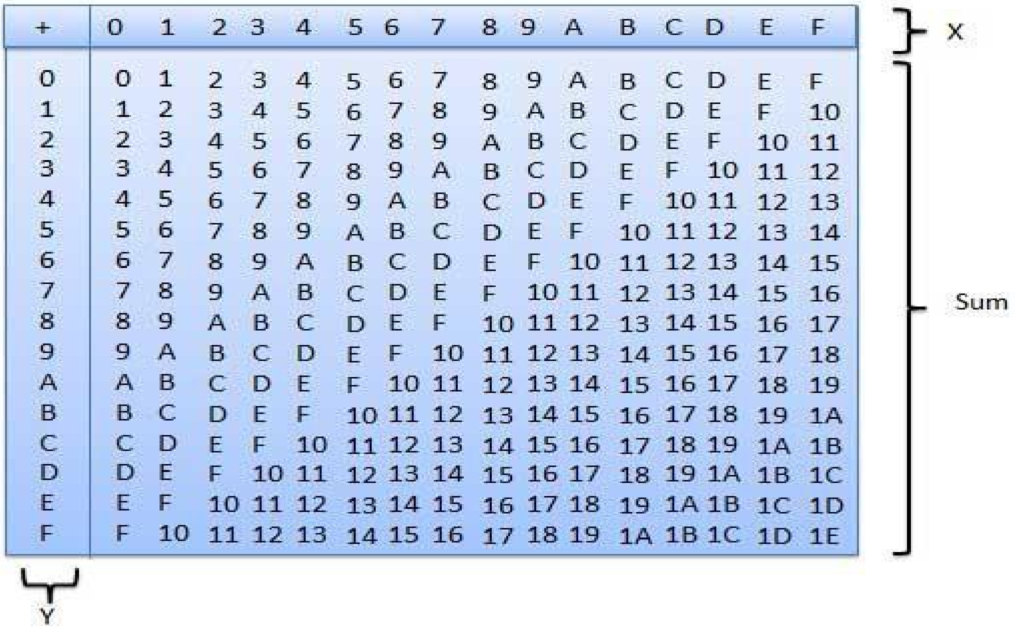
Note: $19FDE_{16}$ is normally written as 19FDE.

-Hexadecimal Addition

Following hexadecimal addition table will help you greatly to handle Hexadecimal addition.

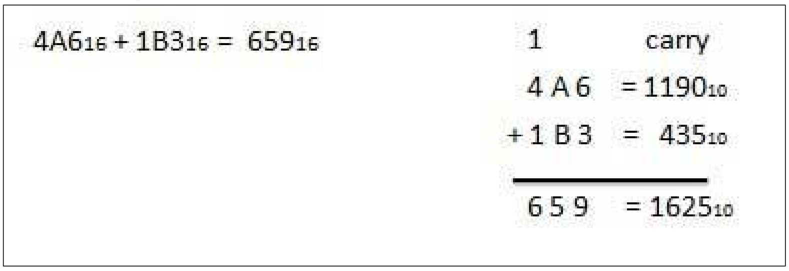
To use this table, simply follow the directions used in this example: Add: A16 and 516.Locate A in the X column then locate the 5 in the Y column. The point in sum area where these two columns intersect is the sum of two numbers.

$A_{16} + 5_{16} = F_{16}$.



+	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
1	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	10
2	2	3	4	5	6	7	8	9	A	B	C	D	E	F	10	11
3	3	4	5	6	7	8	9	A	B	C	D	E	F	10	11	12
4	4	5	6	7	8	9	A	B	C	D	E	F	10	11	12	13
5	5	6	7	8	9	A	B	C	D	E	F	10	11	12	13	14
6	6	7	8	9	A	B	C	D	E	F	10	11	12	13	14	15
7	7	8	9	A	B	C	D	E	F	10	11	12	13	14	15	16
8	8	9	A	B	C	D	E	F	10	11	12	13	14	15	16	17
9	9	A	B	C	D	E	F	10	11	12	13	14	15	16	17	18
A	A	B	C	D	E	F	10	11	12	13	14	15	16	17	18	19
B	B	C	D	E	F	10	11	12	13	14	15	16	17	18	19	1A
C	C	D	E	F	10	11	12	13	14	15	16	17	18	19	1A	1B
D	D	E	F	10	11	12	13	14	15	16	17	18	19	1A	1B	1C
E	E	F	10	11	12	13	14	15	16	17	18	19	1A	1B	1C	1D
F	F	10	11	12	13	14	15	16	17	18	19	1A	1B	1C	1D	1E

Example:



$4A6_{16} + 1B3_{16} = 659_{16}$

1 carry

4 A 6 $= 1190_{10}$

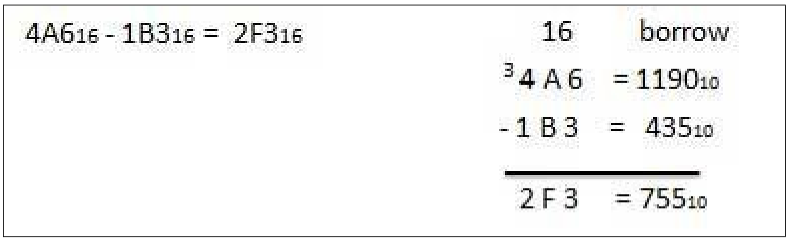
+1 B 3 $= 435_{10}$

6 5 9 $= 1625_{10}$

-Hexadecimal Subtraction

The subtraction of hexadecimal numbers follows the same rules as the subtraction of numbers in any other number system. The only variation is in borrowed number. In the decimal system, you borrow a group of 10_{10}. In the binary system, you borrow a group of 2_{10}. In the hexadecimal system you borrow a group of 16_{10}.

Example:



$4A6_{16} - 1B3_{16} = 2F3_{16}$

16 borrow

^{3}4 A 6 $= 1190_{10}$

-1 B 3 $= 435_{10}$

2 F 3 $= 755_{10}$

Home work :

1- $(5AE)_{16} + (94B)_{16} = (?)_{16}$ **,** **2-**$(52EB)_{16} + (1234)_{16} = (?)_{16}$ **,** **3-**$(10)_{16} + (20)_{16} + (11)_{16} = (?)_{16}$

4-$(FE)_{16} - (37)_{16} = (?)_{16}$ **,** **5-** $(BC66)_{16} - (2F3)_{16} = (?)_{16}$ **,** **6-**$(123)_{16} - (65)_{16} - (12)_{16} = (?)_{16}$

3-Representing Negative Numbers

In the “real world” of mathematics, computers must represent both positive and negative binary numbers. For example, even when dealing with positive arguments, mathematical operations may produce a negative result:

– Example: 124 – 237 = −113.

• Thus needs to be a consistent method of representing negative numbers in binary computer arithmetic operations.

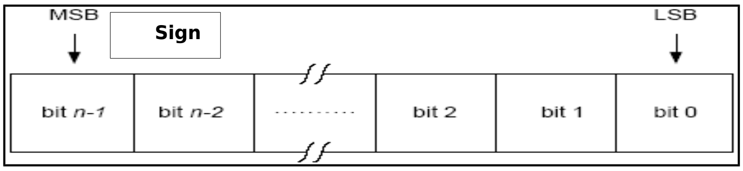
• Basically there are three types of representations of signed binary numbers—sign-magnitude representation, 1’s complement representation, and 2’scomplement representations, which are discussed below.

1-Sign-magnitude representation.

In decimal system, generally a plus (+) sign denotes a positive number whereas a minus (−) sign denotes a negative number. But, the plus sign is usually dropped, and nosign means the number is positive. This type of representation of numbers is known as ***signed numbers.***

But in digital circuits, there is no provision to put a plus or minus sign, since everything in digital circuits have to be represented in terms of 0 and 1. Normally an additional bit is used as the *sign bit*. This sign bit is usually placed as the MSB. Generally a 0 is reserved for a positive number and a 1 is reserved for a negative number.

For example, an 8-bit signed binary number 01101001 represents a positive number whose magnitude is (1101001)2 = (105)10. The MSB is 0, which indicates that the number is positive. On the other hand, in the signed binary form, 11101001 represents a negative number whose magnitude is (1101001)2 = (105)10. The 1 in the MSB position indicates that the number is negative and the other seven bits give its magnitude. This kind of representation of binary numbers is called ***sign-magnitude representation.***



Example :

Signed Integer	Sign Magnitude
+2	0000 0010
+1	0000 0001
0	0000 0000
-1	1000 0001
-2	1000 0010

The drawbacks :to using this method for arithmetic computation are that a different set of rules are required and that zero can have two representations (+0, 0000 0000 and -0, 1000 0000).

Example 1.36. Find the decimal equivalent of the following binary numbers assuming the binary numbers have been represented in sign-magnitude form.

(a) 0101100 **(b)** 101000 **(c)** 1111 **(d)** 011011

Solution.

(a) Sign bit is 0, which indicates the number is positive.
Magnitude 101100 = $(44)_{10}$, **Therefore** $(0101100)_2 = (+44)_{10}$.

(b) Sign bit is 1, which indicates the number is negative.
Magnitude 01000 = $(8)_{10}$, **Therefore** $(101000)_2 = (-8)_{10}$.

(c) Sign bit is 1, which indicates the number is negative.
Magnitude 111 = $(7)_{10}$, **Therefore** $(1111)_2 = (-7)_{10}$.

(d) Sign bit is 0, which indicates the number is positive.
Magnitude 11011 = $(27)_{10}$, **Therefore** $(011011)_2 = (+27)_{10}$.

??? What Complements mean ?

complements are used in the digital computers in order to simplify the subtraction operation and for the logical manipulations. For each radix-r system (radix r represent base of number system) there are two types of complements

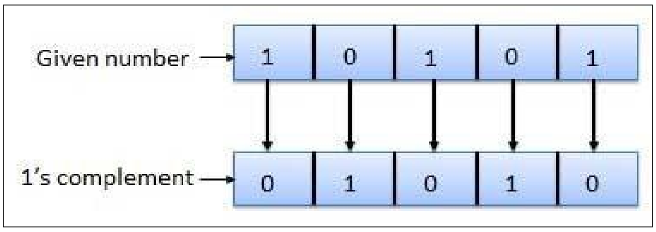
s.n	Complement	Description
1	Radix Complement	The radix complement is referred to as the r's complemen
1	Diminished Radix Complement	The diminished radix complement is referred to as the (r-1)'s complement

There for in Binary system complements has base r = 2. So the two types of complements for the binary system are following:

# 1' complement

The 1's complement of a number is found by changing all 1's to 0's and all 0's to 1's. This is called as taking complement or 1's complement.

Example:



ones' complement can be used to represent negative numbers. The ones' complement form of a negative binary number is the complement of its positive counterpart, which can be obtained by applying the NOT to the positive counterpart. Like sign-magnitude representation, ones' complement has two representations of 0: 00000000 (+0) and 11111111 (−0).

As an example, the ones' complement of 00101011 (43) is 11010100 (−43).

Note: The range of signed numbers using ones' complement in a conventional 8-bit byte is −127 to +127.

Signed integer	Unsigned integer	8 bit ones' complement
0	0	00000000
1	1	00000001
…….	…….	……
125	125	01111101
126	126	01111111
-127	128	10000000
-126	129	10000001
-125	130	10000010
…….	…….	……
-1	245	11111110
-0	255	11111111

+ 1's Complement Addition

To add two numbers represented in this system, we use the conventional binary addition, **but it is then necessary to add any resulting carry back into the**

resulting sum. To see why this is necessary, consider the following example showing the case of the addition of −1 (11111110) to +2 (00000010).

Example: perform (7-3) using 1's complements method ?

```
 Decimal      binary
   7    ---▭ 111   ----▭  111
 - 3   ---▭  011        + 100
_________________________________
                       ¹ 011
                          +  1
                       _______
                          100  the result
```

Example: perform (8-12) using 1's complements method ?

```
 Decimal      binary
   8    -▭    1000  --▭    1000
 - 12  -▭     1100  --▭ +  0011
 __________              ________
  - 4                      1011
```

# 2' complement

The 2's complement of binary number is obtained by adding 1 to the Least Significant Bit (LSB) of 1's complement of the number.

Note: 2's complement = 1's complement + 1
Example of 2's Complement is as follows.

The Two's complement representation allows the use of binary arithmetic operations on signed integers, yielding the correct 2's complement results.

Positive Numbers

Positive 2's complement numbers are represented as the simple binary.

Negative Numbers

Negative 2's complement numbers are represented as the binary number that when added to a positive number of the same magnitude equals zero.

Integer		2's Complement
Signed	**Signed**	
5	**5**	**0000 0101**
4	**4**	**0000 0100**
3	**3**	**0000 0011**
` `2	**2**	**0000 0010**
1	**1**	**0000 0001**
0	**0**	**0000 0000**
-1	**255**	**1111 1111**
-2	**254**	**1111 1110**
-3	**253**	**1111 1101**
-4	**252**	**1111 1100**
-5	**251**	**1111 1011**

Note: The most significant (leftmost) bit indicates the sign of the integer; therefore it is sometimes called the sign bit.

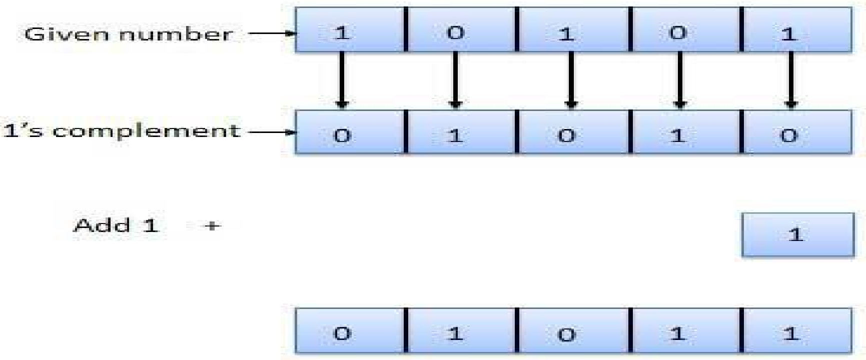
If the sign bit is zero, **then** the number is greater than or equal to zero, or positive.

If the sign bit is one, **then** the number is less than zero, or negative.

+ Calculation of 2's Complement

To calculate the 2's complement of an integer, invert the binary equivalent of the number by changing all of the ones to zeroes and all of the zeroes to ones (also called **1's complement**), and then add one.

Notes: The addition of ***n*-bit** signed binary numbers is straightforward using the 2's complement system. **The addition is carried out just as if all the numbers were positive, and any carry from the sign position is ignored**. This will always yield the correct result except when an overflow occurs. When the word length is *n* bits, we say that an *overflow* has occurred if the correct representation of the sum (including sign) requires more than *n* bits.



For example:

17 ⇔ **-17**
0001 0001(binary 17) ⇔ 1110 1111(two's complement -17)

NOT(0001 0001) = 1110 1110 (**Invert bits**)

1110 1110 + 0000 0001 = **1110 1111** (Add 1)

Home work :Now you try some :Find the two's complement for

a. - 11
b. - 43
c. - 123

+ 2's Complement Addition

Two's complement addition follows the same rules as binary addition.

For example:

```
5 + (-3) = 2    0000 0101 = +5
              + 1111 1101 = -3
                ______________
                0000 0010 = +2
```

+ 2's Complement Subtraction

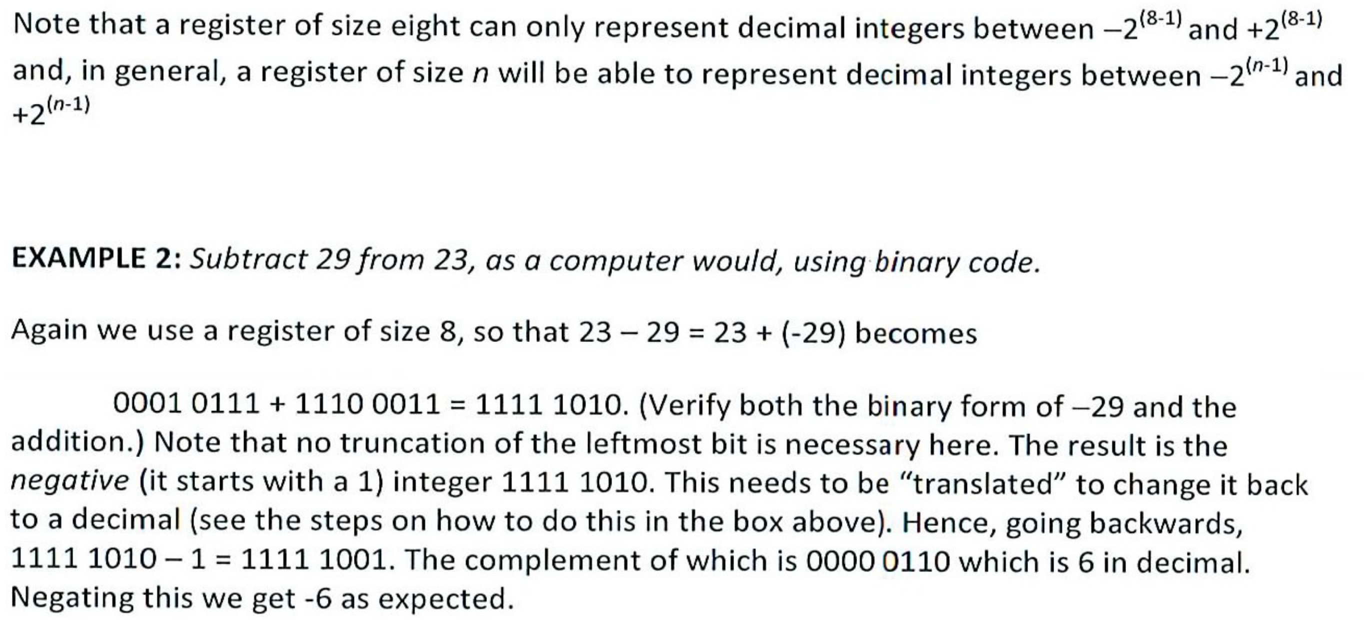
Two's complement subtraction is the binary addition of the minuend to the 2's complement of the subtrahend (adding a negative number is the same as subtracting a positive one).

For example:

```
7 - 12 = (-5)        0000 0111 = +7
                   + 1111 0100 = -12
                   _________________
                     1111 1011 = -5
```

Note: this suggests a new way to subtract in binary due to the fact that subtraction is defined in the following manner :

$$x-y= x+ (-y)$$



Note that a register of size eight can only represent decimal integers between $-2^{(8-1)}$ and $+2^{(8-1)}$ and, in general, a register of size *n* will be able to represent decimal integers between $-2^{(n-1)}$ and $+2^{(n-1)}$

EXAMPLE 2: *Subtract 29 from 23, as a computer would, using binary code.*

Again we use a register of size 8, so that 23 – 29 = 23 + (-29) becomes

0001 0111 + 1110 0011 = 1111 1010. (Verify both the binary form of –29 and the addition.) Note that no truncation of the leftmost bit is necessary here. The result is the *negative* (it starts with a 1) integer 1111 1010. This needs to be "translated" to change it back to a decimal (see the steps on how to do this in the box above). Hence, going backwards, 1111 1010 – 1 = 1111 1001. The complement of which is 0000 0110 which is 6 in decimal. Negating this we get -6 as expected.

Home work :Now you try some : subtract each , as a computer out , using a binary code using registers of size 8 :

a. 26 -15
b. – 31 - 6
c. 144 – 156
d. make up your own exercises as needed

4- Binary Codes

This section describes various popular Binary Code formats. 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

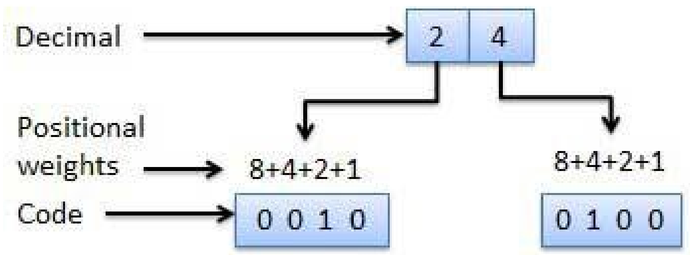
The codes are broadly categorized into following six categories.

- 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.



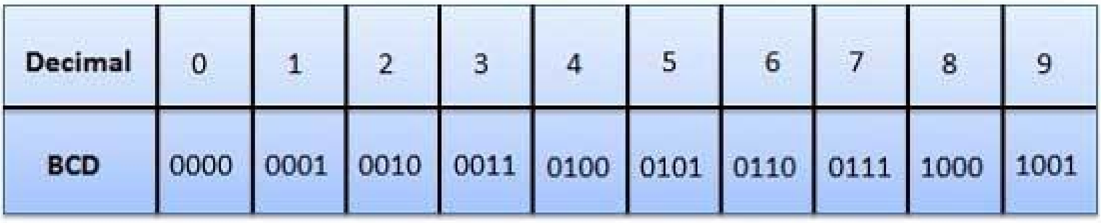
+Non-Weighted Codes

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.

- 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).

Note: *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 Code

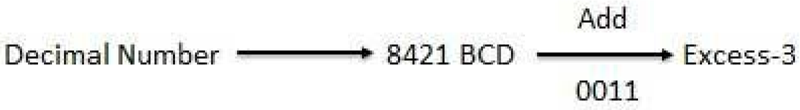
- It is very similar to decimal system.
- We need to remember binary equivalent of decimal numbers 0 to 9 only.

Disadvantages of BCD Code

- 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.

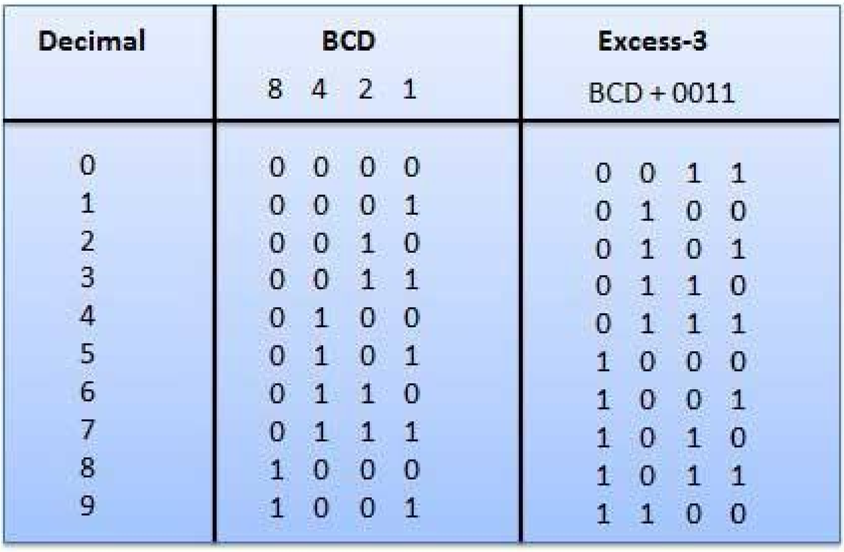
-Excess-3 Code

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:



$$\text{Decimal Number} \xrightarrow{} \text{8421 BCD} \xrightarrow[\text{0011}]{\text{Add}} \text{Excess-3}$$

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

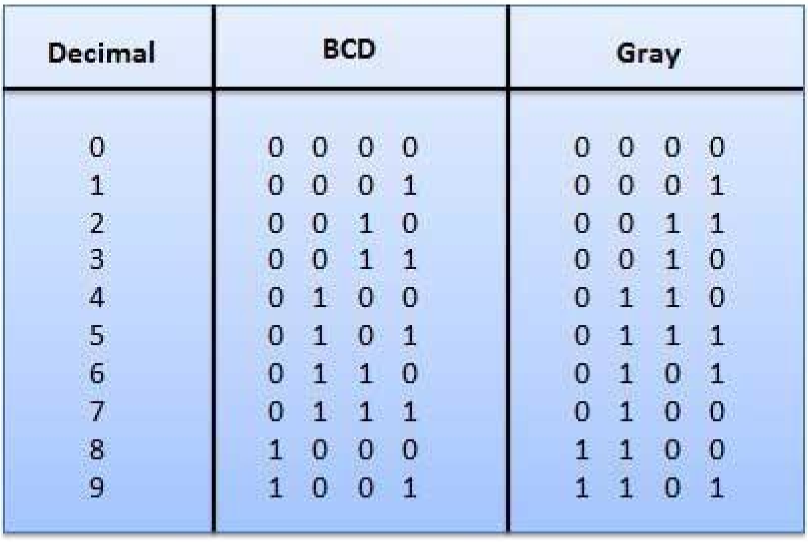
- Gray Code

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.

*Applications 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.



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

5- Codes Conversion

This section describes conversion of various digital code formats to one another. 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 Gray
- Gary to binary

+ 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: 11101_2
Calculating Decimal Equivalent:

Step 1: 111012= $((1 \times 2^4) + (1 \times 2^3) + (1 \times 2^2) + (0 \times 2^1) + (1 \times 2^0))_{10}$
Step 2: 111012= $(16 + 8 + 4 + 0 + 1)_{10}$
Step 3: 111012 $=29_{10}$

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

Step 2 – Convert to BCD

Decimal Number: 29_{10}

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

Step	Decimal	Number Conversion
Step 1	29_{10}	$00102\ 1001_2$
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 Decimal

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: $(00101001)_{\mathbf{BCD}}$ = **Decimal Number**: 2

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

Note :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)$_{BCD}$ = 93

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)$_{BCD}$

+ Gray To Binary Conversion

Steps

Step1: The **MSB** in the left is the **MSB** in binary number. In other word they are The same.

Step2: Add the first digit of the binary number to the second digit in Gary code, the carry is ignored , in other word , take XOR operation between them .

Step3: Generally working from the left to right digit, the n'th digit in the binary number is formed from summing the (n+1)'th digit in the binary number with n'th bit in the Gray code .

Example: Convert the Gary code 11011 to binary number?

Gray code ㅂ **1 1 0 1 1**
ㅇ ㅊ ㅇ ㅊ ㅇ ㅊ ㅇ ㅊ ㅇ
Binary Number ㅂ **1 0 0 1 0**

Then (11011)$_G$= (10010)$_2$

+ Binary To Gray Conversion

Steps:

Step1: The **MSB** digit in Gary code is the same as corresponding digit in the binary number.

Step2: going from left to right , add each adjacent pair of binary digit to get the next Gray digits , regardless carries .

Example: Convert the following binary number 100110 to Gray code?

