Digital Logic Design

Introduction:

A digital computer stores data in terms of digits (numbers). and proceeds in discrete steps from one state to the next. The states of a digital computer typically involve binary digits which may take the form of the presence or absence of magnetic markers in a storage medium, on-off switches or relays. In digital computers, even letters, words and whole texts are represented digitally.

Digital Logic is the basis of electronic systems, such as computers and cell phones. Digital Logic is rooted in binary code, a series of zeroes and ones each having an opposite value. This system facilitates the design of electronic circuits that convey information, including logic gates. Digital Logic gate functions include and, or and not. The value system translates input signals into specific output. Digital Logic facilitates computing, robotics and other electronic applications.

Digital Logic Design is foundational to the fields of electrical engineering and computer engineering. Digital Logic designers build complex electronic components that use both electrical and computational characteristics. These characteristics may involve power, current, logical function, protocol and user input. Digital Logic Design is used to develop hardware, such as circuit boards and microchip processors. This hardware processes user input, system protocol and other data in computers, navigational systems, cell phones or other high-tech systems.

Digital Number Systems

Many number systems are in use in digital technology. The most common are the decimal, binary, octal, and hexadecimal systems. The decimal system is clearly the most familiar to us because it is a tool that we use every day.
Examining some of its characteristics will help us to understand the other systems better.

❖ Decimal System:

The **decimal system** is composed of *10* numerals or symbols. These 10 symbols are 0, 1, 2, 3, 4, 5, 6, 7, 8, 9; using these symbols as *digits* of a number, we can express any quantity. The decimal system, also called the *base-10* system because it has 10 digits. The decimal system is a positional-value system in which the value of a digit depends on its position. For example, consider the decimal number 453 .

We know that the digit 4 actually represents 4 hundreds, the 5 represents 5 tens, and the 3 represents 3 units. In essence, the 4 carries the most weight of the three digits; it is referred to as the most significant digit (MSD). The 3 carries the least weight and is called the least significant digit (LSD).

Consider another example, 27.35 . This number is actually equal to 2 tens plus 7 units plus 3 tenths plus 5 hundredths, or $2 \times 10 + 7 \times 1 + 3 \times 0.1 + 5 \times 0.01$. The decimal point is used to separate the integer and fractional parts of the number.

Moreover, the various positions relative to the decimal point carry weights that can be expressed as powers of 10. This is illustrated in Figure (1), where the number 2745.214 is represented. The decimal point separates the positive powers of 10 from the negative powers. The number 2745.214 is thus equal to

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

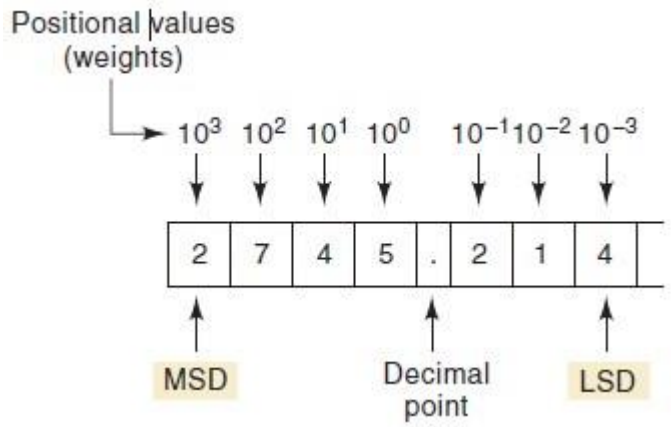


Figure (1): Decimal position values as powers of 10.

In general, any number is simply the sum of the products of each digit value and its positional value.

❖ Binary System:

In the **binary system** there are only two symbols or possible digit values, 0 and 1. Even so, this base-2 system can be used to represent any quantity that can be represented in decimal or other number systems. In general though, it will take a greater number of binary digits to express a given quantity. All of the statements made earlier concerning the decimal system are equally applicable to the binary system. The binary system is also a positional value system, wherein each binary digit has its own value or weight expressed as a power of 2. This is illustrated in Figure (2).

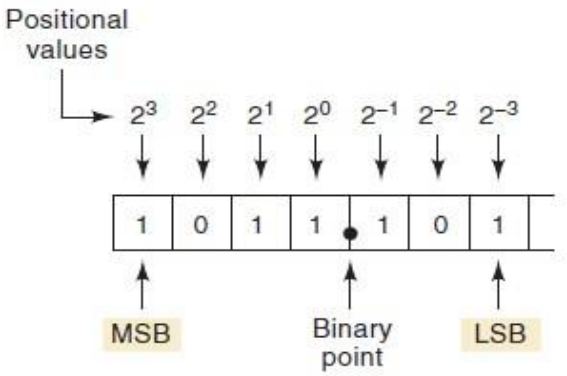


Figure (2): Binary position values as powers of 2.

Here, places to the left of the binary point (counterpart of the decimal point) are positive powers of 2, and places to the right are negative powers of 2. The number 1011.101 is shown represented in the figure. To find its equivalent in the decimal system, we simply take the sum of the products of each digit value (0 or 1) and its positional value:

$$\begin{aligned} 1011.101_2 &= (1 \times 2^3) + (0 \times 2^2) + (1 \times 2^1) + (1 \times 2^0) \\ &\quad + (1 \times 2^{-1}) + (0 \times 2^{-2}) + (1 \times 2^{-3}) \\ &= 8 + 0 + 2 + 1 + 0.5 + 0 + 0.125 \\ &= 11.625_{10} \end{aligned}$$

Notice in the preceding operation that subscripts (2 and 10) were used to indicate the base in which the particular number is expressed. This convention is used to avoid confusion whenever more than one number system is being employed.

In the binary system, the term binary digit is often abbreviated to the term bit, which we will use from now on. Thus, in the number expressed in Figure (2) there are four bits to the left of the binary point, representing the integer part of the number, and three bits to the right of the binary point, representing the fractional part. The most significant bit (MSB) is the leftmost bit (largest weight). The least significant bit (LSB) is the rightmost bit (smallest weight). These are indicated in Figure (2). Here, the MSB has a weight of 2^3; the LSB has a weight of 2^{-3}.

❖ Octal Number System:

Characteristics

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

Example

Octal Number − 12570_8

Calculating Decimal Equivalent −

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

Note: 12570_8 is normally written as 12570.

❖ Hexadecimal Number System:

Characteristics

- 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 an x power of the base (16). Example 16^x where x represents the last position - 1.

Example

Hexadecimal Number: $19FDE_{16}$

Calculating Decimal Equivalent –

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

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

Number System Conversion

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

- Decimal to Other Base System
- Other Base System to Decimal
- Binary to Octal
- Octal to Binary
- Binary to Hexadecimal
- Hexadecimal to Binary

❖ Decimal to Other Base System:

To convert any Number from the decimal system to other systems we divided the Number into two parts:

1. The Integer part:

Steps

Step 1 − Divide the decimal number to be converted by the value of the new base.

Step 2 − Get the remainder from Step 1 as the rightmost digit (least significant digit) of new base number.

Step 3 − Divide the quotient of the previous divide by the new base.

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

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

The last remainder thus obtained will be the Most Significant Digit (MSD) of the new base number.

Example

Decimal Number: 29_{10}

Calculating Binary Equivalent

Operation	Integer	Remainder (Result)	
29 / 2	14	1	(LSB)
14 / 2	7	0	
7 / 2	3	1	
3 / 2	1	1	
1 / 2	0	1	(MSB)

As mentioned in Steps 2 and 4, the remainders have to be arranged in the reverse order so that the first remainder becomes the Least Significant Digit (LSD) and the last remainder becomes the Most Significant Digit (MSD).

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

2. The Fraction part:

Multiply the decimal number by the new base number to give an integer and a fraction. The integer number after each multiplication will be result, Then the new remainder of fraction is multiplied by the new base number to give a new integer and a new fraction. The process is continued until the fraction becomes 0 or until the number of digits has sufficient accuracy.

Example

Convert $(0.6875)_{10}$ to binary

Operation	Integer	Fraction	Result (Integer)
0.6875 x 2	1	0.3750	1
0.3750 x 2	0	0.7500	0
0.7500 x 2	1	0.5000	1
0.5000 x 2	1	0.0000	1

Therefore, the answer is $(0.6875)_{10} = (0.1011)_{2}$.

Example

Convert $(153.513)_{10}$ to Octal

	Integer	Remainder			Integer	
153 / 8 =	19	1	↑	0.513 x 8 = 4.104	4	↓
19 / 8 =	2	3		0.104 x 8 = 0.832	0	
2 / 8 =	0	2	$= (231)_8$	0.832 x 8 = 6.656	6	$= (406)_8$

The answer is $(153.513)_{10} = (231.406)_{8}$

❖ Other Base System to Decimal:

Steps

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

Example

Binary Number $(11101)_2$

Calculating Decimal Equivalent

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

Binary Number $(11101)_2$ = Decimal Number $(29)_{10}$

Example

Convert $(167)_8$ to Decimal

$(167)_8 = 1x8^2 + 6x8^1 + 7x8^0$

$= 64 + 48 + 7$

$= (119)_{10}$

Example

Convert $(35.62)_8$ to Decimal

$(35.62)_8 = 3x8^1 + 5x8^0 + 6x8^{-1} + 2x8^{-2}$

$= 24 + 5 + 0.75 + 0.031$

$= (29.781)_{10}$

Example

Convert $(3C6E.2AF)_{16}$ to Decimal

$(3C6E.2AF)_{16} = 3x16^3 + 12x16^2 + 6x16^1 + 14x16^0 + 2x16^{-1} + 10x16^{-2} + 15x16^{-3}$

$= 12288 + 3072 + 96 + 14 + 0.125 + 0.039 + 0.003$

$= (15470.167)_{10}$

❖ Binary to Octal:

Steps

- Step 1 − Divide the binary digits into groups of three (starting from the right) as in table below.

Octal Number	Groups in Binary
0	000
1	001
2	010
3	011
4	100
5	101
6	110
7	111

- Step 2 − Convert each group of three binary digits to one octal digit.

