Lecture One
Number systems and operations

The study of number systems is important from the viewpoint of understanding how data are represented before they can be processed by any digital system including a digital computer. It is one of the most basic topics in digital electronics. In this chapter, we will discuss different number systems commonly used to represent data. We will begin the discussion with the decimal number system. Although it is not important from the viewpoint of digital electronics, a brief outline of this will be given to explain some of the underlying concepts used in other number systems. This will then be followed by the more commonly used number systems such as the binary, octal and hexadecimal number systems.

1. Decimal numbers:

The decimal number system is a radix-10 number system and therefore has 10 different digits or symbols. These are 0, 1, 2, 3, 4, 5, 6, 7, 8 and 9. All higher numbers after ‘9’ are represented in terms of these 10 digits only. The process of writing higher-order numbers after ‘9’ consists in writing the second digit (i.e. ‘1’) first, followed by the other digits, one by one, to obtain the next 10 numbers from ‘10’ to ‘19’. The next 10 numbers from ‘20’ to ‘29’ are obtained by writing the third digit (i.e. ‘2’) first, followed by digits ‘0’ to ‘9’, one by one.

The place values of different digits in a mixed decimal number, starting from the decimal point, are 10^0, 10^1, 10^2 and so on (for the integer part) and 10^{-1}, 10^{-2}, 10^{-3} and so on (for the fractional part).

As an illustration, in the case of the decimal number 3586.265, the integer part 3586 can be expressed as

$$3586 = 6\times10^{0} + 8\times10^{1} + 5\times10^{2} + 3\times10^{3} = 6+80+500+3000 = 3586$$

and the fractional part 265 can be expressed as

$$265 = 2\times10^{-1} + 6\times10^{-2} + 5\times10^{-3} = 0.2+0.06+0.005 = 0.265$$

2. Binary Numbers:

The binary number system its two digits a base-two system. The two binary digits are 1 and 0 (1,0).

Binary weight	2^3	2^2	2^1	2^0
Weight value	8	4	2	1

A binary digit, called a bit, has two values 0 & 1. Each coefficient **a^j is multiplied by 2^j**, and the results are added to obtain the decimal equivalent of the number. For example,

11010.11 is equal to 26.75 as follows:

$$1\times2^{4} + 1\times2^{3} + 0\times2^{2} + 1\times2^{1} + 0\times2^{0} + 1\times2^{-1} + 1\times2^{-2}$$

$$16 + 8 + 0 + 2 + 0 + 0.5 + 0.25 = (26.75)_{10}$$

In general, a number expressed in a base-r system has coefficients multiplied by powers of **r**.

$$r^{n}\times a_{n} + r^{n-1}\times a_{n-1} + \ldots\ldots + r^{2}\times a_{2} + r^{1}\times a_{1} + 1\times a_{0} + r^{-1}\times a_{-1} + r^{-2}\times a_{-2} + \ldots\ldots + r^{-m}\times a_{-m}$$

Table 1

Decimal	***Binary (0,1)***			
	8	***4***	***2***	***1***
	2^3	2^2	2^1	2^0
0	*0*	*0*	*0*	*0*
1	*0*	*0*	*0*	*1*
2	*0*	*0*	*1*	*0*
3	*0*	*0*	*1*	*1*
4	*0*	*1*	*0*	*0*
5	*0*	*1*	*0*	*1*
6	*0*	*1*	*1*	*0*
7	*0*	*1*	*1*	*1*
8	*1*	*0*	*0*	*0*
9	*1*	*0*	*0*	*1*
10	*1*	*0*	*1*	*0*
11	*1*	*0*	*1*	*1*
12	*1*	*1*	*0*	*0*
13	*1*	*1*	*0*	*1*
14	*1*	*1*	*1*	*0*
15	*1*	*1*	*1*	*1*

LSB (right-most bit) has a weight of $2^0 = 1$.

MSB (left- most bit) has a weight of $2^3 = 8$.

3. Octal numbers:

The octal number system has a radix of 8 and therefore has eight distinct digits. All higher-order numbers are expressed as a combination of these on the same pattern as the one followed in the case of the binary and decimal number systems described above.

Table 2

Decimal	***Octal***
0	*0*
1	*1*
2	*2*
3	*3*
4	*4*
5	*5*
6	*6*
7	*7*
8	*10*
9	*11*
10	*12*
11	*13*
12	*14*
13	*15*
14	*16*
15	*17*

3. Hexadecimal Numbers:

The hexadecimal number system is a radix-16 number system and its 16 basic digits are shown below.

Table 3

Decimal	*Hexadecimal*
0	*0*
1	*1*
2	*2*
3	*3*
4	*4*
5	*5*
6	*6*
7	*7*
8	*8*
9	*9*
10	*A*
11	*B*
12	*C*
13	*D*
14	*E*
15	*F*

Number Systems Conversions:

1- Binary - to - Decimal Conversion:

The decimal value of any binary number can be found by adding the weights of all bits that are 1 and discarding the weights of all bits that are 0.

EX1. The decimal equivalent of the binary number $(1001.0101)_2$ is determined as follows:

- The integer part = 1001
- The decimal equivalent= $1 \times 2^0 + 0 \times 2^1 + 0 \times 2^2 + 1 \times 2^3 = 1 + 0 + 0 + 8 = 9$
- The fractional part =0 .0101
- Therefore, the decimal equivalent = $0 \times 2^{-1} + 1 \times 2^{-2} + 0 \times 2^{-3} + 1\times 2^{-4} = 0 + 0.25 + 0 + 0.0625 = 0.3125$
- Therefore, the decimal equivalent of $(1001.0101)_2$=$(9.3125)_{10}$

2- Octal - to - Decimal Conversion:

The decimal equivalent of the octal number $(137.21)_8$ is determined as follows:

- The integer part = 137
- The decimal equivalent = $7 \times 8^0 + 3 \times 8^1 + 1 \times 8^2 = 7 + 24 + 64 = 95$
- The decimal equivalent = $2 \times 8^{-1} + 1 \times 8^{-2} = 0.265$
- Therefore, the decimal equivalent of $(137.21)_8$= $(95.265)_{10}$

3- Hexadecimal - to - Decimal Conversion:

The decimal equivalent of the hexadecimal number $(1E0.2A)_{16}$ is determined as follows:

- The integer part = 1E0
- The decimal equivalent = $0 \times 16^0 + 14 \times 16^1 + 1 \times 16^2 = 0 + 224 + 256 = 480$
- The fractional part = 2A
- The decimal equivalent = $2 \times 16^{-1} + 10 \times 16^{-2} = 0.164$
- Therefore, the decimal equivalent of $(1E0.2A)_{16} = (480.164)_{10}$

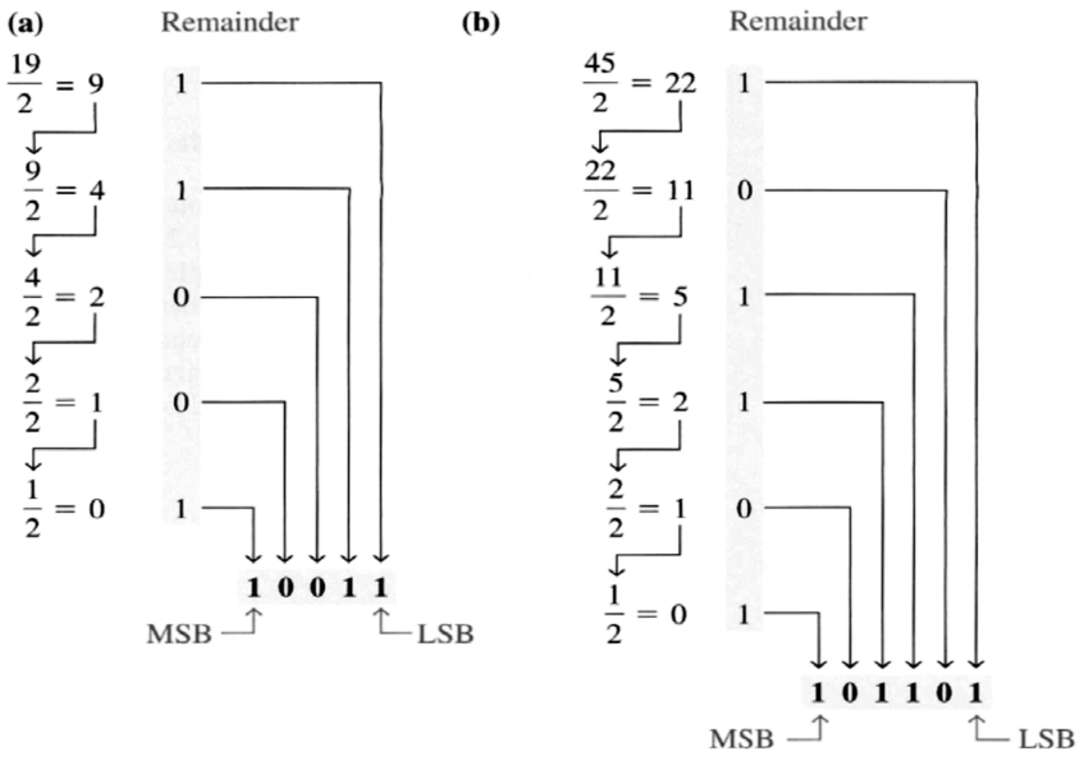
4- Decimal-to-Binary Conversion:

As outlined earlier, the integer and fractional parts are worked on separately. For the integer part, the binary equivalent can be found by successively dividing the integer part of the number by 2 and recording the remainders until the quotient becomes ‘0’. The remainders written in reverse order constitute the binary equivalent. For the fractional part, it is found by successively multiplying the fractional part of the decimal number by 2 and recording the carry until the result of multiplication is ‘0’. The carry sequence written in forward order constitutes the binary equivalent of the fractional part of the decimal number. If the result of multiplication does not seem to be heading towards zero in the case of the fractional part, the process may be continued only until the requisite number of equivalent bits has been obtained.

This method of decimal–binary conversion is popularly known as the double-dabble method. The process can be best illustrated with the help of a division process as explained in the following example:

EX2. Convert the following numbers from decimal to binary:

(a) 19 (b) 45



EX3. Convert $(0.6875)_{10}$ to binary:

	Integer	**Fraction**	**Coefficient**
0.6875*2	1+	0.375	1
0.375*2	0+	0.75	0
0.75*2	1+	0.5	1
0.5*2	1+	0	1

$(0.6875)_{10} = (0.1011)_2$

5- Decimal - to - Octal Conversion:

The process of decimal-to-octal conversion is similar to that of decimal-to-binary conversion. The progressive division in the case of the integer part and the progressive multiplication while working on the fractional part here are by ‘8’ which is the radix of the octal number system. Again, the integer and fractional parts of the decimal number are treated separately. The process can be best illustrated with the help of an example.

EX4. Convert $(73.75)_{10}$ to octal

1. **Integer part**

Dividend	**Remainder**
73/8	1
9/8	1
1/8	1

$(73)_{10} = (111)_8$

2. **Fractional**

	Integer	**Fraction**	**Coefficient**
0.75*8	6+	0	6

$(73.75)_{10} = (111.6)_8$

6- Decimal - to - Hexadecimal Conversion:

The process of decimal-to-hexadecimal conversion is also similar. Since the hexadecimal number system has a base of 16, the progressive division and multiplication factor in this case is 16. The process is illustrated further with the help of an example.

EX4. Let us determine the hexadecimal equivalent of $(82.25)_{10}$

1. **Integer part**

Dividend	**Remainder**
82/16	**2**
5/16	5

2. **Fractional part**

	Integer	**Fraction**	**Coefficient**
0.25*16	4+	0	4

Therefore, the hexadecimal equivalent of $(82.25)_{10} = (52.4)_{16}$

Binary–Octal and Octal–Binary Conversions:

An octal number can be converted into its binary equivalent by replacing each octal digit with its three-bit binary equivalent. We take the three-bit equivalent because the base of the octal number system is 8 and it is the third power of the base of the binary number system, i.e. 2. All we have then to remember is the three-bit binary equivalents of the basic digits of the octal number system. A binary number can be converted into an equivalent octal number by splitting the integer and fractional parts into groups of three bits, starting from the binary point on both sides. The 0s can be added to complete the outside groups if needed.

<u>EX5.</u> Let us find the binary equivalent of $(374.26)_8$ and the octal equivalent of $(1110100.0100111)_2$

1- $(374.26)_8 = (?)_2$

Octal	**3**	**7**	**4**	**2**	**6**
Binary	**011**	**111**	**100**	**010**	**110**

$(374.26)_8 = (011111100.010110)_2$

2- $(1110100.0100111)_2 = (?)_8$

$(001\ \ 110\ \ 100 . 010\ \ 011\ \ 100)_2$

Binary	**001**	**110**	**100**	**010**	**011**	**100**
Octal	**1**	**6**	**4**	**2**	**3**	**4**

$(1110100.0100111)_2 = (164.234)_8$

Hex–Binary and Binary–Hex Conversions:

A hexadecimal number can be converted into its binary equivalent by replacing each hex digit with its four-bit binary equivalent. We take the four-bit equivalent because the base of the hexadecimal number system is 16 and it is the fourth power of the base of the binary number system. All we have then to remember is the four-bit binary equivalents of the basic digits of the hexadecimal number system. A given binary number can be converted into an equivalent hexadecimal number by splitting the integer and fractional parts into groups of four bits, starting from the binary point on both sides. The 0s can be added to complete the outside groups if needed.

EX6. Let us find the binary equivalent of $(17E.F6)_{16}$ and the hex equivalent of $(1011001110.011011101)_2$.

1- $(17E.F6)_{16} = (?)_2$

Hex	**1**	**7**	**E**	**F**	**6**
Binary	**0001**	**0111**	**1110**	**1111**	**0110**

$(17E.F6)_{16} =(101111110.1111011)_2$

2- $(1011001110.011011101)_2 = (?)_{16}$

$(\ 0010\ \ 1100\ \ \ 1110\ .\ \ 0110\ \ \ 1110\ \ 1000\ )_2$

Binary	**0010**	**1100**	**1110**	**0110**	**1110**	**1000**
Hex	**2**	**C**	**E**	**6**	**E**	**8**

$(1011001110.011011101)_2.=(2CE.6E8)_{16}$

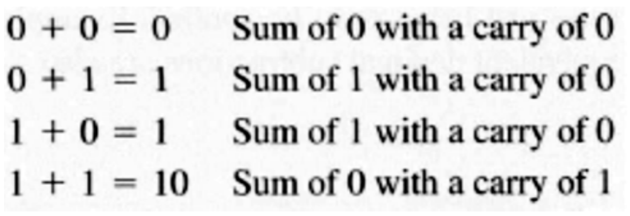
H.W. **Find the octal equivalent of $(2F.C4)_{16}$ and the hex equivalent of $(762.013)_8$?**

Table 4

Decimal	Binary	Octal	Hexadecimal
0	0000	0	0
1	0001	1	1
2	0010	2	2
3	0011	3	3
4	0100	4	4
5	0101	5	5
6	0110	6	6
7	0111	7	7
8	1000	10	8
9	1001	11	9
10	1010	12	A
11	1011	13	B
12	1100	14	C
13	1101	15	D
14	1110	16	E
15	1111	17	F

Binary Arithmetic:

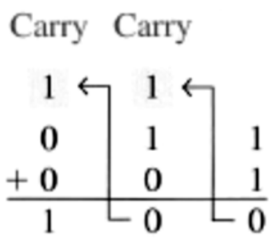
1- Binary Addition



0 + 0 = 0	Sum of 0 with a carry of 0
0 + 1 = 1	Sum of 1 with a carry of 0
1 + 0 = 1	Sum of 1 with a carry of 0
1 + 1 = 10	Sum of 0 with a carry of 1

***Ex.* Find:**

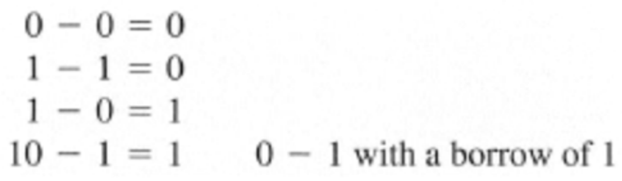
1) $(11+01)_2$=?



