Data Representation: Binary Numbers, Fixed Point Representation. Floating Point Representation. Number base conversions, Octal and Hexa decimal Numbers, components, Signed binary numbers, Binary codes.

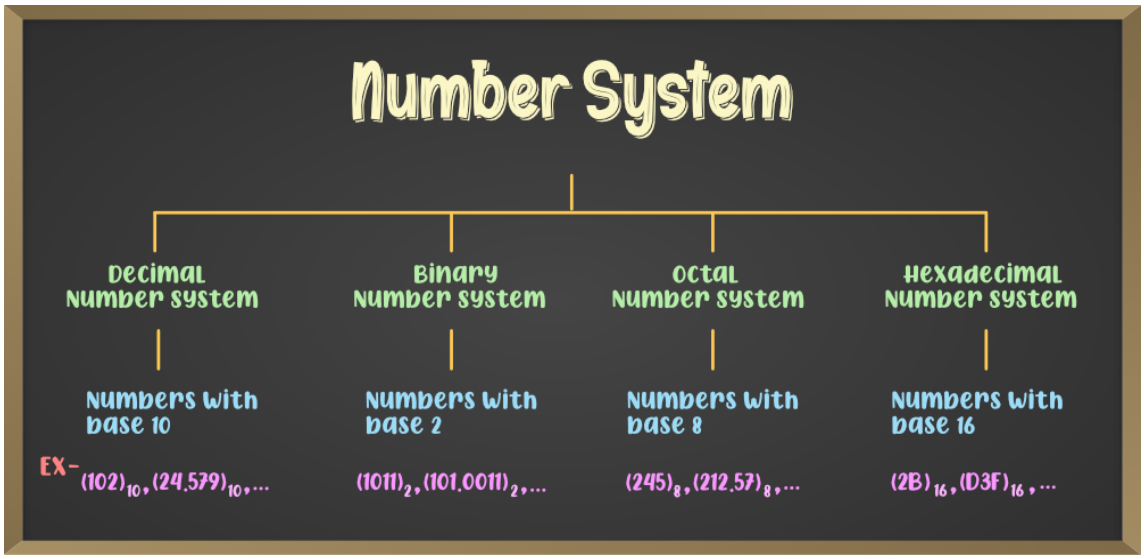
DATA REPRESENTATION

Computer do not understand our natural language such as English, they do not even understand the language that you use to code such as Python or Java.

Just as with a light switch that can be ON or OFF, and with many light switches you can create different combinations of the light being on or off. Computer are not much different, imagine if you had just 8 light switches you can create 256 different combinations with the lights they control. Computers work with the same principle but have billions of combination, each combination representing something different such as the colour of one of the pixels on your screen or a letter on your keyboard.

To make it easy for humans to understand and work with, when a switch in a computer is on we represent this with a 1 and off we represent with a 0. These 1s and 0s are known as binary. The language machines use is called **Machine Language** and different Computer CPUs (Central Processing Units) have different ways of representing language they process.
The data representation section addresses how computers represent different aspect of language, numbers, calculations, colour, images, sound and data compression.

Number system:



There are generally various types of number systems and among them the four major ones are,

- Binary Number System (Number system with Base 2)
- Octal Number System (Number system with Base 8)
- Decimal Number System (Number system with Base 10)
- Hexadecimal Number System (Number system with Base 16)

In the Binary Number System, we have two states “0” and “1” and these two states are represented by two states of a transistor. If the current passes through the transistor then the computer reads “1” and if the current is absent from the transistor then it read “0”.

Each digit in the binary number system is called a “bit”.

A single binary digit is called a Bit, and multiple Bits have the following denotations:

Nibble	-	Four Bits (0000)
Byte	-	Eight Bits (00000000)
Kilobyte	-	1024 bytes
Megabyte	-	1024 Kilobytes
Gigabyte	-	1024 Megabytes
Terabyte	-	1024 Gigabytes
Petabyte	-	1024 Terabytes
Exabyte	-	1024 Petabytes
Zettabyte	-	1024 Exabyte
Yottabyte	-	1024 Zettabyte

Binary Number System (Number system with Base 2):

Binary Number System is the number system in which we use two digits “0” and “1” to perform all the necessary operations. In the Binary Number System, we have a base of 2. The base of the Binary Number System is also called the radix of the number system.

Binary Number Table

Decimal Number	Binary Number	Decimal Number	Binary Number
1	001	11	1011
2	010	12	1100
3	011	13	1101
4	100	14	1110
5	101	15	1111
6	110	16	10000
7	111	17	10001
8	1000	18	10010
9	1001	19	10011
10	1010	20	10100

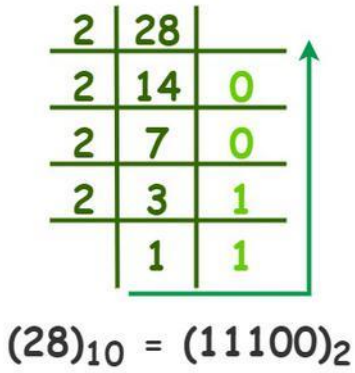
Decimal to Binary Conversion

A decimal number is converted into a binary number by dividing the given decimal number by 2 continuously until we get the quotient as 1, and we write the numbers from downwards to upwards.

Let us go through an example to understand the concept better.

Example1: Convert $(28)_{10}$ into a binary number.

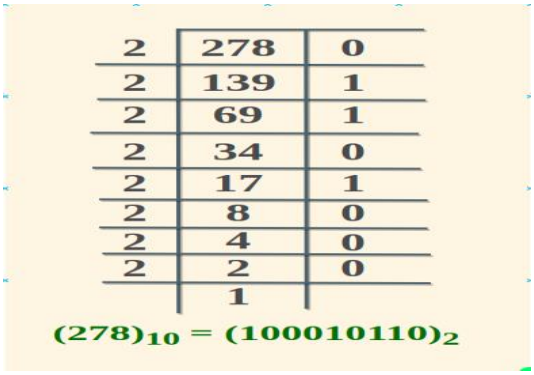
Solution:



Hence, $(28)_{10}$ is expressed as $(11100)_2$.

Example 2: Convert $(278)_{10}$ into a binary number

Solution:



Hence, $(278)_{10}$ in decimal is equivalent to $(100010110)_2$ in binary.

convert floating decimal numbers into binary numbers:

Example1: Convert $(4.25)_{10}$ into a binary number.

Solution: Where,

4 is an integral part.

0.25 is a fractional part.

Using that method, we can represent 4 as $(100)_2$.

0.25 * 2 =0.50 //take 0 and move 0.50 to next step

0.50 * 2 =1.00 //take 1 and stop the process

0.25 = $(01)_2$

Combining both integral and fractional,

4.25 = $(100.01)_2$

Example2: Convert $(10.75)_{10}$ into a binary number.

Solution: Integral Part

10 = $(1010)_2$

0.75 * 2 =>1.50 // take 1 and move .50 to next step

0.50 * 2 =>1.00 // take 1 and stop the process because no remainder

0.75 = $(11)_2$

Combining both integral and fractional,

$10.75 = (1010.11)_2$

Practice Problems on Decimal to Binary Conversion

Q1: Convert 248 in Binary.
Q2: Convert 575 in Binary.
Q3: What is Binary equivalent of 49.
Q4: Convert (56)10 to (….)2.
Q5: What is the Binary Form of 95.
Q6: Convert 290.125 in Binary

Binary to Decimal Conversion:

A binary number is converted into a decimal number by multiplying each digit of the binary number by the power of either 1 or 0 to the corresponding power of 2.

$$D = (a_{n-1} \times 2^{n-1}) + \ldots + (a_3 \times 2^3) + (a_2 \times 2^2) + (a_1 \times 2^1) + (a_0 \times 2^0)$$

Example 1: Convert $(10011)_2$ to a decimal number

Solution: *The given binary number is (10011)2.*

$$(10011)2 = (1 \times 2^4) + (0 \times 2^3) + (0 \times 2^2) + (1 \times 2^1) + (1 \times 2^0)$$
$$= 16 + 0 + 0 + 2 + 1 = (19)10$$

Hence, the binary number $(10011)_2$ is expressed as $(19)_{10}$.

Example 2: Convert $(1110010)_2$ to a decimal number

Solution: $(0 * 2^0) + (1 * 2^1) + (0 * 2^2) + (0 * 2^3) + (1 * 2^4) + (1 * 2^5) + (1 * 2^6)$
$= (0 * 1) + (1 * 2) + (0 * 4) + (0 * 8) + (1 * 16) + (1 * 32) + (1 * 64)$
$= 0 + 2 + 0 + 0 + 16 + 32 + 64$
$= 114$

convert floating binary numbers into decimal numbers:

Example 1: Convert $(110.101)_2$ to a decimal number

Solution: **Step 1: Conversion of 110 to decimal**

$$=> 110_2 = (1*2^2) + (1*2^1) + (0*2^0)$$
$$=> 110_2 = 4 + 2 + 0$$
$$=> 110_2 = 6$$

So equivalent decimal of binary integral is 6.

Step 2: Conversion of .101 to decimal

$$=> 0.101_2 = (1*1/2) + (0*1/2^2) + (1*1/2^3)$$
$$=> 0.101_2 = 1*0.5 + 0*0.25 + 1*0.125$$
$$=> 0.101_2 = 0.625$$

So equivalent decimal of binary fractional is 0.625

Step 3: Add result of step 1 and 2.

$$=> 6 + 0.625 = (6.625)_{10}$$

Example 2: Convert $(1010.1101)_2$ to a decimal number

Solution: $(1 \times 2^3) + (0 \times 2^2) + (1 \times 2^1) + (0 \times 2^0) + (1 \times 2^{-1}) + (1 \times 2^{-2}) + (0 \times 2^{-3}) + (1 \times 2^{-4})$
$=> 8 + 0 + 2 + 0 + 0.5 + 0.25 + 0 + 0.0625$

=> $(10.8125)_{10}$

Practice Problems on Binary to Decimal Conversion

Q1: Convert 10110101 in Decimal
Q2: Convert 11011010101 in Decimal
Q3: What is Decimal equivalent of 101.1
Q4: Convert $(1010.001)_2$ to $(….)_{10}$.
Q5: What is the Binary Form of 10101101.
Q6: Convert 110010.1010 in Binary

Octal Number System (Number system with Base 8):

Octal Number System is a number system with base 8 as it uses eight symbols (or digits) namely 0, 1, 2, 3, 4, 5, 6, and 7. For example, 22_8, 13_8, 17_8, etc. are octal numbers. This number system is mainly used in computer programming as it is a compact way of representing binary numbers with each octal number corresponding to three binary digits.

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

Octal to Decimal Conversion:

A decimal number system has a base 10 consisting of digits 0-9. We can easily convert an octal number to a decimal number by following these simple steps:

- Step 1: Write the octal number.
- Step 2: Multiply each digit of the given octal number with an increasing power of 8 starting from the rightmost digit.
- Step 3: Sum all the products obtained in step 2.

Example 1: Represent 123_8 as a Decimal Number.

Solution: $123_8 = 1 \times 8^2 + 2 \times 8^1 + 3 \times 8^0$

$\Rightarrow 123_8 = 1 \times 64 + 2 \times 8 + 3 \times 1$

$\Rightarrow 123_8 = 64 + 16 + 3$

$\Rightarrow 123_8 = 83_{10}$

Hence 83_{10} is decimal representation of 123_8.

