Manonmaniam Sundaranar University,
Directorate of Distance & Continuing Education,
Tirunelveli - 627 012 Tamilnadu, India

OPEN AND DISTANCE LEARNING(ODL) PROGRAMMES
(FOR THOSE WHO JOINED THE PROGRAMMES FROM THE ACADEMIC YEAR 2023–2024)

III YEAR

B.Sc. Physics

Course Material

Digital Electronics And Microprocessor 8085

Prepared

By

Dr. S. Shailajha
Dr. P. Hema
Assistant Professor
Department of Physics
Manonmaniam Sundaranar University
Tirunelveli – 12

DIGITAL ELECTRONICS AND MICROPROCESSOR 8085

UNIT-I: DECIMAL, BINARY, OCTAL, HEXA DECIMAL NUMBERS SYSTEMS AND THEIR CONVERSIONS – CODES: BCD, Gray and excess-3 codes – code conversions – binary addition, binary subtraction using 1's & 2's complement methods – Boolean laws – De-Morgan's theorem – basic logic gates – universal logic gates (NAND & NOR) – standard representation of logic functions (SOP & POS) – minimization techniques (Karnaugh map: 2, 3, 4 variables).

UNIT-II: ADDERS: half & full adder – subtractors: half & full subtractor – parallel binary adder – magnitude comparator – multiplexers (4:1) & demultiplexers (1:4), encoder (8-line to 3-line) and decoder (3-line to 8-line), BCD to seven segment decoders.

UNIT-III: FLIP-FLOPS: R-S Flip-flop, J-K Flip-Flop, T and D type flip-flops, master-slave flip-flop, truth tables, registers: serial in serial out and parallel in and parallel out – counters asynchronous: mod-8, mod-10, synchronous – ring counter and up-down counter – A/D and D/A converter.

UNIT-IV: GENERAL MEMORY OPERATIONS: ROM, RAM (static and dynamic), PROM, EPROM, EEPROM, EAROM. IC–logic families: RTL, DTL, TTL logic, CMOS NAND & NOR Gates, CMOS Inverter. Programmable Logic Devices – Programmable Logic Array (PLA), Programmable Array Logic (PAL).

UNIT-V:8085 MICROPROCESSOR: Introduction to microprocessor – pin configuration of 8085 – Flags – Registers (General and special purpose) – interrupts and its priority – instruction set of 8085 – addressing modes of 8085 – Assembly language programming using 8085 – programs for addition, subtraction, multiplication and division (8-Bit only).

TEXTBOOKS

1. M. Morris Mano - Digital Design-3rd Edition, PHI, New Delhi.
2. Ronald J. Tocci - Digital Systems-Principles and Applications 6/e, PHI, New Delhi. 1999. (UNITS 1 to IV)
3. S. Salivahana & S. Arivazhagan-Digital circuits and design
4. Microprocessor Architecture, Programming and Applications with the 8085-Penram International Publishing, Mumbai-Ramesh S. Gaonkar
5. Microcomputer Systems the 8086/8088 family-YU-Cheng Liu and Glenn A.

UNIT-I:

DECIMAL, BINARY, OCTAL, HEXA DECIMAL NUMBERS SYSTEMS AND THEIR CONVERSIONS – CODES: BCD, Gray and excess-3 codes – code conversions – binary addition, binary subtraction using 1's & 2's complement methods – Boolean laws – De-Morgan's theorem – basic logic gates – universal logic gates (NAND & NOR) – standard representation of logic functions (SOP & POS) – minimization techniques (Karnaugh map: 2, 3, 4 variables).

1.1. Number Systems:

A digital number system is a positional number system that has some symbols called digits. It provides a complete set of digits, operators, and rules to perform operations. In a digital number system, the number of digits used determines the base of the number system. For example, the binary number system has two digits (0 and 1), hence, the base of the binary number system is 2. Digital number systems form the foundation of the modern computing technologies and digital electronics. They are used to represent, process, and manipulate the information using a digital system.

Types of Digital Number Systems:

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 is the fundamental building block behind the implementation and working of all digital systems. Binary number system has two symbols or digits, i.e., 0 and 1.

Hence, these two digits are used to represent information and perform all the digital operations. Each binary digit is called a bit. 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 2. Binary digits are used in digital system to represent their ON and OFF states. Where, 0 is used to represent the OFF state of the digital system and 1 is used to represent the ON state of the system. Overall, the binary number system forms the foundation of computation, digital communication, and digital information storage.

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, 2^3 respectively. 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,

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

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

Decimal Number System:

Decimal number system is not inherently a digital number system. But it is widely used to represent the digital information in a human readable format. Decimal number system is a base 10 number system having 10 unique digits i.e., 0, 1, 2, 3, 4, 5, 6, 7, 8, and 9. It is the standard number system used by human beings to represent information in a natural way. However, a digital system cannot directly process the information represented in decimal form, so it is converted into binary form and then processed. The base of the decimal number system is 10. So, the value of a decimal number is calculated by the sum of powers of 10.

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,

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

After simplifying the right-hand side terms, we will get the decimal number, which is on the left-hand side.

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 $2^3 = 8$. 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. Octal number system is used in the field of digital electronics to represent binary information in compact form, permissions in Linux or Unix systems, IPv6 address, binary machine code instructions, in error detection algorithms, etc.

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,

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

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

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 $2^4 = 16$. Thus, the value of a hexadecimal number can be calculated by the sum of powers of 16. In the field of digital electronics, the hexadecimal number system is used in memory address representation, digital colors representation, low level computer programming, encoding, assembly language programming, microcontrollers, keyboards, etc. Hexadecimal number system creates a balance between digital representation and human readability.

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. 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 Coded Decimal (BCD):

The binary coded decimal (BCD) is a type of binary code used to represent a given decimal number in an equivalent binary form. BCD-to-decimal and decimal-to-BCD conversions are very easy and straightforward. The BCD equivalent of a decimal number is written by replacing each decimal digit in the integer and fractional parts with its four-bit binary equivalent. As an example, the BCD equivalent of $(23.15)_{10}$ is written as $(0010\ 0011.0001\ 0101)_{BCD}$. The BCD code is more precisely known as the 8421 BCD code, with 8, 4, 2 and 1 representing the weights of different bits in the four-bit groups, starting from MSB and proceeding towards LSB. This feature makes it a weighted code, which means that each bit in the four-bit group representing a given decimal digit has an assigned weight.

Table 1.1 BCD codes

Decimal	8421BCD code	4221 BCD code	5421 BCD code
0	0000	0000	0000
1	0001	0001	0001
2	0010	0010	0010
3	0011	0011	0011
4	0100	1000	0100
5	0101	0111	1000
6	0110	1100	1001
7	0111	1101	1010
8	1000	1110	1011
9	1001	1111	1100

Other weighted BCD codes include the 4221 BCD and 5421 BCD codes. Again, 4, 2, 2 and 1 in the 4221 BCD code and 5, 4, 2 and 1 in the 5421 BCD code represent weights of the relevant bits. Table 1.1 shows a comparison of 8421, 4221 and 5421 BCD codes. As an example, $(98.16)_{10}$ will be written as 1111 1110.0001 1100 in 4221 BCD code and 1100 1011.0001 1001 in 5421 BCD code. Since the 8421 code is the most popular of all the BCD codes, it is simply referred to as the BCD code.

Gray Code:

The Gray code was designed by Frank Gray at Bell Labs and patented in 1953. It is an un weighted binary code in which two successive values differ only by 1 bit. Owing to this feature, the maximum error that can creep into a system using the binary gray code to encode data is much less than the worst-case error encountered in the case of straight binary encoding. Table 1.2 lists the binary and gray code equivalents of decimal numbers 0–15. An examination of the four-bit gray code numbers, as listed in Table 1.2, shows that the last entry rolls over to the first entry. That is, the last and the first entry also differ by only 1 bit. This is known as the cyclic property of the gray code. Although there can be more than one gray code for a given word length, the term was first applied to a specific binary code for non-negative integers and called the binary-reflected gray code or simply the gray code.

Table 1.2 Gray code

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

Excess-3 codes:

The excess-3 code is another important BCD code. It is particularly significant for arithmetic operations as it overcomes the shortcomings encountered while using the 8421 BCD code to add two decimal digits whose sum exceeds 9. The excess-3 code has no such limitation, and it considerably simplifies arithmetic operations.

Table 1.3 Excess-3 code equivalent of decimal numbers.

Decimal number	Excess-3 code	Decimal number	Excess-3 code
0	0011	5	1000
1	0100	6	1001
2	0101	7	1010
3	0110	8	1011
4	0111	9	1100

Table 1.3 lists the excess-3 code for the decimal numbers 0–9. The excess-3 code for a given decimal number is determined by adding ‘3’ to each decimal digit in the given number and then replacing each digit of the newly found decimal number by its four-bit binary equivalent. It may be mentioned here that, if the addition of ‘3’ to a digit produces a carry, as is the case with the digits 7, 8 and 9, that carry should not be taken forward. The result of addition should be taken as a single entity and subsequently replaced with its excess-3 code equivalent. As an example, let us find the excess-3 code for the decimal number 597:

• The addition of ‘3’ to each digit yields the three new digits/numbers ‘8’, ‘12’ and ‘10’.

• The corresponding four-bit binary equivalents are 1000, 1100 and 1010 respectively.

• The excess-3 code for 597 is therefore given by: 1000 1100 1010=100011001010.

Also, it is normal practice to represent a given decimal digit or number using the maximum number of digits that the digital system is capable of handling. For example, in four-digit decimal arithmetic, 5 and 37 would be written as 0005 and 0037 respectively. The corresponding 8421 BCD equivalents would be 0000000000000101 and 0000000000110111 and the excess-3 code equivalents would be 0011001100111000 and 0011001101101010.

Corresponding to a given excess-3 code, the equivalent decimal number can be determined by first splitting the number into four-bit groups, starting from the radix point, and then subtracting 0011 from each four-bit group. The new number is the 8421 BCD equivalent of the given excess-3 code, which can subsequently be converted into the equivalent decimal number.

As an example, following these steps, the decimal equivalent of excess-3 number 01010110.10001010 would be 23.57.

Another significant feature that makes this code attractive for performing arithmetic operations is that the complement of the excess-3 code of a given decimal number yields the excess-3 code for 9's complement of the decimal number. As adding 9's complement of a decimal number B to a decimal number A achieves A – B, the excess-3 code can be used effectively for both addition and subtraction of decimal numbers.

1.2. Code conversions:

1. BCD-to-Binary Conversion:

A given BCD number can be converted into an equivalent binary number by first writing its decimal equivalent and then converting it into its binary equivalent. The first step is straightforward, and the second step was explained in the previous chapter. As an example, we will find the binary equivalent of the BCD number 0010 1001.0111 0101:

• BCD number: 0010 1001.0111 0101.

• Corresponding decimal number: 29.75.

• The binary equivalent of 29.75 can be determined to be 11101 for the integer part and .11 for the fractional part.

• Therefore, (0010 1001.0111 0101) BCD = (11101.11)2.

2. Binary-to-BCD Conversion

The process of binary-to-BCD conversion is the same as the process of BCD-to-binary conversion executed in reverse order. A given binary number can be converted into an equivalent BCD number by first determining its decimal equivalent and then writing the corresponding BCD equivalent. As an example, we will find the BCD equivalent of the binary number 10101011.101:

• The decimal equivalent of this binary number can be determined to be 171.625.

• The BCD equivalent can then be written as 0001 0111 0001.0110 0010 0101.

3. Binary–Gray Code Conversion:

A given binary number can be converted into its Gray code equivalent by going through the following:

steps:

1. Begin with the most significant bit (MSB) of the binary number. The MSB of the Gray code equivalent is the same as the MSB of the given binary number.
2. The second most significant bit, adjacent to the MSB, in the Gray code number is obtained by adding the MSB and the second MSB of the binary number and ignoring the carry, if any. That is, if the MSB and the bit adjacent to it are both '1', then the corresponding Gray code bit would be a '0'.
3. The third most significant bit, adjacent to the second MSB, in the Gray code number is obtained by adding the second MSB and the third MSB in the binary number and ignoring the carry, if any.
4. The process continues until we obtain the LSB of the Gray code number by the addition of the LSB and the next higher adjacent bit of the binary number. The conversion process is further illustrated with the help of an example showing step-by-step conversion of $(1011)_2$ into its Gray code equivalent:

Binary 1011

Gray code 1- - -

Binary 1011

Gray code 11- -

Binary 1011

Gray code 111-

Binary 1011

Gray code 1110

4. Gray Code–Binary Conversion:

A given Gray code number can be converted into its binary equivalent by going through the following steps:

1. Begin with the most significant bit (MSB). The MSB of the binary number is the same as the MSB of the Gray code number.
2. The bit next to the MSB (the second MSB) in the binary number is obtained by adding the MSB in the binary number to the second MSB in the Gray code number and disregarding the carry, if any.
3. The third MSB in the binary number is obtained by adding the second MSB in the binary number to the third MSB in the Gray code number. Again, carry, if any, is to be ignored.
4. The process continues until we obtain the LSB of the binary number.

The conversion process is further illustrated with the help of an example showing step-by-step conversion of the Gray code number 1110 into its binary equivalent:

Gray code 1110

Binary 1- - -

Gray code 1110

Binary 10 - -

Gray code 1110

Binary 101

Gray code 1110

Binary 1011

1.3. Binary addition, Binary subtraction using 1’s & 2’s complement methods:

Basic Rules of Binary Addition and Subtraction:

The basic principles of binary addition and subtraction are similar to what we all know so well in the case of the decimal number system. In the case of addition, adding ‘0’ to a certain digit produces the same digit as the sum, and, when we add ‘1’ to a certain digit or number in the decimal number system, the result is the next higher digit or number, as the case may be. For example, 6 + 1 in decimal equals ‘7’ because ‘7’ immediately follows ‘6’ in the decimal number system. Also, 7 + 1 in octal equals ‘10’ as, in the octal number system, the next adjacent higher number after ‘7’ is ‘10’. Similarly, 9 + 1 in the hexadecimal number system is ‘A’. With this background, we can write the basic rules of binary addition as follows:

1. 0 + 0 = 0.
2. 0 + 1 = 1.
3. 1 + 0 = 1.
4. 1 + 1 = 0 with a carry of ‘1’ to the next more significant bit.
5. 1 + 1 + 1 = 1 with a carry of ‘1’ to the next more significant bit.

Table 1.4 summarizes the sum and carry outputs of all possible three-bit combinations. We have taken three-bit combinations as, in all practical situations involving the addition of two larger bit

Table 1.4 Binary addition of three bits

A	**B**	**Carry-in(C_{in})**	**Sum**	**Carry-out C_o)**	**A**	**B**	**Carry-in(C_{in})**	**Sum**	**Carry-out(C_o)**
0	0	0	0	0	1	0	0	1	0
0	0	1	1	0	1	0	1	0	1
0	1	0	1	0	1	1	0	0	1
0	1	1	0	1	1	1	1	1	1

numbers, we need to add three bits at a time. Two of the three bits are the bits that are part of the two binary numbers to be added, and the third bit is the carry-in from the next less significant bit column. The basic principles of binary subtraction include the following:

1. 0 − 0 = 0.
2. 1 − 0 = 1.
3. 1 − 1 = 0.
4. 0 − 1 = 1 with a borrow of 1 from the next more significant bit.

The above-mentioned rules can also be explained by recalling rules for subtracting decimal numbers. Subtracting ‘0’ from any digit or number leaves the digit or number unchanged. This explains the first two rules. Subtracting ‘1’ from any digit or number in decimal produces the immediately preceding digit or number as the answer. In general, the subtraction operation of larger-bit binary numbers also involves three bits, including the two bits involved in the subtraction, called the minuend (the upper bit) and the subtrahend (the lower bit), and the borrow-in. The subtraction operation produces the difference output and borrow-out, if any. Table 1.5 summarizes the binary subtraction operation. The entries in Table 1.5 can be explained by recalling the basic rules of binary subtraction mentioned above, and that the subtraction operation involving three bits, that is, the minuend (A), the subtrahend (B) and the borrow-in (B_{in}), produces a difference output equal to ($A - B - B_{in}$). It may be mentioned here that, in the case of subtraction of larger-bit binary numbers, the least significant bit column always involves two bits to produce a difference output bit and the borrow-out bit. The borrow-out bit produced here becomes the borrow-in bit for the next more significant bit column, and the process continues until we reach the most significant bit column.

Table 1.5 Binary subtraction

Inputs			**Outputs**	
Minuend (A)	**Subtrahend (B)**	**Borrow-in (B_{in})**	**Difference (D)**	**Borrow-out (B_o)**
0	0	0	0	0
0	0	1	1	1
0	1	0	1	1
0	1	1	0	1
1	0	0	1	0
1	0	1	0	0
1	1	0	0	0
1	1	1	1	1

Addition of Larger-Bit Binary Numbers:

The addition of larger binary integers, fractions or mixed binary numbers is performed column wise in just the same way as in the case of decimal numbers. In the case of binary numbers, however, we follow the basic rules of addition of two or three binary digits, as outlined earlier. The process of adding two larger-bit binary numbers can be best illustrated with the help of an example.

Consider two generalized four-bit binary numbers (A_3 A_2 A_1 A_0) and (B_3 B_2 B_1 B_0), with A_0 and B_0 representing the LSB and A_3 and B_3 representing the MSB of the two numbers. The addition of these two numbers is performed as follows. We begin with the LSB position. We add the LSB bits and record the sum S_0 below these bits in the same column and take the carry C_0, if any, to the next column of bits. For instance, if $A_0 = 1$ and $B_0 = 0$, then $S_0 = 1$ and $C_0 = 0$. Next, we add the bits A_1 and B_1 and the carry C_0 from the previous addition. The process continues until we reach the MSB bits. The four steps are shown ahead. C_0, C_1, C_2 and C_3 are carry's, if any, produced as a result of adding first, second, third and fourth column bits respectively, starting from LSB and proceeding towards MSB. A similar procedure is followed when the given numbers have both integer as well as fractional parts:

$$\begin{array}{lcccc}
 & & & (C_0) & \\
1. & A_3 & A_2 & A_1 & A_0 \\
 & B_3 & B_2 & B_1 & B_0 \\
\hline
 & & & & S_0
\end{array}
\qquad
\begin{array}{lcccc}
 & & (C_1) & (C_0) & \\
2. & A_3 & A_2 & A_1 & A_0 \\
 & B_3 & B_2 & B_1 & B_0 \\
\hline
 & & & S_1 & S_0
\end{array}$$

$$\begin{array}{lcccc}
 & (C_2) & (C_1) & (C_0) & \\
3. & A_3 & A_2 & A_1 & A_0 \\
 & B_3 & B_2 & B_1 & B_0 \\
\hline
 & & S_2 & S_1 & S_0
\end{array}
\qquad
\begin{array}{lccccc}
 & & (C_2) & (C_1) & (C_0) & \\
4. & & A_3 & A_2 & A_1 & A_0 \\
 & & B_3 & B_2 & B_1 & B_0 \\
\hline
 & C_3 & S_3 & S_2 & S_1 & S_0
\end{array}$$

Addition Using the 2's Complement Method

The 2's complement is the most commonly used code for processing positive and negative binary numbers. It forms the basis of arithmetic circuits in modern computers. When the decimal numbers to be added are expressed in 2's complement form, the addition of these numbers, following the basic laws of binary addition, gives correct results. Final carry obtained, if any, while adding MSBs should be disregarded. To illustrate this, we will consider the following four different cases:

1. Both the numbers are positive.
2. Larger of the two numbers is positive.
3. The larger of the two numbers is negative.
4. Both the numbers are negative.

Case 1

• Consider the decimal numbers +37 and +18.

• The 2's complement of +37 in eight-bit representation = 00100101.

• The 2's complement of +18 in eight-bit representation = 00010010.

• The addition of the two numbers, that is, +37 and +18, is performed as follows

$$\begin{array}{r} 00100101 \\ +\ \underline{00010010} \\ \underline{00110111} \end{array}$$

• The decimal equivalent of $(00110111)_2$ is (+55), which is the correct answer.

Case 2

• Consider the two decimal numbers +37 and -18.

• The 2's complement representation of +37 in eight-bit representation = 00100101.

• The 2's complement representation of −18 in eight-bit representation = 11101110.

• The addition of the two numbers, that is, +37 and −18, is performed as follows:

$$\begin{array}{r} 00100101 \\ +\ \underline{11101110} \\ \underline{00010011} \end{array}$$

• The final carry has been disregarded.

• The decimal equivalent of $(00010011)_2$ is +19, which is the correct answer.

Case 3

• Consider the two decimal numbers +18 and −37.

• −37 in 2's complement form in eight−bit representation = 11011011.

• +18 in 2's complement form in eight−bit representation = 00010010.

• The addition of the two numbers, that is, −37 and +18, is performed as follows:

$$\begin{array}{r} 11011011 \\ +\ \underline{00010010} \\ \underline{11101101} \end{array}$$

• The decimal equivalent of $(11101101)_2$, which is in 2’s complement form, is −19, which is the correct answer. 2’s complement representation was discussed in detail in Chapter 1 on number systems.

Case 4

• Consider the two decimal numbers −18 and −37.

• −18 in 2’s complement form is 11101110.

• −37 in 2’s complement form is 11011011.

• The addition of the two numbers, that is, −37 and −18, is performed as follows

$$\begin{array}{r} 11011011 \\ +\ \underline{11101110} \\ \underline{11001001} \end{array}$$

• The final carry in the ninth bit position is disregarded.

• The decimal equivalent of $(11001001)_2$, which is in 2’s complement form, is −55, which is the correct answer.

It may also be mentioned here that, in general, 2’s complement notation can be used to perform addition when the expected result of addition lies in the range from −2n−1 to +(2n−1 − 1), n being the number of bits used to represent the numbers. As an example, eight-bit 2’s complement arithmetic cannot be used to perform addition if the result of addition lies outside the range from −128 to +127. Different steps to be followed to do addition in 2’s complement arithmetic are summarized as follows:

1. Represent the two numbers to be added in 2’s complement form.
2. Do the addition using basic rules of binary addition.
3. Disregard the final carry, if any.
4. The result of addition is in 2’s complement form.

Example

Perform the following addition operations:

1. $(275.75)_{10}$+ $(37.875)_{10}$

2. $(AF1.B3)_{16}$+ $(FFF.E)_{16}$

Solution:1

1. As a first step, the two given decimal numbers will be converted into their equivalent binary numbers (decimal-to-binary conversion has been covered at length in Chapter 1, and therefore the decimal-to-binary conversion details will not be given here):

$(275.75)_{10} = (100010011.11)_2$ and $(37.875)_{10} = (100101.111)_2$

The two binary numbers can be rewritten as $(100010011.110)_2$ and $(000100101.111)_2$ to have the same number of bits in their integer and fractional parts. The addition of two numbers is performed as follows:

$$\begin{array}{c} 100010011.110 \\ \underline{000100101.111} \\ \underline{100111001.101} \end{array}$$

The decimal equivalent of $(100111001.101)_2$ is $(313.625)_{10}$.

2. (AF1.B3)16 = $(101011110001.10110011)_2$ and $(FFF.E)_{16}$ = $(111111111111.1110)_2$. $(111111111111.1110)_2$ can also be written as $(111111111111.11100000)_2$ to have the same number of bits in the integer and fractional parts. The two numbers can now be added as follows:

$$\begin{array}{c} 0101011110001.10110011 \\ \underline{0111111111111.11100000} \\ \underline{1101011110001.10010011} \end{array}$$

The hexadecimal equivalent of $(1101011110001.10010011)_2$ is $(1AF1.93)_{16}$, which is equal to the hex addition of $(AF1.B3)_{16}$ and $(FFF.E)_{16}$.

Example:2

Find out whether 16-bit 2's complement arithmetic can be used to add 14 276 and 18 490.

Solution

The addition of decimal numbers 14 276 and 18 490 would yield 32 766. 16-bit 2's complement arithmetic has a range of −215 to +(215 − 1), i.e. −32 768 to +32 767. The expected result is inside the allowable range. Therefore, 16-bit arithmetic can be used to add the given numbers.

Example :3

Add −118 and −32 firstly using eight-bit 2's complement arithmetic and then using 16-bit 2's complement arithmetic. Comment on the results.

Solution:

• −118 in eight-bit 2's complement representation = 10001010.

• −32 in eight-bit 2's complement representation = 11100000.

• The addition of the two numbers, after disregarding the final carry in the ninth bit position, is 01101010. Now, the decimal equivalent of $(01101010)_2$, which is in 2's complement form, is +106.

The reason for the wrong result is that the expected result, i.e. −150, lies outside the range of eight-bit 2's complement arithmetic. Eight-bit 2's complement arithmetic can be used when the expected result lies in the range from −27 to + (27 − 1), i.e. −128 to +127. −118 in 16-bit 2's complement representation = 1111111110001010.

• −32 in 16-bit 2's complement representation = 1111111111100000.

• The addition of the two numbers, after disregarding the final carry in the 17th position, produces 1111111101101010. The decimal equivalent of $(1111111101101010)_2$, which is in 2's complement form, is −150, which is the correct answer. 16-bit 2's complement arithmetic has produced the correct result, as the expected result lies within the range of 16-bit 2's complement notation.

Subtraction of Larger-Bit Binary Numbers:

Subtraction is also done column wise in the same way as in the case of the decimal number system. In the first step, we subtract the LSBs and subsequently proceed towards the MSB. Wherever the subtrahend (the bit to be subtracted) is larger than the minuend, we borrow from the next adjacent higher bit position having a '1'. As an example, let us go through different steps of subtracting $(1001)_2$ from $(1100)_2$.

In this case, '1' is borrowed from the second MSB position, leaving a '0' in that position. The borrow is first brought to the third MSB position to make it '10'. Out of '10' in this position, '1' is taken to the LSB position to make '10' there, leaving a '1' in the third MSB position. 10 − 1 in the LSB column gives '1', 1 − 0 in the third MSB column gives '1', 0 − 0 in the second

MSB column gives ‘0’ and 1 − 1 in the MSB also gives ‘0’ to complete subtraction. Subtraction of mixed numbers is also done in the same manner. The above-mentioned steps are summarized as follows:

1. $$\begin{array}{cccc} 1&1&0&0\\ 1&0&0&1\\ \hline &&&1\\ \hline \end{array}$$

2. $$\begin{array}{cccc} 1&1&0&0\\ 1&0&0&1\\ \hline &&1&1\\ \hline \end{array}$$

3. $$\begin{array}{cccc} 1&1&0&0\\ 1&0&0&1\\ \hline 0&1&1&0\\ \hline \end{array}$$

4. $$\begin{array}{cccc} 1&1&0&0\\ 1&0&0&1\\ \hline &0&1&1\\ \hline \end{array}$$

Subtraction Using 2’s Complement Arithmetic:

Subtraction is similar to addition. Adding 2’s complement of the subtrahend to the minuend and disregarding the carry, if any, achieves subtraction. The process is illustrated by considering six different cases:

1. Both minuend and subtrahend are positive. The subtrahend is the smaller of the two.
2. Both minuend and subtrahend are positive. The subtrahend is the larger of the two.
3. The minuend is positive. The subtrahend is negative and smaller in magnitude.
4. The minuend is positive. The subtrahend is negative and greater in magnitude.
5. Both minuend and subtrahend are negative. The minuend is the smaller of the two.
6. Both minuend and subtrahend are negative. The minuend is the larger of the two.

Case 1

• Let us subtract +14 from +24.

• The 2’s complement representation of +24 = 00011000.

• The 2’s complement representation of +14 = 00001110.

• Now, the 2’s complement of the subtrahend (i.e. +14) is 11110010.

• Therefore, +24 − (+14) is given by

$$\begin{array}{r} 00011000\\ +\ \underline{11110010}\\ \underline{00001010} \end{array}$$

with the final carry disregarded.

• The decimal equivalent of $(00001010)_2$ is +10, which is the correct answer.

Case 2:

• Let us subtract +24 from +14.

• The 2's complement representation of +14 = 00001110.

• The 2's complement representation of +24 = 00011000.

• The 2's complement of the subtrahend (i.e. +24) = 11101000.

• Therefore, +14 − (+24) is given by

$$\begin{array}{r} 00001110 \\ +\ \underline{11101000} \\ \underline{11110110} \end{array}$$

• The decimal equivalent of $(11110110)_2$, which is of course in 2's complement form, is −10 which is the correct answer.

Case 3

• Let us subtract −14 from +24.

• The 2's complement representation of +24 = 00011000 = minuend.

• The 2's complement representation of −14 = 11110010 = subtrahend.

• The 2's complement of the subtrahend (i.e. −14) = 00001110.

• Therefore, +24 − (−14) is performed as follows:

$$\begin{array}{r} 00011000 \\ +\ \underline{00001110} \\ \underline{00100110} \end{array}$$

• The decimal equivalent of $(00100110)_2$ is +38, which is the correct answer.

Case 4

• Let us subtract −24 from +14.

• The 2's complement representation of +14 = 00001110 = minuend.

• The 2’s complement representation of −24 = 11101000 = subtrahend.

• The 2’s complement of the subtrahend (i.e. −24) = 00011000.

• Therefore, +14 − (−24) is performed as follows:

$$\begin{array}{r} 00001110 \\ +\ \underline{00011000} \\ \underline{00100110} \end{array}$$

• The decimal equivalent of $(00100110)_2$ is +38, which is the correct answer.

Case 5

• Let us subtract −14 from −24.

• The 2’s complement representation of −24 = 11101000 = minuend.

The 2’s complement representation of −14=11110010 = subtrahend.

• The 2’s complement of the subtrahend = 00001110.

• Therefore, −24 − (−14) is given as follows:

$$\begin{array}{r} 11101000 \\ +\ \underline{00001110} \\ \underline{11110110} \end{array}$$

• The decimal equivalent of $(11110110)_2$, which is in 2’s complement form, is −10, which is the correct answer.

Case 6

• Let us subtract −24 from −14.

• The 2’s complement representation of −14 = 11110010 = minuend.

• The 2’s complement representation of −24=11101000 = subtrahend.

• The 2’s complement of the subtrahend = 00011000.

• Therefore, −14 − (−24) is given as follows:

$$\begin{array}{r} 11110010 \\ +\ \underline{00011000} \end{array}$$

$$\underline{00001010}$$

with the final carry disregarded.

The decimal equivalent of $(00001010)_2$, which is in 2’s complement form, is +10, which is the correct answer. It may be mentioned that, in 2’s complement arithmetic, the answer is also in 2’s complement notation, only with the MSB indicating the sign and the remaining bits indicating the magnitude. In 2’s complement notation, positive magnitudes are represented in the same way as the straight binary numbers, while the negative magnitudes are represented as the 2’s complement of their straight binary counterparts. A ‘0’ in the MSB position indicates a positive sign, while a ‘1’ in the MSB position indicates a negative sign. The different steps to be followed to do subtraction in 2’s complement arithmetic are summarized as follows:

1. Represent the minuend and subtrahend in 2’s complement form.
2. Find the 2’s complement of the subtrahend.
3. Add the 2’s complement of the subtrahend to the minuend.
4. Disregard the final carry, if any.
5. The result is in 2’s complement form.
6. 2’s complement notation can be used to perform subtraction when the expected result of subtraction lies in the range from −2n−1 to +(2n−1 − 1), n being the number of bits used to represent the numbers.

Example

Subtract $(1110.011)_2$ from $(11011.11)_2$ using basic rules of binary subtraction and verify the result by showing equivalent decimal subtraction.

Solution:

The minuend and subtrahend are first modified to have the same number of bits in the integer and fractional parts. The modified minuend and subtrahend are $(11011.110)_2$ and $(01110.011)_2$ respectively:

$$\begin{array}{r} 11011.110 \\ -\ \underline{01110.011} \end{array}$$

$$\underline{01101.011}$$

The decimal equivalents of $(11011.110)_2$ and $(01110.011)_2$ are 27.75 and 14.375 respectively. Their difference is 13.375, which is the decimal equivalent of $(01101.011)_2$.

Example

Subtract (a) $(-64)_{10}$ from $(+32)_{10}$ and (b) $(29.A)_{16}$ from $(4F.B)_{16}$. Use 2's complement arithmetic.

Solution:

(a) $(+32)_{10}$ in 2's complement notation = $(00100000)_2$.

$(-64)_{10}$ in 2's complement notation = $(11000000)_2$.

The 2's complement of $(-64)_{10}$ = $(01000000)_2$.

$(+32)_{10} - (-64)_{10}$ is determined by adding the 2's complement of $(-64)_{10}$ to $(+32)_{10}$.

Therefore, the addition of $(00100000)_2$ to $(01000000)_2$ should give the result. The operation is shown as follows:

$$\begin{array}{r} 00100000 \\ +\ \underline{01000000} \\ \underline{01100000} \end{array}$$

The decimal equivalent of $(01100000)_2$ is +96, which is the correct answer as $+32 - (-64) = +96$.

(b) The minuend = $(4F.B)_{16}$ = $(01001111.1011)_2$.

The minuend in 2's complement notation = $(01001111.1011)_2$.

The subtrahend = $(29.A)_{16}$ = $(00101001.1010)_2$.

The subtrahend in 2's complement notation = $(00101001.1010)_2$.

The 2's complement of the subtrahend = $(11010110.0110)_2$.

$(4F.B)_{16} - (29.A)_{16}$ is given by the addition of the 2's complement of the subtrahend to the minuend.

$$\begin{array}{r} 01001111.1011 \\ +\ \underline{11010110.0110} \\ \underline{00100110.0001} \end{array}$$

with the final carry disregarded. The result is also in 2's complement form. Since the result is a positive number, 2's complement notation is the same as it would be in the case of the straight binary code. The hex equivalent of the resulting binary number = $(26.1)_{16}$, which is the correct answer.

BCD Addition and Subtraction in Excess-3 Code:

Below, we will see how the excess-3 code can be used to perform addition and subtraction operations on BCD numbers.

Addition:

The excess-3 code can be very effectively used to perform the addition of BCD numbers. The steps to be followed for excess-3 addition of BCD numbers are as follows:

1. The given BCD numbers are written in excess-3 form by adding '0011' to each of the four-bit groups.
2. The two numbers are then added using the basic laws of binary addition.
3. Add '0011' to all those four-bit groups that produce a carry, and subtract '0011' from all those four-bit groups that do not produce a carry during addition.
4. The result thus obtained is in excess-3 form.

Subtraction:

Subtraction of BCD numbers using the excess-3 code is similar to the addition process discussed above. The steps to be followed for excess-3 subtraction of BCD numbers are as follows:

1. Express both minuend and subtrahend in excess-3 code.
2. Perform subtraction following the basic laws of binary subtraction.
3. Subtract '0011' from each invalid BCD four-bit group in the answer.
4. Subtract '0011' from each BCD four-bit group in the answer if the subtraction operation of the relevant four-bit groups required a borrow from the next higher adjacent four-bit group.
5. Add '0011' to the remaining four-bit groups, if any, in the result.
6. This gives the result in excess-3 code.

The process of addition and subtraction can be best illustrated with the help of following examples.

Example:

Add $(0011\ 0101\ 0110)_{BCD}$ and $(0101\ 0111\ 1001)_{BCD}$ using the excess-3 addition method and verify the result using equivalent decimal addition.

Solution

The excess-3 equivalents of 0011 0101 0110 and 0101 0111 1001 are 0110 1000 1001 and 1000 1010 1100 respectively. The addition of the two excess-3 numbers is given as follows:

$$\begin{array}{c} 0110\ 1000\ 1001 \\ \underline{1000\ 1010\ 1100} \\ \underline{1111\ 0011\ 0101} \end{array}$$

After adding 0011 to the groups that produced a carry and subtracting 0011 from the groups that did not produce a carry, we obtain the result of the above addition as 1100 0110 1000. Therefore, 1100 0110 1000 represents the excess-3 code for the true result. The result in BCD code is 1001 0011 0101, which is the BCD equivalent of 935. This is the correct answer as the addition of the given BCD numbers 0011 0101 0110 = $(356)_{10}$ and 0101 0111 1001 = $(579)_{10}$ yields $(935)_{10}$ only.

Example:

Perform $(185)_{10}-(8)_{10}$ using the excess-3 code.

Solution:

• $(185)_{10} = (0001\ 1000\ 0101)_{BCD}$. The excess-3 equivalent of $(0001\ 1000\ 0101)_{BCD}$ = 0100 1011 1000.

• $(8)_{10} = (008)_{10} = (0000\ 0000\ 1000)_{BCD}$. The excess-3 equivalent of $(0000\ 0000\ 1000)_{BCD}$ = 0011 0011 1011.

• Subtraction is performed as follows:

$$\begin{array}{r} 0100\ 1011\ 1000 \\ -\ \underline{0011\ 0011\ 1011} \end{array}$$

0001 0111 1101

• In the subtraction operation, the least significant column of four-bit groups needed a borrow, while the other two columns did not need any borrow. Also, the least significant column has produced an invalid BCD code group. Subtracting 0011 from the result of this column and adding 0011 to the results of other two columns, we get 0100 1010 1010. This now constitutes the result of subtraction expressed in excess-3 code.

• The result in BCD code is therefore 0001 0111 0111.

• The decimal equivalent of 0001 0111 0111 is 177, which is the correct result.

1.4. Boolean laws:

The laws of Boolean algebra can be used to simplify many a complex Boolean expression and also to transform the given expression into a more useful and meaningful equivalent expression.

These are the rules we use to simplify logical expressions and design efficient circuits.

1. Identity Law:

In the Boolean Algebra, we have identity elements for both AND(.) and OR (+) operations. The identity law states that in Boolean algebra, we have such variables that, on operating with the AND and OR operations we get the same result, i.e.