Binary Number ⇒ 1 0 0 1 1 0

Gray Code ⇒ 1 1 0 1 0 1

Then (100110)2 =$(110101)_G$

└ **Home work :**

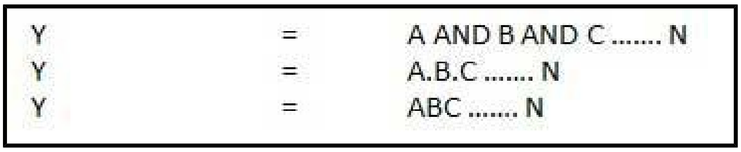
1- convert the BCD code $(1001010100000100)_{BCD}$ decimal number
2- $(12)_{10}$ +$(9)_{10}$ using BCD code
3- convert the gray code 11011 to binary number?
4- Covert (101110101)Gary = (?)2

5- Logic Gates

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

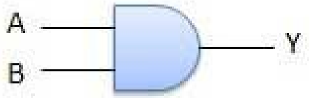
1-AND Gate

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

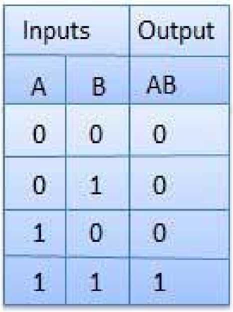


Y	=	A AND B AND C .... N
Y	=	A.B.C .... N
Y	=	ABC .... N

- **Logic Diagram**



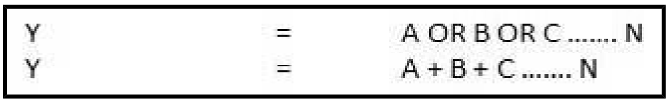
- **Truth Table**



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

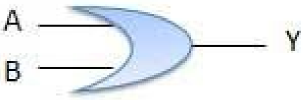
2- OR Gate

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

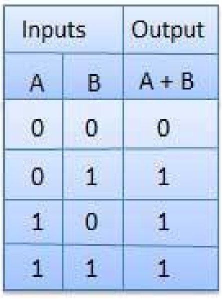


Y = A OR B OR C .... N
Y = A + B + C .... N

- **Logic Diagram**



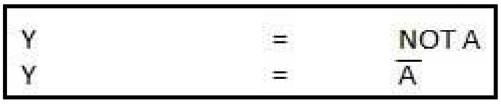
- **Truth Table**



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

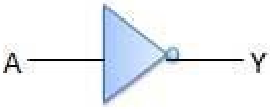
3-NOT Gate

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

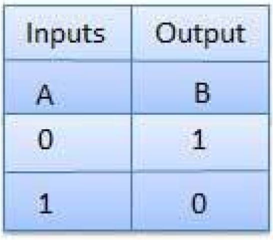


Y = NOT A
Y = $\overline{A}$

- **Logic Diagram**



- **Truth Table**



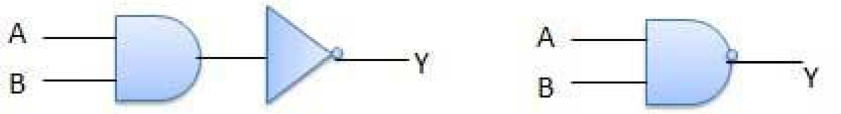
Inputs	Output
A	B
0	1
1	0

4-NAND Gate

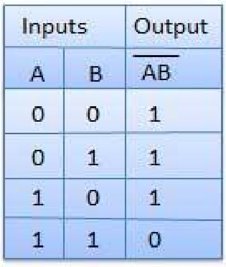
A NOT-AND operation is known as NAND operation. It has n input (n >= 2) and one output.

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

- **Logic Diagram**



- **Truth Table**



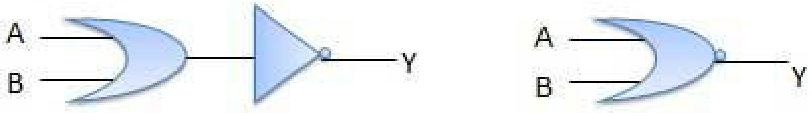
Inputs		Output
A	B	$\overline{AB}$
0	0	1
0	1	1
1	0	1
1	1	0

5-NOR Gate

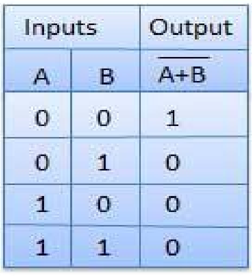
A NOT-OR operation is known as NOR operation. It has n input (n >= 2) and one output.

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

6- XOR Gate

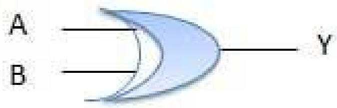
XOR or Ex-OR gate is a special type of gate. It can be used in the half adder, full adder and subtractor. The exclusive-OR gate is abbreviated as EX-OR gate or sometime as X-OR gate. It has n input (n >= 2) and one output.

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

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

Y = $\overline{AB} + \overline{AB}$

- **Logic Diagram**



- **Truth Table**

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

7- XNOR Gate

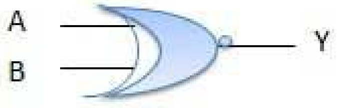
XNOR gate is a special type of gate. It can be used in the half adder, full adder and subtractor. The exclusive-NOR gate is abbreviated as EX-NOR gate or sometime as X-NOR gate. It has n input (n >= 2) and one output.

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

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

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

- **Logic Diagram**



- **Truth Table**

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

6- Logic circuits (networks).

Logic gates can be combined together to produce more complex logic circuits (networks).

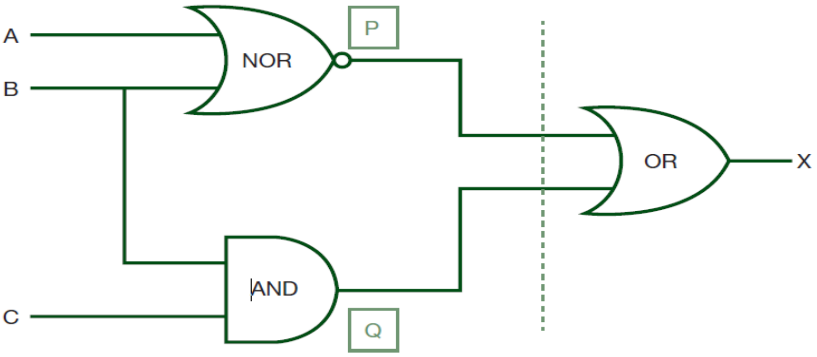
Note: The output from a logic circuit (network) is checked by producing a truth table.

Two different types of problem are considered here:

- drawing the truth table from a given logic circuit (network)
- designing a logic circuit (ne twork) from a given problem and testing it by also drawing a truth table.

Example 1

Produce a truth table from the following logic circuit (network).



To show how this works, we will split the logic circuit into two parts (shown by the dotted line).

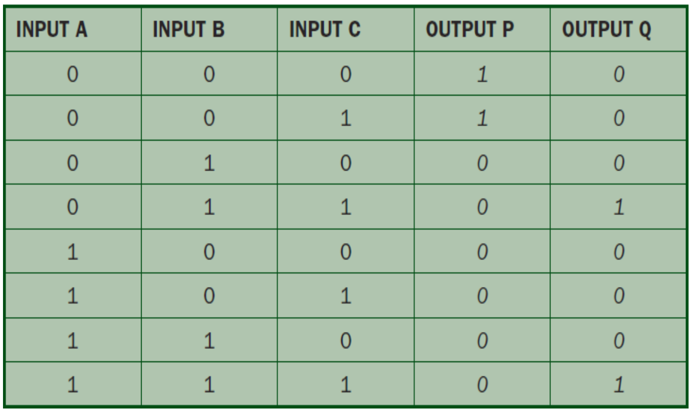
-First part

There are 3 inputs; thus we must have 23 (i.e. 8) possible combinations of 1s and 0s. To find the values (outputs) at points **P** and **Q**, it is necessary to consider the truth tables for the NOR gate (output P) and the AND gate (output Q) i.e.

P = A **NOR** B

Q = B **AND** C

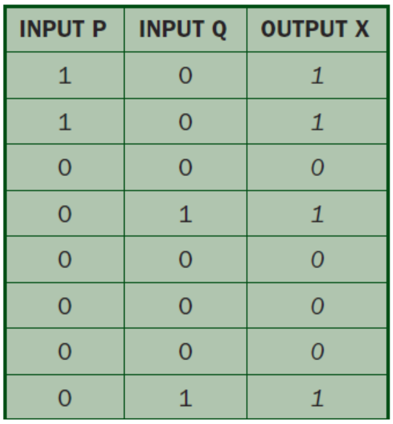
We thus get:



INPUT A	INPUT B	INPUT C	OUTPUT P	OUTPUT Q
0	0	0	1	0
0	0	1	1	0
0	1	0	0	0
0	1	1	0	1
1	0	0	0	0
1	0	1	0	0
1	1	0	0	0
1	1	1	0	1

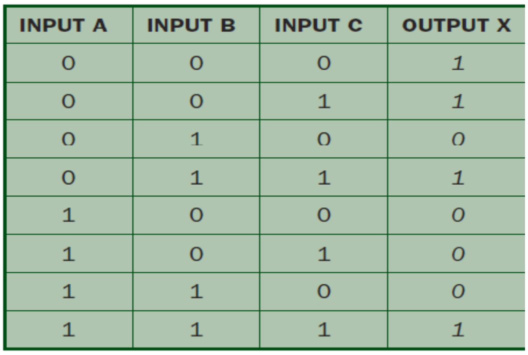
-Second part

There are 8 values from **P** and **Q** which form the inputs to the last **OR** gate. Hence we get X = P **OR** Q which gives the following truth table:



INPUT P	INPUT Q	OUTPUT X
1	0	1
1	0	1
0	0	0
0	1	1
0	0	0
0	0	0
0	0	0
0	1	1

Which now gives us the final truth table for the logic circuit given at the start of the example:



INPUT A	INPUT B	INPUT C	OUTPUT X
0	0	0	1
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	0
1	1	1	1

Example 2

Consider the following problem.

A system used 3 switches A, B and C; a combination of switches determines whether an alarm, X, sounds:

If switch A or switch B are in the ON position and if switch C is in the OFF position then a signal to sound an alarm, X is produced. It is possible to convert this problem into a logic statement.

So we get:

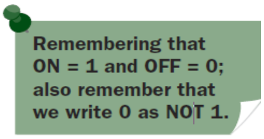
If (A = 1 **OR** B = 1)
The first part is two inputs (A and B) joined by an **OR** gate

AND
The output from the first part and the third part are joined by an **AND** gate

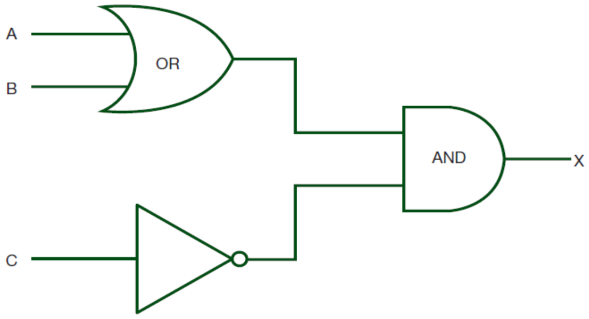
(C = **NOT** 1)
The third part is one input (C) which is put through a **NOT** gate

then X = 1

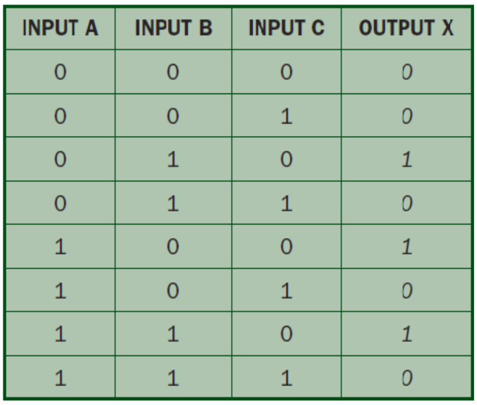
So we get the following logic circuit (network):



Remembering that ON = 1 and OFF = 0; also remember that we write 0 as NOT 1.



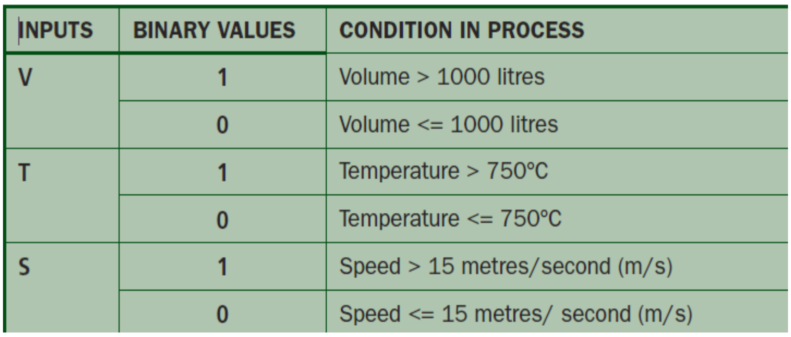
This gives the following truth table:



INPUT A	INPUT B	INPUT C	OUTPUT X
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	0

Home works :

***Qutation1*: A manufacturing process is controlled by a built in logic circuit which is made up of AND, OR and NOT gates only. The process receives a STOP signal (i.e. X = 1) depending on certain conditions, shown in the following table:



INPUTS	BINARY VALUES	CONDITION IN PROCESS
V	1	Volume > 1000 litres
	0	Volume <= 1000 litres
T	1	Temperature > 750°C
	0	Temperature <= 750°C
S	1	Speed > 15 metres/second (m/s)
	0	Speed <= 15 metres/ second (m/s)

A stop signal (X = 1) occurs when:

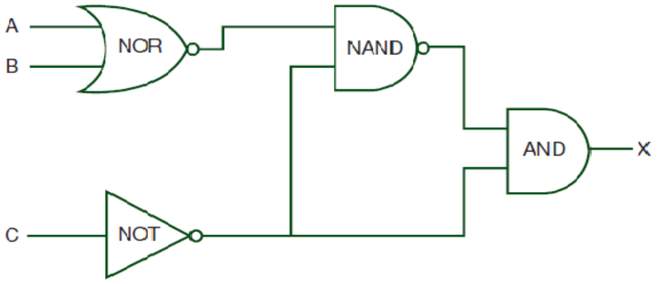
either Volume, V > 1000 litres and Speed, S <= 15 m/s
or Temperature, T <= 750:C and Speed, S > 15 m/s

Draw the logic circuit and truth table to show all the possible situations when the stop signal could be received.

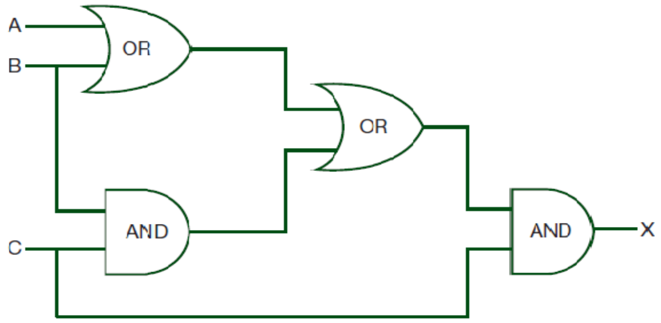
Qutation2:

produce truth tables from the given logic circuits (networks). Remember, if there are two inputs then there will be 4 possible outputs; if there are three inputs then there will be 8 possible outputs.

9.4.3



9.4.4



Important Notes:

1-XOR Notes:

- *The logic function implemented by a 2-input Ex-OR is given as either: "* $Q = \left(A \oplus B\right) = \overline{A}.B + A\,.\,\overline{B}$ *" , which mean "A OR B but NOT both" will give an output at Q.*

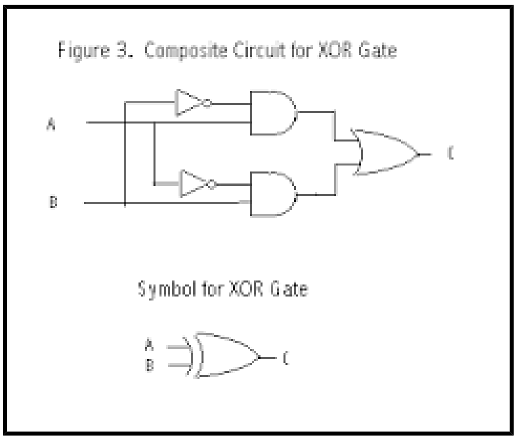


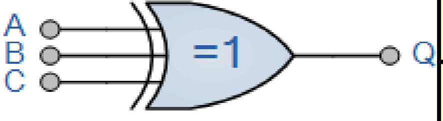
Figure 3. Composite Circuit for XOR Gate

Symbol for XOR Gate

- *in general, an Ex-OR gate will give an output value of logic "1" ONLY when there are an **ODD** number of 1's on the inputs to the gate, if the two numbers are equal, the output is "1".*

- *Then an Ex-OR function with more than two inputs is called an "odd function" or modulo-2-sum (Mod-2-SUM), not an Ex-OR.*

- *This description can be expanded to apply to any number of individual inputs as shown below for a 3-input Ex-OR gate.*

Symbol	Truth Table			
	C	**B**	**A**	**Q**
 **3-input Ex-OR Gate**	0	0	0	0
	0	0	1	1
	0	1	0	1
	0	1	1	0
	1	0	0	1
	1	0	1	0
	1	1	0	0
	1	1	1	1
Boolean Expression Q = A ⊕ B ⊕ C	**"Any ODD Number of Inputs" gives Q**			

2-XNOR Notes :

- *The logic function implemented by a 2-input Ex-NOR gate is given as* $Q = \overline{(A \oplus B)} = \overline{A.B} + A.B$ *,which mean "when both A AND B are the SAME" will give an output at Q.*
- *In general, an Exclusive-NOR gate will give an output value of logic "1" ONLY when there are an EVEN number of 1's on the inputs to the gate (the inverse of the Ex-OR gate) except when all its inputs are "LOW" or "0".*
- *Then an Ex-NOR function with more than two inputs is called an "even function" or modulo-2-sum (Mod-2-SUM), not an Ex-NOR.*

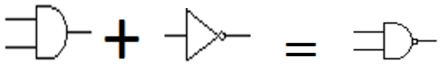
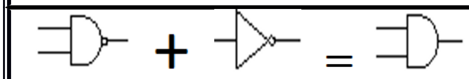
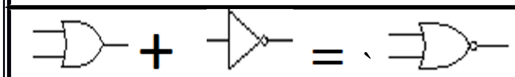
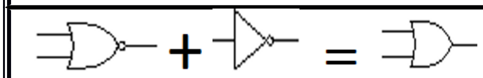
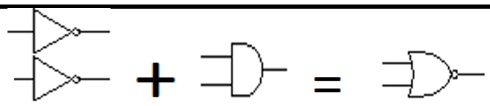
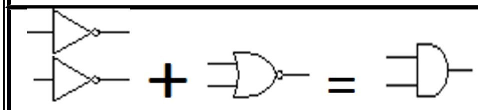
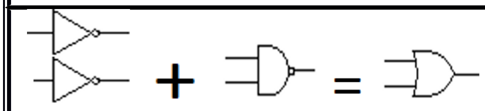
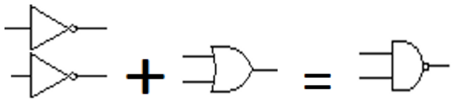
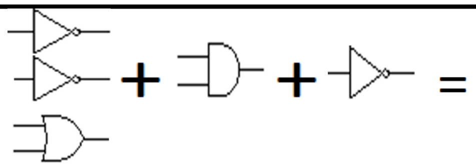
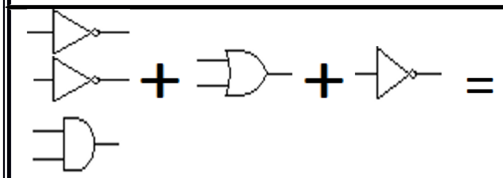
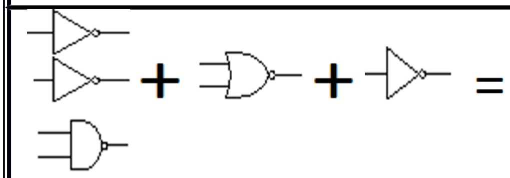
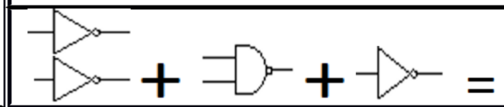
3-Digital Logic Gates Summary

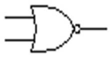
The following logic gates truth table compares the logical functions of the 2-input logic gates .

Inputs		Truth Table Outputs For Each Gate					
A	B	AND	NAND	OR	NOR	EX-OR	EX-NOR
0	0	0	1	0	1	0	1
0	1	0	1	1	0	1	0
1	0	0	1	1	0	1	0
1	1	1	0	1	0	0	1

3- Using inverters to Convert Gates

Frequently it is convenient to convert a basic gate such as an **AND** , **OR** , **NAND** , or **NOR** to another logic function . this can be done easily with the use of inverters (**NOT gate**). The following chart is a handy guide for converting any given gate to any other logic function.

Invert outputs		AND TO NAND
		NAND TO AND
		OR TO NOR
		NOR TO OR
Invert inputs		AND TO NOR
		NOR TO AND
		NAND TO OR
		OR TO NAND
Invert inputs and outputs		AND TO OR
		OR TO AND
		NOR TO NAND
		NAND TO NOR

⌙ **Home work:**

write the Truth table for each case in the table above.

6-Boolean Algebra

This section describes various mathematic laws of 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. *variable* , *complement* and *literal* are terms used in Boolean Algebra .

A **variable** : **is** a symbol used to represent an action , a condition , or data. Any single variable can have only a 1 or a 0 value.

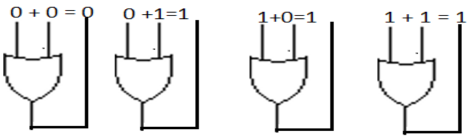
Complement: is the inverse of a variable and is indicated by a bar over the variable (overbar) .for example , the complement of A is $\bar{A}$.

A Literal: is a variable or the complement of a variable.

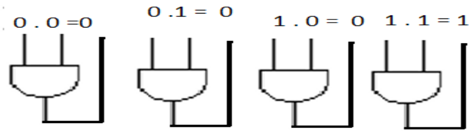
+ Rule in Boolean algebra

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 an over bar (-). Thus complement of variable B is represented as . Thus if B = 0 then $\bar{B}$ = 1 and B = 1 then $\bar{B}$ = 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.** its also equivalent to the **OR** operation as illustrated as follows :



- 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.** its also equivalent to the **and** operation as illustrated as follows :



+Boolean Laws

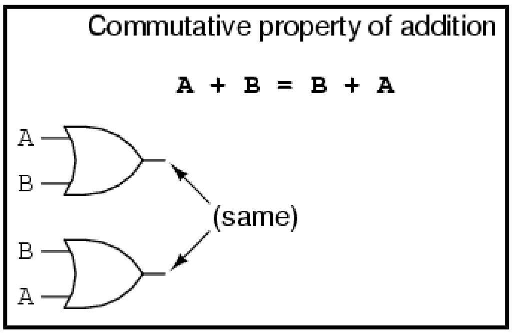
There are six types of Boolean Laws.

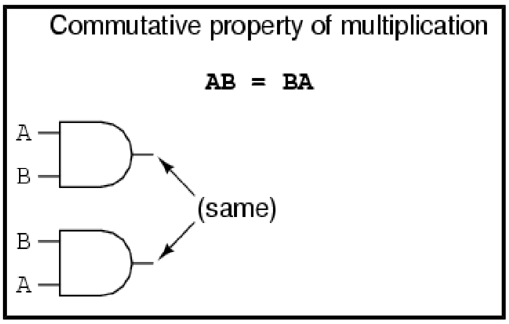
1- COMMUTATIVE LAW

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. Remember, Boolean Algebra as applied to logic circuits ,the commutative law can applied to OR and AND gate makes no difference , as show in next figures.





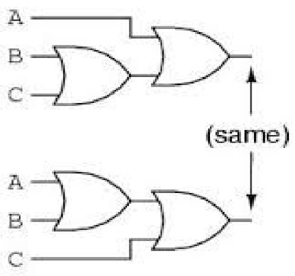
2- ASSOCIATIVE LAW

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

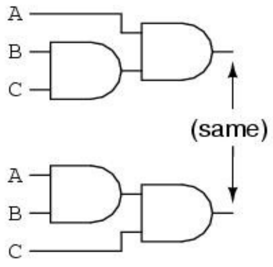
(i) (A . B) .C = A. (B .C) (ii) (A +B)+ C = A +(B +C)

The follows figures show how to applied the associative low to 2-input OR gates and 2-input And gates.

A + (B + C)=(A +B) +C



A(BC) = (AB)C

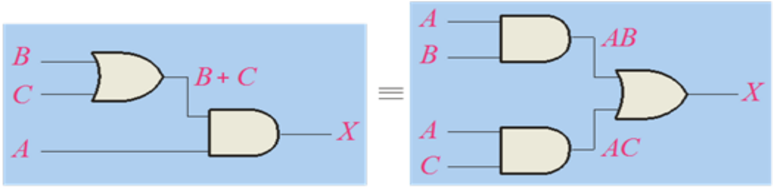


3- DISTRIBUTIVE LAW

Distributive law states the following condition

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

The follows figures show how to applied the distributive low to 2-input OR gates and 2-input And gates.



$A(B+C)$ $AB+AC$

Where the symbol ≡ mean "equivalent to"

Rules of Boolean Algebra

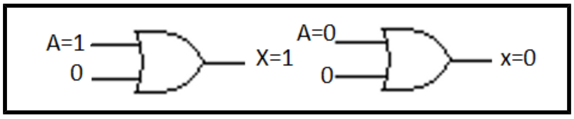
The following table lists the 12 basic rules that are useful in manipulating and simplifying **Boolean expressions** .Rules 1 through 9 will be viewed in terms of their application to logic gates. Rules 10 through 12 will be derived in term of the simpler rules and law previously discussed.

No.	Rule	No.	Rule
1	A +0 =A	7	A .A = A
2	A + 1= 1	8	A. $\bar{A}$ = 0
3	A . 0=0	9	$\bar{\bar{A}}$ = A
4	A .1 = A	10	A +AB =A
5	A +A= A	11	A + $\bar{A}$ B=A +B
6	A + $\bar{A}$ =1	12	(A+B)(A+C)=A+BC

Notes: A ,B or C can represent a single variable or a combination of variables

Rule 1 : A + 0 = A (Identity Law)

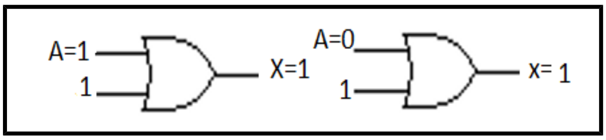
The variable ORed with 0 is always equal to the variable . This rule is illustrated in the following Figure , where the lower input is fixed at 0.



X=A+0=A

Rule 2 : A + 1 = 1 (NULL Elements Law)

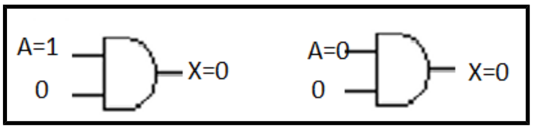
A variable ORed with 1 is always equal to 1. This rule is illustrated in the following Figure , where the lower input is fixed at 1.



X=A+1=1

Rule 3 : A . 0 = 0 (NULL Elements Law)

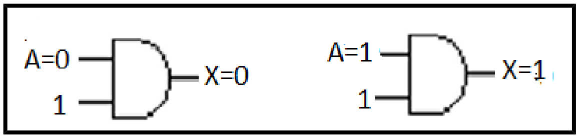
A variable ANDed with 0 is always equal to 0. This rule is illustrated in the following Figure , where the lower input is fixed at 0.