Example 2: Represent 31245_8 as a Decimal Number.

$31245 = (3 \times 8^4) + (1 \times 8^3) + (2 \times 8^2) + (4 \times 8^1) + (5 \times 8^0)$

$\Rightarrow (3 \times 4096) + (1 \times 512) + (2 \times 64) + (4 \times 8) + (5 \times 1)$

$\Rightarrow 12{,}288 + 512 + 128 + 32 + 5$

$\Rightarrow 12{,}965$

Example 3: Represent 5230.17_8 as a Decimal Number.

Solution: $(5230.17)_8 = (5 \times 8^3) + (2 \times 8^2) + (3 \times 8^1) + (0 \times 8^0) + (1 \times 8^{-1}) + (7 \times 8^{-2})$

$= (5 \times 512) + (2 \times 64) + (3 \times 8) + (0 \times 1) + (1 \times 18) + (7 \times 164)$

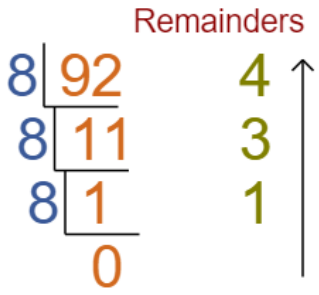
$= 2560 + 128 + 24 + 1/8 + 7/64$

2712 15/64 = $(2712.234375)_{10}$

Decimal to octal conversion:

Example 1: Represent 92_{10} as a Octal Number.

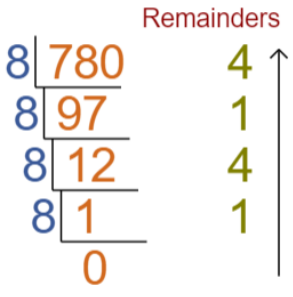
Solution:



$(92)_{10} = (134)_8$

Example 2: Represent 780_{10} as a Octal Number.

Solution:



$(780)_{10} = (1414)_8$

Example 3: Represent 67.5_{10} as a Octal Number.

Solution:

- When **67** is divided by **8**, the quotient is **8** and the remainder is **3**.
- When **8** is divided by **8**, the quotient is **1** and the remainder is **0**.
- When **1** is divided by **8**, the quotient is **0** and the remainder is **1**.
- Write the remainders **from bottom to top**.
- **$(67)_{10} = (103)_8$**

The fractional part of **67.5** is **0.5**. Multiply the fractional part repeatedly by **8** until it becomes **0**.

- **0.5** × **8** = **4.0**

$(0.5)_{10} = (0.4)_8$

$$(67.5)_{10} = (103)_8 + (0.4)_8 = (103.4)_8$$

Octal to Binary:

We can convert a number from octal to binary using two ways:

- Indirect Method: Octal to decimal followed by decimal to binary
- Direct Method: Converting an octal number directly into the binary number system

Direct Method:

Let's convert 54_8 to binary.
The number 54 has two digits: 5 and 4.
From the above chart, we will note down their binary equivalents.
5→101
4→100
Now, combining the two, we get the following binary number: 101100_2.

Indirect Method: Octal to Decimal to Binary

Convert from octal to binary: 54_8.

$5 \times 8^1 + 4 \times 8^0$

$40 + 4$

44_{10}

(44)/2	22	0	0
(22)/2	11	0	1
(11)/2	5	1	2
(5)/2	2	1	3
(2)/2	1	0	4
(1)/2	0	1	5

$= (101100)_2$

Binary to octal:

Let's take example 10100_2 to convert into octal

- $(\mathbf{010})_2 = (\mathbf{2})_8$ $(\mathbf{110})_2 = (\mathbf{6})_8$
- We convert each group of binary numbers to octal and write them in the same order.
- $(\mathbf{10110})_2 = (\mathbf{26})_8$

Hexa decimal number system

Hexadecimal is a number system combining "hexa" for 6 and "deci" for 10. It uses 16 digits: 0 to 9 and A to F, where A stands for 10, B for 11, and so on. Similar to the regular decimal system, it counts up to F instead of stopping at 9. Each digit in hexadecimal has a weight 16 times greater than the previous one, following a positional number system.

When converting to another system, we multiply each digit by the power of 16 based on its position. For example, in the number 7B3, 7 is multiplied by 16 squared, B by 16 to the power of 1, and 3 by 16 to the power of 0.

Facts about Hexadecimal Numbers

- Hexadecimal is a number system with a base value of 16.
- Hexadecimal numbers use 16 symbols or digital values: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F.
- A, B, C, D, E, and F represent 10, 11, 12, 13, 14, and 15 in single-bit form.
- If you see an “0x” as Prefix, it indicates the number is in Hexadecimal. For example, 0x3A
- The position of each digit in a Hexadecimal number has a weight of 16 to the power of its position.

Hexadecimal Number System Table

Following table represents the relationship between hexadecimal number system with decimal as well as binary number system.

Hexadecimal	Decimal	Binary
0	0	0000
1	1	0001
2	2	0010
3	3	0011
4	4	0100
5	5	0101
6	6	0110
7	7	0111
8	8	1000
9	9	1001
A	10	1010
B	11	1011
C	12	1100

Hexadecimal	Decimal	Binary
D	13	1101
E	14	1110
F	15	1111

Hexadecimal Numbers Conversions

The hexadecimal number can be easily converted to various other numbers such as, Binary Numbers, Octal Numbers, Decimal Numbers and vice-versa.

Hexadecimal to Decimal Conversion

Example1: Convert (A7B)16 to decimal.

(A7B)16 = A × 16^2 + 7 × 16^1 + B × 16^0

⇒ (A7B)16 = 10 × 256 + 7 × 16 + 11 × 1 (convert symbols A and B to their decimal equivalents; A = 10, B = 11)

⇒ (A7B)16 = 2560 + 112+ 11

⇒ (A7B)16 = 2683

Therefore, the decimal equivalent of $(A7B)_{16}$ is $(2683)_{10}$.

Example 2: Convert Hexadecimal 1A5 to Decimal

Multiply First Digit (1) by 16 squared (256)

1×16^2 = 256

Multiply Second Digit (A, which is 10 in decimal) by 16 to the power of 1 (16)

10×16^1 = 160

Multiply Third Digit (5) by 16 to the power of 0 (1)

5×16^0 = 5

Adding the results,

1A5 = 1×162 + A×161 + 5×160

⇒ 1A5 = 1×162 + 10×161 + 5×160

⇒ 1A5 = 256 + 160 + 5 = 421

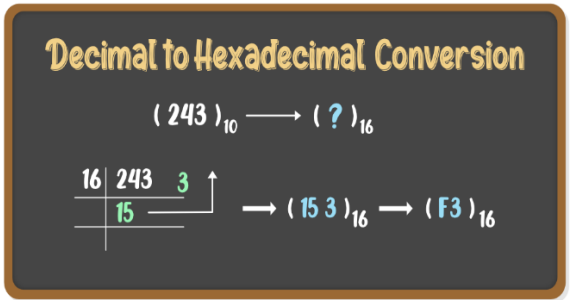
Decimal Equivalent of Hexadecimal number 1A5 is 421

Decimal to Hexadecimal Conversion

Example1: Convert $(92)_{10}$ to hexadecimal.

- When 92 is divided by 16, the quotient is 5 and the remainder is 12 = C.
- When 5 is divided by 16, the quotient is 0 and the remainder is 5.

Example2: Convert $(243)_{10}$ to hexadecimal.



Hexadecimal to decimal Conversion:
Convert $CAFE_{16}$ into decimal

$(CAFE)_{16} = (51966)_{10}$

SOLUTION

CAFÉ
We multiply each digit by its place value and add the products.

$(CAFE)_{16} = (12 \times 16^3) + (10 \times 16^2) + (15 \times 16^1) + (14 \times 16^0)$

$= (12 \times 4096) + (10 \times 256) + (15 \times 16) + (14 \times 1)$

$= 49152 + 2560 + 240 + 14$

$= (51966)_{10}$

Hexadecimal to Binary Conversion
Converting hexadecimal to binary involves two methods: one with a conversion table and the other without a conversion table.
Method 1: Convert Hexadecimal to Binary with Conversion Table
To convert a hexadecimal number to binary using a conversion table, we follow these steps:
Example1: Convert hexadecimal (4D)16 to binary

Look up Decimal Equivalent of each digit in the conversion table.
4 in decimal is $(4)_{10}$, and D in decimal is $(13)_{10}$
Convert each decimal number to binary.
$(4)_{10}$ is $(0100)_2$, and $(13)_{10}$ is $(1101)_2$
Combine the binary numbers
$(4D)_{16}$ is $(01001101)_2$

Method 2: Convert Hexadecimal to Binary without Conversion Table
This method involves multiplying each digit by 16(n-1) to obtain the decimal number, and then dividing by 2 until the quotient is zero.

Binary to hexadecimal:
Convert 11000101_2 into hexa decimal
0101→5
1100→C (C5)16

Octal to Hexadecimal Conversion:
There is a two step process to convert an octal number into hexadecimal:
Step 1: Convert Octal to Binary
For each octal digit, replace it with its three-digit binary equivalent.

Example1: Convert (345)8 to hexadecimal

3 4 5
011 100 101 →011100101
From Lsb:0101→5
1110→E
0000→ 0 $(0E5)_{16}$

Hexadecimal to octal Conversion:
Convert $(AC)_{16}$ into octal
A→1010
C→ 1100 10101100
010 101 100
2 5 4

Binary arithmetic begins with the least significant bit, which is the rightmost bit. The upcoming sections will detail each operation step-by-step.

Binary Addition:
Binary addition involves four straightforward steps, outlined as follows:

- 0 + 0 = 0
- 0 + 1 = 1
- 1 + 0 = 1
- 1 + 1 = 0 (carry 1 to the next significant bit)

An example will help us to understand the addition process. Let us take two binary numbers 10001001 and 10010101

```
    1
 10001001
 10010101
----------
100111110
```

Binary Subtraction:

Here are too four simple steps to keep in memory

- 0 – 0 = 0
- 0 – 1 = 1, borrow 1 from the next more significant bit
- 1 – 0 = 1
- 1 – 1 = 0

A binary arithmetic example is given to understand the operation more clearly

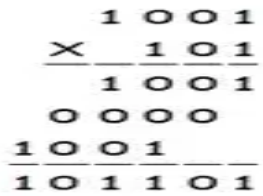
```
 10101010
 10100010
----------
 00001000
```

Binary Multiplication:

- 0×0=0

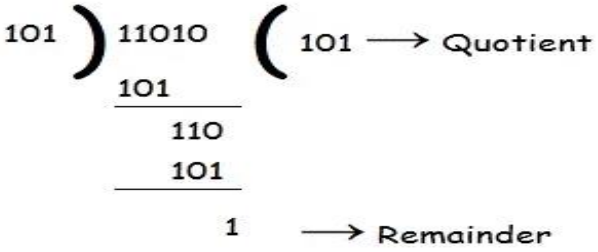
- 1×0=0
- 0×1=0
- 1×1=1 (there is no carry or borrow for this)

The arithmetic of multiplying binary numbers is shown below:



Binary Division

Binary division combines multiplication and subtraction. An illustrative example will help clarify this operation.



Complements: Complement arithmetic is a system of mathematical techniques used in the field of digital electronics to perform various arithmetic operations mainly, subtraction.