- $A + 0 = A$
- $A.1 = A$

2. Commutative Law:

Binary variables in Boolean Algebra follow the commutative law. This law states that operating Boolean variables A and B is similar to operating Boolean variables B and A. That is,

- $A. B = B. A$
- $A + B = B + A$

3. Associative Law:

Associative law states that the order of performing Boolean operator is illogical as their result is always the same. This can be understood as,

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

4. Distributive Law:

Boolean Variables also follow the distributive law, and the expression for the Distributive law is given as:

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

5. Inversion Law:

Inversion law is the unique law of Boolean algebra that states, the complement of the complement of any number is the number itself.

- $(A')' = A$

Apart from these other laws are mentioned below:

6. AND Law:

AND law of the Boolean algebra uses AND operator and the AND law is,

- $A . 0 = 0$
- $A . 1 = A$
- $A . A = A$

7. OR Law:

OR law of the Boolean algebra uses OR operator and the OR law is,

- $A + 0 = A$
- $A + 1 = 1$
- $A + A = A$

8. Complement Law:

The **Complement Law** states that a variable ORed with its complement is always 1, and a variable ANDed with its complement is always 0.

- $A + A' = 1$
- $A . A' = 0$

9. Domination Law:

The **Domination Law** states that any variable ORed with 1 will always be 1, and any variable ANDed with 0 will always be 0.

- $A + 1 = 1$
- $A . 0 = 0$

10. Double Negation Law:

The **Double Negation Law** states that the complement of the complement of a variable is the variable itself.

- $(A')' = A$

1.6.DeMorgan'sTheorem:

De Morgan's Theorem is a powerful theorem in Boolean algebra which has a set of two rules or laws. These two laws were developed to show the relationship between two variable AND, OR, and NOT operations. These two rules enable the variables to be negated, i.e. opposite of their original form. Therefore, DeMorgan's theorem gives the dual of a logic function.

Now, let us discuss the two laws of De Morgan's theorem.

De Morgan's First Theorem (Law 1):

De Morgan's First Law states that the complement of a sum (ORing) of variables is equal to the product (ANDing) of their individual complements. In other words, the complement of two or more ORed variables is equivalent to the AND of the complements of each of the individual variables, i.e.

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

Or, it may also be represented as,

$(A+B)' = A' \cdot B'$

The logic implementation of left side and right side of this law is shown in Figure.

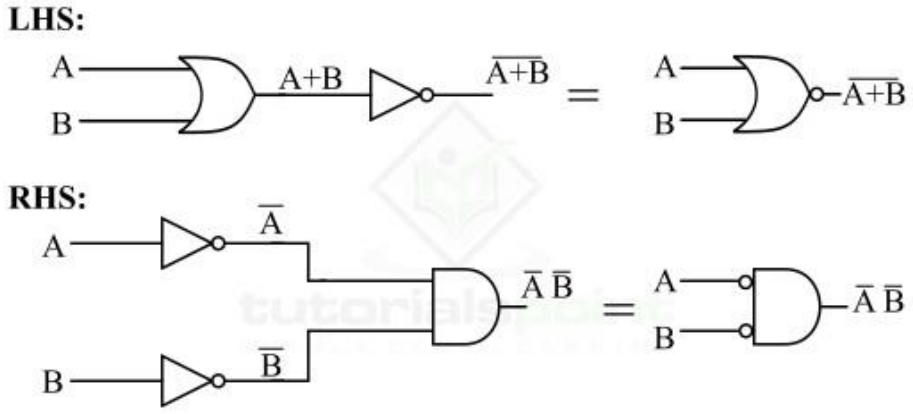


Figure 1 - De Morgan's First Law

Thus, DeMorgan's first law proves that the NOR gate is equivalent to a bubbled AND gate. The following truth table shows the proof of this law.

Left Side			**Right Side**		
Input		**Output**	**Input**		**Output**
A	**B**	**(A + B)'**	**A'**	**B'**	**A'· B'**
0	0	1	1	1	1
0	1	0	1	0	0
1	0	0	0	1	0
1	1	0	0	0	0

This truth table proves that the Boolean expression on the left is equivalent to that on the right side of the expression of De Morgan's first law.

Also, the first law of De Morgan's theorem can be extended to any number of variables, or a combination of variables.

De Morgan's Second Theorem (Law 2)

De Morgan's second law states that the complement of the product (ANDing) of variables is equivalent to the sum (ORing) of their individual complements.

In other words, the complement of two or more ANDed variables is equal to the sum of the complement of each of the individual variables, i.e.,

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

It may also be represented as,

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

The logic implementation of left and right sides of this expression is shown in Figure 2.

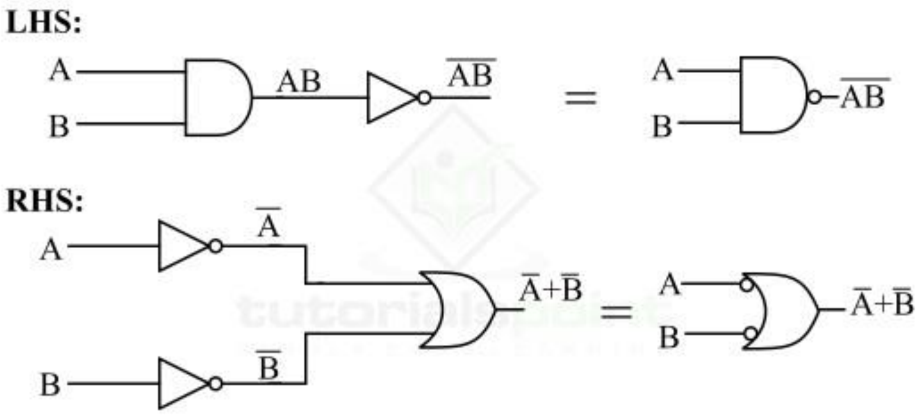


Figure 2 - De Morgan's Second Law

Hence, De Morgan's second law proves that the NAND gate is equivalent to a bubbled OR gate. The following truth table shows the proof of this law.

Left Side			**Right Side**		
Input		**Output**	**Input**		**Output**
A	**B**	**AB**	**A'**	**B'**	**A' + B'**

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

This truth table proves that the Boolean expression on the left side is equivalent to that on the right side of the expression of De Morgan's second law.

Similar to the first law, we may extend the De Morgan's second law for any number of variables or combination of variables.

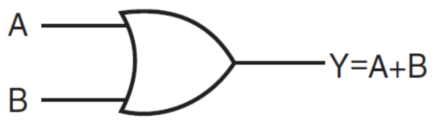
1.7. Basic logic gates:

The logic gate is the most basic building block of any digital system, including computers. Each one of the basic logic gates is a piece of hardware or an electronic circuit that can be used to implement some basic logic expression. While laws of Boolean algebra could be used to do manipulation with binary variables and simplify logic expressions, these are actually implemented in a digital system with the help of electronic circuits called logic gates. The three basic logic gates are the OR gate, the AND gate and the NOT gate.

OR Gate:

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

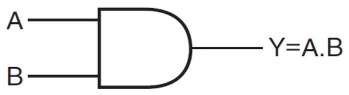
$$Y = A+B$$



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

AND Gate:

An AND gate is a logic circuit having two or more inputs and one output. The output of an AND gate is HIGH only when all of its inputs are in the HIGH state. In all other cases, the output is LOW. When interpreted for a positive logic system, this means that the output of the AND gate is a logic '1' only when all of its inputs are in logic '1' state. In all other cases, the output is logic '0'. The logic symbol and truth table of a two-input AND gate are shown in Figs (a) and (b) respectively. The AND operation on two independent logic variables A and B is written as Y = A.B and reads as Y equals A AND B and not as A multiplied by B. Here, A and B are input logic variables and Y is the output. An AND gate performs an ANDing operation:



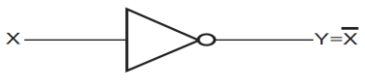
(a)

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

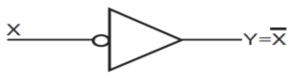
(b)

NOT Gate:

A NOT gate is a one-input, one-output logic circuit whose output is always the complement of the input. That is, a LOW input produces a HIGH output, and vice versa. When interpreted for a positive logic system, a logic ‘0’ at the input produces a logic ‘1’ at the output, and vice versa. It is also known as a ‘complementing circuit’ or an ‘inverting circuit’. Figure shows the circuit symbol and the truth table. The NOT operation on a logic variable X is denoted as $\overline{X}$ or X’. That is, if X is the input to a NOT circuit, then its output Y is given by Y = $\overline{X}$ or X’ and reads as Y equals NOT X. Thus, if X = 0, Y = 1 and if X = 1, Y = 0.



(a)



X	Y
0	1
1	0

(b)

1.7. Universal logic gates (NAND & NOR):

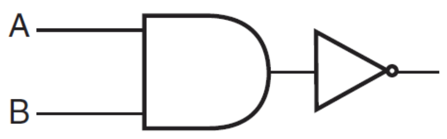
NAND Gate:

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

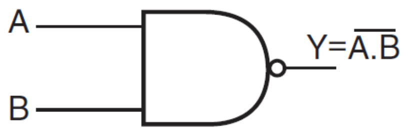
$$Y = \overline{A \cdot B}$$

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

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



(a)



(b)

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

(c)

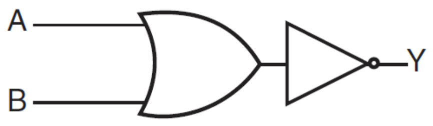
NOR Gate:

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

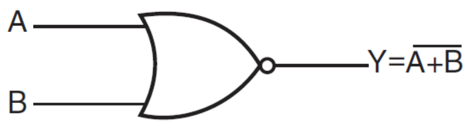
$$Y = \overline{(A + B)}$$

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

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



(a)



(b)

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

(c)

1.8. standard representation of logic functions (SOP & POS):

SOP and POS are two of the important topics of the Boolean algebra. SOP refers to the sum of the product of the minterms, whereas POS is the Product of the sum of terms. SOP and POS help us simplify the complex Boolean variables and are mainly used in K-maps.

What is SOP?

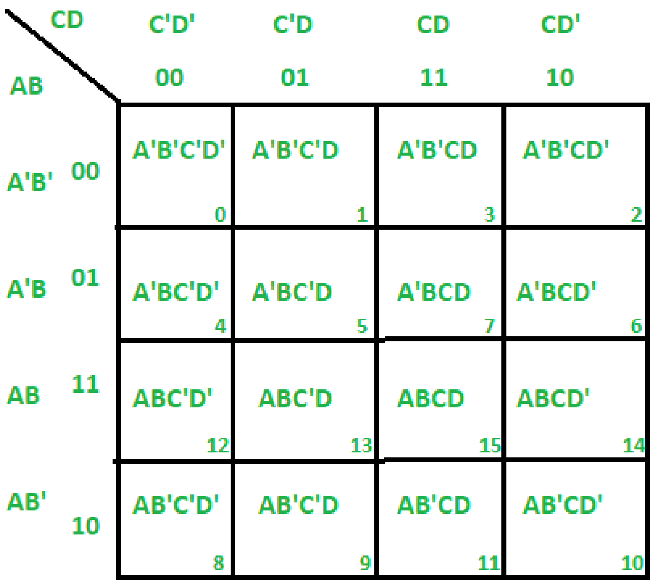
The SOP stands for Sum of Products defined as the sum of minterms. As the name suggests the SOP form represents the OR operation of the product terms i.e., minterms. The SOP represents the minterms in the Boolean algebra. The SOP is denoted by ∑. As the SOP deals with the minterms it works on active high logic.

Example of SOP

- AB + CD
- P'Q + R
- X'Y' + W'Z'

SOP in K-Map

The below image represents the SOP in K-map.



AB \ CD	C'D' 00	C'D 01	CD 11	CD' 10
A'B' 00	A'B'C'D' 0	A'B'C'D 1	A'B'CD 3	A'B'CD' 2
A'B 01	A'BC'D' 4	A'BC'D 5	A'BCD 7	A'BCD' 6
AB 11	ABC'D' 12	ABC'D 13	ABCD 15	ABCD' 14
AB' 10	AB'C'D' 8	AB'C'D 9	AB'CD 11	AB'CD' 10

SOP(MINTERMS)

What is POS?

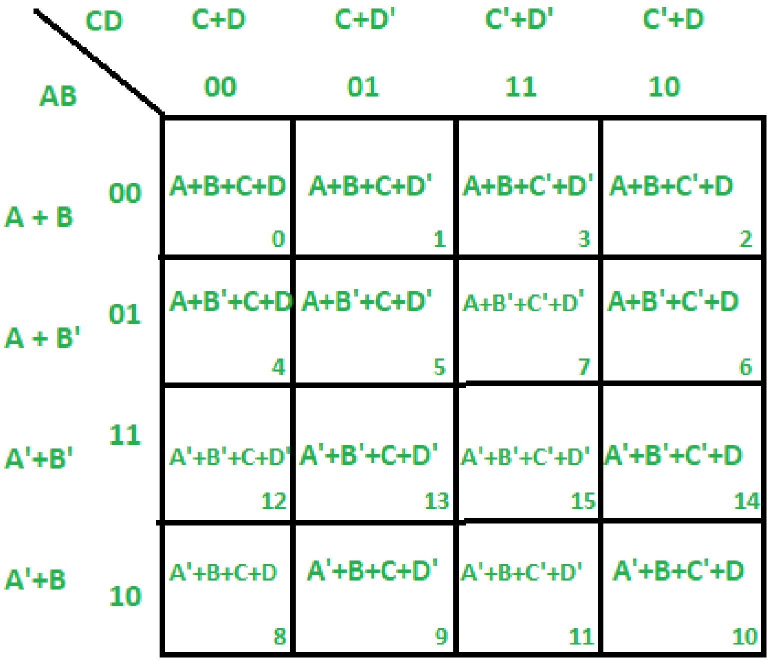
The POS stands for Product of Sum defined as the product of maxterms. As the name suggests the POS form represents the AND operation of the sum terms i.e., maxterms. The POS represents the maxterms in Boolean algebra. The POS is denoted by Π. As the POS deals with the maxterms it works on active low logic.

Example of POS

- (A + B).(C + D)
- (P' + Q).(R + S')

POS in K-Map

Below image represents the POS in K-map.



AB \ CD		C+D 00	C+D' 01	C'+D' 11	C'+D 10
A + B	00	A+B+C+D 0	A+B+C+D' 1	A+B+C'+D' 3	A+B+C'+D 2
A + B'	01	A+B'+C+D 4	A+B'+C+D' 5	A+B'+C'+D' 7	A+B'+C'+D 6
A'+B'	11	A'+B'+C+D 12	A'+B'+C+D' 13	A'+B'+C'+D' 15	A'+B'+C'+D 14
A'+B	10	A'+B+C+D 8	A'+B+C+D' 9	A'+B+C'+D' 11	A'+B+C'+D 10

POS(MAXTERMS)

SOP vs POS

Characterization	**SOP**	**POS**
Definition	SOP is sum of the minterms.	POS is the product of maxterms.
Stands for	SOP stands for Sum of Product.	POS stands for Product of Sum.
Representation	It represents minterms.	It represents maxterms.

Characterization	SOP	POS
Logic	It works on active high logic.	It works on active low logic.
Denotation	It is denoted by Σ.	It is denoted by Π.
Output	The output of SOP is 1.	The output of POS is 0.

1.9. Minimization techniques (Karnaugh map: 2, 3, 4 variables):

Karnaugh Map Method:

In many digital circuits and practical problems, we need to find expressions with minimum variables. We can minimize Boolean expressions of 3, 4 variables very easily using K-map without using any Boolean algebra theorems. It is a tool which is used in digital logic to simplify boolean expression. It helps to simplify logic into simpler form by organizing grid from truth table values. This helps it to create a minimal Boolean expressions by identifying patterns.

K-map can take two forms:

1. Sum of product (SOP)
2. Product of Sum (POS)

According to the need of problem. K-map is a table-like representation, but it gives more information than the TABLE. We fill a grid of the K-map with 0's and 1's then solve it by making groups.

Steps to Solve Expression using K-map:

1. Select the K-map according to the number of variables.
2. Identify minterms or maxterms as given in the problem.
3. For SOP put 1's in blocks of K-map respective to the minterms (0's elsewhere).

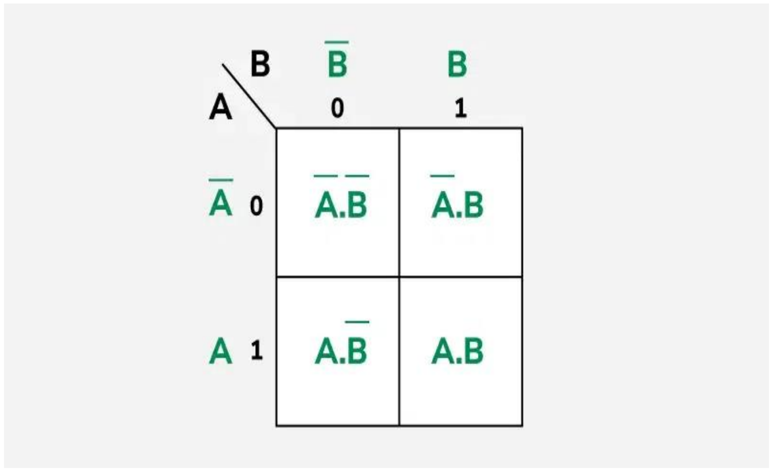
4. For POS put 0's in blocks of K-map respective to the max terms (1's elsewhere).
5. Make rectangular groups containing total terms in power of two like 2,4,8 ..(except 1) and try to cover as many elements as you can in one group.
6. From the groups made in step 5 find the product terms and sum them up for SOP form.

SOP FORM (Sum of Product Form):

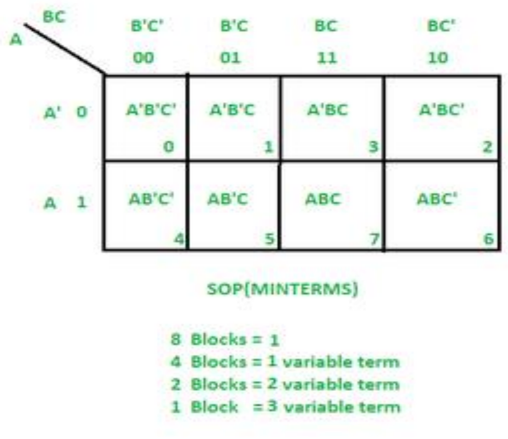
SOP form is way to simplify and write Boolean expressions using AND to combine inputs and OP to combine the results.

1. K-map for 2 variables:

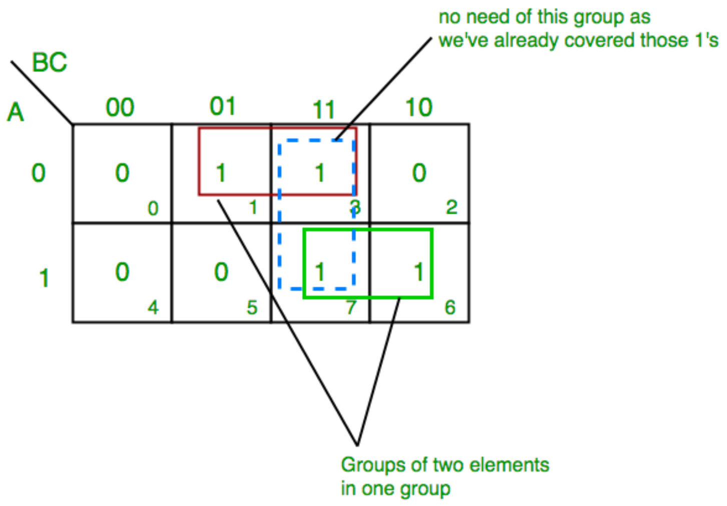
In the 2 variable k-map, four squares are constructed. Each square contains one term of expression with two variables.



2. K-map of 3 variables



Z= ΣA,B,C(1,3,6,7)


no need of this group as we've already covered those 1's
BC
A
00
01
11
10
0
0
0
1
1
1
3
0
2
1
0
4
0
5
1
7
1
6
Groups of two elements in one group

From red group we get product term:

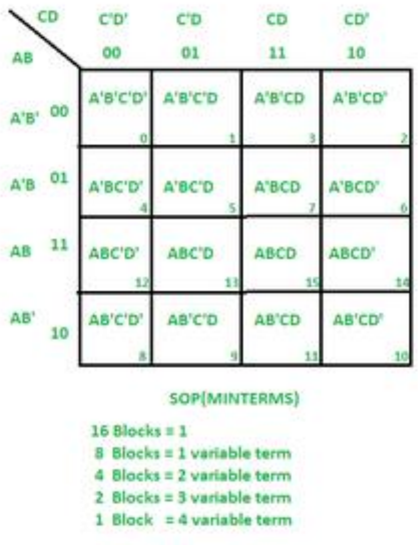
A'C

From green group we get product term:

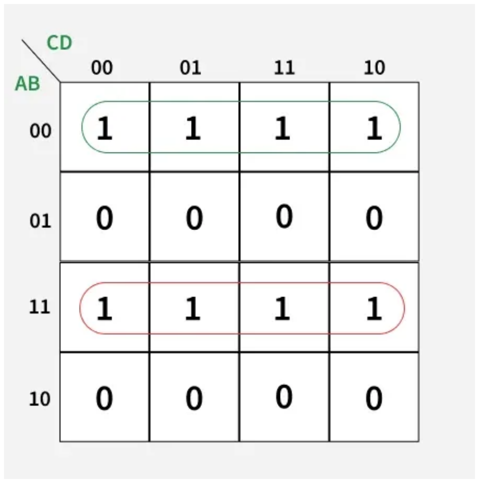
AB

Summing these product terms we get- Final expression (A'C+AB)

3. K-map for 4 variables



F(A,B,C,D)=Σ(0,1,2,3,12,13,15,14)



From red group we get product term:

AB

From green group we get product term:

A'B'

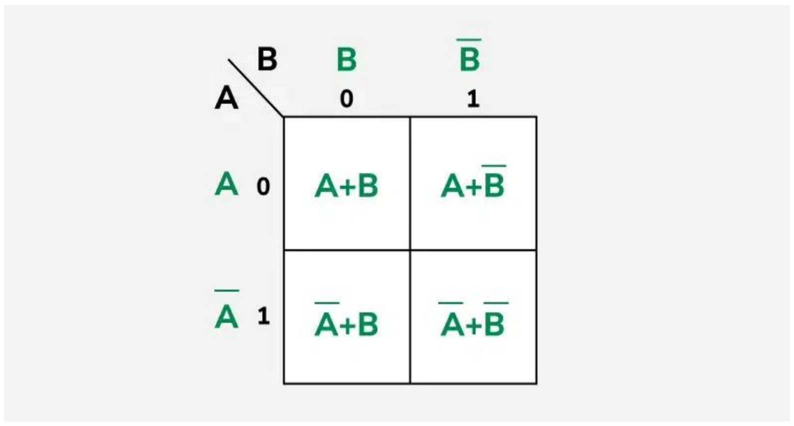
Summing these product terms we get- Final expression (AB+A’B').

POS FORM (Product of Sum Form):

POS form is a way to simplify and write Boolean expressions using OR to combine terms inside parentheses and then AND to combine those groups.

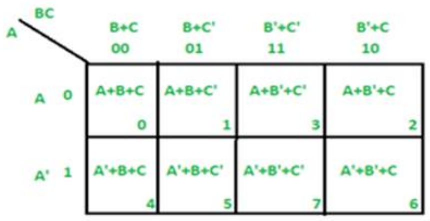
1.K-map of 2 variables

In the 2 variable k-map, four squares are constructed. Each square contains one term of expression with two variables.



2. K-map of 3 variables:

K-map 3 variable POS form

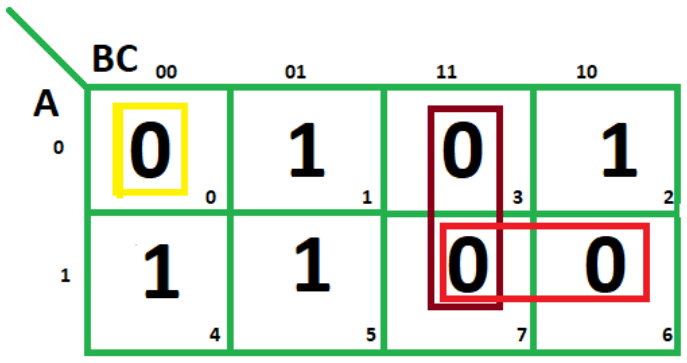


POS (MAXTERMS)

8 Blocks = 0
4 Blocks = 1 variable term
2 Blocks = 2 variable term
1 Block = 3 variable term

.

F(A,B,C)=Σ(0,3,6,7)



From red group we find terms

A B

Taking complement of these two

A' B'

Now sum up them

(A' + B')

From brown group we find terms

B C

Taking complement of these two terms

B’ C’

Now sum up them

(B’+C’)

From yellow group we find terms

A' B' C’

Taking complement of these two

A B C

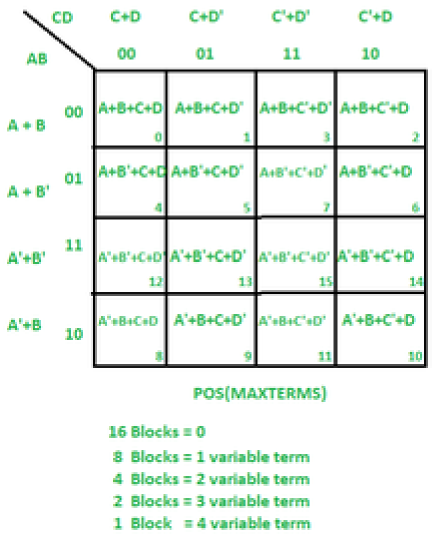
Now sum up them

(A + B + C)

We will take product of these three terms : Final expression -

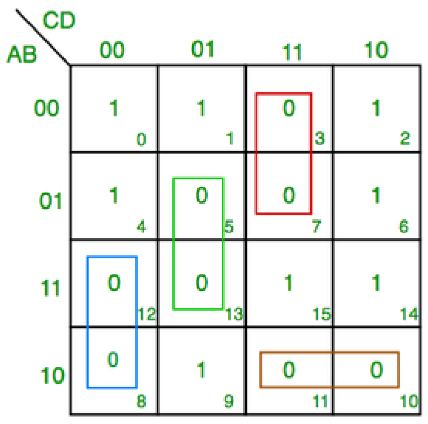
(A' + B’) (B’ + C’) (A + B + C)

3. K-map of 4 variables:



K-map 4 variable POS form

F(A,B,C,D)=Σ(3,5,7,8,10,11,12,13)



From green group we find terms

C’ D B

Taking their complement and summing them

(C+D’+B’)

From red group we find terms

C D A’

Taking their complement and summing them

(C’+D’+A)

From blue group we find terms

A C’ D’

Taking their complement and summing them

(A’+C+D)

From brown group we find terms

A B’ C

Taking their complement and summing them

(A’+B+C’)

Finally we express these as product -

(C+D’+B’).(C’+D’+A).(A’+C+D).(A’+B+C’)

Advantages of K-MAP:

- Makes Logic Simpler: It makes complicated Boolean expressions simpler.
- Minimizes Logic Gates: Simplifying the logic helps us to use fewer logic gates, making circuits more efficient.
- Reduce Errors: The visual representation of k-map helps to avoid errors while simplifying.
- Time-Saving: It's quicker than traditional methods for simplifying logic.

- No need for Boolean Laws: K-map doesn't require deep knowledge of Boolean laws, making it easy for beginners.

UNIT-II: ADDERS: half & full adder – subtractors: half & full subtractor – parallel binary adder – magnitude comparator – multiplexers (4:1) & demultiplexers (1:4), encoder (8-line to 3-line) and decoder (3-line to 8-line), BCD to seven segment decoders.

2.1. Half & full adder:

Half adder:

A half adder is a basic combinational circuit that adds two single-bit binary inputs (A and B) to produce a SUM using an XOR gate and a CARRY using an AND gate, without considering any carry-in from a previous stage.

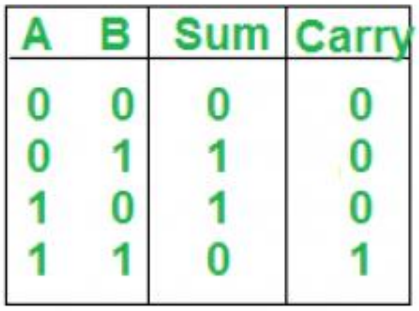
- Performs binary addition of two 1-bit inputs, generating a SUM ($A \oplus B$) and CARRY ($A \cdot B$).
- Cannot handle carry-in from a previous stage, making it suitable only for the first stage of multi-bit addition.

The Boolean expressions for the SUM and CARRY outputs are given by the equations

- SUM $S = A.\overline{B} +.\overline{A}.B$
- CARRY $C = A.B$

Truth Table of Half Adder:

Below is the truth table, illustrating the operation of a half adder



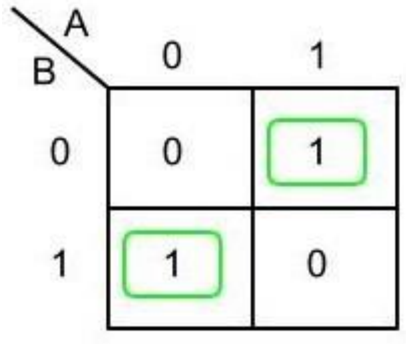
A	B	Sum	Carry
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

Logical Expression of Half Adder

Here we perform two operations Sum and Carry, thus we need two K-maps one for each to derive the expression.

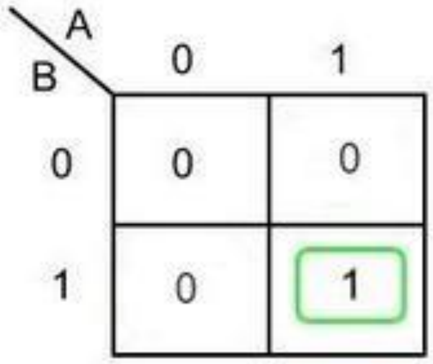
For

Sum = A XOR B



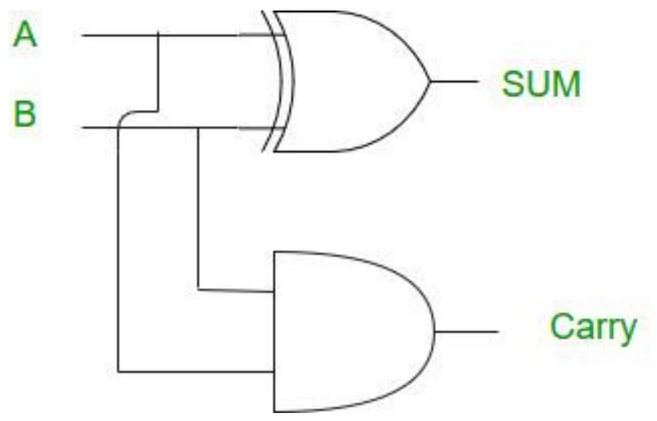
For

Carry = A AND B



Implementation of Half Adder:

Half adder has only two inputs and there is no provision to add a carry coming from the lower order bits when multi addition is performed.



Advantages of Half Adder in Digital Logic

- **Simplicity:** A half adder is a straightforward circuit that requires a couple of fundamental parts like XOR AND entryways. It is not difficult to carry out and can be utilized in numerous advanced frameworks.
- **Speed:** The half adder works at an extremely rapid, making it reasonable for use in fast computerized circuits.

Application of Half Adder in Digital Logic:

- **Arithmetic circuits:** Half adders are utilized in number-crunching circuits to add double numbers. At the point when different half adders are associated in a chain, they can add multi-bit double numbers.
- **Data handling:** Half adders are utilized in information handling applications like computerized signal handling, information encryption, and blunder adjustment.
- **Address unravelling:** In memory tending to, half adders are utilized in address deciphering circuits to produce the location of a particular memory area.
- **Encoder and decoder circuits:** Half adders are utilized in encoder and decoder circuits for computerized correspondence frameworks.
- **Multiplexers and demultiplexers:** Half adders are utilized in multiplexers and demultiplexers to choose and course information.
- **Counters:** Half adders are utilized in counters to augment the count by one.

Full Adder:

Full Adder is a combinational circuit that adds three inputs and produces two outputs. The first two inputs are A and B and the third input is an input carry as C-IN. The output carry is designated as C-OUT and the normal output is designated as S which is SUM.

- The C-OUT is also known as the majority 1's detector, whose output goes high when more than one input is high.
- A full adder logic is designed in such a manner that can take eight inputs together to create a byte-wide adder and cascade the carry bit from one adder to another.
- We use a full adder because when a carry-in bit is available, another 1-bit adder must be used since a 1-bit half-adder does not take a carry-in bit.
- A 1-bit full adder adds three operands and generates 2-bit results.

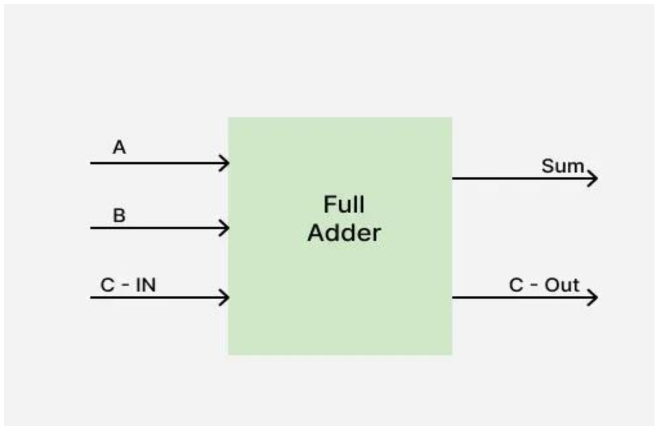
Full Adder Truth Table

A Full Adder takes three binary inputs:

- A (first bit)
- B (second bit)
- C-IN (carry input)

And it produces two outputs:

- Sum (S)
- Carry Out (C-OUT)



Here's the truth table for the full adder:

INPUT			**OUTPUT**	
A	**B**	**C-IN**	**Sum**	**C-OUT**
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

Logical Expressions for SUM

From the truth table, the logical expression for the sum (S) in a full adder is:

$S = A'B'C\text{-}IN + A'BC\text{-}IN' + AB'C\text{-}IN' + ABC\text{-}IN$

Since A'B + AB' =A ⊕ B. This simplifies to:

$S = C\text{-}IN\,(A \oplus B)' + C\text{-}IN'(A \oplus B)$

The final simplified expression is:

S = A ⊕ B ⊕ C-IN

Thus, the sum output is the XOR of **A**, **B**, and **C-IN**.

Logical Expression for C-OUT

From the truth table, the logical expression for **C-OUT** (carry-out) in a full adder is:

C-OUT = A′ B C-IN + A B′ C-IN + A B C-IN′ + A B C-IN

This simplifies to:

C-OUT = A B(C-IN′+C-IN) + C-IN(A′B+AB′)

Since C-IN' + C-IN =1 and A'B + AB' =A ⊕ B. Thus, the final simplified expression is:

C-OUT = A B + C-IN (A ⊕ B)

Logic Circuit of Full Adder

To implement a Full Adder using basic logic gates:

Sum (S) is implemented using XOR gates:

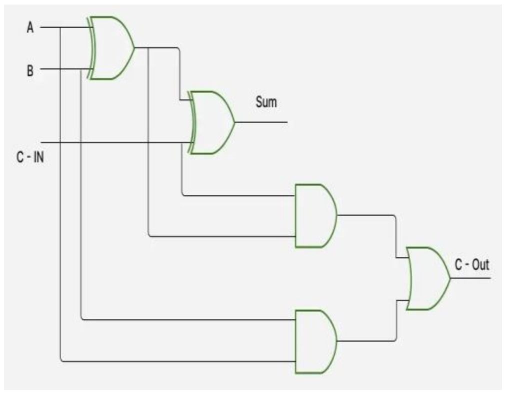
Use two XOR gates:

- First XOR gate: A ⊕ B
- Second XOR gate: (A ⊕ B) ⊕ C-IN to get the final sum S.

Carry (C-Out) is implemented using XOR, AND and OR gates: Finally, the two outputs from the AND gates are combined using an OR gate to generate the final C-OUT output.

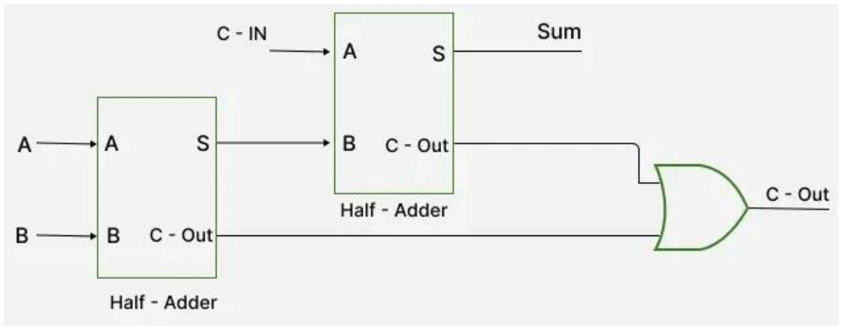
First AND gate: This gate calculates A AND B.

Second AND gate: This gate calculates C-IN AND (A ⊕ B). To do this, you need the result of the first XOR gate (A ⊕ B) as an input to the second AND gate.