X=A . 0=0

Rule 4 : A .1 = A (Identity Law)

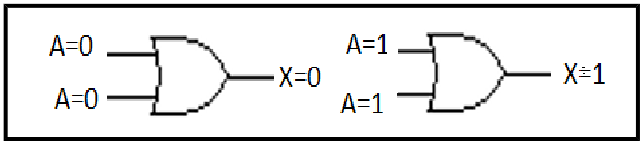
A variable ANDed with 1 is always equal to the variable . This rule is illustrated in the following Figure , where the lower input is fixed at 1.



X=A . 1=A

Rule 5 : A + A= A (Idempotent Law)

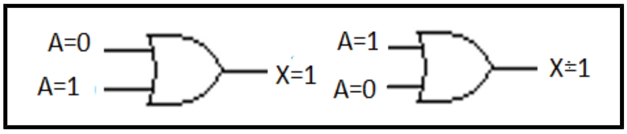
A variable ORed with itself is always equal to the variable . This rule is illustrated in the following Figure , where both inputs are the same variable .



X=A +A=A

Rule 6 : A + $\bar{A}$ = 1

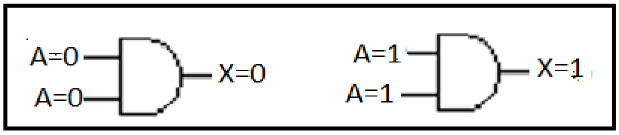
A variable ORed with its complement is always equal to 1. This rule is illustrated in the following Figure , where one input is the complement pf the other.



X=A + $\bar{A}$ =1

Rule 7: A .A = A (Idempotent Law)

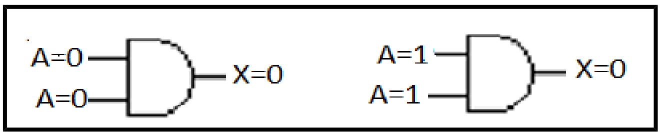
A variable Anded with itself is always equal to the variable . This rule is illustrated in the following Figure , where both inputs are the same variable .



X=A . A= A

Rule 8: A . $\bar{A}$ = 0 (Complement Law)

A variable ANDed with its complement is always equal to 0. This rule is illustrated in the following Figure.



X=A . $\bar{A}$ =0

Rule 9 : $\bar{\bar{A}}$ = A (Complement Law)

The double complement of a variable is always equal to the variable. This rule is illustrated in the following Figure using inverters .

$$\bar{\bar{A}} = A$$

Rule 10: A + AB = A

This rule can be proved by applying the distributive law , rule 2 and rule 4 as follows:

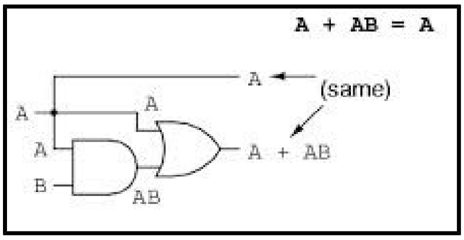
A + AB = A .1 + AB= A(1 + B) factoring (distributive law)
=A . 1 Rule2 :(1 +b)=1
=A Rule 4: A . 1 =A

Note : the proof is shown in table bellow , which shows the troth table and the resulting logic circuit simplification .

1- troth table

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

2- logic circuit



Rule 11: A + $\bar{A}$ B= A + B

This rule can be proved as follows :

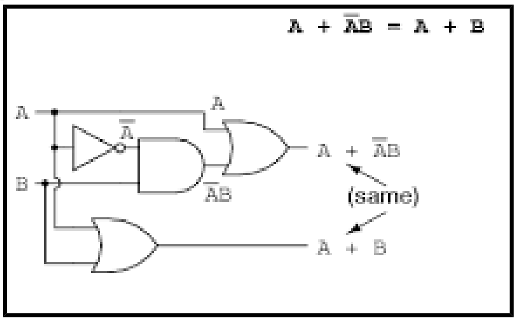
A + $\bar{A}$ **B=**(A +AB) + $\bar{A}$ **B** Rule 10: A= A+ AB
= (AA +AB) + $\bar{A}$ **B** Rule 7 : A=AA
=AA +AB +A $\bar{A}$ + $\bar{A}$ **B** Rule 8:adding A $\bar{A}$ **=0**
=(A + $\bar{A}$)(A + B) factoring
= 1 . (A + B) Rule 6: A + $\bar{A}$ =1
= A + B Rule4 :drop the 1

Note : the proof is shown in table bellow , which shows the troth table and the resulting logic circuit simplification

1- troth table

A	**B**	$\bar{A}$ **B**	**A +** $\bar{A}$ **B**	**A + B**
0	0	0	0	0
0	1	1	1	1
1	0	0	1	1
1	1	0	1	1

2- logic circuit



Rule 12 : (A + B)(A + C) = A+BC

This rule can be proved as follows :

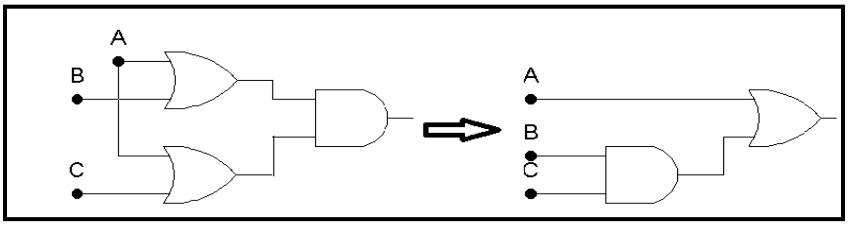
$$\begin{aligned}\mathbf{(A + B)(A+ C)} &= AA + AC + AB + BC\\ &= A + AC + AB + BC\\ &= A(1+C) + AB + BC\\ &= A \cdot 1 + AB + BC\\ &= A(1 + B) + BC\\ &= A + BC\end{aligned}$$

Note : the proof is shown in table bellow , which shows the troth table and the resulting logic circuit simplification

1- Troth Table

A	B	C	A+B	A+C	(A+B) (A +C)	BC	A+BC
0	0	0	0	0	0	0	0
0	0	1	0	1	0	0	0
0	1	0	1	0	0	0	0
0	1	1	1	1	1	1	1
1	0	0	1	1	1	0	1
1	0	1	1	1	1	0	1
1	1	0	1	1	1	0	1
1	1	1	1	1	1	1	1

1- Logic Circuit



+Important Boolean Theorems

Following are few important Boolean functions and theorems.

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

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

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

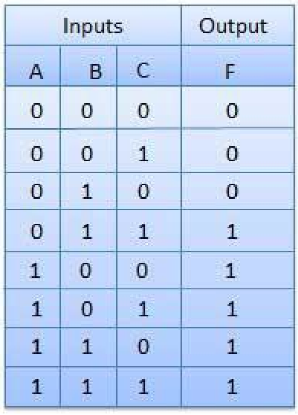
$$Y = A + \overline{BC} + ADC$$

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

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**.

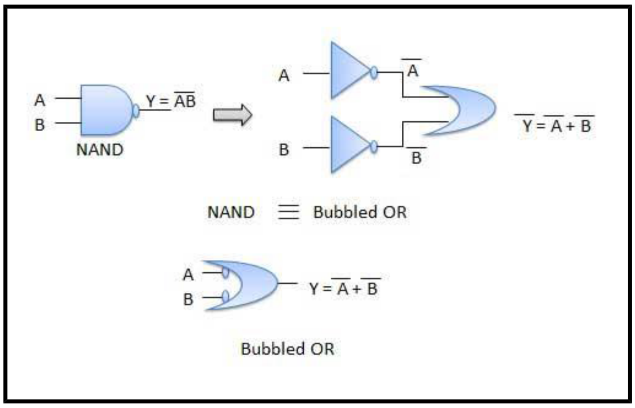
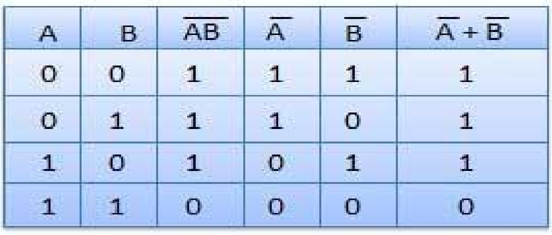
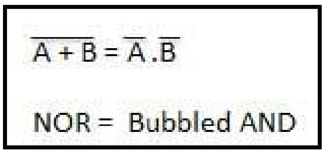


Table showing verification of the De-Morgan's first theorem



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**.

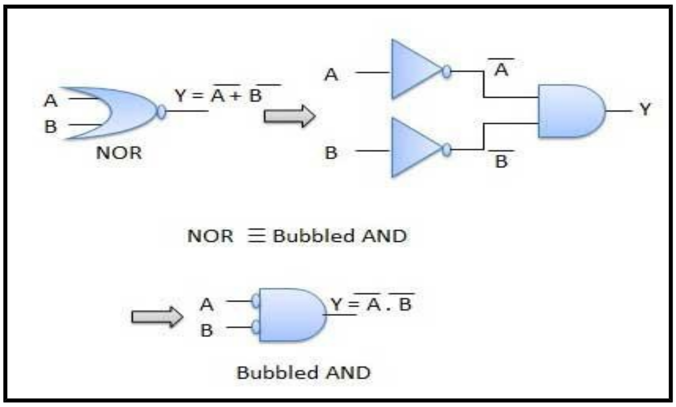
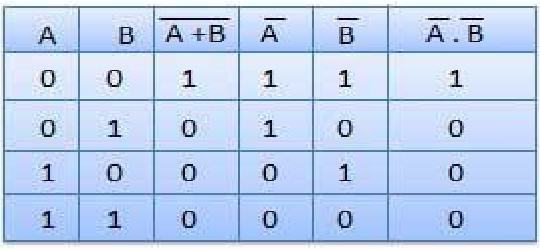


Table showing verification of the De-Morgan's second theorem



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

Example : Apply DemMorgan's theorems to the following expression :

$$1-\ \overline{XYZ} = \overline{X} + \overline{Y} + \overline{Z}$$

$$2-\ \overline{X+Y+Z} = \overline{X}\ \overline{Y}\ \overline{Z}$$

. **That mean :**

$$1-\ \overline{(A\ .\ B\ .\ Z.\ \dots\dots\dots)} = \overline{X} + \overline{Y} + \overline{Z} + \dots\dots\dots$$

$$2-\ \overline{(X + Y + Z + \dots\dots\dots)} = \overline{X}\ .\ \overline{Y}\ .\ \overline{Z}.\ \dots\dots\dots$$

Simplification ***Using Boolean Algebra_***

Many times in the application of Boolean algebra , you have to reduce a particular expression to its simplest form or change its form to a more convenient one to implement the expression most efficiently .

the approach taken un this section is to use the basic laws , and theorems of Boolean algebra to manipulate and simplify an expression .

This method depends on a thorough knowledge of Boolean algebra and considerable practice in its application , not to mention a little ingenuity and cleverness.

Examples: **using Boolean Algebra techniques , simplify the following expression:**

$1-X\left(\overline{X}+Y\right)$

$=X\overline{X}+XY$

$=XY$

$2-AB\left(C+\overline{A}\,\overline{C}\right)$

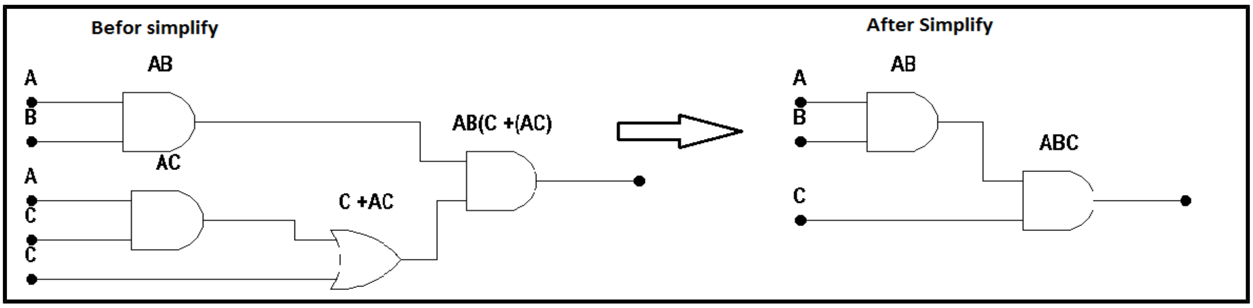
$=ABC+AB\overline{A}\,\overline{C}$

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

$=ABC+0\,.\,B\overline{C}$

$=ABC+0$

$=ABC$



$3-ABC+A\overline{B}C+AB\overline{C}$

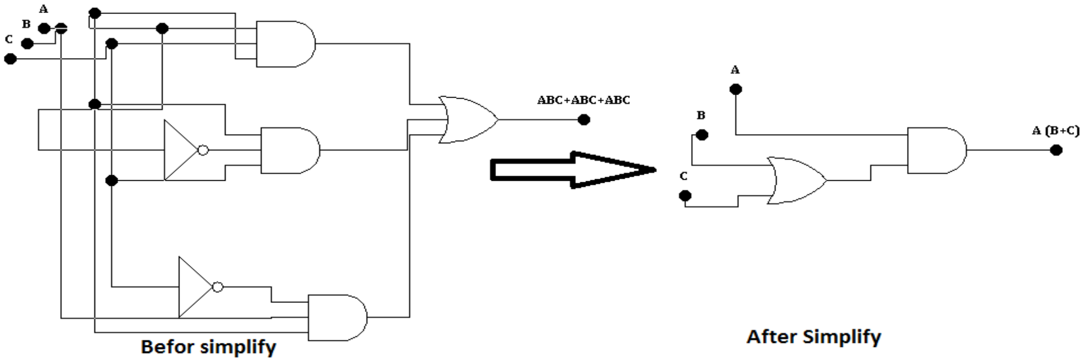
$=AC\left(B+\overline{B}\right)+AB\overline{C}$

$=AC\,.\,1+AB\overline{C}$

$=AC+AB\overline{C}$

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

$=A\left(C+B\right)$



3- AB + A(B+C)+ B(B+C)

= AB +AB +AC +BB +BC by applying distributive law

= AB +AB +AC +B +BC by applying rule 7

= AB +AC +B +BC by applying rule 5

=AB+AC+B by applying rule 10

=B+AC by applying rule 10

As this point the expression is simplified as much as possible.

***Home work* : Draw the logic circuit before and after simplification .**

$$4-\left(C\oplus B\right)\oplus A$$

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

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

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

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

$$=\left(\,CB+C\,\overline{C}+\overline{B}\,B+\overline{B}\,\overline{C}\right).A+\overline{A}\,B\,\overline{C}+\overline{A}\,\overline{B}\,C$$

$$=ABC+A\,\overline{B}\overline{C}+\overline{A}\,B\overline{C}+\overline{A}\,\overline{B}\,C$$

Home work

1 - Simplify the following expression and Draw the logic circuit before and after simplification

$$a)\ A\,\overline{B}+A\left(\overline{A+C}\right)+B\left(\overline{B+C}\right)$$

$$b)\ \overline{A}\,BC+A\overline{B}\,\overline{C}+\overline{A}\,\overline{B}\,\overline{C}+A\,\overline{B}\,C+ABC$$

2- Applying DeMorgan's theorem Draw the logic circuit before and after simplification

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

$$b)\ \overline{ABC+DEF}$$

$$c)\ \overline{A\overline{B}+\overline{C}D+EF}$$

5- Simplifying Logic Circuits

Two methods for simplifying

– Algebraic method (use Boolean algebra theorems)

– Karnaugh mapping method (systematic, step-by-step approach)

ـH-Standard Form Of Boolean Expressions

All Boolean expressions , regardless of their form , can be converted into two standard forms:

- **The sum- of – products form.**
- **The product –of- sums form.**

Standardization makes the evaluation , simplification , and implementation of Boolean expressions much more systematic and sassier.

¬ The Sum – of – Products (SOP) form

When two or more product terms are summed by Boolean addition , the resulting expression is **a sum-of-products (sop).** Some examples are.

$$AB+ABC$$
$$ABC+CDE+\overline{B}C\overline{D}$$
$$\overline{A}B+\overline{A}B\overline{C}+AC$$

In an Pos expression , a single overbar cannot extend over more than one variable in a term can have an overbar . for example , an SOP expression can have the term $\bar{A}\bar{B}\bar{C}$ *but not* $\overline{ABC}$.

- Domain of a Boolean Expression

The **Domain** of a general Boolean expression is the set of variables contained in the expression in either complement or un complement form.

For example , the domain of the expression $\bar{A}B + A\bar{B}C$ is the set of variables A,B ,C and the domain of the expression $AB\bar{C} + C\bar{D}E + \bar{B}C\bar{D}$ is the set of variables A , B , C , D , E.

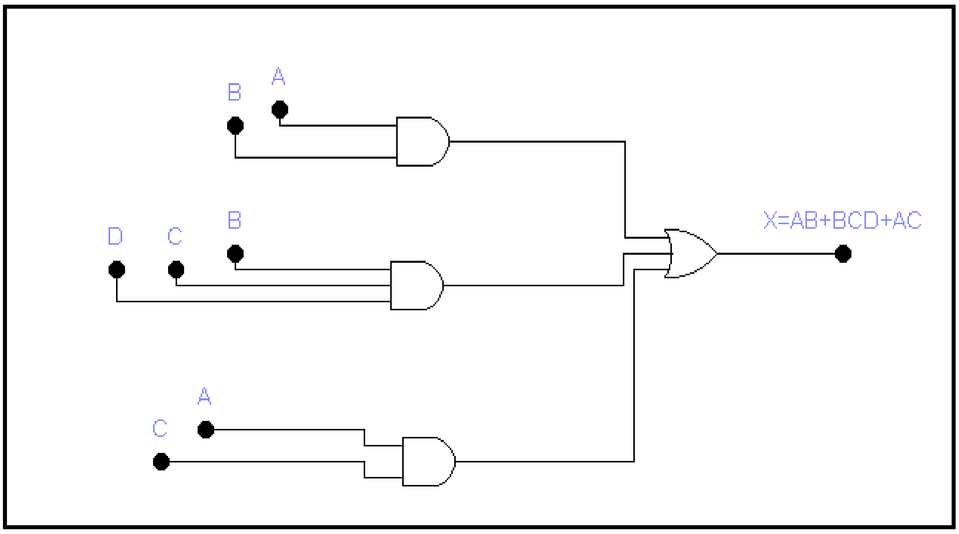
- And / Or implementation of an SOP expression

Implementation an SOP expression is simply require

1- ORing the output of two or more AND gates. A product term is produced by an AND operation.

2- And the sum (addition) of two or more product terms is produced by an OR operation.

Note : Therefore, an **SOP** expression can be implemented by **AND-OR** logic in which the outputs of a number (equal to the number of product terms in the expression) of **AND** gates connect to the inputs of an **OR** gate , as show un the next figure , for the expression **AB+BCD +AC** . the output **X** of the **OR** gate equals the **SOP** expression .



Conversion of a General Expression to SOP Form

Any logic expression can be changed into **SOP** form by applying Boolean algrbra techniques. For example , the expression **A(B+CD)** can be converted to SOP form by applying the distributive law:

A (B +CD)= A +ACD

Example: convert each of the following Boolean expression to the SOP from:

(a) AB + B(CD + EF)

Solution:

AB +BCD +BEF

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

$Soluation:$

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

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

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

Home work

Convert $\overline{A}B\overline{C} + \left(A+\overline{B}\right)\left(B+\overline{C}+A\overline{B}\right)$ **to SOP Form**

The Standard SOP Form

So far , you have been SOP expressions in which some of the product terms do not contain all of variables in the domain of the expression. For example, the expression $\overline{A}B\overline{C} + A\overline{B}D + \overline{A}B\overline{C}D$ variables has a domain made up of the A ,B ,C , and D . However ,

Note1: notice that the complete set of variables in the domain is not represented in the first two terms of the expression ; that is , $D\ \ OR\ \ \overline{D}$ is missing from the second term.

A ***Standard SOP expression*** is one in which all the variables in the domain appear in each product term in the expression.

For example , $\overline{A}B\overline{C}D+\overline{A}\overline{B}CD+AB\overline{C}\,\overline{D}$ is a standard SOP expression . Standard SOP expressions are important in constructing truth tables, and in the **Karnaugh map** simplification method , which is covered in second section .

Note2: Any nonstandard SOP expression (referred to simply as SOP) can converted to the standard from using Boolean algebra.

- Converting Product Terms to Standard SOP

Each product term in an SOP expression that does not contain all the variables in the domain can be expanded to standard form to include all variables in the domain and their complements. As stated in the following steps, a nonstandard SOP expression is converted into standard from using Boolean algebra **Rule 6 (** $A+\overline{A}=1$ **) : a variable added to its complement equals 1 .**

Step1: Multiply each nonstandard product term by a term made up of the sum of a missing variable and its complement .This results in two product terms. As you know , you can multiply anything by 1 without changing its value.

Step2: Repeat step1 until all resulting product terms contain all variables in the domain in either complemented or uncomplemented form. In converting a product term to standard form, the number of product terms is doubled for each missing variable as shows in following example:

Example: Convert the following expression into standard SOP form:

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

Solution: *the domain of this SOP expression is A ,B,C,D . take one term at a time.*

The first term $A\overline{B}C$ *, is missing variable D or* $\overline{D}$ *,so multiply the first term by* $D+\overline{D}$ as follows :

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

In this case, two standard product terms are the result .

The second term $\bar{A}\bar{B}$ *, is missing variables C or* $\bar{C}$ *and D or* $\bar{D}$ *, so first multiply the second term by* $C+\bar{C}$ *as follows:*

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

The two resulting terms are missing variable D or $\bar{D}$ *, so multiply both terms by* $D+\bar{D}$ *as follows :*

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

In this case , four standard product term are result .

The third term, $\bar{A}B\overline{CD}$ is already in standard form. The complete standard SOP form of the original expression is as follows:

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

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

Home work

Convert the expression $W\bar{X}Y + \bar{X}Y\bar{Z} + WX\bar{Y}$ **to Standard SOP Form**

- Binary Representation of a Standard Product Term

A standard product term is equal to 1 for only on combination of variable values. For example , the product term $A\bar{B}C\bar{D}$ is equal to 1 when A=1 ,B=0 ,C=1 , D=0 , as shown below , and is 0 for all other complements of values for the variables.

$$A\bar{B}C\bar{D} = 1.\bar{0}.1.\bar{0} = 1.1.1.1 = 1$$

In this case , the product term has a binary of 1010 (decimal ten)

. **Remember** , a product term is implement with an AND gate whose output is 1 only if each of its inputs is 1. inverters are used to produce the complements of the variables as required.

Notes :An SOP expression is equal to 1 only if one or more of the product terms in the expression is equal to 1

Example : determine the binary value for which the following Standard SOP expression is equal 1:

$$ABCD + A\overline{B}\,\overline{C}D + \overline{A}\,\overline{B}\,\overline{C}\,\overline{D}$$

Solution :

The first term *ABCD is equal to 1 when A=1 , B=1 ,C=1 and D=1 , so*

$$ABCD = 1\ .\ 1\ .\ 1\ .\ 1 = 1\ .\ 1\ .\ 1\ .1 = 1$$

The second term $A\overline{B}\,\overline{C}D$ *is equal to 1 when A=1 , B=0, C=0, and D=1 , so*

$$A\overline{B}\,\overline{C}D = 1\ .\ \overline{0}\ .\ \overline{0}\ .\ 1 = 1\ .\ 1\ .\ 1\ .\ 1 = 1$$

The third term $\overline{A}\,\overline{B}\,\overline{C}\,\overline{D}$ is equal to 1 when A=0 , B=0 , c=0 , and D=0 .

$$\overline{A}\,\overline{B}\,\overline{C}\,\overline{D} = \overline{0}\ .\ \overline{0}\ .\ \overline{0}\ .\ \overline{0} = 1\ .\ 1\ .\ 1\ .\ 1 = 1$$

The Sop expression equals 1 when any or all of the three product terms is 1.

Home work

Determine the binary values for which the following SOP expression is equal to 1: $\overline{X}YZ + X\overline{Y}Z + XY\overline{Z} + \overline{X}Y\overline{Z} + XYZ$ **is this a Standard SOP expression?**

The Products –Of – Sum (POS) Form

When two or more sum terms are multiplied , the resulting expression is a Product –Of – Sum **(POS) ,** some examples are

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

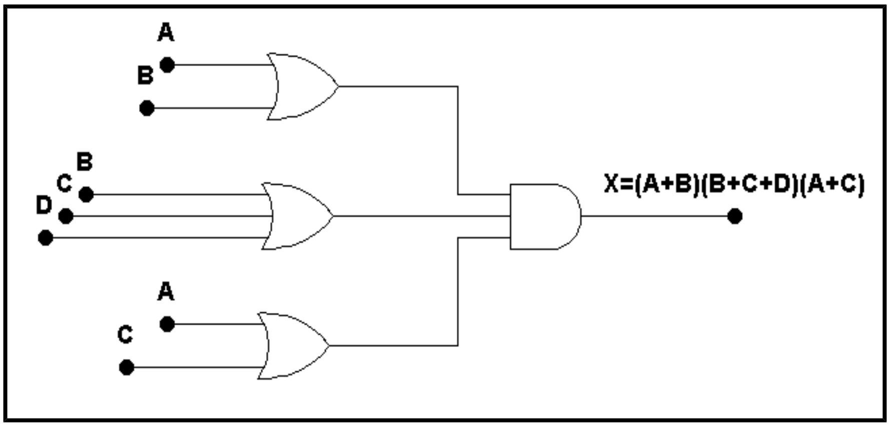
Notes:

1. A POS expression can contain a single–variable term ,as
$\overline{A}(A+\overline{B}+C)(\overline{B}+\overline{C}+D)$.

2. In a POS expression , a single overbar cannot extend over more than one variable; for example , a POS expression can have the term
$\overline{A}+\overline{B}+\overline{C}$ *but not* $\overline{A+B+C}$.

- Implementation of a POS expression

Implementing a POS expression simply require ANDing the outputs of two or more OR gates. The next figure shows the expression (A+B)(B+C+D)(A+C). the output X of the AND gate equals the POS expression.



- The Standard POS Form

So far , you have seen POS expression in which some of the sum terms do not contain all of the variables in the domain of the expression. For example , the expression , $(A+\overline{B}+C)(A+B+\overline{D})(A+\overline{B}+\overline{C}+D)$ has the domain made up of the variables ,A ,B ,C and D .

Notes that the complete set of variables in the domain is not represented in the first two terms of the expression ; that is , D or $\overline{D}$ is missing from the first term and C or $\overline{C}$ is missing from the second term.

A standard POS expression is one in which all the variables in the domain appear in each sum term in the expression , for example ,

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

Is a standard POS expression , Any nonstandard POS expression (referred to simply as POS) can be converted to the standard form using Boolean algebra .

- Converting a Sum Term to Standard POS

Each sum term in a POS expression that does not contain all the variables in the domain can be expanded to standard form to include all variables in the domain and their complements. As stated in the following steps, a nonstandard POS expression is converted into standard form using Boolean algebra **Rule8** $(A\,.\,\overline{A}=0)$ **:A variable multiplied by its complement equal 0.**

Step1: Add to each nonstandard product term a term made up of the product of ,the missing variable and its complement. This results in two sum terms .As you know ,you can add 0 to anything without changing its value.