Here, we will cover the following most widely used types of complements in digital systems

- 9's Complement
- 10's Complement
- 1's Complement
- 2's Complement
- 7's Complement
- 8's Complement
- 15's Complement
- 16's Complement

9's Complement:

In digital electronics, the 9's complement is a type of complement used to perform subtraction of decimal numbers using a digital system. Thus, 9's complement is related to the decimal number system.

- 9's complement is used to perform subtraction because it simplifies the subtraction operation.
- The 9's complement of a given decimal number is found by subtracting each digit of the number from 9.

Decimal Digit	9's Complement
0	9 – 0 = 9

1	9 – 1 = 8
2	9 – 2 = 7
3	9 – 3 = 6
4	9 – 4 = 5
5	9 – 5 = 4
6	9 – 6 = 3
7	9 – 7 = 2
8	9 – 8 = 1
9	9 – 9 = 0

Let us understand it with the help of examples.

Example 1
Find the 9's complement of the decimal number 7824.450.
Solution
Here is the step-by-step process for finding 9's complement of the given decimal number −

- The 9's complement of 7 = 9 – 7 = 2
- The 9's complement of 8 = 9 – 8 = 1
- The 9's complement of 2 = 9 – 2 = 7
- The 9's complement of 4 = 9 – 4 = 5
- The 9's complement of 4 = 9 – 4 = 5
- The 9's complement of 5 = 9 – 5 = 4
- The 9's complement of 0 = 9 – 0 = 9

Thus, the 9's complement of the decimal number 7824.450 is 2175.549.

10's Complement:

In digital electronics, the 10's complement is another type of complement used to perform subtraction of decimal numbers. Again, the purpose of the 10's complement is to simplify the decimal subtraction operation.

There are two methods for finding the 10's complement of a decimal number −

Method I − To find the 10's complement of a given decimal number, firstly we find the 9's complement by subtracting each digit of the number from 9. Then, we add 1 to the 9's complement to obtain the 10's complement, i.e.,

10’s Complement = 9’s Complement + 1

The 10's complement of each decimal digit using this method is given in the following table –

Decimal Digit	9's Complement
0	9 – 0 = 9 + 1 = 10 = 0 (Ignore the carry)
1	9 – 1 = 8 + 1 = 9

2	9 – 2 = 7 + 1 = 8
3	9 – 3 = 6 + 1 = 7
4	9 – 4 = 5 + 1 = 6
5	9 – 5 = 4 + 1 = 5
6	9 – 6 = 3 + 1 = 4
7	9 – 7 = 2 + 1 = 3
8	9 – 8 = 1 + 1 = 2
9	9 – 9 = 0 + 1 = 1

Method II − In this method, we can use the following formula to find the 10's complement of a given decimal number,

$$10\text{'s Complement} = 10^N - \text{Number}$$

Where, N is the number of digits in the decimal number.
Let us understand the process of finding the 10's complement through examples.

Example 1
Find the 10's complement of the decimal number 4872.

Solution
The 10's complement of 4872 can be determined as follows −
Finding the 9's complement of 4872,

$$9999 - 4872 = 5127$$

Adding 1 to the 9's complement to obtain the 10's complement,

$$5127 + 1 = 5128$$

So, the 10's complement of 4872 is 5128.

Subtraction using 9's Complement:

The 9's complement can be used to perform subtraction of decimal numbers. In this method, the difference of two decimal numbers is obtained by adding the 9's complement of the subtrahend to the minuend.

Let us understand the subtraction using 9's complement through an example.

Example 1

Subtract $(517)_{10}$ from $(729)_{10}$.

Solution

In this example, we have,

Minuend = 729

Subtrahend = 517

Finding the 9's complement of 517, we get

999 – 517 = 482

Now, adding 729 and 482 to obtain the difference of 729 and 517, we get,

729 + 482 = 1211

There is an end around carry, indicating the result is positive and is obtained by adding the end around carry to the LSD of intermediate result to obtain the final result, i.e.,

211 + 1 = 212

So, the difference of 729 and 517 is 212.

Example 2

Subtract $(203)_{10}$ from $(159)_{10}$ using 9's complement method.

Solution

In this example,

Minuend = 159

Subtrahend = 203

Taking 9's complement of 203, we get,

999 – 203 = 796

Adding 159 and 796, we get,

159 + 796 = 955

There is no end around carry. Thus, the final result is negative and obtained by taking the 9's complement of 955, i.e.,

999 – 955 = 44

Thus, the final result of subtraction 159 – 203 = –44.

Subtraction using 10's Complement:

Example 1

Subtract $(599)_{10}$ from $(875)_{10}$ using 10's complement arithmetic.

Solution

In this example, we have,

Minuend = 875

Subtrahend = 599

Finding 10's complement of 599, we get,

10’s complement of 599 = 9’s complement + 1

Therefore,

10’s complement of 599 = (999 – 599) + 1 = 401

Adding 875 and 401, we get,

875 + 401 = 1276

There is an end-around carry, showing that the result is positive and is obtained by discarding the carry.

Thus, the difference of 875 and 599 is 276.

Example 2

Subtract $(307)_{10}$ from $(279)_{10}$ using 10's complement arithmetic.

Solution

We have,

Minuend = 279

Subtrahend = 307

Taking 10's complement of 307, we get,

10’s complement of 307 = (999 – 307) + 1 = 693

Adding 279 and 693, we get,

279 + 693 = 972

There is no end-around carry, indicating that the result is negative. The final result is obtained by taking the 10's complement of 972 i.e.,

10's complement of 972 = (999 - 972) + 1 = 28

Hence, the final result is –28.

1's Complement

In digital electronics, the 1's complement is a type of complement used to simplify the subtraction of binary numbers. Also, the 1's complement is used to represent the negative of a given binary number.

We can find the 1's complement of a binary number by changing all the 0s to 1s and all the 1s to 0s in the number.

Example 1

Find the 1's complement of 101101.

Solution

The 1's complement of 101101 can be obtained as follows −

Method I − By flipping each bit −

- The 1's complement of 1 = 0
- The 1's complement of 0 = 1
- The 1's complement of 1 = 0
- The 1's complement of 1 = 0
- The 1's complement of 0 = 1
- The 1's complement of 1 = 0

Method II − By subtract each bit from 1 −

111111 – 101101 = 010010

Hence, the 1's complement of 101101 is 010010.

2's Complement

In digital electronics, the 2's complement is a concept widely used to perform binary subtraction using a digital system.

Here are the following three methods that can be used to determine the 2's complement of a given binary number −

Method I − By finding the 1's complement and then adding 1 to the 1's complement, i.e.,

2’s Complement = 1’s Complement + 1

Method II − By subtracting the given binary number from 2^N, i.e.,

2’s Complement = 2^N – Number

Where, "N" is the number of bits in the number.

Example 1

Find the 2's complement of 1100111.

Solution

We can find the 2's complement of 1100111 as follows −

Method I − Using 1's complement −

1’s complement of 1100111 = 0011000

Adding 1 to 1's complement to obtain the 2's complement,

0011000 + 1 = 0011001

Method II − Using 2's complement formula −

2's Complement = 2^7 – 1100111 = 128 – 1100111

2's Complement = 10000000 – 1100111 = 0011001

Subtraction using 1's Complement:

Example 1

Subtract $(111)_2$ from $(1011)_2$ using 1's complement.

Solution

In this example, we have,

Minuend = 1011

Subtrahend = 0111

Finding 1's complement of subtrahend,

1’s complement of 0111 = 1000

Adding 1011 and 1000, we get,

1011 + 1000 = 1 0011

There is an end-around carry, indicating that the result is positive. The final result is obtained by adding this end-around carry to LSB of the intermediate result (0011) i.e.,

0011 + 1 = 0100

Hence, the binary difference of 1011 and 111 is 100.

Example 2

Subtract $(1100)_2$ from $(111)_2$ using 1's complement arithmetic.

Solution

We are given,

Minuend = 0111

Subtrahend = 1100

Finding the 1's complement of the subtrahend,

1’s complement of 1100 = 0011

Adding 0111 and 0011, we get,

0111 + 0011 = 1010

There is no end-around carry, indicating that the result is negative and is obtained by taking 1's complement of 1010 i.e.,

1’s complement of 1010 = 0101

Hence, the binary difference of 111 and 1100 is -101.

Subtraction using 2's Complement

Example 1

Subtract $(101)_2$ from $(1100)_2$ using 2's complement arithmetic.

Solution

In this example, we are given,

Minuend = 1100

Subtrahend = 0101

Taking 2's complement of the subtrahend, we get,

2's complement of 0101 = (1111 - 0101) + 1 = 1011

Adding 1100 and 1011, we get,

1100 + 1011 = 1 0111

There is an end-around carry, indicating that the result is positive and the final result is obtained by ignoring this end around carry.

Thus, the binary difference of 1100 and 101 is 111.

Example 2

Subtract $(1010)_2$ from $(0110)_2$ using 2's complement arithmetic.

Solution

The given numbers are,

Minuend = 0110

Subtrahend = 1010

Taking 2's complement of the subtrahend, we get,

2's complement of 1010 = (1111 - 1010) + 1 = 0110

Adding minuend and 2's complement of subtrahend, we get,

0110 + 0110 = 1100

Since, there is no end-around carry, indicating that the result is negative. The final result is obtained by taking 2's complement of intermediate result, i.e.,

2’s complement of 1100 = (1111 - 1100) + 1 = 0100

Thus, the binary difference of 0110 and 1010 is -100

In computers, the data are stored in memory registers with binary bits 1's and 0's as the computers only understand binary language. When we enter data in the computer, it is converted into binary and then processed and used by the CPU in different ways. The memory registers have a specific range and a format to store data. Scientists have developed a real number representation method in the memory registers of 8 bit, 16 bit, 32bit.

We have two major approaches for storing real numbers:

Fixed -Point Representation.

Floating-Point Representation

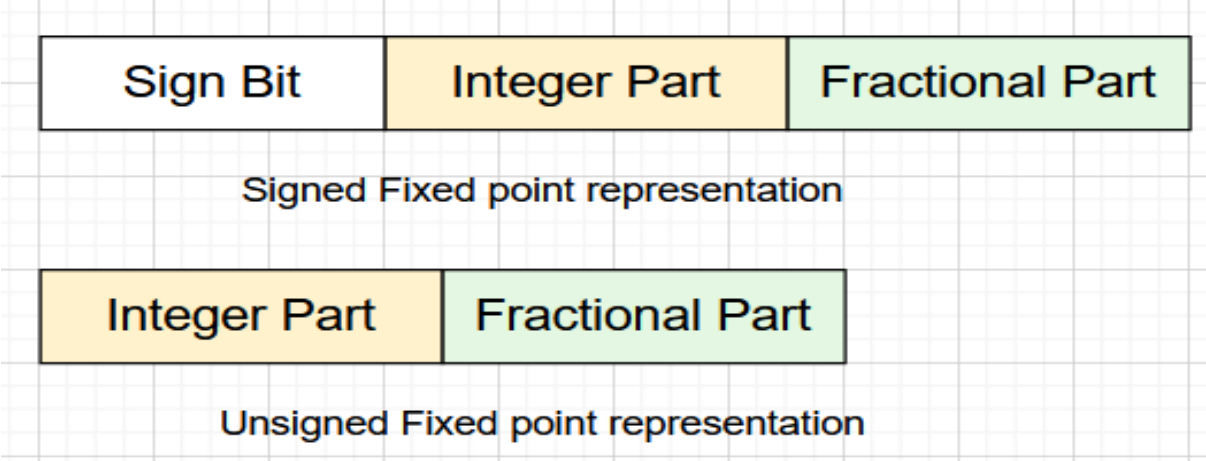
Fixed Point Representation

In computers, fixed-point representation is a real data type for numbers. Fixed point representation can convert data into binary form, and then the data is processed, stored, and used by the computer. It has a fixed number of bits for the integral and fractional parts. For example, if given fixed-point representation is IIIII.FFF, we can store a minimum value of 00000.001 and a maximum value of 99999.999.

