Lecture Notes For DIGITAL ELECTRONICS & COMPUTER ORGANIZATION
Prepared By SHIBANI MADHUSMITA SAHOO

Unit-1

What is a Signal?

A physical quantity that has capability to transmit information from one point to another is called a signal. Some common examples of signals include voltage, current, electromagnetic wave, optical signals, etc.

Signals are the backbone of any electronic processing or communication system. These can be transmitted through various types of communication channels like wires, space (electromagnetic waves), optical fibers, etc.

Two basic types of signals are used for carrying data, viz. **analog signal** and **digital signal**.
What is an Analog Signal?
A type of electronic signal that has continuous values within a given range is called an analog signal. Analog signals are expressed as the continuous functions of time. They are represented as the waveforms of continuously varying current or voltage.

Example of analog signals are voice, speed, pressure, temperature, etc.

What is a Digital Signal?

A digital signal is a type of electronic signal that has a finite set of discrete values representing information.

Key	Analog Signals	Digital Signals
Representation	Analog signals are represented as continuous functions or waveforms of time.	Digital signals are represented as discrete functions of time.
Nature	Analog signals are continuous as they have infinite values within a specified range.	Digital signals are discontinuous as they have distinct values sampled at specific time instants.
Resolution	Analog signals have infinite resolution.	Digital signals have a finite resolution.
Accuracy	Analog signals are more accurate.	Digital signals are relatively less accurate.

Storage	Analog signals are difficult to store.	Digital signals are efficient to store.
Noise immunity	Analog signals are less immune to noise.	Digital signals have high immunity against noise.
Examples	Voice signals, temperature, speed, etc.	Data transmitted over internet, computer generated signals, etc.

In digital electronics, the following four types of digital number systems are mainly used −

Binary Number System
Decimal Number System
Octal Number System
Hexadecimal Number System

Binary Number System
Binary number system has two symbols or digits, i.e., 0 and 1 these two digits. are used to represent information and perform all the digital operations. Each binary digit is called a bit..

Binary Number System

Since there are two digits are used in the binary number system, hence its base is 2. Therefore, the value of a binary number is calculated as the sum of powers of 0 is used to represent the OFF state of the digital system and 1 is used to represent the ON state of the system.

Example

Consider the binary number 1101.011. The integer part of this number is 1101 and the fractional part of this number is 0.011. The digits 1, 0, 1 and 1 of the integer part have weights of 2^0, 2^1, 2^2,2^3respectively. Similarly, the digits 0, 1 and 1 of fractional part have weights of 2^{-1}, 2^{-2}, 2^{-3} respectively.

Mathematically, we can write it as,

$1101.011=(1\times2^3)+(1\times2^2)+(0\times2^1)+(1\times2^0)+(0\times2^{-1})+(1\times2^{-2})+(1\times2^{-3})$
Example

Consider the **decimal number** 1358.246. The integer part of this number is 1358 and the fractional part of this number is 0.246. The digits 8, 5, 3 and 1 have weights of $(10)^0$, $(10)^1$, $(10)^2$ and $(10)^3$ respectively. Similarly, the digits 2, 4 and 6 have weights of $(10)^{-1}$, $(10)^{-2}$ and $(10)^{-3}$ respectively.

Mathematically, we can write it as,

$1358.246=(1\times10^{3})+(3\times10^{2})+(5\times10^{1})+(8\times10^{0})+(2\times10^{-1})+(4\times10^{-2})+(6\times10^{-3})$

Octal Number System

The **octal** number system is another type of digital number system used in the field of digital electronics to represent information. It is a base 8 number system having eight unique digits i.e., 0, 1, 2, 3, 4, 5, 6, and 7.

It is important note that the octal number system is equivalent to 3-bit binary number system as 23 = 8. Hence, this number system can be used in computing and digital electronic applications.

The value of an octal number is obtained by the sum of powers of 8, as 8 is the base of the octal number system.

Example

Consider the **octal** number 1457.236. Integer part of this number is 1457 and fractional part of this number is 0.236. The digits 7, 5, 4 and 1 have weights of $(8)^{0}$, $(8)^{1}$, $(8)^{2}$ and $(8)^{3}$ respectively. Similarly, the digits 2, 3 and 6 have weights of $(8)^{-1}$, $(8)^{-2}$, $(8)^{-3}$ respectively.

Mathematically, we can write it as,

$1457.236=(1\times8^{3})+(4\times8^{2})+(5\times8^{1})+(7\times8^{0})+(2\times8^{-1})+(3\times8^{-2})+(6\times8)^{-3}$

Hexadecimal Number System

The hexadecimal number system is a base 16 number system. It has 16 digits, 0 to 9 and A to F. Where, A represents 10, B represents 11, C represents 12, D represents 13, E represents 14, and F represents 15. The hexadecimal number system is equivalent to a 4-bit binary number system as 24 = 16. Thus,

Example

Consider the hexadecimal number 1A05.2C4. The integer part of this number is 1A05 and the fractional part of this number is 0.2C4. The digits 5, 0, A and 1 have weights of $(16)^{0}$, $(16)^{1}$, $(16)^{2}$ and $(16)^{3}$ respectively $1A05.2C4=(1\times16^{3})+(10\times16^{2})+(0\times16^{1})+(5\times16^{0})+(2\times16^{-1})+(12\times16^{-2})+(4\times16^{-3})$. Similarly, the digits 2, C and 4 have weights of $(16)^{-1}$, $(16)^{-2}$ and $(16)^{-3}$ respectively.

Mathematically, we can write it as,

$1A05.2C4=(1\times16^{3})+(10\times16^{2})+(0\times16^{1})+(5\times16^{0})+(2\times16^{-1})+(12\times16^{-2})+(4\times16^{-3})$

After simplifying the right-hand side terms, we will get a decimal number, which is an equivalent of the hexadecimal number on the left-hand side.

Binary to Decimal Conversion

We can convert a binary number into its equivalent decimal number by using the positional weights method.

In this method of binary to decimal conversion, each digit of the given binary number is multiplied by its positional weight. Then, all the products are added to obtain the equivalent decimal number.

The step-by-step process of converting a binary number to its equivalent decimal number by using positional weights method is explained below −

Step 1 − Write the positional weights for each binary digit.

Step 2 − Multiply each binary digit with its positional weight.

Step 3 − Add the product terms to obtain the equivalent decimal number.

Let us consider some examples to understand the binary to decimal conversion.

Example 1

Convert $(101101)_2$ into decimal equivalent.

The given binary number is $(101101)_2$

Step 1 − Defining positional weights for the given binary number –

Bits	1	0	1	1	0	1
Weights	2^5	2^4	2^3	2^2	2^1	2^0

Step 2 – Calculating product of bits and positional weights –

Bits	Weights	Multiply	Product
1	2^5	1×32	32
0	2^4	0×16	0
1	2^3	1×8	8
1	2^2	1×4	4
0	2^1	0×2	0

1	2^0	1×1	1

Example 3

Convert $(1001.11)_2$ into decimal.

Solution

The given binary number has integer and fractional parts. The integer part is multiplied with positive weights, while the fractional part is multiplied with negative weights as follows −

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

Decimal Number = $8 + 0 + 0 + 1 + 0.5 + 0.25 = (9.75)_{10}$

Thus, the decimal equivalent of (1001.11)2 is $(9.75)_{10}$.

Decimal to Binary Conversion

A decimal number can be converted to their equivalent binary number by using the double-dabble method. In this method, the integer part of the given decimal number is successively divided by 2 and the fractional part is successively multiplied by 2.

In the integer part, the remainders read from bottom to top give the integer part of the binary equivalent. In the fractional part, the carries read from top to bottom give the fractional part of the binary equivalent.

The following steps are followed to convert a decimal number to the binary equivalent –

Step 1 – Divide the integer part of the given decimal number successively by 2 and read the remainders from bottom to top.

Step 2 – Multiply the fractional part of the given decimal number successively by 2 and read the carries from top to bottom.

Let us see some examples to understand the conversion of a decimal number into its equivalent binary number.

Example 1

Convert $(28)_{10}$ to binary equivalent.

The given decimal number is an integer. Thus, we divide the decimal number successively by 2 and read the remainders upwards to obtain the equivalent binary number.

	Decimal	
2	28	
2	14	0
2	7	0
2	3	1
2	1	1
	0	1

Reading the remainders from bottom to top, the result will be $(11100)_2$. It is the binary equivalent of $(28)_{10}$.

Example 2

Convert $(165.75)_{10}$ to its equivalent binary.

Solution

The given decimal number is a mixed number having both integer and fractional parts. Thus, to obtain its equivalent binary number, we convert the integer and fractional parts separately.

The binary equivalent of 165_{10} is obtained as follows,

	Decimal	
2	165	
2	82	1
2	41	0
2	20	1
2	10	0
2	5	0
2	2	1
2	1	0

	0	1

Reading the remainders from bottom to top, the binary equivalent of 16510 is (10100101)2.
Now, let's convert the Fractional Part (0.75) of the given number.
To convert the given decimal fraction into binary, we multiply it by 2, as follows,

Decimal	Product	Carry
0.75 × 2	1.5	1
0.5 × 2	1.0	1
0 × 2	0	

Reading the carries from top to bottom, the result is 0.11. Thus, the binary equivalent of $(0.75)_{10}$ is $(0.11)_2$.

Therefore, $(165.75)_{10} = (10100101.11)_2$

Binary to Octal Conversion

A binary number can be converted into its equivalent octal number by mapping method.
The conversion of a binary number to the octal equivalent is done as per the following steps

Step 1 − Starting from the binary point, make groups of 3-bits on both sides of the binary point.
Step 2 − Replace each group of 3-bit binary by the equivalent octal digit.
The following table shows the equivalent octal digital for each 3-bit binary group –

Octal	Binary		
	$(2)^2 = 4$	$(2)^1 = 2$	$(2)^0 = 1$
0	0	0	0
1	0	0	1
2	0	1	0
3	0	1	1
4	1	0	0
5	1	0	1
6	1	1	0
7	1	1	1

Let us understand the binary to octal conversion with the help of examples.

Example 1

Convert $(110011101.110101)_2$ to its octal equivalent.

Solution

The binary to octal conversion will be performed as follows –

3-bit Group	110	011	101	.	110	101
Octal Equivalent	6	3	5	.	6	5

Hence, the octal equivalent of given binary number is $(635.65)_8$.

Example 2
Convert $(1110111011.11101)_2$ to octal equivalent.

Solution

The conversion of given binary number to octal number is given below –

3-bit Group	1	110	111	011	.	111	01
	001	110	111	011	.	111	010
Octal Equivalent	1	6	7	3	.	7	2

Hence, the octal equivalent of $(1110111011.11101)_2$ is $(1673.72)_8$.

Octal to Binary Conversion

We can also use the mapping method to convert an octal number into its equivalent binary number. In this method, we just replace each digital of the given octal number by its 3-bit binary equivalent.

Lets understand the conversion of octal number to binary equivalent with the help of examples.

Example 1
Convert $(3572.126)_8$ to binary equivalent.

Solution
The given octal number is converted into binary equivalent as follows –

Octal Number	3	5	7	2	.	1	2	6

3-bit Binary Equivalent	011	101	111	010	.	001	010	110

Hence, the binary equivalent of $(3572.126)_8$ is $(011101111010.001010110)_2$.

Example 2

Convert (364.52)8 to its binary equivalent.

Octal Number	3	6	4	.	5	2
3-bit Binary Equivalent	011	110	100	.	101	010

Thus, the binary equivalent of the octal number $(364.52)_8$ is (011110100.101010)2.

Octal to Decimal Conversion

The conversion of an octal number to its equivalent decimal number is same as the binary to decimal conversion. To convert an octal number to its decimal equivalent, we multiply each digit of the octal number by its positional weight and then add all the product terms to obtain the equivalent decimal number. The step-by-step procedure to convert an octal number to its equivalent decimal number is given below −

Step 1 − Write the positional weights for each octal digit.

Step 2 − Multiply each octal digit with its positional weight.

Step 3 − Add the product terms to obtain the equivalent decimal number.

Example 1

Convert $(356.25)_8$ to its decimal equivalent.

Solution

The given octal number can be converted to equivalent decimal number as follows

Octal Digits	Positional Weights	Multiply	Product
3	$(8)^2$	$3 \times (8)^2$	192
5	$(8)^1$	$5 \times (8)^1$	40
6	$(8)^0$	$6 \times (8)^0$	6
.	.	.	.
2	$(8)^{-1}$	$2 \times (8)^{-1}$	0.25
5	$(8)^{-2}$	$5 \times (8)^{-2}$	0.078

Adding all the product terms to obtain the equivalent decimal number,

$(356.25)_8 = 192 + 40 + 6 + 0.25 + 0.078 = (238.328)_{10}$

Decimal to Octal Conversion

We can convert a mixed decimal number (having integer and fractional parts) to its equivalent octal number. For this, we convert the integer and fractional parts separately.

To convert the integer part of the given decimal number to octal, we divide the given decimal number successively by 8 till the quotient is 0. The octal equivalent is obtained by reading the remainders from bottom to top, where the last remainder will be the most significant digit.

To convert the fractional part of the given decimal number to octal, we multiply the given decimal fraction successively by 8 till the product is 0 or the desired accuracy is obtained. The fractional part of the equivalent octal number is obtained by reading the carries from top to bottom.

Lets understand the decimal to octal conversion with the help of examples.

Examples.

Convert $(589.278)_{10}$ to octal.

Solution

st convert the integer part to octal and then we convert the fractional part to octal.

Converting Integer Part $(589)_{10}$ to Octal –

	Decimal	Remainders
8	589	
8	73	5
8	9	1
8	1	1
	0	1

Reading the remainders from bottom to top, the equivalent octal of (589)10 is $(1115)_8$.

Converting the Fractional Part (0.278)10 to Octal –

Decimal	Product	Carry
0.278 × 8	2.224	2
0.224 × 8	1.792	1
0.792 × 8	6.336	6
0.336 × 8	2.688	2

ɔm to obtain the fractional part of the equivalent octal number, the result is $(0.2162)_8$.

Thus, the equivalent octal number of $(589.278)_{10}$ is $(1115.2162)_8$.

Hexadecimal to Binary Conversion

ɔlace each digit of the given hexadecimal number by its equivalent 4-bit binary

llowing table shows the equivalent 4-bit binary group of each hexadecimal digit –

Hexadecimal	Binary			
	$(2)^3 = 8$	$(2)^2 = 4$	$(2)^1 = 2$	$(2)^0 = 1$
0	0	0	0	0
1	0	0	0	1
2	0	0	1	0
3	0	0	1	1
4	0	1	0	0
5	0	1	0	1
6	0	1	1	0
7	0	1	1	1
8	1	0	0	0
9	1	0	0	1

A (10)	1	0	1	0
B (11)	1	0	1	1
C (12)	1	1	0	0
D (13)	1	1	0	1
E (14)	1	1	1	0
F (15)	1	1	1	1

conversion of a hexadecimal number to binary number with the help of examples.

Example 1

Convert $(3A94.C5D)_{16}$ to binary equivalent.

Solution

The given hexadecimal number can be converted into equivalent binary number as follows –

Hexadecimal Number	3	A	9	4	.	C	5	D
	3	10	9	4	.	12	5	13
4-bit Binary Equivalent	0011	1010	1001	0100	.	1100	0101	1101

Convert $(ABD.2E)_{16}$ to binary equivalent.

The conversion of given hexadecimal number to its binary is done as follows

Hexadecimal Number	A	B	D	.	2	E
	10	11	13	.	2	14
4-bit Binary Equivalent	1010	1011	1101	.	0010	1110

Hence, the equivalent binary of $(ABD.2E)_{16}$ is $(101010111101.00101110)_2$.

Binary to Hexadecimal Conversion

To convert a given binary number to its equivalent hexadecimal number, we create groups of 4 bits each on both sides of the binary point. Then, we replace each group of 4-bit binary by the equivalent hexadecimal digit.

Let us understand the conversion of a binary number to its equivalent hexadecimal with the help of examples.