Implementation of Full Adder using Half Adders

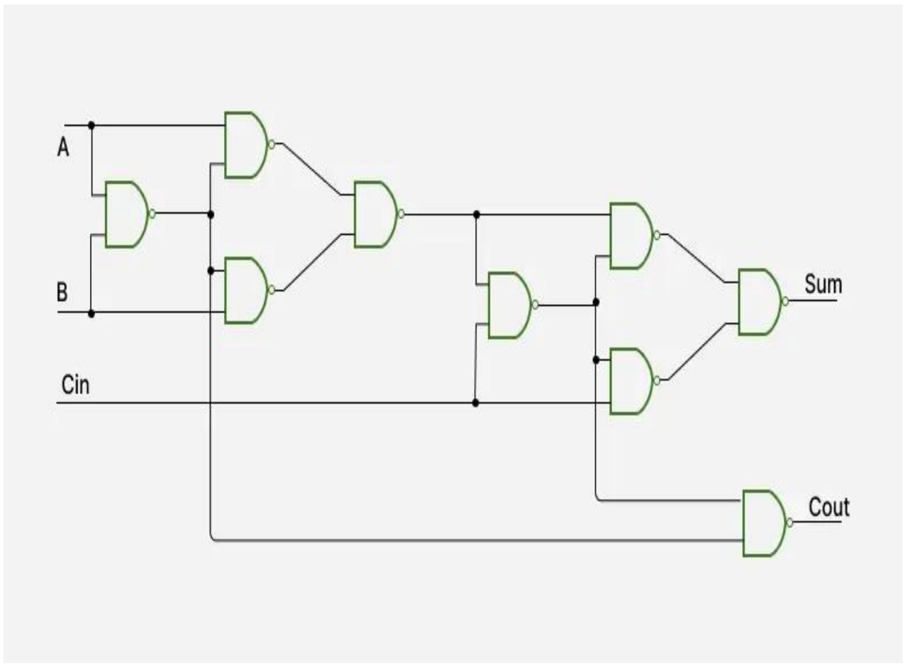
2 Half Adders and an OR gate is required to implement a Full Adder.



With this logic circuit, two bits can be added together, taking a carry from the next lower order of magnitude, and sending a carry to the next higher order of magnitude.

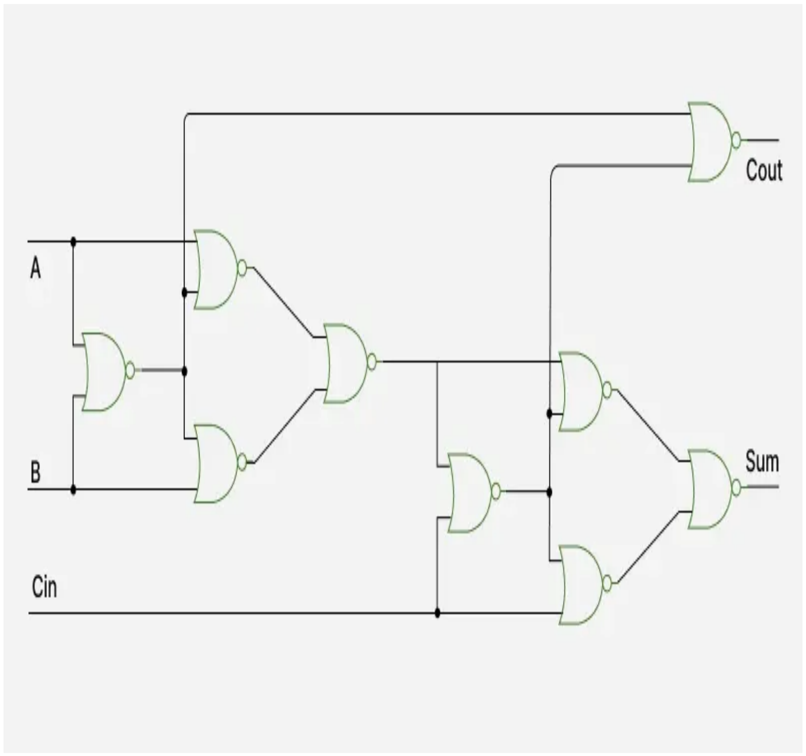
Implementation of Full Adder using NAND gates

Total 9 NAND gates are required to implement a Full Adder.



Implementation of Full Adder using NOR gates

Total 9 NOR gates are required to implement a Full Adder.



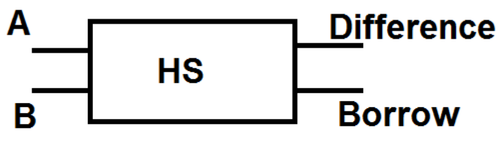
Application of Full Adder in Digital Logic

- **Arithmetic circuits:** Full adders are utilized in math circuits to add twofold numbers. At the point when different full adders are associated in a chain, they can add multi-bit paired numbers.
- **Data handling:** Full adders are utilized in information handling applications like advanced signal handling, information encryption, and mistake rectification.
- **Counters:** Full adders are utilized in counters to addition or decrement the count by one.
- **Multiplexers and demultiplexers:** Full adders are utilized in multiplexers and demultiplexers to choose and course information.

- **Memory tending to:** Full adders are utilized in memory addressing circuits to produce the location of a particular memory area.
- **ALUs:** Full adders are a fundamental part of Number juggling Rationale Units (ALUs) utilized in chip and computerized signal processors.

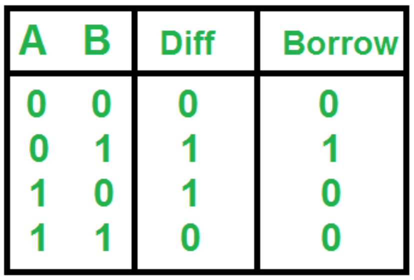
2.2. Half-Subtractor:

A half subtractor is a digital logic circuit that performs the binary subtraction of two single-bit binary numbers. It has two inputs, A and B, and two outputs, Difference and Borrow. The Difference output represents the result of subtracting B from A, while the Borrow output indicates whether a borrow is needed when A is smaller than B. The half subtractor can be implemented using basic logic gates such as XOR, AND, and NOT gates. It is a fundamental building block for more complex arithmetic circuits like full subtractors and multi-bit subtractors.



Half Subtractor

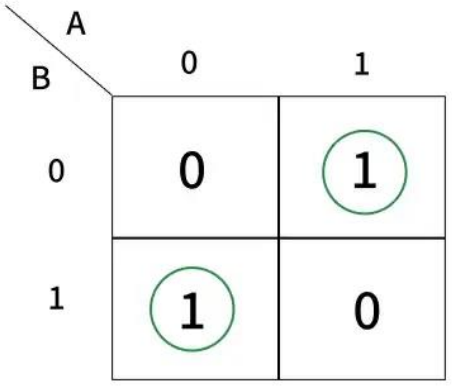
Truth Table of Half Subtractor



A	B	Diff	Borrow
0	0	0	0
0	1	1	1
1	0	1	0
1	1	0	0

Logical Expression of Half Subtractor

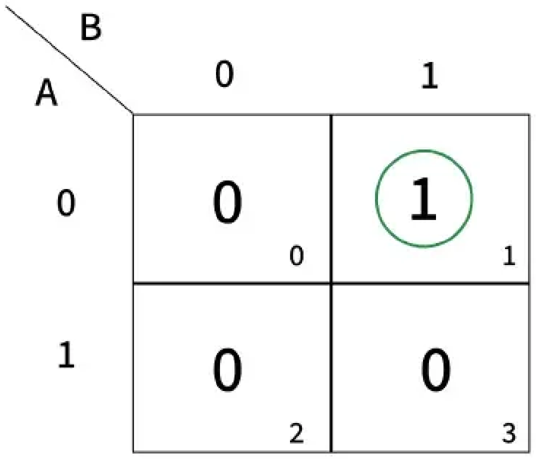
For difference,



The SOP form of the Difference is as follows:

Difference = A'B+AB'

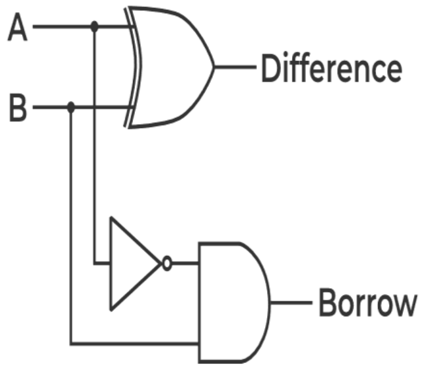
For borrow,



The SOP form of the Borrow is as follows:

Borrow = A'B

Implementation of Half Subtractor:



Application of Half Subtractor in Digital Logic:

- Calculators: Most mini-computers utilize advanced rationale circuits to perform numerical tasks. A Half Subtractor can be utilized in a number cruncher to deduct two parallel digits from one another.
- Alarm Systems: Many caution frameworks utilize computerized rationale circuits to identify and answer interlopers. A Half Subtractor can be utilized in these frameworks to look at the upsides of two parallel pieces and trigger a caution in the event that they are unique.
- Automotive Systems: Numerous advanced vehicles utilize computerized rationale circuits to control different capabilities, like the motor administration framework, stopping mechanism, and theatre setup. A Half Subtractor can be utilized in these frameworks to perform computations and examinations.
- Security Frameworks: Advanced rationale circuits are usually utilized in security frameworks to identify and answer dangers. A Half Subtractor can be utilized in these frameworks to look at two double qualities and trigger a caution in the event that they are unique.

- Computer Frameworks: Advanced rationale circuits are utilized broadly in PC frameworks to perform estimations and examinations. A Half Subtractor can be utilized in a PC framework to deduct two paired values from one another.

Full Subtractor:

A Full Subtractor is a combinational circuit used to perform binary subtraction. It has three inputs:

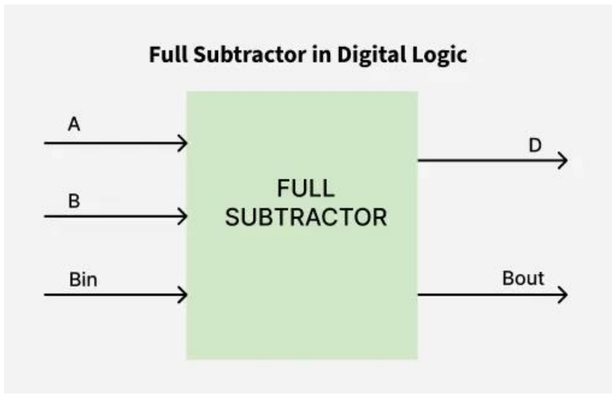
- A (Minuend)
- B (Subtrahend)
- B-IN (Borrow-in from the previous stage)

It produces two outputs:

- Difference (D): The result of the subtraction.
- Borrow-out (B-OUT): Indicates if a borrow is needed for the next stage.

The full subtractor is essential because a half-subtractor can only subtract the least significant bit (LSB) of binary numbers. However, if a borrow is generated during the subtraction of the LSBs, it will affect the subtraction in the next stages. A full subtractor handles this situation by considering the borrow from the previous stage, ensuring accurate subtraction even when a borrow is present.

The full subtractor is used to subtract binary numbers with borrow handling, making it suitable for multi-bit subtraction in digital circuits like Arithmetic Logic Units (ALUs).

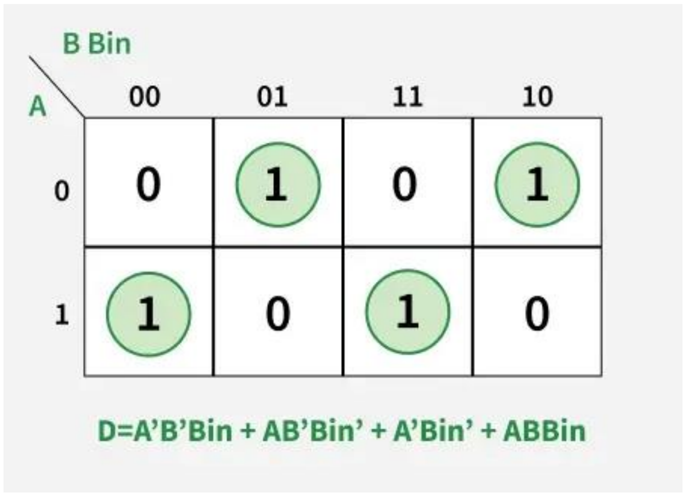


Truth Table of Full Subtractor

Input			Output	
A	B	Bin	D	Bout
0	0	0	0	0
0	0	1	1	1
0	1	0	1	1
0	1	1	0	1
1	0	0	1	0
1	0	1	0	0
1	1	0	0	0
1	1	1	1	1

K-Map for Full Subtractor:

From above table we can draw the K-Map as shown for "difference" and "borrow".



Logical expression for difference

The basic expression is:

D = A'B'Bin + A'BBin' + AB'Bin' + ABBin

Factoring common terms:

D = Bin(A'B' + AB) + Bin'(AB' + A'B)

Recognizing XOR and XNOR properties:

A'B' + AB = A XNOR B

AB' + A'B = A XOR B

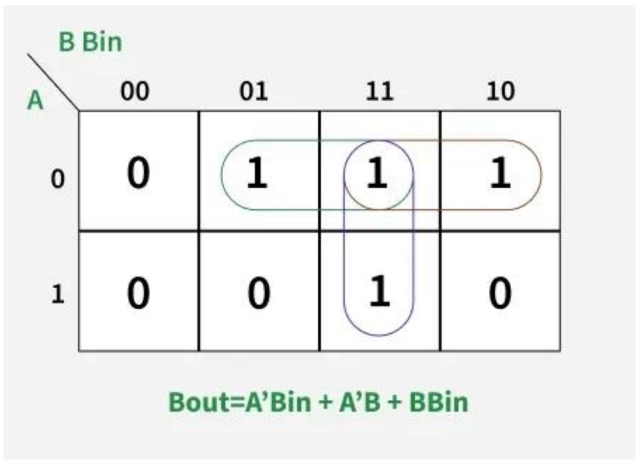
Substituting these values:

D = Bin(A XNOR B) + Bin' (A XOR B)

Using XNOR identity:

D = Bin ⊕ (A ⊕ B)

Thus, the final simplified expression for the difference in a full subtractor is: D = (A ⊕ B) ⊕ Bin



Logical expression for borrow

The **borrow (Bout) output** is derived as follows:

The basic expression:

Bout = A'B'Bin + A'BBin' + A'BBin + ABBin

Factoring common terms:

Bout = A'Bin(B + B') + A'B(Bin + Bin') + BBin(A + A')

Simplifying:

Bout = A'Bin + A'B + BBin

Alternatively, using another approach:

Bout = A'B'Bin + A'BBin' + A'BBin + ABBin

Factoring common terms:

Bout = Bin(AB + A'B') + A'B(Bin + Bin')

Using XOR and XNOR properties:

AB + A'B' = A XNOR B

Substituting these values:

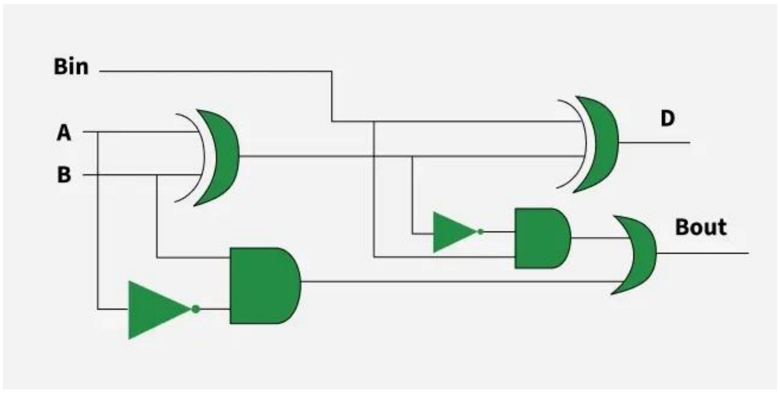
Bout = Bin(A XNOR B) + A'B

Using XNOR identity:

Bout = Bin (A XOR B)' + A'B

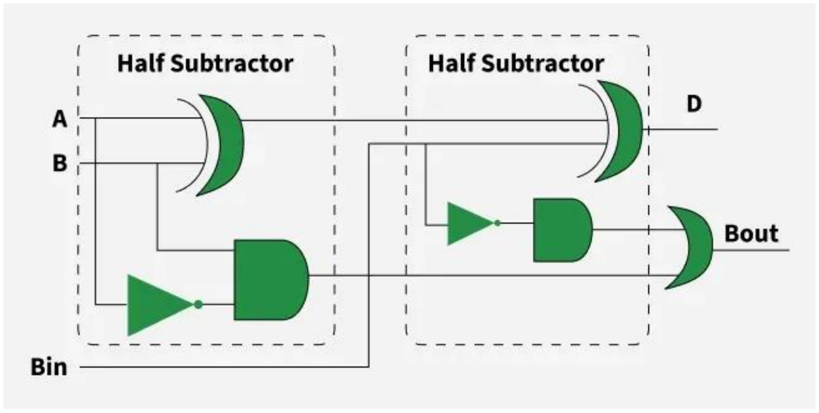
Thus, the final simplified expression for borrow in a full subtractor is:

Logic Circuit for Full Subtractor:



Implementation of Full Subtractor using Half Subtractors

2 Half Subtractors and an OR gate is required to implement a Full Subtractor.



2.3. Parallel binary adder:

Parallel adder is a binary adder circuit implemented to add two binary number having N-bits (for example, to add 4-bit binary numbers, we use 4- bit parallel adder, and so on). As its name implies, the parallel adder is a digital combinational circuit that adds two binary numbers in parallel form and generates the arithmetic sum of those binary numbers in parallel form.

As we already mentioned above that a full adder can perform addition of only two one-bit binary numbers consisting of two input bits and one input carry bit, i.e. addition of three bits. But in actual practice, we have to add such binary numbers whose length is more than one bit. To add such binary numbers, we use parallel binary adder which is capable of adding the two binary numbers of any bit length such as 4-bit, 5-bit, etc.

We can implement an N-bit parallel adder with the help of full-adders connected in a chain fashion. The block diagram representation of an N-bit parallel adder using full adders is shown in Figure-2.

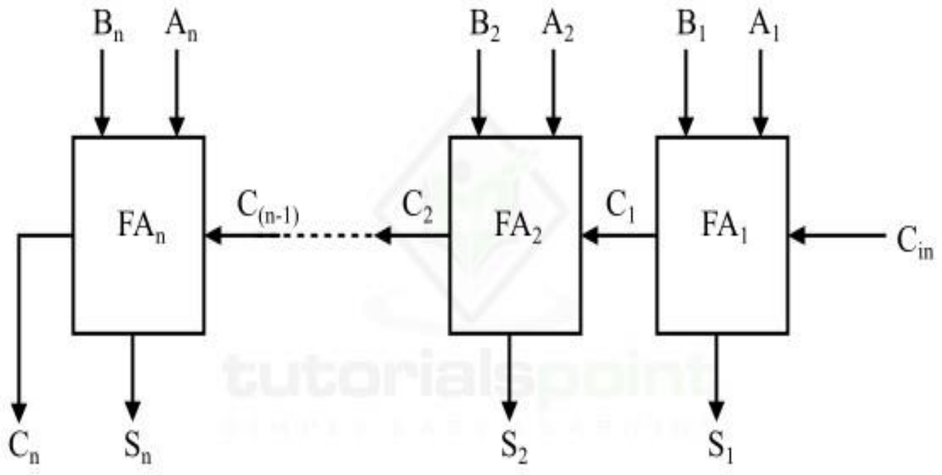


Figure 2 - N-Bit Parallel Adder

From the block diagram of the N-bit parallel adder, it can be seen that the carry output from each full-adder is connected to the carry input terminal of the next higher level full-adder in the chain.

The number of full-adder to realize a parallel adder is determined from the number of bits in the two binary numbers to be added. Therefore, an N-bit parallel adder requires N full-adders to perform the addition in parallel form. For example, a 2-bit parallel adder requires 2 full adders, 4-bit parallel adder consists of 4 full adders, and so on.

Operation of N-Bit Parallel Adder Circuit

The working of the N-bit parallel adder shown in figure-2 can be described in the following steps −

- Initially, the full adder FA_1 adds two input bits A_1 and B_1 along with an input carry bit C_{in}, and it generates the output sum bit S_1 and the carry bit C_1 which is forwarded to the next adder (FA_2) in the chain. The sum bit S_1 is the least significant bit of the output sum.
- At the next stage, the full adder circuit FA_2 becomes active and adds input bits A_2 and B_2 along with C_1. It generates the sum bit S_2 which is the second bit of the output sum, and the carry bit C_2 that is connected to the next full adder FA_3 in the chain.
- This process will continue till the last full adder, i.e. FAn in the chain. The full adder uses carry input $C_{(n-1)}$ to add with the input bits A_n and B_n to produce the last bit of the output sum S_n and the last output carry bit C_n.

Advantages of Parallel Adder:

Some important advantages of parallel adder are listed below −

- The parallel adder adds bits simultaneously.
- It makes addition of binary numbers fast.
- Parallel adder is more economical.

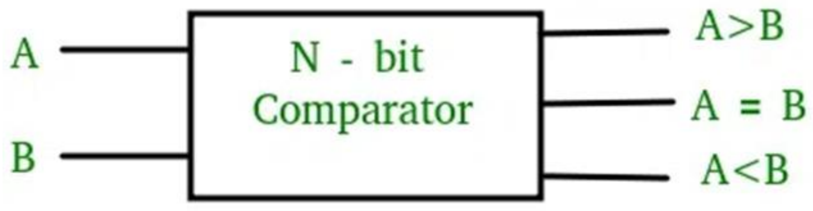
Applications of Parallel Adder:

The important applications of parallel adders are listed below −

- Parallel adders are used in arithmetic logic units that are used for heavy computing applications.
- Parallel adders are also used in parallel cellular automatic machines for parallel computing.
- Parallel adders are utilized for conversion of BCD into excess-1 code.
- Parallel adders are also used for the analysis of multiplication algorithms.

2.4. Magnitude comparator:

Magnitude comparator is a type of Combinational circuit, It Basically compares two binary numbers and determines their relative magnitude. It gives output whether one number is greater than the other, or less than or equal. These comparators are used in digital systems, such as for sorting networks, and decision-making circuits to handle numerical comparisons perfectly without any error.



The circuit works by comparing the bits of the two numbers starting from the most significant bit (MSB) and moving toward the least significant bit (LSB). At each bit position, the two corresponding bits of the numbers are compared. If the bit in the first number is greater than the corresponding bit in the second number, **the A>B output is set to 1**, and the circuit immediately determines that the first number is greater than the second. Similarly, if the bit in the second number is greater than the corresponding bit in the first number, **the A<B output is set to 1**, and the circuit immediately determines that the first number is less than the second.

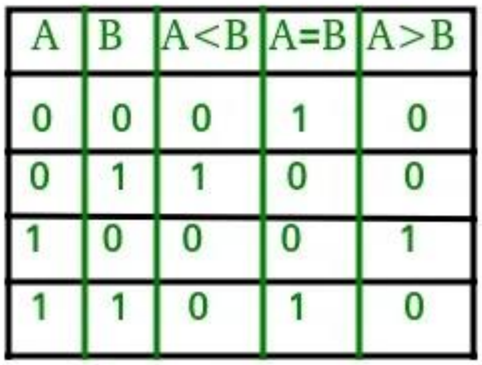
If the two corresponding bits are equal, the circuit moves to the next bit position and compares the next pair of bits. This process continues until all the bits have been compared. If at any point in the comparison, the circuit determines that the first number is greater or less than the second number, the comparison is terminated, and the appropriate output is generated. If all the bits are equal, the circuit generates an **A=B output**, indicating that the two numbers are equal.

There are different ways to implement a magnitude comparator, such as using a combination of XOR, AND, and OR gates, or by using a cascaded arrangement of full adders. The choice of implementation depends on factors such as speed, complexity, and power consumption.

1-Bit Magnitude Comparator

A comparator used to compare two bits is called a single-bit comparator. It consists of two inputs each for two single-bit numbers and three outputs to generate less than, equal to, and greater than between two binary numbers.

The truth table for a 1-bit comparator is given below.

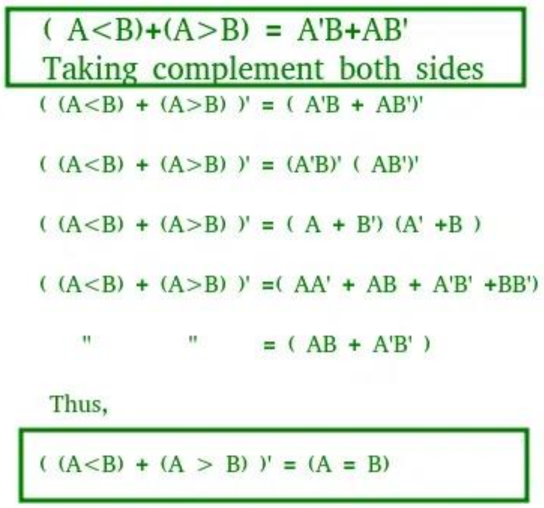


A	B	A<B	A=B	A>B
0	0	0	1	0
0	1	1	0	0
1	0	0	0	1
1	1	0	1	0

From the above truth table logical expressions for each output can be expressed as follows.

A>B: AB' A<B: A'B A=B: A'B' + AB

From the above expressions, we can derive the following formula.



$$\boxed{\begin{array}{l}(A<B)+(A>B) = A'B+AB' \\ \text{Taking complement both sides}\end{array}}$$

$((A<B) + (A>B))' = (A'B + AB')'$

$((A<B) + (A>B))' = (A'B)'(AB')'$

$((A<B) + (A>B))' = (A + B')(A' + B)$

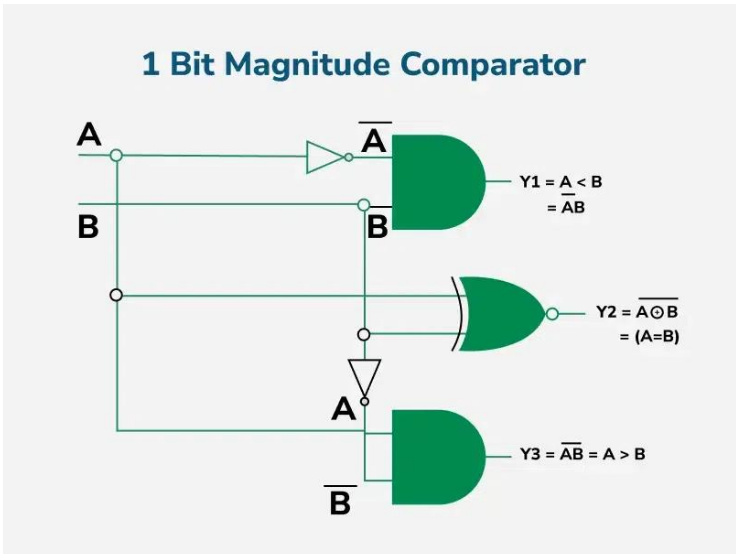
$((A<B) + (A>B))' = (AA' + AB + A'B' + BB')$

$\quad " \quad\quad " \quad = (AB + A'B')$

Thus,

$$\boxed{((A<B) + (A>B))' = (A = B)}$$

By using these Boolean expressions, we can implement a logic circuit for this comparator as given below.


1 Bit Magnitude Comparator
A
B
$\overline{A}$
B
$Y1 = A < B = \overline{A}B$
$Y2 = \overline{A \odot B} = (A=B)$
A
$\overline{B}$
$Y3 = A\overline{B} = A > B$

2-Bit Magnitude Comparator:

A comparator used to compare two binary numbers each of two bits is called a 2-bit Magnitude comparator. It consists of four inputs and three outputs to generate less than, equal to, and greater than between two binary numbers.

The truth table for a 2-bit comparator is given below.

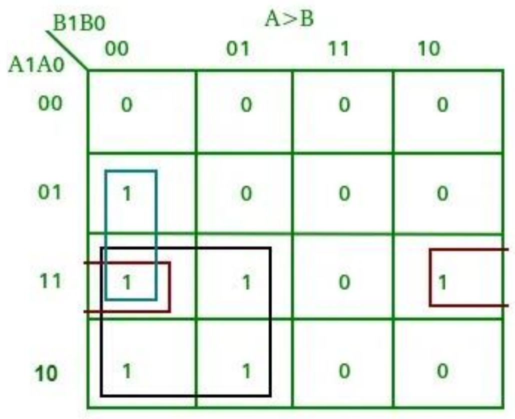
INPUT				OUTPUT		
A1	A0	B1	B0	A<B	A=B	A>B
0	0	0	0	0	1	0
0	0	0	1	1	0	0

INPUT				**OUTPUT**		
0	0	1	0	1	0	0
0	0	1	1	1	0	0
0	1	0	0	0	0	1
0	1	0	1	0	1	0
0	1	1	0	1	0	0
0	1	1	1	1	0	0
1	0	0	0	0	0	1
1	0	0	1	0	0	1
1	0	1	0	0	1	0
1	0	1	1	1	0	0
1	1	0	0	0	0	1

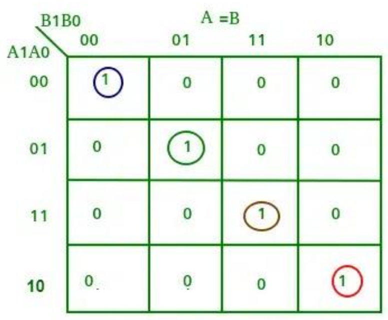
INPUT				OUTPUT		
1	1	0	1	0	0	1
1	1	1	0	0	0	1
1	1	1	1	0	1	0

From the above truth table, K-map for each output can be drawn as follows.

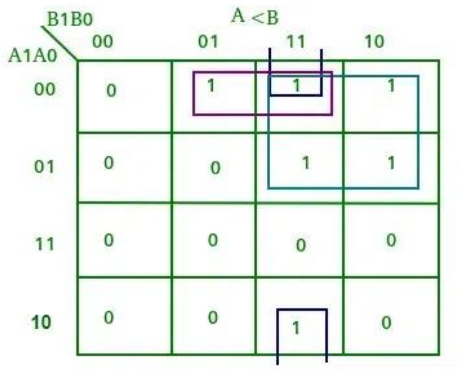
Truth Table of Output A>B



Truth Table of Output A=B



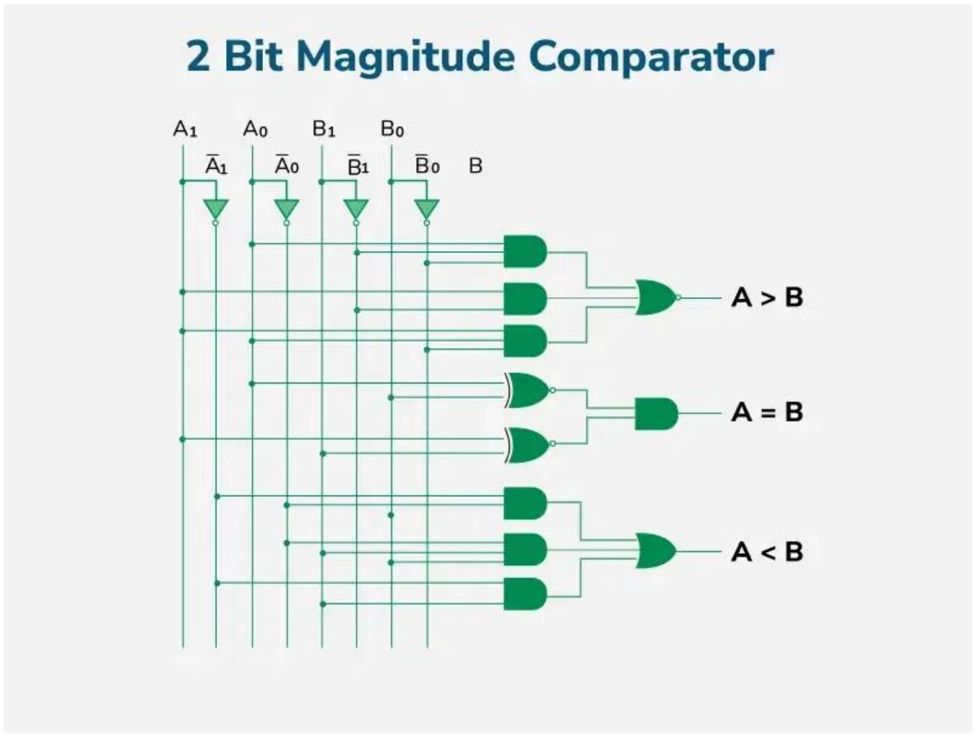
Truth Table of Output A<B



From the above K-maps logical expressions for each output can be expressed as follows.

A>B:A1B1’ + A0B1’B0’ + A1A0B0’A=B: A1’A0’B1’B0’ + A1’A0B1’B0 + A1A0B1B0 + A1A0’B1B0’ : A1’B1’ (A0’B0’ + A0B0) + A1B1 (A0B0 + A0’B0’) : (A0B0 + A0’B0’) (A1B1 + A1’B1’) : (A0 Ex-Nor B0) (A1 Ex-Nor B1)A<B:A1’B1 + A0’B1B0 + A1’A0’B0

By using these Boolean expressions, we can implement a logic circuit for this comparator as given below.



4-Bit Magnitude Comparator:

A comparator used to compare two binary numbers each of four bits is called a 4-bit magnitude comparator. It consists of eight inputs each for two four-bit numbers and three outputs to generate less than, equal to, and greater than between two binary numbers.
In a 4-bit comparator, the condition of A>B can be possible in the following four cases.

1. If A3 = 1 and B3 = 0
2. If A3 = B3 and A2 = 1 and B2 = 0
3. If A3 = B3, A2 = B2 and A1 = 1 and B1 = 0
4. If A3 = B3, A2 = B2, A1 = B1 and A0 = 1 and B0 = 0

Similarly, the condition for A<B can be possible in the following four cases.

1. If A3 = 0 and B3 = 1
2. If A3 = B3 and A2 = 0 and B2 = 1
3. If A3 = B3, A2 = B2 and A1 = 0 and B1 = 1
4. If A3 = B3, A2 = B2, A1 = B1 and A0 = 0 and B0 = 1

The condition of A=B is possible only when all the individual bits of one number exactly coincide with the corresponding bits of another number.

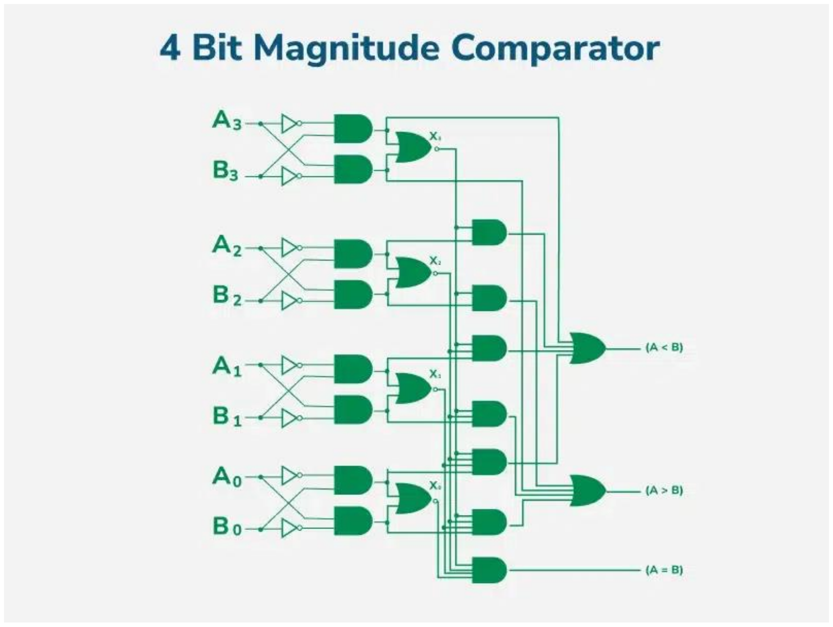
From the above statements, logical expressions for each output can be expressed as follows.

AA, 831331 r: (A3 Ex-Nor 33)A2132' a (A3 Ex-Nor 133) (A2 Ex-Nor 132)A131' a (A3 Ex-Nor 33) (A2 ENor132) (Al Ex-Nor 31)A01301

,13: A3'03 a (A3 Ex-Nor 33)A211:12 a (A3 Ex-Nor 83) (A2 Ex-Nor 132)Ar131 a (A3 Ex-Nor 33) (A2 Ex-Nor32) (Al Ex-Nor 131)A0N30

A=B: (A3 Ex-Nor B3) (A2 Ex-Nor 82) (Al Ex-Nor BI) (AO Ex-Nor BO)

By using these Boolean expressions, we can implement a logic circuit for this comparator as given below.



Applications of Comparators:

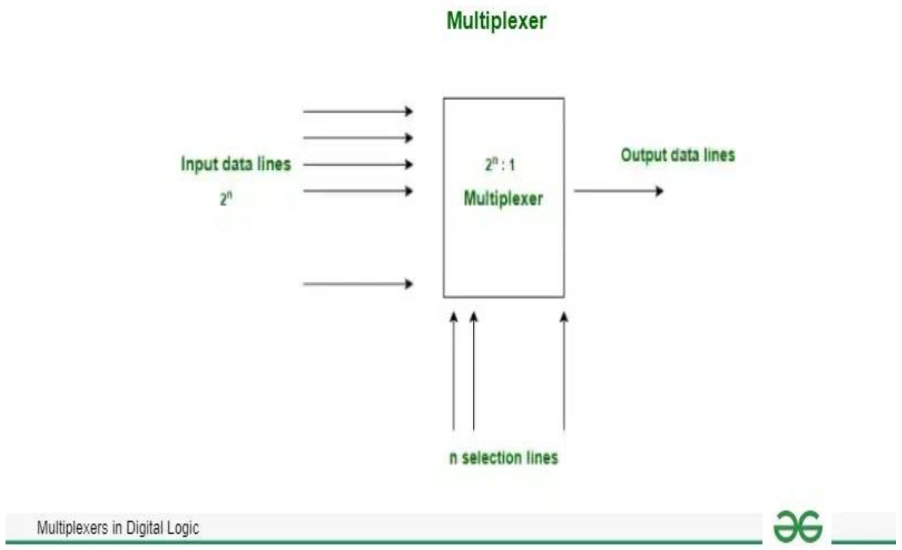
- Comparators are used in central processing units (CPU s) and microcontrollers (MCUs).
- These are used in control applications in which the binary numbers representing physical variables such as temperature, position, etc. are compared with a reference value.
- Comparators are also used as process controllers and for Servo motor control.
- Used in password verification and biometric applications.