There are three parts of the fixed-point number representation:

Sign bit, **Integral part,** and **Fractional part**.

The below figure depicts it.



Sign bit:- The fixed-point number representation in binary uses a sign bit. The negative number has a sign bit 1, while a positive number has a bit 0.

Integral Part:- The integral part in fixed-point numbers is of different lengths at different places. It depends on the register's size; for an 8-bit register, the integral part is 4 bits.

Fractional part:- The Fractional part is of different lengths at different places. It depends on the registers; for an 8-bit register, the fractional part is 3 bits.

Size of Sign Bit, Integer Part, and Fractional Part for different registers are displayed below:

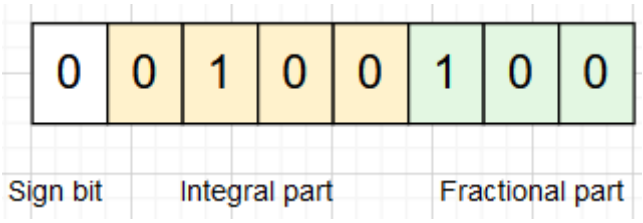
Register	Sign Bit	Integer Part	Fraction Part
8-bit register	1 bit	4 bits	3 bits
16-bit register	1 bit	9 bits	6 bits
32-bit register	1 bit	15 bits	9 bits

How to write numbers in Fixed-point notation?

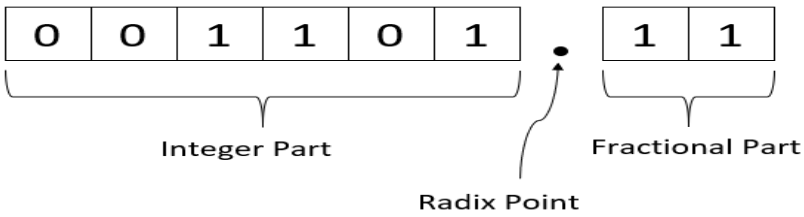
The number considered is 4.5

Step 1: We will convert the number 4.5 to binary form. 4.5 = 100.1

Step 2: Represent the binary number in fixed-point notation with the following format.



Example1:
Given that a fixed-point positive binary number is stored in eight bits word length, where the first six bits are used for the integer part and the next two for the fractional part. How would the $(13.875)_{10}$ be stored? Given that $(13.875)_{10}=(1101.111)_2$

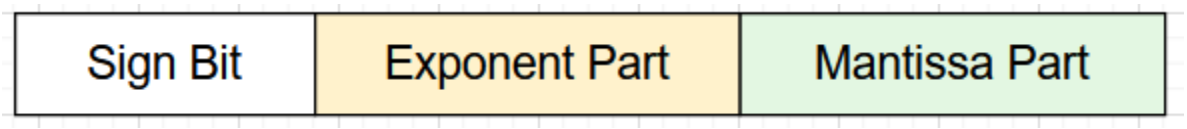


Example 2: Negative Number

- Binary: `1101.0010`
- Sign bit: `1` (Negative)
- Integer part: `101` (which is `5` in decimal)
- Fractional part: `.0010` (which is `0.125` in decimal)
- **Decimal value**: `-5.125`

Floating Point Representation

Floating Point representation doesn't reserve any specific number of bits for the integer or fractional parts. But instead, it reserves certain bits for the number (called the significand or mantissa) and a fixed number of bits to say where the decimal place lies(called the exponent).
The computer uses floating-point number representation to convert the input data into binary form. This binary form number is converted into scientific notation, which is converted into floating-point representation.
The floating-point representation has two types of notation:
1. Scientific notation: Scientific notation is the method of representing binary numbers into $a \ x \ b^e$ form. It is further converted into floating-point representation. For example,
Number = 32625
Number in Scientific Notation = 32.625 x 10^3
Number in binary form = 1101.101*2^{101}
Here, Mantissa is **1101.101** and Base part is $\mathbf{2^{101}}$.
2. Normalization notation: It is a special case of scientific notation. Normalized means that we have at least one non-zero digit after the decimal point.
A floating-point representation has three parts: **Sign bit**, **Exponent Part**, and **Mantissa**. We can see the below diagram to understand these parts.



Parts of floating-point representation

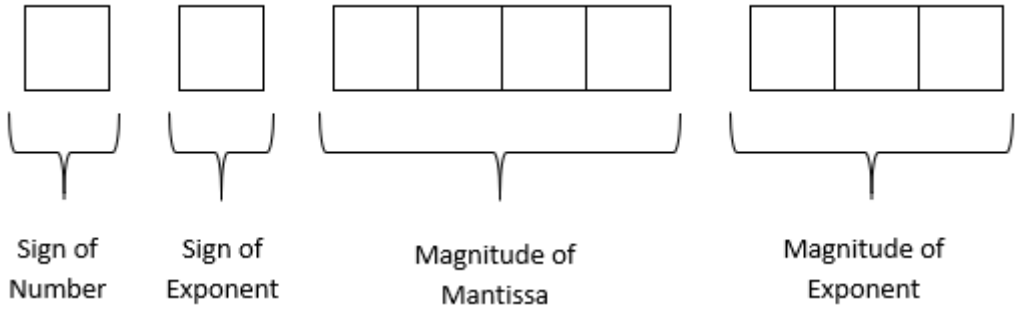
Sign bit:- The floating-point numbers in binary uses a sign bit. A negative number has a sign bit 1, while a positive number has a sign bit 0. The sign of any number depends on mantissa, not on exponent.

Mantissa Part:- The mantissa part is of different lengths at different places. It depends on registers like for a 16-bit register, and mantissa part is of 8 bits.

Exponent Part:- It is the power of the number. It depends on the size of the register. For example, in the 16-bit register, the exponent part is of 7 bits.

How to write numbers in Floating-point notation

The format for the use of the 9-bits is as follows



Following the procedure in Chapter 01.04, the fixed-point binary format of the given base-1010 number is

$$(54.75)10=(110110.11)2$$

Writing the number in floating-point format, we get

$$(54.75)10=(1.1011011)2\times2^{(5)}10$$

The exponent 55 is equivalent in binary format as

$$(5)10=(101)2$$

Hence

$$(54.75)10=(1.1011011)2\times2(101)2$$

The sign of the number is positive, so the bit for the sign of the number will have zero in it.
The sign of the exponent is positive. So, the bit for the sign of the exponent will have zero in it.
The magnitude of mantissa will be

$$m=1011$$

(There are only 44 places for the mantissa, and the leading 11 is not stored as it is always 1 by default).
The magnitude of the exponent is

$$e=101$$

We now have the representation as

0	0	1	0	1	1	1	0	1

Signed binary numbers:

In the decimal number system, a plus (+) sign is used for denoting a positive number and a minus (-) sign for denoting a negative number. The plus sign is normally dropped, and absence of any sign means the number is positive one. This representation of numbers is called the **signed Binary numbers**.

In Signed binary numbers system, negative binary numbers are represented in three ways : Signed magnitude representation. 1's complement representation and 2's complement representation.

Signed Magnitude Representation:

In this representation an additional bit called the sign bit is placed as the MSB to the magnitude of the numbers. The '0' is used for '+' sign and digit (r – 1) is used for '-' sign where r is the base (radix) of the number system and it is (2 – 1) = 1 in binary system. For example in 8 bit binary number having

one sign bit and seven magnitude bits, the largest positive number is 01111111 = +127 and smallest number is 11111111 = -127.

Example 1: Find the decimal equivalent of the following binary numbers assuming signed magnitude representation of the number. (a) 001000, (b) 1111.

(*a*) $[001000]_2 = +[01000]_2 = +8_{10}$ **Ans.**
(MSB is 0 and hence sign is +ve)

(*b*) $[1111]_2 = -[111]_2 = -7_{10}$ **Ans.**
(MSB is 1 and hence sign is –ve)

One's Complement Representation:

In a digital work, two types of complements of binary numbers viz. 1's complement and 2's complement are used for complemental subtraction.

In a binary number, if each 1 is replaced by 0 and each 0 by 1, the resulting number is known as 1's complement of first number. In fact, both numbers are complement of each other. A few examples of 1's complement are given below for illustration

Binary Number	*1's Complement*
100	011
1110	0001
10101	01010
10111	01000
11011011	00100100

2's Complement Representation:

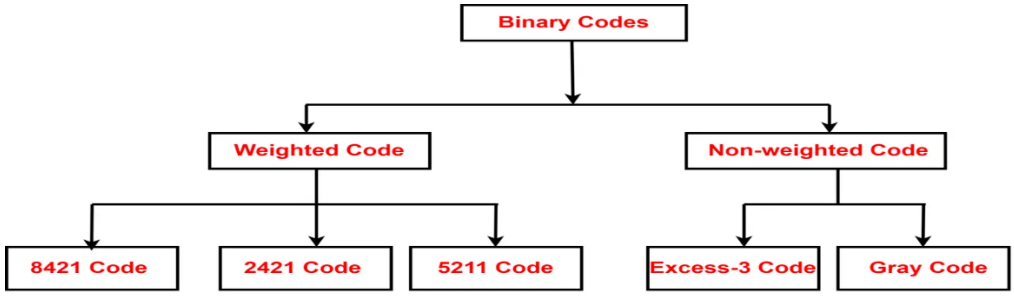
The 2's complement of a binary number is obtained by adding 1 to its 1's complement i.e.,
2's complement = 1's complement + 1.

Binary codes: Binary codes are basically ways of representing information using binary digits, i.e., 0s and 1s. In digital electronics, various binary codes, such as BCD, 2421,5211, Ex-3, Gray, and ASCII, are used to represent text and numeric data.

Types of Binary Codes

The codes can be majorly classified into the following two categories:

- **Weighted Codes**
  - BCD Code or 8421 Code
  - 2421 Code
  - 5211 Code
- **Non-weighted Codes**
  - Excess-3 Code
  - Gray Code



Decimal	8421	2421	5211	Excess-3	Gray code
0	0000	0000	0000	0011	0000
1	0001	0001	0001	0100	0001
2	0010	0010	0011	0101	0011
3	0011	0011	0101	0110	0010
4	0100	0100	0111	0111	0110
5	0101	0101	1000	1000	0111
6	0110	0110	1010	1001	0101
7	0111	0111	1100	1010	0100
8	1000	1110	1110	1011	1100
9	1001	1111	1111	1100	1101

Digital Logic Circuits-I: Basic Logic Functions, Logic gates, universal logic gates, Minimization of Logic expressions. K-Map Simplification, Combinational Circuits, Decoders, Multiplexers.