Example 1

Convert $(1110111001101.111011)_2$ to hexadecimal.

Solution

The conversion of the given binary number to hexadecimal equivalent is done as follows −

4-bit Group	1	1101	1100	1101	.	1110	11
	0001	1101	1100	1101	.	1110	1100
Hexadecimal Equivalent	1	D	C	D	.	E	C

Thus, the hexadecimal equivalent of the given binary number is $(1DCD.EC)_{16}$.

Example 2

Convert (110111110111.1100)2 to hexadecimal.

Solution

We can convert the given binary number into hexadecimal equivalent as follows –

4-bit Group	1101	1111	0111	.	1100
Hexadecimal Equivalent	D	F	7	.	C

Thus, the hexadecimal equivalent of (110111110111.1100)2 is $(DF7.C)_{16}$.

Hexadecimal to Decimal Conversion

To convert a hexadecimal number to its equivalent decimal number, we multiply each digit in the hexadecimal number by its positional weight and then add all the product terms to obtain the final result.

The step-by-step procedure to convert a hexadecimal number to its equivalent decimal number is explained below −

Step 1 − Write the positional weights for each hexadecimal digit.

Step 2 − Multiply each hexadecimal digit with its positional weight.

Step 3 − Add the product terms to obtain the equivalent decimal number.

Let us see some examples to understand the conversion of hexadecimal to decimal number.

Example 1

Convert $(5AB2.8C)_{16}$ to decimal equivalent.

Solution

The conversion of the given hexadecimal number to its decimal equivalent is given below –

Hex Digits	Decimal Equiv.	Positional Weights	Multiply	Product
5	5	$(16)^3$	$5 \times (16)^3$	20480
A	10	$(16)^2$	$10 \times (16)^2$	2560
B	11	$(16)^1$	$11 \times (16)^1$	176
2	2	$(16)^0$	$2 \times (16)^0$	2
.	.	.	.	.
8	8	$(16)^{-1}$	$8 \times (16)^{-1}$	0.5
C	12	$(16)^{-2}$	$12 \times (16)^{-2}$	0.0468

Add all the product terms to obtain the equivalent decimal,

$(5AB2.8C)_{16} = 20480 + 2560 + 176 + 2 + 0.5 + 0.0468 = (23218.5468)_{10}$

Example 2

Convert $(1AF.2)_{16}$ to decimal.

Solution

The decimal equivalent of the given hexadecimal number can be obtained as follows –

Hex Digits	Decimal Equiv.	Positional Weights	Multiply	Product
1	1	$(16)^{-2}$	$1 \times (16)^{-2}$	256
A	10	$(16)^{-1}$	$10 \times (16)^{-1}$	160
F	15	$(16)^{0}$	$15 \times (16)^{0}$	15
.	.	.	.	.
2	2	$(16)^{-1}$	$2 \times (16)^{-1}$	0.125

Adding the product terms to obtain the equivalent decimal number,

$(1AF.2)_{16} = 256 + 160 + 15 + 0.125 = (431.125)_{10}$

Decimal to Hexadecimal Conversion

If a mixed decimal number is given that has integer and fraction parts. Then, to convert the given decimal number to its equivalent hexadecimal, we convert integer and fraction parts separately.

To convert the integer part, we successively divide the decimal integer by 16 till the quotient is 0. The integer part of the equivalent hexadecimal is obtained by reading the remainders from bottom to top.

To convert the fractional part, we multiply the decimal fractional number by 16 till the product is 0 or till the desired accuracy is obtained. The fractional part of the equivalent hexadecimal is obtained by reading the carries from top to bottom.

Let us see some examples to understand the decimal to hexadecimal conversion.

Example : Convert $(524.26)_{10}$ to hexadecimal.

Solution : The given decimal number is a mixed number. Hence, we have to convert its integer and fractional parts separately.

Converting Integer Part $(524)_{10}$ to Hexadecimal –

Decimal	Remainders	
16	524	
16	32	12 (C)
16	2	0
	0	2

Reading the remainder from bottom to top to obtained the hexadecimal equivalent, the result is $(20C)_{16}$.

Converting Fractional Part $(0.26)_{16}$ to Hexadecimal –

Decimal	Product	Carry
0.26 × 16	4.16	4
0.16 × 16	2.56	2
0.56 × 16	8.96	8
0.96 × 16	15.36	15 (F)

Reading the carries from top to bottom to obtain the equivalent hexadecimal number, the result is $(0.428F)_{16}$.

Thus, the hexadecimal equivalent of the decimal number (524.26)10 is $(20C.428F)_{16}$.

Octal to Hexadecimal Conversion

The conversion of octal to hexadecimal is very simple. We first convert the given octal number to binary and then the binary number to the hexadecimal.
The step-by-step process to convert a given octal number to its equivalent hexadecimal is given below −
Step 1 − Convert each digit of the given octal number to its equivalent binary of 3-bits.
Step 2 − Make groups of 4 bits each of the obtained binary number.
Step 3 − Convert each 4-bit binary group to its equivalent hexadecimal.
Let us see some examples to understand the conversion of octal to hexadecimal.
Example 1 : Convert (742.35)8 to hexadecimal.
Solution : The conversion of given octal number to hexadecimal is explained below −

Octal Digits	3-bit Binary	4-bit Binary	Hex Digits
7	111	0001	1
4	100	1110	E
2	010	0010	2

.	.	.	.
3	011	0111	7
5	101	0100	4

Thus, the hexadecimal equivalent of the given octal number is $(1E2.74)_{16}$.

Example 2 : Convert $(1523.742)_8$ to hexadecimal.

Solution

The following table demonstrates the conversion of given octal number to hexadecimal –

Octal Digits	3-bit Binary	4-bit Binary	Hex Digits
1	001	0000	0
5	101	0011	3
2	010	0101	5
3	011	0011	3
.	.	.	.
7	111	1111	F
4	100	0001	1
2	010	0000	0

Hence, the hexadecimal equivalent of the given octal number is $(353.F1)_{16}$.

Hexadecimal to Octal Conversion

The hexadecimal to octal conversion can be perform in the same way as the octal to hexadecimal as explained above. To convert a given hexadecimal number to octal number, we first convert the given hexadecimal number to binary and then the binary number to the octal.

The step-by-step procedure to convert a hexadecimal number to its equivalent octal number is explained below −

Step 1 − Convert each hexadecimal digit to its equivalent 4-bit binary.

Step 2 − Make groups of 3-bits each of the obtained binary number.

Step 3 − Convert each 3-bit binary group to its equivalent octal number.

The following examples demonstrate the method of converting a given hexadecimal number to its equivalent octal number.

Example 1

Convert $(B3A9.5F)_{16}$ to octal.

Solution

The conversion of the given hexadecimal number to its equivalent octal number is explained below –

Hex Digits			B	3	A	9	.	5	F	
4-bit Binary			1011	0011	1010	1001	.	0101	1111	
3-bit Binary	001	011	001	110	101	001	.	010	111	110
Octal Digits	1	3	1	6	5	1	.	2	7	6

Thus, the octal equivalent of the given hexadecimal number is $(131651.276)_{8.}$

Example 2

Convert (AC.F)16 to octal.

Solution : The conversion of given hexadecimal number to its equivalent octal number is demonstrated below −

Hex Digits		A	C	.	F	
		10	12	.	15	
4-bit Binary		1010	1100	.	1111	
3-bit Binary	010	101	100	.	111	100
Octal Digits	2	5	4	.	7	4

Hence, the octal equivalent of the given hexadecimal number is (254.74)8.

What is Boolean Algebra?

Boolean algebra is a mathematics that provides various operators and rules to perform arithmetic and algebraic operations on binary variables and numbers. Boolean algebra is based on the binary number system. It has two possible values i.e., 0 and 1. Here, the value 0 represents the False state, while the value 1 represents the True state.

The operations in Boolean algebra are based on the three fundamental logical operations namely, AND, OR, and NOT.

Logical Operations in Boolean Algebra

AND Operation

In Boolean algebra, a logical operation in which the outcome is true (1) only when all the input values are true (1), otherwise, the output is false (0) is termed as AND operation. The AND operation is represented by a dot (.). For example, A AND B can be represented as A.B in symbolic form.

OR Operation

In Boolean algebra, the OR operation is another logical operation in which the output is false (0) only when all input values are false (0), otherwise the output is true (1). The OR operations is denoted by a plus (+). For example, A OR B can be represented as A + B.

NOT Operation

In Boolean algebra, the NOT operation is performed to obtain the inverted version of the input value. Thus, the result of the NOT operation is false (0), if the input is true (1) and vice-versa. The NOT operation is represented by the symbol "~". For example, NOT A is represented as ~A.

Boolean Variable

A Boolean variable is a symbol that can take one of the two possible binary values i.e., 0 and 1.

Boolean Value

It is nothing but a value representing the state of a variable. It can be either True (1) or False (0).

Laws of Boolean Algebra

All the important laws and rules of Boolean algebra are explained below −

Rules of Logical Operations

There are three basic logical operations namely, AND, OR, and NOT. The following table highlights the rules associated with these three logical operations –

AND Operation	OR Operation	NOT Operation
0 AND 0 = 0	0 OR 0 = 0	NOT of 0 = 1
0 AND 1 = 0	0 OR 1 = 1	NOT of 1 = 0
1 AND 0 = 0	1 OR 0 = 1	
1 AND 1 = 1	1 OR 1 = 1	

These rules of Boolean algebra can be implemented using logic gates.

AND Laws

In Boolean algebra, there are four AND laws given below –

Law 1 − $A \cdot 0 = 0$ (This law is called null law).

Law 2 − $A \cdot 1 = A$ (This law is called identity law).

Law 3 − $A \cdot A = A$

Law 4 − $A \cdot A' = 0$

OR Laws

There are four OR laws described below –

Law 1 − $A + 0 = A$ (This law is called null law).

Law 2 − $A + 1 = 1$ (This law is called identity law).

Law 3 − $A + A = A$

Law 4 − $A + A' = 1$

Complementation Laws

There are following five complementation laws in Boolean algebra −

Law 1 − 0' = 1

Law 2 − 1' = 0

Law 3 − If A = 0, Then A' = 1

Law 4 − If A = 1, Then A' = 0

Law 5 − (A')' = A (This is called double complementation law)

Commutative Laws

There are following two commutative laws in Boolean algebra −

Law 1 − According to this law, the operation A OR B produces the same output as the operation B OR A, i.e., $A + B = B + A$

Hence, the order of the variables does not affect the OR operation.This law can be extended to any number of variables. For example, for three variables, it will be,

$$A + B + C = C + B + A = B + C + A = C + A + B$$

Law 2 − According to this law, the output of the A AND B operation is same as that of the B AND A operation, i.e., $A \cdot B = B \cdot A$

This law states that the order in which the variables are ANDed does not affect the result. We can extend this law to any number of variables. For example, for three variables, we get,

$$A \cdot B \cdot C = A \cdot C \cdot B = C \cdot B \cdot A = C \cdot A \cdot B$$

Associative Laws

Associative laws define the ways of grouping the variables. There are two associative laws as described below.

Law 1 − The expression A OR B ORed with C results the same as the A Ored with B OR C, i.e.,

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

This law can be extended to any number of variables. For example, for 4 variables, we get,

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

Law 2 − The expression A AND B ANDed with C results the same as the expression A ANDed with B AND C, i.e.,

$$(A \cdot B) \cdot C = A \cdot (B \cdot C)$$

We can extend this law to any number of variables. For example, if we have 4 variables, then

$$(ABC)D = A(BCD) = (AB)\cdot(CD)$$

Distributive Laws

In Boolean algebra, there are the following two distributive laws that allow for multiplying or factoring out of expressions.

Law 1 − According to this law, we OR several variables and then AND the result with a single variable.

It gives the same result as the expression in which the single variable is ANDed with each of the several variables and then ORed the product terms, i.e.,

$$A \cdot (B + C) = AB + AC$$

We can extend this law to any number of variables. For example,

$$A(BC + DE) = ABC + ADE$$

$$AB(CD + EF) = ABCD + ABEF$$

Law 2 − According to this law, if we AND several variables and then the result is ORed with a single variable.

It gives the same result as we OR the single variable with each of the several variables and then the sum terms are ANDed together, i.e.,

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

Proof − The proof of this law is explained here,

$$RHS = (A + B)(A + C)$$

$$= AA + AB + AC + BC$$

$$= A + AB + AC + BC$$

$$= A\,(1 + B + C) + BC$$

Since,

$$1 + B + C = 1 + C = 1$$

Therefore,

$$A \cdot 1 + BC = A + BC = LHS$$

Redundant Literal Rule (RLR)

Under this rule, there are two laws in Boolean algebra, which are explained here.

Law 1 − According to this law, if we OR a variable with the AND of the complement of the variable and another variable. Then, it is same as the OR of the two variables, i.e.,

$$A + AB = A + B$$

Proof − The proof of this law is explained here,

$$LHS = A + AB = (A + A)(A + B)$$

$$= 1 \cdot (A + B) = A + B = RHS$$

Law 2 − According to this law, if we AND a variable with the OR of the complement of the variable and another variable, it is equivalent to when we AND the two variables, i.e.,

$$A(A + B) = AB$$

Proof − This law can be proved as follows,

$$LHS = A(A + B) = AA + AB$$

$$= 0 + AB = AB = RHS$$

Both these laws show that the complement of a term appearing in another term is redundant. Hence, the rule is named as Redundant Literal Rule.

Idempotence Laws

The term "idempotence" is a synonym for "same value". There are two idempotence laws in Boolean algebra. They are,

Law 1 − According to this law, ANDing a variable with itself is equal to the variable, i.e.,

$$A \cdot A = A$$

Law 2 − According to this law, ORing a variable with itself is equal to the variable, i.e.,

$$A + A = A$$

Absorption Laws

There are two absorption laws in Boolean algebra and they are explained below.

Law 1 − According to this law, if we OR a variable with the AND of the that variable and another variable, then it is equal to the variable itself, i.e.,

$$A + A \cdot B = A$$

This can be proved as follows,

$$LHS = A + A \cdot B = A \cdot (1 + B)$$

$$= A \cdot 1 = A = RHS$$

Law 2 − According to this law, the AND of a variable with the OR of that variable and another variable is equivalent to the variable itself i.e.,

$$A(A + B) = A$$

This can also be proved as follows,

$$LHS = A (A + B) = AA + AB$$

$$= A + AB = A (1 + B) = A \cdot 1 = A = RHS$$

Hence, this law proves that if a term appears in another term, then the latter term will become redundant and can be removed from the expression.

DeMorgan's Theorem

In Boolean algebra, DeMorgans theorem defines two laws which are explained below.

Law 1 − According to this law, the complement of a sum of variables is equivalent to the product of complement of each of the variables, i.e.,

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

This law can be extended to any number of variables.

Law 2 − The second law of DeMorgans theorem states that the complement of a product of variables is equivalent to the sum of complement of each of the variables, i.e., Type equation here.

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

This law can also be extended to any number of variables.

SOP (Sum of Products) Form

The SOP or Sum of Products form is a form of expressing a logical or Boolean expression. In SOP, different product terms of input variables are logically ORed together. Therefore, in the case of SOP form, we first logically AND the input variables, and then all these product terms are summed together with the help of logical OR operation.

For example – $$f(A,B,C) = ABC + \overline{A}BC + AB\overline{C}$$

This is a logical expression in three variables. Here, ABC, A'BC, and ABC' are the three product terms which are summed together to get the expression in SOP form.

POS (Product of Sum) Form

The POS or Product of Sum form is another form used to represent a logical expression. In POS form, different sum terms of input variables are logically ANDed together. Hence, if we want to express a logical expression in POS form, for that we first logically OR all the input variables and then these sum terms are ANDed using AND operation.

For example –

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

Here, f is a logical expression in three variables. From this example, it can be seen that there are three sum terms which are ANDed together to obtain the POS form of the given expression.

Unit-2

What is a Logic Gate?

A logic gate is an electronic circuit that performs logical operations based on the inputs provided to it and produces a logical output that can be either "true" or "false".