2) $(110101+110111)_2$=?

```
  1 1 0 1 0 1
+
  1 1 0 1 1 1
1 1 0 1 1 0 0
```

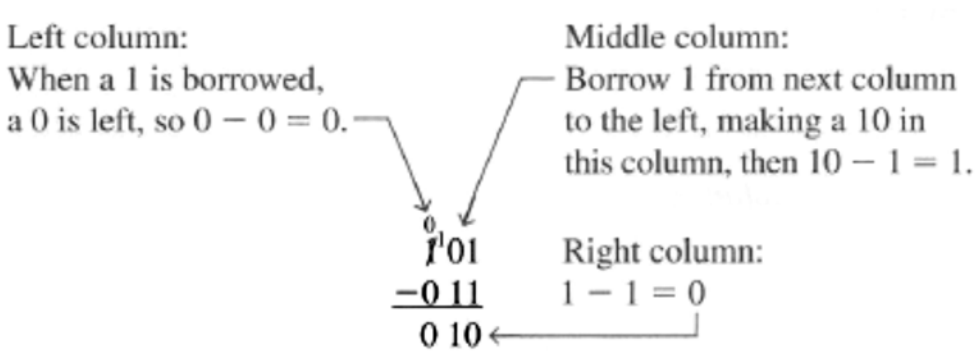
2- Binary Subtraction



0 − 0 = 0	
1 − 1 = 0	
1 − 0 = 1	
10 − 1 = 1	0 − 1 with a borrow of 1

***EX.* Find**

1) $(101-11)_2$=?


Left column:
When a 1 is borrowed,
a 0 is left, so 0 − 0 = 0.
Middle column:
Borrow 1 from next column
to the left, making a 10 in
this column, then 10 − 1 = 1.
0
1 1 01
−0 11
0 10
Right column:
1 − 1 = 0

2) $(111010-11001)_2$=?

```
  1 1 1 0 1 0
-
    1 1 0 0 1
  ___________
  1 0 0 0 0 1
```

3) $(110101-101110)_2$=?

```
  1 1 0 1 0 1
-
  1 0 1 1 1 0
  ___________
  0 0 0 1 1 1
```

4) $(1101-110111)_2$=?

```
    1 1 0 1 1 1
  -
          1 1 0 1
    _____________
-   1 0 1 0 1 0
```

3- Binary Multiplication

$$0 \times 0 = 0$$
$$0 \times 1 = 0$$
$$1 \times 0 = 0$$
$$1 \times 1 = 1$$

***EX*. Find**

1) $(11 \text{x} 11)_2$=?

$$\begin{array}{lr}
 & 11 \\
 & \underline{\times\, 11} \\
\text{Partial} \left\{ \begin{array}{l} \\ \\ \end{array} \right. & \begin{array}{r} 11 \\ \underline{+11\;\;} \end{array} \\
\text{products} & \mathbf{1001}
\end{array}$$

2) $(110101 \text{x} 11)_2$=?

$$\begin{array}{rr}
 & 1\,1\,0\,1\,0\,1 \\
\times & 1\,1 \\
\hline
 & 1\,1\,0\,1\,0\,1 \\
+ & 1\,1\,0\,1\,0\,1\;\; \\
\hline
 & 1\,0\,0\,1\,1\,1\,1\,1
\end{array}$$

3) $(10011 \text{x} 101)_2$=?

$$\begin{array}{rr}
 & 1\,0\,0\,1\,1 \\
\times & 1\,0\,1 \\
\hline
 & 1\,0\,0\,1\,1 \\
+ & 0\,0\,0\,0\,0\;\; \\
+ & 1\,0\,0\,1\,1\;\;\;\; \\
\hline
 & 1\,0\,1\,1\,1\,1\,1
\end{array}$$

4-Binary Division

This operation follows the same procedure as division in decimal number system.

EX. Find:

1) (1 1 0 ÷11) =?

$$110 \div 11 = 10$$

$$\begin{array}{r|l} & 10 \\ \cline{2-2} 11 & 110 \\ & \underline{-11} \\ & 000 \end{array}$$

2) 1 0 1 0 1 ÷ 1 1 = ?

$$\begin{array}{r|r} & 111 \\ \cline{2-2} 11 & 10101 \\ & -\ 11 \\ \cline{2-2} & 100 \\ & -\ 11 \\ \cline{2-2} & 0011 \\ & -\ 11 \\ \cline{2-2} & 00 \end{array}$$

$\therefore$ **1 0 1 0 1 ÷ 1 1** = 1 1 1

3) 1 0 1 1 0 1 ÷ 1 0 1 =?

$$\begin{array}{r|r} & 1001 \\ \cline{2-2} 101 & 101101 \\ & -\ 101 \\ \cline{2-2} & 000101 \\ & -\ 101 \\ \cline{2-2} & 000 \end{array}$$

$\therefore$ **1 0 1 1 0 1 ÷ 1 0 1** = 1 0 0 1

1’s and 2’s Complement of Binary Numbers

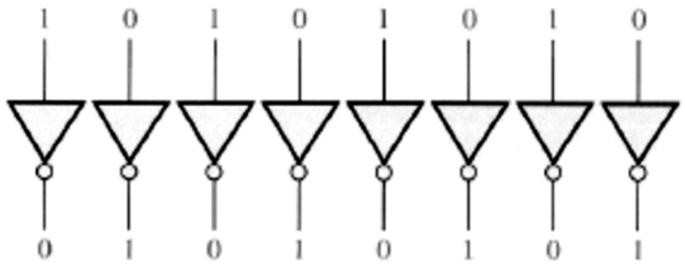
The l's complement and the 2's complement of a binary number are important because they permit the representation of negative numbers. The method of 2's complement arithmetic is commonly used in computers to handle negative numbers.

Finding the 1's Complement

The l's complement of a binary number is found by changing all 1s to 0s and all 0s to 1s, as illustrated below:

```
1 0 1 1 0 0 1 0     Binary number
↓ ↓ ↓ ↓ ↓ ↓ ↓ ↓
0 1 0 0 1 1 0 1     1’s complement
```

The simplest way to obtain the l's complement of a binary number with a digital circuit is to use parallel inverters (NOT circuits), as shown in Fig. below for an 8-bit binary number



Finding the 2' s Complement

The 2's complement of a binary number is found by adding 1 to the LSB of the l's complement.

2's complement = (l's complement) + 1

EX. Find the 2's complement of 10110010.

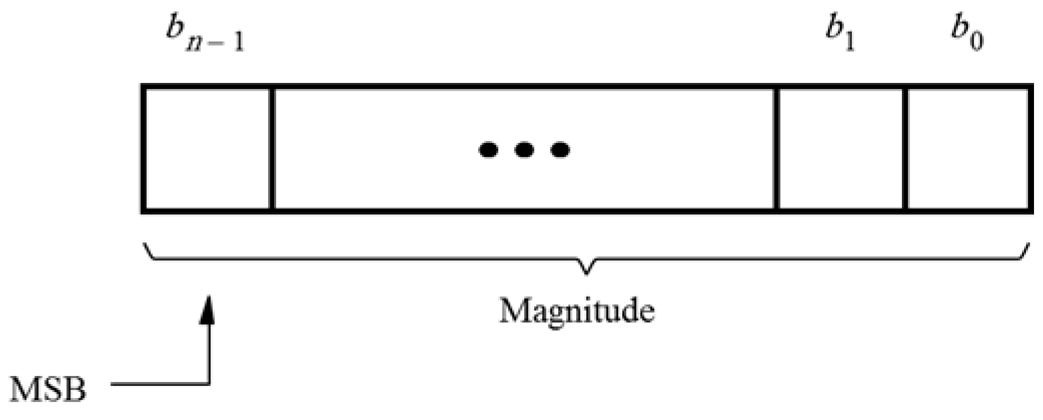
10110010	Binary number
01001101	l's complement
+ 1	add 1
01001110	2's complement

Unsigned and Signed Numbers:

a) Unsigned Numbers:

For an n-bit **unsigned** binary number, all n-bits are used to represent the **magnitude** of the number.

Note:- **Cannot represent negative numbers **



Unsigned Numbers

b) Signed Numbers

Digital systems, such as the computer, must be able to handle both positive and negative numbers. A signed binary number consists of both sign and magnitude information. The sign indicates whether a number is positive or negative, and the magnitude is the value of the number. There are three forms in which signed integer (whole) numbers can be represented in binary: sign magnitude, l's complement, and 2' complement. Of these, the 2's complement is the most important and the sign-magnitude is the least used

The Sign Bit

The left-most bit in a signed binary number is the sign bit, which tells you whether the number is positive or negative. A **0** sign bit indicates a **positive number**, and a **1** sign bit indicates a **negative** number.

A 0 sign bit indicates a positive number, and a 1 sign bit indicates a negative number

Sign-Magnitude Form

When a signed binary number is represented in sign-magnitude, the leftmost bit is the sign bit and the remaining bits are the magnitude bits. The magnitude bits are in true (un-complemented) binary for both positive and negative numbers. For example, the decimal number + 25 is expressed as an 8-bit signed binary number using the sign-magnitude form as 00011001.

The decimal number -25 is expressed as 1001100l.

Notice that the only difference between + 25 and - 25 is the sign bit because the magnitude bits are in true binary for both positive and negative numbers.

In the sign-magnitude form, a negative number has the same magnitude bits as the corresponding positive number but the sign bit is a 1 rather than a zero.

EX. Express the decimal number - 39 as an 8-bit number in the sign-magnitude, 1's complement, and 2's complement forms?

<u>**SOL//**</u> First, write the 8-bit number for + 39.

00100111

In the **sign-magnitude** form, - 39 is produced by changing the sign bit to a 1 and leaving the magnitude bits as they are. The number is

10100111

In the **1's complement** form, -39 is produced by taking the l's complement of +39 (00100111).

11011000

In the **2's complement** form, - 39 is produced by taking the 2's complement of +39 (00100111) as follows:

$$\begin{array}{rl} 11011000 & \quad \textbf{1's complement} \\ +\qquad\qquad 1 & \\ \hline 11011001 & \quad \textbf{2's complement} \end{array}$$

The Decimal Value of Signed Numbers

Sign-magnitude: Decimal values of positive and negative numbers in the sign-magnitude form are determined by summing the weights in all the magnitude bit positions where there are 1s and ignoring those positions where there are zeros. The sign is determined by examination of the sign bit.

EX.

Determine the decimal value of this signed binary number expressed in sign-magnitude:

10010101

SOL//

The seven magnitude bits and their powers-of-two weights are as follows:

2^6	2^5	2^4	2^3	2^2	2^1	2^0
0	0	1	0	1	0	1

Summing the weights where there are 1s,

16 + 4 + 1 = 21

The sign bit is 1; therefore, the decimal number is - 21.

1's Complement:

Decimal values of positive numbers in the l's complement form are determined by summing the weights in all bit positions where there are 1s and ignoring those positions where there are zeros. Decimal values of negative numbers are determined by assigning a negative value to the weight of the sign bit, summing all the weights where there are 1s, and adding 1 to the result.

Example:

Determine the decimal values of the signed binary numbers expressed in 1's complement: (a) 00 010 111 (b) 11 101 000

Solution:

(a) The bits and their powers-of-two weights for the positive number are as follows:

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

Summing the weights where there are 1s, $16 + 4 + 2 + 1 = +23$

(b) The bits and their powers-of-two weights for the negative number are as follows.

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

Notice that the negative sign bit has a weight of -2^7 or -128.

Summing the weights where there are 1s, $-128 + 64 + 32 + 8 = -24$

Adding 1 to the result, the final decimal number is $-24 + 1 = -23$.

2's Complement:

Decimal values of positive and negative numbers in the 2's complement form are determined by summing the weights in all bit positions where there are 1's and ignoring those positions where there are zeros. The weight of the sign bit in a negative number is given a negative value.

Example:

Determine the decimal values of the signed binary numbers expressed in 2's complement:

(a) 01010110 (b) 10101010

(a) The bits and their powers-of-two weights for the positive number are as follows:

$$\begin{array}{cccccccc} -2^7 & 2^6 & 2^5 & 2^4 & 2^3 & 2^2 & 2^1 & 2^0 \\ 0 & 1 & 0 & 1 & 0 & 1 & 1 & 0 \end{array}$$

Summing the weights where there are 1s,

$64 + 16 + 4 + 2 = +86$

(b) The bits and their powers-of-two weights for the negative number are as follows. Notice that the negative sign bit has a weight of $-2^7 = -128$.

$$\begin{array}{cccccccc} -2^7 & 2^6 & 2^5 & 2^4 & 2^3 & 2^2 & 2^1 & 2^0 \\ 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 \end{array}$$

Summing the weights where there are 1s, $-128 + 32 + 8 + 2 = -86$

Ex.Perform the following arithmetic operations in binary as an 8- bit using signed -2's-complement representation for negative numbers

a) (+6) + (+13) b) (-6) +(+13)

c) (+6) + (-13) d) (-6) + (-13)

e) (-6) - (-13)

Sol. ***a)***

$$\begin{array}{lc} +6 & 0\,0\,0\,0\,0\,1\,1\,0 \\ +13 & 0\,0\,0\,0\,1\,1\,0\,1 \\ \hline +19 & 0\,0\,0\,1\,0\,0\,1\,1 \end{array}$$

b) In the **2's complement** form, - 6 is produced by taking the 2's complement of +6 (00000110)

$$\begin{array}{rr} -6 & 11111010 \\ +13 & 00001101 \\ \hline +7 & 00000111 \end{array}$$

Removing the end carry

c) In the **2's complement** form, - 13 is produced by taking the 2's complement of +13 (00001101)

$$\begin{array}{rr} +6 & 00000110 \\ -13 & 11110011 \\ \hline -7 & 11111001 \end{array}$$

للتحقق من الناتج اما

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

-128+64+32+16+8+1=-7

او

The signed binary number 11111001 is negative because the leftmost bit is 1. Its 2’s complement is 00000111, which is the binary equivalent of *(*+7). We therefore recognize the original negative number to be equal to -7.

d)

$$\begin{array}{rr} -6 & 11111010 \\ -13 & 11110011 \\ \hline -19 & 11101101 \end{array}$$

A carry out of the sign-bit position is discarded

للتحقق من الناتج اما

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

-128+64+32+8+4+1=-19

او

The signed binary number 11101101 is negative because the leftmost bit is 1. Its 2’s complement is 00010011, which is the binary equivalent of *(*+19). We therefore recognize the original negative number to be equal to -19.

e) (-6) - (-13) The subtraction is changed to addition (-6) + (+13)

$$\begin{array}{rr} -6 & 1\,1\,1\,1\,1\,0\,1\,0 \\ +13 & 0\,0\,0\,0\,1\,1\,0\,1 \\ \hline +7 & 0\,0\,0\,0\,0\,1\,1\,1 \end{array}$$

Removing the end carry

Binary Coded Decimal (BCD)

Binary coded decimal (BCD) is a way to express each of the decimal digits with a binary code. There are only ten code groups in BCD system, so it is very easy to convert between decimal and BCD. Because we like to read and write in decimal, the interfaces are keypad input and digital readout.

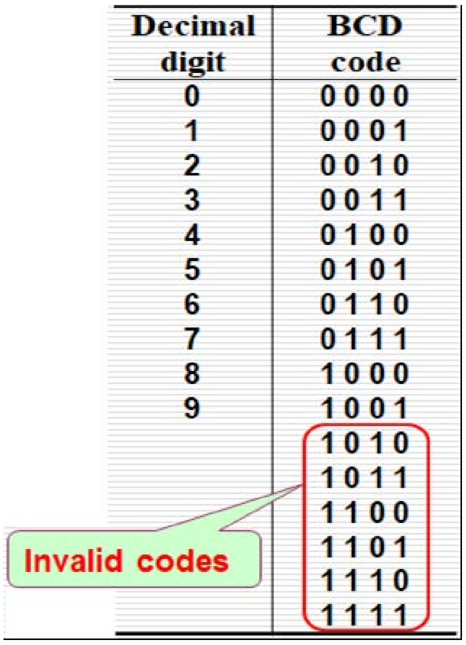
The 8421code:

The 8421 code is a type of BCD (binary coded decimal) code. Binary coded decimal digit, 0 thought 9, is represented by a binary code of four bits. The designation 8421 indicated the binary weights of the four bits **(2^3,2^2,2^1,2^0)**. The ease of conversion between 8421 code number and the familiar decimal numbers is the main advantage of this code. All you have to remember are the ten binary combinations that represent the ten decimal digits as shown in table below. The 8421 code is predominant BCD code, and when we refer to BCD, we always mean the 8421 code unless otherwise stated.