Example

Convert $(10101)_2$ to Octal

$1\ 0\ 1\ 0\ 1_2 = \underbrace{0\ 1\ 0}_{2}\ \underbrace{1\ 0\ 1}_{5}$

$= 25_8$

Example

Convert $(111011100)_2$ to Octal

$111011100_2 = \underbrace{1\ 1\ 1}_{7}\ \underbrace{0\ 1\ 1}_{3}\ \underbrace{1\ 0\ 0}_{4}$

$= 734_8$

❖ Octal to Binary:

Steps

- Step 1 − Convert each octal digit to a 3 digit binary number.
- Step 2 − Combine all the resulting binary groups (of 3 digits each) into a single binary number.

Example

Convert $(746)_8$ to binary

$746_8 = 7\ \ 4\ \ 6$

$= 111\ \ 100\ \ 110 = (111100110)_2$

❖ Binary to Hexadecimal:

Steps

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

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

Example

Convert $(1000011010101100.111111)_2$ to Hexadecimal

$1000011010101100.111111_2 = \underbrace{1\,0\,0\,0}\,\underbrace{0\,1\,1\,0}\,\underbrace{1\,0\,1\,0}\,\underbrace{1\,1\,0}\,0.\underbrace{1\,1\,1\,1}\,\underbrace{1\,1\,0\,0}$

$= \quad 8 \quad 6 \quad A \quad C \;.\; F \quad C$

$= \quad 86AC.FC_{16}$

❖ Hexadecimal to Binary:

Steps

- Step 1 − Convert each octal digit to a 4 digit binary number.
- Step 2 − Combine all the resulting binary groups (of 4 digits each) into a single binary number.

Example

Convert $(59AB.2F)_{16}$ to binary

$59AB.2F_{16}$ = 5 9 A B . 2 F

= 0101 1001 1010 1011 . 0010 1111

= $0101100110101011.00101111_2$

❖ **Numbers with Different Bases**

Decimal (base 10)	Binary (base 2)	Octal (base 8)	Hexadecimal (base 16)
00	0000	00	0
01	0001	01	1
02	0010	02	2
03	0011	03	3
04	0100	04	4
05	0101	05	5
06	0110	06	6
07	0111	07	7
08	1000	10	8
09	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

Arithmetic in binary is much like arithmetic in other numeral systems. Addition, subtraction, multiplication, and division can be performed on binary numerals. And it is essential part of all the digital computers and many other digital systems.

- **Binary Addition**

It is a key for binary subtraction, multiplication, division. There are four rules of 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 written in the given column and a carry of 1 over to the next column.

Example

0011010 + 001100 = 00100110

```
  1 1   <-- carry
  0 0 1 1 0 1 0   = 26(10)
+ 0 0 0 1 1 0 0   = 12(10)
-----------------
  0 1 0 0 1 1 0   = 38(10)
```

- **Binary Subtraction**

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

Example

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 are four rules of 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

Example

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
-------------------
1 0 0 1 1 1 0 0 0   = 312₁₀
```

- **Binary Division**

Binary division is similar to decimal division. It is called as the long division procedure.

Example

101010 / 000110 = 000111

```
          111     =7₁₀
      ------------
000110 ) 101010   =42₁₀
        -110      =6₁₀
        ----
         1001
        -110
        ----
          110
         -110
         ----
            0
```

Example

1001011 ÷ 11 = 011001

```
          011001
      ------------
11  ) 1001011
       -00
       ---
        100
        -11
        ---
         011
         -11
         ---
          000
          -00
          ---
           001
           -00
           ---
            011
            -11
            ---
             00
```

Complements of Numbers

Complements are used in digital computers to **simplify the subtraction operation**. There are two types of complements for each base-*r* system: the radix complement and the diminished radix complement. The first is referred to as the *r*'s complement and the second as the (*r* - 1)'s complement. When the value of the base *r* is substituted in the name, the two types are referred to as the 2's complement and 1's complement for binary numbers and the 10's complement and 9's complement for decimal numbers.

1) 9's-Complement (Diminished Radix Complement): For Decimal Numbers

Given a number N in base r having n digits, the (r - 1)'s complement of N , i.e., its diminished radix complement, is defined as $(r^n - 1) - N$. For decimal numbers, r = 10 and r - 1 = 9, so the 9's complement of N is $(10^n - 1) - N$. In this case, 10^n represents a number that consists of a single 1 followed by *n*0's. $10^n - 1$ is a number represented by *n*9's. For example, if n = 4, we have $10^4 = 10{,}000$ and $10^4 - 1 = 9999$. It follows that the 9's complement of a decimal number is obtained by subtracting each digit from 9.

Example: Find the 9's-complement of 546700.

Solution

```
  9 9 9 9 9 9
  5 4 6 7 0 0  –
-------------
  4 5 3 2 9 9
```

Example: Find the 9's-complement of 46.653.

Solution

```
  9 9 9 9 9
  4 6.6 5 3  –
-----------
  5 3.3 4 6
```

2) 10's- Complement (Radix Complement): For Decimal Numbers

The r's complement of an n-digit number N in base r is defined as $r^n - N$ for $N \neq 0$ and as 0 for N = 0. Comparing with the (r - 1)'s complement, we note that the r's complement is obtained by adding 1 to the (r - 1)'s complement, since $r^n - N = [(r^n - 1) - N] + 1$. Thus, the 10's complement of decimal 2389 is 7610 + 1 = 7611 and is obtained by adding 1 to the 9's complement value.

*Since 10 is a number represented by a 1 followed by *n*0's, $10^n - N$, which is the 10's complement of N, can be formed also by leaving all least significant 0's unchanged, subtracting the first nonzero least significant digit from 10, and subtracting all higher significant digits from 9.

Example: Find the 10's-complement of 246700.

Solution

```
  9 9 9 10
  2 4 6  7 0 0  –
-------------
  7 5 3  3 0 0
```

First leave the two least significant 0's unchanged, subtracting 7 from 10, and subtracting the other three digits from 9. Hence, the 10's-complement of 246700 is 753300.

Example: Find the 10's-complement of 55274.

Solution

```
  9 9 9 9 10
  5 5 2 7  4  –
-------------
  4 4 7 2  6
```

First subtract 4 (lsd) from 10 we obtain 6. The other digits of the given example, i.e. 7, 2, 5, and 5 will be subtracted from 9 and we obtain 2, 7, 4, and 4 respectively. Hence, the 10's-complement of 55274 is 44726.

Example: Find the 10's-complement of 46.653.

Solution