Logic gates are the primary building blocks of all digital circuits and systems. The operation of logic gates is based on the Boolean mathematics.

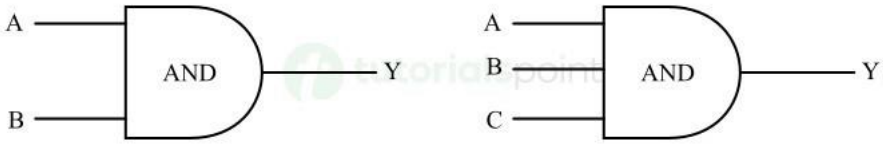
Types of Logic Gates

Logic gates can be broadly classified into the following three categories −
Basic Logic Gates − AND Gate, OR Gate, NOT Gate
Universal Logic Gates − NAND Gate and NOR Gate
Derived Logic Gates − XOR Gate and XNOR Gate
All these gates are combined together to implement complex practical digital systems to perform various computational and logical operations.

What is an AND Gate?

An AND gate is a type of basic logic gate used in various digital circuits and systems. It produces a high or logic 1 or True output, only when all its inputs are high or logic 1 or true. For all other combinations of inputs, it produces a low or logic 0 or False output.

The logic symbols for the two and three input AND gates are depicted in the following figure.



Here, A, B, and C are the input variables and Y is the output variable

Truth Table of AND Gate.

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

From this truth table of two-input AND gate, it can be observed that the output of the AND gate is logic 1 or high, only when both inputs are logic 1 or high.

The following table represents the truth table of a three-input AND gate –

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

This truth table shows that the output is high or logic 1 only when all the three inputs to the AND gate are high or logic 1. For rest input combinations, the output is low or logic 0.

Boolean Expression of AND Gate

The Boolean expression of a two-input AND gate is given by,

$$Y = A \cdot B$$

Where, A and B are inputs and Y is the output. This expression is read as "Y is equal to A AND B." The dot (·) symbol represents the AND operation.

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

$$Y = A \cdot B \cdot C$$

It is read as "Y is equal to A AND B AND C".

In the same way, we can obtain the Boolean expression for an AND gate having any number of input variables.

What is an OR Gate?

An OR gate is a type of logic gate used to perform logical addition. It can have two or more inputs and one output.

The logic symbols for a two-input and a three-input OR gate are shown in the following figure.

Truth Table of OR Gate

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

The following table shows the truth table for a three-input OR gate –

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

From these two truth tables, we can observe that the output of the OR gate is logic 0 or low, only when all the inputs to the OR gate are logic 0 or low. Otherwise, the output of the OR gate is high or logic 1.

Boolean Expression of OR Gate

The Boolean expression of a two-input OR gate is given below –

$Y = A + B$

Similarly, the Boolean expression of a three-input OR gate is given below −

Y = A + B + C

Here, A, B, and C are the inputs and Y is the output.

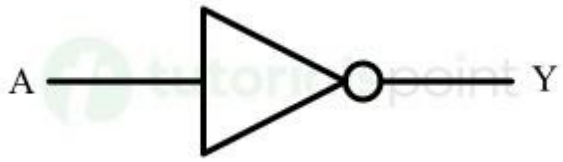
In the same way, we can extend this expression to any number of input variables.

What is a NOT Gate?

The NOT gate is a type of basic logic gate used in digital electronics to implement the inversion function. Since it performs the inversion operation, it is also known as inverter.

It has only one input line and one output line. The output of the NOT gate is high or logic 1 when its input is low or logic 0. The output of the NOT gate is low or logic 0 when its input is high or logic 1.

The logic symbol of the NOT gate is shown in the following figure –



The truth table of NOT gate is a table of input and output that represent the relationship between them. Here is the truth table of the NOT gate –

Input (A)	Output (Y)
0	1
1	0

Boolean Expression of NOT Gate

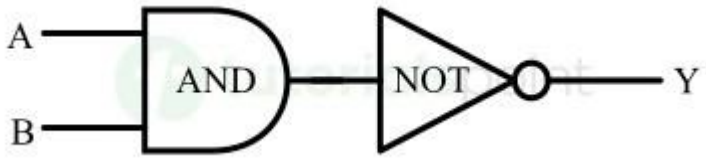
The Boolean expression of the NOT gate is given below −

$$Y=A^{-}=A'$$

Universal Gates NAND and NOR

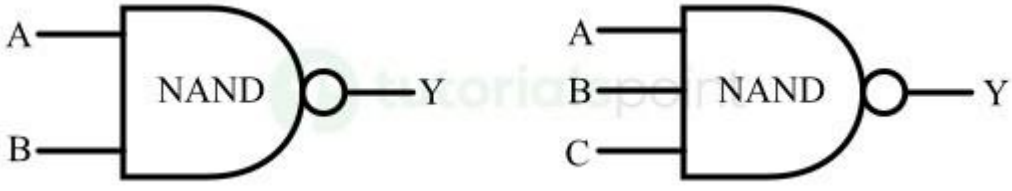
What is a NAND Gate?

The NAND gate is a universal gate that basically a combination of two basic logic gates namely, AND gate and NOT gate.



Logic Symbol of NAND Gate

The logic symbols of a two-input and three-input NAND gates are depicted in the following figure.



Truth Table of NAND Gate

Truth table is a table of inputs and outputs of a NAND gate showing the relationship between them. Here is the truth table of a two-input NAND gate –

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

The truth table for a three-input NAND gate is given below –

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

From these two truth tables, we can observe that the NAND gate produces a low or logic 0 output only when all its inputs are high or logic 1. For any other input combinations, the output is high or logic 1.

Boolean Expression of NAND Gate

The Boolean expression is a logical function that describes the logical relationship between inputs and output of a NAND gate.

The Boolean expression for a two-input NAND gate is give below –

$$Y=\overline{AB}= (AB)'$$

The Boolean expression for a three-input NAND gate is given by,

$Y=\overline{ABC}$= (ABC)'

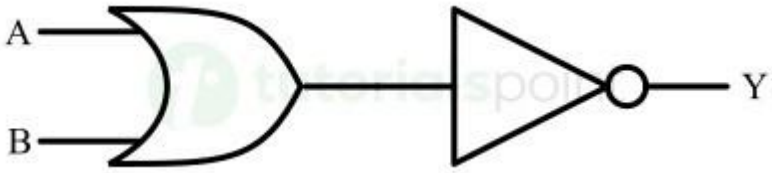
Here, A, B, and C are the input variables and Y is the output variable.

Applications of NAND Gate

Alarm circuits
Buzzer and burglar devices
Automatic temperature regulation systems
Security systems
Automated doors and windows, etc.

What is a NOR Gate?

The NOR gate is another universal gate used in digital electronics to implement Boolean functions. It is a combination of two basic logic gates namely, OR gate and NOT gate. The NOR gate is designed by connecting a NOT gate to the output line and the final output is taken from the output line of the NOT gate as shown in the following figure



The NOR gate can have two or more input lines and one output line. The output of the NOR gate is high or logic 1 only when all its inputs are low or logic 0. For all other input combinations, the output of the NOR gate is low or logic 0.

Logic Symbol of NOR Gate

The logic symbols of a two-input and three-input NOR gates are shown in the following figure.

Truth Table of NOR Gate

The truth table of the NOR gate specifies the output for different input combinations. The truth table of a two-input NOR gate is given below –

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

The following is the truth table of a three-input NOR gate –

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

From these truth tables, we can observe that the output of the NOR gate is high or logic 1 only when all its inputs are low or logic 0, otherwise the output is low or logic 0.

Applications of NOR Gate

Various digital systems
Industrial automation and control systems
Traffic control systems
Alarm circuits
Digital arithmetic circuits like adders and subtractors, etc.

What is an XOR Gate?

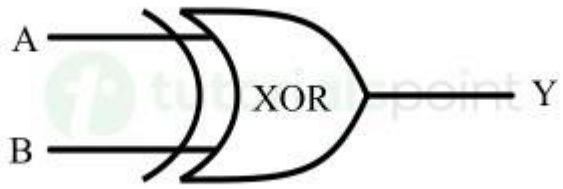
The XOR gate is a type of logic gate in digital electronics which has two inputs and one output. XOR gate is also called Exclusive OR gate or Ex-OR gate. This logic gate is widely used in digital arithmetic circuits like adders and subtractors.

Since the output of the XOR gate is high only when both of the inputs are dissimilar, it is also known as the inequality detector.

It is very important to note that there is no such thing like three or more input XOR gate. Hence, when we need XOR gate for more than two input variables, we use two or more two-input XOR gates.

Logic Symbol of XOR Gate

The logic symbol of an XOR gate is shown in the following figure.



Truth Table of XOR Gate

The truth table of XOR gate is a table that represents the relationship between its inputs and output. The truth table of an XOR gate is given below –

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

1 1 0

From this truth table, we can observe that the output of the XOR gate is high or logic 1 only when the both inputs are different. In the case, when both inputs are similar the output is low or logic 0.

Boolean Expression of XOR Gate

$$Y=A \oplus B$$

This equation can also be expressed as below –

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

Here, the symbol "⊕" denotes the XOR operation.

XOR Gate as an Inverter

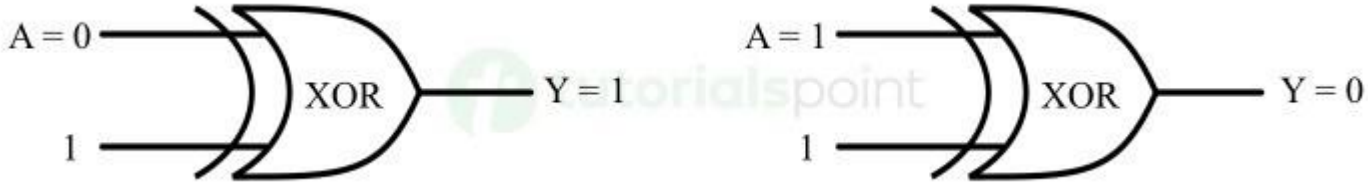
The XOR gate can also be used as an inverter. There is a property of XOR operation that is,.

$$A \oplus 1 = \overline{A}$$

XOR Gate as a Buffer

A buffer gate is a logic gate that produces the same output as the input. It is used to provide some delay in the input and output.

There is a property of XOR logic that is,



Applications of XOR Gate

The following are some key applications of the XOR gate –

1. XOR gate is used in computational and arithmetic circuits like adders, subtractors, etc.
2. XOR gate is used to detect errors, similarities and dissimilarities between two logic levels or signals.
3. XOR gate is used as a controlled inverter or buffer logic.

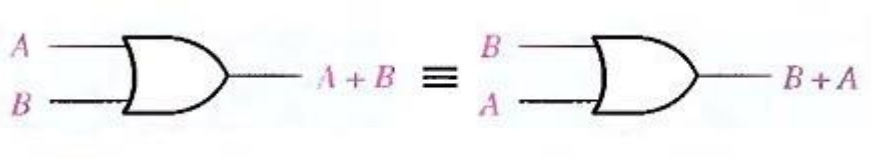
Name	AND form	OR form
Identity law	$1A = A$	$0 + A = A$
Null law	$0A = 0$	$1 + A = 1$
Idempotent law	$AA = A$	$A + A = A$
Inverse law	$A\bar{A} = 0$	$A + \bar{A} = 1$
Commutative law	$AB = BA$	$A + B = B + A$
Associative law	$(AB)C = A(BC)$	$(A + B) + C = A + (B + C)$
Distributive law	$A + BC = (A + B)(A + C)$	$A(B + C) = AB + AC$
Absorption law	$A(A + B) = A$	$A + AB = A$
De Morgan's law	$\overline{AB} = \bar{A} + \bar{B}$	$\overline{A + B} = \bar{A}\bar{B}$

BOOLEAN ALGEBRA AND LOGIC SIMPLIFICATION

Commutative Laws

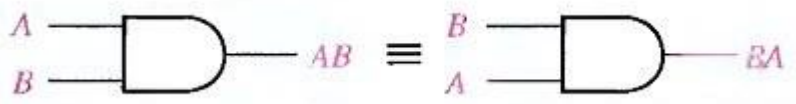
► The commutative law of addition for two variables is written as

A+B = B+A



(Fig .Application of commutative law of addition.)

The commutative law of multiplication for two variables is A.B = B.A

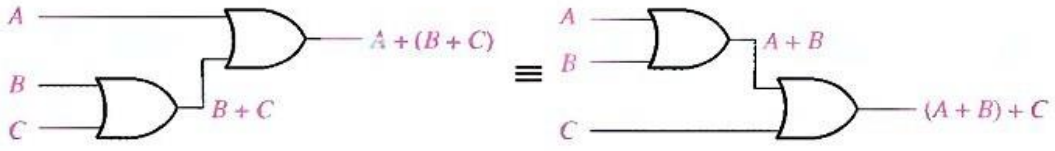


(Application of commutative law of multiplication.)

Associative Laws :

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

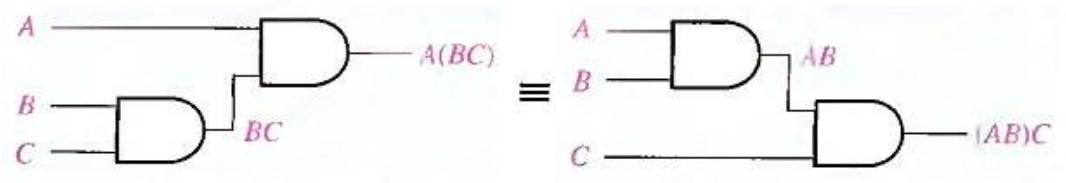
A + (B + C) = (A + B) + C



(Application of associative law of addition.)

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

A(BC) = (AB)C



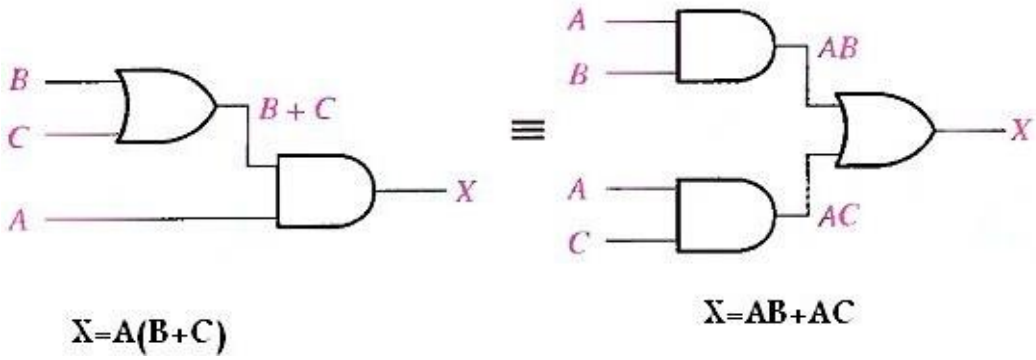
(Application of associative law of multiplication.)

Distributive Law:

► The distributive law is written for three variables as follows:

A(B + C) = AB + AC

illustrates the distributive law in terms of gate implementation.



X=A(B+C) **X=AB+AC**

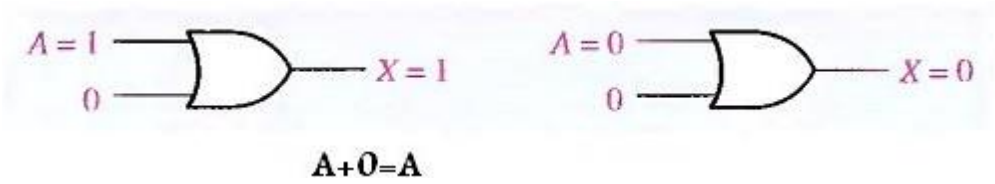
(Application of distributive law.)

Basic rules of Boolean algebra.

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

A, B, or *C* can represent a single variable or a combination of variables.