Binary decimal	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	9
code (BCD)	**0000**	**0001**	**0010**	**0011**	**0100**	**0101**	**0110**	**0111**	**1000**	**1001**

Invalid Codes:

It may be realized that, with four bits, sixteen numbers (0000 through 1111) can be represented but that, in the 8421 code, only ten of these are used. The six code combinations that are not used (**1010, 1011, 1100, 1101, 1110, 1111)** are invalid in 8421 BCD code. To express any decimal number in BCD, simply replace each decimal digit with the appropriate 4-bit code as shown by Example



Decimal digit	BCD code
0	0000
1	0001
2	0010
3	0011
4	0100
5	0101
6	0110
7	0111
8	1000
9	1001
	1010
	1011
	1100
	1101
	1110
	1111

Invalid codes

<u>**EX//**</u> Convert each of the following decimal numbers to BCD codes:

a) 35 b) 98 c) 170

<u>**Solution:**</u>

a) 35 b) 98 c) 170

$$\begin{array}{ccccccc} 3 & 5 & 9 & 8 & 1 & 7 & 0 \\ \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow \\ 0011 & 0101 & 1001 & 1000 & 0001 & 0111 & 0000 \end{array}$$

<u>**EX//**</u>

Convert each of the following BCD codes to decimal:

(a) 10000110 **(b)** 001101010001 **(c)** 1001010001110000

(a) $\underbrace{1000}_{\downarrow \atop \mathbf{8}}\underbrace{0110}_{\downarrow \atop \mathbf{6}}$ **(b)** $\underbrace{0011}_{\downarrow \atop \mathbf{3}}\underbrace{0101}_{\downarrow \atop \mathbf{5}}\underbrace{0001}_{\downarrow \atop \mathbf{1}}$ **(c)** $\underbrace{1001}_{\downarrow \atop \mathbf{9}}\underbrace{0100}_{\downarrow \atop \mathbf{4}}\underbrace{0111}_{\downarrow \atop \mathbf{7}}\underbrace{0000}_{\downarrow \atop \mathbf{0}}$

BCD Addition

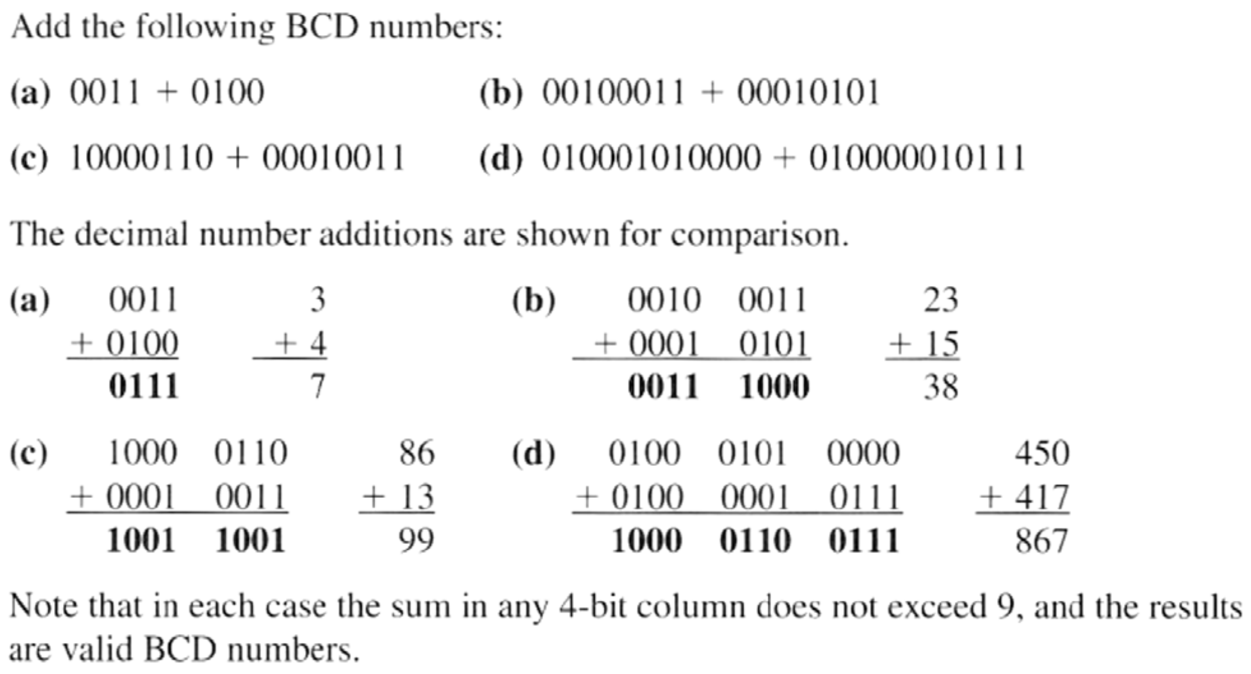
BCD is a numerical code and can be used in arithmetic operation. Addition is the most important operation because the other three operations (subtraction, multiplication, and deviation) can be accomplished by use of the addition. Here, is how to add two BCD numbers

1. Add the two BCD numbers, using the rules for binary addition.

2. If a 4-bit sum is equal to or less than 9, it is a valid BCD number.

3. If a 4-bit sum is greater than 9, or if a carry out of the four-bit group is generated, it is an invalid result. **Add 6(0110)** to 4-bit sum in order to skip the six invalid states and return the code to 8421. If a carry results when 6 is added, simply add the carry to the next 4-bit group.

Ex.



Add the following BCD numbers:

(a) 0011 + 0100 **(b)** 00100011 + 00010101

(c) 10000110 + 00010011 **(d)** 010001010000 + 010000010111

The decimal number additions are shown for comparison.

(a)
$$\begin{array}{r} 0011 \\ +\ 0100 \\ \hline \mathbf{0111} \end{array} \qquad \begin{array}{r} 3 \\ +\ 4 \\ \hline 7 \end{array}$$

(b)
$$\begin{array}{rr} 0010 & 0011 \\ +\ 0001 & 0101 \\ \hline \mathbf{0011} & \mathbf{1000} \end{array} \qquad \begin{array}{r} 23 \\ +\ 15 \\ \hline 38 \end{array}$$

(c)
$$\begin{array}{rr} 1000 & 0110 \\ +\ 0001 & 0011 \\ \hline \mathbf{1001} & \mathbf{1001} \end{array} \qquad \begin{array}{r} 86 \\ +\ 13 \\ \hline 99 \end{array}$$

(d)
$$\begin{array}{rrr} 0100 & 0101 & 0000 \\ +\ 0100 & 0001 & 0111 \\ \hline \mathbf{1000} & \mathbf{0110} & \mathbf{0111} \end{array} \qquad \begin{array}{r} 450 \\ +\ 417 \\ \hline 867 \end{array}$$

Note that in each case the sum in any 4-bit column does not exceed 9, and the results are valid BCD numbers.

Ex.

Add the following BCD numbers

(a) 1001 + 0100 **(b)** 1001 + 1001

(c) 00010110 + 00010101 **(d)** 01100111 + 01010011

The decimal number additions are shown for comparison.

(a)

$$\begin{array}{rrlr}
 & 1001 & & 9\\
 & \underline{+\,0100} & & \underline{+4}\\
 & 1101 & \text{Invalid BCD number } (>9) & 13\\
 & +\,0110 & \text{Add 6} & \\
\hline
\underbrace{\mathbf{0001}} & \underbrace{\mathbf{0011}} & \text{Valid BCD number} & \\
\downarrow & \downarrow & & \\
1 & 3 & &
\end{array}$$

(b)

$$\begin{array}{rrlr}
 & 1001 & & 9\\
 & \underline{+\,1001} & & \underline{+\,9}\\
1 & 0010 & \text{Invalid because of carry} & 18\\
 & +\,0110 & \text{Add 6} & \\
\hline
\underbrace{\mathbf{0001}} & \underbrace{\mathbf{1000}} & \text{Valid BCD number} & \\
\downarrow & \downarrow & & \\
1 & 8 & &
\end{array}$$

(c)

$$\begin{array}{rrlr}
0001 & 0110 & & 16\\
\underline{+\,0001} & \underline{0101} & & \underline{+\,15}\\
0010 & 1011 & \text{Right group is invalid } (>9), & 31\\
 & & \quad\text{left group is valid.} & \\
 & +\,0110 & \text{Add 6 to invalid code. Add} & \\
 & & \quad\text{carry, 0001, to next group.} & \\
\hline
\underbrace{\mathbf{0011}} & \underbrace{\mathbf{0001}} & \text{Valid BCD number} & \\
\downarrow & \downarrow & & \\
3 & 1 & &
\end{array}$$

(d)

$$\begin{array}{rrrlr}
 & 0110 & 0111 & & 67\\
 & \underline{+\,0101} & \underline{0011} & & \underline{+\,53}\\
 & 1011 & 1010 & \text{Both groups are invalid } (>9) & 120\\
 & +\,0110 & +\,0110 & \text{Add 6 to both groups} & \\
\hline
\underbrace{\mathbf{0001}} & \underbrace{\mathbf{0010}} & \underbrace{\mathbf{0000}} & \text{Valid BCD number} & \\
\downarrow & \downarrow & \downarrow & & \\
1 & 2 & 0 & &
\end{array}$$

Logic Gates

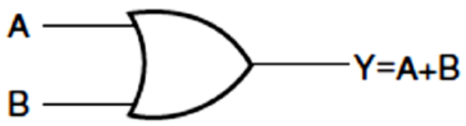
Logic gates are electronic circuits that can be used to implement the most elementary logic expressions, also known as Boolean expressions. The logic gate is the most basic building block of combinational logic. There are three **basic logic gates**, namely the **OR gate, the AND gate and the NOT gate**. Other logic gates that are **derived from these basic gates are the NAND gate, the NOR gate**, **the EXCLUSIVEOR gate and the EXCLUSIVE-NOR gate.**

1. OR Gate:

An OR gate performs an ORing operation on two or more than two logic variables. The OR operation on two independent logic variables A and B is written as Y = A+B and reads as Y equals A OR B and not as A plus B. An OR gate is a logic circuit with two or more inputs and one output. The output of an OR gate is LOW only when all of its inputs are LOW. For all other possible input combinations, the output is HIGH. This statement when interpreted for a positive logic system means the following. The output of an OR gate is a logic '0' only when all of its inputs are at logic '0'. For all other possible input combinations, the output is a logic '1'. Figure (1) shows the circuit symbol and the truth table of a two-input OR gate.

The operation of a two-input OR gate is explained by the logic expression:

Y = A+B



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

Fig. (1): the circuit symbol and the truth table of a two-input OR gate.

Now let us look at the operation of an OR gate with pulse waveform inputs, keeping in mind its logical operation. Again, the important thing in the analysis of gate operation with pulse waveforms is the time relationship of all the waveforms involved. For example, in Fig.(2), inputs A and B are both HIGH (1) during time interval t1 making output X HIGH (1). During time interval t2, input A is LOW (0), but because input B is HIGH (1), the output is HIGH (1). Both inputs are LOW (0) during time interval t3 , so there is a LOW (0) output during this time. During time interval t4 , the output is HIGH (1) because input A is HIGH (1).

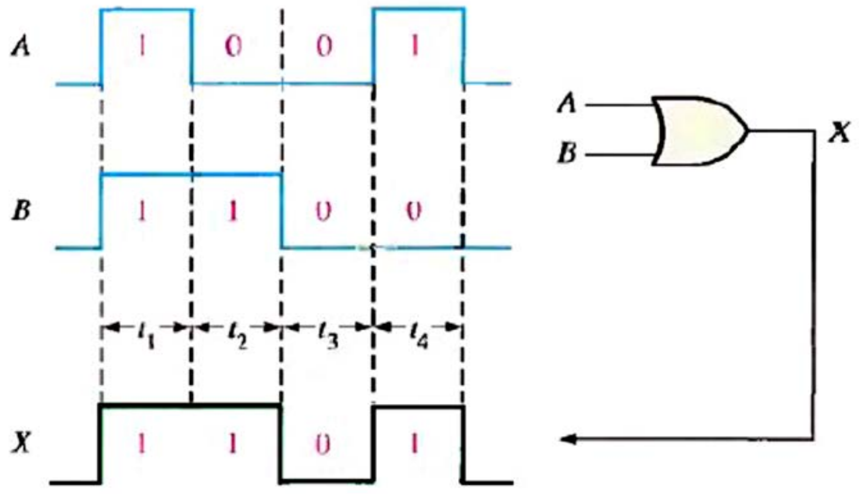


Fig. (2): Example of OR gate operation with a timing diagram showing input and output relationships.

As an illustration, if we have four logic variables and we want to know the logical output of (A+B+C +D), then it would be the output of a four-input OR gate with A, B, C and D as its inputs.

Figures 3(a) and (b) show the circuit symbol of three-input and four-input OR gates. Figure 3(c) shows the truth table of a three-input OR gate. Logic expressions explaining the functioning of three input and four-input OR gates are **Y = A+B+C** and **Y = A+B+C +D**

The total number of possible combinations of binary inputs to a gate is determined by the following formula:

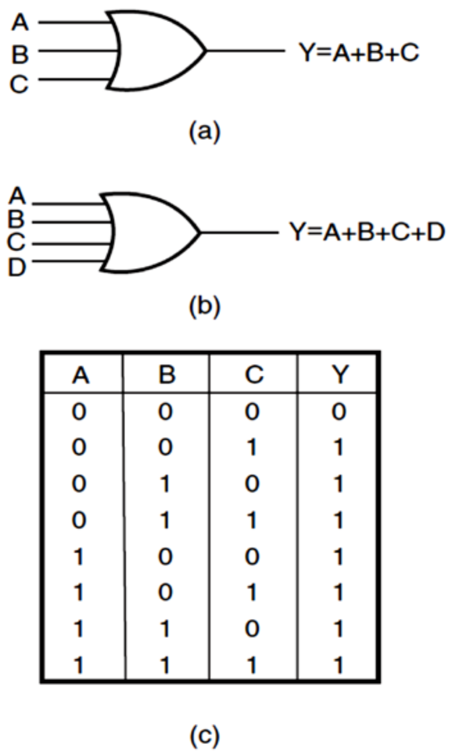
$$N=2^n$$

where N is the number of possible input combinations and n is the number of input variables. To illustrate:

For two input variables: $N = 2^2 = 4$ combinations.

For three input variables: $N = 2^3 = 8$ combinations.

For four input variables: $N = 2^4 = 16$ combinations.



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

Fig. (3)

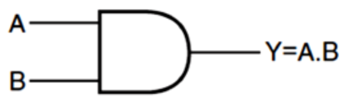
2. AND Gate:

An AND gate is a logic circuit having two or more inputs and one output. The output of an AND gate is HIGH only when all of its inputs are in the HIGH state. In all other cases, the output is LOW. When interpreted for a positive logic system, this means that the output of the AND gate is a logic ‘1’ only when all of its inputs are in logic ‘1’ state. In all other cases, the output is logic ‘0’.

The AND operation on two independent logic variables A and B is written as:

$$Y = A.B$$

and reads as **Y** equals **A AND B** and not as **A multiplied by B**. Here, A and B are input logic variables and Y is the output. An AND gate performs an ANDing operation



(a)

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

(b)

Fig. (4): AND gate.

Let's examine the waveform operation of an AND gate by looking at the inputs with respect to each other in order to determine the output level at any given time. In Fig.(5), inputs A and B are both HIGH (1) during the time interval, t1 making output X HIGH (1) during this interval. During time interval t2 input A is LOW (0) and input B is HIGH (1), so the output is LOW (0). During time interval t3 , both inputs are HIGH (1), and therefore the output is HIGH (1). During time interval t4 , input A is HIGH 0) and input B is LOW (0), resulting in a LOW (0) output. Finally, during time interval t5 , input A is LOW (0), input B is LOW (0), and the output is therefore LOW (0). As you know, a diagram of input and output waveforms showing time relationships is called a timing diagram.