```
  9 9 9 9 10
  4 6.6 5  3  –
-------------
  5 3.3 4  7
```

3) 1's- Complement (Diminished Radix Complement): <u>For Binary Numbers</u>

For binary numbers, r = 2 and r - 1 = 1, so the 1's complement of N is $(2^n - 1) - N$. Again, 2^n is represented by a binary number that consists of a 1 followed by *n*0's. $2^n - 1$ is a binary number represented by *n*1's. For example, if n = 4, we have $2^4 = (10000)_2$ and $2^4 - 1 = (1111)_2$. Thus, the 1's complement of a binary number is obtained by subtracting each digit from 1. However, when subtracting binary digits from 1, we can have either 1 - 0 = 1 or 1 - 1 = 0, which causes the bit to change from 0 to 1 or from 1 to 0, respectively. Therefore, the 1's complement of a binary number is formed by changing 1's to 0's and 0's to 1's.

Example: Find the 1's-complement of 1011010.

Solution 1011010 = 0100101

Example: Find the 1's-complement of 0.0101.

Solution: The 1's-complement of 0.0101 is 0.1010.

The first zero (0) to the left of the binary point that separates the integer and the fractional part remains unchanged.

Example: Find the 1's-complement of 1010.0011.

Solution 1010.0011 = 0101.1100

4) 2's- Complement (Radix Complement): For Binary Numbers

The 2's-complement can be found by unchanging all the least significant 0's and the least significant 1's and replacing all other 0's with 1's and all other 1's with 0's in the remaining number. 2's-complement of the number can also be obtained by adding 1 to the least significant digit in 1's-complement of the given number.

Example: Find the 2's-complement of 1011010.

Solution: Looking from the right of the given number, the least significant one appears at 2nd digit, hence we will not change 10 and for the remaining digits 10110, replace all ones with zeros and all zeros with ones to find the 2's-complement of a number. Hence, the 2's-complement of 1011010 is 0100110.

Example: Find the 2's-complement of 0.0101.

Solution: The 2's-complement of 0.0101 is 0.1011.

The first zero (0) to the left of the binary point, which separates the integer and the fractional parts remains unchanged.

Example: Find the 2's-complement of 1010.0011.

Solution 1010.0011 = 0101.1101

Subtraction with Complements

1) Subtraction with 10's-Complement and 2's-Complement

In the subtraction of digital systems, it is assumed that both the numbers for the subtraction operation are positive. With the help of 10's-complement or 2's-complements the subtraction operation is performed in the following procedure:

• Obtain the 10's-complement or 2's-complement of the subtrahend and add it to the minuend.

• Verify the result (sum), and observe the carry. Discard the carry if it occurs at the end. Take the 10's-complement or 2's-complement of the result (sum) and place minus (–) sign before it if no carry occurs at the end. The examples of the both cases are given below:

Example: Find the subtraction $(51346 - 06934)_{10}$ using the 10's-complement method.

Solution:

Minuend = 51346

Subtrahend = 06938

Minuend	=	51346
10's-complement of subtrahend	=	93062 +
	=	1 44408

Carry
Cancel

Here, an end carry occurs, hence cancel it.

The result of $(51346 - 06938)_{10}$ is $(44408)_{10}$.

Example: Find the subtraction $(06938 - 51346)_{10}$ using the 10's-complement method.

Solution:

Minuend = 06938

Subtrahend = 51346

Minuend	=	06938
10's-complement of subtrahend	=	48654 +
	=	55592

10's-complement of the result 55592 = – 44408

The result of $(06938 - 51346)_{10}$ is $(- 44408)_{10}$.

Example: Find the subtraction $(1110101 - 1001101)_2$ using the 2's-complement method.

Solution:

Minuend = 1110101

Subtrahend = 1001101

11 111 ← carry

Minuend	=	1110101
2's-complement of subtrahend	=	0110011 +
	=	1 0101000

Carry
Cancel

Here, an end carry occurs, hence cancel it.
The result of $(1110101 - 1001101)_2$ is $(0101000)_2$.

Example: Find the subtraction $(1001101 - 1110101)_2$ using the 2's-complement method.

Solution:

Minuend = 1001101

Subtrahend = 1110101

Minuend	=	1001101	
2's-complement of subtrahend	=	0001011	+
	=	1011000	
2's-complement of the result 1011000	=	– 0101000	

The result of $(1001101 - 1110101)_2$ is $(- 0101000)_2$.

2) Subtraction with 9's-Complement and 1's-Complement

In the subtraction of digital systems it is assumed that both the numbers for the subtraction operation are positive. With the help of 9's-complment or 1's-complement the subtraction operation is performed in the following procedure:

• Obtain the 9's-complement or 1's-complement of the subtrahend and add it to the minuend.

• Verify the result (sum), and observe the carry. If there is an end carry, add 1 to lsd of the result, which is referred to as end around carry. If there is no end carry, obtain 9's-complement or 1'scomplement of the result and put a minus (–) sign before it. The examples of both cases are given below.

Example: Find the subtraction $(72532 - 03250)_{10}$ using the 9's-complement method.

Solution:

Minuend = 72532

Subtrahend = 03250

Minuend	=	72532
9's-complement of subtrahend	=	96749 +
	=	1 69281
End around carry		1 +
		69282

The result of $(72532 - 03250)_{10}$ is $(69282)_{10}$.

Example: Find the subtraction $(06938 - 51346)_{10}$ using the 9's-complement method.

Solution:

Minuend = 06938

Subtrahend = 51346

Minuend	=	06938
9's-complement of subtrahend	=	48653 +
	=	55591

9's-complement of the result 55591 = – 44408

The result of $(06938 - 51346)_{10}$ is $(- 44408)_{10}$.

Example: Find the subtraction $(1010100 - 1000011)_2$ using the 1's-complement method.

Solution:

Minuend = 1010100

Subtrahend = 1000011

Minuend	=	1010100
1's-complement of subtrahend	=	0111100 +
	=	1 0010000
End around carry		1 +
		0010001

The result of $(1010100 - 1000011)_2$ is $(0010001)_2$.

Example: Find the subtraction $(1000011 - 1010100)_2$ using the 1's-complement method.

Solution:

Minuend = 1000011

Subtrahend = 1010100

Minuend	=	1000011
1's-complement of subtrahend	=	0101011 +
	=	1101110
1's-complement of the result 1101110	=	– 0010001

The result of $(1000011 - 1010100)_2$ is $(- 0010001)_2$.