Rule 1. A + 0 = A

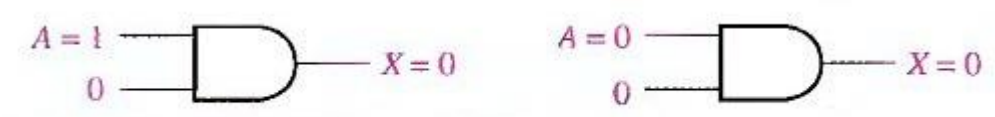


A+0=A

Rule 2. A + 1 = 1

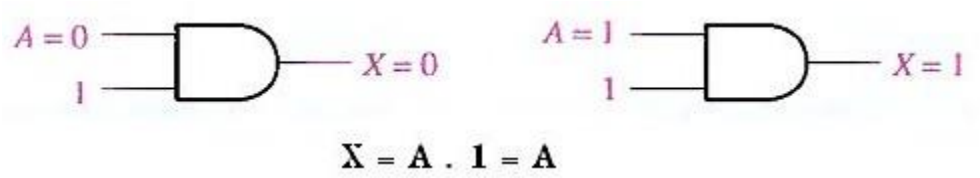
X=A+1=1

Rule 3. A . 0 = 0



X= A.0 = 0

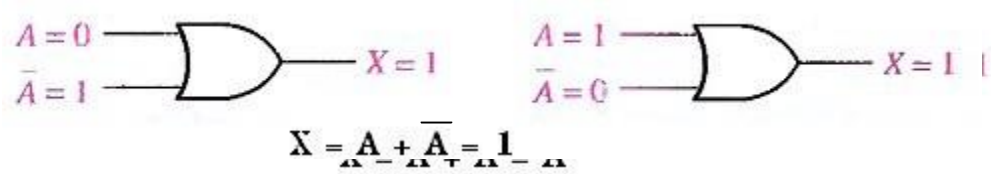
Rule 4. **A . 1 = A**



X = A . 1 = A

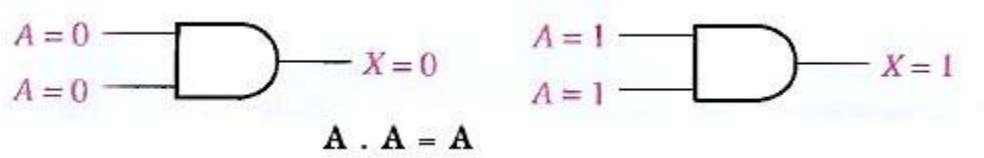
Rule 5. A + A = A

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



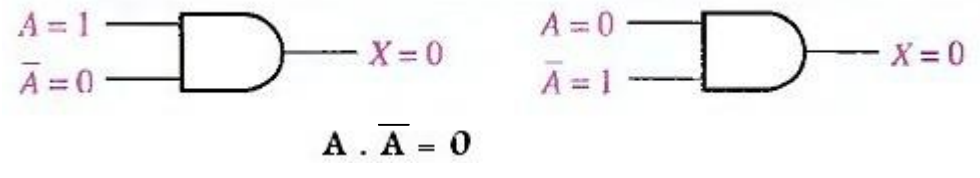
X = A + Ā = 1

Rule 7. A . A = A



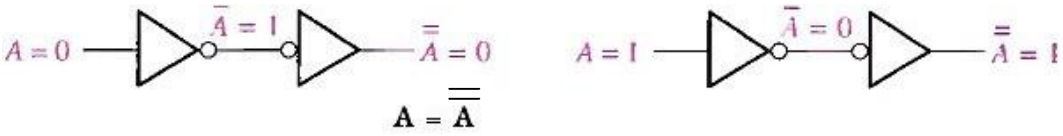
A . A = A

Rule 8. A . $\overline{A}$ = 0



A . Ā = 0

Rule 9 A = $\overline{\overline{A}}$



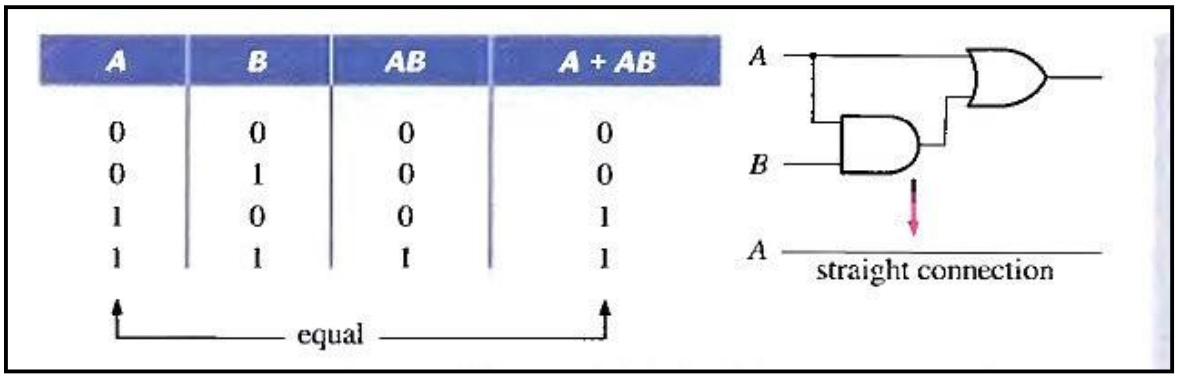
A = Ā

Rule 10. A + AB = A

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

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

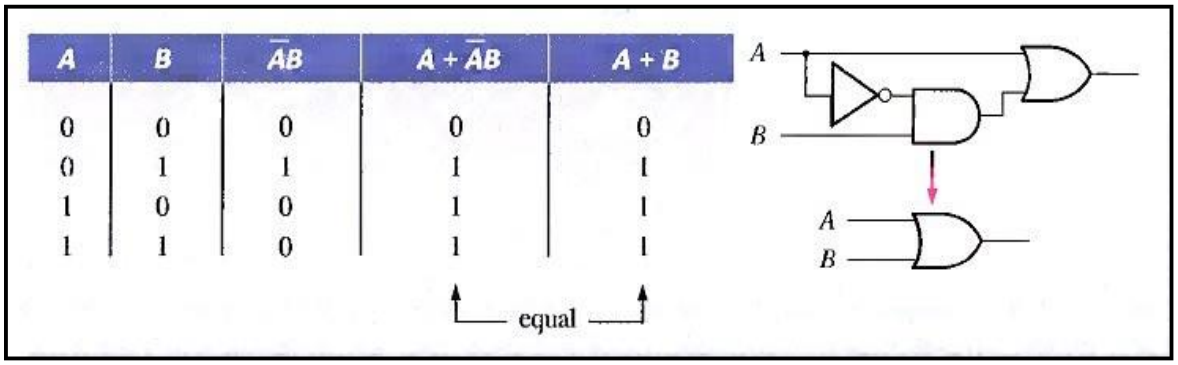
The proof is shown in Table 4-2, which shows the truth table and the resulting logic circuit simplification.



Rule 11. A + AB = A + B

This rule can be proved as follows:

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



The proof is shown in Table 4-3, which shows the truth table and the resulting logic circuit simplification.

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

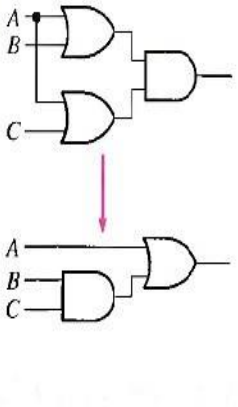
This rule can be proved as follows:

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

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

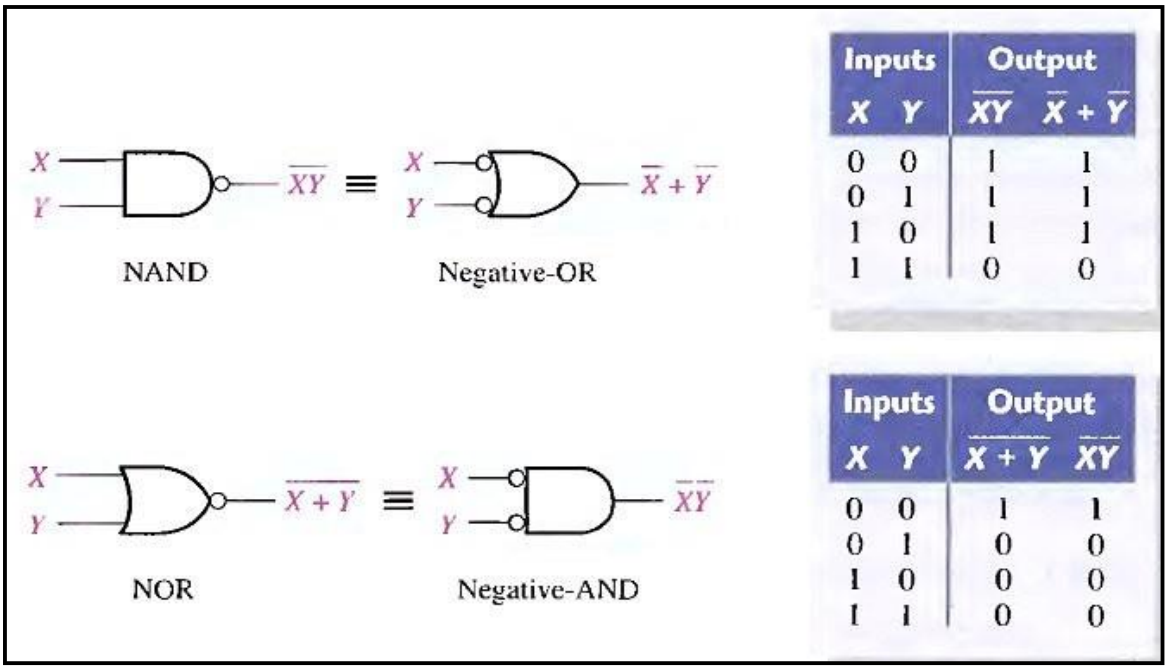
A	B	C	A + B	A + C	(A + B)(A + C)	BC	A + BC
0	0	0	0	0	0	0	0
0	0	1	0	1	0	0	0
0	1	0	1	0	0	0	0
0	1	1	1	1	1	1	1
1	0	0	1	1	1	0	1
1	0	1	1	1	1	0	1
1	1	0	1	1	1	0	1
1	1	1	1	1	1	1	1

equal



DEMORGAN'S THEOREMS

One of DeMorgan's theorems is stated as follows:
The complement of a product of variables is equal to the sum of the complements of the variables,
Stated another way,
The complement of two or more ANDed variables is equivalent to the OR of the complements of the individual variables.
The formula for expressing this theorem for two variables is $\overline{\overline{X}\overline{Y}} = \overline{X} + \overline{Y}$
DeMorgan's second theorem is stated as follows:
The complement of a sum of variables is equal to the product of the complements of the variables.
Stated another way,
The complement of two or more ORed variables is equivalent to the AND of the complements of the individual variables,
The formula for expressing this theorem for two variables is $\overline{X + Y} = \overline{X}\,\overline{Y}$

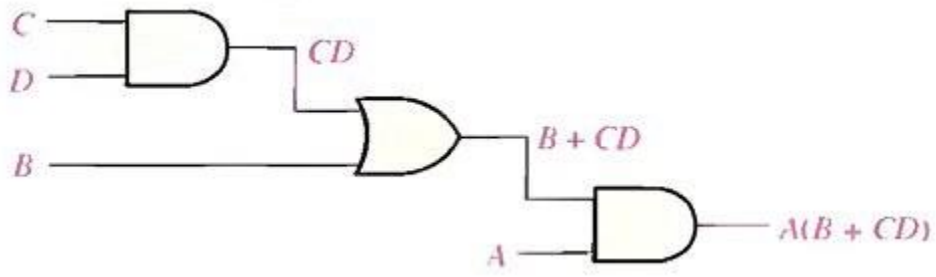


Inputs		Output	
X	Y	$\overline{XY}$	$\overline{X} + \overline{Y}$
0	0	1	1
0	1	1	1
1	0	1	1
1	1	0	0

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

BOOLEAN ANALYSIS OF LOGIC CIRCUITS

Example - A(B + CD)



Constructing a Truth Table for a Logic Circuit A truth table that shows the output for all possible values of the input variables can be developed. in Fig.(4-16), there are four input variables (A, B, C, and D) and therefore sixteen ($2^4 = 16$) combinations of values are possible.

Putting the Results in Truth Table format

The first step is to list the sixteen input variable combinations of 1s and 0s in a binary sequence as shown in Table 4-5.

INPUTS				OUTPUT
A	B	C	D	A(B + CD)
0	0	0	0	0
0	0	0	1	0
0	0	1	0	0
0	0	1	1	0
0	1	0	0	0
0	1	0	1	0
0	1	1	0	0
0	1	1	1	0
1	0	0	0	0
1	0	0	1	0
1	0	1	0	0
1	0	1	1	1
1	1	0	0	1
1	1	0	1	1
1	1	1	0	1
1	1	1	1	1

SIMPLIFICATION USING BOOLEAN ALGEBRA

A simplified Boolean expression uses the fewest gates possible to implement a given expression.

Example

Using Boolean algebra techniques, simplify this expression:

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

Solution

Step 1: Apply the distributive law to the second and third terms in the expression, as follows:

$$AB + AB + AC + BB + BC$$

Step 2: Apply rule 7 (BB = B) to the fourth term.

$$AB + AB + AC + B + BC$$

Step 3: Apply rule 5 (AB + AB = AB) to the first two terms.

$$AB + AC + B + BC$$

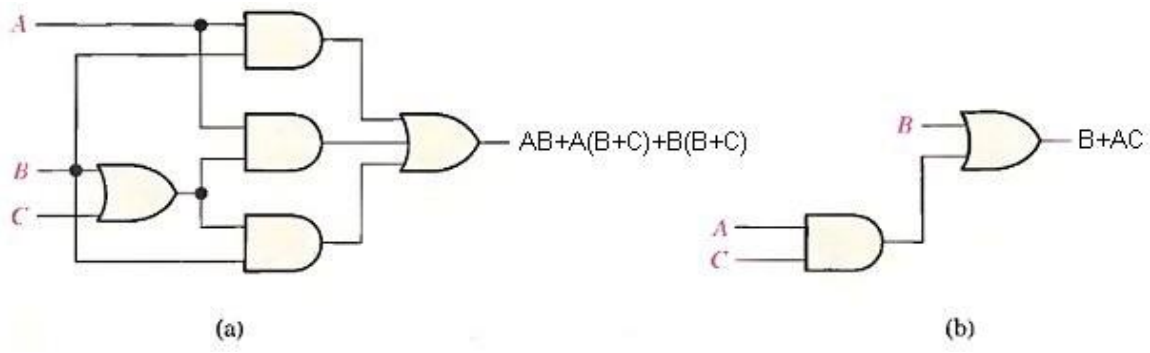
Step 4: Apply rule 10 (B + BC = B) to the last two terms.

$$AB + AC + B$$

Step 5: Apply rule 10 (AB + B = B) to the first and third terms.

$$B+AC$$

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



example

1. Express the Boolean function F = $\bar{x}\,\bar{y}$ z + x $\bar{y}\,\bar{z}$ + x y z ,Truth Table and Logic Gate.
2. Express the Boolean function $F = A + \bar{B}C$,Truth Table and Logic Gate.

Example

Determine the truth table for the following standard POS expression:

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

Solution

There are three variables in the domain and the eight possible binary values are listed in the left three columns of. The binary values that make the sum terms in the expression equal to 0 are A+B+C : 000, A+$\bar{B}$+C : 010, A+$\bar{B}$+$\bar{C}$: 011 $\bar{A}$+B+$\bar{C}$: 101and $\bar{A}$+$\bar{B}$+C : 110

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

INPUTS			OUTPUT	
A	B	C	X	SUM TERM
0	0	0	0	$(A + B + C)$
0	0	1	1	
0	1	0	0	$(A + \overline{B} + C)$
0	1	1	0	$(A + \overline{B} + \overline{C})$
1	0	0	1	
1	0	1	0	$(\overline{A} + B + \overline{C})$
1	1	0	0	$(\overline{A} + \overline{B} + C)$
1	1	1	1	

Karnaugh Maps (Kmap)

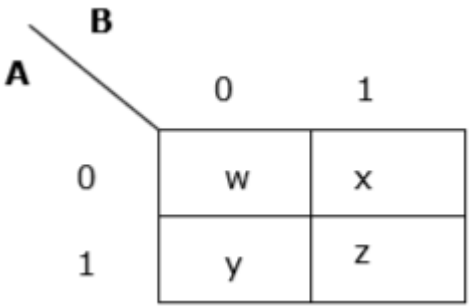
It provides grouping together Boolean expressions with common factors and eliminates unwanted variables from the expression. In a K-map, crossing a vertical or horizontal cell boundary is always a change of only one variable.