Step2: Apply **rule12 : A +BC =(A+B)(A+C)**

Step3: Repeat Step 1 until al resulting sum terms contain all variables in the domain in either complement or un complemented form.

***Example* : Convert the following Boolean expression into Standard POS form.** $(A+\overline{B}+C)(\overline{B}+C+\overline{D})(A+\overline{B}+\overline{C}+D)$

Solution: *the domain of this POS expression is A,B,C,D. take one term at a time,*

The first term $A+\overline{B}+C$ *D or* $\overline{D}$ *, so add* $D\overline{D}$ *and apply* ***rule12*** *as follows:*

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

The Second term $\overline{B}+C+\overline{D}$ *is missing variable A or* $\overline{A}$ *,so add* $A\overline{A}$ *and apply rule 12 as follows:*

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

The third term , $A+\overline{B}+\overline{C}+D$ *, is already in standard form , so, the standard POS form of the original expression is as follows:*

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

Home work

Convert the expression $(A+\overline{B})(B+C)$ **to Standard POS Form**

- Binary Representation of a Standard Sum Term

A standard sum term is equal to 0 for only one combination of variable values. For example ,the sum term $A+\overline{B}+C+\overline{D}$ is 0 when A=0 , B=1 , C=0 , and D=1 , as shown below , and 1 for all for all other combinations of values for the variables ,

$$A+\overline{B}+C+\overline{D}=0+\overline{1}+0+\overline{1}=0+0+0+0=0$$

In this case ,the sum term has a binary value of 0101 (decimal 5) .

Remember , a sum term is implemented with an OR gate whose output is 0 only if each its inputs is 0. Inverters are used to produce the complements of the variables as required .

A POS expression is equal to 0 only if one or more of the sum terms in the expression is equal to 0.

Example: Determine the binary values of the variables for which the following standard POS expression is equal to 0:

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

Solution:

The first term $(A+B+C+D)$ *is equal to 0 when A=0 , B=0 , C=0 and D=0.*

$$A+B+C+D=0+0+0+0=0$$

The second term $A+\overline{B}+\overline{C}+D$ *is equal to 0 when A=0 , B=1, C=1 and D=1*

$$A+\overline{B}+\overline{C}+D=0+\overline{1}+\overline{1}+0=0+0+0+0=0$$

The third term $\overline{A}+\overline{B}+\overline{C}+\overline{D}$ *is equal to 0 when A=1 ,B=1 , C=1 and D=1*

$$\overline{A}+\overline{B}+\overline{C}+\overline{D}=\overline{1}+\overline{1}+\overline{1}+\overline{1}=0+0+0+0=0$$

The POS expression equals 0 when any of the three terms eaual 0.

Home work

Determine the binary values for which the following POS expression is equal to 0 : $(X+\overline{Y}+Z)(\overline{X}+Y+Z)(X+Y+\overline{Z})(\overline{X}+\overline{Y}+\overline{Z})(X+\overline{Y}+\overline{Z})$ **is this a Standard POS expression?**

Convert SOP to POS form

The binary values of the product term in a given SOP expression are not present in the equivalent standard POS expression .Also , the binary values that are not represented in the SOP expression are present in the equivalent POS expression. Therefore ,to convert from standard SOP to standard POS, the following steps are taken:

Step1: Evaluate each product term in the SOP expression . That is , determine the binary numbers that represent the product term.

Step2: Determine all of the binary numbers not included in the evaluation in step1.

Step3: Write the equivalent sum term for each binary number from step2 and express in POS form.

Note: using a similar procedure , you can go from POS to SOP .

Example : Convert the following SOP expression to an equivalent POS expression:

$$\overline{A}\,\overline{B}\,\overline{C} + \overline{A}\,B\overline{C} + \overline{A}\,BC + A\overline{B}\,C + ABC$$

Solution: *the evaluation is as follows:* **000+010+011+101+111**

Since there are three variables in the domain of this expression, there afe a total of eight(2^3) possible combinations.

The SOP expression contains five of these combination , so the POS must contain the other three which are 001 ,100 and 110.

Remember , these are the binary values that take the sum term 0. the equivalent POS expression is $(A+B+\overline{C})(\overline{A}+B+C)(\overline{A}+\overline{B}+C)$

Boolean Expression And Truth Tables

- Converting SOP expressions to Troth Table Format

Recall from pervious sections , that an SOP expression is equal to 1 only if at least one of the product terms is equal to 1.

- The first step un constructing a truth table is to list all possible combinations of binary values of the variables in the expression.
- Next ,convert the SOP expression to standard form if it is not already.
- Finally ,place a 1 in the output column (X) for each binary value that takes the standard SOP expression a 1 and place 0 for all remaining binary values.

This procedure illustrated in the example bellow :

Example :Develop a truth table for the Standard SOP expression

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

Solution

There are three variables in the domain, so there are **eight** possible combinations of binary values of the variables as listed in the left three columns of tables bellow .

The binary values that make the product terms in the expressions equal to 1 are $\overline{A}\,\overline{B}C:001\,;A\overline{B}\,\overline{C}:100\,;\,and\;\;ABC:111$. For each of these binary values , place a 1 in the output column as shown in the table. For each of the remaining binary combinations , place a 0 in the output column.

Input			Outpu t	Product term
A	B	C	X	
0	0	0	0	
0	0	1	1	$\overline{A}\,\overline{B}C$
0	1	0	0	
0	1	1	0	
1	0	0	1	$A\overline{B}\,\overline{C}$
1	0	1	0	
1	1	0	0	
1	1	1	1	ABC

- Converting SOP expressions to Troth Table Format

Recall from previous sections, that a POS expression is equal to 0 only if at least one of the sum terms is equal to 0.

- To construct a truth table from a POS expression ,list all the possible combinations of binary values of the variable just as done for SOP expression.
- Next, convert the POS expression to standard form if it is not already.
- Finally, place a 0 in the output column (X) for each binary value that makes the expression a 0 and place a 1 for all the remaining binary values.

This procedure is illustrated in example bellow:

Example: determine the troth table for the following standard POS expression:

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

Solution

There are three variables in the domain, so there are **eight** possible combinations of binary values of the variables as listed in the left three columns of tables bellow .

The binary values that make the product terms in the expressions equal to 0 are

$$A+B+C:000 \;\; ; \;\; A+\overline{B}+C:010 \;\; ; \;\; A+\overline{B}+\overline{C}:011 \; ;$$
$$\overline{A}+B+\overline{C}:101 ; \;\; and \;\; \overline{A}+\overline{B}+C:100$$

For each of these binary values , place a 0 in the output column as shown in the table. For each of the remaining binary combinations , place a 1 in the output column.

Input			Output	Sum term
A	B	C	X	
0	0	0	0	$A+B+C$
0	0	1	1	
0	1	0	0	$A+\overline{B}+C$
0	1	1	0	$A+\overline{B}+\overline{C}$
1	0	0	1	
1	0	1	0	$\overline{A}+B+\overline{C}$
1	1	0	0	$\overline{A}+\overline{B}+C$
1	1	1	1	

Home work

1- **Create a truth table for the standard SOP expression :** $\overline{A}B\overline{C}+A\overline{BC}$

2- **Create a troth table for the standard POS expression :**

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

- Determining Standard Expressions from Troth Table

To determine the standard SOP represented by a truth table.

- List the binary value of the input variables for which the outputs is 1.
- Convert each binary value to the corresponding product term by replacing each 1 with the corresponding variable and each 0 with the corresponding variable complement.

For example, the binary value **1010** is converted t0 a product term as follows:

$$1010 \rightarrow A\bar{B}C\bar{D}$$

If you substitute , you can see that the product term is 1:

$$A\bar{B}C\bar{D}=1.\bar{0}.1.\bar{0}=1.1.1.1=1$$

To determine the standard POS represented by a truth table.

- List the binary value of the input variables for which the outputs is 0.
- Convert each binary value to the corresponding sum term by replacing each 1 with the corresponding variable complement and each 0 with the corresponding variable .

For example, the binary value **1010** is converted to a sum term as follows:

$$1001 \rightarrow \bar{A}+B+C+\bar{D}$$

If you substitute , you can see that the sum term is 0:

$$\bar{A}+B+C+\bar{D}=\bar{1}+0+0+\bar{1}=0+0+0+0=0$$

ـH The KARNUPH Map

- Introduction

A karnauph map provides a systematic method for simplifying Boolean expressions and, if properly used ,will produce the simplest SOP or POS expression possible, known as ***the minimum expression***. As you have seen , the effectiveness of the algebraic simplification depends on your familiarity with the laws, rules , and theorems of Boolean algebra and the on your ability to apply them. The Karnauph map on other hand , provide a "cookbook" method for simplification . other simplification techniques include the ***Quine-McClusky*** method and the **Espresso algorithm**

-A Karnauph map

Is similar to a truth table because it present all of the possible value of input variables and the resulting output for each value.

- Instead of being organized into columns and rows like a troth table, the Karnauph map is an array of **cells** in which each cell represents a binary value of the input variables.
- The cells are arranged in a way so that simplification of a given expression is simply a matter of properly grouping the cells.
- Karnauph maps can be used for expressions with two , three , for , and five variables, but we will discuss only 2-variable , 3-variables and 4-variables situations to illustrate the principles.
- The number of cells in Karnauph map, as well as the number of rows in a troth table, is equal to the total number of possible input variable combinations.
- For two variable , the number of cells is 2^2=4 cells , for three variable 2^3=8 cells , and four variables 2^4=16 cells .

-A 2- Variable Karnauph map

The 2-variable karnauph map is an array of four cells, as shown in table(1) bellow .

- In this case A and B are used for the variables although other letters could be used.
- Binary values of A are along the left side and the values of B are across the top.
- The value of a given cell is the binary values of A at the left in the same row combined with the value of B at the to in the same column.

For example, the cell in the upper left corner has a binary value 00 and the cell in the lower right corner has a binary value 11 . Table(2) shows the standard product terms that are represented by each cell in the 2-varable Karnaugh map.

Table(1)

B

A	**0**	**1**
0	00	01
1	10	11

Table(2)

B

A	**0**	**1**
0	$\bar{A}\bar{B}$	$\bar{A}B$
1	$A\bar{B}$	AB

-A 3- Variable Karnauph map

The 3-variable karnauph map is an array of eight cells, as shown in table(1) bellow .

- In this case A ,B and C are used for the variables although other letters could be used.
- Binary values of A and B are along the left side and the values of C are across the top.
- The value of a given cell is the binary values of A and B at the left in the same row combined with the value of C at the to in the same column.

For example, the cell in the upper left corner has a binary value 000 and the cell in the lower right corner has a binary value 101 . Table(2) shows the standard product terms that are represented by each cell in the 3-varable Karnaugh map.

Table(1)

C

AB	**0**	**1**
00	000	001
01	010	011
11	110	111
10	100	101

Table(2)

C

AB	**0**	**1**
00	$\bar{A}\bar{B}\bar{C}$	$\bar{A}\bar{B}C$
01	$\bar{A}B\bar{C}$	$\bar{A}BC$
11	$AB\bar{C}$	ABC
10	$A\bar{B}\bar{C}$	$A\bar{B}C$

- A 4- Variable Karnauph map

The 4-variable karnauph map is an array of eight cells, as shown in table(1) bellow .

- In this case A ,B C ,and D are used for the variables although other letters could be used.
- Binary values of A and B and C and D are along the left side and the values of C are across the top.
- The value of a given cell is the binary values of A and B at the left in the same row combined with the value of C and D at the to in the same column.

For example, the cell in the upper right corner has a binary value 0010 and the cell in the lower right corner has a binary value 1010 . Table(2) shows the standard product terms that are represented by each cell in the 4-varablr Karnaugh map.

Table(1)

CD AB	00	01	11	10
00	0000	0001	0011	0010
01	0100	0101	0111	0110
11	1100	1101	1111	1110
10	1000	1001	1011	1010

Table(2)

CD AB	00	01	11	10
00	$\overline{A}\overline{B}\overline{C}\overline{D}$	$\overline{A}\overline{B}\overline{C}D$	$\overline{A}\overline{B}CD$	$\overline{A}\overline{B}C\overline{D}$
01	$\overline{A}B\overline{C}\overline{D}$	$\overline{A}B\overline{C}D$	$\overline{A}BCD$	$\overline{A}BC\overline{D}$
11	$AB\overline{C}\overline{D}$	$AB\overline{C}D$	$ABCD$	$ABC\overline{D}$
10	$C\overline{B}\overline{C}\overline{D}$	$C\overline{B}\overline{C}D$	$A\overline{B}CD$	$A\overline{B}C\overline{D}$

- Cell Adjacency

The cells in a Karnauph map are arranged so that there is only a single-variable between adjacent cells. **Adjacency is defined by a single-variable change.**

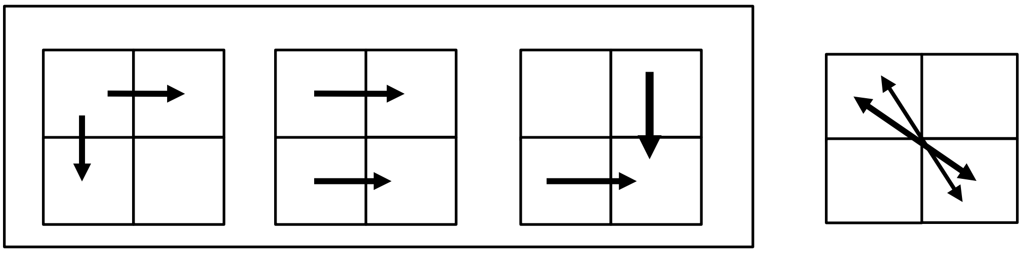
In The 2-Variable map , the 00 cell is adjacent to the 10 cell, and the 10 cell but not adjacent to 11 cell.

In the 3-Varibale map, the 010 cell is adjacent to 000 cell, and the 110 cell. the 010 cell is not adjacent to the 001 cell, the 100, or the 101 cell.

Note1: Physically ,each cell is adjacent to the cells that are immediately next to it on any of its four sides.

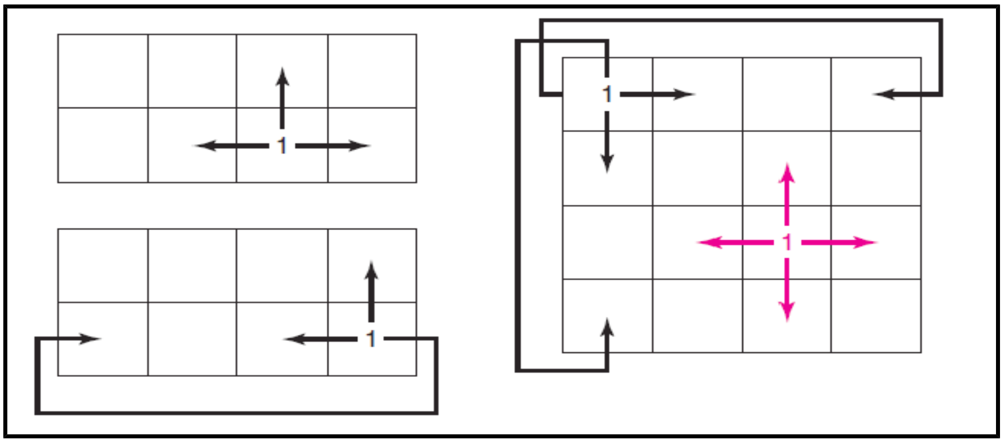
Note2: A cell is not adjacent to the cells that diagonally touch any of its corners.

Notes3: Also, the cells in the top row are adjacent to the corresponding cells in the bottom row and the cells in the outer left column are adjacent to the corresponding cells in the outer right column. ***This is called "wrap-around" adjacency because you can think of the map as wrapping around from top to bottom to form a cylinder or from left to right to form a cylinder.***



Correct adjacency **wrong adjacency**

2-variable adjacency



3 and 4 –variable adjacency

We will explain the way to use karnauph map to minimize SOP standard expression only .

KARNAUGH Map SOP Minimization

As stated in the last section, the Karnaugh map is used for simplifying Boolean expressions to their minimum form. A minimized SOP expression contains the fewest possible terms with the fewest possible variables per term.

Generally, a minimum SOP expression can be implemented with fewer logic gates than a standard expression .

- Mapping a standard SOP Expression

- For an SOP expression in standard form, a 1 is placed on the Karnaugh map for each product term in the expression.

- Each 1 is placed in a cell corresponding to the value of a product term.

  For example, for the product term $A\bar{B}C$,a 1 goes in the 101 cell on a 3-variablemap.

- When an SOP expression is completely mapped, there will be a number of 1's on the Karnaugh map equal to the number of product terms in the standard SOP expression.

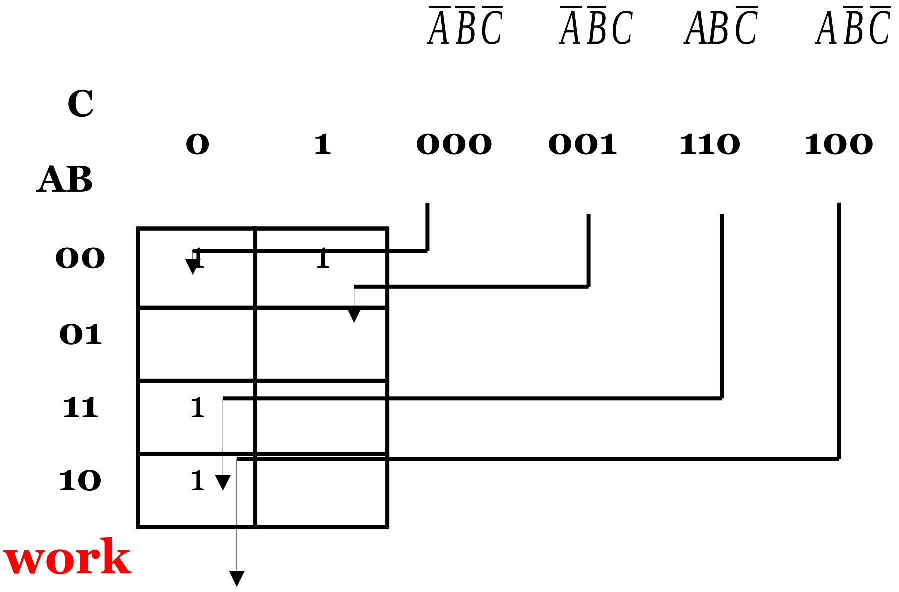
- The cells that do not have a 1 are the cells for which the expression is 0.

- Usually ,when working with SOP expressions, the 0's are left off the map.

The following steps and the illustration in the next figure show the mapping process .

Step1: Determine the binary value of each product term in the standard SOP expression. After some practice ,you can usually do the evaluation of terms mentally.

Step2: As each product term is evaluated, place a 1 on the Karnaugh map in the cell having the same value as the product term.



Home work

Map the following standard SOP expression on a Karnauph map

$$1-\ \bar{A}\bar{B}C+\bar{A}B\bar{C}+AB\bar{C}+ABC$$
$$2-\ \bar{A}BC+A\bar{B}C+A\bar{B}\bar{C}$$
$$3-\bar{A}\bar{B}CD+\bar{A}B\bar{C}\bar{D}+AB\bar{C}D+ABCD+AB\bar{C}\bar{D}+\bar{A}\bar{B}\bar{C}D+A\bar{B}C\bar{D}$$

- Karnaugh Map Simplification of SOP expressions

The process that results in an expression containing the fewest possible terms with the fewest possible variables is called **minimization**. After an SOP expression has been mapped , a minimum SOP expression is obtained by ***grouping the 1's*** and ***determining the minimum SOP expression from the map***.

- ***grouping the 1's*** : you can group 1s on the Karnaugh map according to the following rules by enclosing those adjacent cells containing 1s.the goal is to maximize the size of the groups and to minimize the number of groups.

  1. A group must contain either 1,2,4,8 , or 16 cells,which are all powers of two. In the case of a 3-variables map,2^3=8 cells is the maximum group.
  2. Each cell in a group must be adjacent to one or more cells in that some group, but all cells in the group do not have to be adjacent to each other.
  3. always include the largest possible number of 1s in a group in accordance with rule 1.
  4. Each1 on the map must be included in at least one group. The 1s already in a group can be included in another group as long as the overlapping groups include non common 1s.

Example :group the 1s in each of the Karnauph maps in the following 2-varable map.

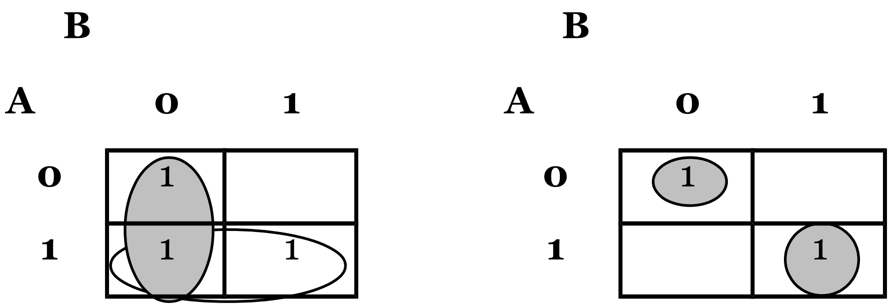
2-variable map

	B	
A	**0**	**1**
0	1	
1	1	1

	B	
A	**0**	**1**
0	1	
1		1

Solution: the grouping are shown the next figure. In same cases, there may be more than one way to group the1s to form maximum grouping .

2-variable map



B

A	0	1
0	1	
1	1	1

B

A	0	1
0	1	
1		1

Example :group the 1s in each of the Karnauph maps in the following 3-varable map.

3- variable map

C

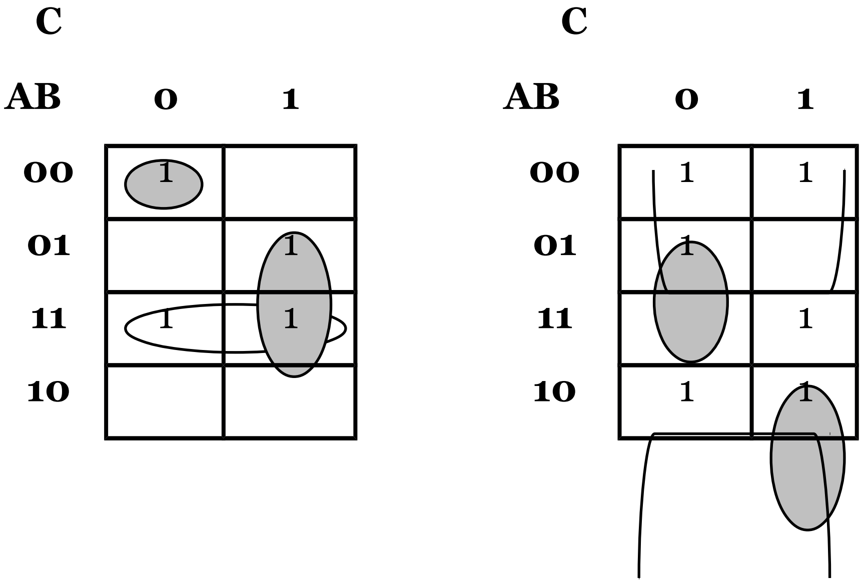
AB	0	1
00	1	
01		1
11	1	1
10		

C

AB	0	1
00	1	1
01	1	
11		1
10	1	1

Solution: the grouping are shown the next figure. In same cases, there may be more than one way to group the1s to form maximum grouping .

3- variable map



C

AB	0	1
00	1	
01		1
11	1	1
10		

C

AB	0	1
00	1	1
01	1	
11		1
10	1	1

Example :group the 1s in each of the Karnauph maps in the following 4-varable map

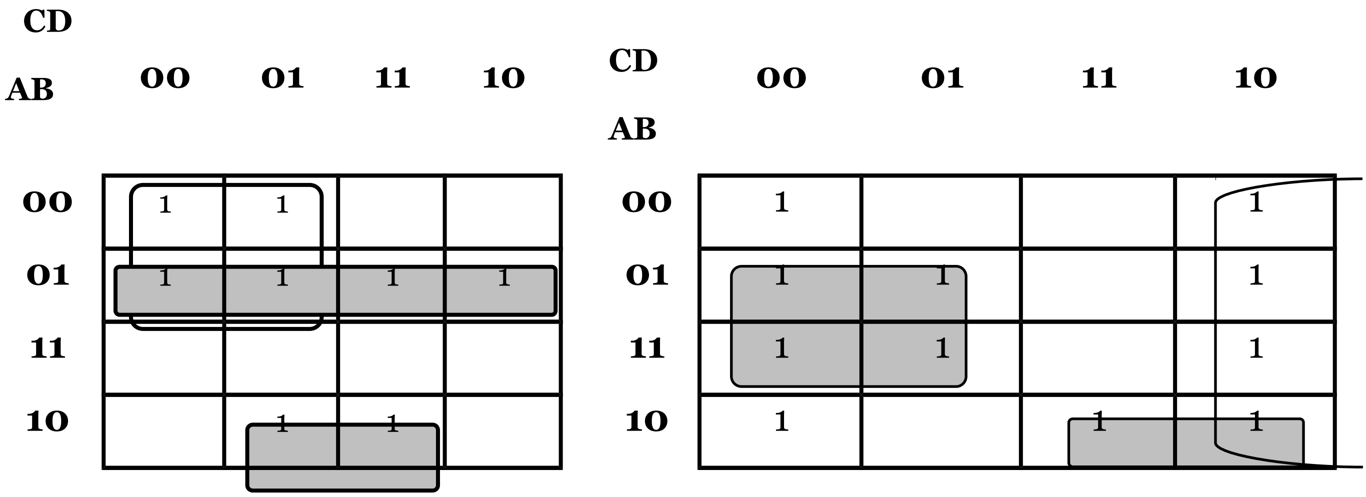
4-variable map

CD AB	00	01	11	10
00	1	1		
01	1	1	1	1
11				
10		1	1	

CD AB	00	01	11	10
00	1			1
01	1	1		1
11	1	1		1
10	1		1	1

Solution: the grouping are shown the next figure. In same cases, there may be more than one way to group the1s to form maximum grouping .

4-variable map



- *Determining the minimum SOP Expression From the map*

When all the 1s representing the standard product terms in an expression are properly mapped and grouped, the process of determining thee resulting minimum SOP expression begins.

The following rules ate applied to find the minimum product terms and the minimum SOP expression:

1. Group the cell that have 1's. Each group of cells containing 1's creates one product item composed of all variables that occur in only one form (either uncomplemented or complemented) within the group. ***Variable that occur both uncomplemented and complemented within the group are eliminated . these are called contradictory variables***

2. Determine the minimum product term for each group.

a. For a 2-Variable map

(1) A 1-cell group yields a 2-variable product term .

(2) A 2-cell group yields a 1-variable product term.

(3) An 4-cell group yields a value of 1 for the expression.

b. For a 3-Variable map

(1) A 1-cell group yields a 3-variable product term .

(2) A 2-cell group yields a 2-variable product term.

(3) A 4-cell group yields a 1-variable product term.

(4) An 8-cell group yields a value of **1** for the expression

c. For a 4-Variable map

(1) A 1-cell group yields a 4-variable product term**.**

(2) A 2-cell group yields a 3-variable product term.