2.5. Multiplexers (4:1) & demultiplexers (1:4):

Multiplexers:

A multiplexer is a combinational circuit that has many data inputs and a single output, depending on control or select inputs. For N input lines, log2(N) selection lines are required, or equivalently, for 2n input lines, n selection lines are needed.

- Multiplexers are also known as "N-to-1 selectors," parallel-to-serial converters, many-to-one circuits, and universal logic circuits.
- They are mainly used to increase the amount of data that can be sent over a network within a certain amount of time and bandwidth.

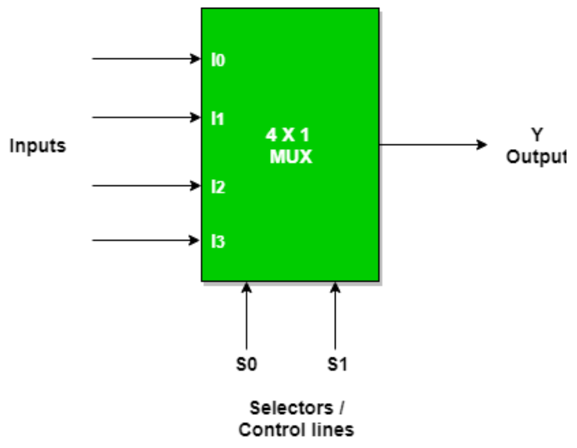


4×1 Multiplexer

The 4x1 Multiplexer which is also known as the 4-to-1 multiplexer. It is a multiplexer that has 4 inputs and a single output. The Output is selected as one of the 4 inputs which is based on the selection inputs. The number of the Selection lines will depend on the number of the input which is determined by the equation $log_2 n$,In 4x1 Mux the selection lines can be determined as $log_4 = 2$,slo two selections are needed.

Block Diagram of 4×1 Multiplexer

In the Given Block Diagram I0, I1, I2, and I3 are the 4 inputs and Y is the Single output which is based on Select lines S0 and S1.

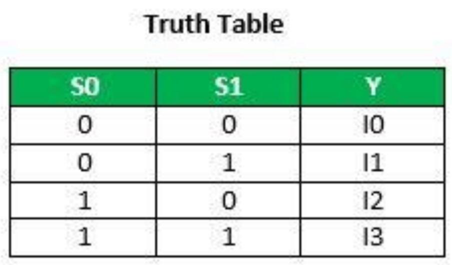


The output of the multiplexer is determined by the binary value of the selection lines

- When S1S0=00, the input I0 is selected.
- When S1S0=01, the input I1 is selected.
- When S1S0=10, the input I2 is selected.
- When S1S0=11, the input I3 is selected.

Truth Table of 4×1 Multiplexer

Given Below is the Truth Table of 4x1 Multiplexer



Truth Table

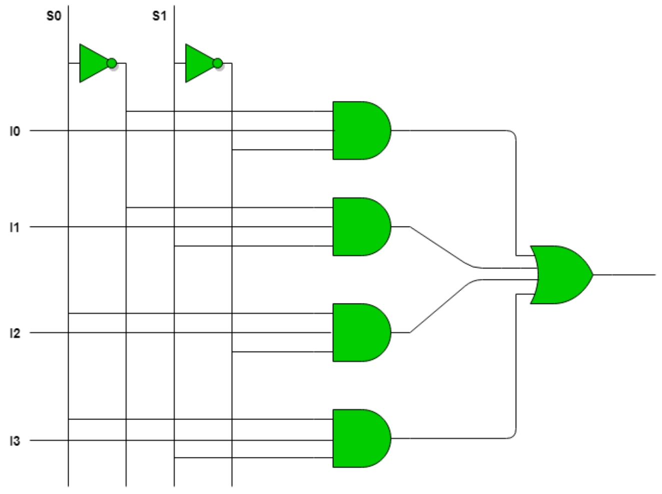
S0	S1	Y
0	0	I0
0	1	I1
1	0	I2
1	1	I3

So, final equation,

$$Y = S0'.S1'.I0 + S0'.S1.I1 + S0.S1'.I2 + S0.S1.I3$$

Circuit Diagram of 4x1 Multiplexers

Using truth table the circuit diagram can be given as

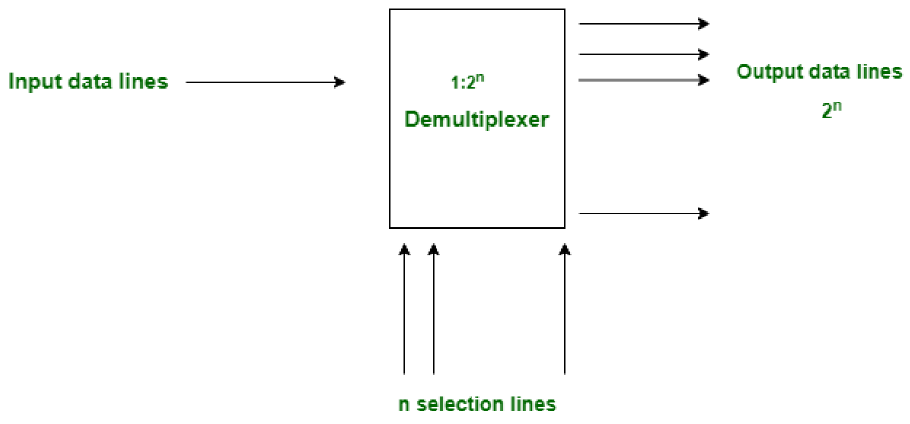


Multiplexer can act as universal combinational circuit. All the standard logic gates can be implemented with multiplexers.

Demultiplexer (DEMUX):

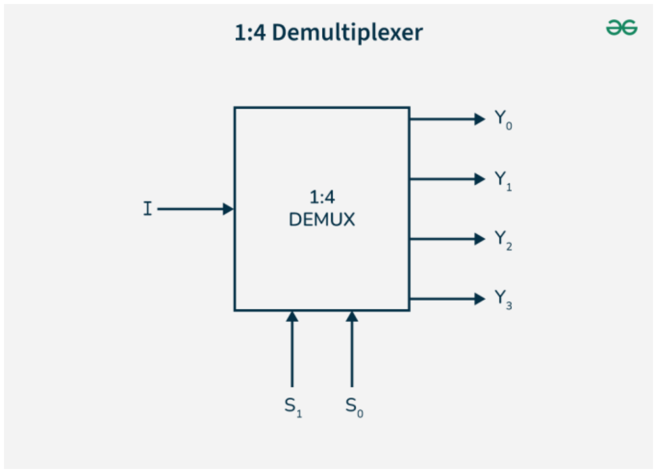
The DEMUX is a digital information processor. It takes input from one source and also converts the data to transmit towards various sources. The demultiplexer has one data input line. The demultiplexer has several control lines (also known as select lines). These lines determine to which output the input data should be sent. The number of control lines determines the number of output lines.

Here is the basic block diagram of a DEMUX as mentioned below.



Truth Table Of A 1X4 DEMUX

A 1x4 DEMUX has only one input which is denoted as I. There are two selection lines i.e. S1 and S0. At last, the DEMUX has output lines including Y3, Y2, Y1 &Y0. Here is the 1x4 DEMUX with diagram as mentioned below.



Now let us discuss the truth table of the 1x4 DEMUX as mentioned below.

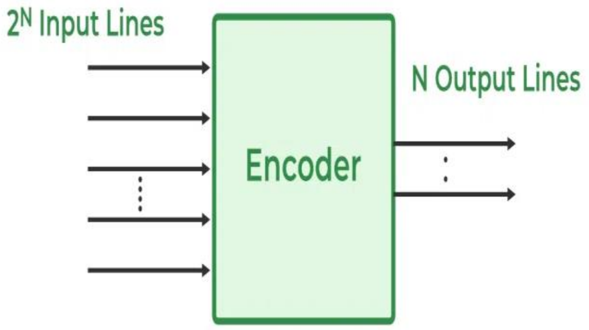
Inputs		**Outputs**			
S1	**S0**	**Y3**	**Y2**	**Y1**	**Y0**
0	0	0	0	0	**I**
0	1	0	0	**I**	0
1	0	0	**I**	0	0
1	1	**I**	0	0	0

2.6. Encoder (8-line to 3-line) and decoder (3-line to 8-line):

Encoder (8-line to 3-line):

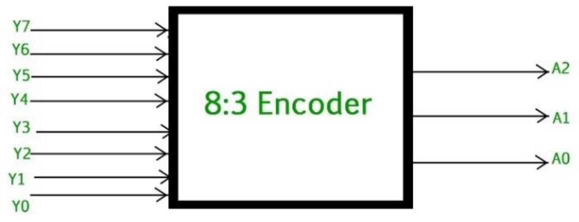
An encoder is a digital combinational circuit that converts multiple input signals into a binary code. It typically has one active input at a time and generates a binary output representing the position of that active input. For example, a 4-to-2 encoder has four inputs and produces a 2-bit output code. Encoders are useful in digital systems for data compression and efficient transmission. Variants like priority encoders assign importance to inputs to manage simultaneous signals.

- The number of inputs is usually 2n, with n output lines.
- Inputs are usually active high, meaning high voltage represents an active signal.
- Binary-weighted encoders assign codes based on input position using binary values.
- Encoders help convert parallel inputs into compact binary formats for processing.



Octal to Binary Encoder (8 to 3 Encoder):

The 8 to 3 Encoder or octal to Binary encoder consists of 8 inputs: Y7 to Y0 and 3 outputs: A2, A1 & A0. Each input line corresponds to each octal digit value and three outputs generate corresponding binary code. The figure below shows the logic symbol of octal to the binary encoder.



Truth Table

The truth table for the 8 to 3 encoder is as follows:

INPUTS(Y7Y6Y5Y4Y3Y2Y1Y0)	**OUTPUTS(A2A1A0)**
00000001	000
00000010	001
00000100	010
00001000	011
00010000	100
00100000	101
01000000	110
10000000	111

Logical expression for A2, A1, and A0:

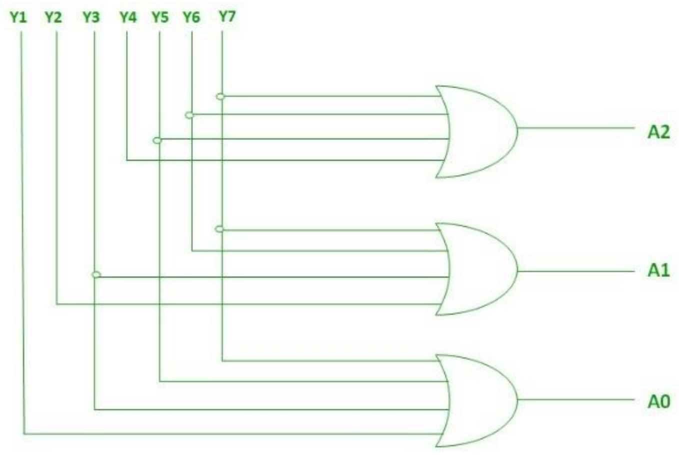
$A2 = Y7 + Y6 + Y5 + Y4$

$A1 = Y7 + Y6 + Y3 + Y2$

$A0 = Y7 + Y5 + Y3 + Y1$

Circuit Diagram

The above two Boolean functions A2, A1, and A0 can be implemented using four input OR gates:



Decoder (3-line to 8-line):

A device used for the conversion of binary into decimal. It is a combinational logic circuit that receives the n input lines and generates a maximum of 2^n unique output lines. The output might be less than 2^n lines. This condition will occur when we are not using it up to its maximum capacity.

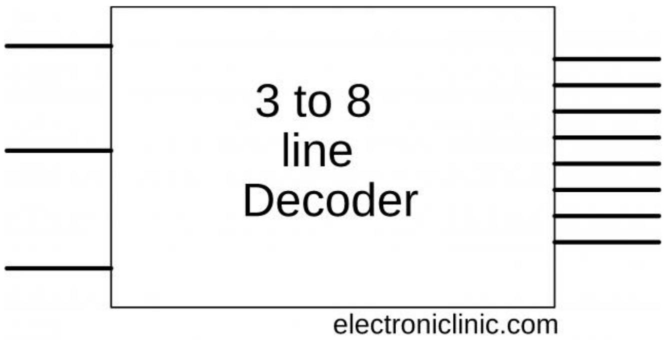
There are several types of binary decoders, but in all cases, a decoder is an electronic circuit with multiple inputs and multiple output signals, which converts every unique combination of input states to a specific combination of output states. In addition to integer data inputs, some decoders also have one or more "enable" inputs. When the enable input is negated (disabled), all decoder outputs are forced to their inactive states.

Depending on its function, a binary decoder will convert binary information from n input signals to as many as 2n unique output signals. Some decoders have less than 2n output lines; in such cases, at least one output pattern may be repeated for different input values.

Why do we need a Decoder?

A decoder is a circuit that changes a code into a set of signals. It is called a decoder because it does the reverse of encoding, but we will begin our study of encoders and decoders with decoders because they are simpler to design.

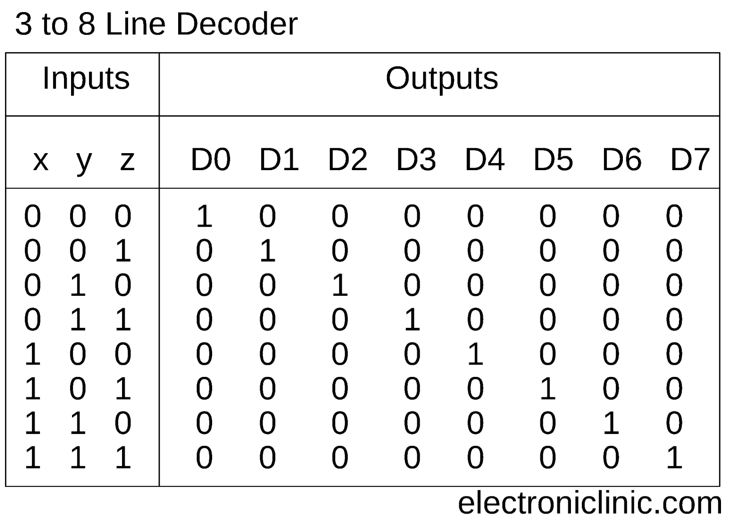
One of the most frequently asked questions, what is the main difference between demultiplexer and decoder is that a demultiplexer is a combinational circuit that accepts only one input and directs it into one of the several outputs. On the contrary, the decoder is a combinational circuit that can accept many inputs and generate the decoded output.



3 to 8 Decoder designing steps

- Problem: 3-to-8-line decoder.
- The number of available inputs are 3 and outputs are 8.
- Let us represent the inputs and outputs by symbol letters. Let us represent the inputs by x, y, and z; and the outputs by D_0, D_1, D_2, … D_7.

3 to 8 Decoder Truth Table:

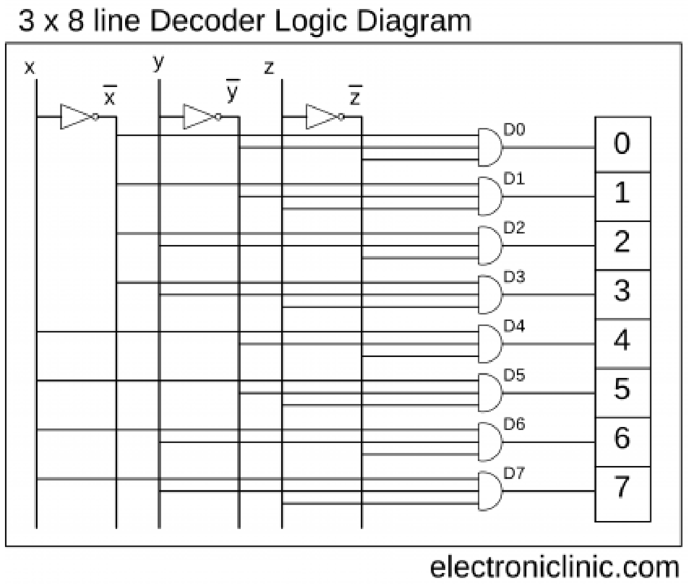


3 to 8 Line Decoder

Inputs			Outputs							
x	y	z	D0	D1	D2	D3	D4	D5	D6	D7
0	0	0	1	0	0	0	0	0	0	0
0	0	1	0	1	0	0	0	0	0	0
0	1	0	0	0	1	0	0	0	0	0
0	1	1	0	0	0	1	0	0	0	0
1	0	0	0	0	0	0	1	0	0	0
1	0	1	0	0	0	0	0	1	0	0
1	1	0	0	0	0	0	0	0	1	0
1	1	1	0	0	0	0	0	0	0	1

electroniclinic.com

The logical diagram of the 3×8 line decoder is given below.



3-to-8-line Decoder has a memory of 8 stages. **It** is convenient to **use** an AND **gate** as the basic **decoding** element for the output because **it** produces a "HIGH" or logic "1" output only when all of its inputs are logic "1". You can clearly see the logic diagram is developed using the AND gates and the NOT gates. The Inputs are represented by x, y, and z while the compliments are represented with the bars over the letters. The AND gates are represented by D0, D1, D2, D3, D4, D5, D6, and D7. The corresponding inputs are connected with the AND gates as per the boolean functions given above.

2.8. BCD to seven segment decoder:

Binary Coded Decimal (BCD)

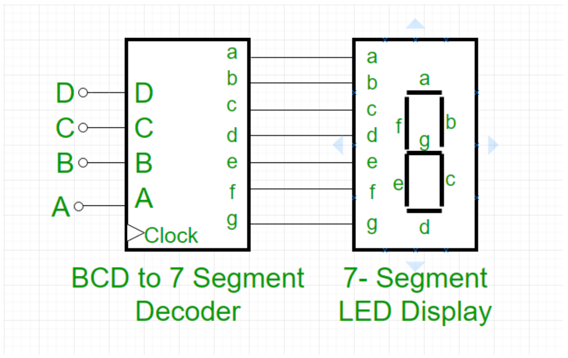
BCD is the encoding scheme each of the decimal numbers (0-9) is represented by its equivalent binary pattern (which is generally of 4-bits).

Seven segments:

Seven Segment display is an electronic device which consists of seven Light Emitting Diodes (LEDs) arranged in some definite pattern (common cathode or common anode type), which is used to display Hexadecimal numerals (in this case decimal numbers, as input is BCD i.e., 0-9). Two types of seven segment LED display:

1. **Common Cathode Type:** In this type of display all cathodes of the seven LEDs are connected together to the ground or -Vcc (hence, common cathode) and LED displays digits when some 'HIGH' signal is supplied to the individual anodes.
2. **Common Anode Type:** In this type of display all the anodes of the seven LEDs are connected to battery or +Vcc and LED displays digits when some 'LOW' signal is supplied to the individual cathodes.

But, seven segment display does not work by directly supplying voltage to different segments of LEDs. First, our decimal number is changed to its BCD equivalent signal then BCD to seven segment decoder converts that signals to the form which is fed to seven segment display. This BCD to seven segment decoders has four input lines (A, B, C and D) and 7 output lines (a, b, c, d, e, f and g), this output is given to seven segment LED display which displays the decimal number depending upon inputs.



Truth Table

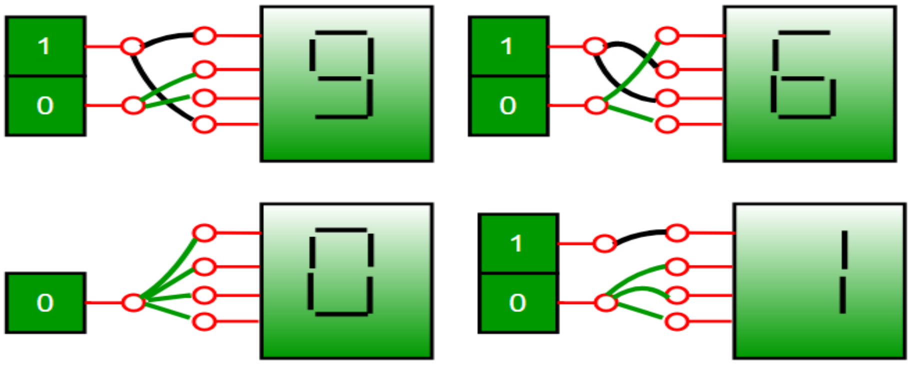
For common cathode type BCD to seven segment decoder:

A	B	C	D	a	b	c	d	e	f	g
0	0	0	0	1	1	1	1	1	1	0
0	0	0	1	0	1	1	0	0	0	0
0	0	1	0	1	1	0	1	1	0	1
0	0	1	1	1	1	1	1	0	0	1
0	1	0	0	0	1	1	0	0	1	1
0	1	0	1	1	0	1	1	0	1	1
0	1	1	0	1	0	1	1	1	1	1
0	1	1	1	1	1	1	0	0	0	0
1	0	0	0	1	1	1	1	1	1	1
1	0	0	1	1	1	1	1	0	1	1

Note -

- For Common Anode type seven segment LED display, we only have to interchange all '0s' and '1s' in the output side i.e., (for a, b, c, d, e, f, and g replace all '1' by '0' and vice versa) and solve using K-map.
- Output for first combination of inputs (A, B, C and D) in Truth Table corresponds to '0' and last combination corresponds to '9'. Similarly rest corresponds from 2 to 8 from top to bottom.
- BCD numbers only range from 0 to 9, thus rest inputs from 10-F are invalid inputs.

Example -



Explanation -

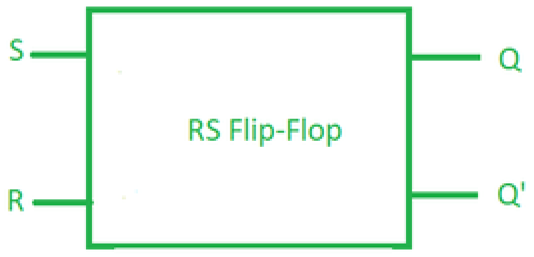
For combination where all the inputs (A, B, C and D) are zero (see Truth Table), our output lines are a = 1, b = 1, c = 1, d = 1, e = 1, f = 1 and g = 0. So, 7 segment display shows 'zero' as output.

Similarly, for combination where one of the inputs is one (D = 1) and rest are zero, our output lines are a = 0, b = 1, c = 1, d = 0, e = 0, f = 0 and g = 0. So only LEDs 'b' and 'c' (see diagram above) will glow and 7 segment display shows 'one' as output.

UNIT-III: FLIP-FLOPS: R-S Flip-flop, J-K Flip-Flop, T and D type flip-flops, master-slave flip-flop, truth tables, registers: serial in serial out and parallel in and parallel out – counters asynchronous: mod-8, mod-10, synchronous – ring counter and up-down counter – A/D and D/A converter.

3.1. R-S Flip-flop:

The RS flip-flop is used to store binary information (i.e. 0 or 1). It consists of two inputs, SET and RESET. In RS flip-flop 'R' Stands for RESET and 'S' stands for SET. The flip-flop keeps its present state even when one or both inputs are deactivated. The flip-flop enters the '0' state when the RESET input is activated, and the '1' state when the SET input is activated.



RS Flip-Flop block diagram

The block diagram of the RS flip-flop is shown above. Since RS flip-flops are used for storing binary information it is used in simple digital applications like data registers and memory cells that require binary storage. Nevertheless, more sophisticated flip-flop designs, such as the D flip-flop or JK flip-flop, are frequently used over the RS flip-flop for their increased reliability and versatility in complex digital systems due to restrictions in handling specific input conditions.

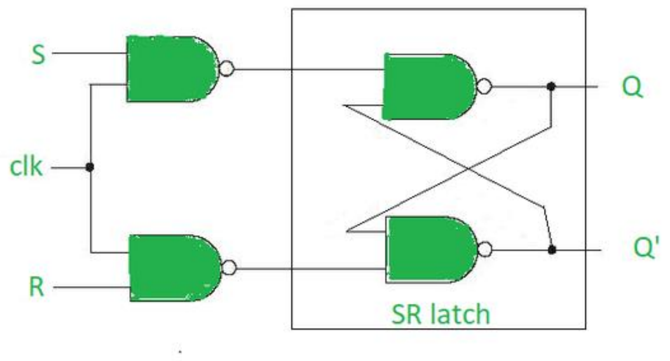
Construction and Working of RS flip flop:

RS flip can be constructed using basic logic gates such as NAND gates or NOR gates. Below shown a RS flip-flop constructed using a NAND gate same we can construct a RS flip flop using a NOR gate.

Working of RS flip-flop depends on its inputs.

- In initial state output can in any state SET or RESET.
- In SET, S=1 and R=0.
- Conversely in RESET, S=0 and R=1.
- When R=0 and S=0 it holds the current state
- When R=1 and S=1 it falls under Undefined / Forbidden state

RS flip-flop Using NAND gates:



RS flip-flop

The RS flip-flop can be constructed using 4 two input NAND gate which is shown in the above figure.

Truth Table for RS Flip-flop

Sl no.	Clock	S	R	Qn+1
1	0	X	X	Qn
2	1	0	0	No change

Sl no.	Clock	S	R	Qn+1
3	1	0	1	0
4	1	1	0	1
5	1	1	1	Undefined / Forbidden state

Here, S is the Set input, R is the reset input, Qn+1 is the next state and State tells in which state it enters

Characteristic Table For RS Flip-flop:

The characteristic equation tells us about what will be the next state of flip flop in terms of present state.

Sl no.	Qn	S	R	Qn+1
1	0	0	0	0
2	0	0	1	0
3	0	1	0	1
4	0	1	1	X

Sl no.	Qn	S	R	Qn+1
5	1	0	0	1
6	1	0	1	0
7	1	1	0	1
8	1	1	1	X

Characteristic Equation: Qn+1 = S + Qn. R'

Here, **S** is the Set input, **R** is the reset input, **Qn** is the current state input and **Qn+1** is the next state outputs.

Excitation table For RS Flip-flop:

Excitation Table basically tells about the excitation which is required by flip flop to go from current state to next state.

Sl no.	Qn	Qn+1	S	R
1	0	0	0	X
2	0	1	1	0

Sl no.	Qn	Qn+1	S	R
3	1	0	0	1
4	1	1	X	0

Here, **Qn** is the current state, **Qn+1** is the next state outputs and **S**, **R** are the set and reset inputs respectively.

Difference between SR and RS Flip-Flop:

Both SR and RS flipflop are electronic devices used for storing the binary information (i.e. 0 or Both of them are nearly same with more or less same functionality, but differ in their inputs. SR ('S' stand for SET and 'R' stand for RESET) flip flop has inputs SET and RESET. The flip-flop's output changes depending on the inputs, with '1' triggering the SET state, '1' triggering the RESET state, and '0' indicating the current state. Here SET has high priority when both the inputs are '1'. RS ('R' stand for RESET and 'S' stand for SET) flip flop has inputs RESET and SET. The flip-flop maintains its current state when both inputs are '0', as it enters the RESET state when the R input is set to '1'. Here RESET has high priority when both the inputs are '1'. In conclusion to difference between them we can say that both of them are nearly same with more or less same applications and functionality but differ in their input types.

Applications of RS Flip-flop

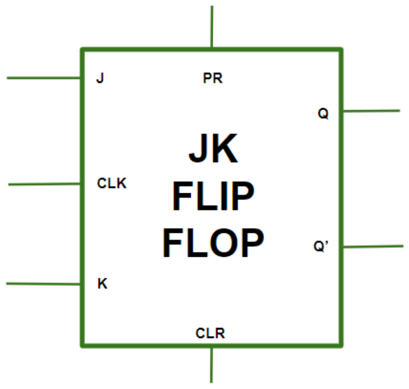
RS flipflop are data storage device used to store binary information. It is used is mainly devices requires binary information. It’s used in:

1. Asynchronous Counters (RS flipflops are used in building the asynchronous counters).
2. Shift Registers (RS flip-flops can be used to build shift registers).
3. State Machines.

4. Debouncing Circuit (used in stabilizing output of a switch/button).
5. Address Decoding.
6. Clock Synchronization.

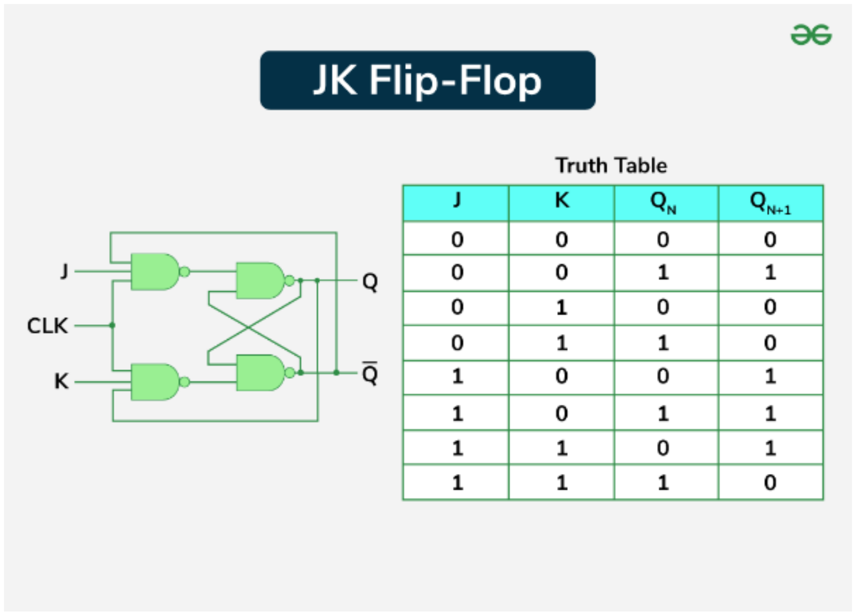
3.2. J-K Flip-Flop:

It is one kind of sequential logic circuit which stores binary information in bitwise manner. It consists of two inputs and two outputs. Inputs are Set(J) & Reset(K) and their corresponding outputs are Q and Q'. JK flipflop has two modes of operation which are synchronous mode and asynchronous mode. In synchronous mode, the state will be changed with the clock(clk) signal, and in asynchronous mode, the change of state is independent from its clock signal. Let's see its diagram structure.



The JK flip flop diagram above represents the basic structure which consists of Clock (CLK), Clear (CLR), and Preset (PR).

Below is the circuit diagram of JK Flip Flop. Two 3-input NAND gates are used in place of the original two 2-input AND gates. The outputs at Q and Q' are coupled to each gate's third input. Since the two inputs are now interlocked, the SR flip-flop's cross-coupling enables the previously invalid condition of (S = "1", R = "1") to be employed to perform the "toggle action".



Truth Table

J	K	Q_N	Q_{N+1}
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	0

In a circuit "set", the bottom NAND gate interrupts the J input coming from the "0" position of Q'. In the "RESET" state, the top NAND gate interrupts the K input coming from the 0 positions of Q. We can use Q and Q' to control the input because they are always different. The flip flop is toggled according to the truth table when both inputs "J" and "K" are set to 1.

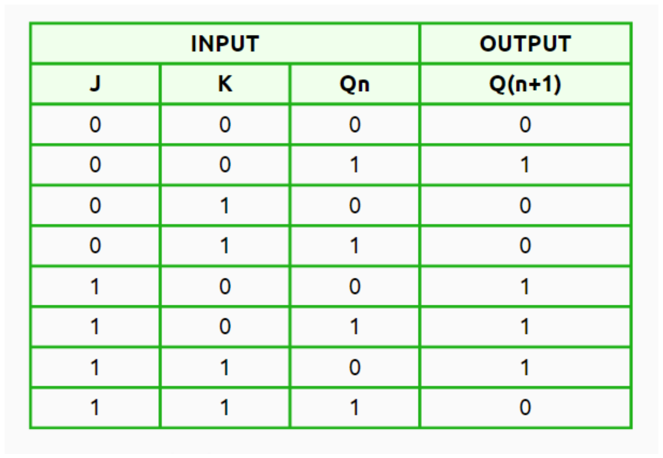
Truth Table of JK Flip Flop

Inputs					**Outputs**		
PR	CLR	CLK	J	K	Q(n+1)	Q'(n+1)	**Comments**
0	1	NA	NA	NA	1	0	Set (Preset)
1	0	NA	NA	NA	0	1	Reset (Clear)

Inputs					**Outputs**		**Comments**
1	1	0	NA	NA	Q(n)	Q'(n)	Initial Stage
1	1	1	0	0	Q(n)	Q'(n)	Initial Stage
1	1	1	1	0	1	0	Set
1	1	1	0	1	0	1	Reset
1	1	1	1	1	Q'(n)	Q(n)	Toggle

Characteristic Table

A JK flip-flop is a kind of sequential logic circuit that keeps track of binary data. Its characteristic table shows how the output (Qn+1) changes using inputs (J & K) along with the last state (Qn).



INPUT			OUTPUT
J	K	Qn	Q(n+1)
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	0

Let's break down the characteristic table. There are four states to understand:

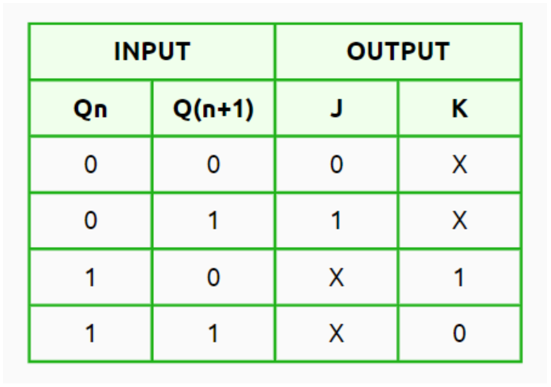
1. J = 0, K = 0 (No Change): Here, the output Q(n+1) stays the same. This means the next state (Q(n+1)) is just like the current one (Qn).
2. J = 0, K = 1 (Reset State): In this case, the next state is reset to 0 (Q(n+1) = 0), no matter what the current state is (Qn).
3. J = 1, K = 0 (Set State): Now, the next state gets set to 1 (Q(n+1) = 1), again, no matter what's happening in the current state (Qn).
4. J = 1, K = 1 (Toggle State): In this state, the output Q(n+1) toggles. So if the current state is set (Qn = 1), it will change to 0 (Q(n+1) = 0). If it's reset (Qn = 0), then it flips to 1 (Q(n+1) = 1).

Excitation Table:

The excitation table shows us which input combinations we should use for a JK flip-flop to get the output we want.

- X - Don't Care
- Qn - Current State
- Q(n+1) - Next State
- J and K - Two input values

Here, X for Don't Care. This means it can be either 0 or 1, & it won't change how the flip-flop works.



INPUT		OUTPUT	
Qn	Q(n+1)	J	K
0	0	0	X
0	1	1	X
1	0	X	1
1	1	X	0

This table guides us on what input values to use so we can move from the current (Qn) to the next state (Q(n+1)).

Let’s look at the transitions:

1. From **0 to 0**: When the current state is 0 (Qn = 0) and you want the next one to stay 0 (Q(n+1) = 0), then set J = 0 & K can be either 0 or 1, which is shown as X (Don’t Care).
2. From **0 to 1**: If the current state is 0 (Qn = 0) but you want the next state to flip to 1 (Q(n+1) = 1), then J should be set to 1 & K = X (doesn't matter for K).
3. From **1 to 0**: When you start with a current state of 1 (Qn = 1) & wish for it to switch to 0 (Q(n+1) = 0), then J should be X and K must be set at 1.
4. From **1 to 1:** If you're at a current state of 1 (Qn = 1) & want it to stay at 1 for the next state (Q(n+1) = 1), then just set J = X and K should be set at 0.

This table is used a lot in circuit design. It helps show the inputs needed to get those desired output transitions.

Applications of JK Flip-Flop:

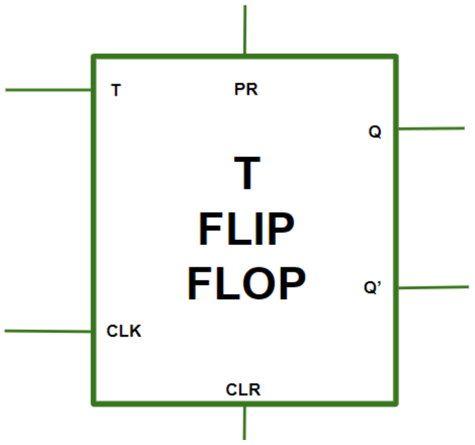
- Counters: Counters are very essential components for the application of frequency dividers and event sequencers where there is a need of storing and propagating the count value. We can design binary synchronous and asynchronous counters using JK-flipflop.
- Shift Registers: For data storage and manipulation, serial-to-parallel or parallel-to-serial data conversion the shift registers are widely used. Registers can store and shift the binary data in a sequential manner. We can design it by JK-flipflops.
- Memory Units: JK-flipflop itself act as a memory unit to store binary information. By making a sequential chain of JK-flipflops we can use it even as RAM.