❖ Signed Magnitude of Binary Numbers

Digital circuitry requires both positive and negative numbers. A signed binary number consists of a sign, either positive or negative and magnitude. In a signed magnitude representation of binary numbers, the most significant digit is **zero** for the representation of positive binary number and **one** for the representation of negative binary numbers. This msd (0 or 1) represents whether the number is positive or negative and the magnitude is the value of the numbers. There are three methods by which a signed number can be represented i.e. **signed magnitude**, **1's-complement** and **2's-complement**. Digital computers store negative binary numbers in the form of 2's-complement. Of the three methods mentioned above, for the representation of signed binary numbers, 2's-complement method is most widely used and sign magnitude is rarely used.

As an example, consider the number 9, represented in binary with eight bits. +9 is represented with a sign bit of 0 in the leftmost position, followed by the binary equivalent of 9, which gives 00001001, And there are three different ways to represent -9 with eight bits:

signed-magnitude representation: 10001001

signed-1's-complement representation: 11110110

signed-2's-complement representation: 11110111

❖ Binary Codes

code is a system of rules to convert information- such as a letter, word, sound, image, or gesture- into another form or representation, sometimes shortened or secret, for use it to security reasons.

There are many types of codes: -

1) Binary-Coded-Decimal Code (BCD – 8421) (Weighted Code)

If each digit of a decimal number is represented by its binary equivalent, the result is a code called binary-coded-decimal (hereafter abbreviated BCD). Since a decimal digit can be represented from (0 – 9), four bits are required to code each digit (the binary code for 9 is 1001).

Table below gives the four-bit code for one decimal digit.

Decimal Symbol	BCD Digit (8421)
0	0000
1	0001
2	0010
3	0011
4	0100
5	0101
6	0110
7	0111
8	1000
9	1001

To illustrate the BCD code, take a decimal number such as 874. Each digit is changed to its binary equivalent as follows:

8	7	4	(decimal)
↓	↓	↓	
1000	0111	0100	(BCD)

As another example, let us change 943 to its BCD-code representation:

9	4	3	(decimal)
↓	↓	↓	
1001	0100	0011	(BCD)

Example: Convert 0110100000111001 (BCD) to its decimal equivalent.

Solution: Divide the BCD number into four-bit groups and convert each to decimal.

0110	1000	0011	1001
6	8	3	9

2) Gray Code (non-weighted code)

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

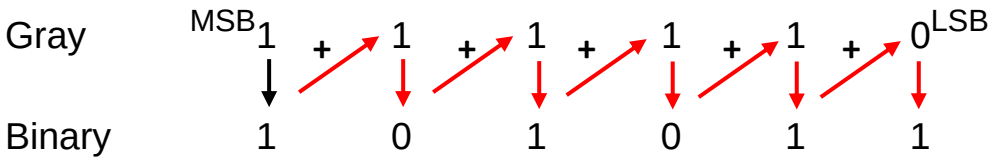
Example: Convert $(101011)_2$ to Gray code.

Solution:

	MSB					LSB
Binary	1	0	1	0	1	1
	↓	↓	↓	↓	↓	↓
Gray	1	1	1	1	1	0

Example: Convert Gray code (111110) to Binary.

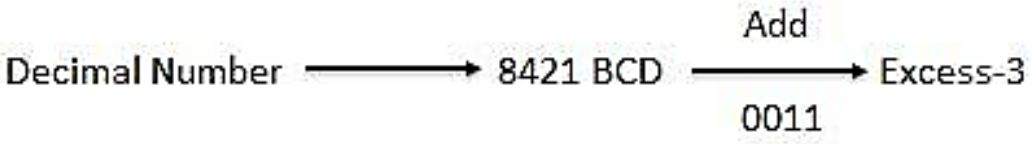
Solution:



Decimal	Binary Code (input)	Gray Code (output)
0	0000	0000
1	0001	0001
2	0010	0011
3	0011	0010
4	0100	0110
5	0101	0111
6	0110	0101
7	0111	0100
8	1000	1100
9	1001	1101
10	1010	1111
11	1011	1110
12	1100	1010
13	1101	1011
14	1110	1001
15	1111	1000

3) Excess-3 code (non-weighted code)

The Excess-3 code is also called as XS-3 code. It is non-weighted code used to express decimal numbers. The Excess-3 code words are derived from the 8421 BCD code words adding $(0011)_2$ or $(3)_{10}$ to each code word in 8421. The excess-3 codes are obtained as follows: -



Example: Represent $(23)_{10}$ in XS-3 code .

Solution:

	2	3
BCD	0010	0011
+	0011	0011
XS-3	0101	0110

4) Alphanumeric codes

A binary digit or bit can represent only two symbols as it has only two states '0' or '1'. But this is not enough for communication between two computers because there we need many more symbols for communication. These symbols are required to represent 26 alphabets with capital and small letters, numbers from 0 to 9, punctuation marks and other symbols.

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

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

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

Binary Logic Gates

The basic building blocks of a computer are called ***logical gates***. Gates are basic circuits that have at least one (and usually more) ***input*** and exactly one output. Input and output values are the logical values ***true*** and ***false***. In computer architecture it is common to use 0 for false and 1 for true. Gates have no memory. The value of the output depends only on the current value of the inputs. A useful way of describing the relationship between the inputs of gates and their output is the truth table. In a truth table, the value of each output is tabulated for every possible combination of the input values. We usually consider three basic kinds of gates, ***AND***-gates, ***OR***-gates, and ***NOT***-gates (or ***Inverters***).

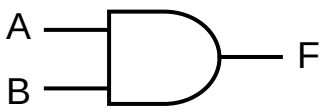
❖ Basic Gates

1) AND Gate: -

The AND operation is represented by a dot(**.**) or by the absence of an operator.

E.g. A.B=F AB=F are all read as A AND B=F. the logical operation AND is interpreted to mean that F=1 if A=1 and B=1 otherwise F=0

Symbol



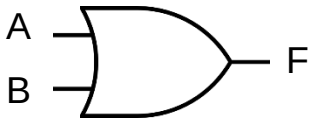
Truth Table

A	B	F = (A.B)
0	0	**0**
0	1	**0**
1	0	**0**
1	1	**1**

2) OR Gate: -

The OR operation is represented by a (+) sign for example, A+B=F is interpreted as A OR B=F meaning that F=1 if A=1 or B=1 or if both A=1 and B=1. If both A and B are 0, then F=0.

Symbol



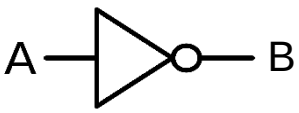
Truth Table

A	B	F = (A+B)
0	0	**0**
0	1	**1**
1	0	**1**
1	1	**1**

3) NOT Gate or INVERTER: -

The inverter is a little different from AND and OR gates in that it always has exactly one input as well as one output. Whatever logical state is applied to the input, the opposite state will appear at the output. This operation is represented by a bar or a prime. For example, $A' = \overline{A} = B$ is interpreted as NOT A =B.

Symbol



Truth Table

A	B
0	1
1	0

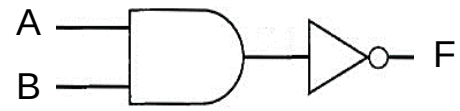
❖ Combined gates