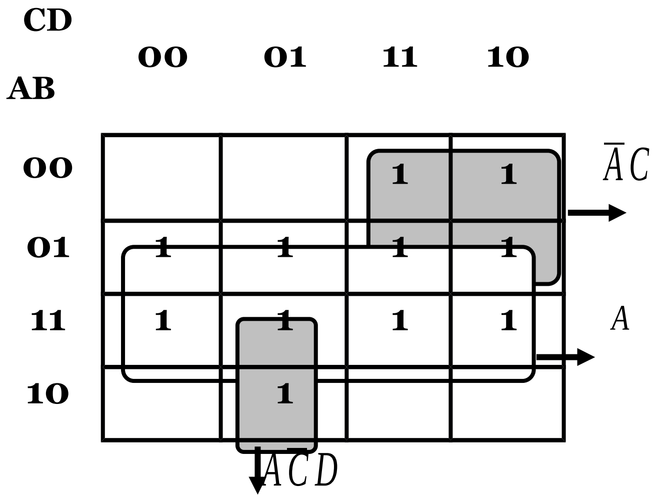
(3) A 4-cell group yields a 2-variable product term.

(4) An 8-cell group yields a 1-variable term.

(5) A 16-cell group yields a value of 1 for the expression .

3. When all the minimum product terms are derived from the Karnaugh map, they are summed to form the minimum SOP expression,

Example : Determine the product terms for the Karnaugh map in figure bellow , and write the resulting minimum SOP expression.



Solution: Eliminate variables that are in a grouping in both complemented and uncomplemented forms. In the above figure ,

- the product term for the 8-cell group is B because the cells withen that group contain both A and $\bar{A}$, C and $\bar{C}$, and D and $\bar{D}$, which are eliminated.
- The 4-cell group contains $B ,\bar{B} ,D,$ *and* $\bar{D}$, leaving the variables $\bar{A}$ and C. which form the product term $\bar{A}C$.
- The 2-cell group contains B and $\bar{B}$, leaving variables A , $\bar{C}$, and D which form the product term $A\bar{C}D$.

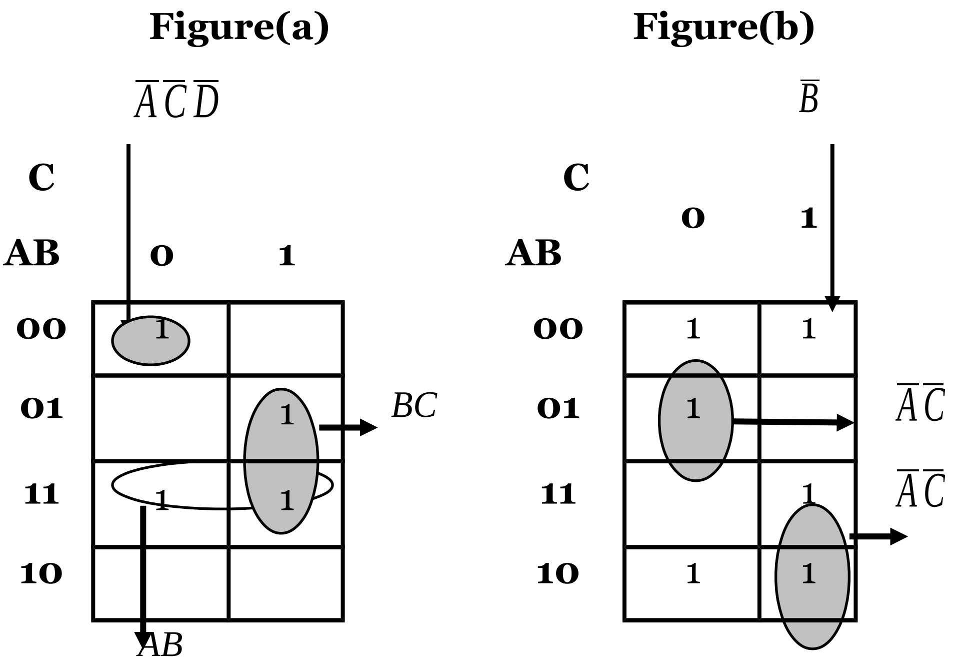
Notes: how overlapping is used to maximize the size of the groups. The resulting minimum SOP expression is the sum of these product terms:

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

Home work

for the Karnagh map in the above example , add a 1 in the lower right cell (1010) and determine the resulting SOP expression.

Example : Determine the product terms for the Karnaugh map in two figures bellow , and write the resulting minimum SOP expression.



Figure(a)

Figure(b)

$\overline{A}\overline{C}\overline{D}$
C
AB
0
1
00
01
11
10
1
1
1
1
BC
AB
$\overline{B}$
C
AB
0
1
00
01
11
10
1
1
1
1
1
1
$\overline{A}\overline{C}$
$\overline{A}\overline{C}$

Solution: *the resulting minimum product term for each group shown in figure(a) and (b) are:*

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

(b) $\overline{B} + \overline{A}\,\overline{C} + AC$

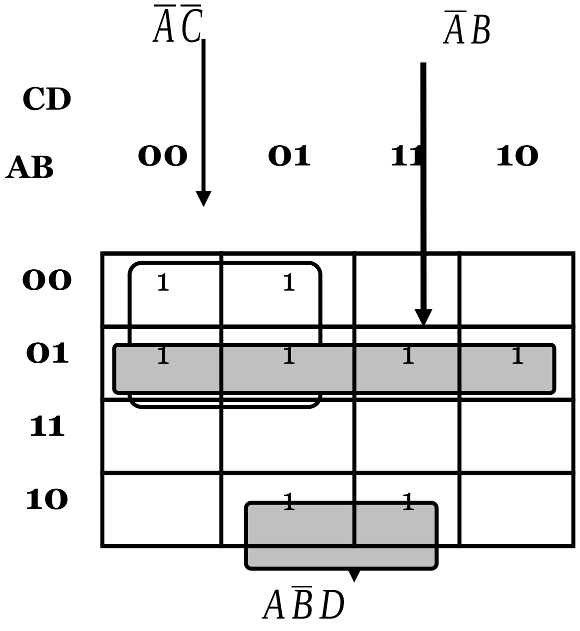
Example : Determine the product terms for the Karnaugh map in two figures bellow , and write the resulting minimum SOP expression.

Solution: *the resulting minimum product term for each group shown in figure(c) and (d) are:*

(c) $\bar{A}B + \bar{A}\bar{C} + A\bar{B}D$

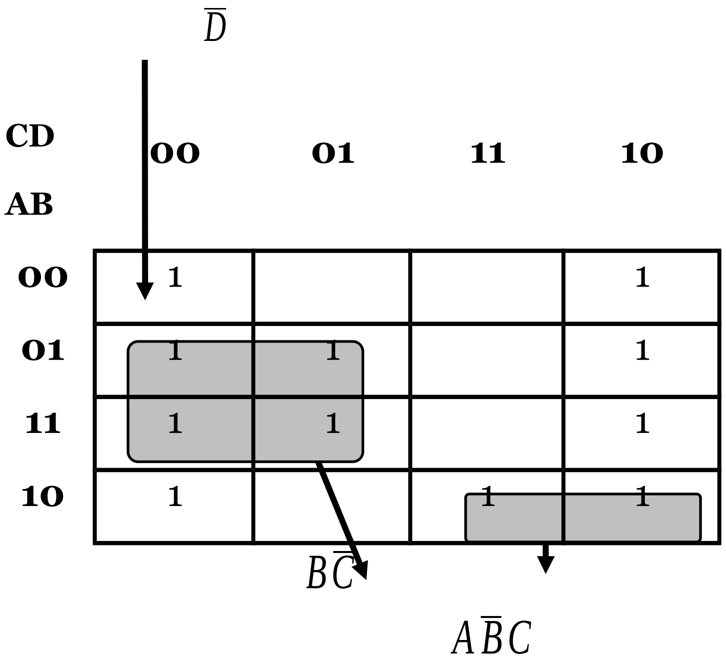
(d) $\bar{D} + A\bar{B}C + B\bar{C}$

Figure (c)



AB \ CD	00	01	11	10
00	1	1		
01	1	1	1	1
11				
10		1	1	

Figure (d)



AB \ CD	00	01	11	10
00	1			1
01	1	1		1
11	1	1		1
10	1		1	1

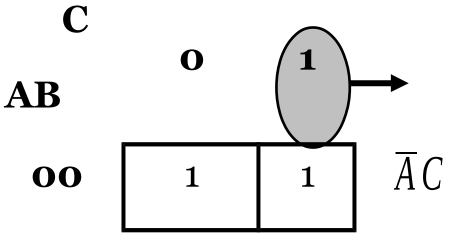
Example: *Use a Karnaugh map to minimize the following standard SOP expression:*

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

Solution: the binary values of the expression are

101 + 011 + 001 + 000 + 100

Map the standard SOP expression and group the cells are shown in the following figure.



AB \ C	0	1
00	1	1

01		1	
11			
10	1	1	$\overline{B}$

The resulting minimum SOP expression is : $\overline{B}+\overline{A}C$

Home work

Use the Karnaugh map to minimize the following SOP expression:

$$1-\overline{B}\,\overline{C}\,\overline{D}+\overline{A}B\overline{C}\,\overline{D}+AB\overline{C}\,\overline{D}+\overline{A}\,\overline{B}CD+A\overline{B}CD+\overline{A}\,\overline{B}C\overline{D}+\overline{A}BC\overline{D}+ABC\overline{D}+A\overline{B}C\overline{D}$$

$$2-\overline{W}\,\overline{X}\,\overline{Y}\,\overline{Z}+W\overline{X}YZ+W\overline{X}\,\overline{Y}Z+\overline{W}YZ+W\overline{X}\,\overline{Y}\,\overline{Z}$$

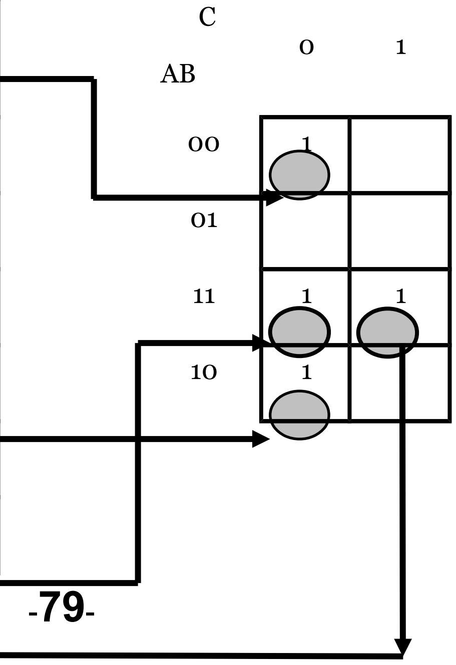
Mapping Directly from the Truth Table

You have seen how to map a Boolean expression; now you will learn how to go directly from a truth table to a Karnauph map. Recall that a truth table gives the output of a Boolean expression for all possible input variable combination.

An example of a Boolean expression and its troth table representation is shown in the next figure. Notice in the truth table that the output X is 1 for four different input variable combinations.

$$X=\overline{A}\,\overline{B}\,\overline{C}+A\overline{B}\,\overline{C}+AB\overline{C}+ABC$$

Input			Output
A	**B**	**C**	**X**
0	0	0	1
0	0	1	0
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	1



AB \ C	0	1
00	1	
01		
11	1	1
10	1	

1	1	1	1

The 1s in the output column of the truth table are mapped directly onto a Karnaugh map into the cells corresponding to the values of the associated input variable combinations.

Notes :*In the above figure you can see that the Boolean expression , the truth table , and the Karnaugh map are simply different ways to represent a logic function.*

"Don’t care " Condition

Sometime a situation arises in which some input variables combinations are not allowed. For example , recall tat in BCD code covered in previous sections , there are six invalid combinations: 1010 ,1011 ,1100,1101 , and 1111.

Since these un allowed states will never occur in an application involving the BCD code, they can be treated as **"Don’t care"** terms with respect to their effect on the output. That is , for these **"don’t care "** terms either a **1** or a **0** may be assigned to the output; it really does not matter since they will never occur.

The **"don’t care"** terms can be used to advantage on the Karnaugh map. The figure bellow shows that for each "don’t care" term, an X is placed in the cell. When grouping the 1s , the Xs can be treated as 1s to make a larger grouping or as 0s if they cannot be used to advantage. ***The larger a group , the simpler the resulting term will be.***

Input				**Output**
A	**B**	**C**	**D**	**X**
0	0	0	0	0
0	0	0	1	0
0	0	1	0	0
0	0	1	1	0
0	1	0	0	0
0	1	0	1	0
0	1	1	0	0
0	1	1	1	1
1	0	0	0	1
1	0	0	1	1

1	0	1	0	X
1	0	1	1	X
1	1	0	0	X
1	1	0	1	X
1	1	1	0	X
1	1	1	1	X

Don't

(a) Truth Table

AB \ CD	**00**	**01**	**11**	**10**	
00					
01			**1**		$\overline{A}BCD$
11	**X**	**X**	**X**	**X**	BCD
10	**1**	**1**	**X**	**X**	

$A\overline{B}\overline{D}$ A

(b)without "don't care" Y= $A\overline{B}\overline{C}+\overline{A}BCD$ **,with "don't care"** Y= $A+BCD$

Notes: the truth table (a) describe a logic function that has a 1 output only when the BCD code for 7 ,8 , or 9 is present on the inputs. If the **"don't care"** are used as 1s the resulting expression for the function is $A+BCD$, a indicated in part(b). if the "don't cares" are not used as 1s ,the resulting expression is $A\overline{B}\overline{C}+\overline{A}BCD$; so you can see the advantage of using **"don't care"** terms to get the simplest expression.

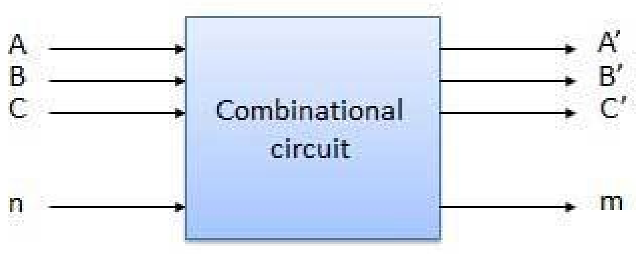
6- Combinational Logic Analysis

Combinational circuit is circuit in which we combine the different gates(AND ,OR ,NAND ,NOR ,XOR ,XNOR ,NOT ,..) in the circuit for example encoder, decoder, multiplexer , demultiplexer and any circuit build bay any combinations of theses gates . 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 n number of inputs and m number of outputs.

- Combinational circuit black Diagram



- Implementing Combinational Logic

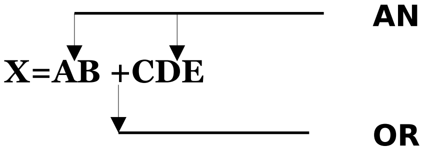
In this section , examples are used to illustrate how to implement a logic circuit from a Boolean expression or a truth table. Minimization of a logic circuit using the methods covered in previous section.

- **From a Boolean Expression to a logic circuit**

Let's examine the following Boolean expression:

X=AB +CDE

A brief inspection shows that this expression is composed of two terms , AB and CDE , with a domain of five variables. The first term is formed by ANDing A with B, and the second term is formed by ANDing C,D, and D. the two terms are then ORed to form the output X . These operations are indicated in the structure of the expression as follows:

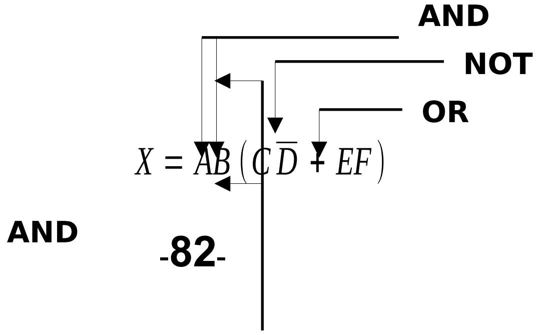


Homework: _Draw the logic circuit for the above Boolean expression_

Example: let's implement the following expression:

$$X = AB\left(C\overline{D} + EF\right)$$

Solution:



Notes: Unless an intermediate term, such as $C\overline{D}+EF$, is required as an output for some other purpose, it usually best to reduce a circuit to its SOP form in order to reduce the overall propagation delay time .the expression is converted to SOP as follows, $AB\left(C\overline{D}+EF\right)=ABC\overline{D}+ABEF$.

Homework:* *Draw the logic circuit for the above SOP Boolean expression

- ## From a Truth Table to a Logic Circuit

If you begin with a truth table instead of an expression, you can write the SOP expression from the truth table and then implement the logic circuit. The table bellow specifies a logic function.

Input			Output	Product term
A	B	C	X	
0	0	0	0	
0	0	1	0	
0	1	0	0	
0	1	1	1	$\overline{A}BC$
1	0	0	1	$A\overline{B}\overline{C}$
1	0	1	0	
1	1	0	0	
1	1	1	0	

The Boolean SOP expression obtained from the truth table by ORing the product terms for which **X=1** is X= $\overline{A}BC + A\overline{B}\overline{C}$

Homework:

1- ***Draw the logic circuit for the above SOP Boolean expression .***

2- ***Develop a logic circuit with four input variable that will only produce a 1 output when exactly three input variables are 1s.***

Function Of Combinational Logic

As we mention above , Combinational circuit is circuit in which we combine the different gates, We're going to elaborate few important combinational circuits as follows.

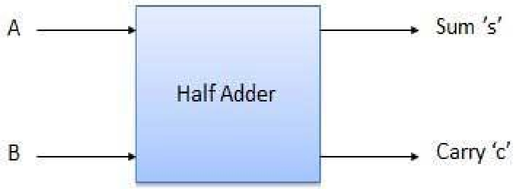
1- Basic Adders

Adders are important in computers and also in other types of digital systems in which numerical data are processed. An understanding of basic adder operation is fundamental to the study of digital systems. In this section, the half-adder and the full-adder are introduced.

- **Half - Adder (HA)**

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**.

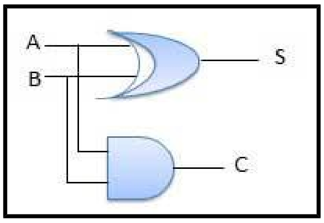
Block Diagram



Truth Table

Input		Outputs	
A	B	$\sum(s)$	C_o
0	0	0	0
1	0	1	0
0	1	1	0
1	1	0	1
Binary digits to be added		Sum	Carry out
		XOR	AND

Circuit Diagram



Now, what is the Boolean expression needed for the (Co) output ?
the Boolean expression is **C_o=A.B** which represent AND gate

Now, what is the Boolean expression needed for the ($\sum(s)$) output ?

The Boolean expression I s $\sum(s) = \overline{A}.B + A.\overline{B}$ which represent XOR gate.
We Can also simplify HA function using Karnough Map.

A B	0	1
0		1
1	1	
$S = \overline{A}B + A\overline{B}$ $S = A \oplus B$		

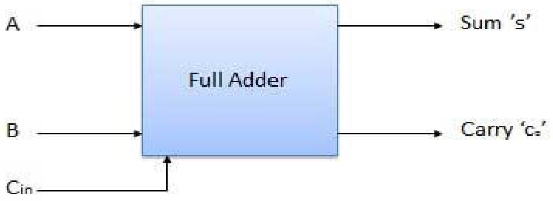
A B	0	1
0		
1		1
$C_o = A.B$		

Note: the half-adder circuit adds only the LSB column (1s column) in a binary addition problem.

- **Full – Adder (FA)**

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

The full adder must be used when it possible to have an extra Carry input ,then the full adder has three inputs: **A , B , and C_{in}**. These three inputs must be added to get the $\sum(s)$ and **C_o** output .

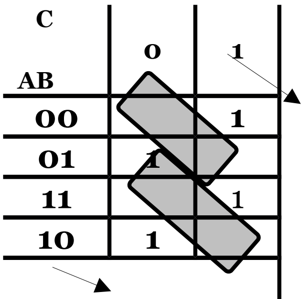
Input			Outputs	
A	**B**	**C_{in}**	$\sum(s)$	**C_o**
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1
A + B + C_{in}			**Sum**	**Carry out**

$\sum(s) = \sum(1,2,4,7)$

Co $= \sum(3,5,6,7)$

We can simplify **FA** Function using K-Map

$\sum(s) = \sum(1,2,4,7)$



AB \ C	0	1
00		1
01	1	
11		1
10	1	

$\overline{A}B\overline{C_{in}} + \overline{A}\,\overline{B}C_{in}$

$¿\overline{A}(B\overline{C_{in}} + \overline{B}C_{in})$

$¿\,\overline{A}.(B \oplus C_{in})$

$ABC_{in} + A\overline{B}\,\overline{C_{in}}$

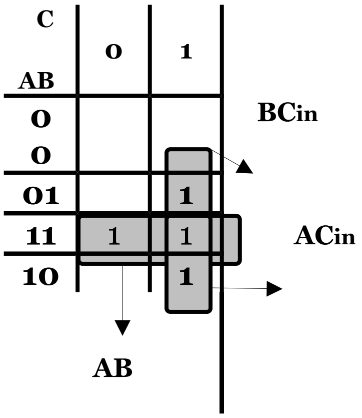
$¿A.(BC_{in} + \overline{B}\,\overline{C_{in}})$

$¿\,A.\overline{(B \oplus C_{in})}$

Note : Let **X=(B ⊕ Cin)** **then** $\therefore \sum = \overline{A}X + A\overline{X} \rightarrow \sum = A \oplus X \quad \therefore \sum = A \oplus B \oplus C_{in}$

By the same way

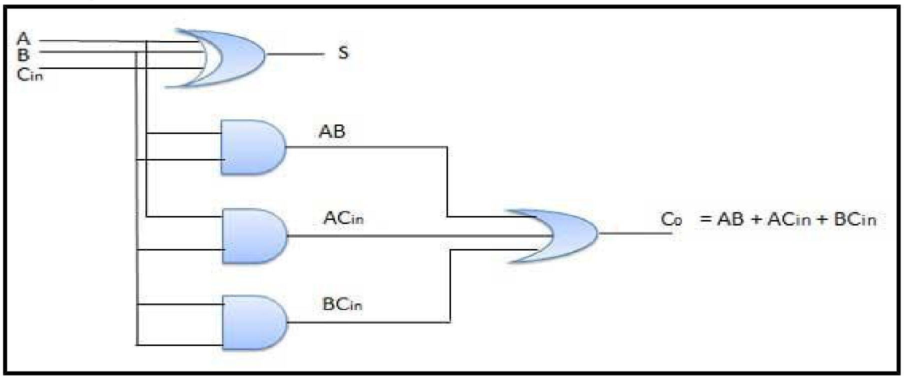
Co $= \sum(3,5,6,7)$



AB \ C	0	1
0 0		
01		1
11	1	1
10		1

$\therefore C_o = \sum BC_{in} + AB + AC_{in}$

Circuit Diagram



Homework:

How to construct Full -Adder from two Half –Adders ?

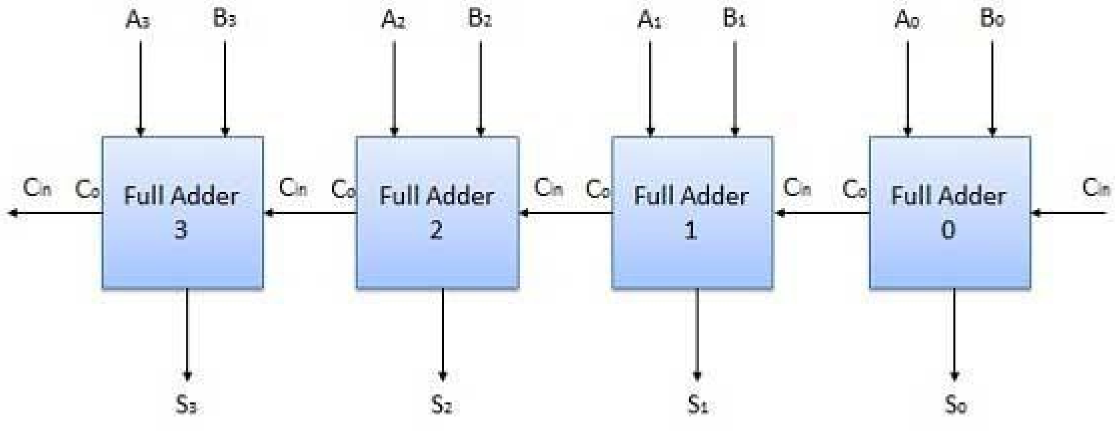
- ## N- bit Adder

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 are shown in fig. The four bit parallel adder is a very common logic circuit.

Block Diagram



Note : we can replace the first full adder with Half adder

2- Subtractors

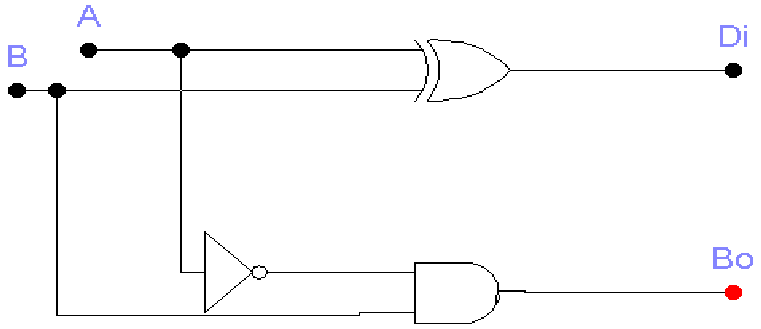
- **half subtractors**

You will find that adders and suntractors are very similar. You use half subtractors and full subtractors just as you use half and full adders. Converting the rules to truth table from as in bellow .

On the input side,(B) is subtracted from(A) to give output Di(Diffrence). If B is larger then A , we need a borrow , which is shown in the column labeled Bo (borrow out).

Inputs		Outputs	
A	B	Di	Bo
0	0	0	0
0	1	1	1
1	0	1	0
1	1	0	0
A - B		Difference	Borrow out

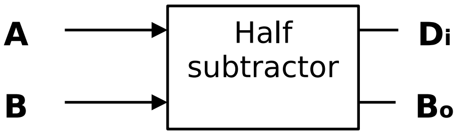
* form the truth table, we can determine the Boolean expression for the half-subtractor.



The expression for the Di column is : **Di= A ⊕ B** , this is the same as for the half adder .

The Boolean expression for the Bo column is $B_0 = \overline{A} , B$, combining these two expression in a logic diagram gives the logic circuit for a half subtractor.