Example

An arbitrary truth table is taken below −

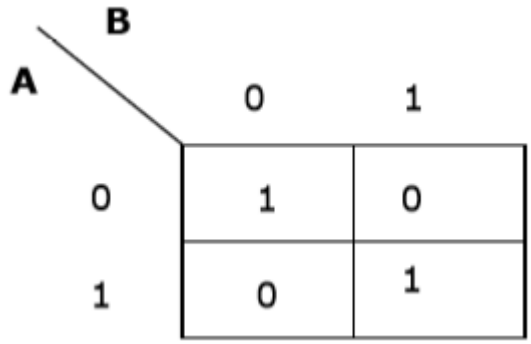
A	B	A operation B
0	0	w
0	1	x
1	0	y
1	1	z

Now we will make a k-map for the above truth table –



Example 2

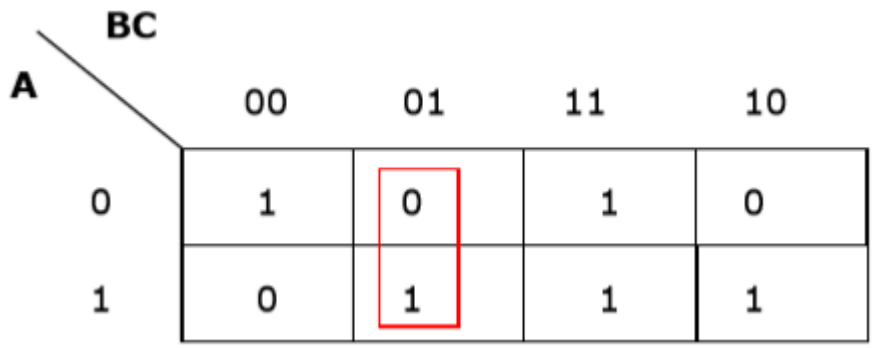
Now we will make a K-map for the expression − AB+ AB



Simplification Using K-map

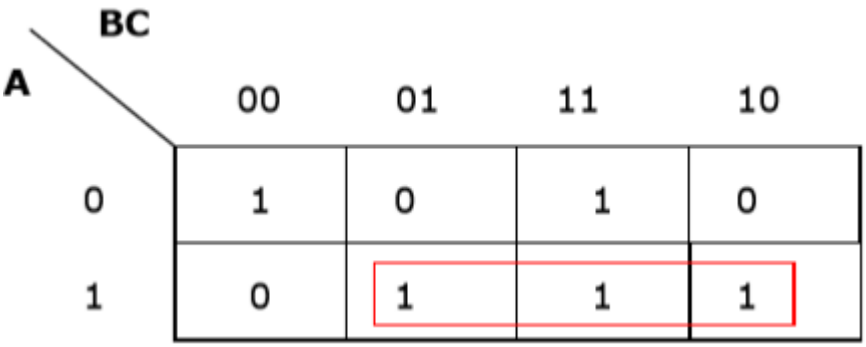
K-map uses some rules for the simplification of Boolean expressions by combining together adjacent cells into single term. The rules are described below –

Rule 1 − Any cell containing a zero cannot be grouped.



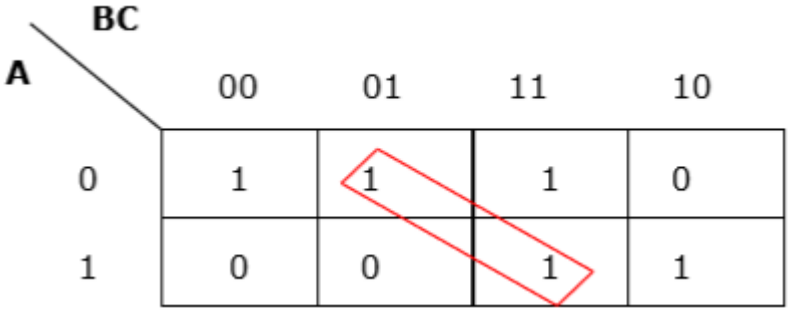
Wrong grouping

Rule 2 − Groups must contain 2n cells (n starting from 1).

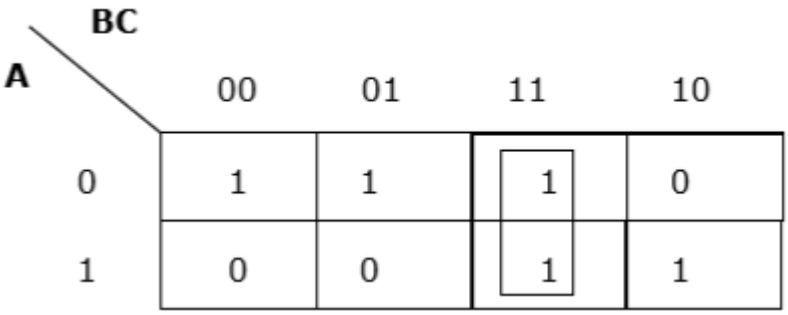


Wrong grouping

Rule 3 − Grouping must be horizontal or vertical, but must not be diagonal.

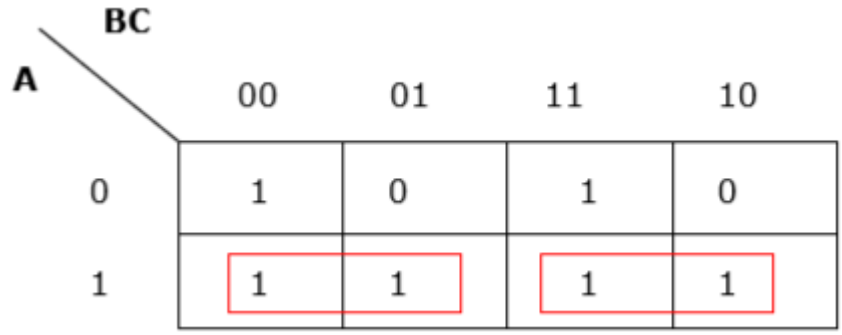


Wrong diagonal grouping

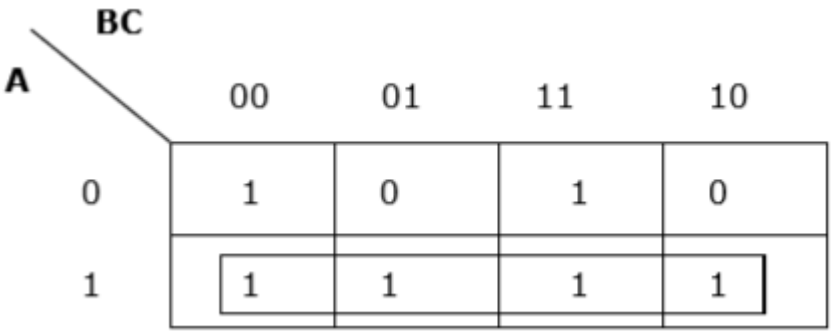


Proper vertical grouping

Rule 4 − Groups must be covered as largely as possible.

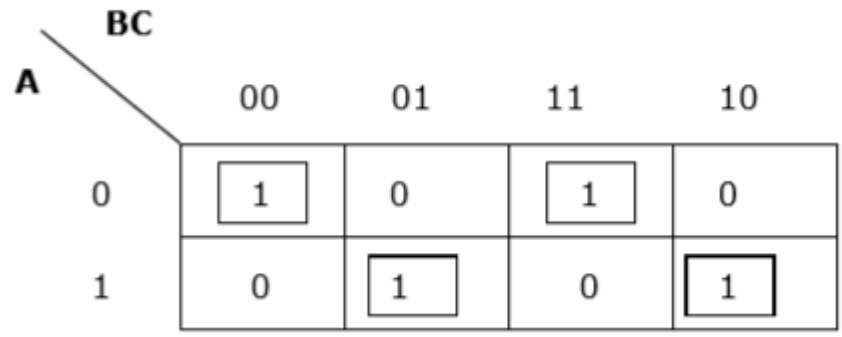


Insufficient grouping



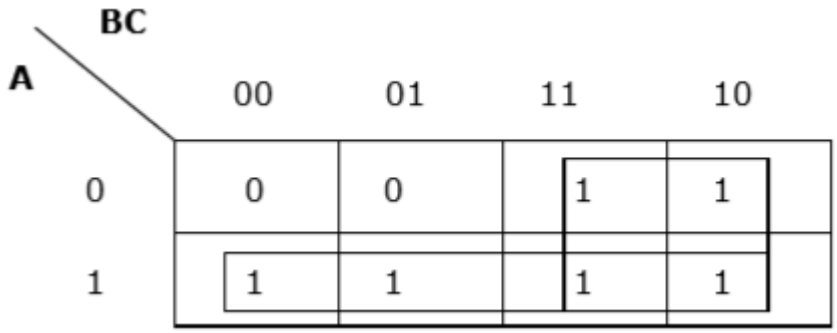
Proper grouping

Rule 5 − If 1 of any cell cannot be grouped with any other cell, it will act as a group itself.



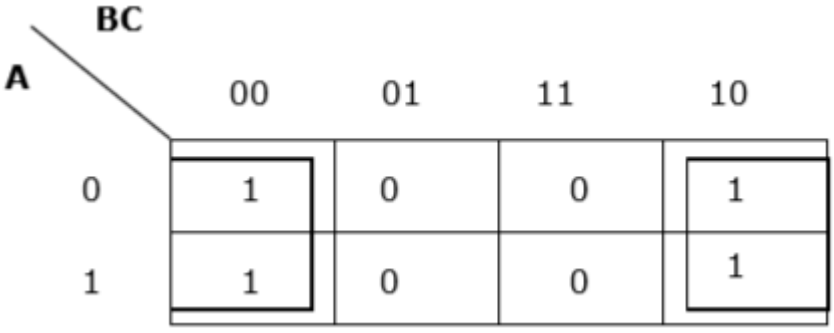
Proper grouping

Rule 6 − Groups may overlap but there should be as few groups as possible.



Proper grouping

Rule 7 – The leftmost cell/cells can be grouped with the rightmost cell/cells and the topmost cell/cells can be grouped with the bottommost cell/cells.



Proper grouping

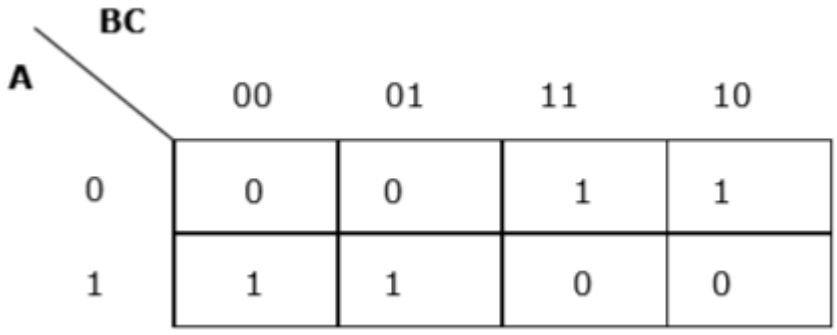
Problem

Minimize the following Boolean expression using K-map –

F(A,B,C)=A’BC+A’BC’+AB’C’+AB’C

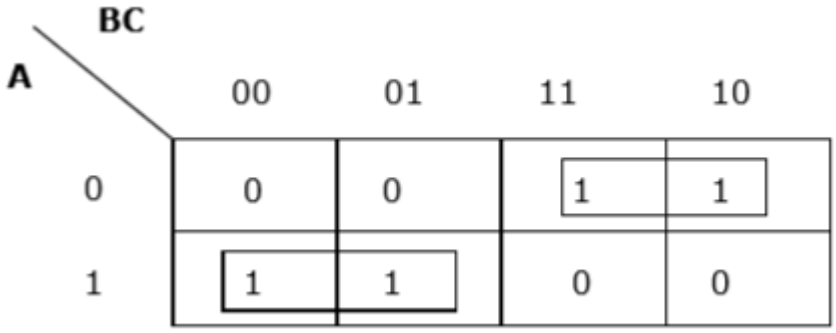
Solution

Each term is put into k-map and we get the following –



K-map for F (A, B, C)

Now we will group the cells of 1 according to the rules stated above –



K-map for F (A, B, C)

We have got two groups which are termed as AB and AB . Hence, F(A,B,C)=AB+AB=A⊕B . It is the minimized form.

Example 1

Simplify the following 3-variable Boolean function in SOP form using K-Map.

F(P,Q,R)=∑m(0,1,3,5,7)

Solution

The K-Map representation of the given Boolean function is shown in Fig

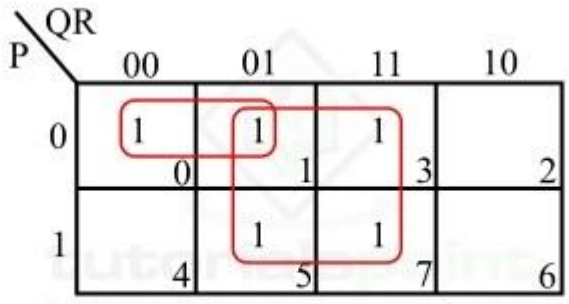


Figure 1

The simplification of this K-map is done as per the following steps −

There are no isolated 1s.

The minterm m1 forms a 4-square with minterms m3, m5, and m7. Make it and read it as R.

The minterm m0 forms a 2-square with the minterm m1. Make it and read it as P^-Q^-

Write all the product terms in SOP form.

Thus, the simplified SOP expression is,

$F=R+P^-Q^-$

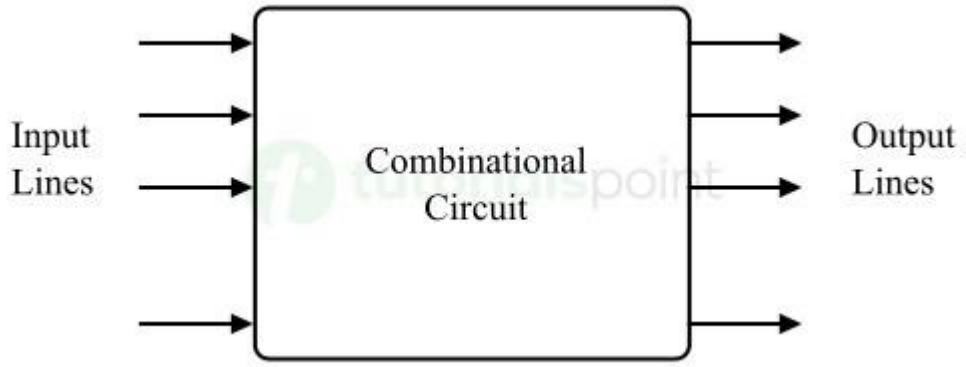
UNIT-3

Combinational Circuits :

A combinational circuit, also called a combinational logic circuit, it is a digital electronic circuit whose output is determined by present inputs only .Therefore, a combinational circuit is considered to not have a memory element in its circuit that stores previous inputs and outputs.

Block Diagram of Combinational Circuit

The following figure depicts the block diagram of a combinational logic circuit.



There are only three key elements in the circuit diagram of a combinational circuit, they are −

Input Lines − The input lines are used to enter the input values into the combinational circuit.

Processing Unit − It is the main element that processes the input values depending on the type of the circuit. For example, a full adder adds three binary bits.

Output Lines − The output lines are used to take results generated by the circuit.

Characteristics of Combinational Circuits

The following are the main characteristics of combinational circuits −

The output of a combinational circuit, at any instant of time, depends only on the present input values at that instant of time.

Combinational circuits do not use any kind of memory element in their circuits. Thus, the previous state of input and output values do not have any effect on the present operation of the circuit.