3.3. T and D type flip-flops:

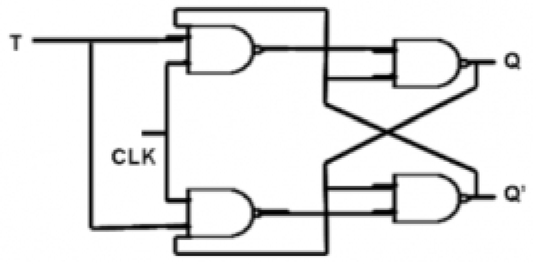
T Flip Flop:

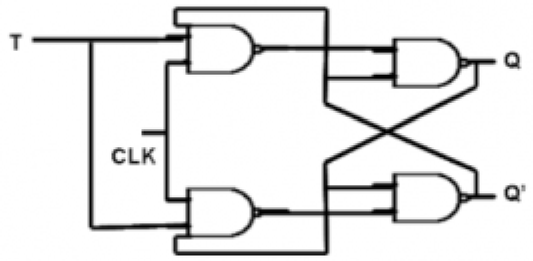
The T Flip Flop consists of data input (T), a clock input (CLK), and two outputs: Q and Q' (the complement of Q).

Block Diagram of T Flip Flop



Circuit Diagram and Truth Table of T Flip Flop





T	Q	Q(t+1)
0	0	0
1	0	1
0	1	1
1	1	0

Operation of the T Flip-Flop

- Case 1 (T=0): In this condition the flip-flop remains in its current state regardless of clock input, Also the Output Q will remain unchanged unit the value of T will not change.
- Case 2 (T=1): In this condition the flip flop will change when T input is 1, At each rising or falling edge of the clock signal the output Q will be in complementary state.

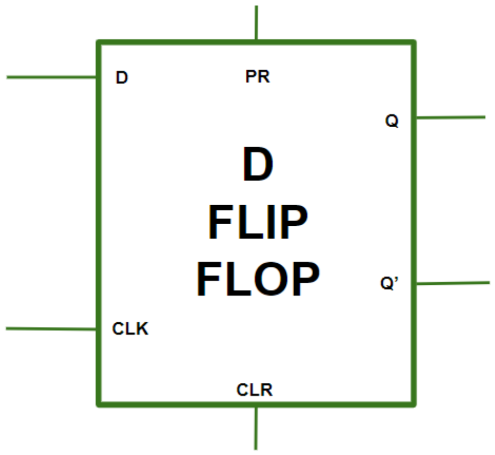
Characteristics Equation for T Flip Flop:

$$Q_{N+1} = Q'_N T + Q_N T' = Q_N \text{ XOR } T$$

D Flip Flop:

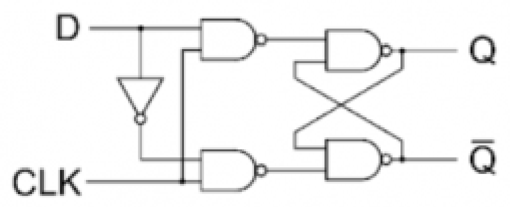
The D Flip Flop Consists a single data input(D), a clock input(CLK),and two outputs: Q and Q' (the complement of Q).

Block Diagram of D Flip Flop



Circuit Diagram and Truth Table of D Flip Flop:

Given Below is the Diagram of D Flip Flop with its Truth Table



Q	D	Q(t+1)
0	0	0
0	1	1
1	0	0
1	1	1

Operation of the D Flip-Flop

Given Below is the operation of D Flip-Flip

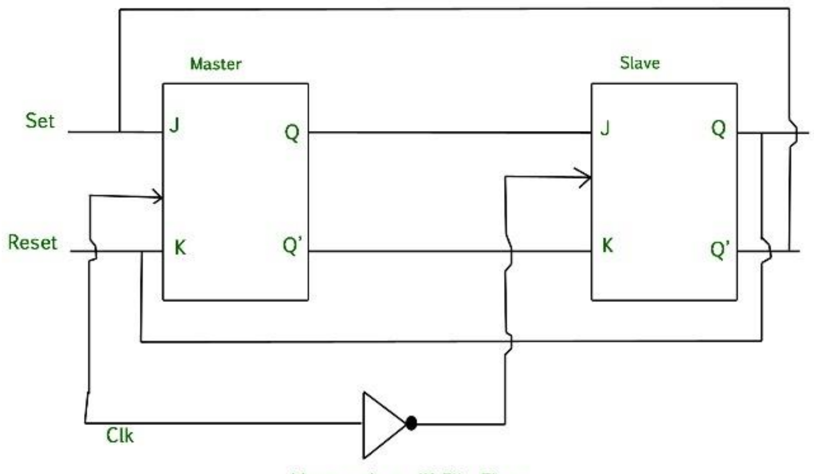
- Case 1 (PR=CLR=0): This condition represents as invalid state where both PR (present) and CLR (clear) inputs are inactive.
- Case 2 (PR=0 and CLR=1):This state is set state in which PR is inactive (0) and CLR is active(1) and the output Q is set to 1.
- Case 3 (PR=1 and CLR=0):This state is reset state in which PR is active (1) and CLR is inactive (0) and the complementary output Q' is set to 1.
- Case 4 (PR=CLR=1):In This state the flip flop behaves as normal, both PR and CLR inputs are active(1).

Characteristics Equation for D Flip Flop:

QN+1 = D

3.4. Master-slave flip-flop:

Master Slave JK flip flop - The Master-Slave Flip-Flop is basically a combination of two JK flip-flops connected together in a series configuration. Out of these, one acts as the “master” and the other as a “slave”. The output from the master flip flop is connected to the two inputs of the slave flip flop whose output is fed back to inputs of the master flip flop. In addition to these two flip-flops, the circuit also includes an inverter. The inverter is connected to clock pulse in such a way that the inverted clock pulse is given to the slave flip-flop. In other words, if CP=0 for a master flip-flop, then CP=1 for a slave flip-flop and if CP=1 for master flip flop then it becomes 0 for slave flip flop.

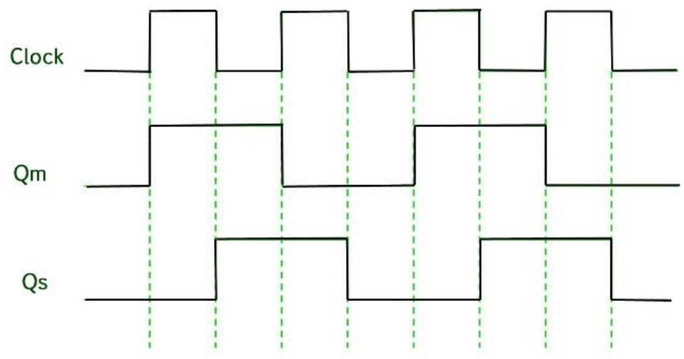


Master-slave JK Flip-Flop

Working of a master slave flip flop :

1. When the clock pulse goes to 1, the slave is isolated; J and K inputs may affect the state of the system. The slave flip-flop is isolated until the CP goes to 0. When the CP goes back to 0, information is passed from the master flip-flop to the slave and output is obtained.
2. Firstly the master flip flop is positive level triggered and the slave flip flop is negative level triggered, so the master responds before the slave.
3. If J=0 and K=1, the high Q' output of the master goes to the K input of the slave and the clock forces the slave to reset, thus the slave copies the master.
4. If J=1 and K=0, the high Q output of the master goes to the J input of the slave and the Negative transition of the clock sets the slave, copying the master.
5. If J=1 and K=1, it toggles on the positive transition of the clock and thus the slave toggles on the negative transition of the clock.
6. If J=0 and K=0, the flip flop is disabled and Q remains unchanged.

Timing Diagram of a Master Slave flip flop:



1. When the Clock pulse is high the output of master is high and remains high till the clock is low because the state is stored.
2. Now the output of master becomes low when the clock pulse becomes high again and remains low until the clock becomes high again.
3. Thus toggling takes place for a clock cycle.
4. When the clock pulse is high, the master is operational but not the slave thus the output of the slave remains low till the clock remains high.
5. When the clock is low, the slave becomes operational and remains high until the clock again becomes low.
6. Toggling takes place during the whole process since the output is changing once in a cycle.

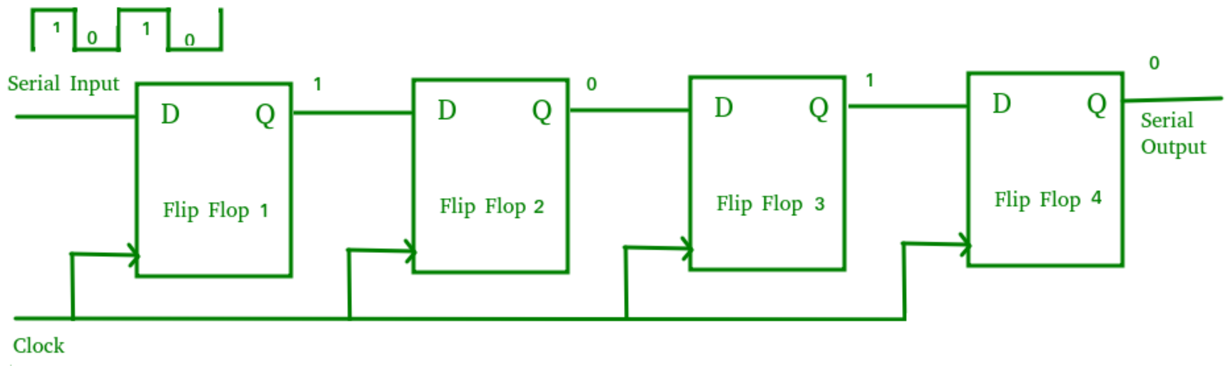
This makes the Master-Slave J-K flip flop a Synchronous device as it only passes data with the timing of the clock signal.

3.5. Registers: serial in serial out and parallel in and parallel out:

Serial In Serial Out (SISO) Shift Register:

A Serial-In Serial-Out shift register is a sequential logic circuit that allows data to be shifted in and out one bit at a time in a serial manner. It consists of a cascade of flip-flops connected in series, forming a chain. The input data is applied to the first flip-flop in the chain, and as the clock pulses, the data propagates through the flip-flops, ultimately appearing at the output.

The logic circuit provided below demonstrates a serial-in serial-out (SISO) shift register. It comprises four D flip-flops that are interconnected in a sequential manner. These flip-flops operate synchronously with one another, as they all receive the same clock signal.



4 Bit SISO Register:

The synchronous nature of the flip-flops ensures that the shifting of data occurs in a coordinated manner. When the clock signal rises, the input data is sampled and stored in the first flip-flop. On subsequent clock pulses, the stored data propagates through the flip-flops, moving from one flip-flop to the next.

Each D flip-flop in the circuit has a Data (D) input, a Clock (CLK) input, and an output (Q). The D input represents the data to be loaded into the flip-flop, while the CLK input is connected to the common clock signal. The output (Q) of each flip-flop is connected to the D input of the next flip-flop, forming a cascade.

Working of SISO:

A SISO is called simply because it has one input and one output. SISO stands for Single Input Single Output and is commonly used in relation to systems as well as signals. In plain terms, SISO means a system with only one input and one output.

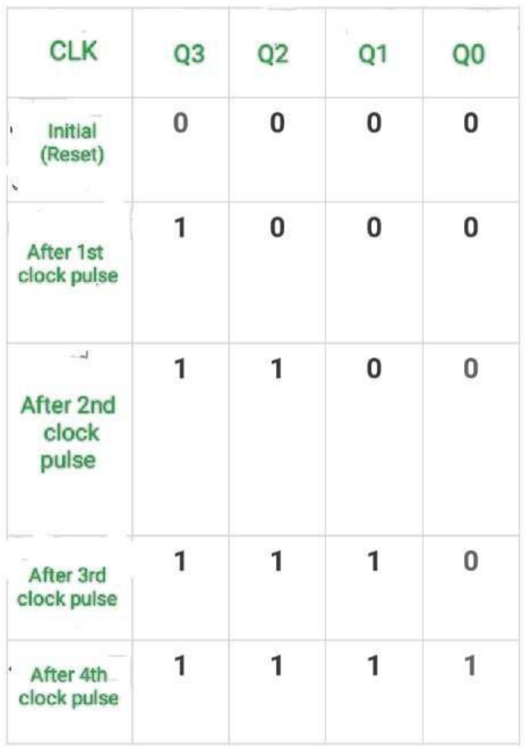
The operating principle of SISO involves feeding the intended signal into the system where it gets processed in order to produce an output. Therefore, input signal refers to what goes into the system while output signal describes the result or response produced by the system depending on that particular input.

Signals at both the input and the output can be mathematically modeled by functions or equations in a SISO system. A system can be represented through a transfer function, describing how the system will modify the input signal to produce the output signal.

Different properties of the system, such as stability, frequency response, or time response, are taken into consideration to understand the behavior and performances of a system under different conditions.

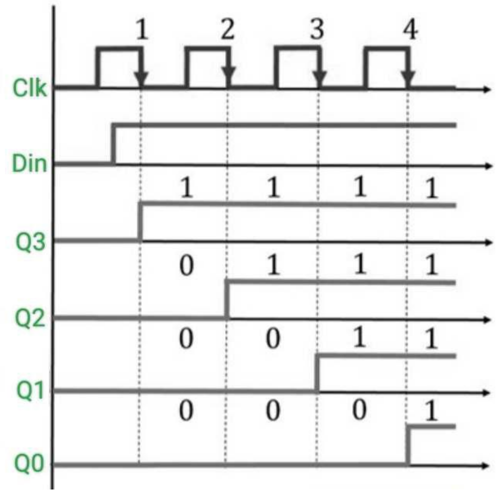
Overall, a SISO system means that there is one input signal processed to obtain a corresponding output signal; therefore, it is central to the notion of systems and signals.

Truth Table for SISO



CLK	Q3	Q2	Q1	Q0
Initial (Reset)	0	0	0	0
After 1st clock pulse	1	0	0	0
After 2nd clock pulse	1	1	0	0
After 3rd clock pulse	1	1	1	0
After 4th clock pulse	1	1	1	1

Waveform Representation



Functionality and Operation:

The operation of a SISO shift register relies on two primary components: the flip-flops and the clock signal.

- Flip-Flops: A flip-flop is a fundamental building block of sequential circuits. In the case of a SISO shift register, each flip-flop stores a single bit of data. The number of flip-flops determines the length or size of the shift register.
- Clock Signal: The clock signal synchronizes the movement of data through the shift register. With each clock pulse, the data shifts from one flip-flop to the next. The clock signal ensures that the data propagates in a controlled and synchronized manner.

When the clock signal transitions from low to high (or high to low, depending on the specific implementation), the input data is sampled and stored in the first flip-flop. On subsequent clock pulses, the stored data moves through the chain of flip-flops. The output of the shift register is taken from the last flip-flop in the series.

Applications of SISO Shift Registers:

Serial-In Serial-Out shift registers find applications in a wide range of digital systems. Here are a few common examples:

- Data Storage and Retrieval: SISO shift registers are often used to store and retrieve data in applications where serial transmission is more efficient or feasible. For instance, in communication systems, data is often transmitted serially, and a shift register allows for temporary storage before further processing.
- Serial-to-Parallel Conversion: By using a SISO shift register in combination with additional circuitry, serial data can be converted into parallel form. This conversion is useful when interfacing between serial and parallel devices, such as microprocessors and peripheral devices.
- Delay and Time-Sequence Generation: The cascaded nature of shift registers enables the generation of delayed versions of a signal or the generation of complex time sequences. By tapping into different stages of the shift register, various delayed versions of the input signal can be obtained.
- Data Encryption and Decryption: Shift registers can be employed in cryptographic applications for data encryption and decryption. By utilizing feedback connections and appropriate logical operations, the shift register can act as a key generator or cipher unit.
- Frequency Division and Counting: Serial-In Serial-Out shift registers are used in frequency division and counting applications. By connecting the output of a flip-flop back to the input, the shift register can function as a binary counter, dividing the input frequency by a factor of two with each clock pulse.

Parallel in and parallel out Registers:

A PIPO shift register is a collection of flip-flops arranged in a series, with each flip-flop capable of storing one bit of data. The primary characteristic that distinguishes a PIPO shift register is its ability to load data and output it in parallel. Unlike other shift registers that deal with serial input and output, the parallel loading and output capabilities of a PIPO shift register make it a powerful tool in many digital systems.

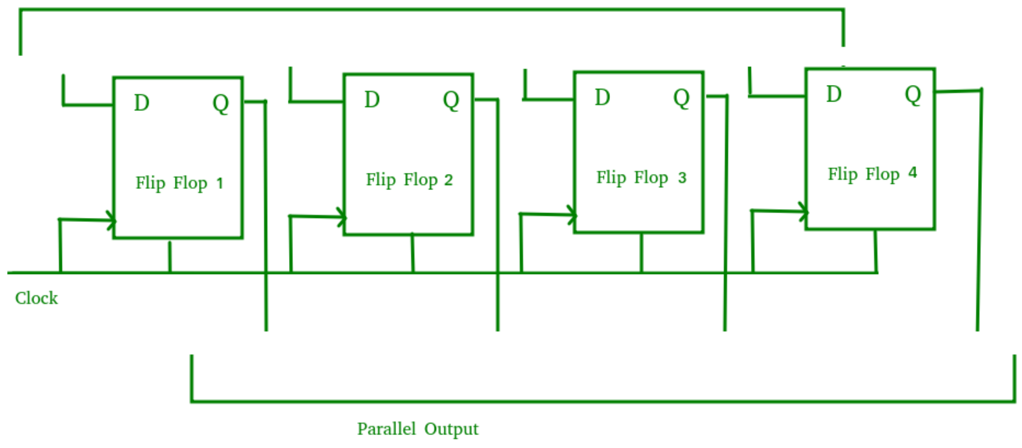
Key Terminologies Related to PIPO

- Shift Register: A digital circuit consisting of flip-flops used to sequentially store and shift data.
- Flip-Flop: Simple building block of a variable name that can store some data. It can be in the 0 or 1 state.
- Parallel Input: Input to the PIPO transfer register from the data simultaneously uploaded to the flip-flops. Allows multiple files to be entered simultaneously.
- Parallel output: Output of the PIPO key register, which can access data stored in the flip-flops. It allows viewing multiple files at the same time.
- Clock Signal: The time signal is used to synchronize the renaming job. Determines the amount of data that has been changed or replaced in the list.
- Serial Input: Another input method for converting data to serial input, where data is entered bit by bit sequentially.
- Serial output: Another output method for extracting data from a switch, where the data is output bit by bit sequentially.
- Shift Left: A function that shifts data in a scroll to the left. The leftmost bits are discarded and the new ones are replaced from the right.
- Shift right: A shift operation where the data in the shift is shifted to the right. Bits on the right are discarded and new bits are moved on the left.

How Does PIPO Shift Register Work?

Let's discuss a example of 4-bit shift register to understand the operations of a PIPO shift register. It consists of four flip-flops, labelled D0, D1, D2, and D3. Each flip-flop can store one bit of data. The data can be loaded into the flip-flops simultaneously through the parallel input, known as the Data input. Once the data is loaded, it can be read out simultaneously from each flip-flop through the parallel outputs.

PIPO Shift Register Circuit Diagram



Each flip-flop's contents are passed sequentially to the following flip-flop to shift the data out. This can be accomplished by providing a clock signal that triggers the shifting operation. When the clock signal is activated, the data in each flip-flop is transferred to the adjacent flip-flop, allowing new data to be loaded into the first flip-flop.

Applications of PIPO Shift Registers:

- Data storage: PIPO transfer registers are widely used for temporary data storage in digital systems. The parallel upload feature provides fast data entry, while the parallel output allows for on-demand data re-storage.
- Data Transfer: PIPO shift registers are used in applications where data needs to be transferred from one location to another in a fast manner. By loading the data in parallel and then outputting it simultaneously, PIPO shift registers facilitate high-speed data transmission.
- Data Manipulation: PIPO shift registers are valuable for performing various data manipulation operations, such as data sorting, arithmetic calculations, and pattern recognition. The ability to process multiple bits of data in parallel enhances the speed and efficiency of these operations.

- Serial to Parallel Conversion: PIPO conversion can convert serial data to parallel data. This is important when dealing with devices that need to work together. Load data serially and output in parallel, easily switch serial-parallel systems seamless integration.

3.6. Counters:

A **Counter** is a device which stores (and sometimes displays) the number of times a particular event or process has occurred, often in relationship to a clock signal. Counters are used in digital electronics for counting purpose, they can count specific event happening in the circuit. For example, in UP counter a counter increases count for every rising edge of clock. Not only counting, a counter can follow the certain sequence based on our design like any random sequence 0,1,3,2… .They can also be designed with the help of flip flops. They are used as frequency dividers where the frequency of given pulse waveform is divided. Counters are sequential circuit that count the number of pulses can be either in binary code or BCD form. The main properties of a counter are timing , sequencing , and counting.

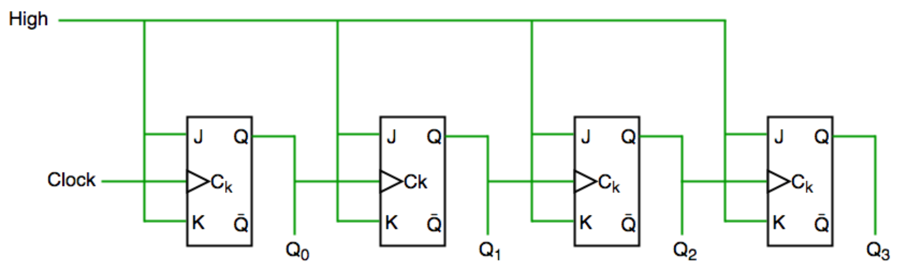
Counter Classification:

Counters are broadly divided into two categories

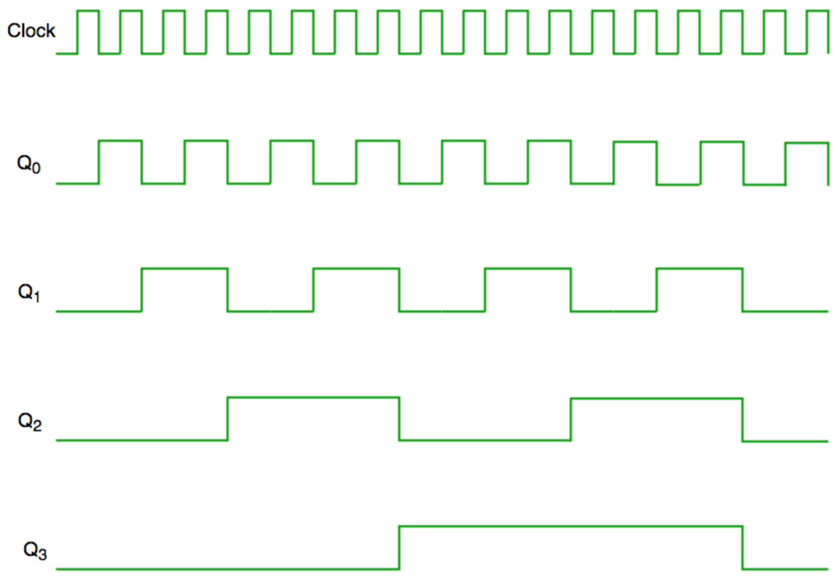
1. Asynchronous counter
2. Synchronous counter

1. Asynchronous Counter:

In asynchronous counter we don't use universal clock, only first flip flop is driven by main clock and the clock input of rest of the following flip flop is driven by output of previous flip flops. We can understand it by following diagram



(a) Asynchronous counter



(b) Timing Diagram

It is evident from timing diagram that Q0 is changing as soon as the rising edge of clock pulse is encountered, Q1 is changing when rising edge of Q0 is encountered(because Q0 is like clock pulse for second flip flop) and so on. In this way ripples are generated through Q0,Q1,Q2,Q3 hence it is also called **RIPPLE counter and serial counter.** A ripple counter is a cascaded arrangement of flip flops where the output of one flip flop drives the clock input of the following flip flop

MOD-8 Counter and State Diagram

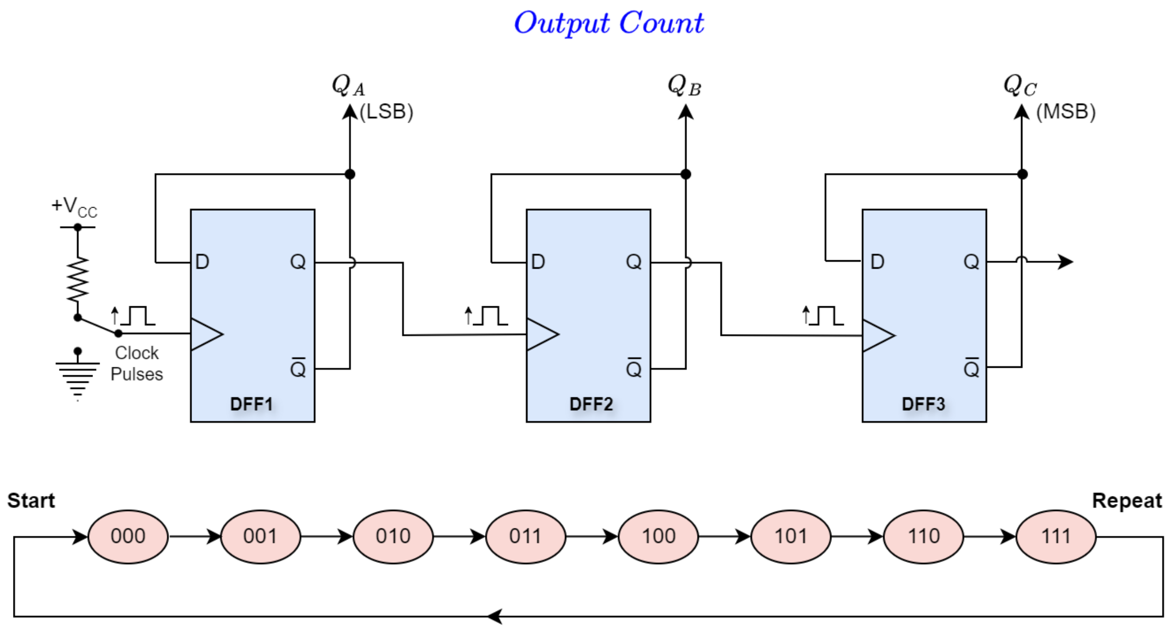
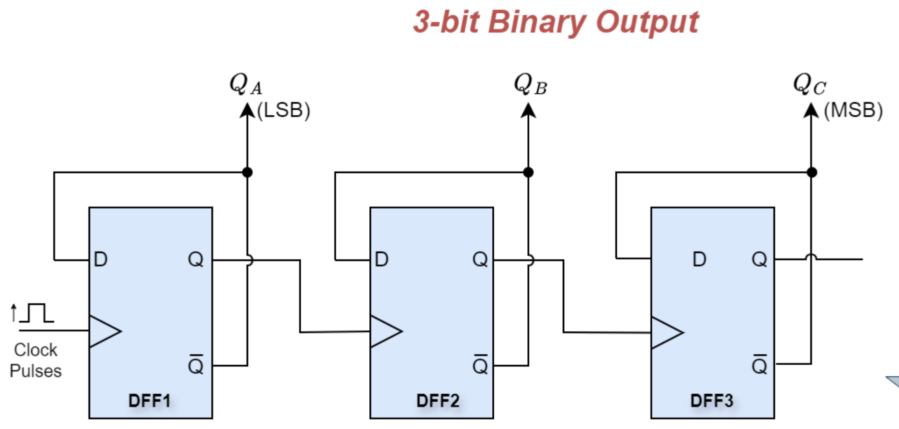


Fig:MOD-8 Counter and State Diagram

By constructing mod counters with a natural count of 2^n states, we may therefore create counters that have mod counts of 2, 4, 8, 16, and so on, before repeating. However, sometimes it is required to have a modulus counter without a modulo that is a power of two that resets its count to zero during the regular counting procedure. As an illustration, consider a counter with a modulus of 3, 5, 6, or 10.

MOD-8 Counter and Truth Table



Truth Table

Count	Q_C	Q_B	Q_A
1	0	0	0
2	0	0	1
3	0	1	0
4	0	1	1
5	1	0	0
6	1	0	1
7	1	1	0
8	1	1	1

Fig: MOD-8 Counter and Truth Table

We need to change the above 3-bit counter circuit such that it will reset itself back to zero after a count of five to build a MOD-5 counter. It is a count sequence that goes like this: 1→2→3→4→5→reset, and so on.

Since 000 is a legitimate count state, a MOD-5 counter would yield a 3-bit binary count sequence ranging from 0 to 4. This would give us the following binary count sequence: 000, 001, 010, 011, 100. As a result, as indicated by the state diagram below, the counter circuit must reset itself at the next counting stage, which is the count of six. In this case, the output condition would be QA = 1, QB = 0, and QC = 1 in binary.

10 Mod Counters:

The Decade Counter is a fine example of a modulo-m counter circuit that creates a counter with a modulus of 10 by utilizing external combinational circuits. Divide-by-10 counters, like the TTL 74LS90, are appropriate for human interface applications that necessitate a digital display since their counting sequence consists of 10 states.

Using external AND and OR gates, we can detect the occurrence of the 9th counting state and reset the decade counter back to zero. The decade counter has four outputs that result in a 4-bit binary value. It counts up from 0 to 9 continuously after receiving an input clock pulse, just as other mod counters.

The counter returns to 0000 instead of incrementing to 1010 when it hits count 9 (1001 in binary). The JK flip-flops (TTL 74LS73), which change states on the negative trailing edge of the clock signal as indicated, can be used to create the basic circuit of a decade counter.

MOD-10 Decade Counter

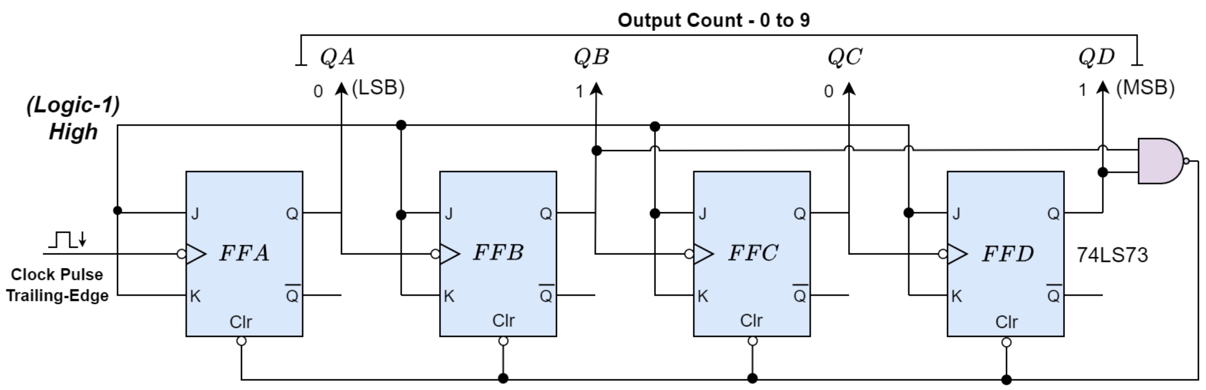
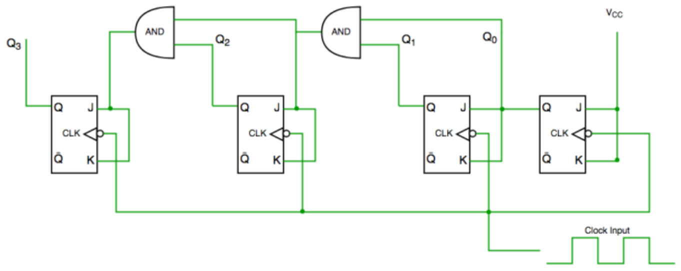


Fig: MOD-10 Decade Counter

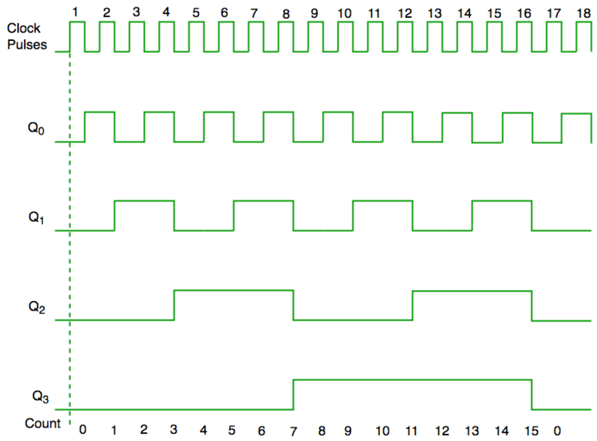
Synchronous counter:

Unlike the asynchronous counter, synchronous counter has one global clock which drives each flip flop so output changes in parallel. The one advantage of synchronous counter over asynchronous counter is, it can operate on higher frequency than asynchronous counter as it does not have cumulative delay because of same clock is given to each flip flop. It is also called as parallel counter.

Synchronous counter circuit



Timing diagram synchronous counter



From circuit diagram we see that Q0 bit gives response to each falling edge of clock while Q1 is dependent on Q0, Q2 is dependent on Q1 and Q0 , Q3 is dependent on Q2,Q1 and Q0.

Advantages of Synchronous Counters:

- **Faster Operation**: All flip-flops trigger simultaneously for quicker response.
- **Precise Timing**: Synchronized operation reduces timing errors.

- **Low Propagation Delay**: No ripple effect between flip-flops.
- **High Reliability**: Stable and error-resistant performance.
- **Ideal for High-Speed Use**: Suitable for fast timers and processors.
- **Simplified Control**: Single clock source makes timing easier to manage.

3.7. Ring counter:

A ring counter is a typical application of the Shift register. The ring counter is almost the same as the shift counter. The only change is that the output of the last flip-flop is connected to the input of the first flip-flop in the case of the ring counter but in the case of the shift register it is taken as output. Except for this, all the other things are the same. It is a type of digital counter used in circuits, typically built with flip-flops. It works by shifting a 1 through a series of flip-flops, one at a time, in a loop. It’s called a ring counter because the 1 loops back to the start once it reaches the end, forming a cycle.

No. of states in Ring counter = No. of flip-flop used

Key Characteristics of a Ring Counter:

- Single bit '1': Only one bit in the counter is set to 1 at any given time.
- Cyclic Behavior: The pattern of the bits follows a cyclic behavior and repeats after a fixed number of steps.
- No Reset: The ring counter doesn't automatically reset to zero. Instead, it starts at a defined state and then continues cycling.

Working Of Ring Counter:

- A Ring Counter is typically built using flip-flops (D, T, or JK flip-flops).
- It consists of n flip-flops, where n is the number of states in the counter. The number of flip-flops determines the number of unique states the counter will cycle through.
- One bit in the counter is set to 1, and the rest of the bits are set to 0. In each clock cycle, the 1 bit shifts through the flip-flops, and the pattern repeats.
- The output of the last flip-flop is fed back into the first flip-flop, forming a loop, hence the name Ring Counter.

Example of a 4-bit Ring Counter:

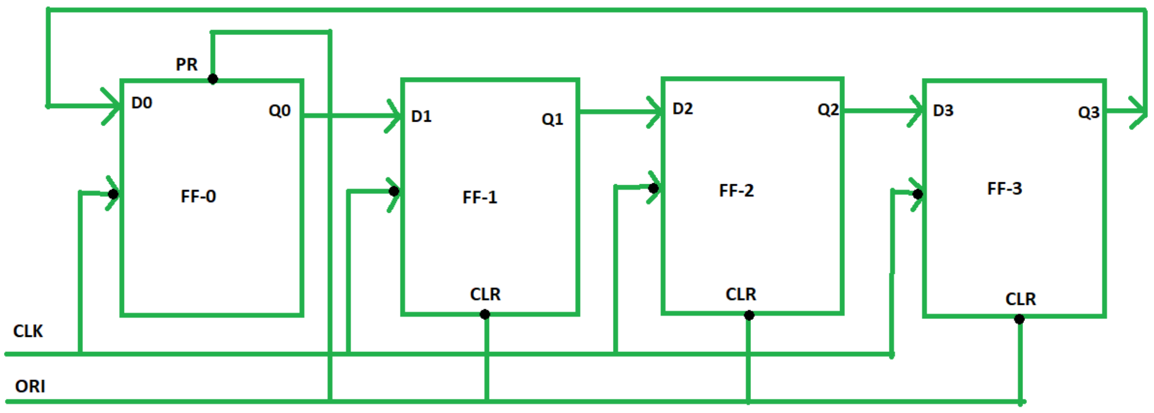
Let’s consider a 4-bit Ring Counter:

1. Initial state: 1000 (the first flip-flop is set to 1, and others are set to 0).
2. Clock cycle 1: The '1' bit shifts to the next flip-flop, resulting in the state 0100.
3. Clock cycle 2: The '1' bit shifts to the next flip-flop, resulting in the state 0010.
4. Clock cycle 3: The '1' bit shifts to the next flip-flop, resulting in the state 0001.
5. Clock cycle 4: The '1' bit shifts back to the first flip-flop, and the cycle repeats, resulting in the state 1000.

State Sequence:

1000 → 0100 → 0010 → 0001 → 1000.

So, for designing a 4-bit Ring counter we need 4 flip-flops as designed below :



Ring Counter

Components and Signals:

- Overriding Input (ORI):
  The ORI is used to override the normal functionality of the flip-flops. In this case, Preset (PR) and Clear (CLR) are used as ORI.
  - Preset (PR): When the PR signal is 0, the output of the flip-flop (Q) is set to 1. This is an active-low signal.