Block Diagram



- **Full subtractors**

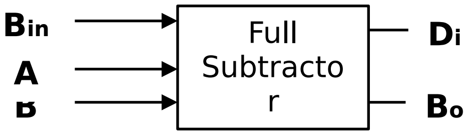
When you subtract several columns of binary digits, you must take into account the borrowing . the truth table that describe the full subtractors as fellow:

Input			Outputs	
A	B	Bin	Di	Bo
0	0	0	0	0

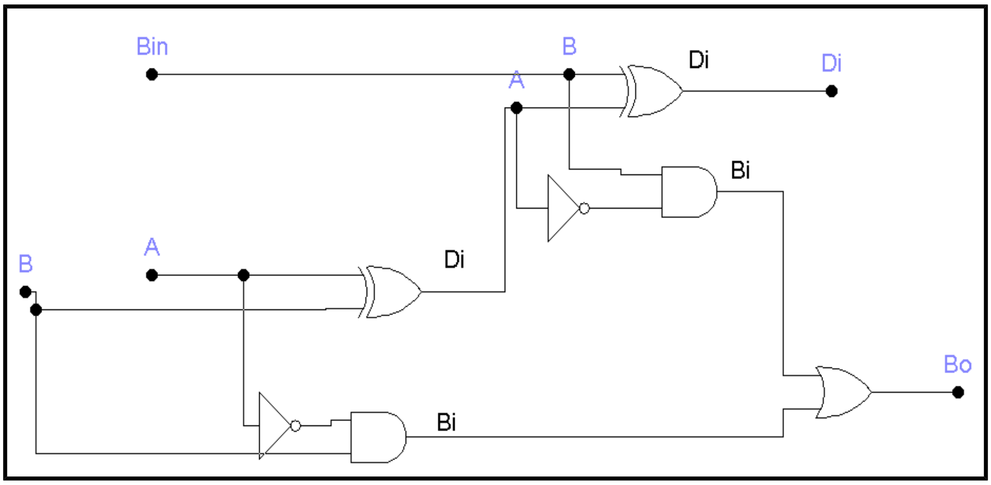
0	0	1	1	1
0	1	0	1	1
0	1	1	0	1
1	0	0	1	0
1	0	1	0	0
1	1	0	0	0
1	1	1	1	1
A - B - B_{in}			D_i	B_o

You might keep track of the differences and borrow . to solve such problem we must use full subtractor which has pervious truth table that considers all the possible combinations in Binary subtraction .

Block Diagram



Logic circuit

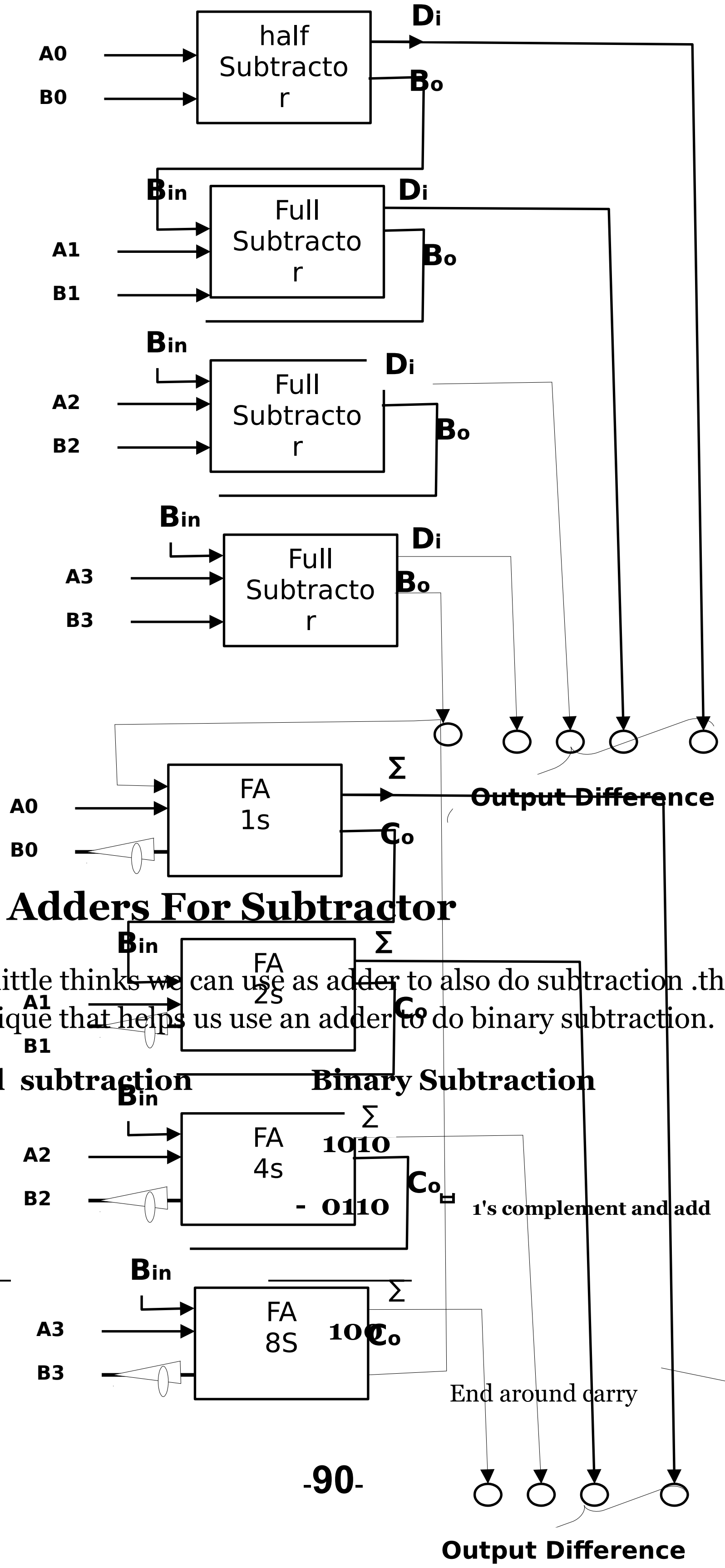


- **Parallel Subtractors**

Half and full subtractors are wired together to perform as a parallel subtractor , foe example , to subtract binary number B3 B2 B1 B0 from binary number A3 A2 A1 A0 we make a 4-bit parallel subtractors.

A3 A2 A1 A0

- B3 B2 B1 B0



• Using Adders For Subtractor

With A few a little thinks we can use as adder to also do subtraction .there is a mathematical technique that helps us use an adder to do binary subtraction.

Exp: Decimal subtraction Binary Subtraction

10	1010		1010
- 6	- 0110	1's complement and add	+1001
4	100		1 0011
		End around carry	+ 1

100

In this special technique the steps are first to first wire the 1's complement of the number being subtracted (change all 1's to 0's and all 0'sto 1's) and then add .

Now let us us adders to do binary subtraction in this example, the temporary answer to this addition is shown as (10011).next , the last carry on the left I carried around to the 1's place. This is called an end-around carry. When the end around carry is added to the rest of the number, the result is the difference between the original binary numbers , (1010) and (0110). The an swer to this problem is 0100.

3- Comparator

The comparison of two numbers is an operations that determines if one number is greater then , less than , or equal to other number.

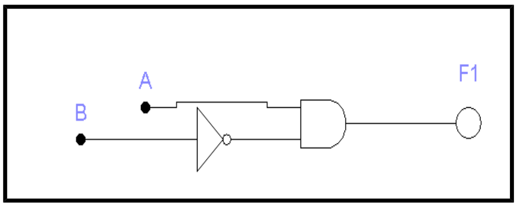
A Magnitude comparator : is a combinational circuit that compares two numbers A and B , and determines there relative . The outcome of the comparison is specified by three binary variables that indicate whether A > B , A=B or A <B .

A	B	A>B	A<B	A=B	A>=B	A<=B	A<>B

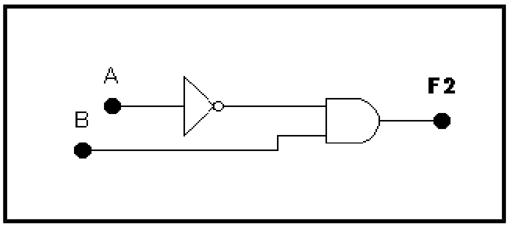
0	0	0	0	1	1	1	0
0	1	0	1	0	0	1	1
1	0	1	0	0	1	0	1
1	1	0	0	1	1	1	0

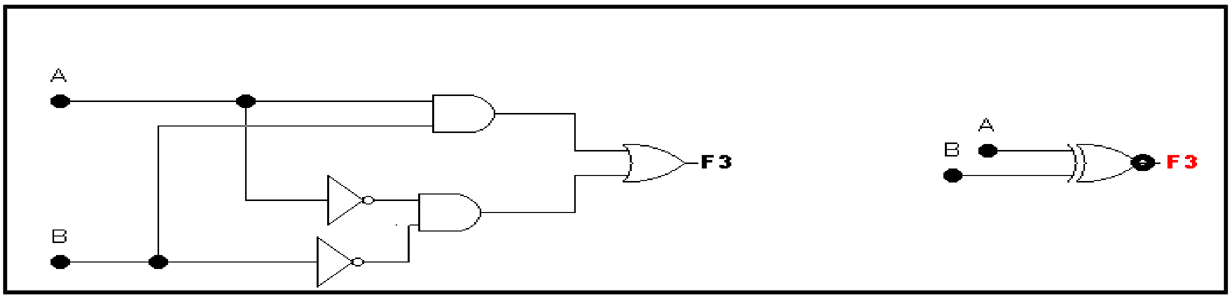
ـﻟby us logic circuit

$1-A > B \Leftrightarrow F1=A\overline{B}$



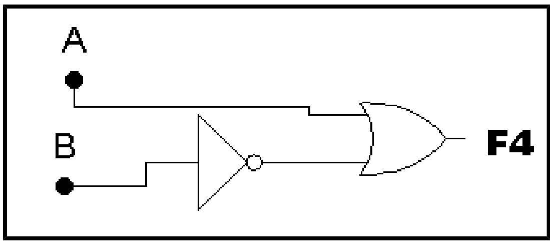
$2-A < B \Rightarrow F2=\overline{A}B$





$4-A >= B \Rightarrow F4=A+\overline{B}$

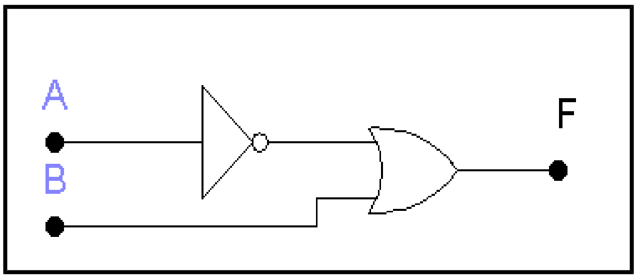
A \ B	0	1
0	1	1
1		1



$5- \ A <= B \Rightarrow F5 = \overline{A} + B$

B

A	0	1
0	1	
1	1	1



This is a comparison of two binary variables each has one bit only .

Note : Now what about comparing two numbers with n-bits ?

$Let \ A = a_{n-1} \, a_{n-2} \ldots\ldots\ldots a_1 a_0$
$Let \ B = b_{n-1} \, b_{n-2} \ldots\ldots\ldots b_1 b_0$

where a_0 , b_0 is **LSB** while a_{n-1} , b_{n-2} is **MSB** of two number

Equality Relation

We say that if the number A equal to B we are implement an XNOR gate to do this state , but with one bit number, then now about numbers with n- bits ?

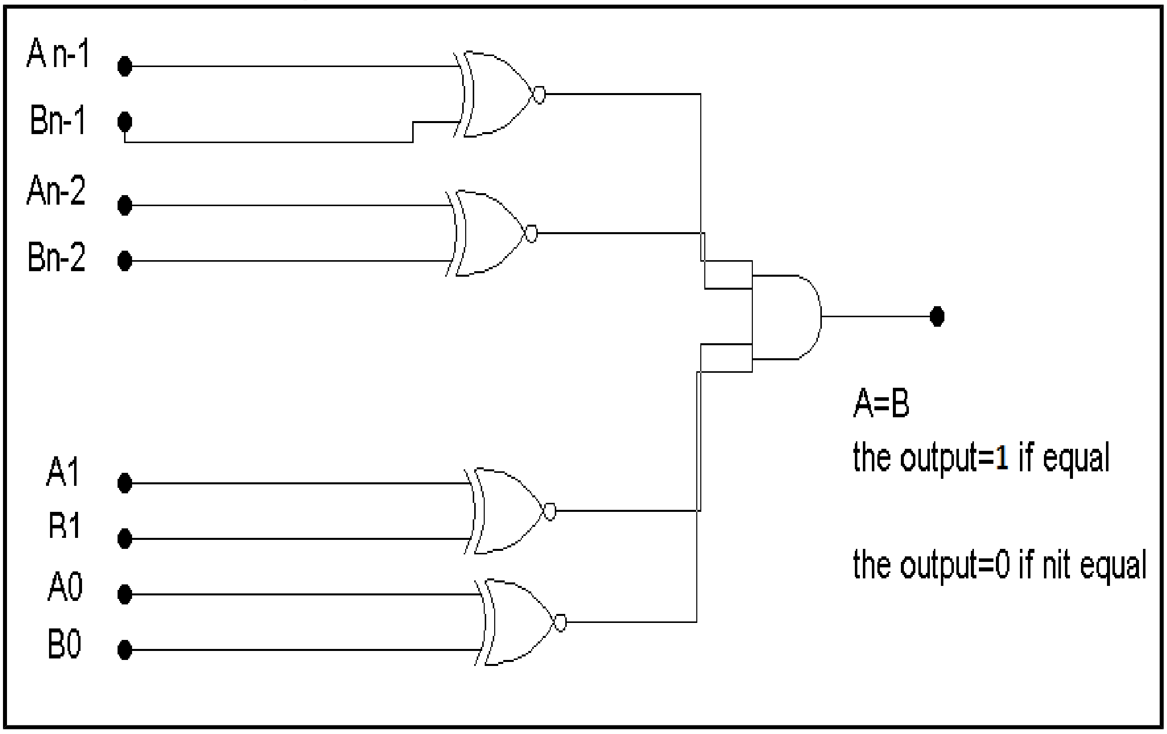
Note: the two numbers are equal if all pairs of significant digit are equal:

$$\underrightarrow{a_{n-1} = b_{n-1}} \ , \ \ldots\ldots\ldots\ldots\ldots\ldots \ , \ a_0 = b_0$$

That means for the equality condition being true (equal 1) if all equality relation of each pair must equal to 1, this dictates **AND** gate to combine the outputs to gather to get the final output 1.

Therefore Equality Relation:-

$$A = B \ \ if \ \left(\overline{a_{n-1} \oplus b_{n-1}}\right) . \left(\overline{a_{n-2} \oplus b_{n-2}}\right) \ \ldots\ldots\ldots\ldots\ldots\ldots \ , \left(\overline{a \oplus b_0}\right) = 1$$



Greater than relation

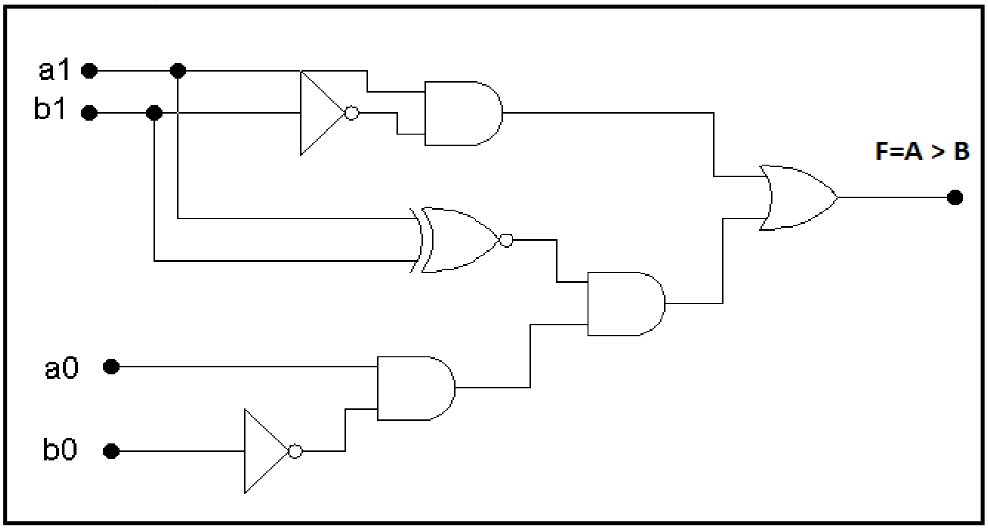
If the corresponding digit of is (1) and that of B is (0) , we conclude that A >B.

Then How can we implement a comparator circuit to do this comparison:

1- if N= 2

$$
\begin{aligned}
&Let\ \ A = a_1 a_0 \\
&Let\ \ B = b_1 b_0 \\
&to\ \ prof\ \ that\ \ \ A > B\ \ if\ \ \ a_1 a_0 > b_1 b_0 \\
&\ \ = (a_1 > b_1) + (a_1 = b_1)\,.\,(a_0 > b_0) \\
&and\ \ from\ \ truth\ \ table: \\
&(a_1 > b_1) = a_1 \overline{b_1} \\
&(a_1 = b_1) = (\overline{a_1 \oplus b_1}) \\
&(a_0 > b_0) = a_0 \overline{b_0}
\end{aligned}
$$

$$then\ \ F(A > B) = a_1 \overline{b_1} + (\overline{a_1 \oplus b_1})\,.\,a_0 \overline{b_0}$$



2- if N= 3

$$Let\ \ A = a_2 a_1 a_0$$
$$Let\ \ B = b_2 b_1 b_0$$

then

$$F(A > B) = (a_2 > b_2) + (a_2 = b_2) . (a_1 > b_1) + (a_2 = b_2) . (a_1 = b_1) . (a_0 > b_0)$$

$$\because\ (a_2 > b_2) = a_2 \overline{b_2}$$
$$(a_2 = b_2) = (\overline{a_2 \oplus b_2})$$
$$(a_1 > b_1) = a_1 \overline{b_1}$$
$$(a_1 = b_1) = (\overline{a_1 \oplus b_1})$$
$$(a_0 > b_0) = a_0 \overline{b_0}$$

$$\therefore\ A > B = (a_2 \overline{b_2}) + (\overline{a_2 \oplus b_2}) . (a_1 \overline{b_1}) + (\overline{a_2 \oplus b_2}) . (\overline{a_1 \oplus b_1}) . (a_0 \overline{b_0})$$

Note : in the same way of comparison we can find

- If N=2

$$Then\ \ A < B = \overline{a_1}\, b_1 + (\overline{a_1 \oplus b_1}) . \overline{a_0}\, b_0$$

- If N= 3

Then $A < B = (\overline{a}_2 \, b_2)+(\overline{a_2 \oplus b_2}) \, . \, (\overline{a}_1 \, b_1) + (\overline{a_2 \oplus b_2}) \, . \, (\overline{a_1 \oplus b_1}) \, . \, (\overline{a}_0 \, b_0)$

4- Decoders

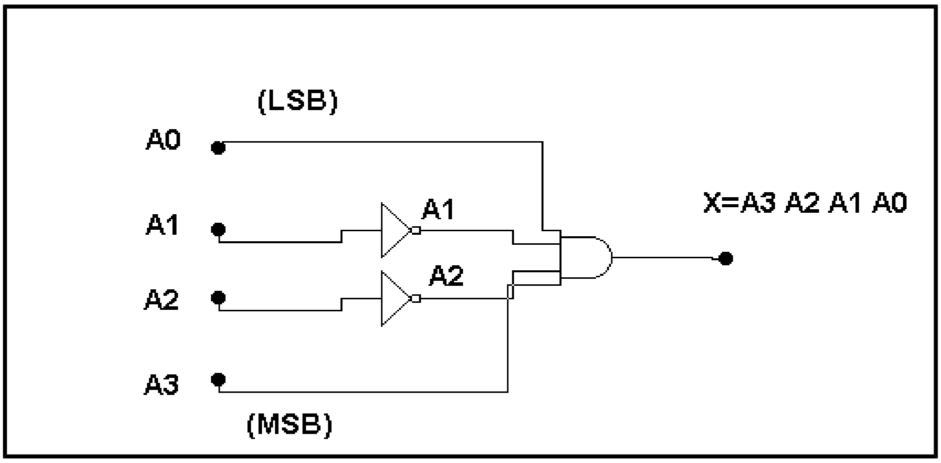
A decoder is a digital circuit that detects the presence of a specified combination of bits (code)on its inputs and indicates the presence of that code by a specified output level .

In its general form : a decoder has **n** input lines to handle n bits and from one to **2^n** output lines to indicate the presence of one or more n-bit combinations.

The Basic Binary Decoder

Suppose you need to determine when a binary 1001 occurs on the inputs of a digital circuit. An AND gate can be used as the basic decoding element because it produces a HIGH or (1) output only when all of its inputs are HIGH (1).

Therefore , you must make sure that all of the inputs to the AND gate are HIGH when the binary number 1001 occurs; this can be done by inverting the two middle nits (the 0s), as shown in the next figure .



you should verify that the output is (0) except when A0=1 ,A1=0, A2=0, AND A3=1 are applied to the inputs. A0 is the LSB and A3 is the MSB.

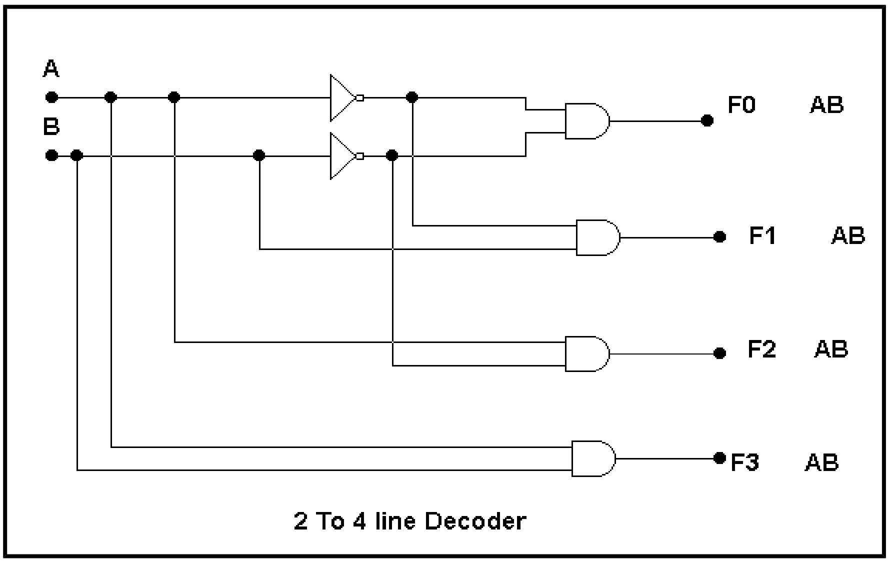
Note: If a NAND gate is used in place of the AND gate in pervious figure , a LOW output will indicate the presence of the proper binary code, which is 1001 in this case.

Note : Then the decoders presented here are called **n to m line decoders,** where m <= $\mathbf{2^n}$. the purpose is to generate the $\mathbf{2^n}$ (or fewer) minterms of n input variable. The name decoder is also used in conjunction with other code converters such as **BCD-to-Seven segment decoder**.

2 to 4 line decoder

We can design 2 to 4 line decoder by using the following truth table:

Inputs		Outputs			
A	B	F0	F1	F2	F3
0	0	1	0	0	0
0	1	0	1	0	0
1	0	0	0	1	0
1	1	0	0	0	1



2 To 4 line Decoder

3 to 8 line decoder

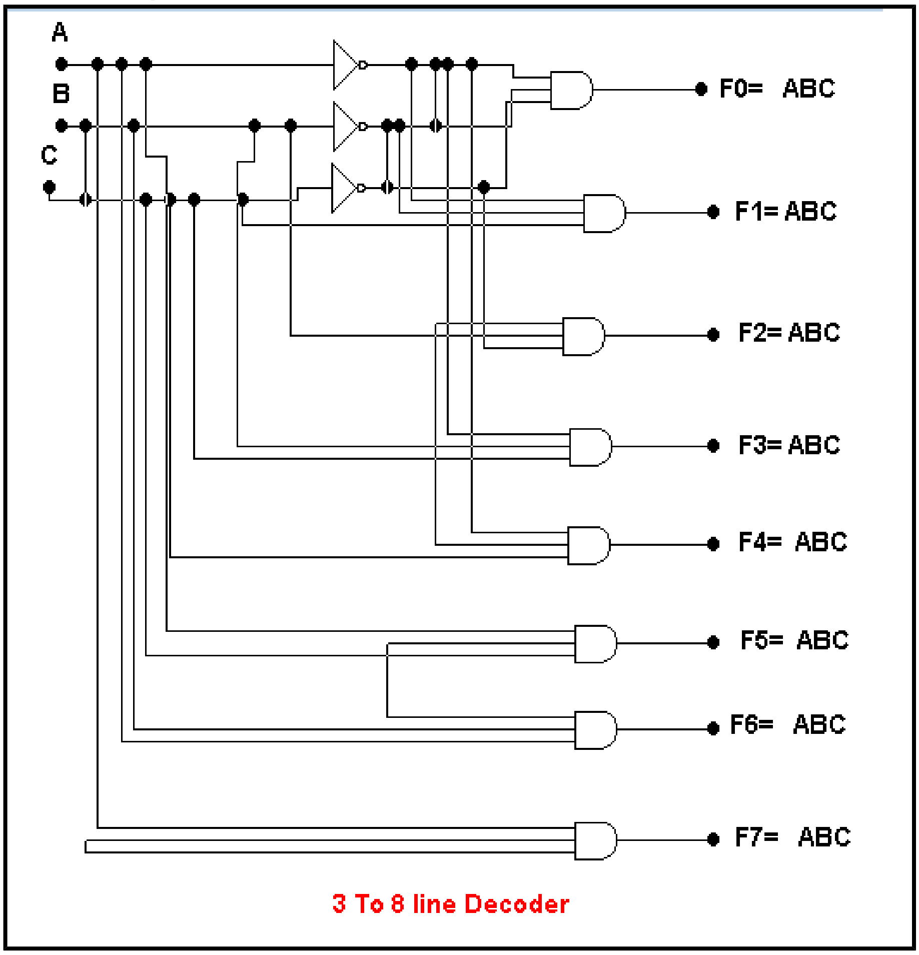
We can design 3 to 8 line decoder by using the following truth table:

A	B	C	F0	F1	F2	F3	F4	F5	F6	F7
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	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

The three inputs are decoded into eight outputs each representing one of the mineterms of the three input variables.

The three inputs are decoded into eight outputs each representing one of the mineterms of the three input variables.

The three inverters provides the complement of the inputs, and each of the eight AND gates generates one of the mineterms. A particular application of this decoder is Binary –To -Octal conversion.