The output of a combinational circuit can be entirely predicted using its logical operation and input values.

Combinational circuits produce an instantaneous output in response to any change in its input values.

Types of Combinational Circuits

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

Binary Adders :

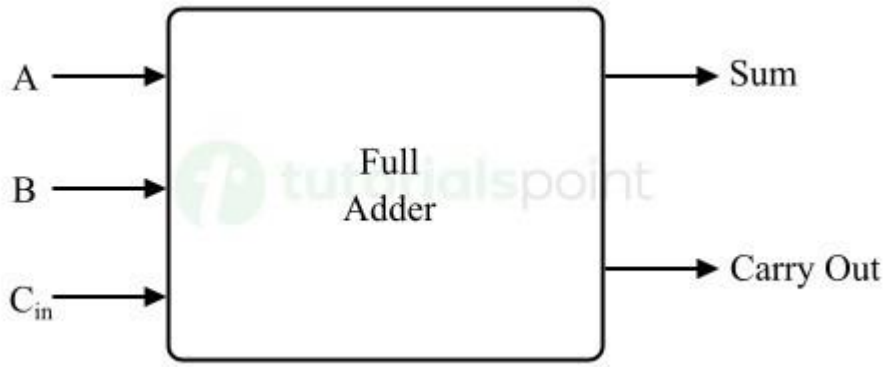
A binary adder is a combinational circuit that performs the addition of binary digits or bits. Depending on the design and configuration, there are two types of binary adders namely, Half Adder and Full Adder.

Half Adder

The half adder is a combinational logic circuit with two inputs and two outputs. The half adder circuit is designed to add two single-bit binary numbers A and B. This circuit has two outputs namely, sum and carry.

Full Adder

The full adder is designed to overcome the drawback of a half adder which is the ability to add only two bits. Therefore, the full adder is a three-input and two-output combinational circuit. Where, the inputs are two one-bit numbers A and B, and a carry C from the previous addition. The outputs are sum and carry output.



Binary Subtractors

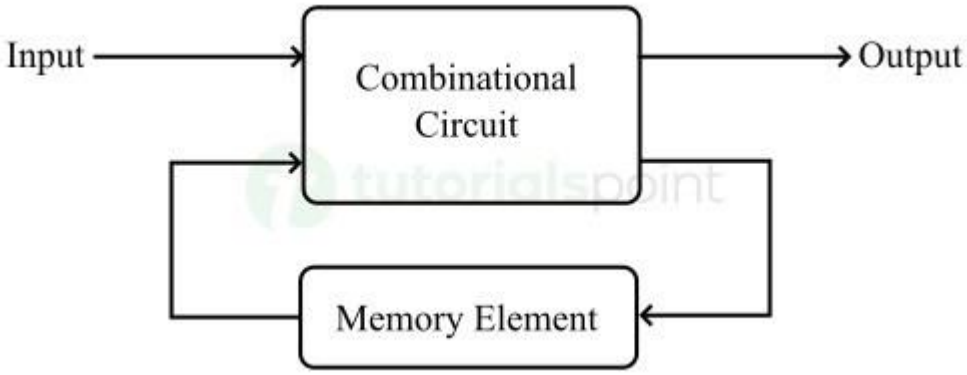
A binary subtractor is a combinational logic circuit used to subtract one binary number from another. Similar to binary adder, there are two types of binary subtractors namely, half-subtractor and full-subtractor.

Half Subtractor

A half subtractor is a combination circuit with two inputs (A and B) and two outputs (difference and borrow). It produces the difference between the two binary bits at the input and also produces an output (Borrow) to indicate if a 1 has been borrowed. In binary subtraction (A-B), A is called a Minuend bit and B is called a Subtrahend bit.

What is a Sequential Circuit?

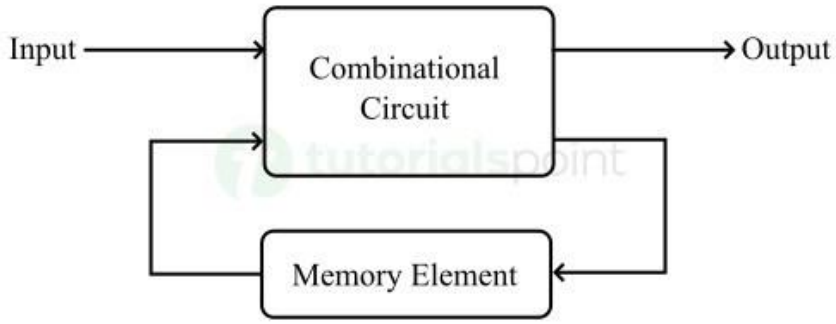
A sequential circuit is a logic circuit that consists of a memory element to store history of past operation of the circuit. Therefore, the output of a sequential circuit depends on present inputs as well as past outputs of the circuit.



Sequential Circuit

A sequential circuit is a type of digital logic circuit whose output depends on present inputs as well as past operation of the circuit.

The block diagram of a typical sequential circuit



A sequential circuit is basically a combination of a combinational circuit and a memory element. The combinational circuit performs the logical operations specified, while the memory element records the history of operation of the circuit. This history is then used to perform various logical operations in future.

Main Components of Sequential Circuit

A sequential circuit consists of several different digital components to process and hold information in the system. Some key components of a sequential circuit explained −

Logic Gates

The logic gates like AND, OR, NOT, etc. are used to implement the data processing mechanism of the sequential circuits. These logic gates are basically interconnected in a specific manner to implement combinational circuits to perform logical operations on input data.

Memory Element

In sequential circuits, the memory element is another crucial component that holds history of circuit operation. Generally, **flip-flops** are used as the memory element in sequential circuits.

In sequential circuits, a feedback path is provided between the output and the input that transfers information from output end to the memory element and from memory element to the input end.

All these components are interconnected together to design a sequential circuit that can perform complex operations and store state information in the memory element.

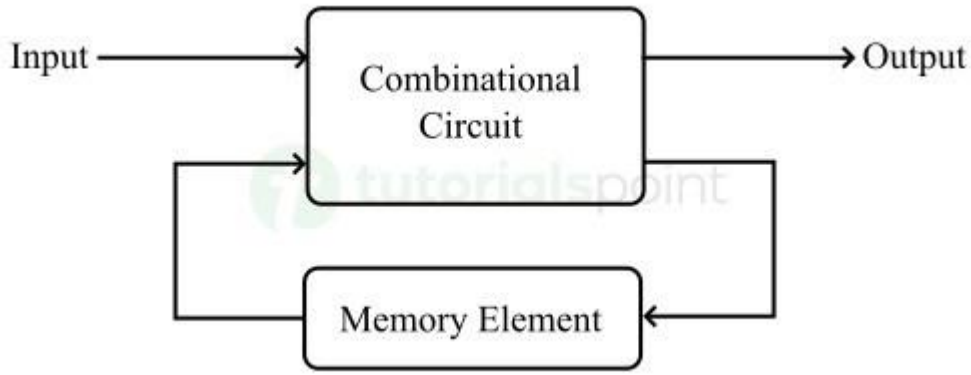
Types of Sequential Circuits

Based on structure, operation, and applications, the sequential circuits are classified into the following two types −

Asynchronous Sequential Circuit

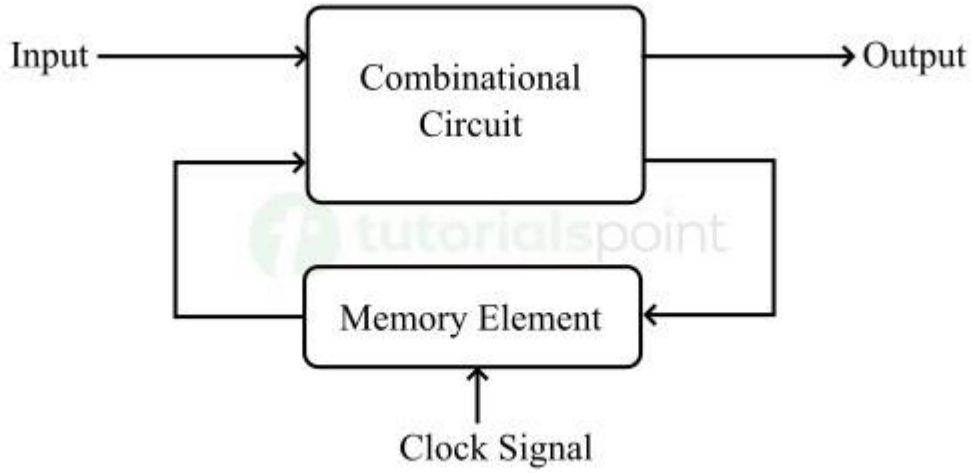
A type of sequential circuit whose operation does not depend on the clock signals is known as an asynchronous sequential circuit. This type of sequential circuits operates using the input pulses that means their state changes with the change in the input pulses.

The **ripple** counter is a common example of asynchronous sequential circuit.



Synchronous Sequential Circuit.

A synchronous sequential circuit is a type of sequential circuit in which all the memory elements are synchronized by a common clock signal. Hence, synchronous sequential circuits take a clock signal along with input signals.



Flip-Flops

A flip-flop is a sequential digital electronic circuit having two stable states that can be used to store one bit of binary data. Flip-flops are the fundamental building blocks of all memory devices.

Types of Flip-Flops

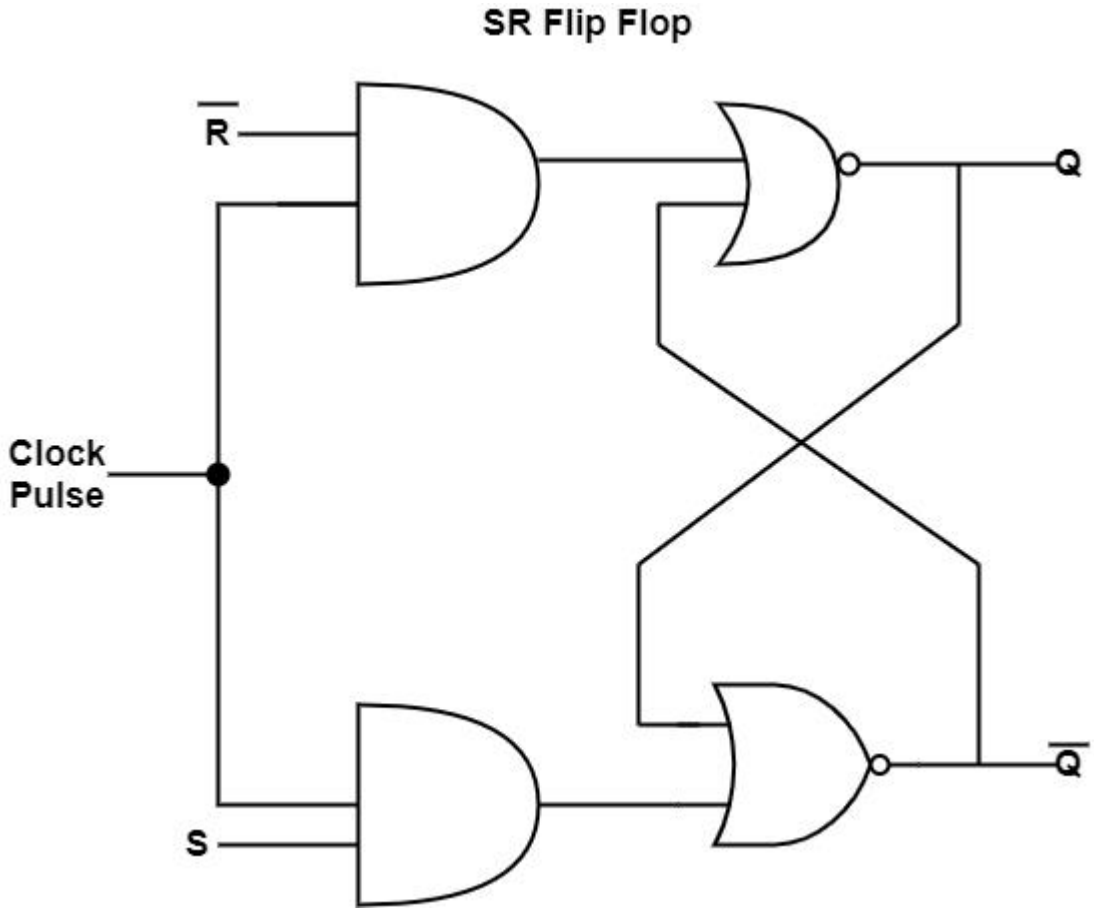
S-R Flip-Flop

J-K Flip-Flop
D Flip-Flop
T Flip-Flop

What is Set-Reset (SR) Flip-flop? SET-RESET Flip-flop

Flip flops are an application of logic gates. A flip-flop circuit can stay in a binary state continually (as long as power is transferred to the circuit) before conducted by an input signal to switch states. SET-RESET flip-flop includes two NOR gates and also two NAND gates. These flip-flops are also known as SR Latch.
(latch is a type of memory circuit that stores one bit of binary data (0 or 1) and can hold that state until new inputs change it)

The SR flip-flop has two inputs such as the 'Set' input and a 'Reset' input. The two outputs of SR flip-flop are the main output Q and its complement $\overline{Q}$.The diagram shows the circuit diagram of an SR flip-flop.Q is the current state of the out put .$Q_{N.1}$ represent the inputs (S and R)



Truth Table of SR Flip Flop

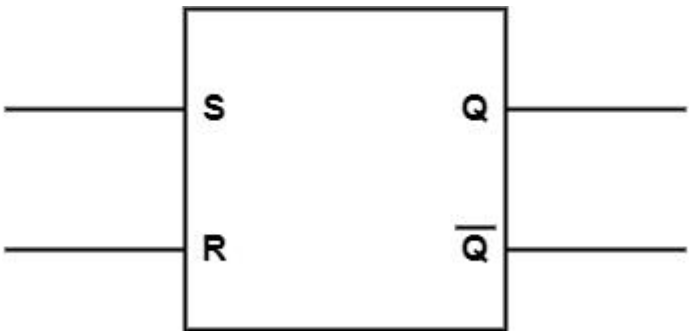
S	R	$Q_{N.1}$	$\overline{Q_{N \cdot 1}}$
0	0	Q_N	$\overline{Q_N}$
0	1	0	1
1	0	1	0
1	1	Indeterminate	Indeterminate

A team of cross-coupled NOR gates can describe an SR flip-flop, wherein, the output of one gate is related to one of the two inputs of the other gate and vice versa. The complementary input of one NOR gate is 'R' while the complementary input of the other gate is 'S'.

The input 'R' makes the output Q and the gate with the 'S' input makes the output $\overline{Q}$

.