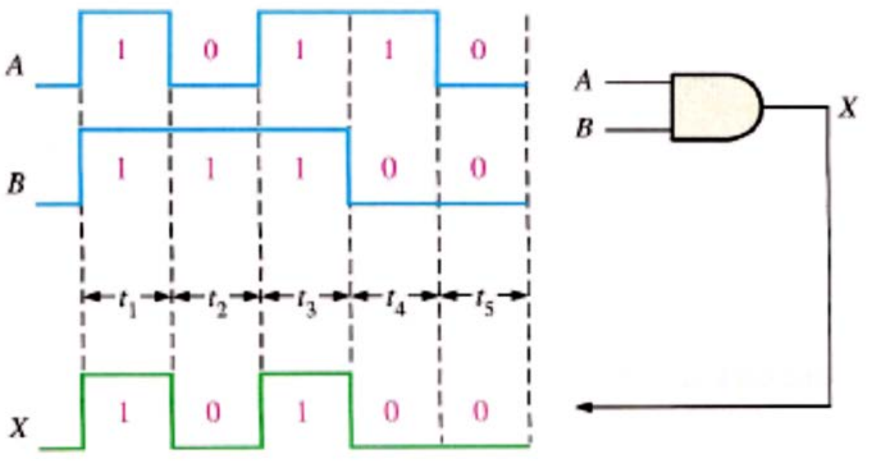
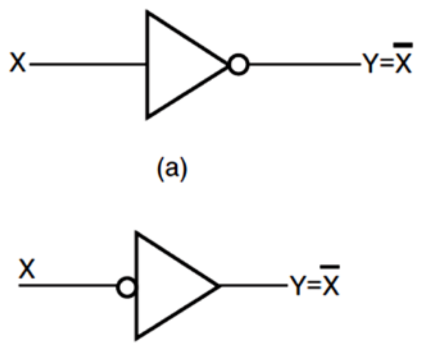


Fig. (5)

3. NOT Gate:

A NOT gate is a one-input, one-output logic circuit whose output is always the complement of the input. That is, a LOW input produces a HIGH output, and vice versa. When interpreted for a positive logic system, a logic ‘0’ at the input produces a logic ‘1’ at the output, and vice versa. It is also known as a ‘complementing circuit’ or an ‘inverting circuit’. Figure 6 shows the circuit symbol and the truth table.



X	Y
0	1
1	0

Fig. (6): Circuit symbol and the truth table of a NOT circuit.

The operation of an inverter (NOT circuit) can be expressed as follows:
If the input variable is called A and the output variable is called X, then

$$X = \overline{A}$$

This expression states that the output is the complement of the input, so if A= 0, then X = 1, and if A = 1, then X = 0.

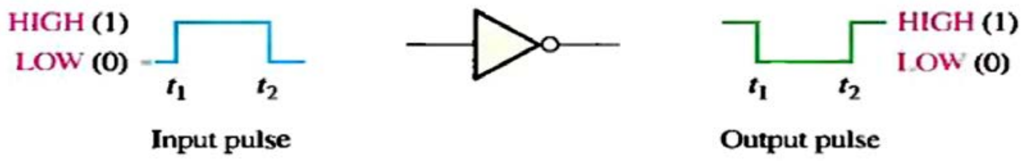


Fig. (7): Inverter operation.

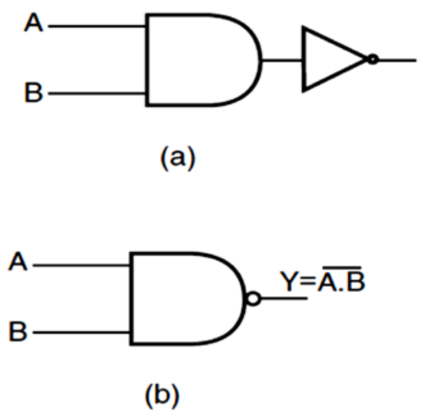
4. NAND Gate:

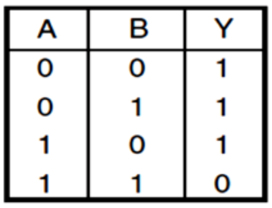
NAND stands for NOT AND. An AND gate followed by a NOT circuit makes it a NAND gate [Fig.8 (a)]. Fig.8 (b) shows the circuit symbol of a two-input NAND gate. The truth table of a NAND gate is obtained from the truth table of an AND gate by complementing the output entries [Fig.8(c)]. The output of a NAND gate is a logic '0' when all its inputs are a logic '1'. For all other input combinations, the output is a logic '1'. NAND gate operation is logically expressed as:

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

In general, the Boolean expression for a NAND gate with more than two inputs can be written as:

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





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

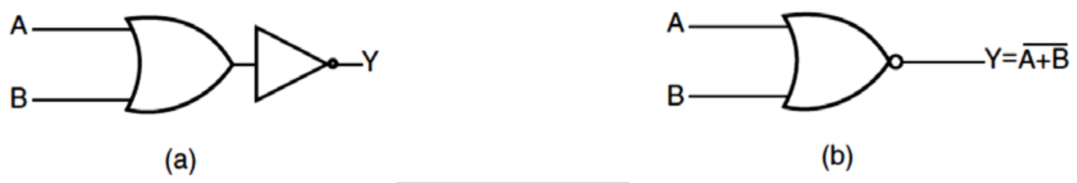
(c)

Fig. (8): (a) Two-input NAND implementation using an AND gate and a NOT circuit, (b) the circuit symbol of a two-input NAND gate, and (c) the truth table of a two-input NAND gate.

5. NOR Gate:

NOR stands for NOT OR. An OR gate followed by a NOT circuit makes it a NOR gate [Fig. 9(a)]. The truth table of a NOR gate is obtained from the truth table of an OR gate by complementing the output entries. The output of a NOR gate is a logic ‘1’ when all its inputs are logic ‘0’. For all other input combinations, the output is a logic ‘0’. The output of a two-input NOR gate is logically expressed as:

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



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

(c)

Fig. (9): (a) Two-input NOR implementation using an OR gate and a NOT circuit, (b) the circuit symbol of a two-input NOR gate and (c) the truth table of a two-input NOR gate.

In general, the Boolean expression for a NOR gate with more than two inputs can be written as:

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

THE EXCLUSIVE-OR AND EXCLUSIVE-NOR:

Exclusive-OR and exclusive-NOR gates are formed by a combination of other gates already discussed. However, because of their fundamental importance in many applications, these gates are often treated as basic logic elements with their own unique symbols.

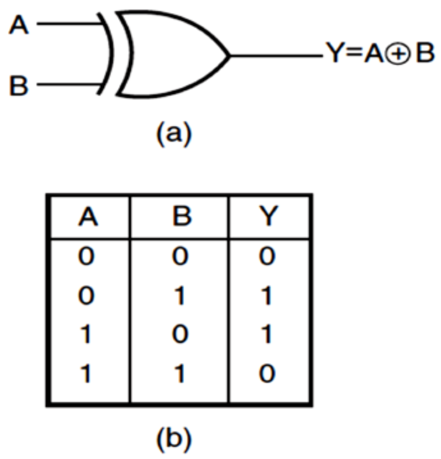
6. The Exclusive-OR Gate:

Standard symbol for an exclusive-OR (XOR for short) gate is shown in Fig. (10). <u>The XOR gate has only two inputs</u>.

For an exclusive-OR gate, output X is HIGH when input A is LOW and input B is HIGH, or when input A is HIGH and input B is LOW: X is LOW when A and B are both HIGH or both LOW.

The output of a two-input EX-OR gate is expressed as:

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



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

Fig. (10): X-OR gate.

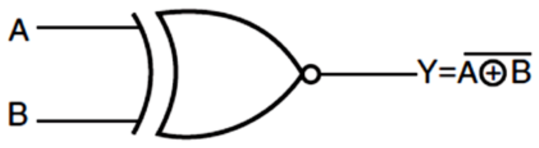
7. The Exclusive-NOR Gate:

Standard symbols for an exclusive-NOR (XNOR) gate are shown in Fig.(11). Like the XOR gate, an XNOR has only two inputs. The bubble on the output of the XNOR symbol indicates that its output is opposite that of the XOR gate. When the two input logic levels are opposite, the output of the exclusive-NOR gate is LOW. The operation can be stated as follows (A and B are inputs, X is the output):

For an exclusive-NOR gate, output X is LOW when input A is LOW and input B is HIGH, or when A is HIGH and B is LOW; X is HIGH when A and B are both HIGH or both LOW.

The truth table of an EX-NOR gate is obtained from the truth table of an EX-OR gate by complementing the output entries. Logically,

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



(a)

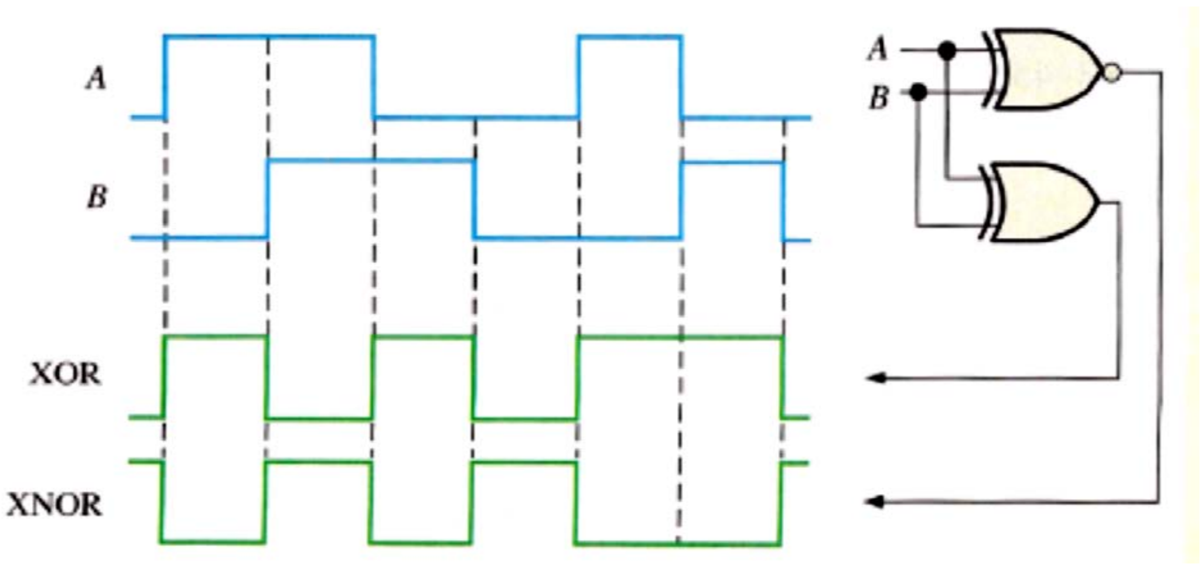
A	B	Y
0	0	1
0	1	0
1	0	0
1	1	1

(b)

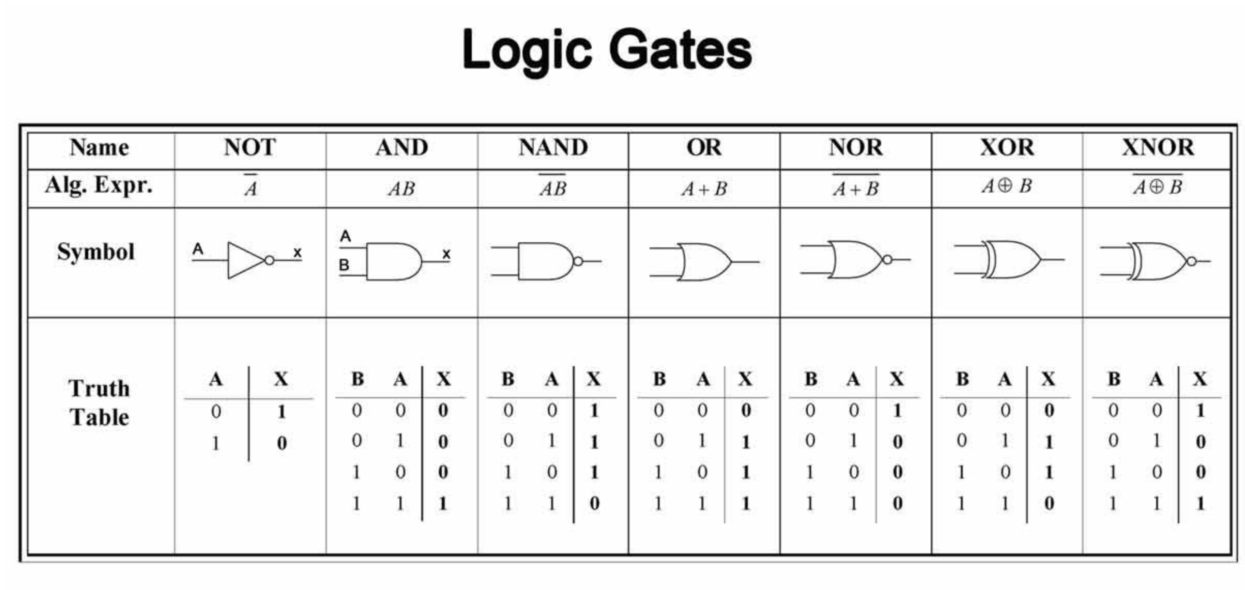
Fig. (11): XNOR gate.

Example:

Determine the output waveforms for the XOR gate and for the XNOR gate, given the input waveforms, A and B, in Figure below:



Summary

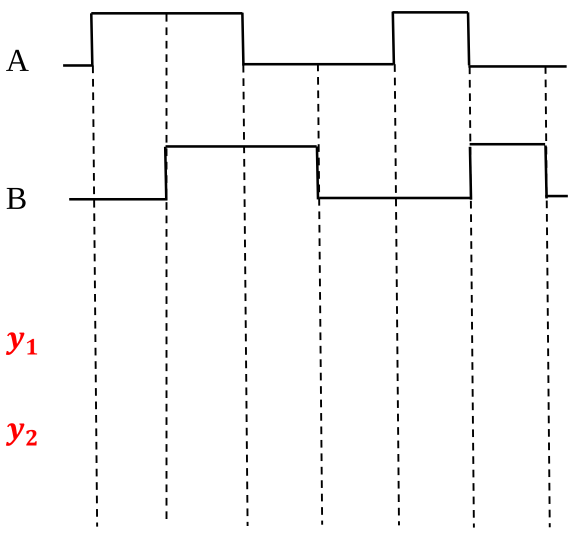


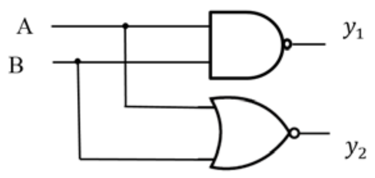
Logic Gates

Name	NOT	AND	NAND	OR	NOR	XOR	XNOR
Alg. Expr.	$\overline{A}$	AB	$\overline{AB}$	$A+B$	$\overline{A+B}$	$A\oplus B$	$\overline{A\oplus B}$
Symbol	A x	A B x					
Truth Table	A X 0 1 1 0	B A X 0 0 0 0 1 0 1 0 0 1 1 1	B A X 0 0 1 0 1 1 1 0 1 1 1 0	B A X 0 0 0 0 1 1 1 0 1 1 1 1	B A X 0 0 1 0 1 0 1 0 0 1 1 0	B A X 0 0 0 0 1 1 1 0 1 1 1 0	B A X 0 0 1 0 1 0 1 0 0 1 1 1

H.W.

Determine the output waveforms for the NAND gate and for the NOR gate, given the input waveforms, A and B, in Figure below:





BOOLEAN ALGEBRA and Logic Simplification

BOOLEAN Operations and Expressions:

Variable, complement, and literal are terms used in Boolean algebra. A variable is a symbol used to represent a logical quantity. Any single variable can have a **1** or a **0** value. The complement is the inverse of a variable and is indicated by a bar over variable (overbar). For example, the complement of the variable **A** is A If **A = 1,** then **$\overline{A}$ = 0.** If **A = 0**, then **$\overline{A}$ = 1**. The complement of the variable A is read as "not **A**" or **"A** bar."

Sometimes a prime symbol rather than an overbar is used to denote the complement of a variable; for example, B' indicates the complement of B.

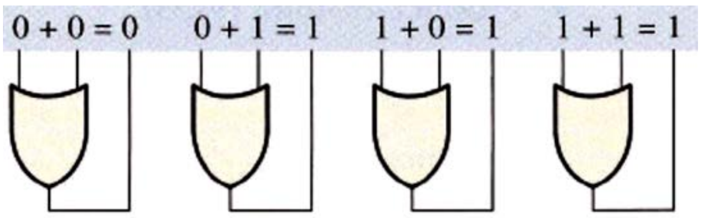


Fig. (1): OR Gate.

A literal is a variable or the complement of a variable.

Boolean Addition:

sum term is a sum of literals. In logic circuits, a sum term is produced by an OR operation with no AND operations involved. Some examples of sum terms are

$A + B$

$A + \overline{B}$,

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

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

<u>EX.1-</u> Determine the values of A, B, C, and D that make the sum term:

$A + \overline{B} + C + \overline{D}$ equal to 0.