   - Clear (CLR): When the CLR signal is 0, the output of the flip-flop (Q) is set to 0. This is also an active-low signal.

- Preset (PR) = 0, Q = 1:
  When PR is 0, the output of the flip-flop is forced to 1, regardless of other inputs or the clock signal.
- Clear (CLR) = 0, Q = 0:
  When CLR is 0, the output of the flip-flop is forced to 0, overriding the other inputs.

PRESETED 1

ORI	CLK	Q0	Q1	Q2	Q3
low	X	1	0	0	0
high	low	0	1	0	0
high	low	0	0	1	0
high	low	0	0	0	1
high	low	1	0	0	0

- The Preset (PR) and Clear (CLR) signals are used to control the state of the flip-flops.
- The ORI input of each flip-flop is connected to the Preset (PR) for FF-0 (the first flip-flop) and to Clear (CLR) for FF-1, FF-2, and FF-3 (the other flip-flops).
   - At FF-0, when PR = 0, the output Q = 1 is generated, and this creates the initial state of the counter.
   - At the other flip-flops (FF-1, FF-2, FF-3), the CLR = 0, forcing the outputs Q = 0 at these flip-flops.

- This setup results in Pre-set 1 at FF-0, and the other flip-flops hold 0. This "1" at FF-0 then propagates through the flip-flops in a cyclic manner, creating the sequence that is characteristic of the Ring Counter.

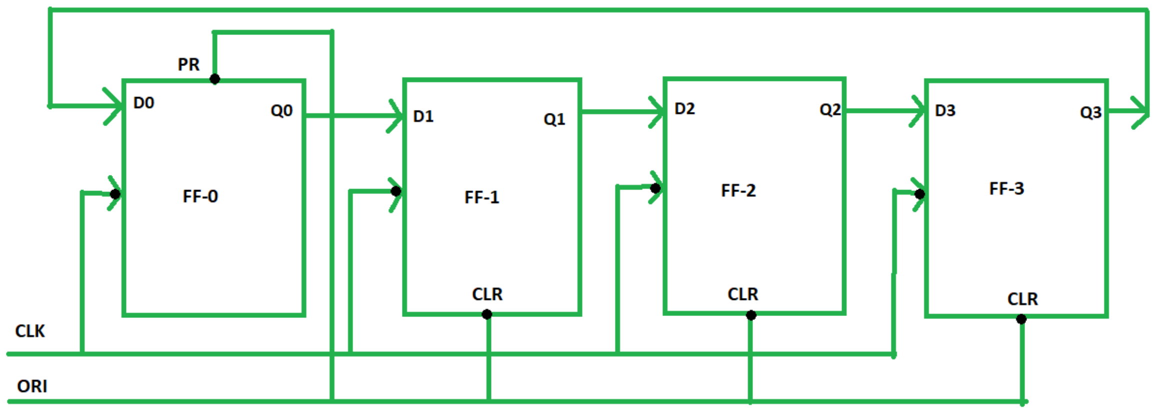
This Preseted 1 is generated by making ORI low and that time Clock (CLK) becomes don't care. After that ORI is made to high and apply low clock pulse signal as the Clock (CLK) is negative edge triggered. After that, at each clock pulse, the preseted 1 is shifted to the next flip-flop and thus forms a Ring. In this way can design a 4-bit Ring Counter using four D flip-flops.

Types of Ring Counter:

There are two types of Ring Counter:

Straight Ring Counter:

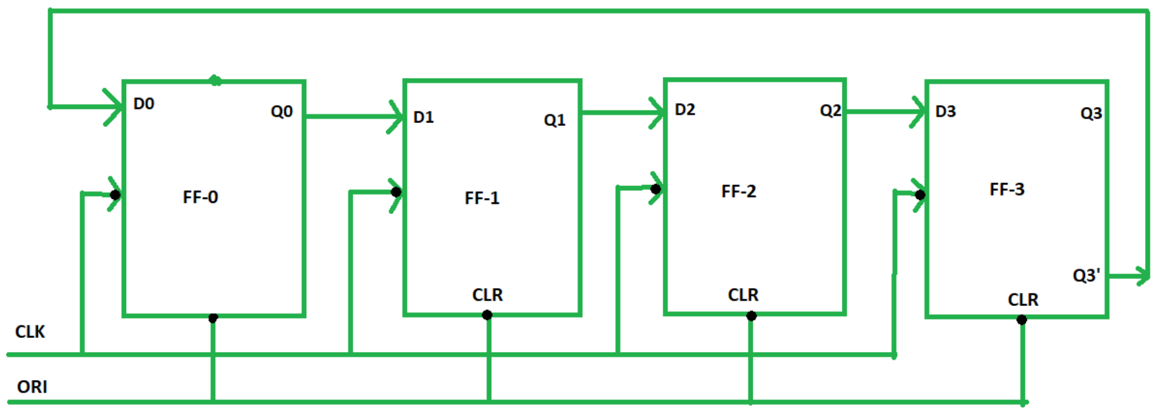
It is also known as One hot Counter. In this counter, the output of the last flip-flop is connected to the input of the first flip-flop. The main point of this Counter is that it circulates a single one (or zero) bit around the ring. Here, we use Preset (PR) in the first flip-flop and Clock (CLK) for the last three flip-flops.



Straight Ring Counter

Twisted Ring Counter:

It is also known as a switch-tail ring counter, walking ring counter, or Johnson counter. It connects the complement of the output of the last shift register to the input of the first register and circulates a stream of ones followed by zeros around the ring. Here, we use Clock (CLK) for all the flip-flops. In the Twisted Ring Counter, the number of states = 2 X the number of flip-flops.



Twisted Ring Counter

Advantages of Ring Counter:

- Simplicity: It's easy to design, especially for small applications.
- Low Resource Use: You don't need a lot of components compared to other counters.
- Predictable Output: It will always go through the same set of states in the same order.
- Easy to Decode: Each state is distinct and easy to decode because it uses one-hot encoding only one output is high at a time.
- Glitch-Free Output: Since only one flip-flop is active at a time, there's less chance of output glitches, making it more reliable in certain applications.

Up-down counter:

Up/Down counter is the combination of both the counters in which we can perform up or down counting by changing the Mode control input.

Design of 3 Bit Asynchronous UP/DOWN Counter:

It is used more than separate up or down counter.

1. In this a mode control input (say M) is used for selecting up and down mode.

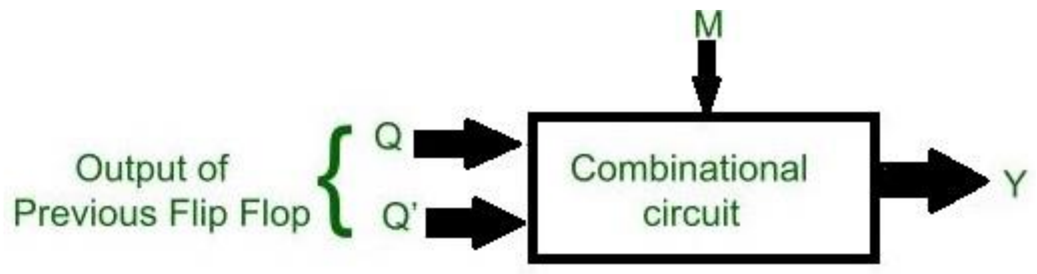
2. A combinational circuit is required between each pair of flip-flop to decide whether to do up or do down counting.

For n = 3, i.e for 3 bit counter -

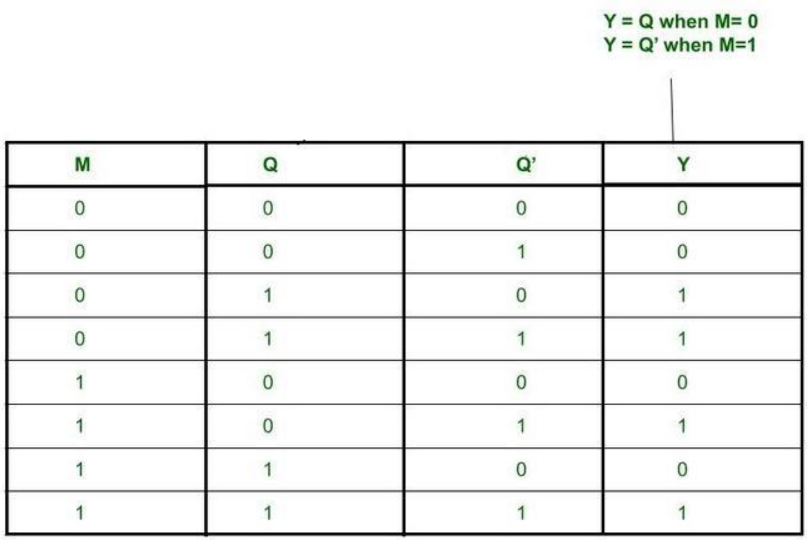
Maximum count = 2n -1 and number of states are 2n.

Steps involve in design are :

Step 1 : Decision for Mode control input -



When M = 0, then Y= Q, therefore it will perform Up counting (As discussed above). When M = 1, then Y= Q' therefore it will perform Down counting (As discussed above). Combinational circuit is required for deciding mode control(i.e., whether counter will perform Up counting or Down counting).
So the all possible combinations are -

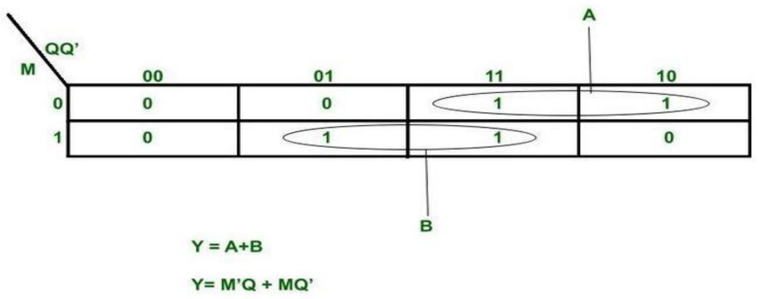


Y = Q when M= 0
Y = Q' when M=1

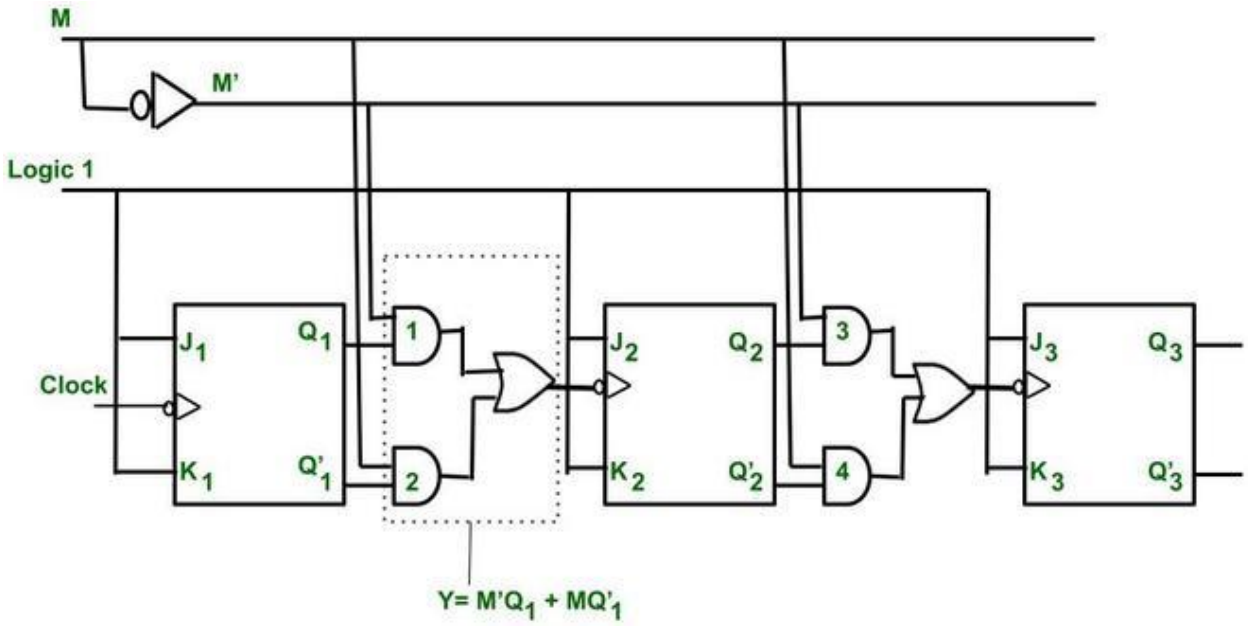
M	Q	Q'	Y
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	1

Here Y is the output function

K-map for finding output Y that will be given as clock to next FF.



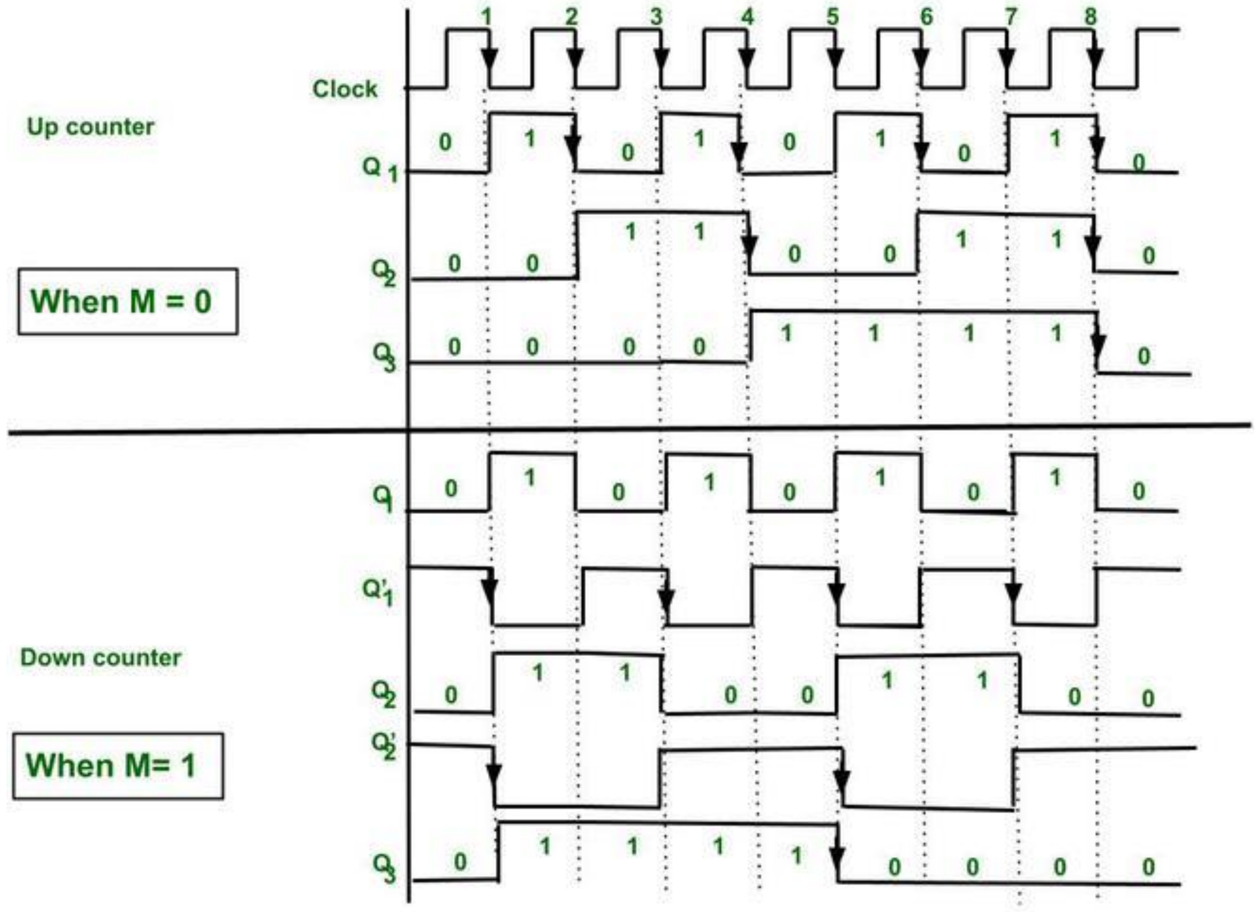
Step 2 : Insertion of Combinational logic between every pair of FFs -



Up/Down Counter:

Timing diagram

Initially Q3 = 0, Q2 = 0, Q1 = 0.



Timing diagram for 3 bit asynchronous up/down counter

Case 1 - When M=0, then M' =1.

Put this in Y= M'Q + MQ'= Q So Q is acting as clock for next FFs. Therefore, the counter will act as Up counter.

Explanation of Up counter:

- The 1st FF is connected to logic 1. Therefore, it will toggle for every falling edge.
- The 2nd FF input is connected to Q1.Therefore it changes its state when Q1= 1 and there is falling edge of clock.
- Similarly, 3rd FF is connected to Q2. Therefore, it changes its state when Q2= 1 and there is falling edge of clock.

- By this we can generate counting states of Up counter.
- After every 8th falling edge the counter is again reaching to state 0 0 0. Therefore, it is also known as divide by 8 circuit or mod 8 counter.

Case 2 - When M=1, then M' =0. Put this in Y= M'Q + MQ'= Q'. So Q' is acting as clock for next FFs. Therefore, the counter will act as Down counter.

Explanation of Down counter:

- The 1st FF is connected to logic 1. Therefore, it will toggle for every falling edge.
- The 2nd FF input is connected to Q'1.Therefore it changes its state when Q'1= 1 and there is falling edge of clock.
- Similarly, 3rd FF is connected to Q'2. Therefore, it changes its state when Q'2= 1 and there is falling edge of clock.
- By this we can generate counting states of down counter.
- After every 8th falling edge the counter is again reaching to state 0 0 0.
- Therefore, it is also known as divide by 8 circuit or mod 8 counter.

Up and Down Falling State:

In the asynchronous up/down counter design, up and down falling states are called specific conditions that cause the counter to either increment (up) or decrement (down).

Up Falling State : This state results when the counter goes up, incremented by one count. In an asynchronous counter, the up falling state is normally caused by some specific input signal or by some certain condition. On the occurrence of this kind of condition, the counter proceeds to the next count.

Down Falling State : The down falling state is when the counter decrements, going down by one count. Much like the up falling state, this state also starts with some input signal or condition which will prompt the counter to advance to the previous count.

These states are very critical to an asynchronous up/down counter since it decides on the direction in which the counter progresses. If these states have been effectively controlled, then according to the input signals, the counter will count up or down accurately.

Advantages:

Designing an up/down counter asynchronously has several benefits.

- **Flexibility :** An up/down counter that is not synchronous provides flexible counting up and down. This flexibility can be important for cases when the counting direction must change dynamically based on some conditions or inputs.
- **Simplicity :** Compared to synchronized counters, asynchronous counters are relatively easy to build. They operate without using clock signals, enabling them to be simpler than others in certain cases.
- **Speed :** Asynchronous counters can count at hh speeds because they do not depend on the clock signal for counting.
- **Simple Installation :** Timing-critical system whose requirements suit the functioning of an asynchronous counter can find it easy to incorporate such devices.
- **Decreased Energy Usage :** The asynchronous counters have a lower power consumption than synchronous ones because in their operation they do not need a continuous clocking signal. This can be advantageous for applications that are sensitive to power supply.

Generally speaking, some of the advantages associated with designing an asynchronous Up/Down counter include flexibility, simplicity, speed, lower power consumption and easy integration in different digital circuits.

3.8. A/D and D/A converter:

Analog to Digital Conversion:

Digital Signal: A digital signal is a signal that represents data as a sequence of discrete values; at any given time it can only take on one of a finite number of values.

Analog Signal: An analog signal is any continuous signal for which the time varying feature of the signal is a representation of some other time-varying quantity i.e., analogous to another time varying signal.

Importance of Analog to Digital Conversion

- The main role of ADC in modern technology development process is the transition of voice communication systems from outdated analogue signal processing to the more advanced voice over IP, or VoIP, systems of today is largely due to the contribution.
- The teletypewriters and other computer input devices needed to be connected to a modem which was connected to a mainframe or other front end computer system to communicate with the required computer systems. In contrast to the ultrahigh-speed networks of today, modem transmission speeds were modest to process.
- The systems for smaller office applications and the digital private branch exchange, or PBX, were developed by using ADC technology as the foundation to process properly.

Techniques of Analog-to-Digital Conversion

The following techniques can be used for Analog to Digital Conversion -

a. PULSE CODE MODULATION:

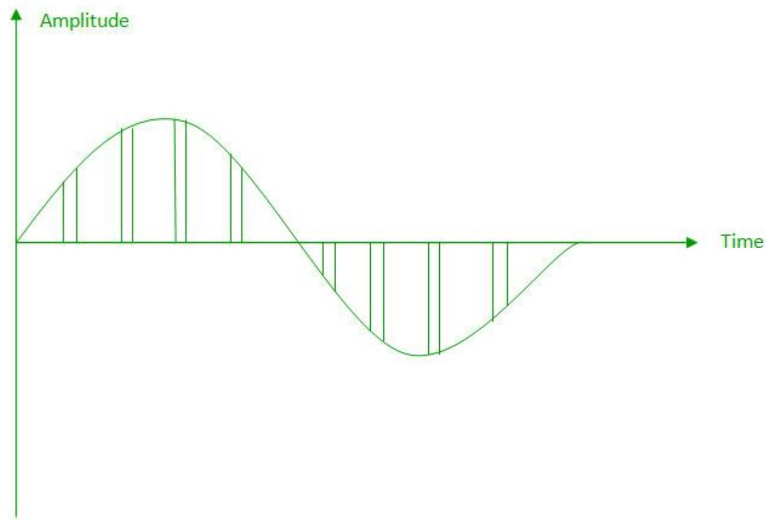
The most common technique to change an analog signal to digital data is called pulse code modulation (PCM). A PCM encoder has the following three processes:

1. Sampling
2. Quantization
3. Encoding

Low pass filter : The low pass filter eliminates the high frequency components present in the input analog signal to ensure that the input signal to sampler is free from the unwanted frequency components. This is done to avoid aliasing of the message signal.

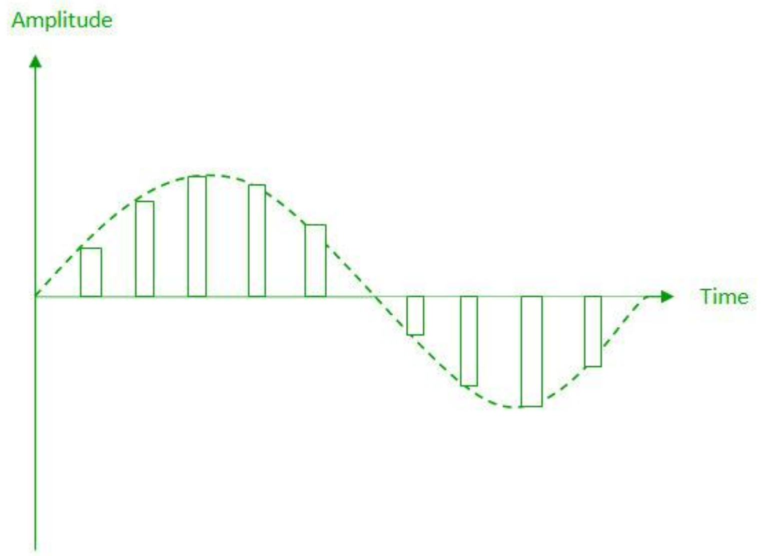
1. **Sampling -** The first step in PCM is sampling. Sampling is a process of measuring the amplitude of a continuous-time signal at discrete instants, converting the continuous signal into a discrete signal. There are three sampling methods: **(i) Ideal Sampling:** In ideal Sampling also known as Instantaneous sampling pulses from the analog signal are

sampled. This is an ideal sampling method and cannot be easily implemented. **(ii) atural Sampling:** Natural Sampling is a practical method of sampling in which pulse have finite width equal to T.The result is a sequence of samples that retain the shape of the analog



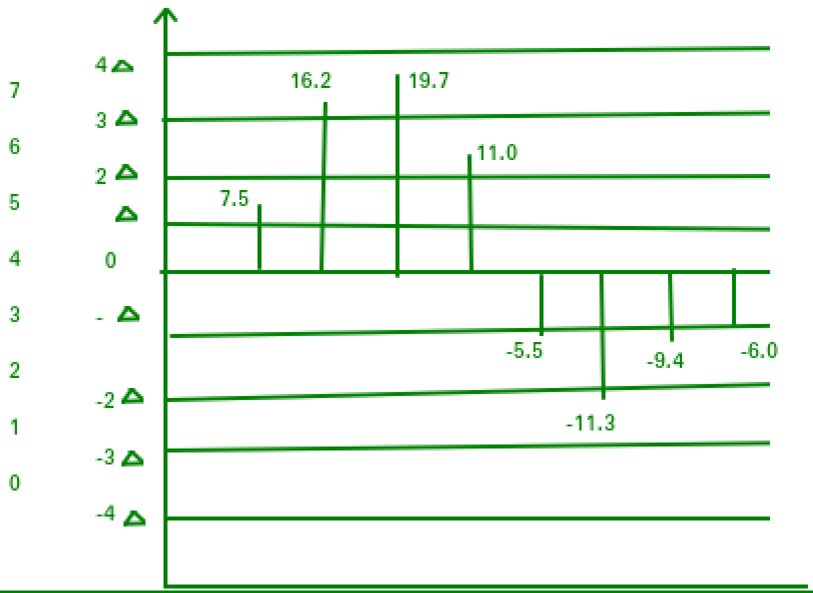
signal.

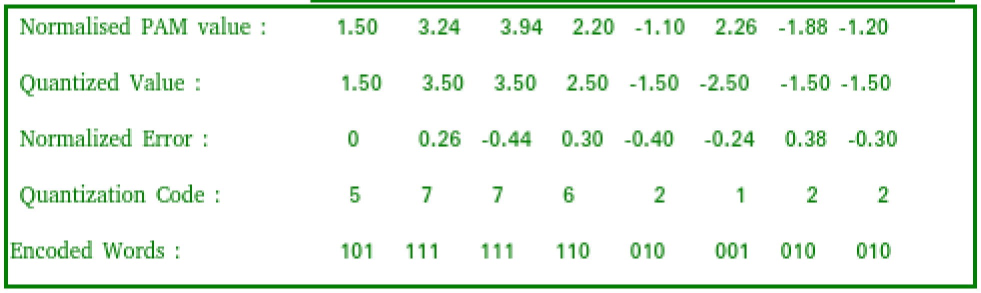
(iii) Flat top sampling: In comparison to natural sampling flat top sampling can be easily obtained. In this sampling technique, the top of the samples remains constant by using a circuit. This is the most common sampling method used**.**



Nyquist Theorem: One important consideration is the sampling rate or frequency. According to the Nyquist theorem, the sampling rate must be at least 2 times the highest frequency contained in the signal. It is also known as the minimum sampling rate and given by: Fs =2*fh

2. **Quantization -** The result of sampling is a series of pulses with amplitude values between the maximum and minimum amplitudes of the signal. The set of amplitudes can be infinite with non-integral values between two limits. The following are the steps in Quantization:
    1. We assume that the signal has amplitudes between Vmax and Vmin
    2. We divide it into L zones each of height d where, d= (Vmax- Vmin)/ L





Normalised PAM value :	1.50	3.24	3.94	2.20	-1.10	2.26	-1.88	-1.20
Quantized Value :	1.50	3.50	3.50	2.50	-1.50	-2.50	-1.50	-1.50
Normalized Error :	0	0.26	-0.44	0.30	-0.40	-0.24	0.38	-0.30
Quantization Code :	5	7	7	6	2	1	2	2
Encoded Words :	101	111	111	110	010	001	010	010

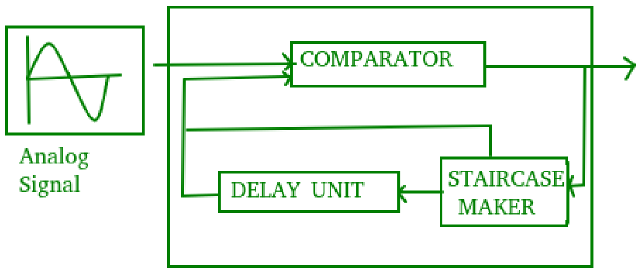
3. The value at the top of each sample in the graph shows the actual amplitude.

4. The normalized pulse amplitude modulation(PAM) value is calculated using the formula amplitude/d.
5. After this we calculate the quantized value which the process selects from the middle of each zone.
6. The Quantized error is given by the difference between quantized value and normalised PAM value.
7. The Quantization code for each sample based on quantization levels at the left of the graph.

3. **Encoding -** The digitization of the analog signal is done by the encoder. After each sample is quantized and the number of bits per sample is decided, each sample can be changed to an n bit code. Encoding also minimizes the bandwidth used. Note that the number of bits for each sample is determined from the number of quantization levels. If the number of quantization levels is L, the number of bits is n bit = log 2 L.

b. DELTA MODULATION:

Since PCM is a very complex technique, other techniques have been developed to reduce the complexity of PCM. The simplest is delta Modulation. Delta Modulation finds the change from the previous value. **Modulator -** The modulator is used at the sender site to create a stream of bits from an analog signal. The process records a small positive change called delta. If the delta is positive, the process records a 1 else the process records a 0. The modulator builds a second signal that resembles a staircase. The input signal is then compared with this gradually made staircase signal.



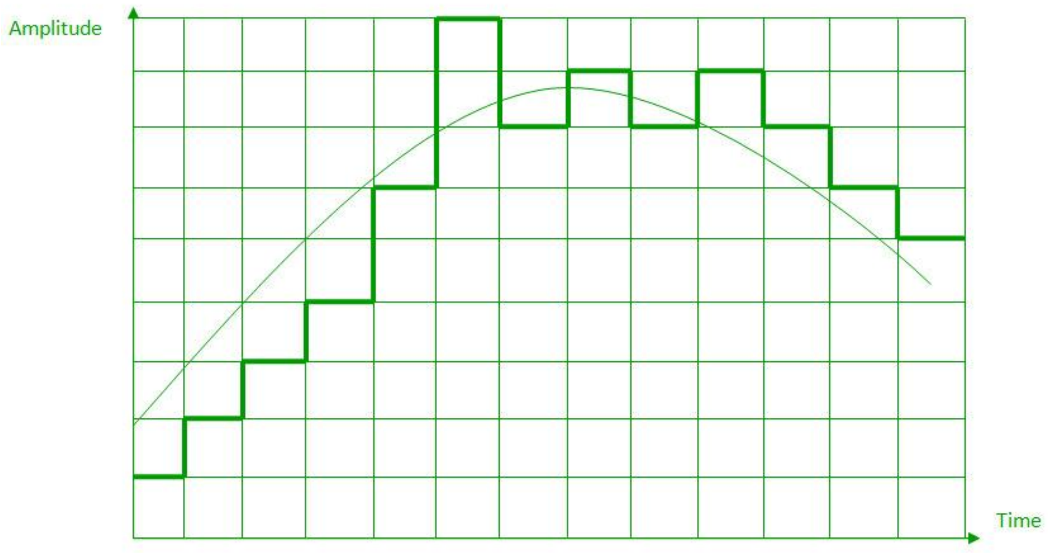
We have the following rules for output:

1. If the input analog signal is higher than the last value of the staircase signal, increase delta by 1, and the bit in the digital data is 1.
2. If the input analog signal is lower than the last value of the staircase signal, decrease delta by 1, and the bit in the digital data is 0.

Demodulator - The demodulator takes the digital data and, using the staircase maker and the delay unit, creates the analog signal. The created analog signal, however, needs to pass through a low-pass filter for smoothing.

c. ADAPTIVE DELTA MODULATION:

The performance of a delta modulator can be improved significantly by making the step size of the modulator assume a time-varying form. A larger step-size is needed where the message has a steep slope of modulating signal and a smaller step-size is needed where the message has a small slope. The size is adapted according to the level of the input signal. This method is known as adaptive delta modulation (ADM).



Applications

- **Digital Signal Processing:** In this process, the systems for processing, storing, or transporting almost any analogue signal into digital format require ADCs to perform well. Let's an example, in TV tuner cards this is use as fast video analog-to-digital converters.

- **Recording Music System:** The modern digital audio workstation-based sound recording and music reproduction technologies both are basically rely heavily on analog-to-digital converters.
- **Scientific Instruments or Projects:** The digital imaging systems are normally use analog-to-digital converters for digitizing the instruments and projects pixels.

Digital to Analog Conversion:

Digital Signal -

A digital signal is a signal that represents data as a sequence of discrete values; at any given time it can only take on one of a finite number of values.

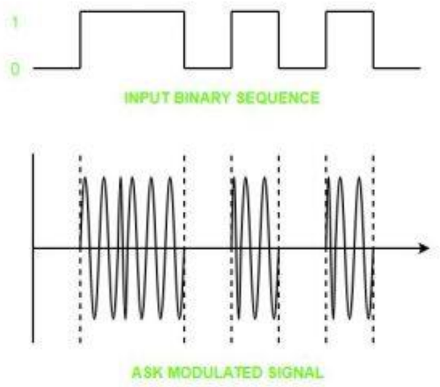
Analog Signal -

An analog signal is any continuous signal for which the time varying feature of the signal is a representation of some other time varying quantity i.e., analogous to another time varying signal.

The following techniques can be used for Digital to Analog Conversion:

1. Amplitude Shift keying -

Amplitude Shift Keying is a technique in which carrier signal is analog and data to be modulated is digital. The amplitude of analog carrier signal is modified to reflect binary data. The binary signal when modulated gives a zero value when the binary data represents 0 while gives the carrier output when data is 1. The frequency and phase of the carrier signal remain constant.



Advantages of amplitude shift Keying:

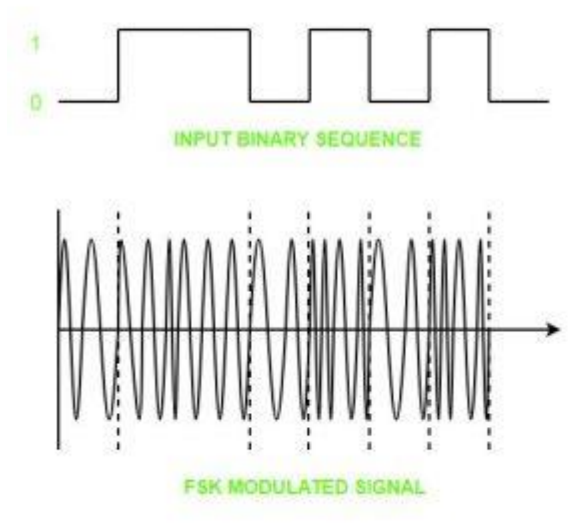
- It can be used to transmit digital data over optical fiber.
- The receiver and transmitter have a simple design which also makes it comparatively inexpensive.
- It uses lesser bandwidth as compared to FSK thus it offers high bandwidth efficiency.

Disadvantages of amplitude shift Keying -

- It is susceptible to noise interference and entire transmissions could be lost due to this.
- It has lower power efficiency.

2. Frequency Shift keying -

In this modulation the frequency of analog carrier signal is modified to reflect binary data. The output of a frequency shift keying modulated wave is high in frequency for a binary high input and is low in frequency for a binary low input. The amplitude and phase of the carrier signal remain constant.



Advantages of frequency shift Keying -

- Frequency shift keying modulated signal can help avoid the noise problems beset by ASK.

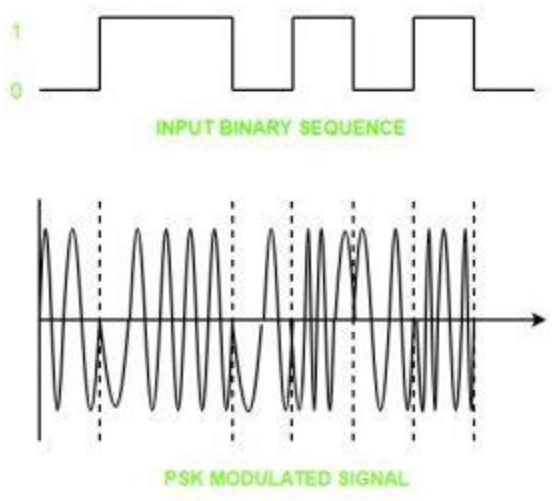
- It has lower chances of an error.
- It provides high signal to noise ratio.
- The transmitter and receiver implementations are simple for low data rate application.

Disadvantages of frequency shift Keying -

- It uses larger bandwidth as compared to ASK thus it offers less bandwidth efficiency.
- It has lower power efficiency.

3. Phase Shift keying:

In this modulation the phase of the analog carrier signal is modified to reflect binary data. The amplitude and frequency of the carrier signal remain constant.



It is further categorized as follows:

1. **Binary Phase Shift Keying (BPSK):** BPSK also known as phase reversal keying or 2PSK is the simplest form of phase shift keying. The Phase of the carrier wave is changed according to the two binary inputs. In Binary Phase shift keying, difference of 180 phase shift is used between binary 1 and binary 0. This is regarded as the most robust digital modulation technique and is used for long distance wireless communication.
2. **Quadrature phase shift keying:** This technique is used to increase the bit rate i.e we can code two bits onto one single element. It uses four phases to encode two bits per symbol.

QPSK uses phase shifts of multiples of 90 degrees. It has double data rate carrying capacity compare to BPSK as two bits are mapped on each constellation points.

Advantages of phase shift Keying -

- It is a more power efficient modulation technique as compared to ASK and FSK.
- It has lower chances of an error.
- It allows data to be carried along a communication signal much more efficiently as compared to FSK.