Sometimes, it is practical to combine functions of the basic gates into more complex gates, for instance in order to save space in circuit diagrams. In this section, we show some such combined gates together with their truth tables.

1) NAND Gate: -

The NAND-gate is an AND-gate with an inverter on the output. This operation is represented by $F=\overline{A.B}$. So instead of drawing several gates like this:

Symbol



We draw a single NAND-gate with a little ring on the output like this:

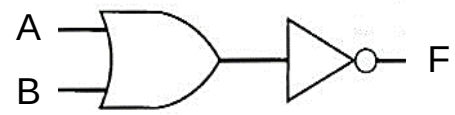
Truth Table

A	B	$F = (\overline{A.B})$
0	0	**1**
0	1	**1**
1	0	**1**
1	1	**0**

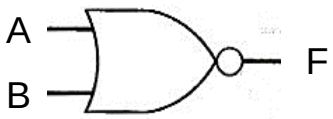
2) NOR Gate: -

The NOR-gate is an OR-gate with an inverter on the output. This operation is represented by $F=\overline{A+B}$. So instead of drawing several gates like this:

Symbol



We draw a single NOR-gate with a little ring on the output like this:



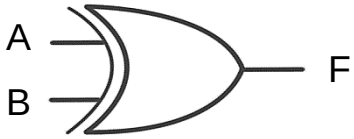
Truth Table

A	B	$F = (\overline{A+B})$
0	0	**1**
0	1	**0**
1	0	**0**
1	1	**0**

3) Exclusive OR Gate(Ex-OR/ XOR): -

The Exclusive-OR-Gate is similar to an OR-gate. It can have an arbitrary number of inputs, and its output value is 1 if exactly one input is 1 (and thus the others 0). Otherwise, the output is 0. This operation is represented by $F=A.\bar{B}+\bar{A}.B = A \oplus B$.

Symbol



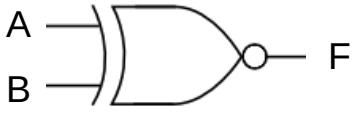
Truth Table

A	B	F = A⊕B
0	0	**0**
0	1	**1**
1	0	**1**
1	1	**0**

4) Exclusive NOR Gate(Ex-NOR/ XNOR): -

The Exclusive-NOR-Gate is similar to an NOR-gate. It can have an arbitrary number of inputs, and its output value is 1 if the two inputs are of the same values (1 and 1 or 0 and 0). Otherwise, the output is 0. This operation is represented by $F=A.B+\bar{A}.\bar{B} = A \odot B$.

Symbol



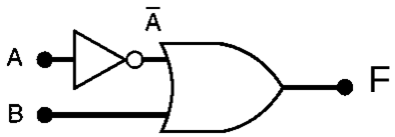
Truth Table

A	B	F = A⊙B
0	0	**1**
0	1	**0**
1	0	**0**
1	1	**1**

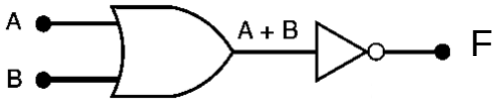
Examples: Draw the following functions?

1) $F = \overline{A}+B$ $\quad F = \overline{A+B}$

Solution:

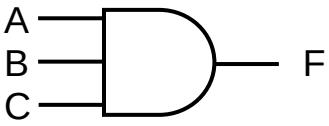


Solution:



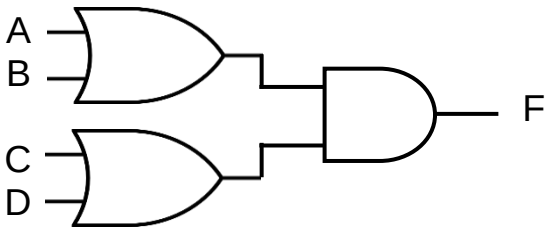
2) $F = A.B.C$

Solution:



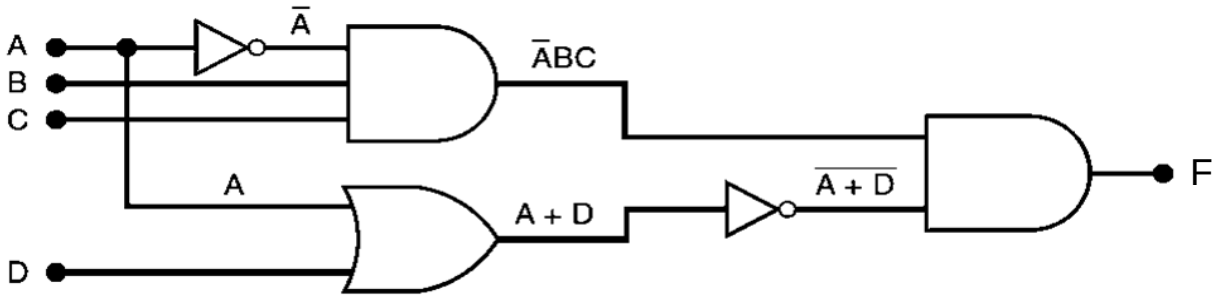
3) $F = A+B.C+D$

Solution:



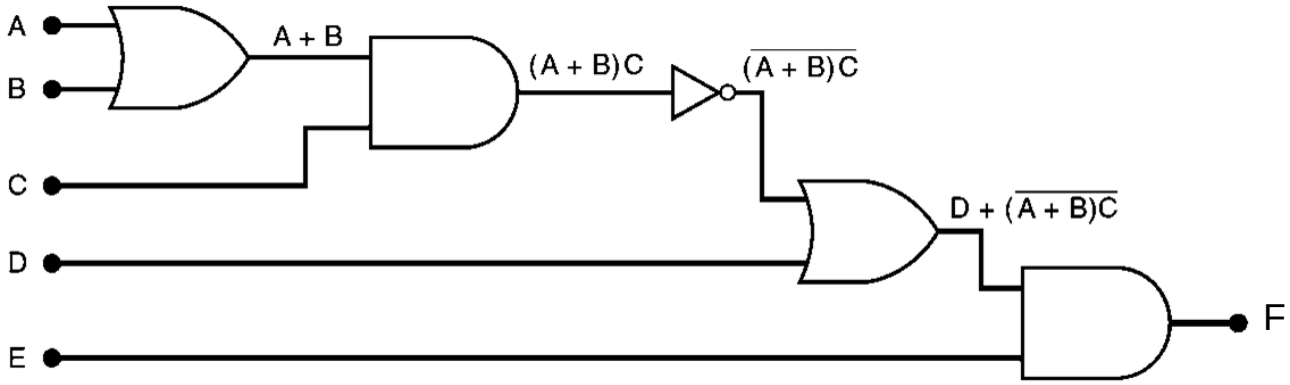
4) $F = \overline{A}BC(\overline{A + D})$

Solution:



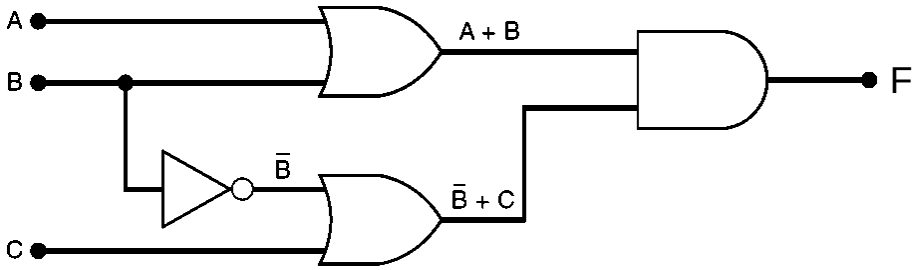
5) $F = [D + (\overline{A + B)\,C}] \bullet E$

Solution:

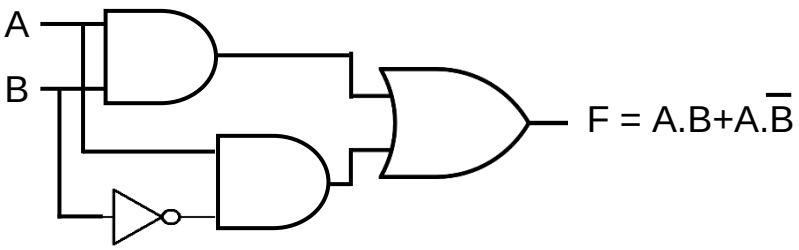


6) $F = (A + B)(\overline{B} + C)$

Solution:



Example: Find the truth table of F?



Solution:

A	B	$\overline{B}$	AB	$A\overline{B}$	F
0	0	**1**	**0**	**0**	**0**
0	1	**0**	**0**	**0**	**0**
1	0	**1**	**0**	**1**	**1**
1	1	**0**	**1**	**0**	**1**

Boolean Algebra

Boolean Algebra is an algebra, which deals with binary numbers & binary variables. Hence, it is also called as Binary Algebra or logical Algebra. A mathematician, named George Boole had developed this algebra in 1854. The variables used in this algebra are also called as Boolean variables.

Boolean Algebra is set of rules, used to simplify the given logic expression without changing its functionality.

❖ Postulates and Basic Laws of Boolean Algebra

In this section, let us discuss about the Boolean postulates and basic laws that are used in Boolean algebra. These are useful in minimizing Boolean functions.

– Boolean Postulates

Consider the binary numbers 0 and 1, Boolean variable (x) and its complement (x'). Either the Boolean variable or complement of it is known as literal. The four possible logical OR and logical AND operations among these literals and binary numbers are shown below.

AND	OR
x.1 = x	x + 0 = x
x.0 = 0	x + 1 = 1
x.x = x	x + x = x
x.x′ = 0	x + x′ = 1

These are the simple Boolean postulates. We can verify these postulates easily, by substituting the Boolean variable with '0' or '1'.

Note− The complement of complement of any Boolean variable is equal to the variable itself. i.e., (x')'=x.

– Basic Laws of Boolean Algebra

Following are the three basic laws of Boolean Algebra.

- Commutative law
- Associative law
- Distributive law

- **Commutative Law**

If any logical operation of two Boolean variables give the same result irrespective of the order of those two variables, then that logical operation is said to be Commutative. The logical OR & logical AND operations of two Boolean variables x & y are shown below

$$x + y = y + x$$

$$x.y = y.x$$

- **Associative Law**

If a logical operation of any two Boolean variables is performed first and then the same operation is performed with the remaining variable gives the same result, then that logical operation is said to be Associative. The logical OR & logical AND operations of three Boolean variables x, y & z are shown below.

$$x + (y + z) = (x + y) + z$$

$$x.(y.z) = (x.y).z$$

- **Distributive Law**

If any logical operation can be distributed to all the terms present in the Boolean function, then that logical operation is said to be Distributive. The distribution of logical OR & logical AND operations of three Boolean variables x, y & z are shown below.

$$x.(y + z) = x.y + x.z$$

$$x + (y.z) = (x+y).(x+z)$$

These are the Basic laws of Boolean algebra. We can verify these laws easily, by substituting the Boolean variables with '0' or '1'.

❖ Theorems of Boolean Algebra

The following two theorems are used in Boolean algebra.

- Duality theorem
- DeMorgan's theorem

❖ Duality Theorem

This theorem states that the dual of the Boolean function is obtained by interchanging the logical AND operator with logical OR operator and zeros with ones. For every Boolean function, there will be a corresponding Dual function.

Let us make the Boolean equations (relations) that we discussed in the section of Boolean postulates and basic laws into two groups. The following table shows these two groups.

Group1	Group2
x + 0 = x	x.1 = x
x + 1 = 1	x.0 = 0
x + x = x	x.x = x
x + x' = 1	x.x' = 0
x + y = y + x	x.y = y.x
x + (y + z) = (x + y) + z	x.(y.z) = (x.y).z
x.(y + z) = x.y + x.z	x + (y.z) = (x+y).(x+z)

Note:- x+x'y = x+(x')y = x+y (cancel)

x'+xy = x'+(x)y = x'+y (cancel)

❖ DeMorgan's Theorem