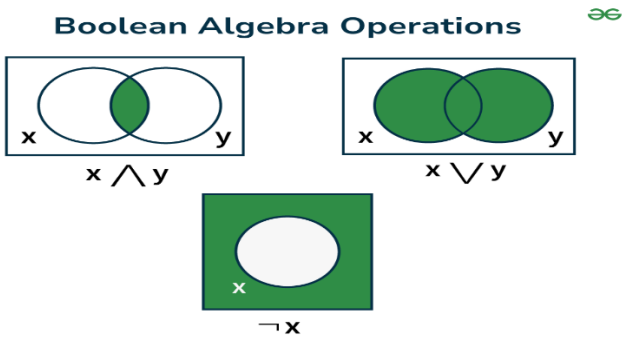
What is Boolean Algebra?

Boolean Algebra is a branch of algebra that deals with boolean values—true and false. It is fundamental to digital logic design and computer science, providing a mathematical framework for describing logical operations and expressions.

Boolean Algebra Operations

Various operations are used in Boolean algebra but the basic operations that form the base of Boolean Algebra are.

- ***Negation or NOT Operation***
- ***Conjunction or AND Operation***
- ***Disjunction or OR Operation***



These operations have their own symbols and precedence and the table added below shows the symbol and the precedence of these operators.

Operator	Symbol	Precedence
NOT	‘ (or) ¬	First
AND	. (or) ∧	Second
OR	+ (or) ∨	Third

Negation or NOT Operation

Using the NOT operation reverse the value of the Boolean variable from 0 to 1 or vice-versa. This can be understood as:

- *If A = 1, then using NOT operation we have (A)’ = 0*
- *If A = 0, then using the NOT operation we have (A)’ = 1*
- *We also represent the negation operation as ~A, i.e if A = 1, ~A = 0*

Conjunction or AND Operation

Using the AND operation satisfies the condition if both the value of the individual variables are true and if any of the value is false then this operation gives the negative result. This can be understood as,

- *If A = True, B = True, then A . B = True*
- *If A = True, B = False, Or A = false, B = True, then A . B = False*
- *If A = False, B = False, then A . B = False*

Disjunction (OR) Operation
Using the OR operation satisfies the condition if any value of the individual variables is true, it only gives a negative result if both the values are false. This can be understood as,

- *If A = True, B = True, then A + B = True*
- *If A = True, B = False, Or A = false, B = True, then A + B = True*
- *If A = False, B = False, then A + B = Falses*

Boolean Algebra Table

Given Below is the Expression for the Boolean Algebra

Operation	Symbol	Definition
AND Operation	· or ∧	Returns true only if both inputs are true.
OR Operation	+ or ∨	Returns true if at least one input is true.
NOT Operation	¬ or ~	Reverses the input.
XOR Operation	⊕	Returns true if exactly one input is true.
NAND Operation	↓	Returns false only if both inputs are true.
NOR Operation	↑	Returns false if at least one input is true.
XNOR Operation	↔	Returns true if both inputs are equal.

Boolean Algebra Rules
In Boolean Algebra there are different fundamental rules for logical expression.

- **Binary Representation:** In Boolean Algebra the variables can have only two values either 0 or 1 where 0 represents Low and 1 represents high. These variables represents logical states of the system.
- **Complement Representation:** The complement of the variables is represented by (¬) or (‘) over the variable. This indicates logical negation or inversion of the variable’s value. So Complement of variable A can be represented by $\bar{A}$,if the value of A=0 then its complement is 1.
- **OR Operation:** The OR operation is represented by (+) between the Variables. OR operation returns true if at least one of the operands is true. For Examples let us take three variables A,B,C the OR operation can be represented as A + B + C.
- **AND Operation:** The AND Operation is denoted by (.) between the Variables. AND operation returns true only if all the operands are true. For Examples let us take three variables A,B,C the AND operation can be represented A.B.C or ABC.

- **Laws for Boolean Algebra**
- The basic laws of the Boolean Algebra are added in the table added below,

Law	OR form	AND form
Identity Law	P + 0 = P	P.1 = P
Idempotent Law	P + P = P	P.P = P
Commutative Law	P + Q = Q + P	P.Q = Q.P
Associative Law	P + (Q + R) = (P + Q) + R	P.(Q.R) = (P.Q).R
Distributive Law	P + QR = (P + Q).(P + R)	P.(Q + R) = P.Q + P.R
Inversion Law	(A')' = A	(A')' = A
De Morgan's Law	(P + Q)' = (P)'.(Q)'	(P.Q)' = (P)' + (Q)'

Logic Gate : A **logic gate** is an electronic circuit designed by using electronic components like diodes, transistors, resistors, and more. As the name implies, a logic gate is designed to perform logical operations in digital systems like computers, communication systems, etc.
Therefore, we can say that the building blocks of a digital circuit are logic gates, which execute numerous logical operations that are required by any digital circuit. A logic gate can take two or more inputs but only produce one output. The output of a logic gate depends on the combination of inputs and the logical operation that the logic gate performs.

Types of Logic Gates:
The logic gates can be classified into the following major types:
1. Basic Logic Gates
There are three basic logic gates:
1. AND Gate
2. OR Gate
3. NOT Gate

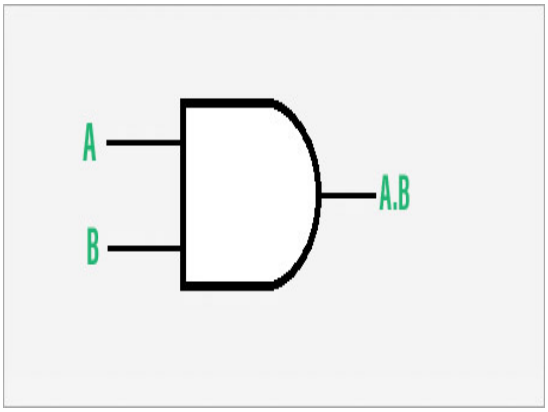
2. Universal Logic Gates
In digital electronics, the following two logic gates are considered as universal logic gates:
1. NOR Gate
2. NAND Gate

3. Derived Logic Gates or special Logic Gates
The following two are the derived logic gates used in digital systems:

1. XOR Gate
2. XNOR Gate

AND Gate

In digital electronics, the AND gate is one of the basic logic gate that performs the logical multiplication of inputs applied to it. It generates a high or logic 1 output, only when all the inputs applied to it are high or logic 1. Otherwise, the output of the AND gate is low or logic 0.



Input		Output
A	**B**	**A AND B**
0	0	0
0	1	0
1	0	0
1	1	1

Properties of AND Gate:

The following are two main properties of the AND gate:

- AND gate can accept two or more than two input values at a time.
- When all of the inputs are logic 1, the output of this gate is logic 1.

The operation of an AND gate is described by a mathematical expression, which is called the Boolean expression of the AND gate.

For two-input AND gate, the Boolean expression is given by,

$Z=A.BZ=A.B$

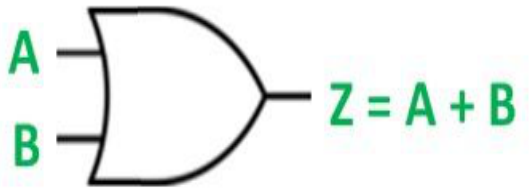
Where, A and B are inputs to the AND gate, while Z denotes the output of the AND gate.

We can extend this expression to any number of input variables, such as,

$Z=A.B.C.D...Z=A.B.C.D...$

OR Gate:

In digital electronics, there is a type of basic logic gate which produces a low or logic 0 output only when its all inputs are low or logic 0. For all other input combinations, the output of the OR gate is high or logic 1. This logic gate is termed as OR gate. An OR gate can be designed to have two or more inputs but only one output. The primary function of the OR gate is to perform the logical sum operation.



Input		Output
A	**B**	**A OR B**
0	0	0
0	1	1
1	0	1
1	1	1

Properties of OR Gate:

An OR gate have the following two properties:

- It can have two or more input lines at a time.
- When all of the inputs to the OR gate are low or logic 0, the output of it is low or logic 0.

The operation of an OR gate can be mathematically described through a mathematical expression called Boolean expression of the OR gate.

The boolean expression for a two input OR gate is given by,

$$Z = A + B$$

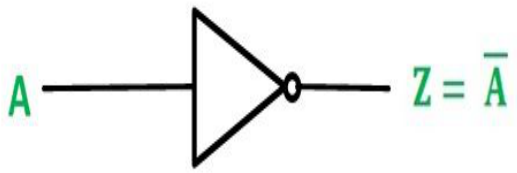
The boolean expression for a three-input OR gate is,

$$Z = A + B + C$$

Here, A, B, and C are inputs and Z is the output variables. We can extend this boolean expression to any number of input variables.

NOT Gate

In digital electronics, the NOT gate is another basic logic gate used to perform **complement of an input signal** applied to it. It takes only one input and one output.



Input	Output
A	**NOT A**
0	1
1	0

Properties of NOT Gate:

- The output of a NOT gate is complement or inverse of the input applied to it.
- NOT gate takes only one output.

The logical operation of the NOT gate is described by its boolean expression, which is given below.

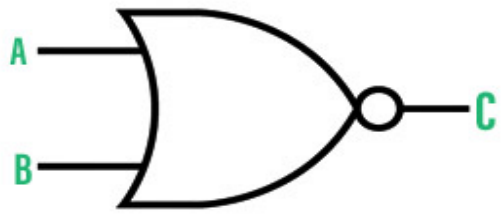
$$Z = \bar{A}$$

NOR Gate

The NOR gate is a type of universal logic gate that can take two or more inputs but one output. It is basically a combination of two basic logic gates i.e., OR gate and NOT gate. Thus, it can be expressed as,

NOR Gate = OR Gate + NOT Gate

In other words, a NOR gate is an OR gate followed by a NOT gate.



Input		Output
A	**B**	**A NOR B**
0	0	1
0	1	0
1	0	0
1	1	0

Properties of NOR Gate:

The following are two important properties of NOR gate:

- A NOR gate can have two or more inputs and gives an output.
- A NOR gate gives a high or logic 1 output only when its all inputs are low or logic 0.

Similar to basic logic gates, we can describe the operation of a NOR gate using a mathematical equation called boolean expression of the NOR gate.

The boolean expression of a two input NOR gate is given below:

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

NAND Gate

In digital electronics, the NAND gate is another type of universal logic gate used to perform logical operations. The NAND gate performs the inverted operation of the AND gate.

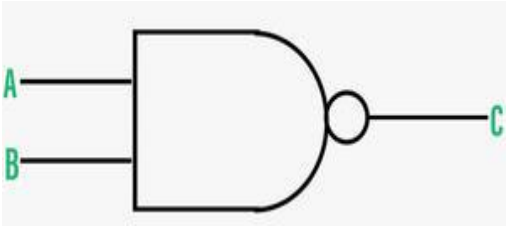
Properties of NAND Gate:

The following are the two key properties of NAND gate:

- NAND gate can take two or more inputs at a time and produces one output based on the combination of inputs applied.
- NAND gate produces a low or logic 0 output only when its all inputs are high or logic 1.

We can describe the expression of NAND gate through a mathematical equation called its boolean expression. Here is the boolean expression of a two input NAND gate.

$C=AB$



Input		Output
A	**B**	**A NAND B**
0	0	1
0	1	1
1	0	1
1	1	0

XOR Gate

In digital electronics, there is a specially designed logic gate named, XOR gate, which is used in digital circuits to perform **modulo sum**. It is also referred to as **Exclusive OR gate or Ex-OR gate**.

The XOR gate can take only two inputs at a time and give an output. The output of the XOR gate is high or logic 1 only when its two inputs are dissimilar.