Disadvantages of phase shift Keying -

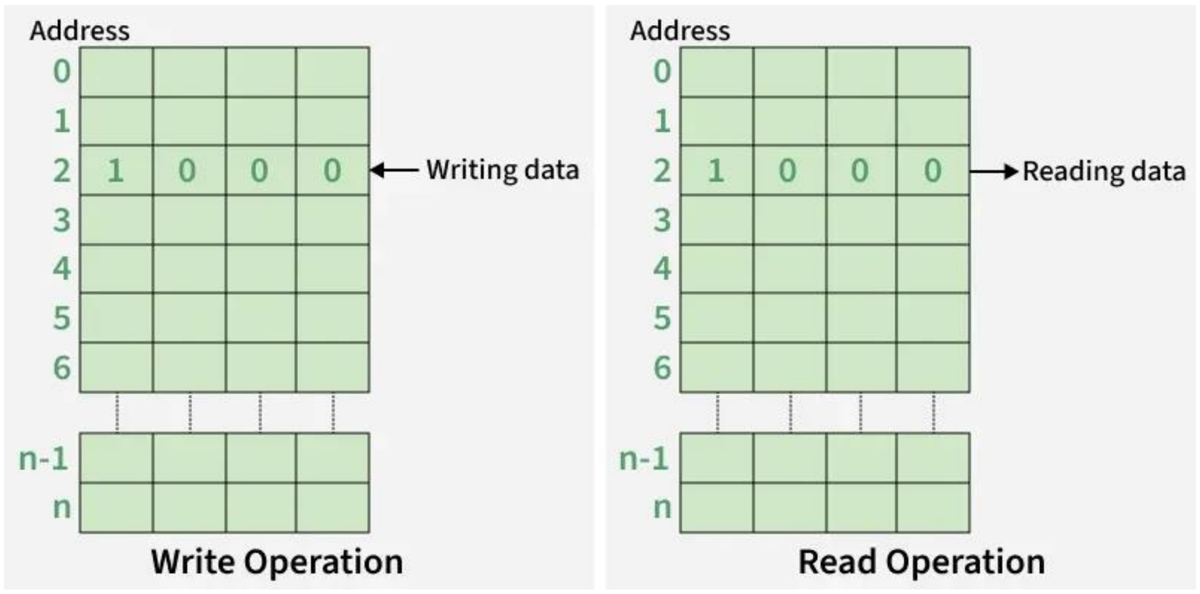
- It offers low bandwidth efficiency.
- The detection and recovery algorithms of binary data is very complex.
- It is a non-coherent reference signal.

UNIT-IV: GENERALMEMORYOPERATIONS: ROM, RAM (static and dynamic), PROM, EPROM, EEPROM, EAROM. IC–logic families: RTL, DTL, TTL logic, CMOS NAND & NOR Gates, CMOS Inverter. Programmable Logic Devices – Programmable Logic Array (PLA), Programmable Array Logic (PAL).

4.1. GENERALMEMORYOPERATIONS:

Memory is an essential component of a computer system, responsible for storing data and instructions needed for processing. It enables the CPU to execute programs efficiently and ensures smooth system operation.

- **Memory Cell:** Smallest unit storing 1 bit of data with a unique address.
- **Word & Byte:** A word is a group of bits; 1 byte = 8 bits.
- **Capacity**: Total number of bits a memory can hold.

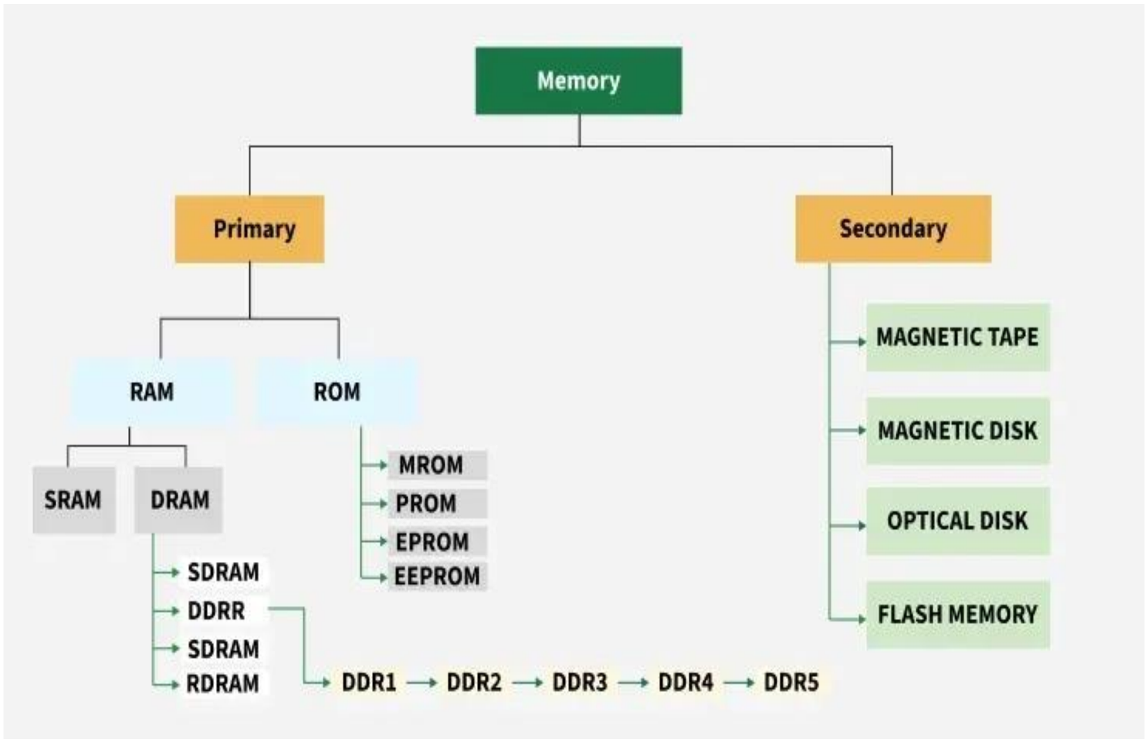


Classification of Memory:

Memory is classified into primary and secondary types based on speed, accessibility, and volatility.

- **Primary Memory**: Directly accessible by the CPU; fast but limited in capacity.

- **Secondary Memory**: Used for long-term storage; slower but larger in capacity.



Read Only Memory (ROM)

Memory plays a crucial role in how devices operate, and one of the most important types is Read-Only Memory (ROM). Unlike RAM (Random Access Memory), which loses its data when the power is turned off, ROM is designed to store essential information permanently.

What is Read-Only Memory (ROM)?

ROM stands for Read-Only Memory. It is a non-volatile memory was used to operate the system. As its name refers to read-only memory, we can only read the stored programs and data.

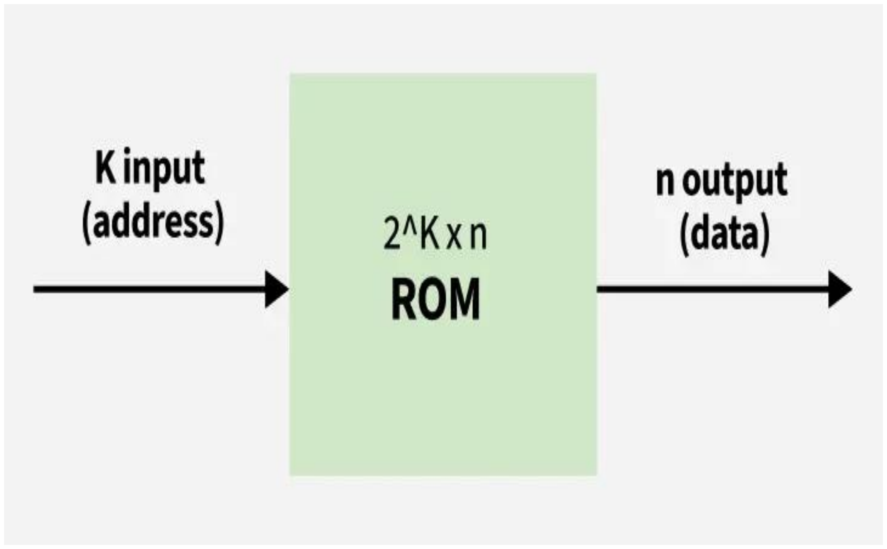
- Information stored in ROM is permanent.
- Information and programs are stored on ROM in binary format (0s and 1s).
- It is used in the start-up process of the computer.

Evolution of ROM Technology

The development of ROM has seen key advancements over the years:

Year	Type	Key Advancement	Use Cases
1956	Mask ROM (MROM)	Hardwired during manufacturing	Early calculators, embedded systems
1956	PROM	One-time programmable by users	Custom firmware
1971	EPROM	Erasable with UV light, reprogrammable	Legacy computer BIOS
1983	EEPROM	Electrically erasable, reusable	Microcontrollers, car key fobs
1984	Flash Memory	Block-level erasure, high speed	USB drives, SSDs, smartphones

Block Diagram of ROM:



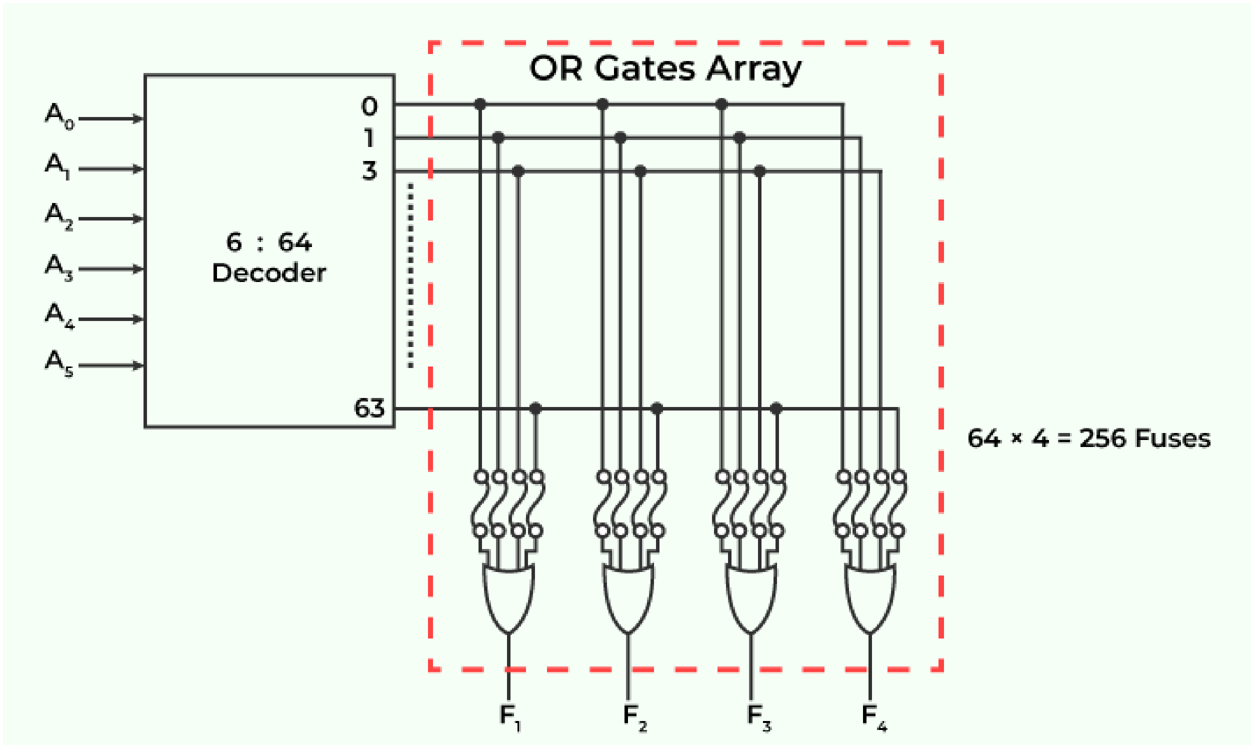
The main purpose of the ROM block diagram is to represent how ROM (Read-Only Memory) works within a computer system. It helps illustrate the flow of data and how the system accesses the stored information.

In a Read-Only Memory (ROM) system, there are k input lines and n output lines. The input address from which we wish to retrieve the ROM content is taken using the k input lines. Since each of the k input lines can have a value of 0 or 1, there are a total of 2 k addresses that can be referred to by these input lines, and each of these addresses contains n bits of information that is output from the ROM.

Internal Structure of ROM

The internal structure of ROM has two basic components:

1. Decoder
2. OR Gates



Internal Construction of 64×4 ROM

A circuit known as a decoder converts an encoded form, such as binary coded decimal, or BCD, into a decimal form. As a result, the output is the binary equivalent of the input. The outputs of the decoder will be the output of every OR gate in the ROM. Let's use a 64 x 4 ROM as an example. This read-only memory has 64 words with a 4-bit length. As a result, there would be four output lines.

Since there are only six input lines and there are 64 words in this ROM, we can specify 64 addresses or minimum terms by choosing one of the 64 words that are available on the output lines from the six input lines. Each address entered has a unique selected word.

Working of ROM:

A small, long-lasting battery within the computer powers the ROM, which is made up of two primary components: the OR logic gates and the decoder. In ROM, the decoder receives binary input and produces decimal output. The decoder's decimal output serves as the input for ROM's OR gates. ROM chips have a grid of columns and rows that may be switched on and off. If they are turned on, the value is 1, and the lines are connected by a diode. When the value is 0, the lines are not connected.

Each element in the arrangement represents one storage element on the memory chip. The diodes allow only one direction of flow, with a specific threshold known as forward break over. This determines the current required before the diode passes the flow on. Silicon-based circuitry typically has a forward break-over voltage of 0.6 V. ROM chips sometimes transmit a charge that exceeds the forward break over to the column with a specified row that is grounded to a specific cell. When a diode is present in the cell, the charge transforms to the binary system, and the cell is "on" with a value of 1.

Types of Read-Only Memory (ROM)

ROM Type	Erasure Method	Reprogrammable	Use Cases/Examples
Mask ROM (MROM)	Hardwired during manufacturing	No	Early embedded systems, firmware

ROM Type	Erasure Method	Reprogrammable	Use Cases/Examples
PROM	One-time programming	No	Custom firmware for specific applications
EPROM	UV light	Yes (with UV)	Firmware updates, legacy computer systems
EEPROM	Electrical signals	Yes	Microcontrollers, BIOS, small firmware updates
Flash Memory	Block-level electrical erasure	Yes	USB drives, SSDs, memory cards, smartphones
PLD-ROM	Configurable logic	Yes	FPGA, CPLD, custom hardware logic

Advantages of ROM:

- Non-Volatile – Retains data without power.
- Security – Prevents unauthorized changes.
- Reliable – Data remains intact over time.
- Cost-Effective – Cheap for large-scale production.
- Fast Access – Quick retrieval of stored data.

Random Access Memory (static and dynamic):

Random Access Memory (RAM) is a type of computer memory that stores data temporarily. When you turn off your computer, the data in RAM disappears, unlike the data on your hard drive, which stays saved. RAM helps your computer run programs and process information faster. This is similar to how the brain's memory helps us remember things.

What is RAM (Random Access Memory)?

It is one of the parts of the Main memory, also famously known as Read Write Memory. Random Access memory is present on the motherboard and the computer's data is temporarily stored in RAM. As the name says, RAM can help in both reading and writing. RAM is a volatile memory, which means, it is present as long as the Computer is in the ON state, as soon as the computer turns OFF, the memory is erased.

How Does RAM Work?

RAM is made up of small transistors and capacitors that store electrical charges representing data bits. Here's how it works:

- Data Storage: RAM temporarily stores data needed by the CPU.
- Volatility: Data is lost when the power is turned off, so it's important to save work to permanent storage (e.g., hard drive or SSD).
- Speed: RAM is much faster than secondary storage, allowing quick access to data.

Features of RAM

- RAM is volatile, meaning that the data is erased when the device is turned off.
- It is referred to as the primary memory of the computer, as it directly supports the CPU during operation.
- RAM is relatively expensive because it allows for fast, direct access to data.
- As the fastest type of memory, RAM serves as internal memory within the computer, enabling quick data retrieval.
- The overall speed of the computer is greatly influenced by the amount of RAM. With less RAM, the computer takes longer to load and may slow down significantly.

Types of RAM

RAM is further divided into two types, SRAM - Static Random Access Memory and DRAM- Dynamic Random Access Memory. Let's learn about both of these types in more detail.

1. SRAM (Static Random Access memory)

SRAM is used for Cache memory, it can hold the data as long as the power availability is there. It is refreshed simultaneously to store the present information. It is made with CMOS technology. It contains 4 to 6 transistors and it also uses clocks. It does not require a periodic refresh cycle due to the presence of transistors. Although SRAM is faster, it requires more power and is more expensive. Since SRAM requires more power, more heat is lost here as well, another drawback of SRAM is that it can not store more bits per chip, for instance, for the same amount of memory stored in DRAM, SRAM would require one more chip.

Function of SRAM

The function of SRAM is that it provides a direct interface with the Central Processing Unit at higher speeds.

Characteristics of SRAM

- SRAM is used as the Cache memory inside the computer.
- SRAM is known to be the fastest among all memories.
- SRAM is costlier.
- SRAM has a lower density (number of memory cells per unit area).
- The power consumption of SRAM is less but when it is operated at higher frequencies, the power consumption of SRAM is compatible with DRAM.

2. DRAM (Dynamic Random Access memory)

DRAM is used for the Main memory, it has a different construction than SRAM, it uses one transistor and one capacitor (also known as a conductor), which is needed to get recharged in milliseconds due to the presence of the capacitor. Dynamic RAM was the first sold memory integrated circuit. DRAM is the second most compact technology in production (the First is

Flash Memory). DRAM has one transistor and one capacitor in 1 memory bit. Although DRAM is slower, it can store more bits per chip, for instance, for the same amount of memory stored in SRAM, DRAM requires one less chip. DRAM requires less power and hence, less heat is produced.

Function of DRAM

The function of DRAM is that it is used for programming code by a computer processor to function. It is used in our PCs (Personal Computers).

Characteristics of DRAM

- DRAM is used as the Main Memory inside the computer.
- DRAM is known to be a fast memory but not as fast as SRAM.
- DRAM is cheaper as compared to SRAM.
- DRAM has a higher density (number of memory cells per unit area)
- The power consumption by DRAM is more

Types of DRAM

- SDRAM: Synchronous DRAM, increases performance through its pins, which sync up with the data connection between the main memory and the microprocessor.
- DDR SDRAM: (Double Data Rate) It has features of SDRAM also but with double speed.
- ECC DRAM: (Error Correcting Code) This RAM can find corrupted data easily and sometimes can fix it.
- RDRAM: It stands for Rambus DRAM. It used to be popular in the late 1990s and early 2000s. It was developed by a company named Rambus Inc. At that time it competed with SDRAM. Its latency was higher at the beginning but it was more stable than SDRAM, consoles like Nintendo 64 and Sony Play Station 2 used that.
- DDR2, DDR3, AND DDR4: These are successor versions of DDR SDRAM with upgrades in performance.

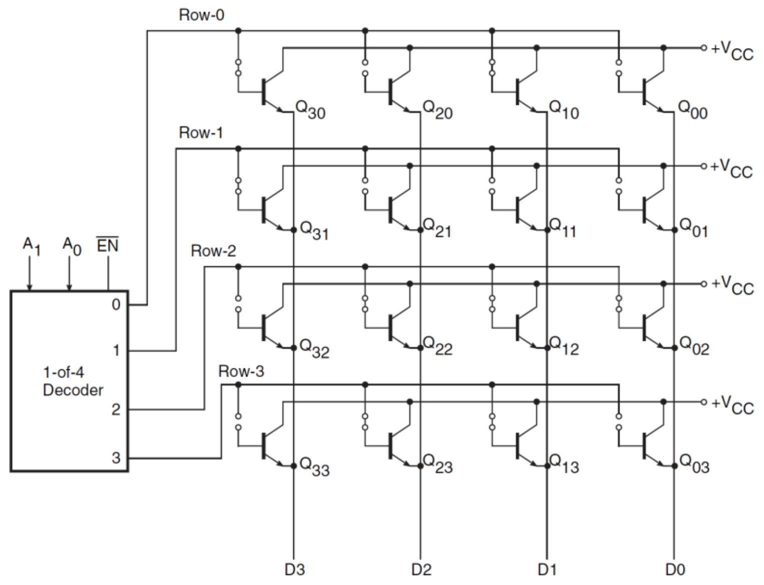
Advantages of RAM

- Speed: RAM is faster than other types of storage like ROM, hard drives or SSDs, allowing for quick access to data and smooth performance of applications.
- Multitasking: More RAM allows a computer to handle multiple applications simultaneously without slowing down.
- Flexibility: RAM can be easily upgraded, enhancing a computer's performance and extending its usability.
- Volatile Storage: RAM automatically clears its data when the computer is turned off, reducing the risk of unwanted data accumulation.

Programmable ROM(PROM):

In the case of PROMs, instead of being done at the manufacturer's premises during the manufacturing process, the programming is done by the customer with the help of a special gadget called a PROM programmer. Since the data, once programmed, cannot be erased and reprogrammed, these devices are also referred to as one-time programmable ROMs.

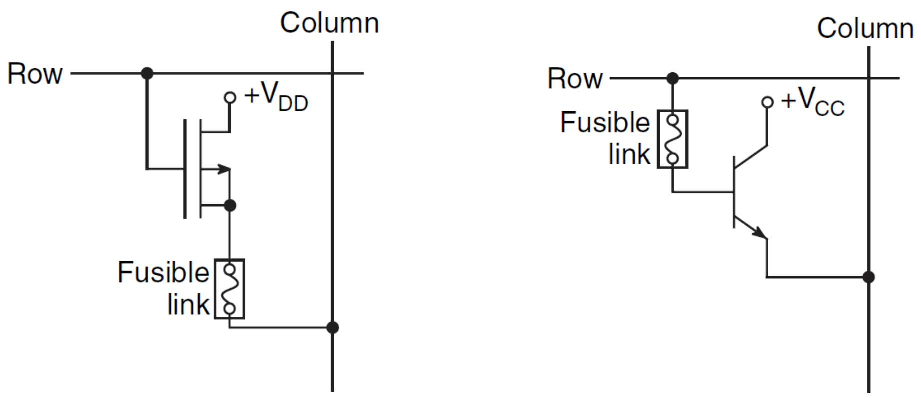
The basic memory cell of a PROM is similar to that of a mask-programmed ROM. Figures show a MOSFET-based memory cell and bipolar memory cell respectively. In the case of a PROM, each of the connections that were left either intact or open in the case of a mask-programmed ROM are made with a thin fusible link, as shown in Fig. 15.18. The different interconnect technologies used in programmable logic devices are comprehensively covered in Chapter 9. Basic fuse technologies used in PROMs are metal links, silicon links and PN junctions. These fusible links can be selectively blown off to store desired data. A sufficient current is injected through the fusible link to burn it open to store '0'. The programming operation, as said earlier, is done with a PROM programmer. The PROM chip is plugged into the socket meant for the purpose. The programmer circuitry selects each address of the PROM one by one, burns in the required data and then verifies the correctness of the data before proceeding to the next address. The data are fed to the programmer from a keyboard or a disk drive or from a



Truth Table

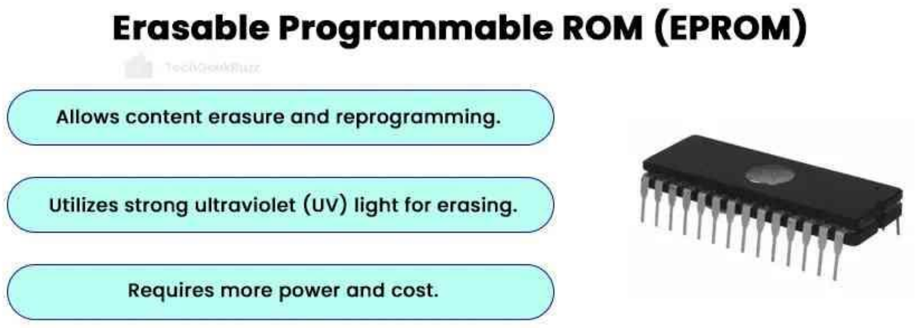
Address		Data			
A_1	A_0	D_3	D_2	D_1	D_0
0	0	1	0	1	0
0	1	1	0	0	0
1	0	0	1	1	1
1	1	0	1	1	0

computer. PROM chips are available in various word sizes and capacities. 27LS19, 27S21, 28L22, 27S15, 24S41, 27S35, 24S81, 27S45, 27S43 and 27S49 are respectively 32×8, 256×4, 256×8, 512×8, 1K×4, 1K×8, 2K×4, 2K×8, 4K×8 and 8K×8 PROMS. The typical access time in the case of these devices is in the range 50–70 ns. MOS PROMs are available with much greater capacities than bipolar PROMs. Also, the power dissipation is much lower in MOS PROMs than it is in the case of bipolar PROMs with similar capacities.



(a) (b)

Erasable Programmable ROM (EPROM)


Erasable Programmable ROM (EPROM)
Allows content erasure and reprogramming.
Utilizes strong ultraviolet (UV) light for erasing.
Requires more power and cost.

Erasable Programmable ROM is a type of read-only memory whose contents can be erased and reprogrammed multiple times without replacing the chip. EPROM uses a strong ultraviolet (UV) source to erase the contents. The rays are passed through a window designed on a memory chip. This process of erasing data is called burning. It leverages a special device called EPROM programmers to erase data that plugs into an EPROM burner.

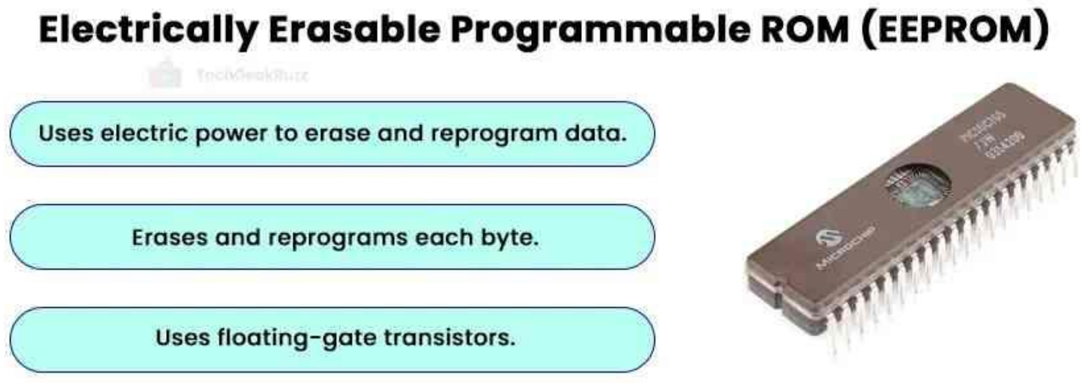
- EPROMs are enclosed in a transparent fused quartz window on the top of the device. This allows the ultraviolet light to fall on the memory chip to erase contents.
- The transparent quartz window is covered with an opaque label to avoid accidental erasure of data on EPROM. This is because natural sunlight also contains some essence of UV light . In the case of strong sunlight, the UV light falling on the transparent window can erase data on EPROM. Hence, it is called UV-EPROM.

- A programmed EPROM can retain its data for 10 to 20 years. Some EPROM chips retain it for more years.
- Another type of EPROM available is one-time EPROM. It does not have a transparent quartz window for UV light to reach the silicon chip. It uses X-Rays to erase contents.

Early PC BIOS chips were EPROM. The transparent window was covered with a BIOS publisher's name, the BIOS revision, and the copyright notice with an adhesive label.

EPROM chips generally consume more power and are more expensive than ROM and PROM. The primary disadvantage of EPROM is that it cannot erase data byte by byte. Instead, it erases or reprograms the entire data at once. Erasing and reprogramming takes a lot of time, depending on the quantity of data.

Electrically Erasable Programmable ROM (EEPROM)


Electrically Erasable Programmable ROM (EEPROM)
Uses electric power to erase and reprogram data.
Erases and reprograms each byte.
Uses floating-gate transistors.

EEPROM or E2PROM is another type of ROM whose contents can be erased and reprogrammed electrically. Unlike EPROM, EEPROM enables users to erase and reprogram individual bytes of data. As a result, it is also referred to as a byte-erasable chip.

- Before the advent of EEPROM, EPROM chips were widely popular. However, its drawbacks led to the development of EEPROM chips. These chips are a collection of floating-gate transistors.

- Originally, EEPROMs were very slow, as they supported single-byte operations. They supported a limited number of erasing and programming.
- Modern EEPROMs are faster due to multi-byte page operations. They now support millions of operations.

The following are the two major types of EEPROM:

- Electrically Alterable Read-Only Memory (EAROM)
- Flash Memory

Electrically Alterable Read-Only Memory (EAROM)

Introduction:

Electrically Alterable Read-Only Memory (EAROM) is a type of non-volatile memory that allows data to be electrically modified after manufacture. Unlike normal ROM, EAROM enables limited rewriting, making it suitable for storing data that changes occasionally.

Characteristics of EAROM

- Non-volatile: retains data even when power is turned off.
- Electrically alterable: contents can be updated electrically, without removing the chip.
- Partial rewriting: small amounts of data (bits/bytes) can be modified.
- Slow write speed: writing requires more time and often a higher voltage.
- Limited write endurance: can only be rewritten a certain number of times.
- Fast read operations: reading is quick and does not require special voltage.

Working Principle

- EAROM uses floating-gate transistors to store charges.
- To write/alter data, a higher electrical voltage is applied to change the charge on the floating gate.
- To read, the memory simply senses whether the floating gate holds a charge (representing 0 or 1).
- Because the write operation stresses the device, frequent updates reduce its lifespan.

Types of EAROM

EAROM is considered a subset of EEPROM technologies. Two categories:

1. Byte-alterable EAROM – only a few bytes can be altered at once.
2. Block-alterable EAROM – alters larger sections but still slower than RAM/flash.

Differences Between EAROM and EEPROM

Feature	EAROM	EEPROM
Alteration	Small, selective portions	Entire bytes or blocks
Speed	Slower writes	Faster writes
Usage	Rarely updated data	Moderately updated data
Write voltage	Higher	Lower
Endurance	Lower	Higher

Applications of EAROM

EAROM is used where data is:

- Semi-permanent
- Needs occasional updates
- Must be retained without power

Typical uses:

- Calibration and tuning information

- System configuration parameters
- Security and identification codes
- Analog system settings
- Digital watches, calculators (older devices)
- Microcontroller parameter storage

Advantages

- Data remains even when the system is powered off
- Allows electrical modification (no physical replacement)
- Supports localized changes (bit/byte-level)
- Reliable for long-term storage of stable data

4.2. IC–logic families:

Logic families are different types of technologies being used to build different logic gates. Logic gates are digital circuits that perform basic logic operations like AND, OR, NOT, NAND, and NOR. In other words, it is a group of compatible ICs with the same logic levels and supply voltages fabricated for performing various logical functions. Here, when we say that ICs have the same logic level, we are referring to two types of logic levels that exist -

- In positive logic: 0 is formed by a low voltage level, and a high voltage level forms 1. It means the ON state refers to high voltage as input or output while OFF means low voltage as input or output.
- In negative logic: 0 is formed by a high voltage level, and 1 is formed by a low voltage level. Here, the situation is reversed to that of positive logic. ON means a low voltage input or output while OFF means high voltage as input or output.

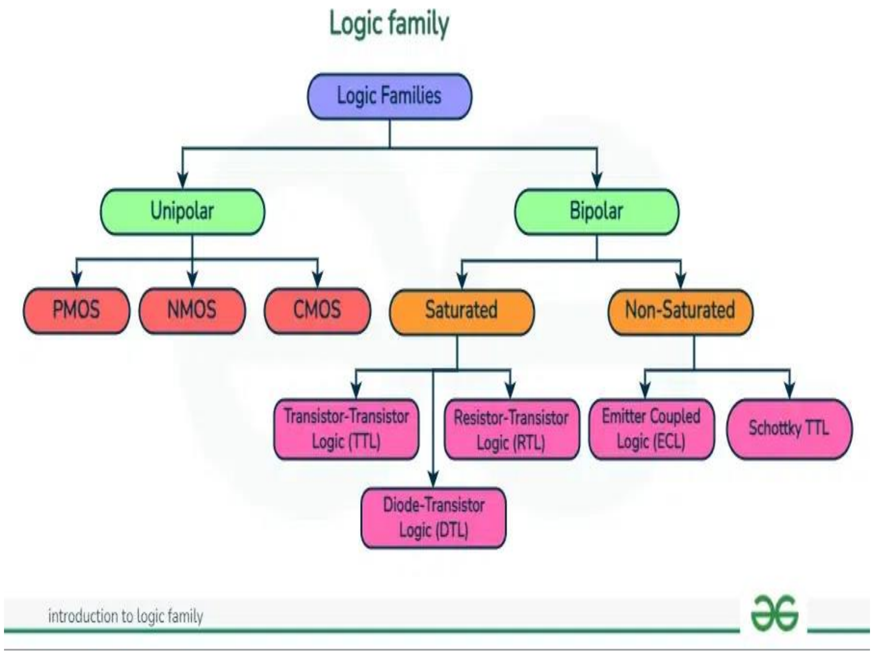
Logical functions are the logical gate operations. The most common logical functions are - AND, OR, NOT, NAND, NOR, XOR.

Logic families are fabricated using various semiconductor technologies utilizing diodes and transistors as switching components. Diodes are simple switching elements having two states, ON as when in forward bias and OFF as when in reverse bias.

In the same way, transistors are elements having three terminals- collector, base, and emitter and they utilize base voltage to switch their states by allowing them to flow current from collector to emitter. Various factors are responsible for choosing which logic families can be used for the given specific scenarios which include switching speed, fan-out capabilities, power consumption, etc.

Classification of Logic Families

Logic families can be broadly categorized as per the following diagram

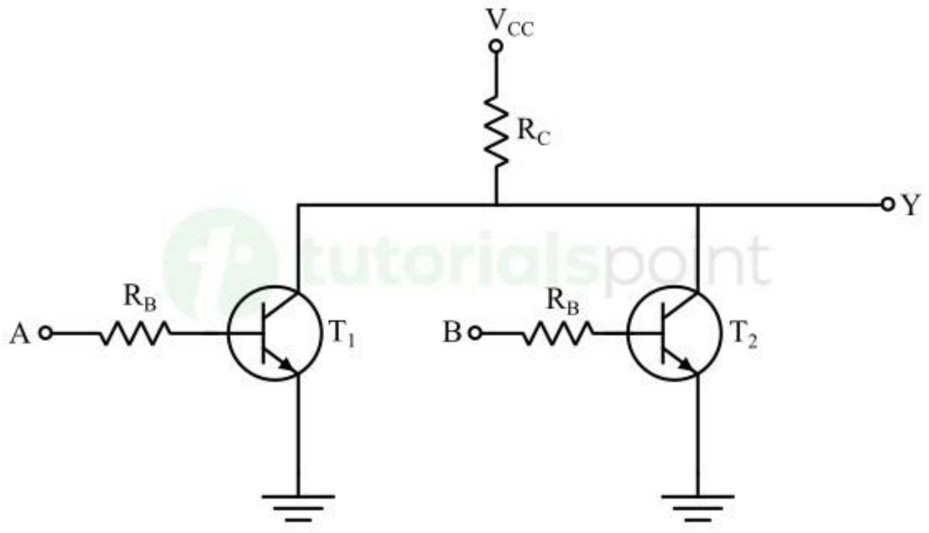

Logic family
Logic Families
Unipolar
Bipolar
PMOS
NMOS
CMOS
Saturated
Non-Saturated
Transistor-Transistor Logic (TTL)
Resistor-Transistor Logic (RTL)
Diode-Transistor Logic (DTL)
Emitter Coupled Logic (ECL)
Schottky TTL
introduction to logic family

Resistor-Transistor Logic (RTL)

As the name implies, this logic family utilizes resistors and transistors as their key elements. In the RTL family, the transistors operate in the cut-off or saturation regions depending on the input voltage applied to them. The RTL family was one of the earliest logic families used in the field of digital electronic design.

In short, in the RTL family, the logic circuits are designed using resistors and transistors only.

For example, the circuit of a two-input resistor-transistor logic NOR gate is shown in the following figure. Here, A and B are the inputs and Y is the output of the gate.



The operation of this RTL NOR gate for different input combinations is highlighted in the following table

Input A	Input B	Transistor T_1	Transistor T_2	Output Y
0	0	Off	Off	1
0	1	Off	On	0
1	0	On	Off	0
1	1	On	On	0

Similarly, we can also implement other types of logic gates as well.

Advantages of RTL Family

The following are some key advantages of resistor-transistor logic family −

- Electronic circuits designed using RTL logic family are simple in design, as they consist of a minimum number of resistors and transistors.
- Circuits manufactured in RTL family are less expensive. These circuits consume less amount of power than circuits implemented in other logic families.

Disadvantages of RTL Family

The following are some major drawbacks of resistor-transistor logic families −

- RTL circuits have low noise margin. This limitation makes them susceptible to noise and interference.
- These circuits have poor fan-out.
- RTL circuits are slower in operation due to high propagation delay.
- RTL family is not suitable for designing complex circuits due to some practical limitations in terms of design scalability and performance.