This theorem is useful in finding the **complement of Boolean function**. It states that the complement of logical OR of at least two Boolean variables is equal to the logical AND of each complemented variable.

DeMorgan's theorem with 2 Boolean variables x and y can be represented as

$$(x + y)' = x'.y'$$

The dual of the above Boolean function is

$$(x.y)' = x' + y'$$

Therefore, the complement of logical AND of two Boolean variables is equal to the logical OR of each complemented variable. Similarly, we can apply DeMorgan's theorem for more than 2 Boolean variables also.

Examples:- Find the complement and dual of the following functions.

Examples	complement	dual
1) F = x'yz' + x'y'z	= (x'yz' + x'y'z)' = (x'yz')'.(x'y'z)' = (x + y' + z)(x + y + z')	= (x' + y + z')(x' + y' + z)
2) F = x(y'z' + yz)	= [x(y'z' + yz)]' = x' + (y'z' + yz)' = x' + (y'z')'(yz)' = x' + (y + z)(y' + z')	= x + (y' + z')(y + z)
3) F = A+B+C+D	= A'.B'.C'.D'	= A.B.C.D

❖ Simplification of Boolean Functions

Till now, we discussed the postulates, basic laws and theorems of Boolean algebra. Now, let us simplify some Boolean functions.

Examples: simplify the following Boolean functions.

1) F = AB + AB' = A.(B+B')

= A.1 = A

2) F = AB + AB'C + AB'C'

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

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

= A(B+C+C')

= A(B+1)

= A.1 = A

A	B	C	B'	C'	AB	AB'C	AB'C'	F
0	0	0	1	1	0	0	0	0
0	0	1	1	0	0	0	0	0
0	1	0	0	1	0	0	0	0
0	1	1	0	0	0	0	0	0
1	0	0	1	1	0	0	1	1
1	0	1	1	0	0	1	0	1
1	1	0	0	1	1	0	0	1
1	1	1	0	0	1	0	0	1

∴ F = A

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

= AA+AC+AB+BC

= A+AC+AB+BC

= A(1+C)+AB+BC

= A+AB+BC

= A(1+B)+BC

= A+BC

4) F = (A+B)(A+B')(A'+B)(A'+B')

= (A+BB')(A'+BB')

= (A+0)(A'+0)

= A.A'

= 0

5) F = AC'+ABC+ACD'+CD

= A(C'+BC)+C(AD'+D)

= A(C'+B)+C(A+D)

= AC'+AB+AC+CD

= A(C'+C)+AB+CD

= A+AB+CD

= A(1+B)+CD

= A+CD

6) F = AB+A'BC'+A'BC

= AB+A'B(C'+C)

= AB+A'B

= B(A+A')

= B

A	B	C	A'	C'	AB	A'BC'	A'BC	F
0	0	0	1	1	0	0	0	0
0	0	1	1	0	0	0	0	0
0	1	0	1	1	0	1	0	1
0	1	1	1	0	0	0	1	1
1	0	0	0	1	0	0	0	0
1	0	1	0	0	0	0	0	0
1	1	0	0	1	1	0	0	1
1	1	1	0	0	1	0	0	1

$\therefore$ F = B

Canonical and Standard Forms

Logical functions are generally expressed in terms of different combinations of logical variables with their true forms as well as the complement forms. Binary logic values obtained by the logical functions and logic variables are in binary form. An arbitrary logic function can be expressed in the following forms:

1) Sum of Products (SOP)
2) Product of Sums (POS)

Sum of Products (SOP): In Boolean algebra, The logical sum of two or more logical product terms is referred to as a sum of products expression. It is basically an OR operation on AND operated variables.

For example: Y = AB + BC + AC or Y = A′B + BC + AC′ are sum of products expressions.

Product of Sums (POS): Similarly, the logical product of two or more logical sum terms is called a product of sums expression. It is an AND operation on OR operated variables.

For example: Y = (A + B + C)(A + B′ + C)(A + B + C′) or Y = (A + B + C)(A′ + B′ + C′) are product of sums expressions.

❖ **Standard form:**

The standard form of the Boolean function is when it is expressed in sum of the products or product of the sums fashion. The examples stated above, like Y =AB + BC + AC or Y = (A + B + C)(A + B′ + C)(A + B + C′) are the standard forms.

However, Boolean functions are also sometimes expressed in nonstandard forms like F = (AB + CD)(A′B′ + C′D′), which is neither a sum of products form nor a product of sums form. However, the same expression can be converted to a standard form with help of various Boolean properties, as:

F = (AB + CD)(A′B′ + C′D′) = A′B′CD + ABC′D′

❖ Canonical Sum of Product Expression (Minterm)

A product term containing all n variables of the function in either true or complemented form is called the minterm. Each minterm is obtained by an AND operation of the variables in their true form or complemented form. For a two-variable function, four different combinations are possible, such as, A′B′, A′B, AB′, and AB. These product terms are called the fundamental products or standard products or minterms. In the minterm, a variable will possess the value 1 if it is in true or uncomplemented form, whereas, it contains the value 0 if it is in complemented form. For three variables function, eight minterms are possible as listed in the following table:

A	B	C	Minterms	designation
0	0	0	A'B'C'	m_0
0	0	1	A'B'C	m_1
0	1	0	A'BC'	m_2
0	1	1	A'BC	m_3
1	0	0	AB'C'	m_4
1	0	1	AB'C	m_5
1	1	0	ABC'	m_6
1	1	1	ABC	m_7

When a Boolean function is expressed as the logical sum of all the minterms from the rows of a truth table, for which the value of the function is 1, it is referred to as the canonical sum of product expression. The same can be expressed in a compact form by listing the corresponding decimal-equivalent codes of the minterms containing a function value of 1. For example, if the canonical sum of product form of a three-variable logic function F has the minterms A′BC, AB′C, and ABC′, this can

be expressed as the sum of the decimal codes corresponding to these minterms as below:

$$F(A,B,C) = \Sigma(3,5,6)$$
$$= m_3 + m_5 + m_6$$
$$= A'BC + AB'C + ABC'$$

where Σ (3,5,6) represents the summation of minterms corresponding to decimal codes 3, 5, and 6.

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

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

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

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

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

$$F(A, B) = A + B$$
$$= A.1 + B.1$$
$$= A(B + B') + B(A + A')$$
$$= AB + AB' + AB + A'B$$
$$= AB + AB' + A'B \qquad \text{(as } AB + AB = AB\text{)}$$

$\therefore$ F (A, B) = AB + AB′ + A′B

= $m_3 + m_2 + m_1$

= Σ (1,2,3)

A	B	Minterms	designation
0	0	A'B'	m_0
0	1	A'B	m_1
1	0	AB'	m_2
1	1	AB	m_3

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

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

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