For the solution must the variables equal (0) therefore,

A=0, B=1 so that $\overline{B}$ =0, C=0 and D=1 so that $\overline{D}$=0

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

Boolean Multiplication:

In Boolean algebra, a product term is the product of literals. In logic circuits, a product term is produced by an AND operation with no OR operations involved. Some examples of product terms are:

AB, $A\overline{B}$, ABC, and $\overline{A}BC\overline{D}$.

A product term is equal to **1** only if each of the literals in the term is **1**. A product term is equal to **0** when one or more of the literals are **0**.

EX.1- Determine the values of A, B, C, and D that make the product term $\overline{A}B\overline{C}D$ equal to 1?

For the solution to be 1, therefore A=1,B=0 so that $\overline{B}$=1,C=1,and D=0 so that $\overline{D}$=1.

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

EX.2- Evaluate the following expression when **A=1 , B=0 , C=1**

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

Sol.

$$F = 1 + \overline{1}\cdot 0 + 0\cdot\overline{1} = 1 + 0 + 0 = 1$$

Laws and Rules of Boolean algebra:

Laws of Boolean algebra:

The basic laws of Boolean algebra-the commutative laws for addition and multiplication, the associative laws for addition and multiplication, and the distributive law-are the same as in ordinary algebra.

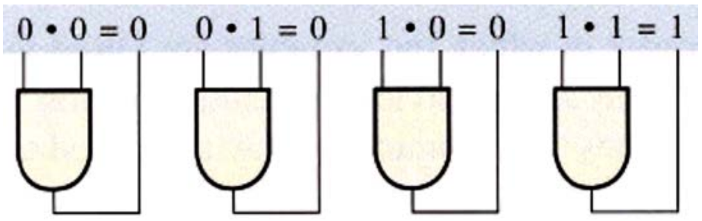


Fig. (2): And Gate.

Commutative Laws:

The commutative law of addition for two variables is written as:

A+B = B+A

This law states that the order in which the variables are ORed makes no difference. Remember, in Boolean algebra as applied to logic circuits, addition and the OR operation are the same. **(The symbol ≡ means "equivalent to.")**.

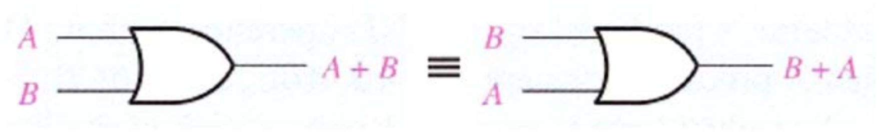


Fig. (3): Application of commutative law of addition.

The commutative law of multiplication for two variables is:

A.B = B.A

This law states that the order in which the variables are ANDed makes no difference. Fig.(4), il1ustrates this law as applied to the AND gate.

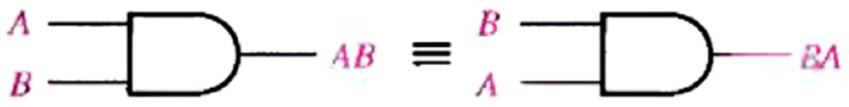


Fig. (4): Application of commutative law of multiplication.

Associative Laws:

The associative law of addition is written as follows for three variables:

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

This law states that when ORing more than two variables, the result is the same regardless of the grouping of the variables. Fig.(4), illustrates this law as applied to 2-input OR gates.

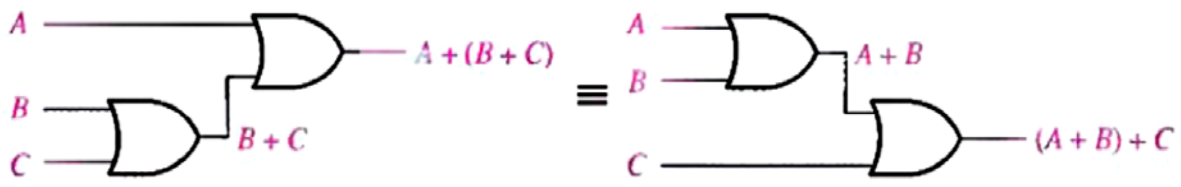


Fig. (5): Application of associative law of addition.

The associative law of multiplication is written as follows for three variables:

A(BC) = (AB)C

This law states that it makes no difference in what order the variables are grouped when ANDing more than two variables. Fig.(6) illustrates this law as applied to 2 -input AND gates.

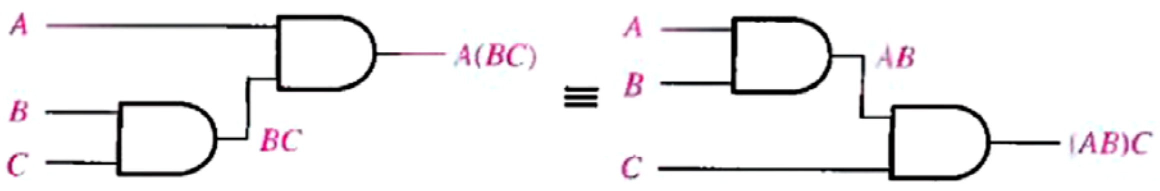


Fig. (6): Application of associative law of multiplication.

Distributive Law:

The distributive law is written for three variables as follows:

A (B + C) = AB + AC

This law states that ORing two or more variables and then ANDing the result with a single variable is equivalent to ANDing the single variable with each of the two or more variables and then ORing the products. The distributive law also expresses the process of factoring in which the common variable A is factored out of the product terms, for example,

AB + AC = A (B + C)

Fig.(7) illustrates the distributive law in terms of gates.

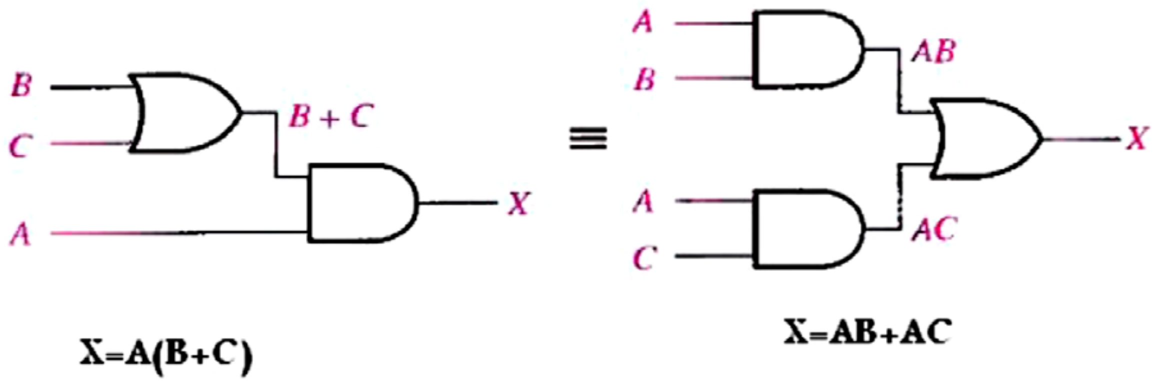
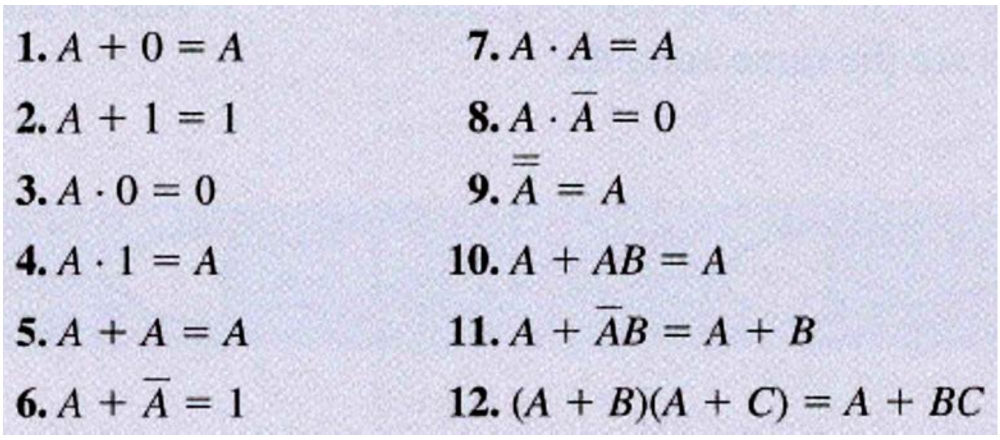


Fig. (7): Application of distributive law.

Rules of Boolean Algebra:

Table 1 lists 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 terms of the simpler rules and the laws previously discussed.

Table 1 Basic rules of Boolean algebra



1. $A + 0 = A$	**7.** $A \cdot A = A$
2. $A + 1 = 1$	**8.** $A \cdot \overline{A} = 0$
3. $A \cdot 0 = 0$	**9.** $\overline{\overline{A}} = A$
4. $A \cdot 1 = A$	**10.** $A + AB = A$
5. $A + A = A$	**11.** $A + \overline{A}B = A + B$
6. $A + \overline{A} = 1$	**12.** $(A + B)(A + C) = A + BC$

1. A+0=A

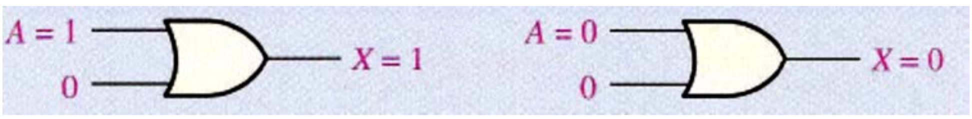


Fig. (8)

X = A+0 = A

2. **A+1=1**

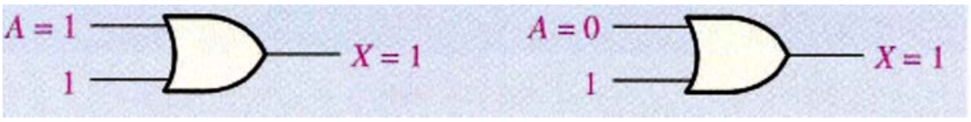


Fig. (9)

X = A+1 = 1

3. **A.0=0**

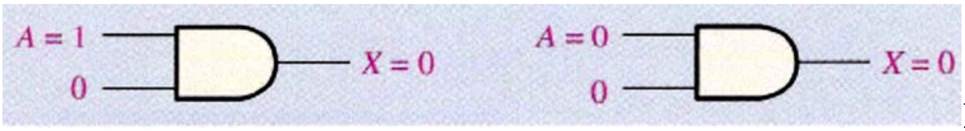


Fig. (10)

X = A.0 = 0

4. **A.1=A**

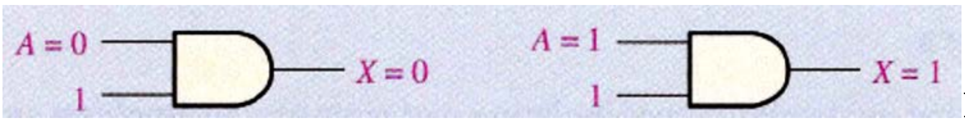


Fig. (11)

X = A.1 = A

5. **A+A=A**

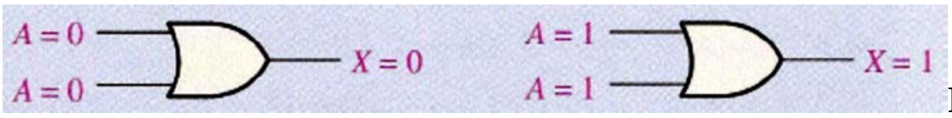


Fig. (12)

X = A+A = A

6. **A+$\overline{A}$=1**

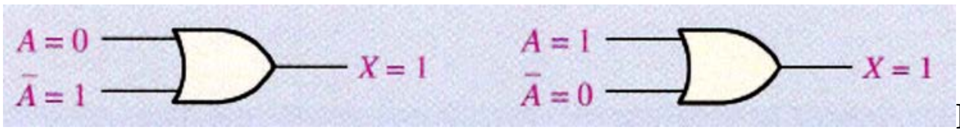


Fig. (13)

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

7. **A. A=A**

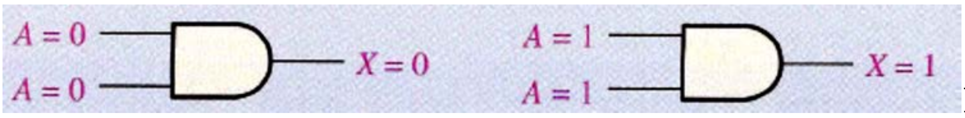


Fig. (14)

X = A. A = A

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

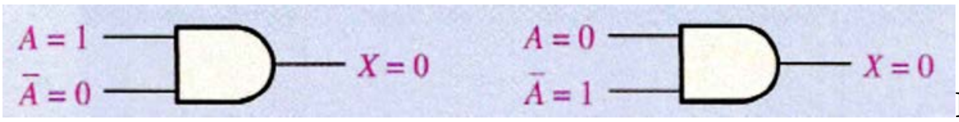


Fig. (15)

9. $\bar{\bar{A}}=A$

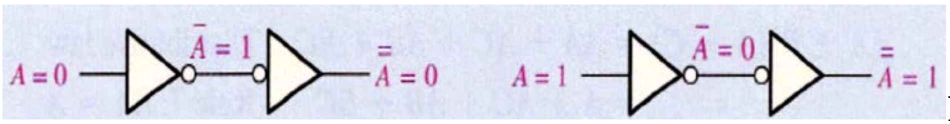


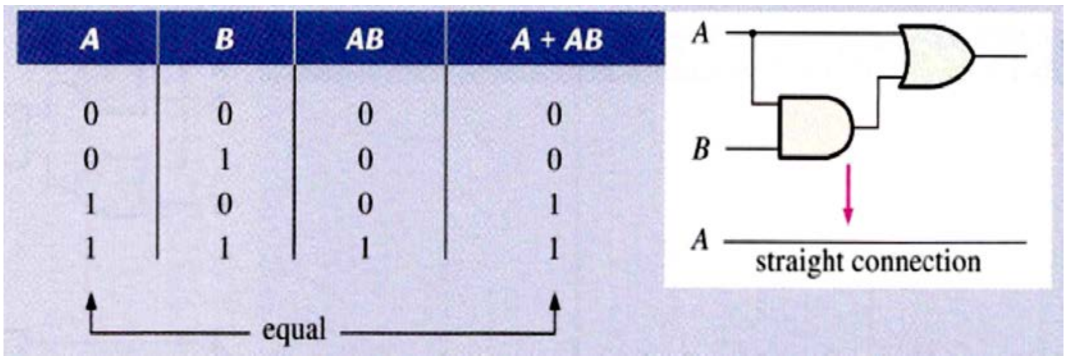
Fig. (16)

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

10. A+AB=A

A+AB = A (1+B)	**Factoring (distributive law)**
= A . 1	**Rule 2: (1 + B) = 1**
= A	**Rule 4: A . 1 = A**

The proof is shown in Table 2, which shows the truth table and the resulting logic circuit simplification:



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

Fig. (17)

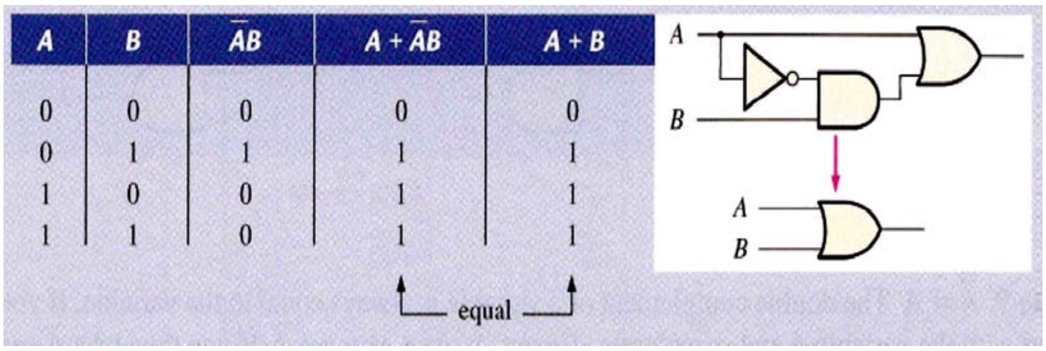
11. $A + \overline{A}B = A + B$

This rule can be proved as follows:

$A + \overline{A}B = (A + AB) + \overline{A}B$	**Rule 10: $A = A + AB$**
$= (AA + AB) + \overline{A}B$	**Rule 7: $A = AA$**
$= AA + AB + A\overline{A} + \overline{A}B$	**Rule 8: adding $A\overline{A} = 0$**
$= (A + \overline{A})(A + B)$	**Factoring**
$= 1. (A + B)$	**Rule 6: $A + \overline{A} = 1$**
$= A + B$	**Rule 4: drop the 1**

The proof is shown in Table 3, which shows the truth table and the resulting logic circuit simplification.