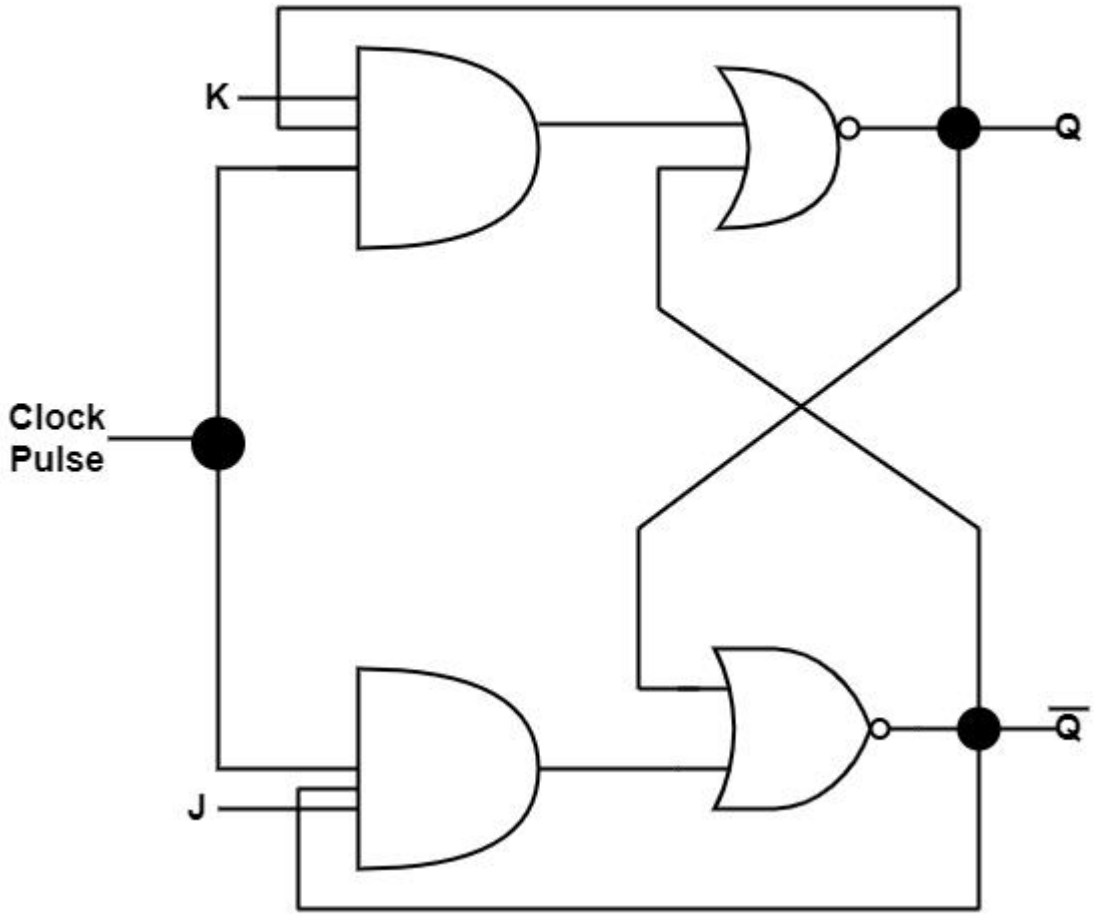
The logic symbol of the SR flip-flop is shown in the figure.



What is JK Flip Flop?

JK flip-flop can be treated as an alteration of the SR flip-flop. J represents SET, and 'K' represents CLEAR. In the JK flip-flop, the "S" input is known as the "J" input, and the "R" input is known as the "K" input. The output of the JK flip-flop does not modify if both "J" and "K" are "0". If both the inputs are "1", then the output dial to its free.

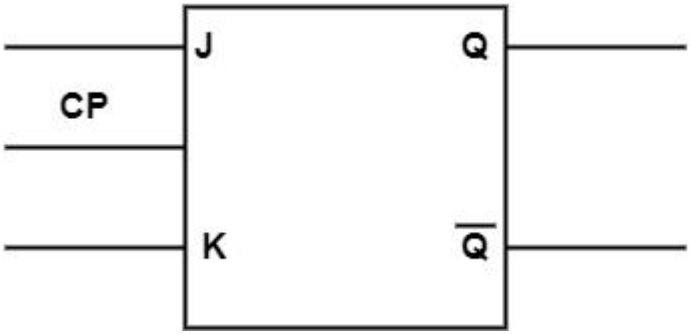
JK Flip Flop



Truth Table of JK Flip Flop

S	R	QN-1
0	0	Q_N
0	1	0
1	0	1
1	1	$\overline{Q_N}$

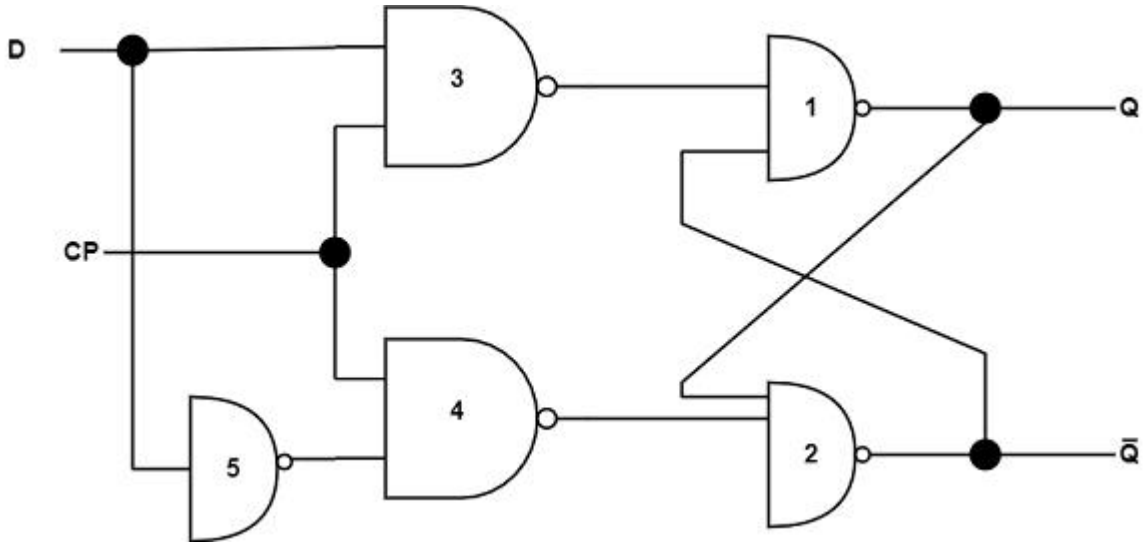
The logic symbol for the JK flip-flop



What is D Flip Flop?

The D flip-flop is a clocked flip-flop with a single digital input 'D'. Each time a D flip-flop is clocked, its output follows the state of 'D'. The D Flip Flop has only two inputs D and CP. The D inputs go precisely to the S input and its complement is used to the R input.

Considering the pulse input is at 0, the outputs of gates 3 and 4 are at the 1 level and the circuit cannot convert state regardless of the value of D. The D input is sampled when CP = 1. If D is 1, the Q output goes to 1, locating the circuit in the set state. If D is 0, output Q goes to 0, and the circuit switches to a clear state.

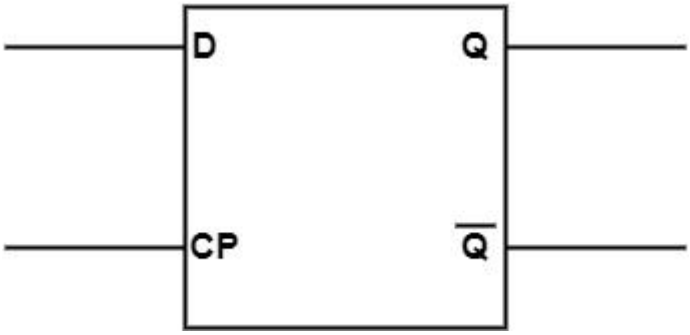


Truth Table of D Flip

S	D	Q_{N+1}
0	0	0

0	1	1
1	0	0
1	1	1

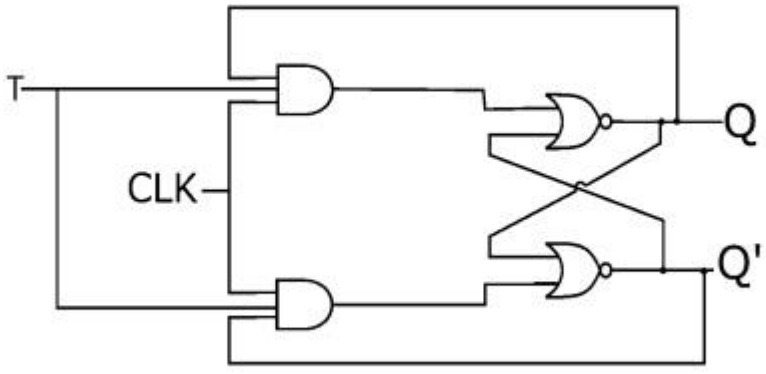
The logic symbol for the D flip-flop



What is T Flip Flop?

The T flip-flop is also called toggle flip-flop. It is a change of the JK flip-flop. The T flip flop is received by relating both inputs of a JK flip-flop. The T flip-flop is received by relating the inputs 'J' and 'K'. When T = 0, both AND gates are disabled. Therefore, there is no change in the output. When T= 1, the output toggles.

The diagram demonstrates the circuit diagram of a T flip-flop.



Truth Table of T Flip-Flop

Q	T	Q

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

Applications of Flip-Flops

Counters

Shift Registers

Storage Registers, etc.

What is a Synchronous Counter?

If the clock pulses are applied to all the flip-flops in a counter simultaneously, then such a counter is called as synchronous counter.

Basically, all the flip-flops in a synchronous counter are arranged in a cascade connection and each flip-flop is individually connected to an external clock. It allows the clocking of all the flip-flops at the same time instant with the same clock input. It means the output of each flip-flop varies in synchronization with the clock input.

Due to this, the common clock signal causes the change in the state of each individual flip-flop simultaneously. Resultantly it leads to no ripple effect, thus there is no propagation delay in a synchronous counter.

Logic gates are used in synchronous counters to control the count sequence.

What is an Asynchronous Counter?

Asynchronous counters are also known as serial counters because the flip-flops that constitute the counter are connected serially and the input clock pulse is provided to the first flip-flop in the connection.

The output of the first flip-flop acts as the input of the next adjacent flip-flop in the forward direction. In this manner, the clock input ripples through the counter. Hence, these counters are also known as ripple counters.

Due to the ripple effect, the timing signal in an asynchronous counter gets delayed by some amount on passing through each flip flop. Hence, it results in a propagation delay.

Difference Between Synchronous and Asynchronous Counters

Key	Synchronous Counter	Asynchronous Counter
Trigger	In case of Synchronous Counters, all the constituent flip-flops are triggered with same clock simultaneously.	In case of Asynchronous Counters, there is triggering of different flip-flops with different clock.
Operation Speed	Operation speed of a synchronous counter is faster as compared to that of an asynchronous counter.	The operation speed of an asynchronous counter is comparatively slower than a synchronous counter.
Error Prone	Synchronous Counters are less error-prone; they hardly produce any decoding errors because each flip-flop is individually clocked.	Asynchronous Counters are more error-prone and produce decoding errors in the system.
Complexity	All the flip-flops in a synchronous counter coordinate with the clock, hence its design and implementation is complex as compared to that of an asynchronous counter.	In an asynchronous counter, the output of one flip-flop acts as the input of the next flip-flop, hence its design and implementation is quite simple.
Sequence	A Synchronous counter can be operated in any desired count sequence, as it could get manipulated by changing the clock sequence.	An Asynchronous counter can operate only in a fixed count sequence, i.e., UP and DOWN.
Delay	There is no propagation delay observed in case of Synchronous Counters.	In case of asynchronous counters, there is a subsequent propagation delay from one flip-flop to another.

The instruction cycle

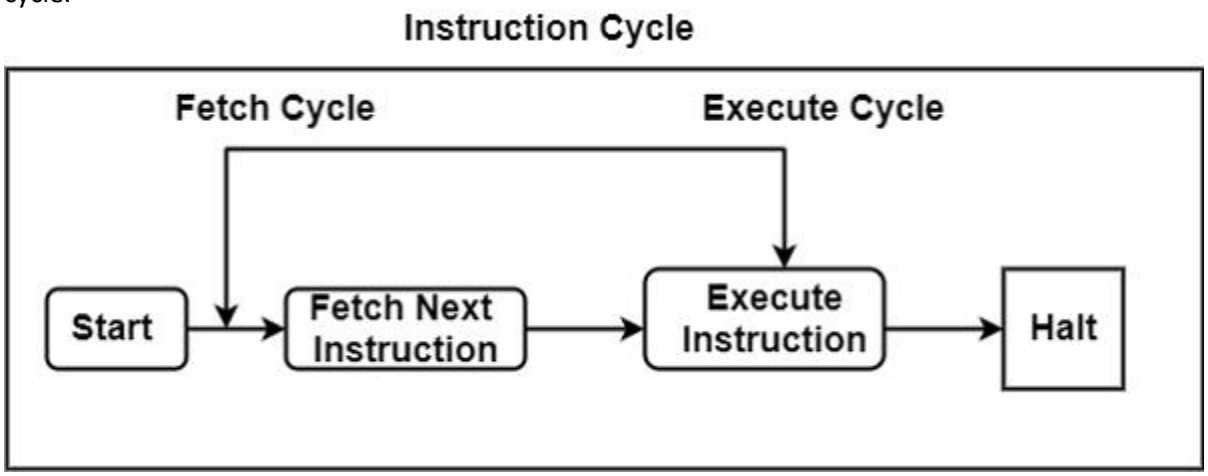
The instruction cycle, also known as the fetch-decode-execute cycle, is the fundamental sequence of operations that a computer's CPU performs to execute a single instruction. This cycle repeats continuously from the moment the computer is powered on until it's turned off.

A program consisting of a series of instructions. The program is implemented on the computer by going through a cycle for each instruction.

In the basic computer, each instruction cycle includes the following procedures −

- It can fetch instruction from memory.
- It is used to decode the instruction.
- It can execute the instruction.

After the following three procedures are done, the control switches back to the first step and repeats the similar process for the next instruction. The figure shows the phases contained in the instruction cycle.



Fetch Cycle

The address instruction to be implemented is held at the program counter. The processor fetches the instruction from the memory that is pointed by the PC.

Next, the PC is incremented to display the address of the next instruction. This instruction is loaded onto the instruction register. The processor reads the instruction and executes the important procedures.

Execute Cycle

The data transfer for implementation takes place in two methods are as follows −

- **Processor-memory** − The data sent from the processor to memory or from memory to processor.
- **Processor-Input/Output** − The data can be transferred to or from a peripheral device by the transfer between a processor and an I/O device.

In the execute cycle, the processor implements the important operations on the information, and consistently the control calls for the modification in the sequence of data implementation. These two methods associate and complete the execute cycle.

What are the technicalities involved in the instruction cycle?

The primary function of a CPU is to execute and run a program. A set of instructions is to be processed to run such a program. Every given program undergoes the instruction cycle stages until the whole program is finally executed.

This cycle begins as soon as the system is switched on and ends when it is shut down. Therefore, every single task on a computer undergoes this cycle, and the process is repeated until the system is shut down.

What are the registers used in cycles?
Here's an example of an instruction cycle in a series of steps.

Memory address registers (MAR):

- The record specifies the memory address (MAR) to carry out a read or write operation
- Links to the system bus's address lines

Memory Buffer Register (MBR):

- To fetch the address of the next instruction and then store it in the program counter (PC)
- Links to the system bus's data lines

- It contains either the value to be held in memory or the most recent value ready from memory
- The instruction register (IR):
- Fetches the most recent instruction and stores the same.
- What are the steps involved in the instruction cycle?
- The instruction cycle comprises three main stages and is also addressed as the fetch-decode-execute cycle or fetch-execute cycle because of the steps involved. The stages are as follows:

- **Fetch stage**
- **Decode stage**
- **Execute stage**
- What are the procedures which are included in the instruction cycle?

- The following procedures are included in each instruction cycle:
- The CPU reads the effective address from memory if the instruction has an indirect address
- The register retrieves instructions from memory.
- It is capable of carrying out the command
- Decoding education is the primary task assigned
- The loop continues carrying out the task repeatedly until a halt condition is met. Till then, the cycle keeps on restarting.

UNIT-5

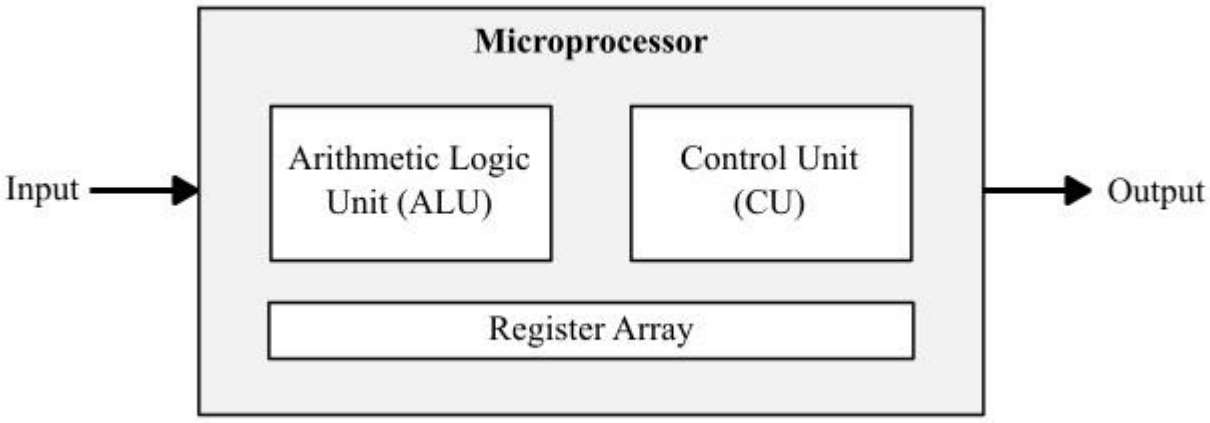
What is a Microprocessor?