Input		Output
A	**B**	**A XOR B**
0	0	0
0	1	1
1	0	1
1	1	0

Properties of XOR Gate:
The following two are the main properties of the XOR gate:

- It can accept only two inputs at a time. There is nothing like a three or more input XOR gate.
- The output of the XOR gate is logic 1 or high, when its inputs are dissimilar.

The operation of the XOR gate can be described through a mathematical equation called its boolean expression. The following is the boolean expression for the output of the XOR gate.

$$Z=A \oplus B$$

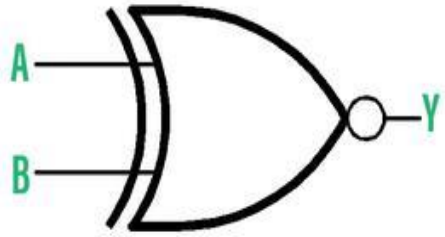
Here, Z is the output variable, and A and B are the input variables.
This expression can also be written as follows:
$Z=AB\bar{}+A\bar{}B$

XNOR Gate

The XNOR gate is another type of special purpose logic gate used to implement **exclusive operation in digital circuits**. It is used to implement the Exclusive NOR operation in digital circuits. It is also called the Ex-NOR or Exclusive NOR gate. It is a combination of two logic gates namely, XOR gate and NOT gate. Thus, it can be expressed as,

XNOR Gate = XOR Gate + NOT Gate

The output of an XNOR gate is high or logic 1 when its both inputs are similar. Otherwise the output is low or logic 0. Hence, the XNOR gate is used as a similarity detector circuit.



Input		Output
A	**B**	**A XNOR B**
0	0	1
0	1	0
1	0	0
1	1	1

Properties of XNOR Gate:

The following are two key properties of XNOR gate:

- XNOR gate takes only two inputs and produces one output.
- The output of the XNOR gate is high logic 1 only when it has similar inputs.

The operation of XNOR gate can be described through a mathematical equation called the boolean expression of XNOR gate. Here is the boolean expression of the XNOR gate.

$Y=A \odot BY=A \odot B$

We can also write this expression as follows:

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

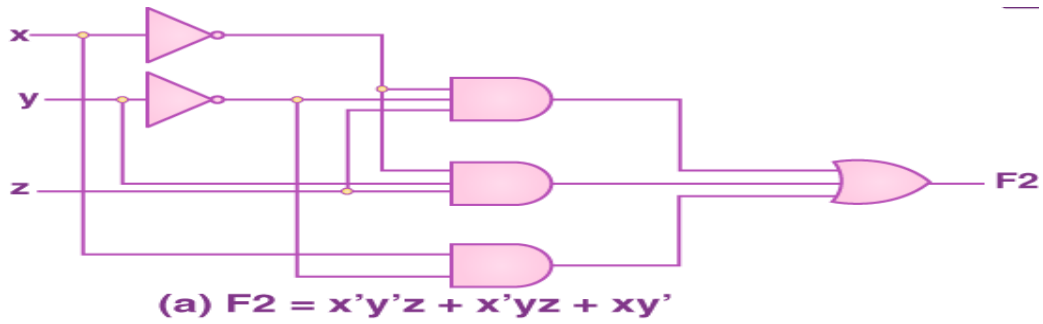
Minimization of Boolean Logic Expressions:

Minimization refers to the process in which we simplify the algebraic expressions of any given boolean function. This process is very important as it helps in the reduction of the overall cost and complexity of an associated circuit.

the total number of literals present in such expressions is usually pretty high (also the complexity of the gates of digital logic implementing a Boolean function is related directly to the algebraic expression's complexity from which the function would be implemented), it is preferable that we simplify the given algebraic expression to its most simplified form.

This process in which we simplify the given algebraic expression present in a boolean expression is known as minimization. This minimization is very crucial since it helps us in the reduction of cost and the overall complexity of an associated circuit.

For instance, the function F = x'y'z +x'yz + xy' can be minimized to F = x'z + xy'. Here are the circuits that are associated with the expressions given above –



(a) F2 = x'y'z + x'yz + xy'

(b) F2 = xy' + x'z

Example 2 – Let us minimize the boolean function given as follows using the method of algebraic manipulation-

F = ABC'D' + ABC'D + AB'C'D + ABCD + AB'CD + ABCD' + AB'CD'

Solution – The properties used here refer to the 3 most common laws mentioned above.

F = ABC' (D' + D) + AB'C'D + ACD (B + B') + ACD' (B + B')

= ABC' + AB'C'D + ACD + ACD' Using Property- 1

= ABC' + AB'C'D + AC (D + D')

= ABC' + AB'C'D + AC Using Property-1

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

= A (B +C) + AB'C'D Using Property-2

= AB + AC + AB'C'D

= AB + AC + AC'D Using Property-2

= AB + AC + AD Using Property-2

Karnaugh Map (K-map):

Definition: Karnaugh Map usually abbreviated as **K-map** is a systematic approach used for simplifying Boolean expressions or logic functions. It is majorly used method for **minimizing the Boolean expressions**. K map is basically known to be a different method for the representation of truth table.

The Karnaugh map (K map) according to the variables involved can be either 1, 2, 3 or 4 variables. The number of cells for K-maps in case of different variables will be identified by:

$$2^n$$

where n is the total number of variables.

So, in case of one variable K-map, n will be equal to 1, thus the number of cells in 1 variable K-map will be 2. Similarly, for two variable K-map, n will be 2, hence a number of cells, in this case, will be 4. Also, for 3 variable K-map, the number of cells will be 8 and for n equals to 4, the number of cells will be 16.

In this way, we can say that the number of variables decides the number of cells in the K-map.

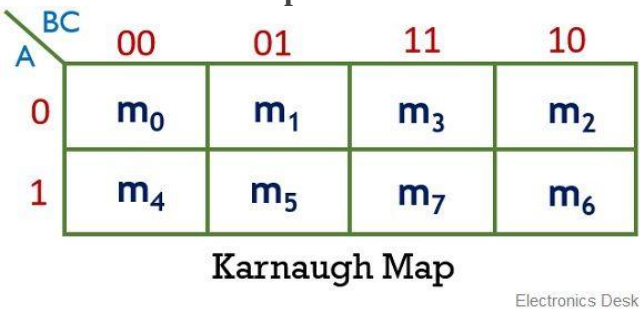
Here in this article, we will have a discussion on the rules to draw a K-map, the assigning of variables to the cells and 3, 4 variables Karnaugh map.

Rules to draw Karnaugh Map

We know that K-map is used for simplification of Boolean expressions. However, some rules are associated whenever a K-map is plotted. The rules are given below:

- In K-map while adding binary terms according to the variables assigned, no 2 variables of adjacent columns can be changed simultaneously.

Have a look at the example of **3 variable K-map** shown below:



Karnaugh Map

Electronics Desk

Here, as we can see that in the third column 11 is represented while in 4th column 10 is shown. This is so because as we have written 01 in the 2nd column. And if we write 10 in the 3rd column then simultaneously 2 variables will get changed. As 0 changes to 1 and 1 changes to 0 simultaneously. This is the reason why m_2 is assigned at the rightmost column of the 1st row and m_3 is placed before m_2. In a similar way, m_6 is present after m_7.

- When a minterm K-map is designed then, in this condition, 1 is assigned at all those cells for which the output is 1, while 0 is provided at those cells where output is 0. But sometimes for simplicity, these 0's is omitted and the cells are kept vacant in the case when 0 is required to be filled.

The condition is reversed at the time of designing a maxterm K-map. Here 1 is assigned to the cell in case of 0 at the output and 0 is assigned in case of 1 at the output.

- While assigning the don't care conditions to the cells of the K-map, 'X' or 'd' is placed at the respective cell.
- At the time of grouping of bits inside the K-map, the highest priority is provided to a group of 16, further the priority decreases with 8, 4 and 2 bits' pair.
- The rule of adjacency is highly followed while pairing the bits inside the Karnaugh map. This is the reason; diagonal pairing is not performed in K-map.

Let us move further and understand the 3 and 4 variables K-map by some examples.

Karnaugh Map for 3 Variables

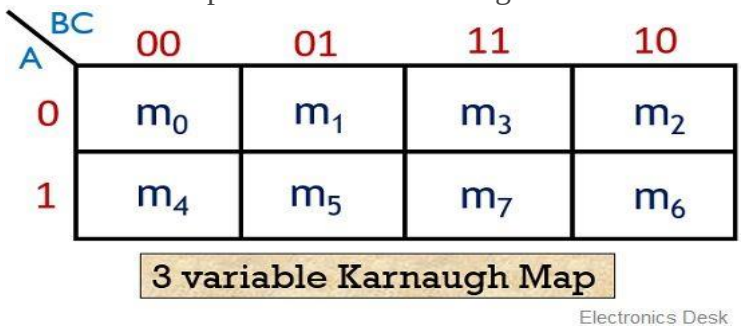
Suppose that we have to simplify a 3 variable Boolean expression using K map.

We know that the number of cells of the K-map is dependent on the number of variables. So, for 3 variable K map, the number of cells will be 2^3 i.e., 8.

Let us reduce the function given below using K-map

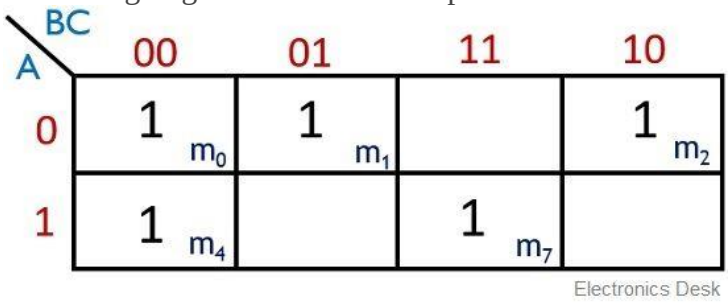
F (A, B, C) = Σm (0, 1, 2, 4, 7)

The figure below represents the K-map for 3 variables having 8 cells.



As we have already discussed earlier the reason behind the assigning of variables to the Karnaugh map. So, let us now proceed towards filling of bits to the minterm K-map.

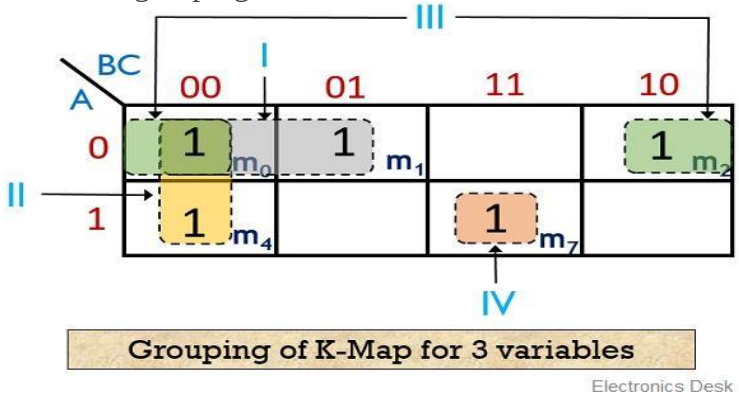
The figure below shows the assigning of bits to the K-map:



Here, we can clearly see that for minterm K-map, 1 is assigned at m_0, m_1, m_2, m_4, m_7, as given in the function.

So, firstly here we will check the priority, as we can see that neither 16, 8 nor 4 1's is present in order to perform the grouping. So, we will check for grouping of 2 bits.