Table 3



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

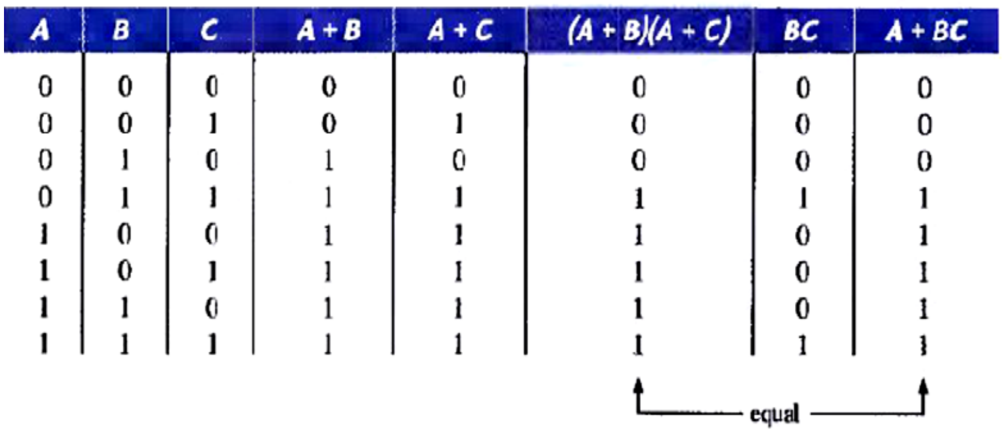
12. $(A + B)(A + C) = A + BC$

This rule can be proved as follows:

It **$(A + B)(A + C) = AA + AC + AB + BC$**	**Distributive law**
$= A + AC + AB + BC$	**Rule 7: $AA = A$**
$= A(1 + C) + AB + BC$	**Factoring (distributive law)**
$= A. 1 + AB + BC$	**Rule 2: $1 + C = 1$**
$= A(1 + B) + BC$	**Factoring (distributive law)**
$= A. 1 + BC$	**Rule 2: $1 + B = 1$**
$= A + BC$	**Rule 4: $A . 1 = A$**

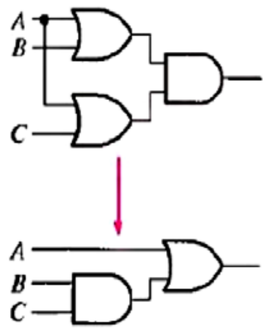
The proof is shown in Table 4, which shows the truth table and the resulting logic circuit simplification.

Table 4

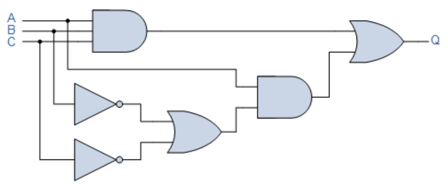


A	B	C	$A+B$	$A+C$	$(A+B)(A+C)$	BC	$A+BC$
0	0	0	0	0	0	0	0
0	0	1	0	1	0	0	0
0	1	0	1	0	0	0	0
0	1	1	1	1	1	1	1
1	0	0	1	1	1	0	1
1	0	1	1	1	1	0	1
1	1	0	1	1	1	0	1
1	1	1	1	1	1	1	1

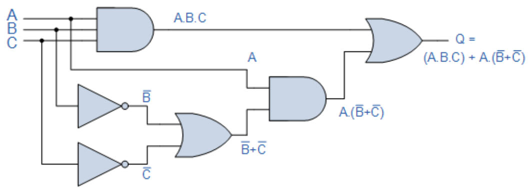
equal



Ex.1-Find the Boolean algebra expression for the following system



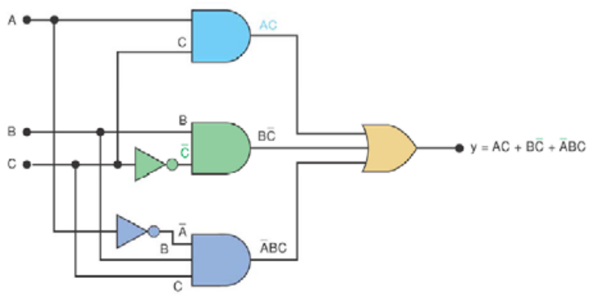
Sol.



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

Ex.2- draw the circuit for y =AC + B$\overline{C}$ + $\overline{A}$BC

Sol.



***Ex*.3-** Simplify the following expression by use of Boolean rules.

1) $y = \overline{A}B\overline{D} + \overline{A}\overline{B}D$

Sol.

$y = A\overline{B}(D+\overline{D})$	by distributive law
$= A\overline{B}\cdot 1$	by rule 6
$= A\overline{B}$	by rule 4

2) $z = (\overline{A}+B)(A+B)$

Sol.

$z = \overline{A}A + \overline{A}B + BA + BB$	by distributive law
$= 0 + \overline{A}B + BA + B$	by rule 8 and rule 7
$= \overline{A}B + BA + B$	by rule 1
$= B(\overline{A} + A + 1)$	by distributive law
$= B\cdot 1$	by rule 6 and rule 2
$= B$	by rule 4

3) $x = ACD + \overline{A}BCD$

Sol.

$x = CD(A + \overline{A}B)$	by distributive law
$= CD(A + B)$	by rule 11
$= ACD + BCD$	by distributive law

***Ex*.4-** Minimize the following expression by use of Boolean rules.

(a) $X = ABC + \overline{A}B + AB\overline{C}$

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

Sol.

(a) $X = ABC + \overline{A}B + AB\overline{C}$

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

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

$= AB + \overline{A}B$ as $C + \overline{C} = 1$

$= (A + \overline{A})B.$
$= 1.B$
$= B$

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

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

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

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

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

$= [(\overline{A} + \overline{B})\,.\,(B + \overline{B})]\,\overline{C}$ by rule ١٢

$= (\overline{A} + \overline{B})\,.\,1]\,\overline{C}$

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

H.W. Given the Boolean function $\boldsymbol{y = A\overline{B}C + \overline{A}\,\overline{B}C + AB\overline{C} + \overline{A}BC + ABC}$

(a) Obtain the truth table of the function.
(b) Draw the logical diagram using the original Boolean expression.
(c) Simplify the function to a minimum number of literals using Boolean algebra.
(d) Obtain the truth table of the function using the simplified expression.
(e) Draw the logical diagram from the simplified expression and compare the total number of gates with the diagram of part (b).

DEMORGAN'S Theorems:

DeMorgan, a mathematician who knew Boole, proposed two theorems that are an important part of Boolean algebra. In practical terms. DeMorgan's theorems provide mathematical verification of the equivalency of the NAND and negative-OR gates and the equivalency of the NOR and negative-AND gates.

One of DeMorgan's theorems is stated as follows:

The complement of a product of variables is equal to the sum of the complements of the variables,

Stated another way,

The complement of two or more ANDed variables is equivalent to the OR of the complements of the individual variables.

The formula for expressing this theorem for two variables is:

$$\mathbf{\overline{XY} = \overline{X} + \overline{Y}}$$

DeMorgan's second theorem is stated as follows:

The complement of a sum of variables is equal to the product of the complements of the variables.

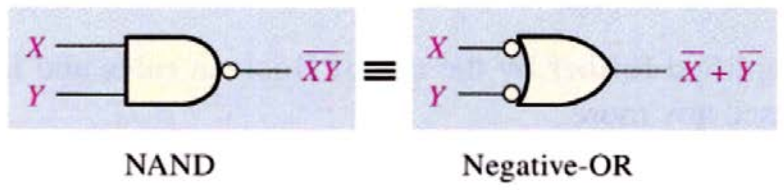
Stated another way,

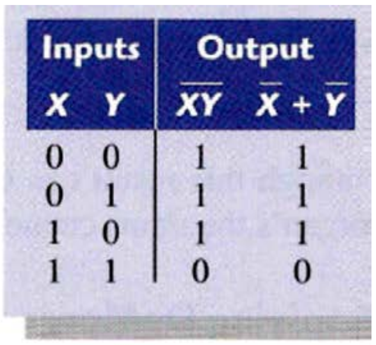
The complement of two or more ORed variables is equivalent to the AND of the complements of the individual variables,

The formula for expressing this theorem for two variables is:

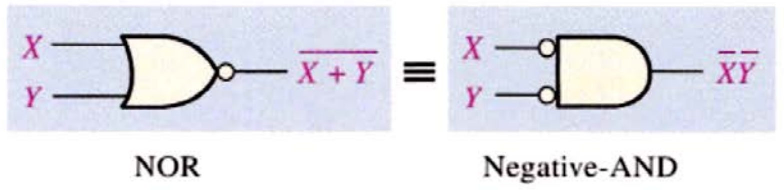
$$\mathbf{\overline{X + Y} = \overline{X}\,\overline{Y}}$$

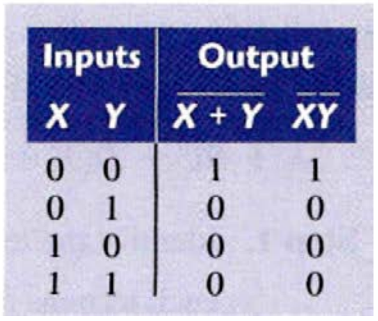
Fig.(18) shows the gate equivalencies and truth tables for the two equations above.





Inputs		Output	
X	Y	$\overline{XY}$	$\overline{X}+\overline{Y}$
0	0	1	1
0	1	1	1
1	0	1	1
1	1	0	0





Inputs		Output	
X	Y	$\overline{X+Y}$	$\overline{X}\,\overline{Y}$
0	0	1	1
0	1	0	0
1	0	0	0
1	1	0	0

Fig. (18): Gate equivalencies and the corresponding truth tables that illustrate DeMorgan's theorems.

As stated, DeMorgan's theorems also apply to expressions in which there are more than two variables. The following examples illustrate the application of DeMorgan's theorems to 3-variable and 4-variable expressions.

<u>EX//</u> Apply DeMorgan's theorems to the expressions $\overline{XYZ}$ and $\overline{X + Y + Z}$.

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

$\overline{X + y + Z} = \overline{X}\,\overline{Y}\,\overline{Z}$

Applying DeMorgan's Theorems:

The following procedure illustrates the application of DeMorgan's theorems and Boolean algebra to the specific expression:

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

Step 1. Identify the terms to which you can apply DeMorgan's theorems, and think of each term as a single variable.

Let:

$\overline{A+B\overline{C}}$ =X and $D(\overline{E+\overline{F}})$ =Y

Step 2. Since $\overline{X + Y} = \overline{X}\,\overline{Y}$,

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

Step 3. Use rule 9 ($A = \overline{\overline{A}}$) to cancel the double bars over the left term (this is not part of DeMorgan's theorem).

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

Step 4. Applying DeMorgan's theorem to the second term**,**

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

Step 5. Use rule 9 ($A = \overline{\overline{A}}$) to cancel the double bars over the $E + \overline{F}$ part of the term.

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

<u>**EX//**</u> Apply DeMorgan's theorems to each of the following expressions:

(a) $\overline{(A + B + C)\,D}$ **(b)** $\overline{ABC + DEF}$ **(c)** $\overline{A\overline{B} + \overline{C}D + EF}$

<u>***SOL//***</u>

(a) Let $A + B + C = X$ and $D = Y$. The expression $\overline{(A + B + C)D}$ is of the form $\overline{XY} = \overline{X} + \overline{Y}$ and can be rewritten as

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

Next, apply DeMorgan's theorem to the term $\overline{A + B + C}$.

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

(b) Let $ABC = X$ and $DEF = Y$. The expression $\overline{ABC + DEF}$ is of the form $\overline{X + Y} = \overline{X}\,\overline{Y}$ and can be rewritten as

$$\overline{ABC + DEF} = (\overline{ABC})(\overline{DEF})$$

Next, apply DeMorgan's theorem to each of the terms $\overline{ABC}$ and $\overline{DEF}$.

$$(\overline{ABC})(\overline{DEF}) = (\overline{A} + \overline{B} + \overline{C})(\overline{D} + \overline{E} + \overline{F})$$

(c) Let $A\overline{B} = X$, $\overline{C}D = Y$, and $EF = Z$. The expression $\overline{A\overline{B} + \overline{C}D + EF}$ is of the form $\overline{X + Y + Z} = \overline{X}\,\overline{Y}\,\overline{Z}$ and can be rewritten as

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

Next, apply DeMorgan's theorem to each of the terms $\overline{A\overline{B}}$, $\overline{\overline{C}D}$, and $\overline{EF}$.

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

Standard Forms of Boolean Expressions:

All Boolean expressions, regardless of their form, can be converted into either of two standard forms: the sum-of-products form or the product-of sums form. Standardization makes the evaluation, simplification, and implementation of Boolean expressions much more systematic and easier.

A binary variable may appear either in its normal form (x) or in its complement form ($\bar{x}$). Now consider two binary variables x and y combined with an **AND** operation. Since each variable may appear in either form, there are four possible combinations: $\bar{x}$ y , x $\bar{y}$, $\bar{x}$ $\bar{y}$, and x y. Each of these four AND terms is called a minterm, or a standard product. In a similar manner, **n** variables can be combined to form $2^{\mathbf{n}}$ minterms. The $2^{\mathbf{n}}$ different minterms may be determined by a method similar to the one shown in Table **2** for three variables.

Each minterm is obtained from an AND term of the n variables, with each variable being primed if the corresponding bit of the binary number is a = 0 and unprimed if a = 1. A symbol for each minterm is also shown in the table and is of the form m_j, where the subscript **j** denotes the decimal equivalent of the binary number of the minterm designated.

In a similar fashion, n variables forming an OR term, with each variable being primed or unprimed, provide $2^{\mathbf{n}}$ possible combinations, called maxterms, or standard sums. The eight maxterms for three variables, together with their symbolic designations, are listed in Table 2. Any $2^{\mathbf{n}}$ maxterms for **n** variables may be determined similarly. It is important to note that (1) each maxterm is obtained from an OR term of the n variables, with each variable being unprimed if the corresponding bit is a = 0 and primed if a = 1, and (2) each **maxterm is the complement of its corresponding minterm and vice versa.**

A Boolean function can be expressed algebraically from a given truth table by forming a minterm for each combination of the variables that produces a 1 in the function and then taking the OR of all those terms.

For example, the function f1 in Table 2 is determined by expressing the combinations 001, 100, and 111 as $\overline{x}\,\overline{y}\,z$, $x\,\overline{y}\,\overline{z}$, and xyz, respectively. Since each one of these minterms results in **f1 = 1**, we have

$$\mathbf{f1 = \overline{x}\,\overline{y}\,z + x\,\overline{y}\,\overline{z} + xyz = m1 + m4 + m7}$$

Table 2, Minterms and Maxterms for Three Binary Variables

X	Y	Z	Minterms		Maxterms	
			Term	Designation	Term	Designation
0	0	0	$\overline{X}\,\overline{Y}\,\overline{Z}$	m_0	$X+Y+Z$	M_0
0	0	1	$\overline{X}\,\overline{Y}\,Z$	m_1	$X+Y+\overline{Z}$	M_1
0	1	0	$\overline{X}\,Y\,\overline{Z}$	m_2	$X+\overline{Y}+Z$	M_2
0	1	1	$\overline{X}\,Y\,Z$	m_3	$X+\overline{Y}+\overline{Z}$	M_3
1	0	0	$X\,\overline{Y}\,\overline{Z}$	m_4	$\overline{X}+Y+Z$	M_4
1	0	1	$X\,\overline{Y}\,Z$	m_5	$\overline{X}+Y+\overline{Z}$	M_5
1	1	0	$X\,Y\,\overline{Z}$	m_6	$\overline{X}+\overline{Y}+Z$	M_6
1	1	1	$X\,Y\,Z$	m_7	$\overline{X}+\overline{Y}+\overline{Z}$	M_7

Sum of Minterms (sum-of-products ' SOP'):

The minterms whose sum defines the Boolean function are those which give the 1's of the function in a truth table.

<u>**EX//**</u> Express the Boolean function $\mathbf{F = A + \overline{B}C}$ as a sum of minterms (sop). The function has three variables: A, B, and C.