3 To 8 line Decoder

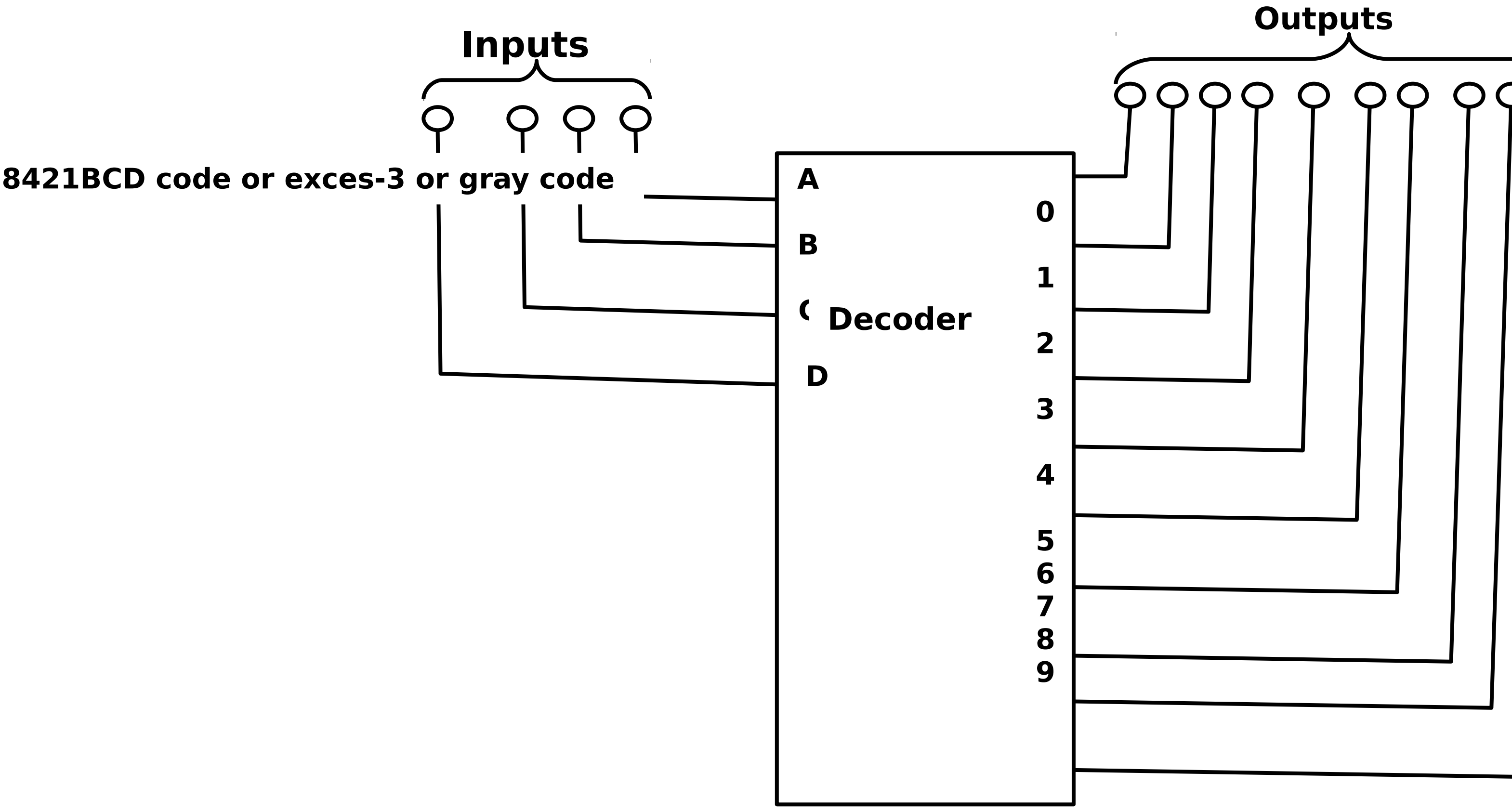
The input variables represent a binary number, and the outputs represent the eight digits in the octal number system.

However , a (3 –to - 8) line decoder can be used for decoding any **3-bit code** to provide **eight** output, one for each element of the code.

The operation of the decoder may be clarified by the truth table. For each possible input combination there are seven outputs that are equal to **(0)** and only one that is equal to **(1)** .

The output whose value is equal to **(1)** represents the minterms equivalent of binary number a variable in the input lines.

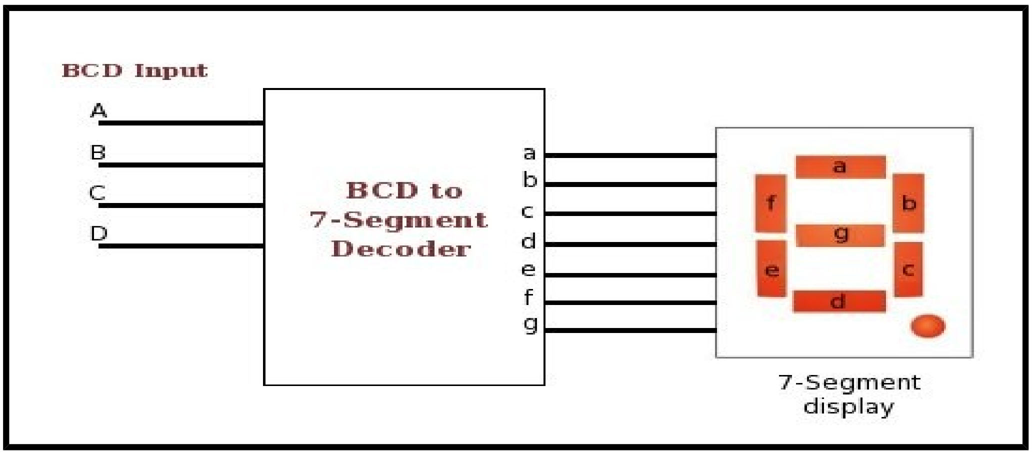
Notes : Decoders as we say came in several varieties in typical decoder , the inputs may be: 8421 BCD , axcess-3 , or gray code .



A typical decoder block diagram

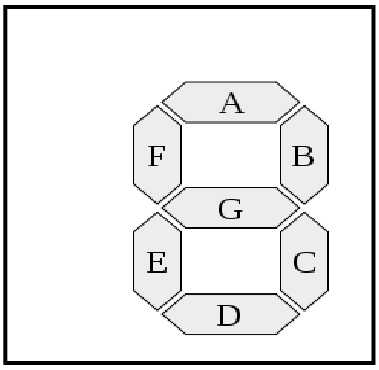
4-H BCD-to-Seven segment decoder

The decoders are translating the (8421 BCD) code to (a seven segement display)

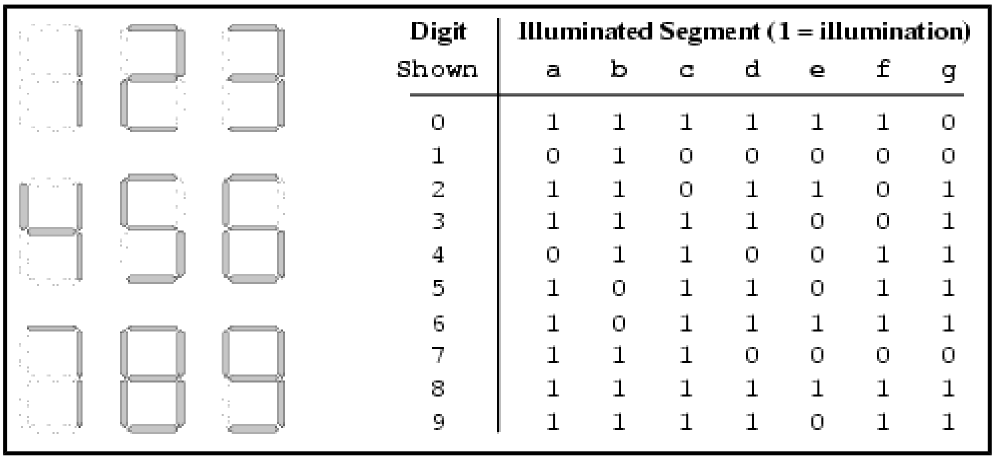


Suppose you wanted the decimal (4) to light on the display board, the BCD umber (0100) at the input of the BCE-to-Seven-segment decoder.

The Decoder activates outputs(a ,c ,d , f and g) to light segments.



Segment identification

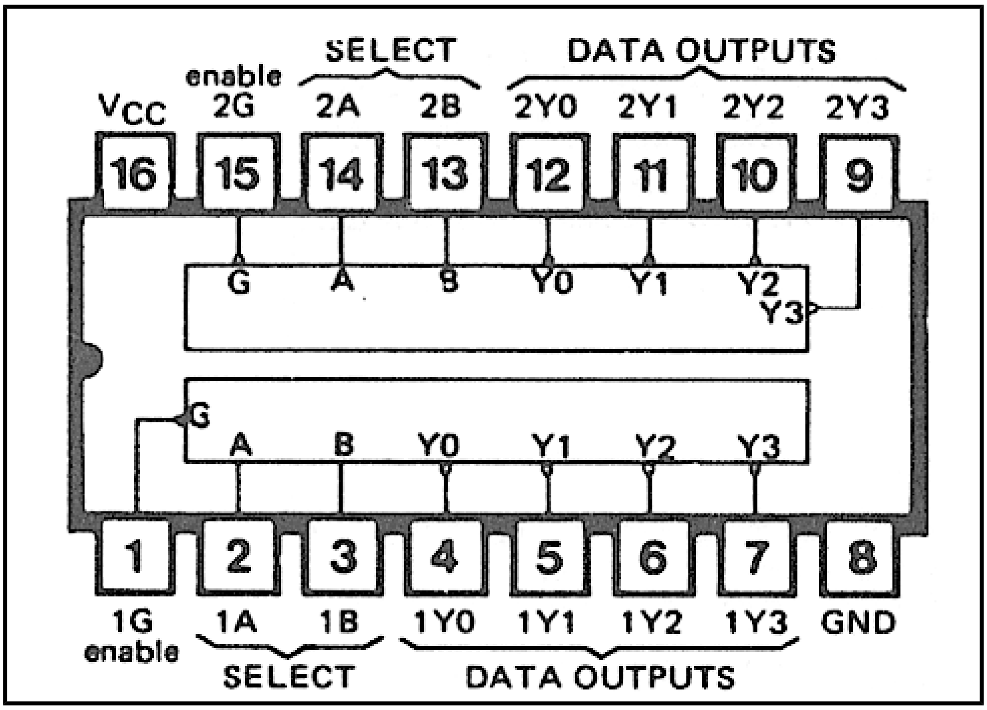


Digit Shown	Illuminated Segment (1 = illumination) a	b	c	d	e	f	g
0	1	1	1	1	1	1	0
1	0	1	0	0	0	0	0
2	1	1	0	1	1	0	1
3	1	1	1	1	0	0	1
4	0	1	1	0	0	1	1
5	1	0	1	1	0	1	1
6	1	0	1	1	1	1	1
7	1	1	1	0	0	0	0
8	1	1	1	1	1	1	1
9	1	1	1	1	0	1	1

Decimal numbers on typical seven-segment display

Decimal	Inputs				Outputs						
	D	C	B	A	a	b	c	d	E	f	g
0	0	0	0	0	1	1	1	1	1	1	0
1	0	0	0	1	0	1	1	0	0	0	0
2	0	0	1	0	1	1	0	1	1	0	1
3	0	0	1	1	1	1	1	1	0	0	1
4	0	1	0	0	0	1	1	0	0	1	1
5	0	1	0	1	1	0	1	1	0	1	1
6	0	1	1	0	1	0	1	1	1	1	1
7	0	1	1	1	1	1	1	0	0	0	0
8	1	0	0	0	1	1	1	1	1	1	1
9	1	0	0	1	1	1	1	1	0	1	1

↵ The 74139 decoder



2-line-to 4 Decoder IC. No(74139)

This type of decoders (which we used in laboratory) with an enable input to control the circuit operation, as we see in previous figure . the decoder is enabled when (enable)is equal to (0).

The circuit operates with complement outputs and complement input , when (E) is equal to (0) only one output can be equal to (0) at any given time all other outputs are equal to (1) .

The output whose value is equal to (0) represents the minterm selected by inputs (A) and (B).

The circuit is disabled when (E) is equal to (1). When the circuit is disabled ,none of the outputs are equal to (0) and none of the minterms.

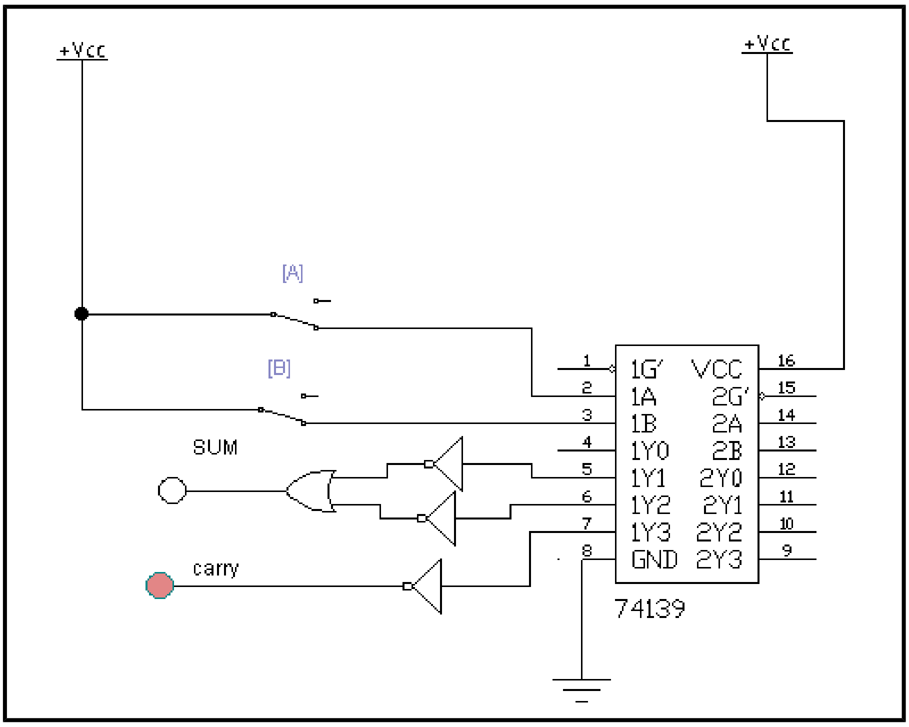
Example : Design Half-adder using (74139) (2-to-4) Decoder .

As indicated by the truth table , only two output can be equal to (1) at (S) output , while (C0) has only one output equal to (0).

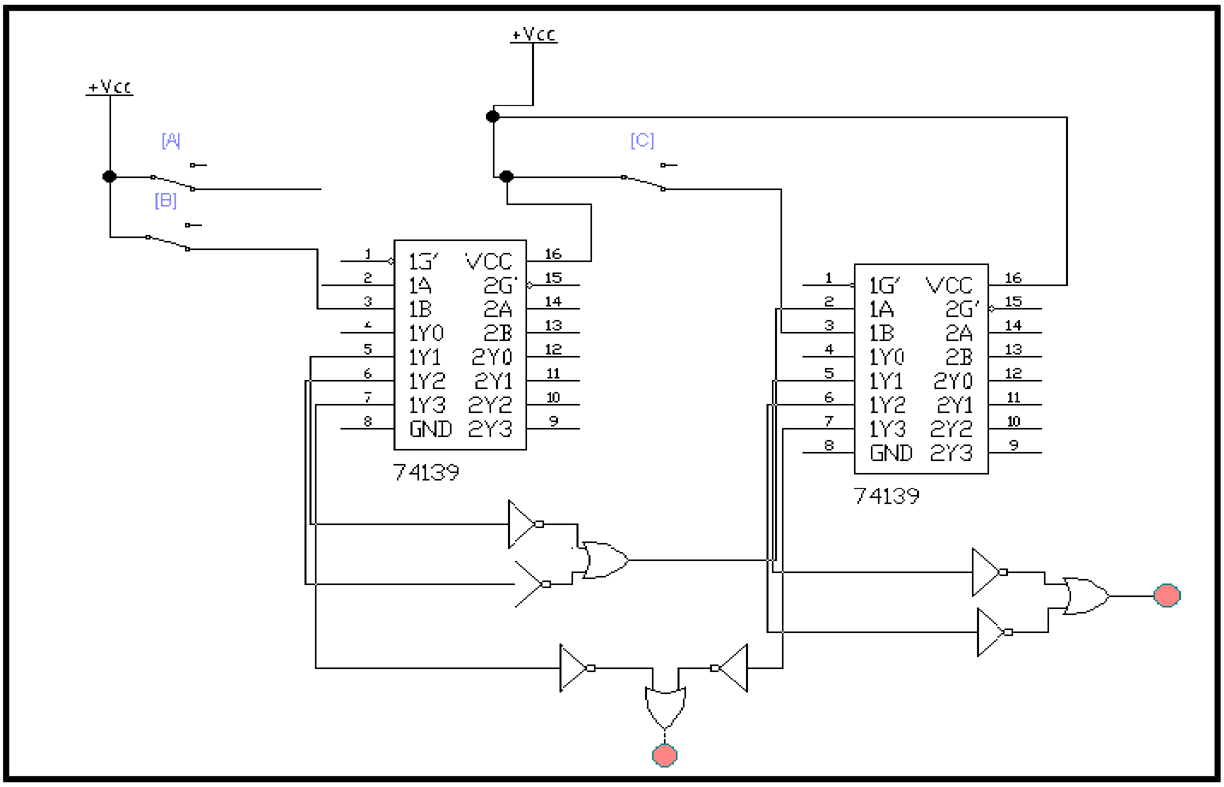
Input		Outputs	
A	B	$\sum(s)$	C_o
0	0	0	0
1	0	1	0
0	1	1	0
1	1	0	1
Binary digits to be added		Sum	Carry out
		XOR	AND

-Truth table of half-adder

The (74139) decoder provide a complement outputs as is shown in IC figure , therefore we must connect **not gate** for each active output.



Example :Desgin Full-adder from two (74139) decoder.



5- Encoders

An Encoder is a digital circuit that perform the inverse operation of a decoder . An encoder has 2^n (or fewer) input and n output lines.

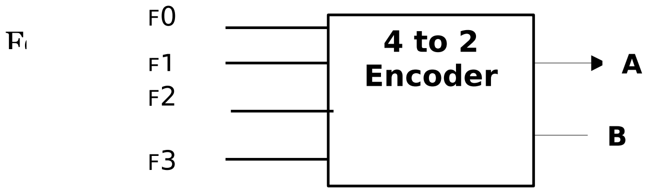
The output lines generate the binary code corsponding to the input value. One of the most common Decoders is **(4 to 2)** encoder.The encoder in this system must translate the decimal input signals to binary output signals.

Inputs				**Outputs**	
F0	**F1**	**F2**	**F3**	**A**	**B**
1	**0**	**0**	**0**	**0**	**0**
0	**1**	**0**	**0**	**0**	**1**
0	**0**	**1**	**0**	**1**	**0**
0	**0**	**0**	**1**	**1**	**1**

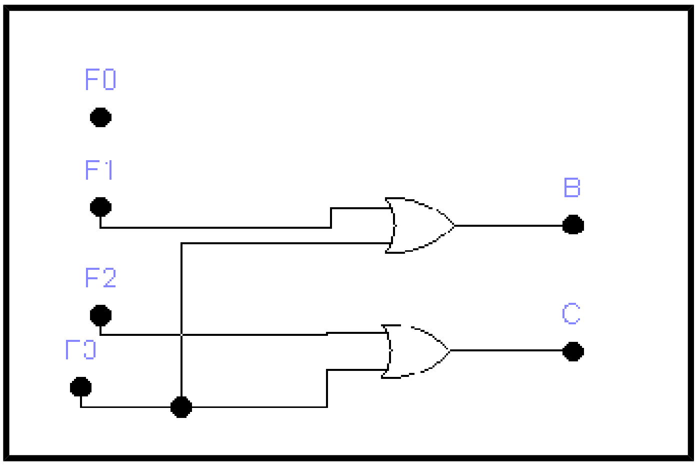
Truth table Of (4-2) Encoder

The truth table is the origin of most logic circuit you must have al the combinations that generate a logic 1 in the truth table, therefore

A=F2 + F3 , B=F1 + F3



Block Diagram Of Encoder



Logic Diagram of Encoder

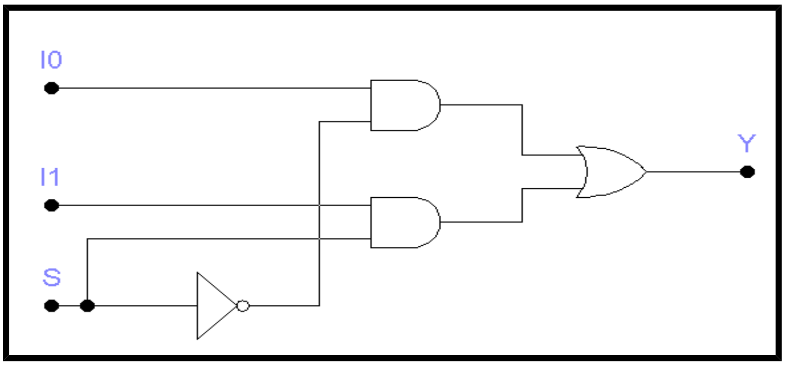
6- Multiplexers

A multiplexer is a combinational circuit that selects binary information from one of many input lines and directs it to a single output line.

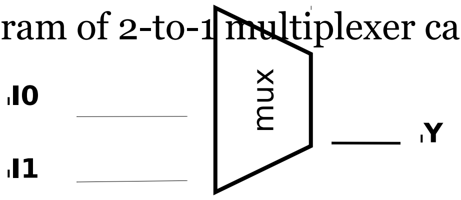
The selection of a particular input line is controlled by a set of selection lines. Normally , there are **2ⁿ** inputs and **n** selection lines whose bit combination determine which input is selected and **1** output.

2 – to- 1 Line multiplexer

A 2 to 1 multiplexer connects one of two sources to a common destination as shown in figure below:



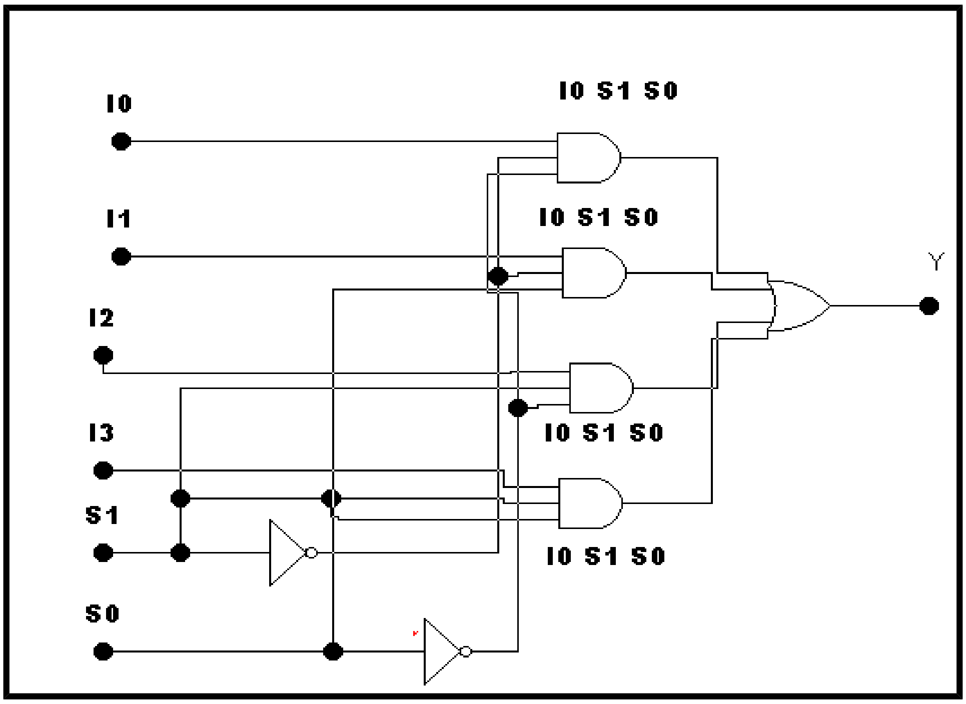
the block diagram of 2-to-1 multiplexer can be shown fellow :



The circuit has two data input lines, one output line, and one selection line **S**.

When **S=0** , the upper **AND** gate is enabled and I0 has a path to the output. When **S=1** the lower **AND** gate is enabled and I1 has a path to the output. The multiplexer acts like an electronic switch that selects one of two sources.

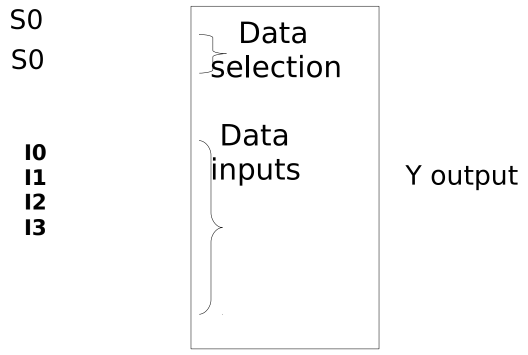
4 – to - 1 Line multiplexer



S1	S2	Y
0	0	I0
0	1	I1

1	0	I2
1	1	I3

- Each of the four inputs , **I0** through **I3** , is applied to one input of an **AND** gate . selection lines **S1** and **S0** are decoded to select a particular AND gate.
- The outputs of the **AND** gates are applied to a single **OR** gate that provides the **1-line output**.
- The function table lists the input that passed to the output for each combination of the binary selection values .
- To demonstrate the circuit operation , consider the case when s1s0=10. The AND gate associated with input I2 has two of its input equal to 1 And the third input connected to I2 has the other three AND gates have at least one input equal to 0, which makes their outputs equal to 0.
- The OR gat output is now equal to the value of I2 , providing a path from the selected input to the output.
- A multiplexer is also called **data selector**, since is selects one of many inputs and steers the binary information to the output line.
- The AND gates and inverters in the multiplexer resemble a decoder circuit and indeed they decode the selection input lines.
- In general **2^n-to-1 line** multiplexers constructed from an **2^n** input lines , one to each AND gate.
- The outputs of the AND gates are applied to a single OR gate.
- The size of a multiplexer is specified by the number **2^n** of its data input lines and the single output line.
- The **n** selection lines are implied from **2^n** data lines.
- As in decoders, multiplexers may have an enable input to control the operation of the unit . when the enable input is in the in active state, the outputs are disabled , and when it is in the active state , thr circuit functions as a normal multiplexer.