The figure below represents the grouping of bits for the above function:



Here, we can see that two 1, present at the 1st row of column 00 and 01 are forming pair. Similarly, two 1's of the first column and rows 0, 1 are forming a pair. Also, the two corners, 1 at the first row is forming a group of 1.

However, still, a single 1 is left that is unable to participate in a grouping as no other 1 is present at its adjacent position. So, this 1 is considered to be as a group of single 1. As represented in the figure given above.

Let us now see how the above function is realized using K-map.

So, the function for all 4 implicants will be:

$$\text{I implicant} = \bar{A}\bar{B}$$

$$\text{II implicant} = \bar{B}\bar{C}$$

$$\text{III implicant} = \bar{A}\bar{C}$$

$$\text{IV implicant} = \text{ABC}$$

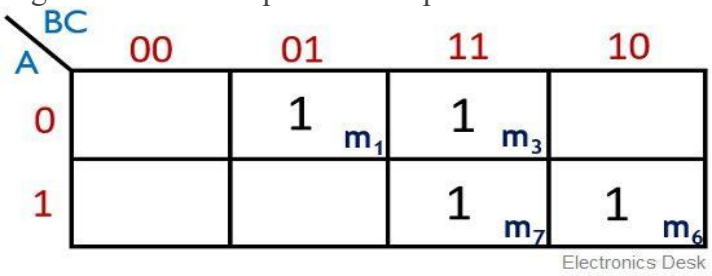
So, combinely the realized Boolean expression will be given as:

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

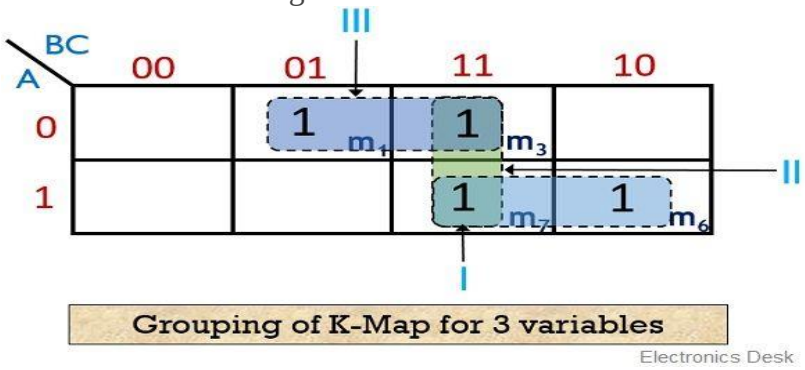
Let us take another example to have a better understanding of the same.

F (A, B, C) = Σm (1, 3, 6, 7)

Let's have a look at the figure below that represents the placement of bits inside the K-map:



The grouping of the bits is shown in the figure below:



Here also, neither a group of 8 nor of 4 is forming. Thus 3 implicants can be formed by grouping as shown in the figure above.

Thus the function will be:

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

But a noteworthy point is that here redundancy of bit is generating. This is so because the two 1's present at BC position is already grouped individually with their adjacent bit. The pairing of two different bits which are separately paired comes under redundancy theorem.

So, in this case, the implicant BC will be ignored. And the realized Boolean expression will be:

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

Let us now proceed to understand how 4 variable K-map gets realized.

Karnaugh map for 4 variables

In case of realizing 4 variable K-map, the number of cells will be 2^4 i.e., 16. At the time of assigning variable to the 4 X 4 K-map, again the thing that is to be kept in mind is no two variables can be changed simultaneously.

As we have done by swapping the variables of 3rd and 4th column in the 3 variable K-map.

The figure below represents a Karnaugh map for 4 variables:

AB \ CD	00	01	11	10
00	m_0	m_1	m_3	m_2
01	m_4	m_5	m_7	m_6
11	m_{12}	m_{13}	m_{15}	m_{14}
10	m_8	m_9	m_{11}	m_{10}

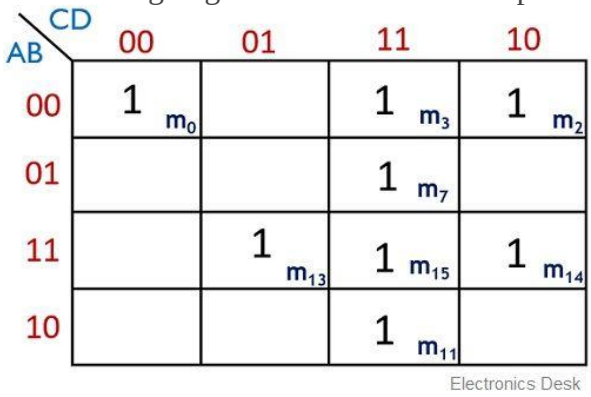
4 variable Karnaugh Map

Electronics Desk

Suppose the function to be realized is given as:

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

The figure below represents the bit assigning to the 4 variable K-map:



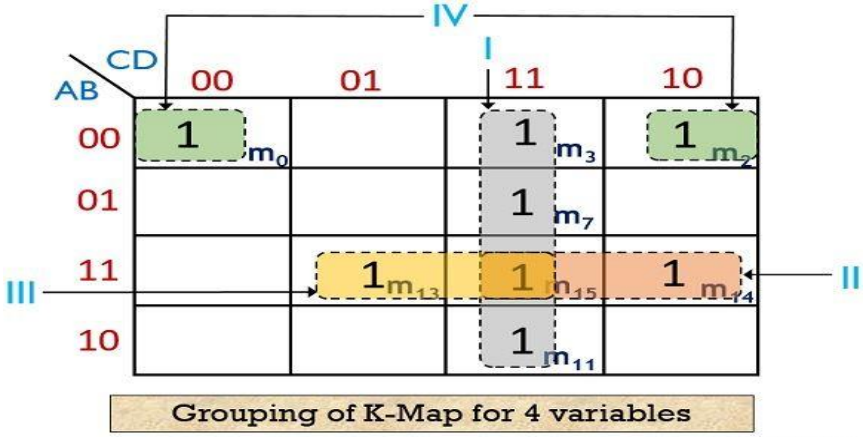
Electronics Desk

Here it is clear from the above figure that 1 is placed at m_0, m_2, m_3, m_7, m_{11}, m_{13}, m_{14}, m_{15}.

Now, after the bits get assigned, the grouping is performed. So, again the grouping is done according to their priority.

As we can see that no combination of 8 bits is present in a way to form a group. However, then a group of 4 1's is formed. Similarly, the two 1's at the corners of the 1st row are grouped together.

The figure here shows the grouping of bits for the 4 variable functions:



Grouping of K-Map for 4 variables

Electronics Desk

So, the above K-map can be realized as:

$$\text{I implicant} = CD$$

$$\text{II implicant} = ABC$$

$$\text{III implicant} = ABD$$

$$\text{IV implicant} = \bar{A}\bar{B}\bar{D}$$

Hence the combined realized Boolean expression for the above K-map will be

$$F = CD + ABC + ABD + \bar{A}\bar{B}\bar{D}$$

Don't Care Condition in Karnaugh Map

Till now we have discussed the conditions where the desired output is generated according to some input conditions. But there exist some cases in which the output remains unspecified because of invalid input conditions.

These outputs are denoted as 'd' or X in the K-map and are known as **don't care condition**.

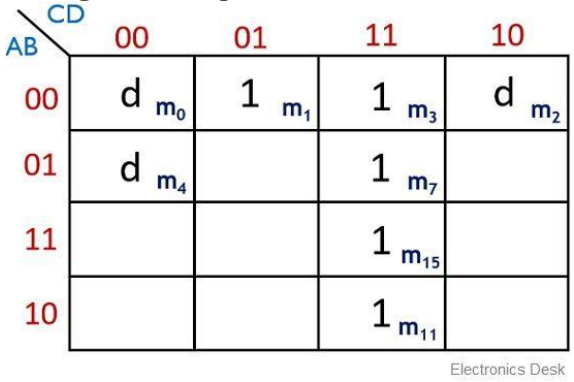
Basically whenever these don't care terms are represented in the K-map then these are utilized in the realization of Boolean expression if required. Otherwise, these are ignored.

Let us now take an example of a function with 4 variables with don't care condition.

Suppose the function be:

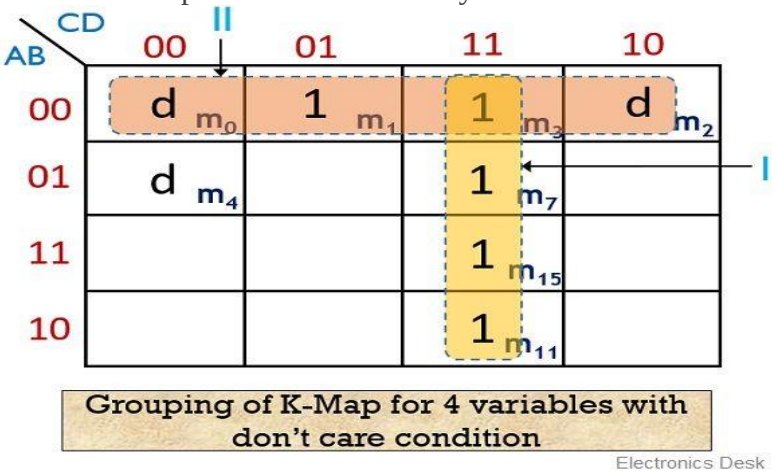
F (A, B, C, D) = Σm (1, 3, 7, 11, 15) + Σd (0, 2, 4)

So, for the above function the designed K-map is shown below:



Here as we can see clearly that at cell position, m_0, m_2, m_4 'd' are placed representing the don't care condition. However, we have already discussed that in the case of SOP realization these 'd' can be considered at the time of grouping.

Thus the grouping inside the K-map can be done in a way shown below:



Here, by making use of don't care terms, 2 groups of 4 can be formed. Or we can say 2 quads are formed.

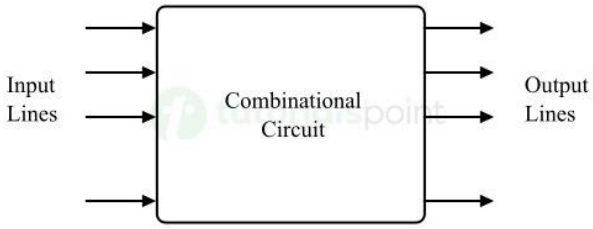
So, the realized function is given as

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

In this way, various functions are realized using a Karnaugh map technique.

Combinational circuit: A combinational circuit, also called a combinational logic circuit, is a digital electronic circuit whose output is determined by present inputs only.

The output of a combinational logic circuit does not depend on the history of the circuit operation. In other words, a combinational circuit is a digital logic circuit whose output depends only on the present input values and does not depend on any feedback or previous input or output values.



- A combinational circuit is a type of digital logic circuit whose output depends on the present input values only and does not depend on past input and output values.
- A combinational circuit is considered to not have a memory element in its circuit that stores previous inputs and outputs.
- The most important characteristic of a combinational circuit is that it does not have any feedback path between input and output. Therefore, the combinational circuits can be categorized as open-loop systems.

Types of Combinational Circuits: Some common types of combinational circuits and their functions are explained below −

- Binary Adders
- Binary Subtractors
- Multiplexers (MUX)
- Demultiplexers (DEMUX)
- Encoders
- Decoders
- Comparators

Decoder: In digital electronics, a combinational logic circuit that converts an N-bit binary input code into M output channels in such a way that only one output channel is activated for each one of the possible combinations of inputs is known as a decoder.