<u>***Sol.***</u> The first term **A** is missing two variables; therefore,

$$\mathbf{A = A(B + \overline{B}) = AB + A\overline{B}}$$

This function is still missing one variable, so

$$\mathbf{A = AB(C + \overline{C}) + A\overline{B}(C + \overline{C})}$$

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

The second term $\overline{B}C$ is missing one variable; hence,

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

Combining all terms, we have

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

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

But $A\overline{B}C$ appears twice, and according to theorem 1 (x + x = x), it is possible to remove one of those occurrences. Rearranging the minterms in ascending order, we finally obtain

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

$$\mathbf{= m1 + m4 + m5 + m6 + m7}$$

When a Boolean function is in its sum-of-minterms form, it is sometimes convenient to express the function in the following brief notation:

$$\mathbf{F(A, B, C) = \sum(1, 4, 5, 6, 7)}$$

The summation symbol ∑ stands for the ORing of terms

An alternative procedure for deriving the minterms of a Boolean function is to obtain the truth table of the function directly from the algebraic expression and then read the minterms from the truth table.

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

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

$$\mathbf{F = m1 + m4 + m5 + m6 + m7}$$

$$\mathbf{F(A, B, C) = \sum(1, 4, 5, 6, 7)}$$

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

Product of Maxterms (product-of sums 'POS'):

Each of the 2^{2n} functions of n binary variables can be also expressed as a product of maxterms.

To express a Boolean function as a product of maxterms, it must first be brought into a form of OR terms. This may be done by using the distributive law, **$x + yz = (x + y)(x + z)$**. Then any missing variable **x** in each OR term is ORed with **$x\bar{x}$** The procedure is clarified in the following example.

EX// Express the Boolean function $F = xy + \bar{x}z$ as a product of maxterms(pos)**.**

Sol. First, convert the function into OR terms by using the distributive law:

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

The function has three variables: x, y, and z. Each OR term is missing one variable; therefore,

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

Combining all the terms and removing those which appear more than once, we finally obtain

$$F = (x + y + z)(x + \bar{y} + z)(\bar{x} + y + z)(\bar{x} + y + \bar{z})$$
$$= \quad M0 \qquad\qquad M2 \qquad\qquad M4 \qquad\qquad M5$$

A convenient way to express this function is as follows:

$$F(x, y, z) = \Pi(0, 2, 4, 5)$$

The product symbol, **Π**, denotes the ANDing of maxterms; the numbers are the indices of the maxterms of the function

An alternative procedure for deriving the maxterms of a Boolean function is to obtain the truth table of the function directly from the algebraic expression and then read the maxterms from the truth table.

$F = xy + \overline{x}z$

$F = M0\ M2\ \ M4\ \ M5$

$F(x, y, z) = \Pi(0, 2, 4, 5)$

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

X	Y	Z	F
0	0	0	⓪
0	0	1	1
0	1	0	⓪
0	1	1	1
1	0	0	⓪
1	0	1	⓪
1	1	0	1
1	1	1	1

Conversion between Canonical Forms:

The complement of a function expressed as the sum of minterms equals the sum of minterms missing from the original function. This is because the original function is expressed by those minterms, which make the function equal to 1, whereas its complement is a 1 for those minterms for which the function is a 0. As an example, consider the function:

$F(A, B, C) = \sum(1, 4, 5, 6, 7)$

This function has a complement that can be expressed as:

$\overline{F}\,(A, B, C) = \sum(0, 2, 3) = m_0 + m_2 + m_3$

Now, if we take the complement of $\overline{F}$ by DeMorgan's theorem, we obtain F in a different form:

$F = \overline{(m_0 + m_2 + m_3)} = \overline{m_0}\,.\,\overline{m_2}\,.\,\overline{m_3} = M_0M_2M_3 = \Pi(0, 2, 3)$

The last conversion follows from the definition of minterms and maxterms as shown in Table 2 . From the table, it is clear that the following relation holds:

$\overline{m_j} = M_j$

That is, the **maxterm with subscript j is a complement of the minterm with the same subscript j and vice versa.**

H.W.1- Express the Boolean function **F = AB + BC + AC** as a sum of minterms. The function has three variables: A, B, and C.

H.W.2- Find (POS, SOP) from truth table that given bellow

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

KARNAUGH MAPS

Karnaugh mapping is a method used to simplify a truth table using sum of products or product of sums along with simultaneous optimization of the output function. Karnaugh maps are the graphical equivalent of a truth table. In other words, Karnaugh maps are an easy way of designing and optimizing a circuit from a truth table.

Rules for K-Maps

1. Each cell with a 1 must be included in at least one group.

2. Try to form the largest possible groups.

3. Try to end up with as few groups as possible.

4. Groups may be in sizes that are powers of 2: $2^0 = 1$, $2^1 = 2$, $2^2= 4$, $2^3 =8$, $2^4 = 16$, ...

5. Groups may be square or rectangular only (including wraparound at the grid edges). No diagonals or zig-zags can be used to form a group.

6. The larger a group is, the more redundant inputs there are:

i. A group of 1 has no redundant inputs.

ii. A group of 2 has 1 redundant input.

iii. A group of 4 has 2 redundant inputs.

iv. A group of 8 has 3 redundant inputs.

v. A group of 16 has 4 redundant inputs

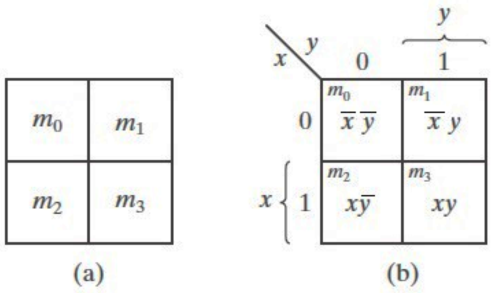
TWO-VARIABLE KARNAUGH MAP:

The two-variable map is shown in Fig. 1 (a). There are four minterms for two variables; hence, the map consists of four squares, one for each minterm. The map is redrawn in (b) to show the relationship between the squares and the two variables x and y . The 0 and 1 marked in each row and column designate the values of variables. Variable x appears primed in row 0 and unprimed in row 1. Similarly, y appears primed in column 0 and unprimed in column 1.

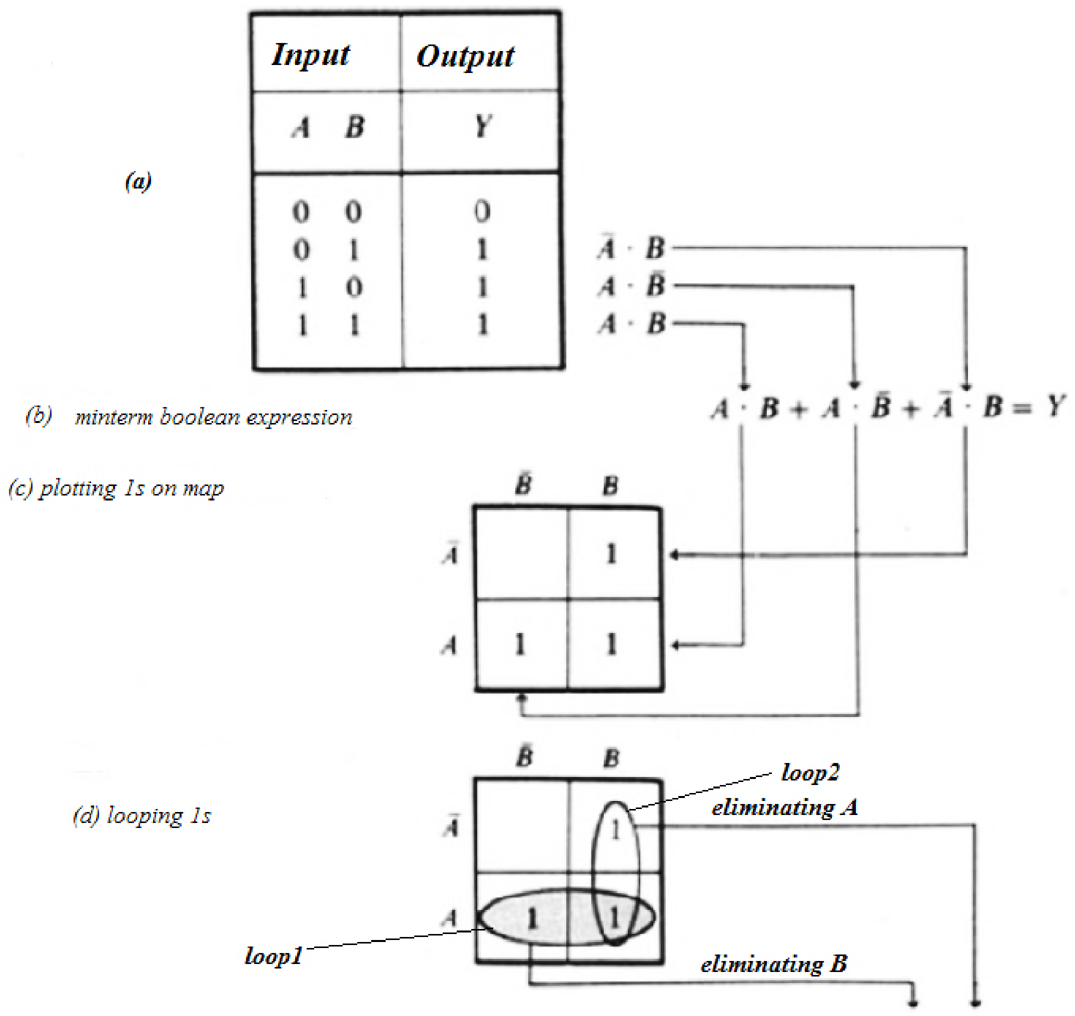
If we mark the squares whose minterms belong to a given function, the two-variable map becomes another useful way to represent any one of the **16** Boolean functions of two variables. As an example, the function xy is shown in Fig. 1 (a). Since xy is equal to **m_3 ,** a **1** is placed inside the square that belongs to m3. Similarly, the function **x + y** is represented in the map of Fig. 1 (b) by three squares marked with **1's**. These squares are found from the minterms of the function:

$$m1 + m2 + m3 = \overline{x}\, y + x\, \overline{y} + xy = x + y$$

A	B	f =1
0	0	0
0	1	1
1	0	1
1	1	1



Two-variable K-map.



Input		Output
A	B	Y
0	0	0
0	1	1
1	0	1
1	1	1

(e) Eliminating variables to form simplified Boolean expression A + B=Y

Fig 2

The 3-Variable Karnaugh Map

The 3-variable Karnaugh map is an array of eight cells. as shown in Fig.(3). In this case, A, B, and C are used for the variables although other letters could be used.

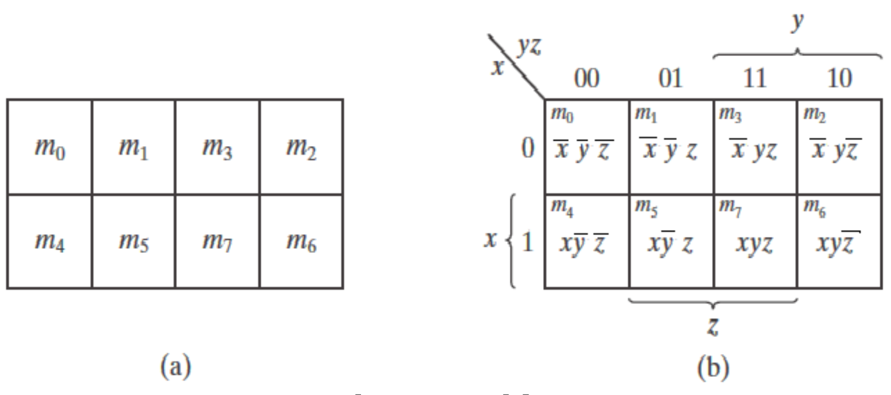

m_0 m_1 m_3 m_2
m_4 m_5 m_7 m_6
(a)
yz x y
00 01 11 10
0 $\bar{x}\bar{y}\bar{z}$ $\bar{x}\bar{y}z$ $\bar{x}yz$ $\bar{x}y\bar{z}$
x 1 $x\bar{y}\bar{z}$ $x\bar{y}z$ xyz $xy\bar{z}$
z
(b)

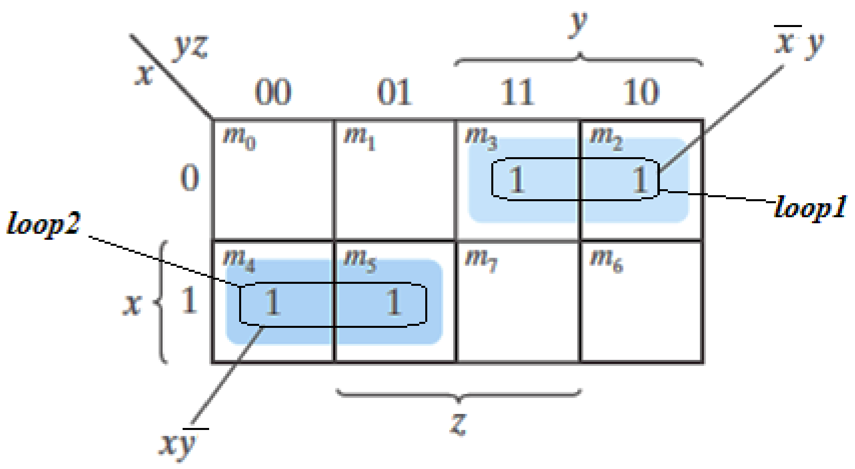
Fig3: three variable K-map

<u>EX//</u> Simplify the Boolean function

F (x, y, z) = ∑(2, 3, 4, 5)

loop1= $\bar{X}\,Y\,Z + \bar{X}\,Y\,\bar{Z} = \bar{x}\,y$

loop2= $x\,\bar{y}\,\bar{z} + x\,\bar{y}\,z = x\,\bar{y}$



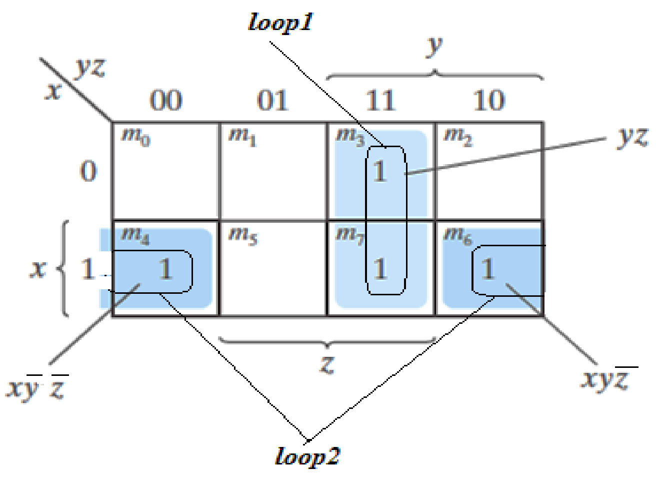
Map for Example , F (x, y, z) = ∑(2, 3, 4, 5) = $\bar{x}\,y + x\,\bar{y}$

<u>Ex//</u> Simplify the Boolean function:

F (x, y, z) = ∑(3, 4, 6, 7)

loop1= $\bar{x}\,y\,z + x\,y\,z = y\,z$

loop2= $x\,\bar{y}\,\bar{z} + x\,y\,\bar{z} = x\,\bar{z}$

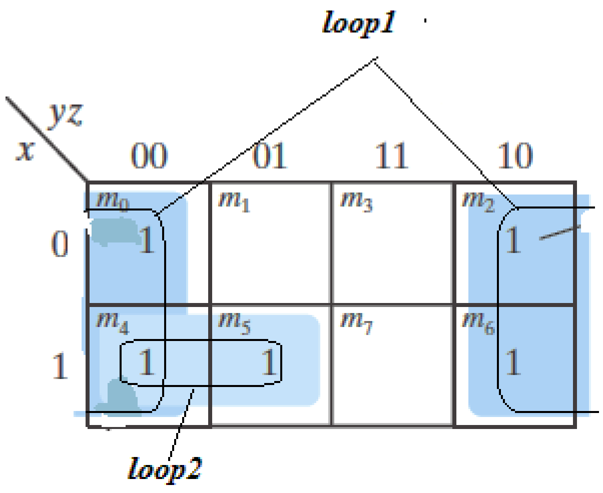


$f= Y\,Z+X\,\overline{Z}$

<u>EX//</u> Simplify the Boolean function F (x, y, z) = ∑(0, 2, 4, 5, 6)

$\text{loop1}= \bar{x}\,\bar{y}\,\bar{z}+ \bar{x}\,y\,\bar{z} + x\,\bar{y}\,\bar{z} + x\,y\,\bar{z} = \bar{z}$