When (S0=0 and S1=0) the output Y will be I0

And if (S0=1 and S1=0) ▭ y= I1

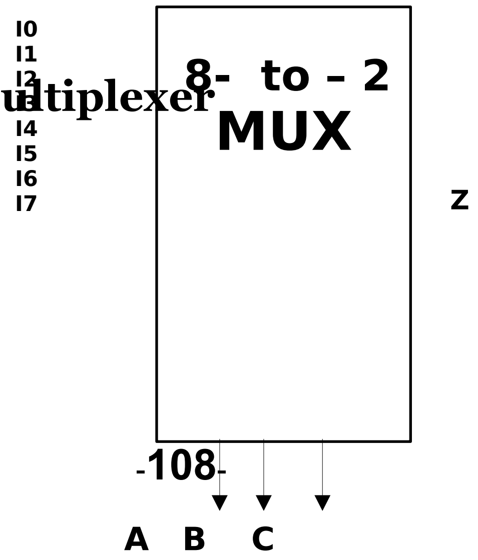
And if (S0=1 and S1=1) ▭ y= I2

And if (S0=1 and S1=1) ▭ y= I3

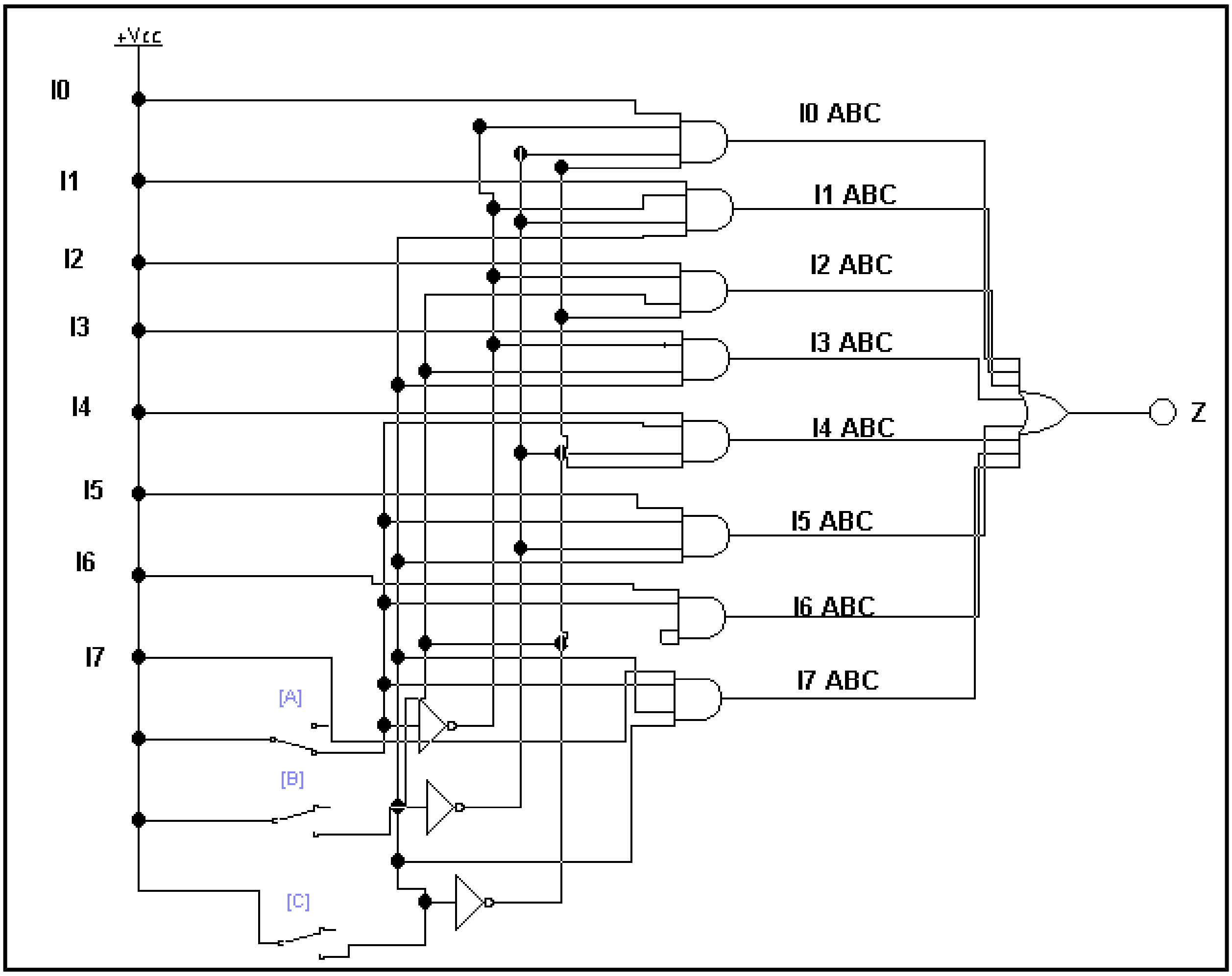
Data inputs	**Data Selected**		**Output**
	S1	**S0**	**Y**
I0	**0**	**0**	$I_0 \rightarrow \overline{S1}\,\overline{S0}$
I1	**0**	**1**	$I_1 \rightarrow \overline{S1}\,S0$
I2	**1**	**0**	$I_0 \rightarrow S1\,\overline{S0}$
I3	**1**	**1**	$I_0 \rightarrow S1S0$

Selection lines(s1) and (so) are decoded to select a particular AND gate. The outputs of the AND gates are applied to a single OR gates that provide the (1-line) output.

8 – to - 1 Line multiplexer



An 8-to-1 Mux can be used to realize any 4-variable function with no added gates. Three of variable are used as control inputs to Mux and the remaining variable is used as required on the data inputs.



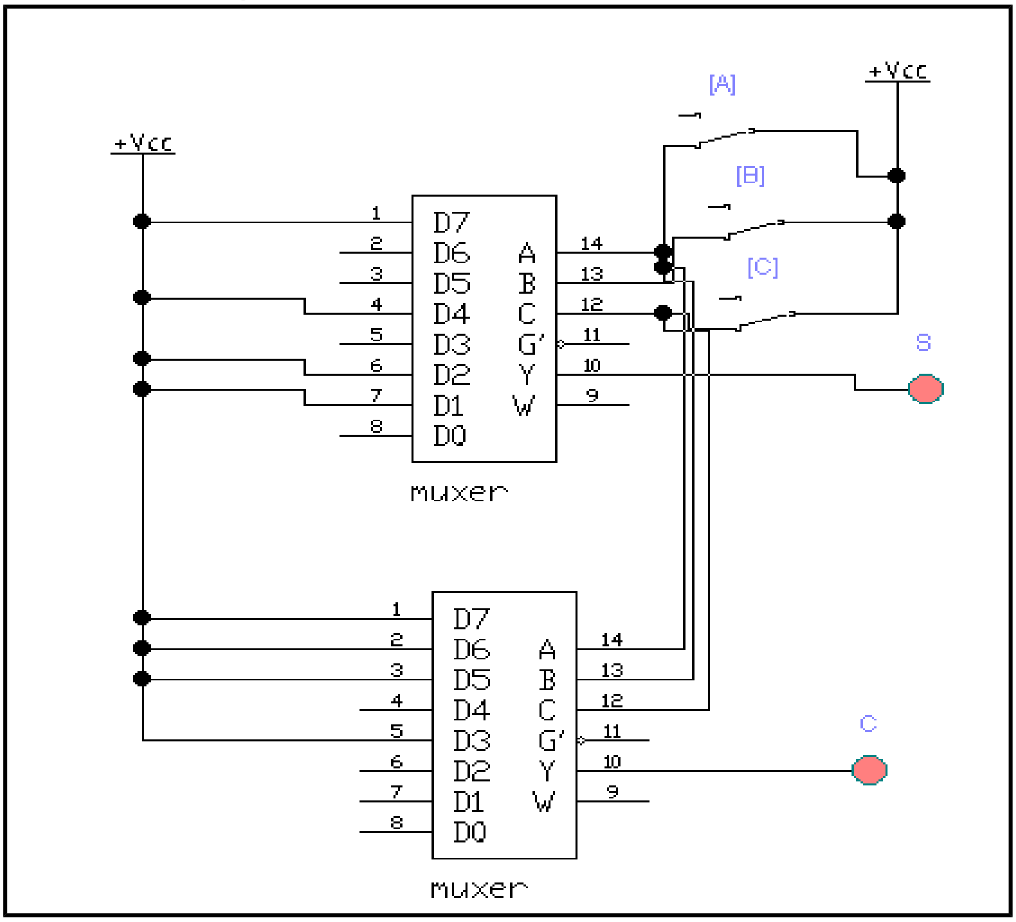
Data Inputs	Data Selected			Output
	A	B	C	
I_0	0	0	0	I_0
I_1	0	0	1	I_1

I_2	0	1	0	I_2
I_3	0	1	1	I_3
I_4	1	0	0	I_4
I_5	1	0	1	I_5
I_6	1	1	0	I_6
I_7	1	1	1	I_7

Example1: using (8*1) multiplexer to realize full adder

from truth table

inputs			output		
A	B	C_{in}	$\sum(s)$	C_o	Decimal NO.
0	0	0	0	0	0
0	0	1	1	0	1
0	1	0	1	0	2
0	1	1	0	1	3
1	0	0	1	0	4
1	0	1	0	1	5
1	1	0	0	1	6
1	1	1	1	1	7
$A + B + C_{in}$			Sum	Carry out	



1. *first we have 2^3 inputs =8*

2. *then we have n=3 selection line to each mux*

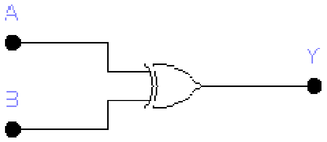
3. *one output*

4. *full adder need 2 outputs sum and carry then we must connect two mux.*

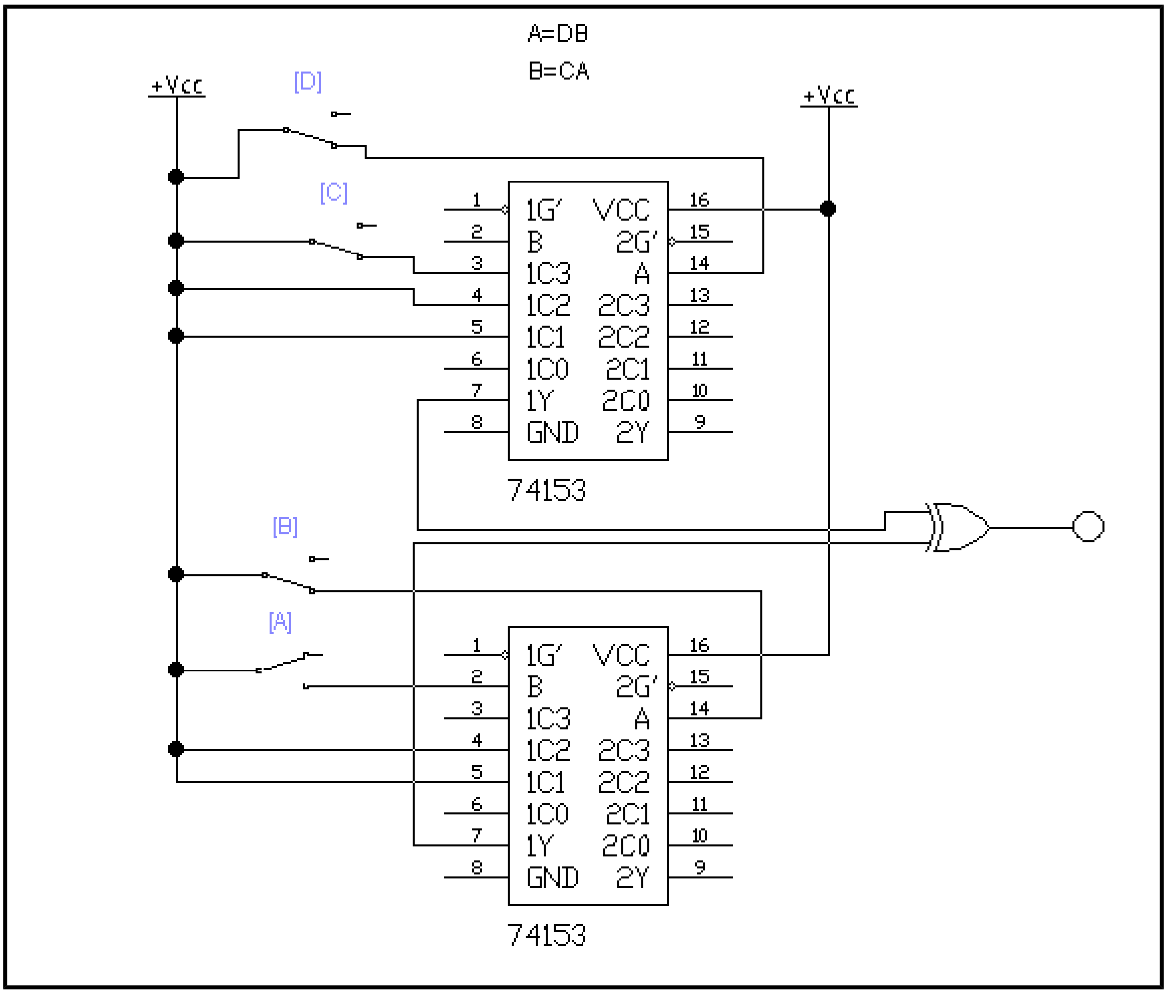
Example2: ***Design A<>B comparator by using Multiplexers when A, B consist of two digits(N=2) ? hint (use 74153 Mux)***

Answer

A <> B ⇔ XOR Gates



Decimal	Bo	Ao	y
0	0	0	0
1	0	1	1
2	1	0	1
3	1	1	0

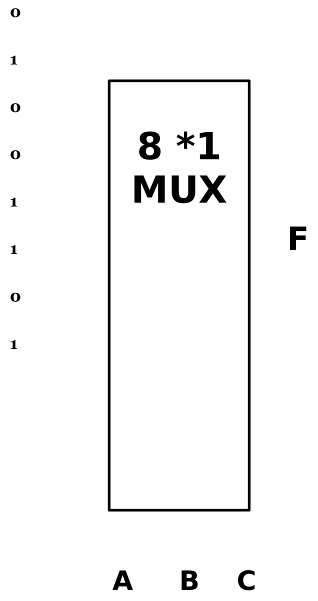


<u>Example3</u>: using multiplexer to realize the switching function

F(A , B , C)= $\sum(1 , 4 , 5 , 7)$

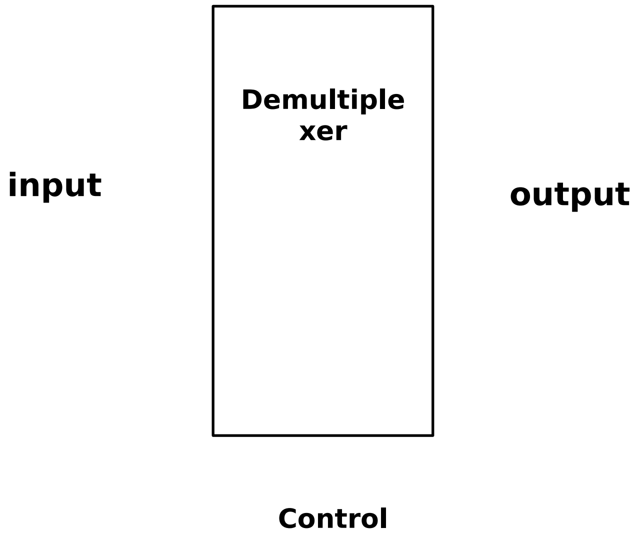
Decimal	A	B	C	Y
0	0	0	0	0

1	0	0	1	1
2	0	1	0	0
3	0	1	1	0
4	1	0	0	1
5	1	0	1	1
6	1	1	0	0
7	1	1	1	1



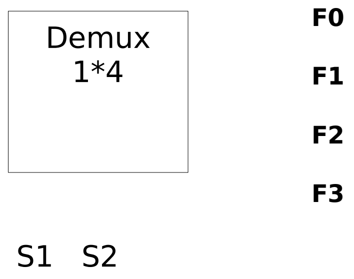
7- Demultiplexer

Is a combinational circuit that perform the inverse operation of Mux

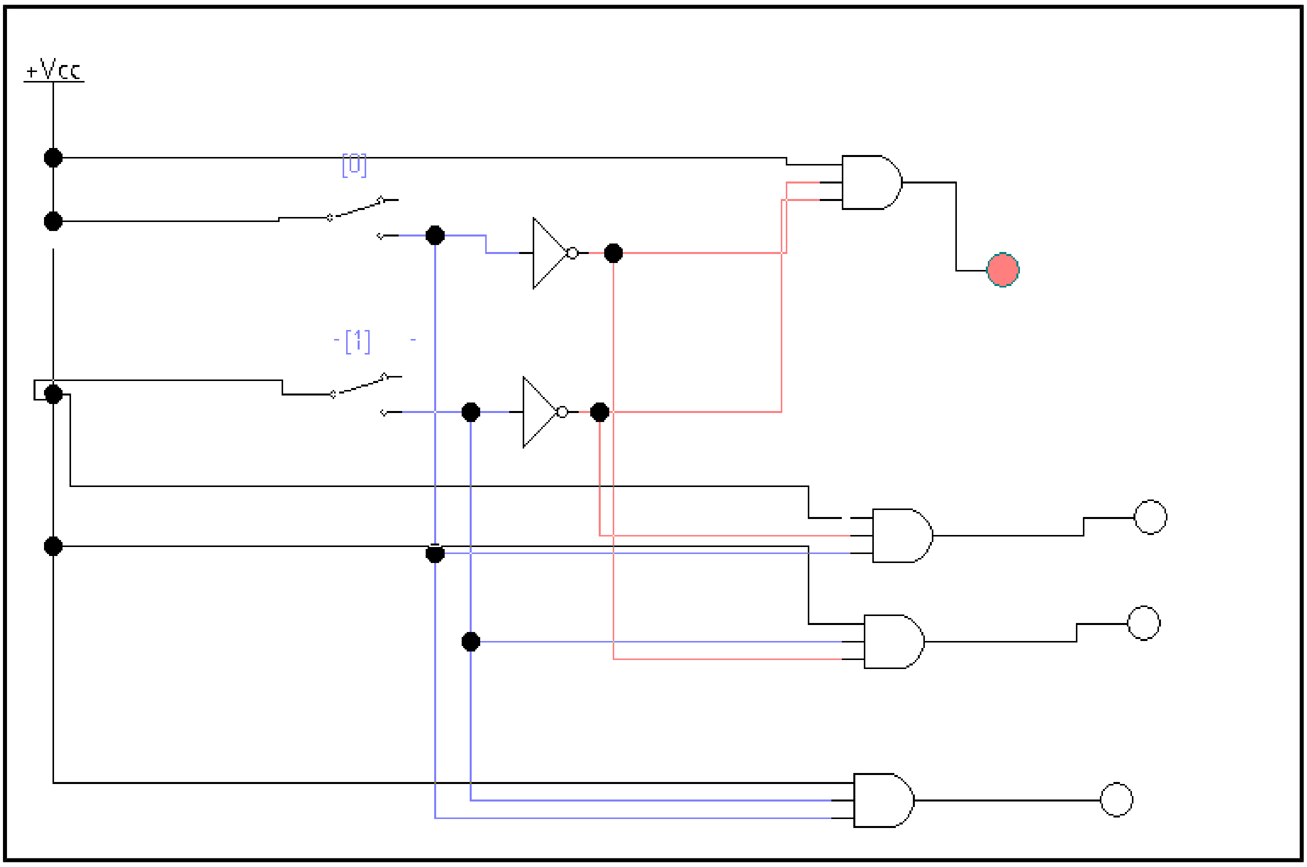


It has one input and **2^n** outputs and **n** control lines. The uses of demux is less than the mux circuit in electronics world .

Example : design (1 * 4) Demux



S1	So	output
0	0	F0
0	1	F1
1	0	F2
1	1	F3



7-Introduction to Computer Architecture

Most computers have similar architectures that combine software and hardware.

- **Hardware**

The term hardware refers to the physical components of your computer such as the system unit, mouse, keyboard, monitor, processors, memory and peripheral devices etc...

- **Software**

The software is the collection of instructions which makes the computer work. For instance, when you type in words via the keyboard, the software is responsible for displaying the correct letters, in the correct place on the screen. Software is held either on your computer's hard disk, CD-ROM, DVD or on a diskette (floppy disk) and is loaded (i.e. copied) from the disk into the computers RAM (Random Access Memory), as and when required.

Software includes the operating system which controls the computer hardware and application software, such as word processing, spreadsheets, etc…

+ Input devices

Input devices allow you to input information to the computer and include things such as the keyboard and mouse.

+ Output devices

Output devices allow you to output information from the computer and include the printer and the monitor.

+ Peripheral device

A peripheral device is any device which you can attach to your computer. Thus, you could attach a scanner or modem to the back of your system unit.

Main Parts of a Personal Computer

1-The System Unit

The "system unit" is the name given to the main PC box which houses the various elements which go together to make up the PC. For instance within the system unit is the computer system's motherboard, which contains all the main components, such as the CPU. The system unit also houses items such as the hard disk, the floppy disk and CD-ROM drives etc.

2-The System (Mother) Board

The system (mother) board is contained within your system unit and all the vital computer systems plug directly into the system board. The CPU is normally housed on your system board along with all the other electronic components. Other items such as the hard disk are attached to the system board, either directly or via cables. These boards are getting smaller and smaller as the components become more integrated.

3-The CPU

The CPU (Central Processing Unit) is normally an Intel Pentium (or equivalent) and it is one of the most important components within your computer. It determines how fast your computer will run and is measured by its MHz or GHz speed. Thus, a 2 GHz Pentium is much faster than say a
GHz Pentium CPU. It is the CPU which performs all the calculations within the computer, when running programs such as word-processors, spreadsheets and databases.

4-Memory (RAM)

The RAM (Random Access Memory) within your computer is where the operating system is loaded to when you switch on your computer and also where your applications are copied to when you start an application, such as a word processor or database program. When you create data, (e.g. letters and pictures), these are initially created and held in RAM and then copied to disk when you save the data. As a rule of thumb, the more RAM you have installed in your computer the better.

5-ROM-BIOS

The ROM-BIOS (Read Only Memory - Basic Input Output System) chip is a special chip held on your computer's system (mother) board. It contains software which is required to make your computer work with your operating system, for instance it is responsible for copying your operating system into RAM when you switch on your computer.

6-Serial Port

The serial port is a socket located at the back of your computer which enables you to connect items to the computer, such as a modem. They are commonly labeled as COM1 or COM2.

7-Parallel Port

The parallel port is a socket located at the back of your computer which enables you to connect items to the computer, such as a printer. It is commonly labeled as LPT1 or LPT2.

8-Universal Serial Bus (USB)

The Universal Serial Bus is a relatively new item within the PC. You will see one or more USB sockets at the back of the system unit, allowing you to plug in devices designed for the USB. These devices include printers, scanners and digital cameras.

Computer hardware

Computer based information system (CBIS) are composed of hardware, software, databases, people, telecommunications, and procedures. The components are organized to input, processing, output data and information. Physical equipment used for the input, processing, output and storage activities of computer system.

It consists of the following:
- Central processing unit (CPU)
- Memory (primary and secondary storage)
- Input technology
- Output technology
- Communication technology

1-The Central Processing Unit

The central processing unit (CPU) perform the actual computation inside any computer, the CPU is a microprocessor for example, Pentium III) made up of millions of microscopic transistors embedded in a circuit on a silicon wafer or chip. Examples of specific microprocessor.

The microprocessor has different portions which perform different functions:

1-**Control Unit**: this controls the flow of information.
2-**Arithmetic Logic Unit** (ALU) performs arithmetic calculations.
3-**Registers**: which store very small amount of data and instructions for short period of time.

* Control unit

-Direct and coordinates all units of the computer to execute program steps.
-Direct and coordinate all operation of the computer systems.
These operations include;

1- Control to the input and output devices.
2- Entry and retrieval of information from memory.
3-Routing of information between the memory, arithmetic and logic unit

Control unit automatically coordinates the operation of the entire computer system, although the control unit does not performed any actual processing on the data, it acts as a central nervous system uses to sent control signal to other units.

*Arithmetic and Logic Unit (ALU)

Perform the processing of data including arithmetic operation such as addition, subtraction, multiplication, division and logic operation including comparison (ex. A<B) and sorting.

*Register

Register are devices capable of storing information, receiving data from other areas within the computer and transferring information as directed by the control unit, it is used for temporary storage of data or instruction and the most important register are :

1- Program counter (PC): it contains the address of the next instruction to be executed.
2- Instruction register (IR): it contains the instruction being executed.
3-Address register (AR) : holds the address of memory location.

2-Computer Memory

There are two basic categories of memory:

A)Primary storage (main memory): The memory is the part of the computer that holds information (data and instruction) for processing so name because small amounts of data and information that will be immediately used by the CPU are stored there.

The specific functions of main memory are to hold (store):

1- All data to be processed.
2- Intermediate result of processing.
3-Final result of processing.

B-Secondary Storage: where much larger amount of data and information (an entire software program, for example) are stored for extended period of time.

- Memory Capacity

Bit: All computers work on a binary numbering system, i.e. they process data in ones or zeros. This 1 or 0 level of storage is called a bit. Often hardware is specified as a 32-bit computer, which means that the hardware can process 32 bits at a time. Software is also described as 16 bit, 32 bit or 64 bit software.

CPU process only 0s and 1s, all data are translated through computer languages into series of these binary digits, or bits.

Eight bits are needed to represent a character. This 8-bit string is known as a byte. The storage capacity of a computer is measured in bytes. The hierarchy of byte memory capacity is as follows:

1- Byte: A byte consists of eight bits.
2- Kilobyte: A kilobyte (KB) consists of 1024 bytes.

3- Megabyte: A megabyte (MB) consists of 1024 kilobytes, (1024*1024) byte or 1,048,576 byte) approximately 1,000,000 bytes.

4- Gigabyte: A gigabyte (GB) consists of 1024 megabytes, (1024*1024*10240byte) or (1,073,741,824 byte), approximately 1,000,000,000 bytes.

5- Terabyte: A terabyte (TB) consists of approximately 1,000,000,000,000 bytes.

A: There are four main types primary (main) memory:

1-**Registers:** are part of CPU, they have the least capacity, storing limited amounts of instructions and data only immediately before and after processing.

2-Random Access Memory (RAM): it stores more information than registers and is farther away from the CPU, but it stores less than secondary storage and is much closer to the CPU than is the secondary storage. When you start most software programs on your computer, the entire program is brought from secondary storage into RAM.

As you use the program, small parts of the programs instructions and data are sent into the instructions as close to the CPU. Again, getting the data and instructions as close to the CPU as possible is key to the computer's speed, as is the fact that the RAM is a type of microprocessor chip. As we shall discuss later, the chip is much faster (and more costly) than are secondary storage devices.

3-Cashe Memory: many modern computer applications (Microsoft office 98, for example) are very complex and have huge numbers of instructions .it takes considerable RAM capacity (usually a minimum 16MB) to store the entire instruction set. Or you may be using an application that exceeds your RAM. in that case, your computer has to go into secondary storage to retrieve the instruction. to alleviate this problem, software is often written in smaller blocks of instruction. As need, these blocks can be brought from secondary storage into RAM; this is still slow however, cashe memory is the place closer to the CPU where the computer can temporarily store those blocks used most often. Those used less often remain in RAM until they are transferred to cache; those used infrequently stay stored in secondary storage. Cashe memory is faster than RAM. Because, the instructions travel a shorter distance to the CPU.

4-Read Only Memory (ROM)

Most people who use computers have lost precious data at one time due to a" computer crash" or power failure. What is usually lost is whatever is in RAM, cashe, or the registers, this loss occurs because these types of memory is volatile. Read-only-memory (ROM) is the place (a type of chip) where certain critical instructions are safeguarded.ROM is nonvolatile and retains these instructions when the power to the computer is turned off. The (read only) means that these instructions can be read only by the computer and cannot be changed by the user.

B) Secondary Storage (Backing Storage)

Secondary storage is designed to store very large amounts of data for extended periods of time .secondary storage can have memory capacity of gigabyte or more; only small portions of the data are placed in primary storage at any one time.

Secondary storage **has the following Characteristics:**

1-it is nonvolatile

2-it takes much more time to retrieve data from secondary storage than it does from RAM

3-it much more cost effective than primary storage

4-it can take place on a variety of media each with its own technology, as is cussed below:

a) Magnetic tape
b) Magnetic disc
c) Magnetic diskette (floppy disc)
d) Optical discs