A microprocessor is a semiconductor electronic device designed to perform data processing and digital operations in a computing machine like a computer. It is also termed as a processor or central processing unit or CPU. It is generally built in the form of a single IC (integrated circuit). The first commercially available microprocessor was developed by Intel Corporation in the year of 1971,

The primary function of a microprocessor is Inputting , Processing, Outputting

Block Diagram of Microprocessor

The block diagram of a typical microprocessor is shown in the following



It consists of three main parts which are described as follows –

Arithmetic Logic Unit (ALU) − It is an electronic circuit that performs arithmetic and logical operations on data received from an input device or memory.

Control Unit (CU) − This electronic circuit of the microprocessor is responsible controlling the flow of data and instructions within the device or the system.

Register Array − Register array is nothing but a collection of digital registers to provide small and fast storage to temporarily hold data and instructions in the microprocessor during processes.

Arithmetic Logic Unit (ALU)

ALU is the most important component of a microprocessor, as it is responsible performing all computing tasks and operations. As its name implies, ALU can perform the following two main operations namely,

Arithmetic Operations − It performs mathematical operations like addition, subtraction, multiplication, division, etc. of numbers.

Logical Operations − It also performs logical functions that involve comparison or decision-making processes.

Arithmetic logic unit uses instructions and input data in binary format to produce desired results.

The structure of the arithmetic logic unit is divided into the following two segments −

Arithmetic Circuit − This segment of ALU consists of binary adders and subtractors and handles all arithmetic operations like addition, subtraction, multiplication, and division.

Logic Circuit − This part of ALU comprises of digital circuits like AND gate, OR gate, NOT gate, etc. and performs logical operations such as comparison, shifting, and decision-making.

Arithmetic logic unit, being a part of a microprocessor, provides the following functionalities It allows for executing instructions.

It provides capabilities to manipulate input data and generate desired outputs.

It ensures the speed, efficiency, and overall performance of the microprocessor.

Control Unit (CU)

Control Unit (CU) is another key component of a microprocessor, as it allows for managing and coordinating the operations of all other components of the system. It is responsible for ensuring that every part of the microprocessor works smoothly and reliably.

It is nothing but a digital circuit that directs the flow and execution of instructions and data within a microprocessor.

The structure of a typical control unit of a microprocessor comprises of the following key components −

Instruction Register − It temporarily stores the instructions being executed.

Instruction Decoder − It decodes the binary instructions and breaks them into opcode and operands to identify the operations.

Control and Timing Circuit − It generates control signals to synchronize all microprocessor operations and guide them to execute in correct sequence. It also sends control signals to various microprocessor components to enable and disable them as needed.

In a microprocessor, the control unit typically performs the following key functions −

--It allows ALU to fetch instructions from memory and load them into instruction register.

--It decodes instructions to identify the operations to be performed.

--It produces control signals for synchronization and coordination among various components of the microprocessor.

--It controls the flow of data and execution of instructions in the correct sequence.

Registers

Registers in a microprocessor are another important component. They are high-speed, small-sized storage devices, provided in a microprocessor to hold data and instruction temporarily. The following are some common types of registers used in microprocessors −

General Purpose Registers − These registers temporarily hold data to be processed and intermediate results during calculations.

Special Purpose Registers − These registers are provided to perform a specific task like to store address of next instruction that has to be executed.

Instruction Register − These registers are provided to temporarily store current instructions.

Base and Index Registers − These registers are used for memory addressing purposes.

Flag Registers − These registers store flags that denote the output of operations in arithmetic logic unit.

Memory Address Registers − These registers store the address of memory locations.

Data Registers − These registers are provided to temporarily store data that are being transferred between memory and the ALU.

How does a Microprocessor Work?

Fetch − It is the very first function that a microprocessor performs. In this step, the microprocessor access data and instructions from memory unit or an input device.

Decode − After receiving data and instructions, the microprocessor decodes them and interprets for computing process.

Execute − In this step, the microprocessor performs the requested operations on the data.

Store − Finally, the results produced by the operations are stored in the memory unit.

Types of Microprocessors

CISC (Complex Instruction Set Computer) Microprocessor

Examples of CISC microprocessors include Intel 386, 486, Pentium, Pentium II, Pentium Pro, etc.

RISC (Reduced Instruction Set Computer) Microprocessor examples of RISC microprocessors are Alpha 21164, IBM RS6000, Alpha 21064, etc.

EPIC (Explicitly Parallel Instruction Computing) Microprocessor

Applications of Microprocessors

Microprocessors are used in a wide range of common electronic and computing devices like laptops, desktops, smart watches, smart TVs, etc.

Microprocessors are also used in microcontrollers .

Microprocessors specially designed for digital signal processing are used in applications like telecommunication, audio processing, image processing, etc.

Microprocessors are also used in robotic or autonomous devices like surveillance drones, autonomous aircrafts, etc.

The specialized microprocessors named Application Specific Integrated Circuits (ASICs) microprocessors are designed for specific tasks and customization depending on the application requirements.

GPU (Graphical Processing Unit) microprocessors are designed and used for performing high performance graphics functions.

Features of a Microprocessor

Here is a list of some of the most prominent features of any microprocessor −

Cost-effective − The microprocessor chips are available at low prices and results its low cost.

Size − The microprocessor is of small size chip, hence is portable.

Low Power Consumption − Microprocessors are manufactured by using metaloxide semiconductor technology, which has low power consumption.

Versatility − The microprocessors are versatile as we can use the same chip in a number of applications by configuring the software program.

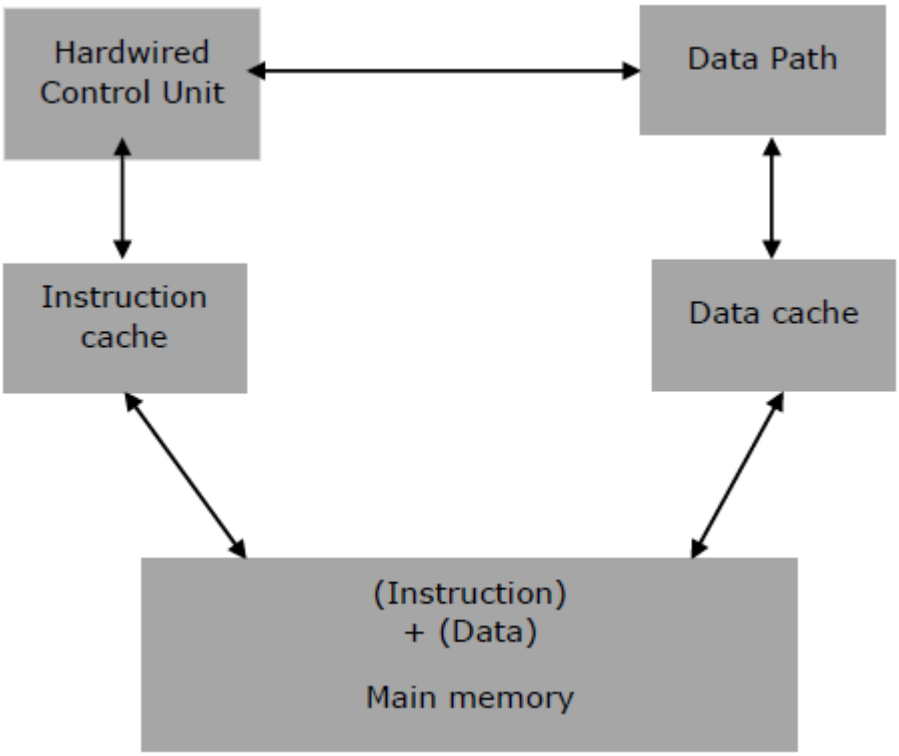
Reliability − The failure rate of an IC in microprocessors is very low, hence it is reliable.

Instruction Set Architecture (ISA)

ISA is a design architecture of microprocessors. It defines the set of instructions that the microprocessor can execute. There are two different types of ISAs commonly used in microprocessors namely, RISC (Reduced Instruction Set Computing) and CISC (Complex Instruction Set Computing).

Characteristics of RISC

RISC microprocessor architecture uses highly-optimized set of instructions. It is used in portable devices like Apple iPod due to its power efficiency.



The major characteristics of a RISC processor are as follows

It consists of simple instructions.

It supports various data-type formats.

It utilizes simple addressing modes and fixed length instructions for pipelining.

It supports register to use in any context.

One cycle execution time.

LOAD and STORE instructions are used to access the memory location.

It consists of larger number of registers.

It consists of less number of transistors.

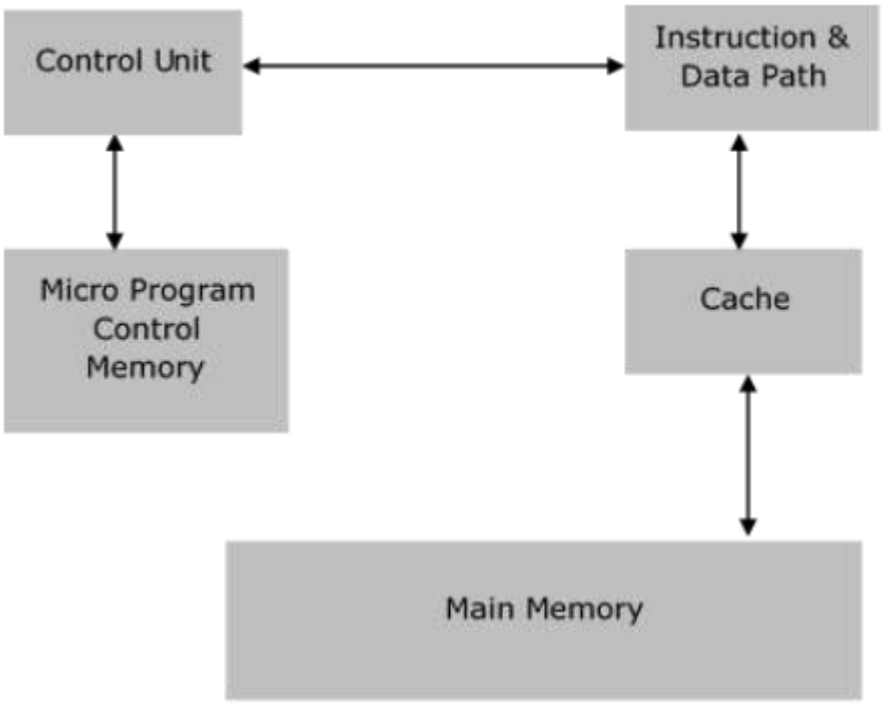
CISC Processor

CISC stands for Complex Instruction Set Computer.SC Processor .It is designed to minimize the number of instructions per program, ignoring the number of cycles per instruction. The emphasis is on building complex instructions directly into the hardware.

Some of the CISC Processors are

IBM 370/168	VAX 11/780	Intel 80486e

Architecture of CISC



Characteristics of CISC

Variety of addressing modes.
Larger number of instructions.
Variable length of instruction formats.
Several cycles may be required to execute one instruction.
Instruction-decoding logic is complex.
One instruction is required to support multiple addressing modes.

Special Processors

These are the processors which are designed for some special purposes. Few of the special processors are briefly discussed −

Coprocessor

A coprocessor is a specially designed microprocessor, which can handle its particular function many times faster than the ordinary microprocessor.
For example − Math Coprocessor.
Some Intel math-coprocessors are –
8087-used with 8086
80287-used with 80286
80387-used with 80386

Components of Microprocessor

A microprocessor consists of many components to perform various functions. The following are some key components of a typical microprocessor −
Arithmetic Logic Unit (ALU)
Control Unit (CU)
Registers

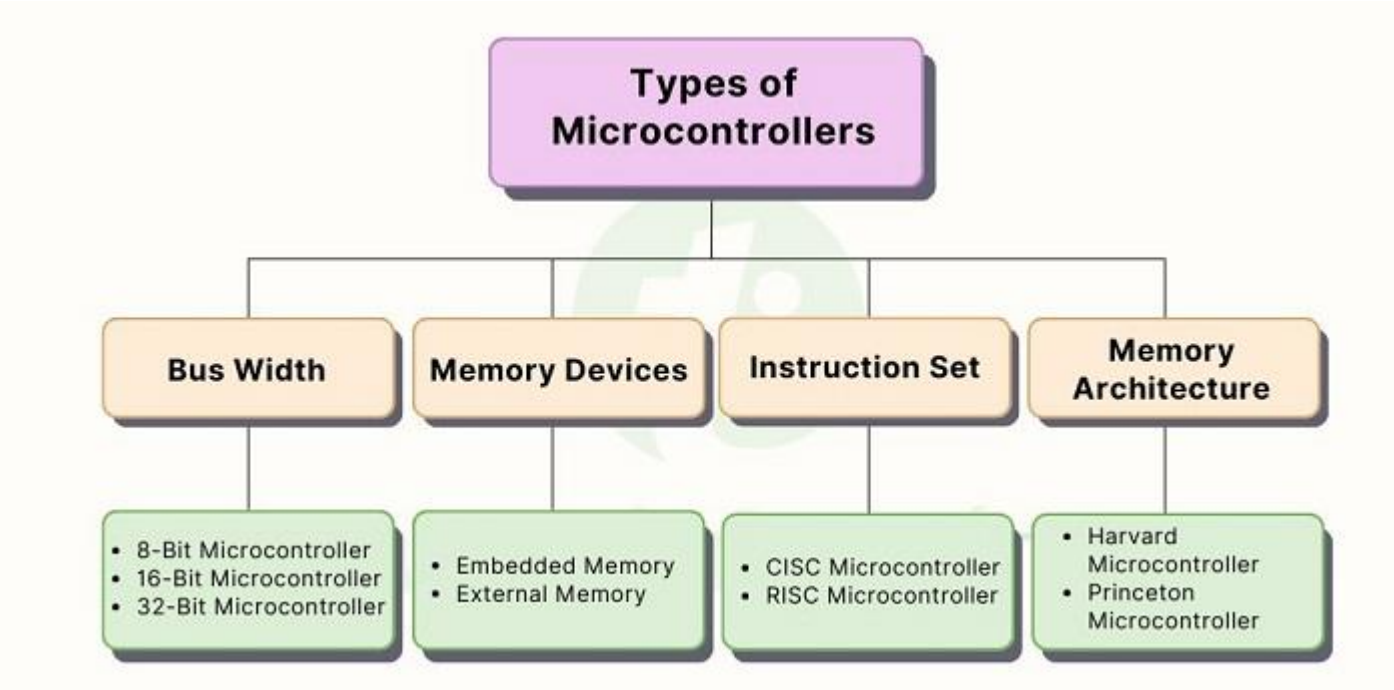
Cache Memory
Clock Generator
System Bus
These six elements are the main components of a microprocessor

Microcontrollers

Microcontrollers are important and widely used components in most electronic systems and devices, ranging from washing machines to robotic systems and industrial control systems. Microcontroller act as the brain behind automated working of all these devices

Types of Microcontrollers

We can categorize microcontrollers into various classes based on the following important parameters –
Number of Bits
Memory Devices
Instruction Set
Memory Architecture
Let's have a detailed discussion on each category microcontrollers classification.



Advantages of Microcontrollers

1. Low Time Requirement for Operations

2. Easy to Use and Maintain