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

= A (B + B′) (C + C′) + BC (A + A′)

= (AB + AB′) (C + C′) + ABC + A′BC

= ABC + AB′C + ABC′ + AB′C′ + ABC + A′BC

= ABC + AB′C + ABC′ + AB′C′ + A′BC (as ABC + ABC = ABC)

$\therefore$ F (A,B,C) = ABC + AB′C + ABC′ + AB′C′ + A′BC.

= $m_7 + m_5 + m_6 + m_4 + m_3$

= Σ (3,4,5,6,7)

A	B	C	Minterms	designation
0	0	0	A'B'C'	m_0
0	0	1	A'B'C	m_1
0	1	0	A'BC'	m_2
0	1	1	A'BC	m_3
1	0	0	AB'C'	m_4
1	0	1	AB'C	m_5
1	1	0	ABC'	m_6
1	1	1	ABC	m_7

❖ Canonical Product of Sum Expression (Maxterm)

A sum term containing all n variables of the function in either true or complemented form is called the maxterm. Each maxterm is obtained by an OR operation of the variables in their true form or complemented form. Four different combinations are possible for a two-variable function, such as, A + B, A + B', A' + B, and A' + B'. These sum terms are called the standard sums or maxterms. Note that, in the maxterm, a variable will possess the value 0, if it is in true or uncomplemented form, whereas, it contains the value 1, if it is in complemented form. Like minterms, for a three-variable function, eight maxterms are also possible as listed in the following table:

A	B	C	Maxterms	designation
0	0	0	A+B+C	M_0
0	0	1	A+B+C'	M_1
0	1	0	A+B'+C	M_2
0	1	1	A+B'+C'	M_3
1	0	0	A'+B+C	M_4
1	0	1	A'+B+C'	M_5
1	1	0	A'+B'+C	M_6
1	1	1	A'+B'+C'	M_7

When a Boolean function is expressed as the logical product of all the maxterms from the rows of a truth table, for which the value of the function is 0, it is referred to as the canonical product of sum expression. The same can be expressed in a compact form by listing the corresponding decimal equivalent codes of the maxterms containing a function value of 0. For example, if the canonical product of

sums form of a three-variable logic function F has the maxterms A + B + C, A + B′ + C, and A′ + B + C′, this can be expressed as the product of the decimal codes corresponding to these maxterms as below:

$$F(A,B,C) = \Pi(0,2,5)$$
$$= M_0 M_2 M_5$$
$$= (A + B + C)(A + B' + C)(A' + B + C')$$

where $\Pi(0,2,5)$ represents the product of maxterms corresponding to decimal codes 0, 2, and 5.

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

1) Check each term in the given logic function. Retain it if it is a maxterm, continue to examine the next term in the same manner.

2) Examine for the variables that are missing in each sum term that is not a maxterm.

If the missing variable in the maxterm is X, add that maxterm with (X.X′).

3) Expand the expression using the properties and postulates as described earlier and discard the redundant terms.

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

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

Solution: In the above three-variable expression, C is missing from the first term, A is missing from the second term, and B is missing from the third term. Therefore, CC′ is to be added with first term, AA′ is to be added with the second, and BB′ is to be added with the third term. This is shown below:

$$F(A,B,C) = (A + B')(B + C)(A + C')$$
$$= (A + B' + 0)(B + C + 0)(A + C' + 0)$$
$$= (A + B' + CC')(B + C + AA')(A + C' + BB')$$
$$= (A + B' + C)(A + B' + C')(A + B + C)(A' + B + C)(A + B + C')(A + B' + C')$$

[using the distributive property, as X + YZ = (X + Y)(X + Z)]

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

[as (A + B′ + C′) (A + B′ + C′) = A + B′ + C′]

∴ F (A, B, C) = (A + B′ + C) (A + B′ + C′) (A + B + C) (A′ + B + C) (A + B + C′)

= M_2 . M_3 . M_0 . M_4 . M_1

= ΠT (0,1,2,3,4)

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

F (A, B, C) = A + B′C

Solution: In the above three-variable expression, the function is given at sum of the product form. First, the function needs to be changed to product of the sum form by applying the distributive law as shown below:

F (A, B, C) = A + B′C

= (A + B′) (A + C)

Now, in the above expression, C is missing from the first term and B is missing from the second term. Hence CC′ is to be added with the first term and BB′ is to be added with the second term as shown below.

F (A, B, C) = (A + B′) (A + C)

= (A + B′ + CC′) (A + C + BB′)

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

[using the distributive property, as X + YZ = (X + Y) (X + Z)]

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

[as (A + B′ + C) (A + B′ + C) = A + B′ + C]

∴ F (A, B, C) = (A + B′ + C) (A + B′ + C′) (A + B + C)

Example: Represent (SOP) and (POS) expressions by using the following truth table:

A	B	C	F	SOP	POS
0	0	0	0		A+B+C
0	0	1	0		A+B+C'
0	1	0	1	A'BC'	
0	1	1	0		A+B'+C'
1	0	0	1	AB'C'	
1	0	1	1	AB'C	
1	1	0	1	ABC'	
1	1	1	0		A'+B'+C'

SOP:-

F = A'BC'+ AB'C'+AB'C+ ABC'

F = ∑ (2,4,5,6)

POS:-

F = (A+B+C) (A+B+C') (A+B'+C') (A'+B'+C')

F = ΤΤ (0,1,3,7)

Example: Simplify the expression for F(A,B) = ∑ (0,2,3)

Solution: F = $m_0 + m_2 + m_3$

= A'B'+AB'+AB

= B'(A'+A)+AB

= B'+AB

= B'+A

Karnaugh Map (K- Map) Method

In previous Lectures, we have simplified the Boolean functions using Boolean postulates and theorems. It is a time consuming process and we have to re-write the simplified expressions after each step.

To overcome this difficulty, Karnaugh introduced a method for simplification of Boolean functions in an easy way. This method is known as Karnaugh map method or K-map method. It is a graphical method, which consists of 2^n cells for 'n' variables. The adjacent cells are differed only in single bit position.

– K- Map & Minimization Steps:-

Step 1: generate K-map

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

Step 2: group all adjacent 1s without including any 0s

- ✓ All groups (prime implicants) must be rectangular and contain a "power-of-2" number of 1s.
  - 1, 2, 4, 8, 16, 32, …
- ✓ An essential group (essential prime implicant) contains at least 1 minterm not included in any other groups.
  - A given minterm may be included in multiple groups.

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

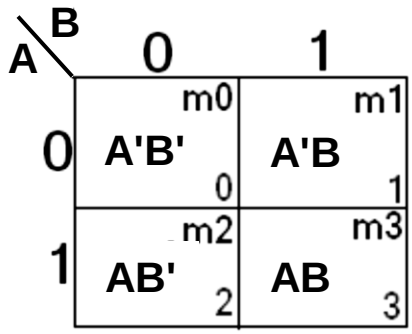
Step 4: sum all essential groups plus a minimal set of remaining groups to obtain a minimum SOP.