In other words, a combinational logic circuit which converts N input lines into a maximum of 2N output lines is called a decoder.

Therefore, a decoder is a combination logic circuit that is capable of identifying or detecting a particular code. The operation that a decoder performs is referred to as decoding. A general block diagram of a decoder is shown in Figure-1.

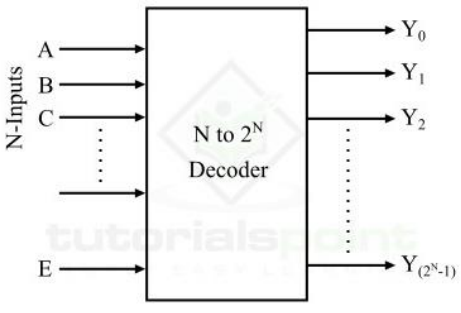


Figure 1 - Decoder

Here, the decoder has N input lines and M (2^N) output lines. In a decoder, each of the N input lines can be a 0 or a 1, hence the number of possible input combinations or codes be equal to 2^N.

Types of Decoders:

There are several types of decoder present. But, based on the input and output lines present, decoders may classified into the following three types −

- 2 to 4 Decoder
- 3 to 8 Decoder
- 4 to 16 Decoder

Now, let us discuss each type of decoder in detail one by one.

2 to 4 Decoder:

The 2 to 4 decoder is one that has 2 input lines and 4 (2^2) output lines. The functional block diagram of the 2 to 4 decoder is shown in Figure-2.

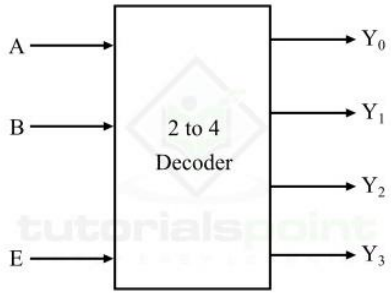


Figure 2 - 2 to 4 Decoder

Inputs			**Outputs**			
E	**A**	**B**	**Y_3**	**Y_2**	**Y_1**	**Y_0**
0	X	X	0	0	0	0
1	0	0	0	0	0	1
1	0	1	0	0	1	0
1	1	0	0	1	0	0
1	1	1	1	0	0	0

Using this truth table, we can derive the Boolean expression for each output as follows −

$$Y0=E\cdot\overline{A}\cdot\overline{B}$$

$$Y1=E\cdot\overline{A}\cdot B$$

$$Y2=E\cdot A\cdot\overline{B}$$

$$Y3=E\cdot A\cdot B$$

As each output term contains products of input variables that can be implemented with the help of AND gates. Therefore, the logic circuit diagram of the 2 to 4 decoder is shown in Figure-3.

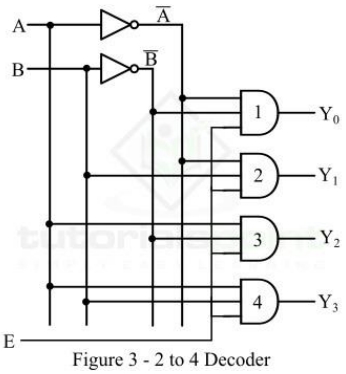


Figure 3 - 2 to 4 Decoder

Operation

The operation of logic circuit of the 2 to 4 decoder is described as follows −

- When enable input (E) is inactive, i.e. set to 0, none of the AND gates will function.
- When enable input (E) is made active by setting it to 1, then the circuit works as explained below.
- When A = 0 and B = 0, the AND gate 1 becomes active and produces output Y0.
- When A = 0 and B = 1, the AND gate 2 becomes active and produces output Y1.
- When A = 1 and B = 0, the AND gate 3 becomes active and produces output Y2.
- When A = 1 and B = 1, the AND gate 4 becomes active and produces output Y3.

3 to 8 Decoder

The 3 to 8 decoder is one that has 3 input lines and 8 (2^3) output lines. The functional block diagram of the 3 to 8 decoder is shown in Figure-4.

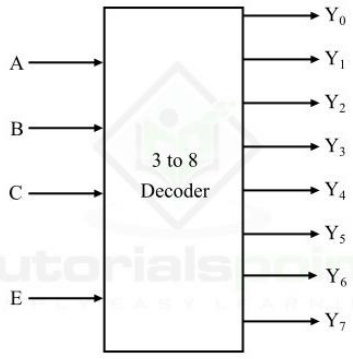


Figure 4 - 3 to 8 Decoder

When this decoder is enabled with the help of enable input E, then it's one of the eight outputs will be active for each combination of inputs. The operation of this 3-line to 8-line decoder can be analyzed with the help of its function table which is given below.

	Inputs			**Outputs**							
E	**A**	**B**	**C**	**Y_7**	**Y_6**	**Y_5**	**Y_4**	**Y_3**	**Y_2**	**Y_1**	**Y_0**
0	X	X	X	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0	0	1
1	0	0	1	0	0	0	0	0	0	1	0

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

Using this function table, we can derive the Boolean expression for each output as follows −

$Y0=EA\bar{B}\bar{C}$
$Y1=EA\bar{B}C$
$Y2=EB\bar{C}\bar{C}$
$Y3=EA\bar{B}C$
$Y4=EAB\bar{C}$
$Y5=EAB\bar{C}$
$Y6=EABC$
$Y7=EABC$

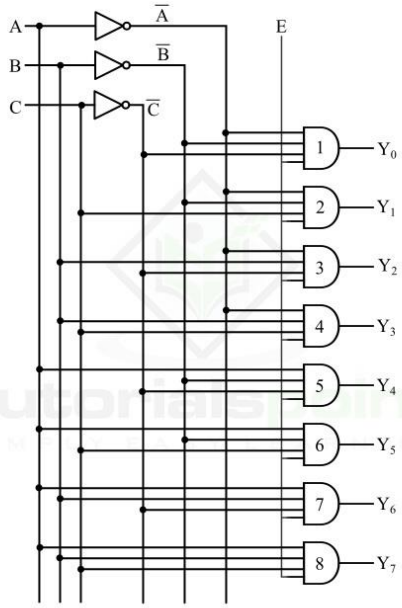


Figure 5 - 3 to 8 Decoder

Operation

The operation of logic circuit of the 3 to 8 decoder is described as follows −

- When enable input (E) is inactive, i.e. set to 0, none of the AND gates will function.
- When enable input (E) is made active by setting it to 1, then the circuit works as described below.
- When A = 0, B = 0, and C = 0, the AND gate 1 becomes active and produces output Y_0.
- When A = 0, B = 0, and C = 1, the AND gate 2 becomes active and produces output Y_1.
- When A = 0, B = 1, and C = 0, the AND gate 3 becomes active and produces output Y_2.
- When A = 0, B = 1, and C = 1, the AND gate 4 becomes active and produces output Y_3.
- When A = 1, B = 0, and C = 0, the AND gate 5 becomes active and produces output Y_4.
- When A = 1, B = 0, and C = 1, the AND gate 6 becomes active and produces output Y_5.
- When A = 1, B = 1, and C = 0, the AND gate 7 becomes active and produces output Y_6.
- When A = 1, B = 1, and C = 1, the AND gate 8 becomes active and produces output Y_7.

Multiplexer: A digital logic circuit that accepts several data inputs and allows only one of them at a time to flow through the output is called a multiplexer or MUX.

It is a combination logic circuit that is designed to accept multiple input signals and transfer only one of them through the output line. In simple words, a multiplexer is a digital logic device that selects one-out-of-N ($N = 2^n$) input data sources and transmits the selected data to a single output line.

The multiplexer is also called **data selector** as it selects one from several. The block diagram of a typical **2^n:1** multiplexer is shown in Figure 1.

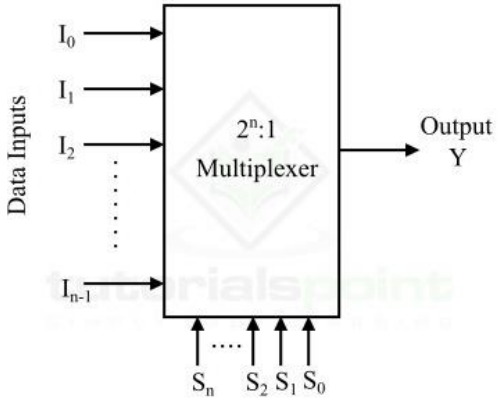


Figure 1 - Digital Multiplexer

Types of Multiplexers: Based on input data lines and select lines, the multiplexer can be of several types.

- 2×1 Multiplexer
- 4×1 Multiplexer

2×1 Multiplexer

The block diagram of a 2×1 multiplexer is shown in Figure 2. The 2×1 multiplexer is basic two input multiplexer which has two data input lines designated as I_0 and I_1, one data select line denoted by S and one output line denoted by Y. The 2×1 mux is used to connect two 1-bit data sources to a common designation.

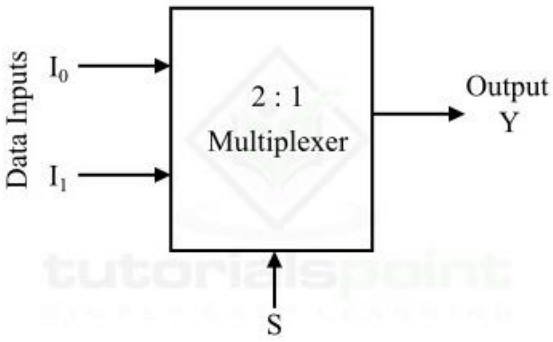


Figure 2 - 2:1 Multiplexer

In the 2×1 multiplexer, the logic level of the digital signal applied to the select line S determines which data input will pass through the output line. The operation of the 2×1 multiplexer can be understood from the following truth table.

Select Line (S)	Output (Y)
0	I_0
1	I_1

4×1 Multiplexer

4×1 Multiplexer has four data inputs I_3, I_2, I_1 & I_0, two selection lines s_1 & s_0 and one output Y. The **block diagram** of 4×1 Multiplexer is shown in the following figure.

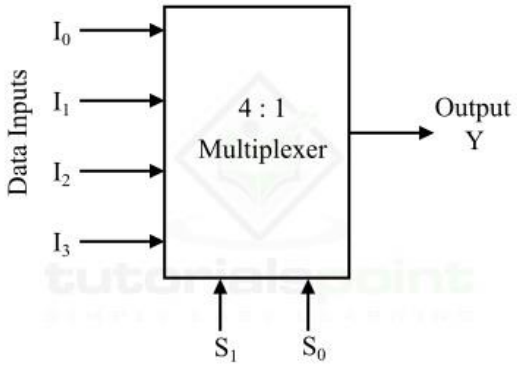


Figure 3 - 4:1 Multiplexer

One of these 4 inputs will be connected to the output based on the combination of inputs present at these two selection lines. **Truth table** of 4×1 Multiplexer is shown below.

Selection Lines		Output
S_1	**S_0**	**Y**
0	0	I_0
0	1	I_1
1	0	I_2
1	1	I_3

From Truth table, we can directly write the **Boolean function** for output, Y as

$$Y=S1'S0'I0+S1'S0I1+S1S0'I2+S1S0I3Y$$

We can implement this Boolean function using Inverters, AND gates & OR gate. The **circuit diagram** of 4×1 multiplexer is shown in the following figure.