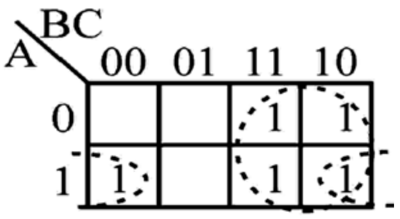
$\text{loop2} = x\,\bar{y}\,\bar{z} + x\,\bar{y}\,z = x\,\bar{y}$



$F=\overline{Z}+X\,\overline{Y}$

EX// **Simplify by using K-map:**

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



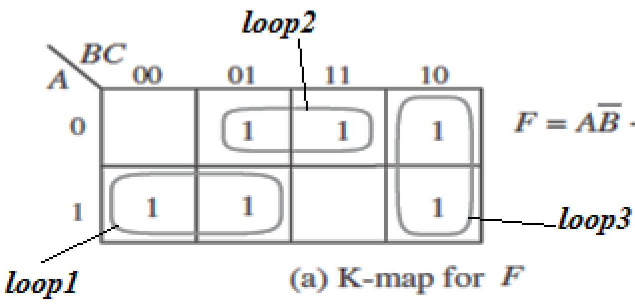
Y = AC’ + B

EX// **Simplify f =∑(1,2,3,4,5,6):**

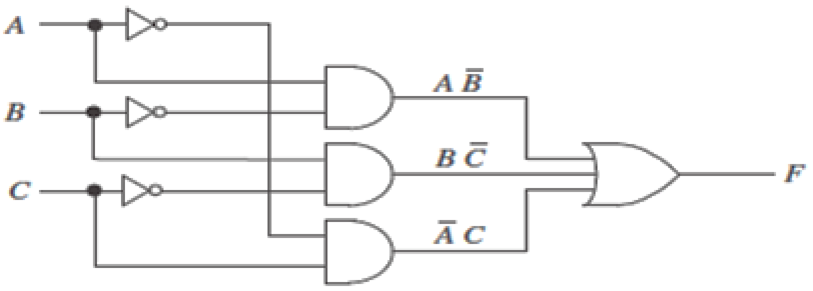
loop1 $= A\,\bar{B}\,\bar{C} + A\,\bar{B}\,C = A\,\bar{B}$

loop2 $= \bar{A}\,\bar{B}\,C + \bar{A}\,B\,C = \bar{A}\,C$

loop3 $= \bar{A}\,B\,\bar{C} + A\,B\,\bar{C} = B\,\bar{C}$



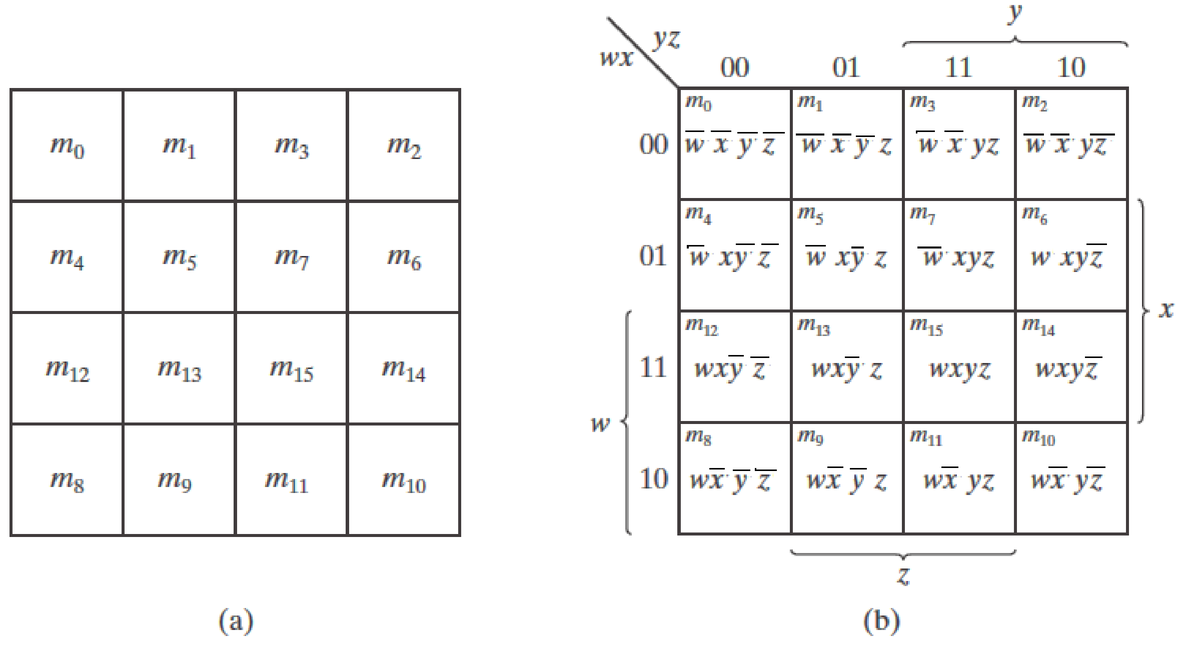
(a) K-map for F



(b) Logic Diagram for the output, F

FOUR-VARIABLE K-MAP

The map for Boolean functions of four binary variables (w, x, y, z) is shown in Fig. 8 .In Fig. 8(a) are listed the 16 minterms and the squares assigned to each. In Fig.8(b), the map is redrawn to show the relationship between the squares and the four variables.



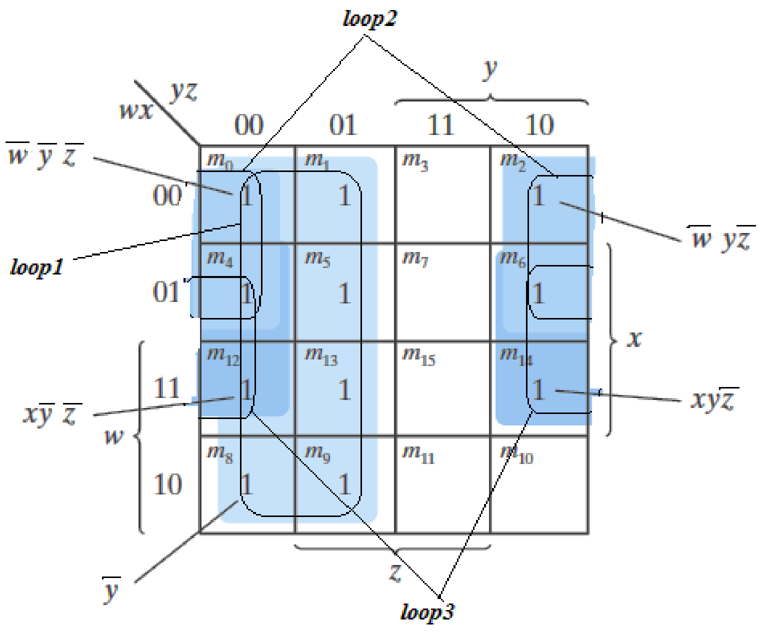
m_0 m_1 m_3 m_2
m_4 m_5 m_7 m_6
m_{12} m_{13} m_{15} m_{14}
m_8 m_9 m_{11} m_{10}
(a)
wx yz
00 01 11 10
y
x
w
z
$\overline{w}\,\overline{x}\,\overline{y}\,\overline{z}$ $\overline{w}\,\overline{x}\,\overline{y}\,z$ $\overline{w}\,\overline{x}\,yz$ $\overline{w}\,\overline{x}\,y\overline{z}$
$\overline{w}\,x\overline{y}\,\overline{z}$ $\overline{w}\,x\overline{y}\,z$ $\overline{w}\,xyz$ $\overline{w}\,xy\overline{z}$
$wx\overline{y}\,\overline{z}$ $wx\overline{y}\,z$ $wxyz$ $wxy\overline{z}$
$w\overline{x}\,\overline{y}\,\overline{z}$ $w\overline{x}\,\overline{y}\,z$ $w\overline{x}\,yz$ $w\overline{x}\,y\overline{z}$
(b)

Fig(8) four variables

Ex //Simplify the Boolean function

F (w, x, y, z) = ∑(0, 1, 2, 4, 5, 6, 8, 9, 12, 13, 14)

w	x	y	z	F
0	0	0	0	1
0	0	0	1	1
0	0	1	0	1
0	0	1	1	0
0	1	0	0	1
0	1	0	1	1
0	1	1	0	1
0	1	1	1	0
1	0	0	0	1
1	0	0	1	1
1	0	1	0	0
1	0	1	1	0
1	1	0	0	1
1	1	0	1	1
1	1	1	0	1
1	1	1	1	0



loop1= $\bar{w}\,\bar{x}\,\bar{y}\,\bar{z} + \bar{w}\,\bar{x}\,\bar{y}\,z + \bar{w}\,x\,\bar{y}\,\bar{z} + \bar{w}\,x\,\bar{y}\,z + w\,\bar{x}\,\bar{y}\,\bar{z} + w\,\bar{x}\,\bar{y}\,z + w\,x\,\bar{y}\,\bar{z} + w\,x\,\bar{y}z$
$= \bar{y}$

loop2= $\bar{w}\,\bar{x}\,\bar{y}\,\bar{z} + \bar{w}\,\bar{x}\,y\,\bar{z} + \bar{w}\,x\,\bar{y}\,\bar{z} + \bar{w}\,x\,y\,\bar{z} = \bar{w}\,\bar{z}$

loop3 = $\bar{w}\,x\,\bar{y}\,\bar{z} + \bar{w}\,x\,y\,\bar{z} + w\,x\,\bar{y}\,\bar{z} + w\,x\,y\,\bar{z} = x\,\bar{z}$

$f = \bar{y} + \bar{w}\,\bar{z} + x\,\bar{z}$

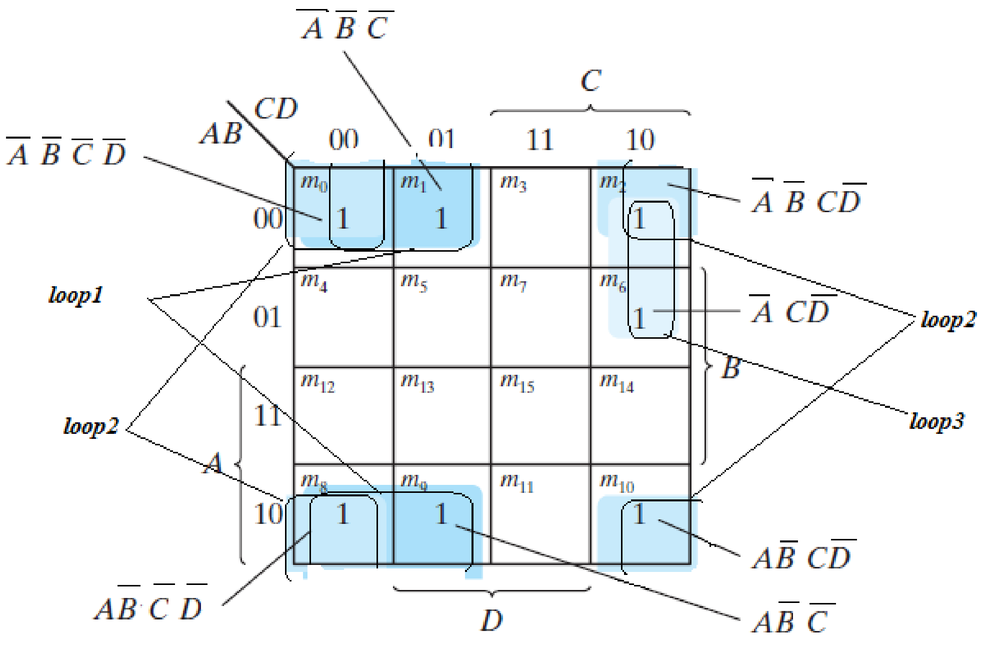
<u>Ex //</u> Simplify the Boolean function:

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

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

$f = \bar{A}\,\bar{B}\,\bar{C}\,\bar{D} + \bar{A}\,\bar{B}\,\bar{C}\,D + \bar{A}\,\bar{B}\,C\,\bar{D} + \bar{A}\,B\,C\,\bar{D} + A\,\bar{B}\,\bar{C}\,\bar{D} + A\,\bar{B}\,\bar{C}\,D + A\,\bar{B}\,C\,\bar{D}$

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



$\overline{A}\,\overline{B}\,\overline{C}$
C
CD
AB
00
01
11
10
$\overline{A}\,\overline{B}\,\overline{C}\,\overline{D}$
$\overline{A}\,\overline{B}\,C\overline{D}$
$\overline{A}\,C\overline{D}$
loop1
loop2
loop3
B
A
m_0 m_1 m_3 m_2
m_4 m_5 m_7 m_6
m_{12} m_{13} m_{15} m_{14}
m_8 m_9 m_{11} m_{10}
1
$A\overline{B}\,\overline{C}\,\overline{D}$
$A\overline{B}\,C\overline{D}$
$A\overline{B}\,\overline{C}$
D

loop1=$\overline{A}\ \overline{B}\ \overline{C}\ \overline{D}$+ $\overline{A}\ \overline{B}\ \overline{C}$ D+A $\overline{B}\ \overline{C}\ \overline{D}$ + A $\overline{B}\ \overline{C}$ D = $\overline{B}\ \overline{C}$

loop2= $\overline{A}\ \overline{B}\ \overline{C}\ \overline{D}$+ $\overline{A}\ \overline{B}$ C $\overline{D}$+A $\overline{B}$ C $\overline{D}$ + A $\overline{B}\ \overline{C}\ \overline{D}$= $\overline{B}\overline{D}$

loop3 = $\overline{A}\ \overline{B}$ C $\overline{D}$+ $\overline{A}$ B C $\overline{D}$ =$\overline{A}C\overline{D}$

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

Don’t-care conditions:

In most applications, we simply don’t care what value is assumed by the function for the unspecified minterms. For this reason, it is customary to call the unspecified minterms of a function don’t-care conditions. These don’t-care conditions can be used on a map to provide further simplification of the Boolean expression.

A don’t-care minterm is a combination of variables whose logical value is not specified. Such a minterm cannot be marked with a 1 in the map, because it would require that the function always be a 1 for such a combination. Likewise, putting a 0 on the square requires the function to be 0. To distinguish the don’t-care condition from 1’s and 0’s, an X is used. Thus, an X inside a square in the map indicates that we don’t care whether the value of 0 or 1 is assigned to F for the particular minterm.

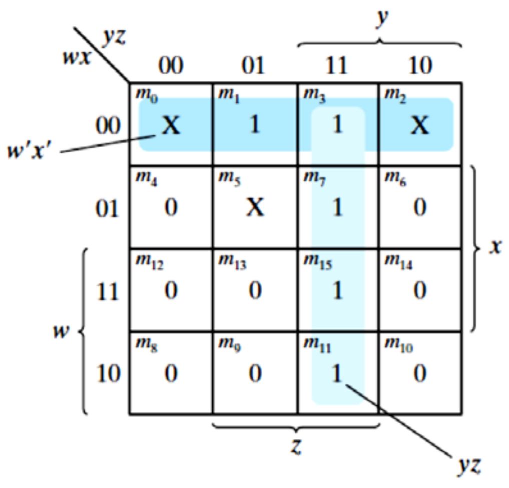
In choosing adjacent squares to simplify the function in a map, the don’t-care minterms may be assumed to be either 0 or 1. When simplifying the function, we can choose to include each don’t-care minterm with either the 1’s or the 0’s, depending on which combination gives the simplest expression.

Ex// Simplify the Boolean function

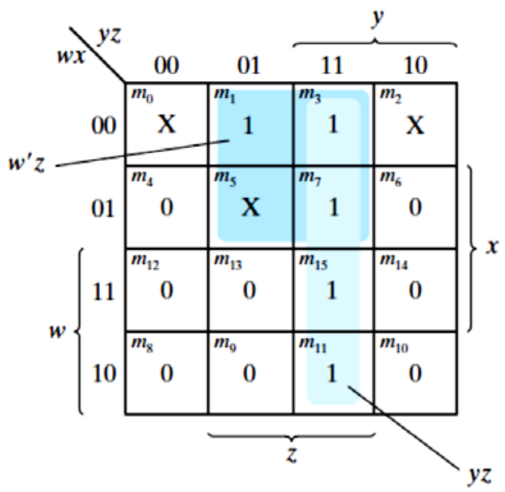
$F(w, x, y, z) = \sum(1, 3, 7, 11, 15)$

which has the don’t-care conditions

$d(w, x, y, z) = \sum(0, 2, 5)$



(a) $F = yz + w'x'$



(b) $F = yz + w'z$