– K-Maps for 2 to 5 Variables:-

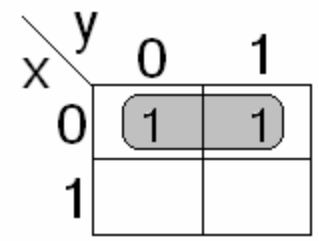
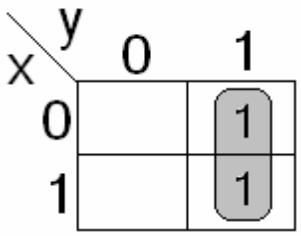
K-Map method is most suitable for minimizing Boolean functions of 2 variables to 5 variables. Now, let us discuss about the K-Maps for 2 to 5 variables one by one.

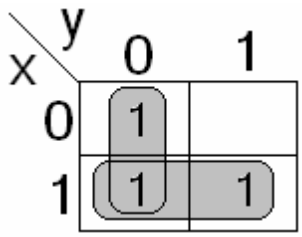
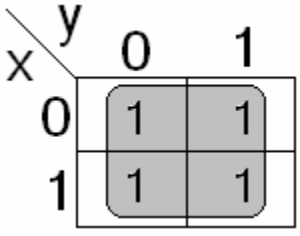
- **Two-Variable K-Map**

The two-variable map is shown below. There are four minterms for two variables; hence, the map consists of four squares, one for each minterm.



Example: Simplify the following Boolean functions using (K- Map) Method.

1) F(x,y) = x'y + x'y'	2) F(x,y) = xy + x'y
Solution:	Solution:
	
F(x,y) = x'	F(x,y) = y

3) F(x,y) = x'y' + xy' + xy	4) F(x,y) = xy + x'y + xy' + x'y'
Solution:	Solution:
	
F(x,y) = x + y'	F(x,y) = 1

- **Three-Variable K-Map**

A three-variable K-map is shown below. There are eight minterms for three binary variables; therefore, the map consists of eight squares. Note that the minterms are arranged, not in a binary sequence, but in a sequence similar to the Gray code.

The characteristic of this sequence is that **only one bit changes in value from one adjacent column to the next.**

As more adjacent squares are combined, we obtain a product term with fewer literals.

One square represents one minterm, giving a term with three literals.

Two adjacent squares represent a term with two literals.

Four adjacent squares represent a term with one literal.

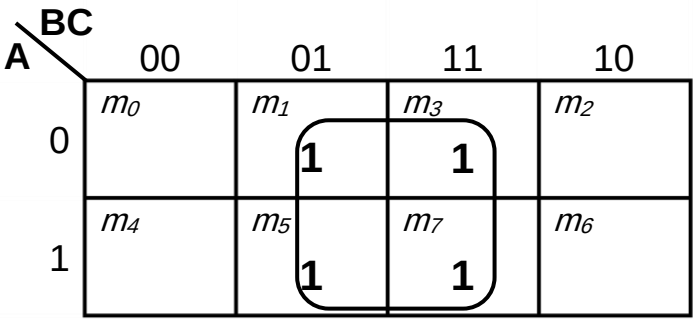
Eight adjacent squares encompass the entire map and produce a function that is always equal to 1.

A \ BC	00	01	11	10
0	m_0 A'B'C'	m_1 A'B'C	m_3 A'BC	m_2 A'BC'
1	m_4 AB'C'	m_5 AB'C	m_7 ABC	m_6 ABC'

Example: Simplify the following Boolean functions using (K- Map) Method.

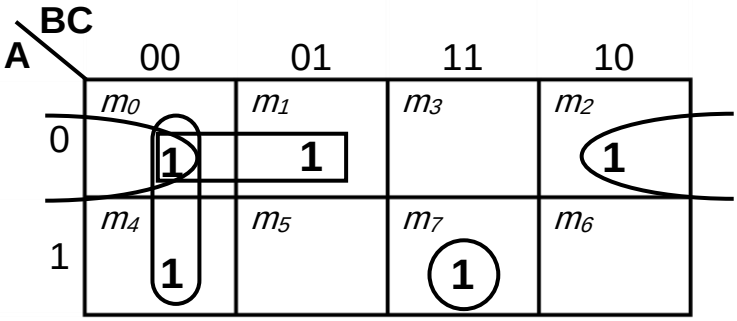
1) F(A,B,C) = ∑ (1,3,5,7)

Solution: F= C



2) F(A,B,C) = ∑ (0,1,2,4,7)

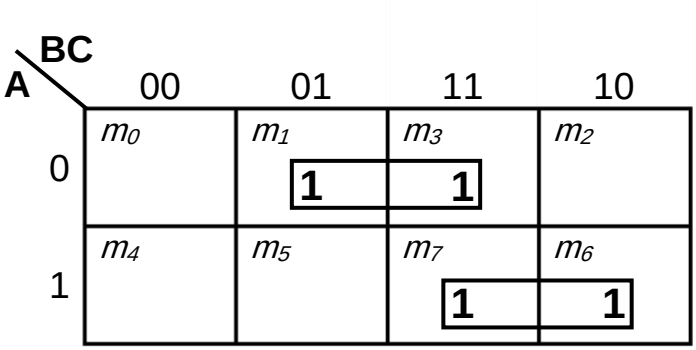
Solution:



F= B'C' + A'B' + ABC + A'C'

3) F(A,B,C) = ∑ (1,3,6,7)

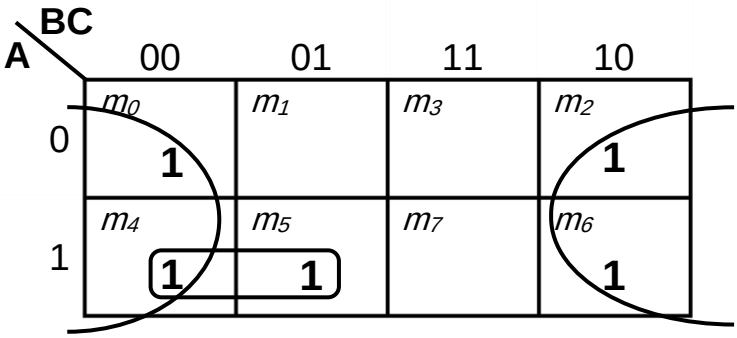
Solution:



F= AB + A'C

4) F(A,B,C) = ∑ (0,2,4,5,6)

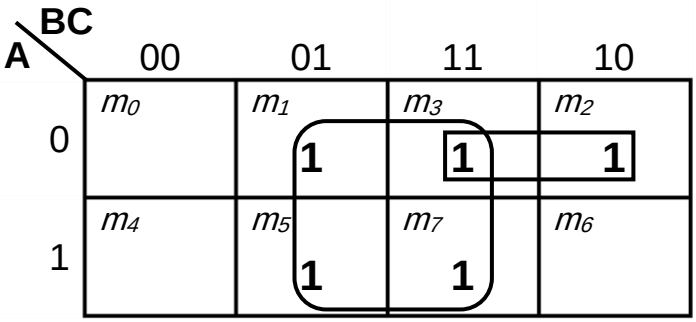
Solution:



F= C' + AB'

5) F = A'B'C + A'BC' + A'BC + AB'C + ABC

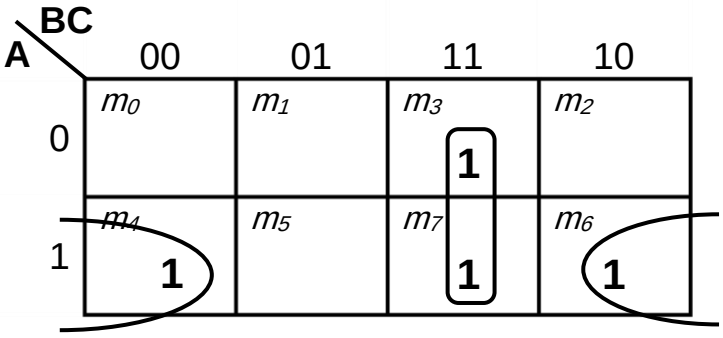
Solution:



F= C + A'B

6) F(A,B,C) = ∑ (3,4,6,7)

Solution:



F= AC' + BC

- **Four-Variable K-Map**

The number of squares in 4 variable K-map is sixteen, since the number of variables is four. The following figure shows 4 variable K-Map. The combination of adjacent squares that is useful during the simplification process is easily determined from inspection of the four-variable map:

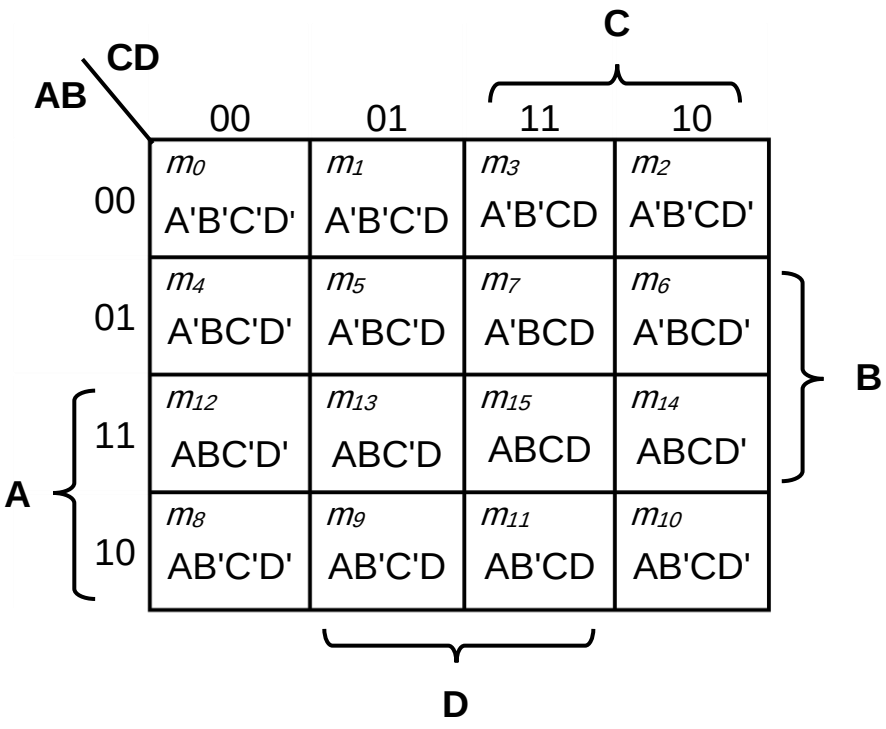
One square represents one minterm, giving a term with four literals.

Two adjacent squares represent a term with three literals.

Four adjacent squares represent a term with two literals.

Eight adjacent squares represent a term with one literal.

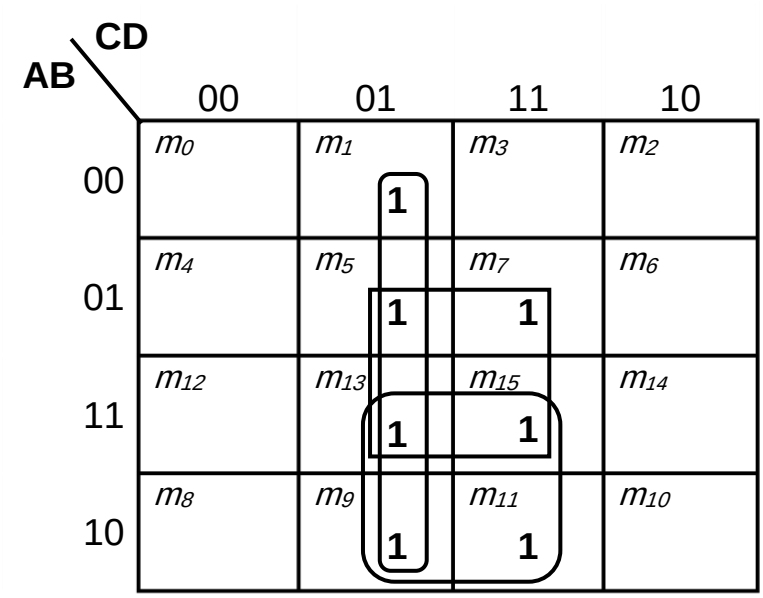
Sixteen adjacent squares produce a function that is always equal to 1.



Example: Simplify the following Boolean functions using (K- Map) Method.

1) F(A,B,C,D) = ∑ (1,5,7,9,11,13,15)

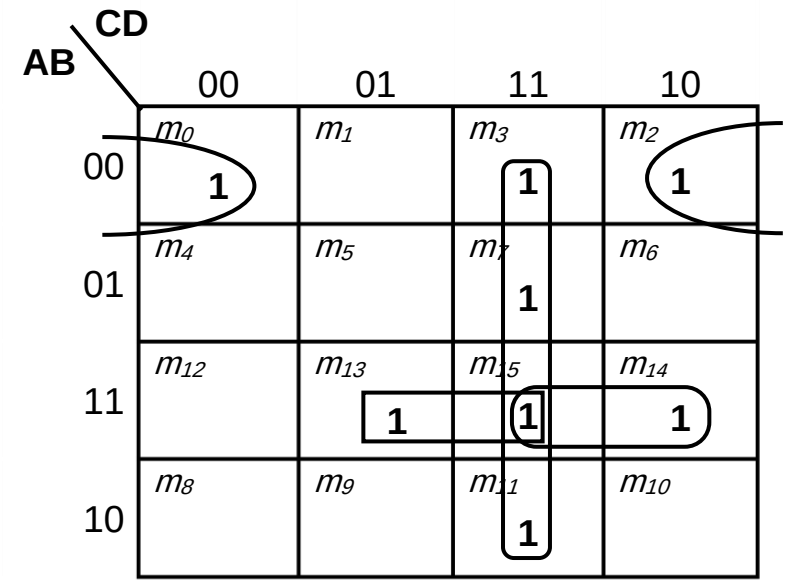
Solution:



F= AD + BD + C'D

2) F(A,B,C,D) = ∑ (0,2,3,7,11,13,14,15)

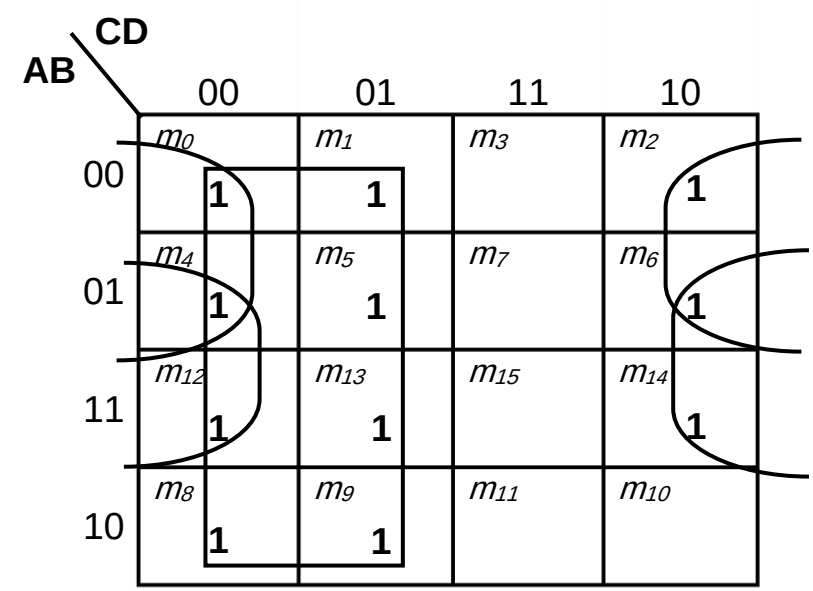
Solution:



F= CD + A'B'D' + ABC + ABD

3) F(A,B,C,D) = ∑ (0, 1, 2, 4, 5, 6, 8, 9, 12, 13, 14)

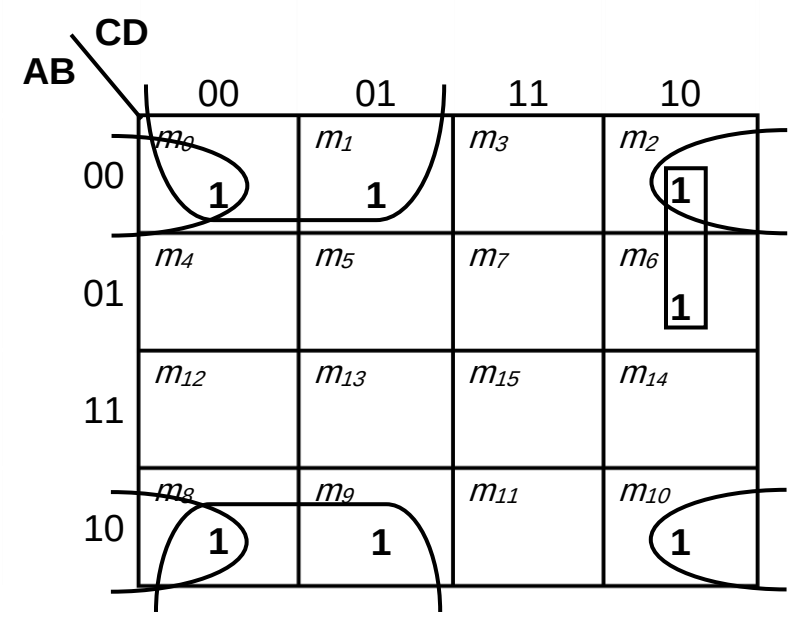
Solution:



F= C' + A'D' + BD'

4) F = A'B'C'D' + A'B'C'D + A'B'CD' +A'BCD' + AB'C'D' + AB'C'D + AB'CD'

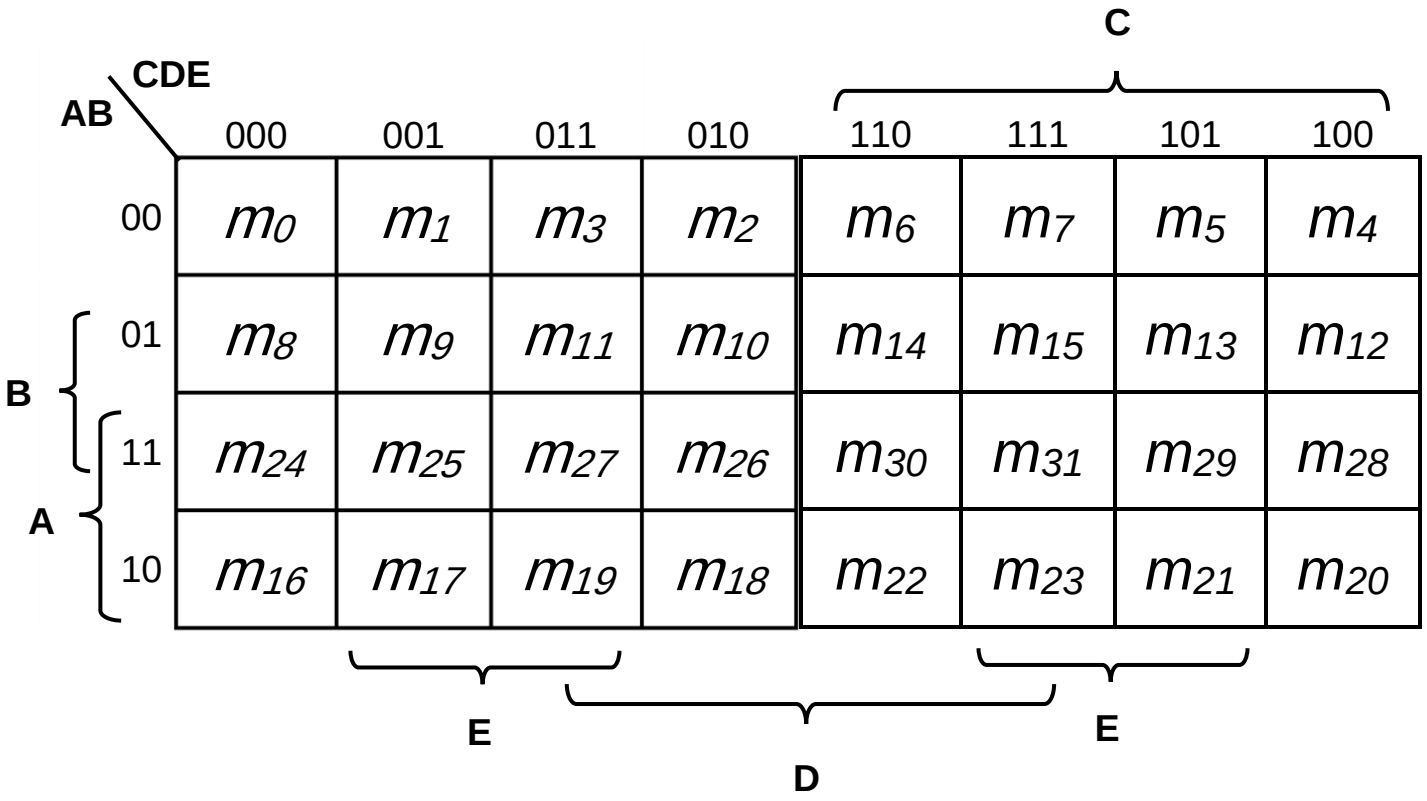
Solution:



F= B'D'+ B'C' + A'CD'

- **Five-Variable K-Map**

Maps for more than four variables are not as simple to use as maps for four or fewer variables. A five-variable map needs 32 squares and a six-variable map needs 64 squares.



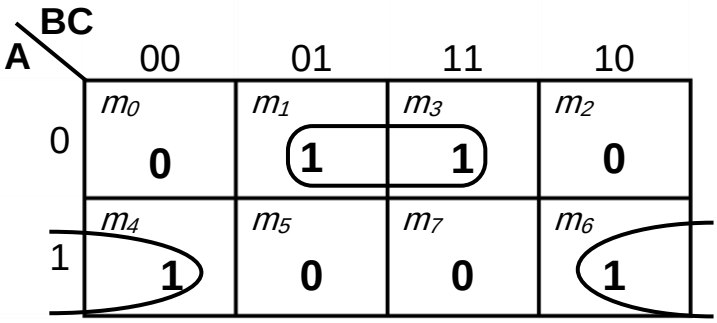
❖ **Product-Of-Sums Simplification**

The procedure for obtaining a minimized function in product-of-sums form follows from the basic properties of Boolean functions. The 1's placed in the squares of the map represent the minterms of the function. The minterms not included in the standard sum-of-products form of a function denote the complement of the function. From this observation, we see that the complement of a function is represented in the map by the squares not marked by 1's. If we mark the empty squares by 0's and combine them into valid adjacent squares, we obtain a simplified sum-of-products expression of the complement of the function (i.e., of F'). The complement of F' gives us back the function F in product-of-sums form (a consequence of

DeMorgan's theorem). Because of the generalized DeMorgan's theorem, the function so obtained is automatically in product-of-sums form.

Example: Simplify the following Boolean function into sum-of-products form and product-of-sums form: F(A,B,C) = ∑ (1, 3, 4, 6)

Solution:



F = A'C + AC' (SOP)

F' = AC + A'C' (POS)

Taking the complement of F', we obtain the simplified function in product-of-sums form:

F = (A' + C')(A + C)

Example: Simplify the following Boolean function into sum-of-products form and product-of-sums form: F(A,B,C,D) = ∑ (0, 1, 2, 5, 8, 9, 10)

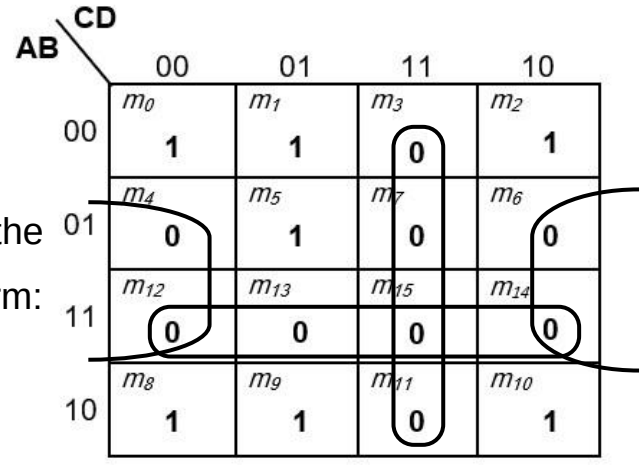
Solution:

F = B'D' + B'C' + A'C'D (SOP)

F' = AB + CD + BD' (POS)

Taking the complement of F', we obtain the simplified function in product-of-sums form:

F = (A' + B')(C' + D')(B' + D)



❖ don't- care conditions

don't- care conditions can be used on a map to provide further simplification of the Boolean expression.

To distinguish the don't-care condition from 1's and 0's, an X or d is used.

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.

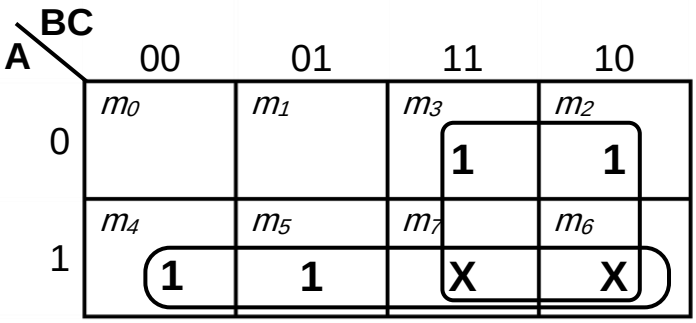
Example: Simplify the following Boolean functions using (K- Map) Method.

1) F(A,B,C) = ∑ (2, 3, 4, 5)

which has the don't-care conditions

d(A,B,C) = ∑ (6,7)

Solution:



F = A + B

2) F(A, B, C, D) = ∑ (1, 3, 7, 11, 15)

which has the don't-care conditions

d(A, B, C, D) = ∑ (0, 2, 5)

Solution:

F = A'B' + CD