Applications of RTL Family

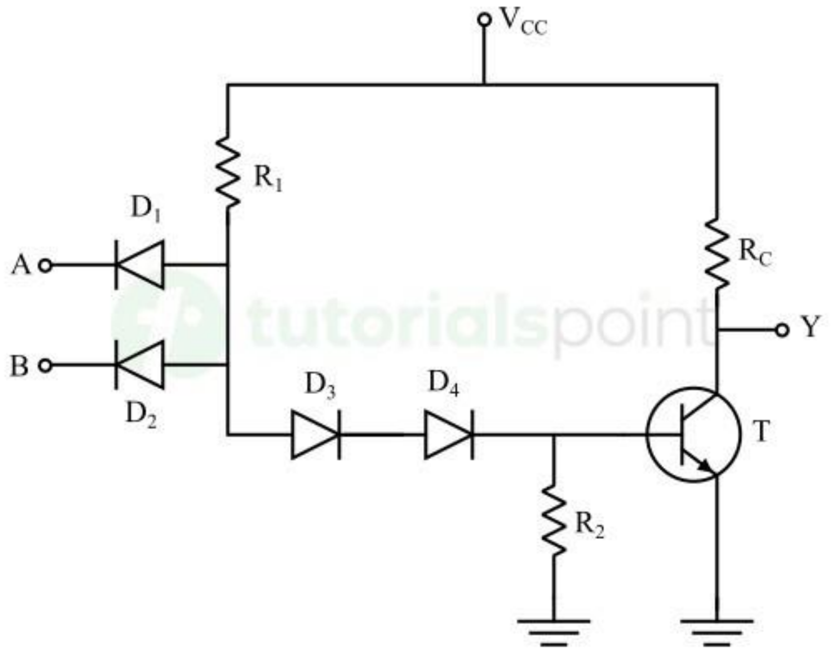
Resistor-Transistor Logic (RTL) family finds some limited applications in the field of digital electronics. Some common applications of RTL family are listed below −

- RTL family is cost-effective and easy to understand and design. For this reason, it is widely used for educational purposes in labs and classrooms to demonstrate digital electronic concepts to students.
- RTL family is also used to design circuits for low-frequency control applications. Due to simplicity and ease of implementation, RTL family can be used for prototyping and experimental purposes.

Diode Transistor Logic (DTL)

In diode-transistor logic (DTL) family, the diodes and transistors are the key elements combinedly used to implement digital logic functions.

The following example circuit demonstrates the electronic circuit design in DTL family.



It is a two-input NAND gate. Where, A and B are inputs of the NAND gate and Y is the output of the gate.

The operation of this two-input NAND gate is explained in the following truth table −

Input A	**Input B**	**Diode D_1**	**Diode D_2**	**Transistor T**	**Output Y**
0	0	Forward biased	Forward biased	Off	1

0	1	Forward biased	Reverse biased	Off	1
1	0	Reverse biased	Forward biased	Off	1
1	1	Reverse biased	Reverse biased	On	0

We can also implement other types of logic circuits using the diode-transistor logic family.

Advantages of DTL Family

The following are some key advantages of diode-transistor logic family −

- DTL circuits are easy and simple to design and implement, as they consist of only diodes, transistors, and resistors.
- DTL circuits are cost-effective as they use basic electronic components like diodes and transistors which are generally cheap.
- DTL circuits have good noise immunity. Hence, these circuits are relatively less susceptible to noise and interference than some other types of logic families.
- DTL circuits have high fan-out. The power dissipation in DTL circuits is comparatively low.

Limitations of DTL Family

Apart from the advantages given above, the DTL circuits also have some disadvantages which are listed below −

- DTL family circuits require higher amount of power as compared to other logic families.
- DTL circuits consist of a greater number of elements than other types of logic families.

- DTL circuits have a moderate speed of operation. This is due to high propagation delay.
- DTL circuits are not suitable to design more complex digital circuits due to increased complexity and size of the circuit.

Applications of DTL Family

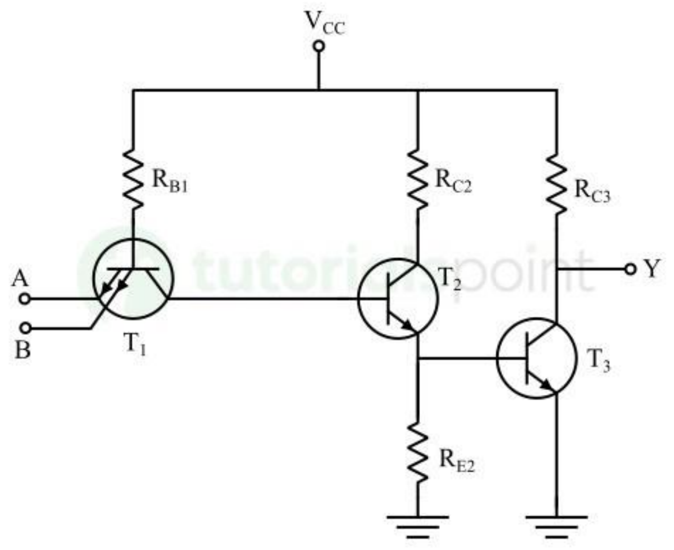
The following are some common applications of diode-transistor logic family −

- DTL family was popular in early digital computers and other digital systems.
- These days, DTL circuits are mainly used for educational purposes to explain the implementation of digital logic designs to students.
- DTL circuits are used to design custom electronic projects.

Transistor-Transistor Logic (TTL)

Transistor-Transistor Logic (TTL) is one of the most popular logic family in the field of digital electronics. In this logic family, the transistor is the key functional element which is operated as a switch to perform the logical operations.

Let us now understand how we can design a logic circuit in TTL family. The following figure shows a two-input NAND gate −



Here, A and B are the input terminals and Y is the output terminal. The operation of this circuit is summarized in the following table.

Input A	**Input B**	**Emitter Junction of Transistor T_1**	**Emitter Junction of Transistor T_2**	**Transistor T_2 and T_3**	**Output Y**
0	0	Forward biased	Forward biased	Off	1
0	1	Forward biased	Reverse biased	Off	1
1	0	Reverse biased	Forward biased	Off	1
1	1	Reverse biased	Reverse biased	On	0

In the same manner, we can also design other logic gates in the transistor-transistor logic (TTL) family.

Advantages of TTL Family

Here are some of the key advantages of TTL family −

- TTL circuits have high speeds of operation and hence well-suited to use in high-speed digital systems.
- TTL circuits are standardized to make them compatible with a variety of digital circuits and systems.
- TTL circuits have good noise immunity. Thus, they are suitable to use in noisy environments.

Disadvantages of TTL Family

Although, TTL circuits have several advantages as listed above. But they also have some disadvantages which are given below −

- TTL circuits consume more power than other types of logic families. This limitation makes the TTL circuits less energy efficient.

- TTL circuits generate significant heat during operation and this is due to high power consumption. Thus, a proper heat management system is required.
- TTL logic levels are relatively strict, requiring specific voltage levels for proper operation. This can sometimes lead to compatibility issues with other logic families.
- TTL circuits have a significant propagation delay that limit their use in certain high-speed systems.

Applications of TTL Family

Transistor-Transistor Logic (TTL) family are widely used in various applications in the field of digital electronics. Some of the common applications of TTL family are listed below −

- TTL circuits are widely used in digital computers, memory units, CPUs, etc.
- TTL circuits are also used in embedded systems for different purposes such as interfacing with sensors, processing data in real-time applications, and more.
- In communication systems, TTL circuits are for signal conditioning, protocol handling, data processing, etc. TTL circuits are commonly used in a variety of testing and measuring instruments.

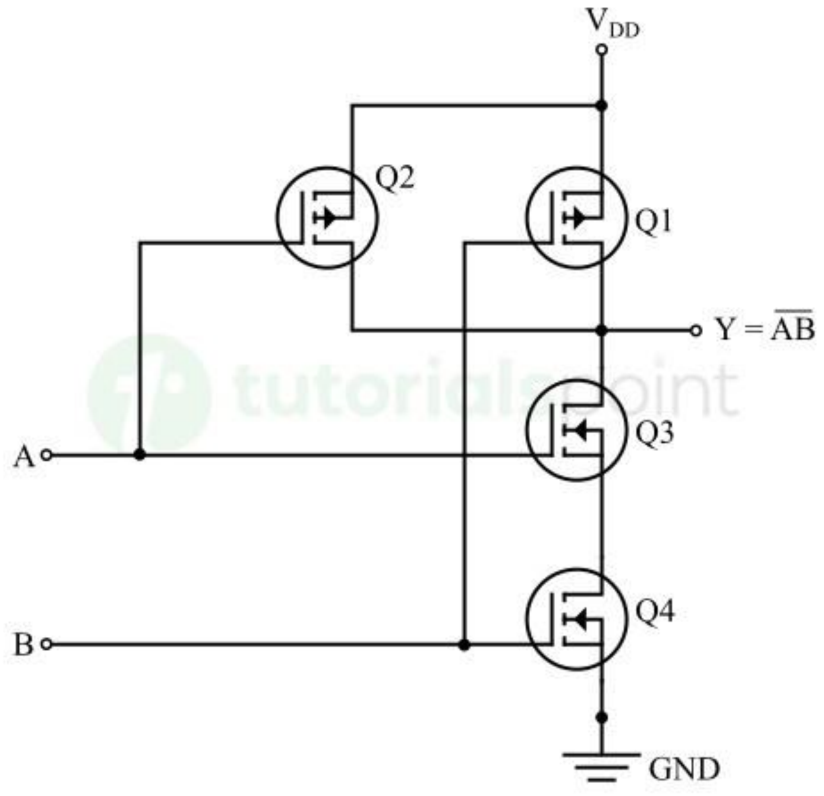
CMOS NAND & NOR Gates:

NAND Gate Using CMOS Technology:

The NAND gate can be implemented in CMOS technology by using PMOS and NMOS transistors. The circuit diagram of a two input NAND gate in CMOS technology is shown in the following figure. It consists of two PMOS transistors Q1 and Q2 and two NMOS transistors Q3 and Q4. The PMOS transistors are connected in parallel between the power supply V_{DD} and the output terminal Y. Similarly, the NMOS transistors are connected in series between the output terminal Y and the ground terminal GND. Now, let us understand the operation of this CMOS NAND gate.

Case 1: When Input A is Low and Input B is Low

In this case, when both inputs A and B are low, the PMOS transistors Q1 and Q2 are ON and the NMOS transistors Q3 and Q4 are OFF. Hence, there is a closed path between the supply voltage V_{DD} and the output terminal Y.



Thus, the output Y will be connected to the voltage level V_{DD}. Also, there is no path between the output terminal and the ground terminal as both NMOS transistors are OFF. Under this ondition, the output line will maintain the voltage level at V_{DD}, which indicates the output High.

Thus, when A = 0 and B = 0, then Y = 1

Case 2: When Input A is Low and Input B is High

In this case, the PMOS transistor Q1 will be ON while the PMOS transistor Q2 will be OFF. The NMOS transistor Q3 will be OFF and the NMOS transistor Q4 will be ON.

For this switching condition of the CMOS transistors, the power supply V_{DD} will get a path to the output terminal through the PMOS transistor Q1. Since, the NMOS transistor Q3 and Q4 are connected in series and the NMOS transistor Q3 is OFF. Hence, there is no path between the output terminal and the ground terminal. Therefore, the output terminal Y maintain the voltage level at V_{DD} and results in a High output.

Thus, when A = 0 and B = 1, then Y = 1

Case 3: When Input A is High and Input B is Low

In this case, the PMOS transistor Q1 will be OFF and the PMOS transistor Q2 will be ON. The NMOS transistor Q3 will be ON and the NMOS transistor Q4 will be OFF.

Under this switching condition of the CMOS transistors, the output terminal will connect to the power supply through the PMOS transistor Q2. Since, both NMOS transistors are connected in series and the NMOS transistor Q4 is OFF. Hence, there is no path between the output terminal and the ground terminal. Therefore, the output line will maintain the voltage level at V_{DD} and results in a High output.

Thus, when A = 1 and B = 0, then Y = 1

Case 4: When Input A is High and Input B is High

In this case, both PMOS transistors Q1 and Q2 will be OFF and both NMOS transistors will be ON. In this case, there is no path between the output terminal and the power supply V_{DD}, but there is a direct path between the output terminal and the ground terminal. This results in a ground voltage level at the output terminal and produces a Low output.

Hence, when A = 1 and B = 1, then Y = 0

The operation of this CMOS NAND gate is shown in the following truth table

Inputs		**Output**
A	B	Y
Low (0)	Low (0)	High (1)
Low (0)	High (1)	High (1)

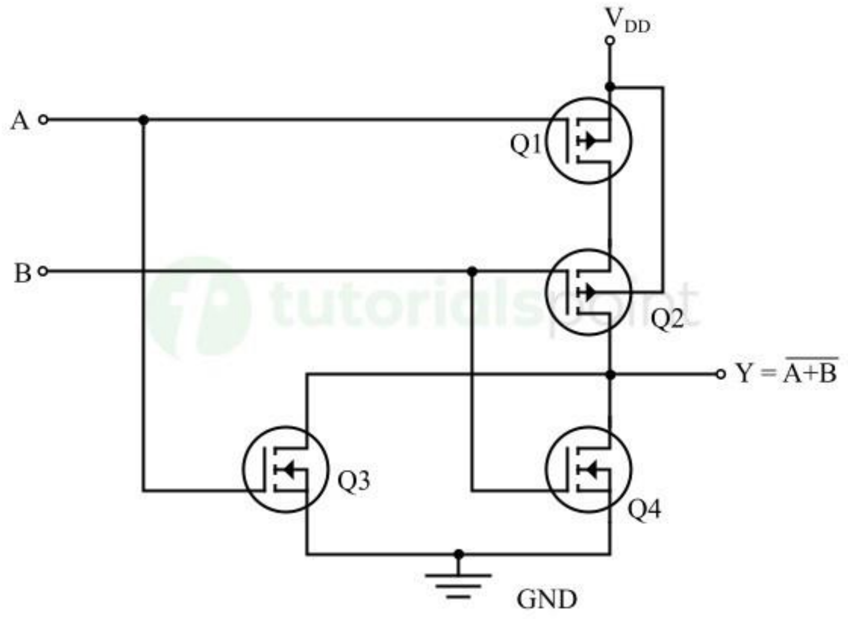
High (1)	Low (0)	High (1)
High (1)	High (1)	Low (0)

This is all about NAND gate implementation using CMOS technology and its operation for different input combinations.

Let us now discuss the implementation and operation of NOR gate using CMOS technology.

NOR Gate Using CMOS Technology:

Similar to CMOS NAND gate, we can also design a NOR gate using PMOS and NMOS transistors. The circuit diagram of a two input NOR gate using CMOS technology is shown in the following figure −



This CMOS NOR gate is designed by using two PMOS transistors Q1 and Q2 and two NMOS transistor Q3 and Q4. Where the PMOS transistors are connected in series between the supply voltage V_{DD} and the output terminal Y. The NMOS transistors are connected in parallel between the output terminal Y and the ground terminal GND.

Now, let us understand how does this CMOS circuit operate as a two input NOR gate.

Case 1: When Input A is Low and Input B is Low

In this case, both PMOS transistors Q1 and Q2 will be ON and both NMOS transistors Q3 and Q4 will be OFF.

Under this switching condition of the CMOS transistors, there is a path between the supply voltage V_{DD} and the output terminal Y through the ON PMOS transistors. But there is no path between the output terminal Y and the ground terminal GND. This maintains the output at the voltage level V_{DD} and hence the output will be High.

Thus, when A = 0 and B = 0, then Y = 1

Case 2: When Input A is Low and Input B is High

In this case, the PMOS transistor Q1 is ON, the PMOS transistor Q2 is OFF, the NMOS transistor Q3 is OFF, and the NMOS transistor Q4 is ON.

Since, the PMOS transistors Q1 and Q2 are connected in series and the transistor Q2 is OFF. Thus, there is no path between the power supply V_{DD} and the output terminal Y. But there is a connection between the output line Y and the ground terminal GND through the ON NMOS transistor Q4. This sets the output terminal to ground voltage and makes the output Low.

Therefore, when A = 0 and B = 1, then Y = 0

Case 3: When Input A is High and Input B is Low

In this condition, the PMOS transistor Q1 is OFF, the PMOS transistor Q2 is ON, the NMOS transistor Q3 is ON, and the NMOS transistor Q4 is OFF.

In this case, there is no closed path between the power supply V_{DD} and the output line Y due to OFF PMOS transistor Q1. But there is a closed between the output line Y and the ground terminal GND through the ON NMOS transistor Q3. Hence, the output terminal is connected to the ground potential and makes the output Low.

Thus, when A = 1 and B = 0, then Y = 0

Case 4: When Input A is High and Input B is High

In this case, both PMOS transistors Q1 and Q2 are OFF and both NMOS transistors Q3 and Q4 are ON. Under this condition, there is no path between the supply voltage V_{DD} and the output terminal Y. But there is a closed path between the output terminal Y and the ground terminal GND. This maintains the output line at ground voltage level and hence the output will be Low.

Thus, when A = 1 and B = 1, then Y = 0

This complete operation of the CMOS NOR gate can be summarized in the form of a truth table which is given below.

Inputs		Output
A	B	Y
Low (0)	Low (0)	High (1)
Low (0)	High (1)	Low (0)
High (1)	Low (0)	Low (0)
High (1)	High (1)	Low (0)

Advantages of NAND and NOR Gates using CMOS Technology

NAND and NOR gates implemented in CMOS technology offer several benefits over other technologies. Some of the key advantages of CMOS NAND and NOR gates are listed here −

- CMOS NAND and NOR gates consume relatively low power. This advantage makes these logic gates well-suited to use in battery powered devices.
- NAND and NOR gates designed using CMOS technology have high immunity against noise and interference. They can be designed to have a wider range of operating voltage.
- The CMOS technology offers high-density integration that allows for implementing a large number of NAND and NOR gates on a single chip. These gates provide

symmetrical output characteristics that allow them to integrate with different types of digital circuits seamlessly.

- CMOS technology is one of the well-established, mature, and cost-effective semiconductor manufacturing technology. Hence, the CMOS NAND and NOR gates are relatively easy to manufacture and cost effective.

Applications of CMOS NAND and NOR Gates

The CMOS NAND and NOR gates are widely used in the following applications due to their benefits and versatility −

- CMOS NAND and NOR gates are widely used in the logic circuit designs to perform logical operations.
- In digital systems, the CMOS NAND and NOR gates are used to implement arithmetic circuits like adders, subtractors, multipliers, etc.
- They are also used in memory units to implement memory cell structures.
- CMOS NAND and NOR gates are also used to design multiplexers and demultiplexers.
- Some other common applications of CMOS NAND and NOR gates include digital signal processing, digital timing circuits, analog to digital conversion, digital communication, etc.

CMOS Inverter

The CMOS inverter is crucial in electronics, employed in memory chips and microprocessors among others. It generates complementary outputs whenever there are input signals through it. These inverters allow flexibility among circuit designers because they can be classified into many categories depending on their abilities and arrangements.

For their functionality, NMOS and PMOS transistors should counteract each other whereby they may react to input voltages by oscillating between conductive and non-conductive modes. Silicon substrates are host to connecting metal layers, transistors, and resistors in CMOS inverter components.

What is CMOS Inverter?

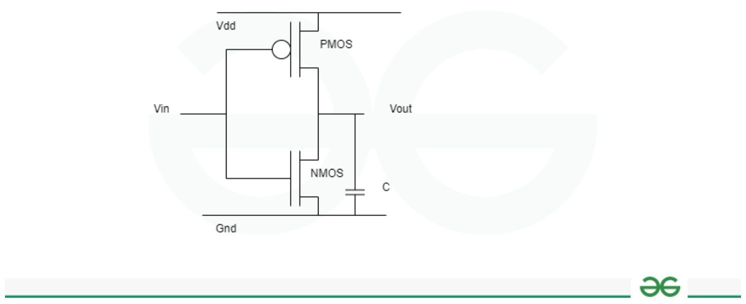
CMOS, short for Complementary Metal-Oxide-Semiconductor, is the type of silicon chip electronics technology that has been used in many devices, which handle signal passing in their circuits.

For many electronic devices, a CMOS serves as the brain. It is a small but very significant part that regulates the flow of signals through circuits. CMOS helps in controlling the actions of electronic devices based on the signals they receive, much as our brain instructs our bodies on what to do.

Types of CMOS Inverter

- Conventional CMOS Inverter: A single series-connected NMOS and PMOS transistor forms up a conventional CMOS inverter.
- Static CMOS Inverter: In this kind of inverter, the circuit always contains both PMOS and NMOS transistors. Its low power usage and high noise immunity are a result of this condition.
- Dynamic CMOS Inverter: These inverters allow higher speeds but may use more power than static CMOS inverters since they use a clock signal to control the behavior of the transistors.
- Pseudo-NMOS Inverter: This configuration is simpler, but it requires more power due to employing only an NMOS transistor for pull-down and a resistor for pull-up.

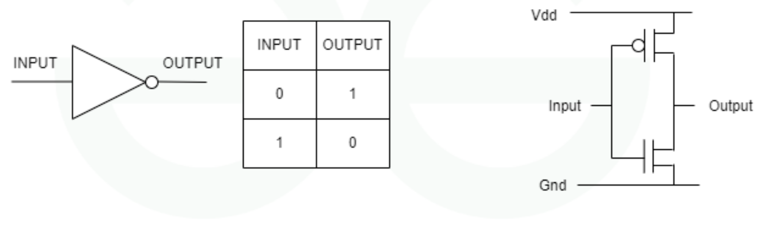
Schematic Diagram of CMOS Inverter



- The diagram is shown with two transistors tied up in series between the ground and the power source in it.
- Unlike PMOS transistors where the source is linked to power supply and the drain is tied to the output , an NMOS transistor's source is directly linked with earth while its drain is linked with output .
- The input to the inverter is actually a mutual connection between the gates of the two transistors.
- When the input level reaches logic level 0, the NMOS transistor is on which causes PMOS transistor to be cut-off therefore causing high output. On the contrary, in a scenario where NMOS transistor goes off, inverter's output is low due to high input voltage.

Operations of CMOS Inverter

- In order to create a CMOS inverter, one has to merge two types of transistors : PMOS and NMOS respectively.
- The process requires that you connect them in this way, thus, there must be one NMOS and one PMOS transistor consisting a CMOS inverter made on the same silicon chip.
- The input terminal is made up of NMOS and PMOS transistors that have an input voltage applied to their gates together with an output terminal which is connected to drains of the two transistors.
- The sources of PMOS and NMOS transistors are attached to distinct power supply voltages unlike in the case of the other terminals. Specifically speaking, the NMOS transistor is connected to ground (0 V) while its counterpart, PMOS transistor is connected to a positive power supply voltage (Vdd).



INPUT	OUTPUT
0	1
1	0

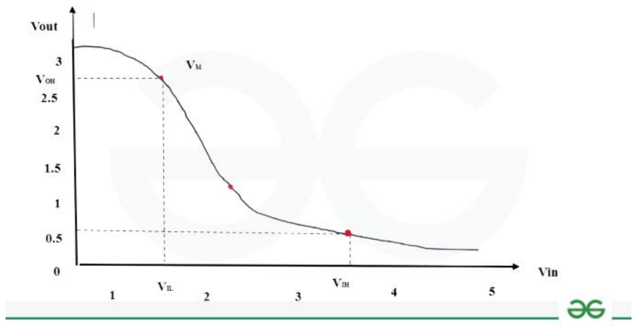
Working of CMOS Inverter

- Input High (Logic 1): An NMOS transistor is turned on by input of high voltage (logic 1) while a PMOS transistor is turned off there. When these two things happen, the output voltage (logic 0) is lowered through reduced resistance path between an output terminal and ground.
- Input Low (Logic 0): In contrast, when a low voltage (logic 0) is provided to the input terminal, the NMOS transistor switches off and the PMOS transistor conducts. The output voltage (logic 1) rises as a result of the low resistance path that exists between the output terminal and the positive power supply voltage (VDD).
- The CMOS inverter operates more easily because of the complimentary characteristics of the NMOS and PMOS transistors. Because one of the transistors conducts while the other is off depending on the input voltage, the output of the transistors is inverted with respect to the input signal.
- CMOS inverters offer very low static power dissipation (no DC current flows between VDD and ground while the input is at a constant logic level) as a result of this complimentary pairing.

Characteristics of CMOS Inverter

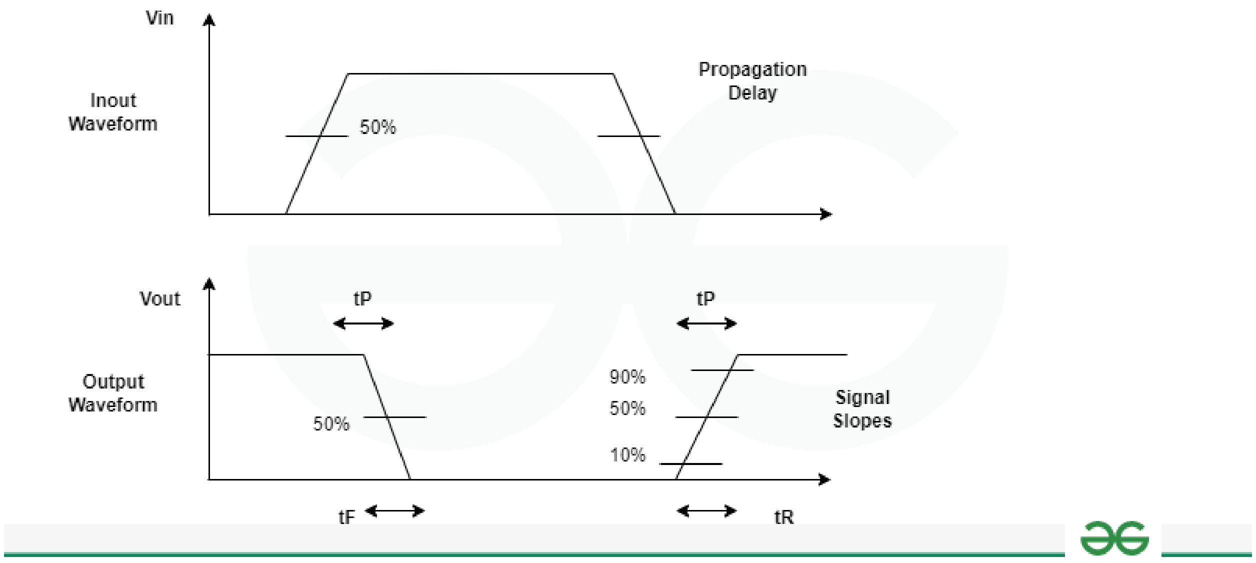
Inverter Static Characteristics (VTC):

- When the circuit is static, VTC shows how the output voltage changes with the input voltage. The threshold voltage is the point at which the output states abruptly change from high to low. The production decreases below this level and stays high above it. To ensure balanced functioning, CMOS inverters should ideally provide a symmetric VTC around the midpoint voltage.
- The VTC is like an upside down step function because of the increased accuracy in turning ON/OFF transitions. Quality in the transition region is suggested by the existence of well-defined slopes that enable precise switching. By comparing the lowest input value within each ON or OFF operation region with the highest output value, one could calculate the tolerance for noise.



Inverter Static Characteristics or Inverter Dynamic Characteristics

- These show how the inverter responds to changes in input signal over time, which is crucial for circuit speed and efficiency. Effective charging and discharging of the output node is possible in CMOS inverters due to the quick switching speeds offered by complementary NMOS and PMOS transistors.
- Rise Time or tR: the time for a signal to go from 10% to 90% of its final value.
- Peak Time or tP: Time taken for a response to reach maximum value.
- Fall Time or tF: the time taken for 90%-10% drop in the signal depending on its value.



Components of CMOS Inverter:

1. NMOS Transistor: An NMOS transistor consists of a gate, source, and drain terminals that make it an N-channel metal-oxide-semiconductor transistor. When the gate terminal is connected positively to the source terminal, an NMOS transistor conducts.
2. PMOS Transistor: A P-channel metal-oxide-semiconductor transistor which has (the) gate, source and drain terminals similar to those of (its) counterpart (an) NMOS transistor. Commonly referred to, (some) PMOS transistor conducts when a negative voltage.
3. Substrate: Silicon is used in building the PMOS as well as NMOS transistors. It ensures that they are both mechanically stable and electrically isolated.
4. Interconnects: To create the required circuit configuration, the various transistor terminals are connected via interconnecting metal layers. Within the CMOS inverter, these metal layers guarantee appropriate signal routing and electrical connectivity.
5. Gate Connection: The gate of the PMOS and NMOS transistors is connected to the input terminal of the CMOS inverter.

6. Power Supply Connections: PMOS and NMOS transistors are usually attached to different points on the power supply devices. Whereas PMOS transistors are normally attached to a positive power supply voltage, known as VDD, NMOS transistors often use ground (0 volts).

Important Terminologies

1. Threshold Voltage (Vth): The threshold voltage, Vth, is the voltage required to switch on a transistor while it is still the least compared to the other input voltages.
2. Propagation delay: It is the combination of both transistor switching delay and signal propagation time before an output responds or changes when there is a change of input.
3. Noise Margin: Noise Margin is the difference between the maximum input voltage for a valid HIGH and the lowest input voltage for a valid LOW . It determines the tolerance of an IC to noise.
4. Power Consumption: Power Consumption rate is low in CMOS inverters because their design characteristics make them consume less of electrical power than any other type of inverter.

Applications of CMOS Inverter

- The fundamental components of different digital logic gates used in processors, memory circuits, and other digital systems are CMOS inverters.
- They are used to buffer and distribute clock signals among integrated circuits in clock distribution networks.
- In mixed-signal circuits, CMOS inverters can be utilized for level shifting, which involves converting between various voltage levels.
- They are employed in oscillator circuits in digital systems to produce timing references and clock signals.
- Frequency divider circuits use CMOS inverters to divide the frequency of input signals.
- They are used in circuits with voltage regulators to regulate power supplies steadily.

- The CMOS inverters are used to drive the LEDs (Light Emitting Diodes) in display panels, and indicator circuits too.
- LEDs are integrated in the key regions of computer-memory known as static-random-access memory (SRAM) cells.
- To condition and process signals from different sensors, sensor interface circuits use CMOS inverters.
- They optimize power consumption by controlling and managing the distribution of power in electronic devices.

4.2. Programmable Logic Devices (PLDs):

Programmable Logic Devices (PLDs) are a collection of integrated circuits which are configured to perform various logical functions. PLDs play an important role in the field of engineering and technology, as they form the basis of innovation and support engineers to develop automated digital systems to improve process flexibility and efficiency. Here, "programmable" means defining a function that can be performed multiple times without human intervention.

Programmable Logic Devices (PLDs) are the integrated circuits. They contain an array of AND gates & another array of OR gates. There are three kinds of PLDs based on the type of array(s), which has programmable feature.

- Programmable Read Only Memory
- Programmable Array Logic
- Programmable Logic Array

The process of entering the information into these devices is known as programming. Basically, users can program these devices or ICs electrically in order to implement the Boolean functions based on the requirement. Here, the term programming refers to hardware programming but not software programming.

4.2.1. Programmable Logic Array (PLA):

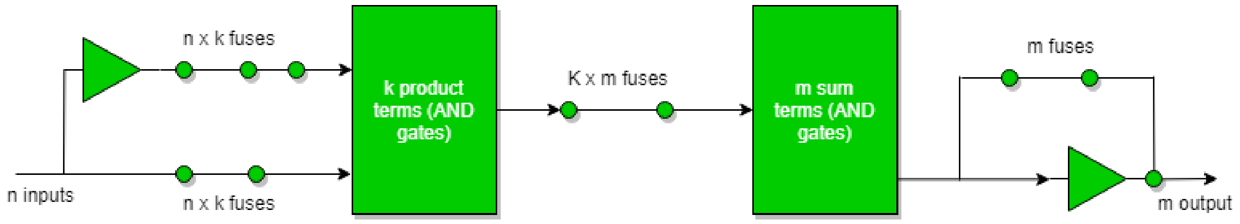
A Programmable Logic Array (PLA) is a digital device used to build custom combinational logic circuits. It contains programmable AND and OR gate networks, allowing the user to set up the logic functions needed for a specific task. Since PLAs are not given a fixed function during manufacturing, they can be configured before use to perform a variety of logic operations, making them a flexible option for creating specialized hardware designs.

PLA is similar to a ROM in concept; however, it does not provide full decoding of variables and does not generate all minterms as in the ROM. Though its name consists of the word "programmable", it does not require any type of programming like in C and C++.

Features of Programmable Logic Array

- Programmable AND and OR Gates: PLA has two types of arrays, namely programmable AND gate array and programmable OR gate array so that the logic circuits can be designed in any way.
- Reconfigurability: As compared with other logic devices, the operation of PLAs is highly flexible and these devices may be easily programmed to perform any of the numerous logical functions.
- Partial Minterm Generation: It is also to be noted here that PLA does not provide the full decoding of variables like ROM but it functions only the necessary minterms.
- Combination of Memory and Logic: PLA has both the memory and the logic features hence making it suitable for various applications.

Basic Block Diagram for PLA



Following Truth table will be helpful in understanding function on no of inputs:

A	B	C	F1	F2
0	0	0	0	0
0	0	1	0	0
0	1	0	0	0
0	1	1	1	0
1	0	0	0	1
1	0	1	1	1
1	1	0	1	0
1	1	1	1	1

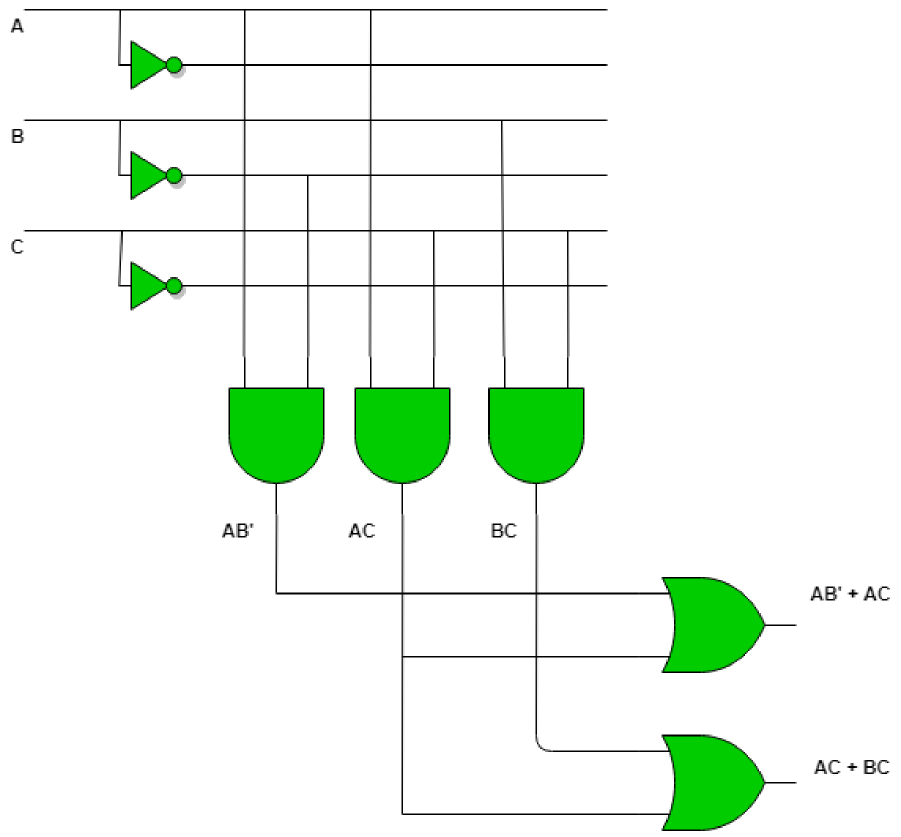
F1 = AB'C' + ABC' + ABC

on simplifying we get: F1 = AC' + AB

F2 = A'BC + AB'C + ABC

on simplifying we get: F2 = BC + AC

For the realization of the above function following circuit diagram will be used.



PLA is used for the implementation of various combinational circuits using a buffer, AND gate, and OR gate. In PLA, all the minterms are not realized but only required minterms are implemented. As PLA has a programmable AND gate array and a programmable OR gate array, it provides more flexibility but the disadvantage is, it is not easy to use.

The Operation of a PLA can be Summarized in Three Steps

1. Programming: The user defines the logic function to be implemented by the PLA by programming the input and output configurations into the device.
2. Product term generation: The inputs are applied to the AND gate array to produce a set of product terms.
3. Sum term generation: The product terms are then applied to the OR gate array to generate the final output.

PLAs are often used in digital systems as they are versatile and allow complex functions to be implemented easily. They are particularly useful for implementing Boolean expressions with many variables as the arrays of AND gates and OR gates can be configured to handle large numbers of inputs.

Applications of Programmable Logic Array

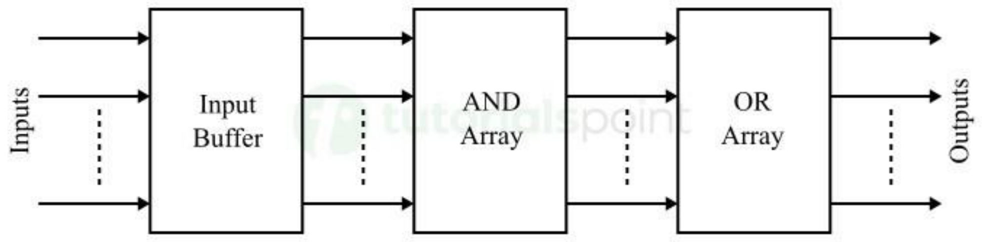
- PLA is used to provide control over data path.
- PLA is used as a counter.
- PLA is used as a decoder.
- PLA is used as a BUS interface in programmed I/O.

4.2.2. Programmable Array Logic (PAL):

In the field of digital electronics, there are several different types of programmable logic devices or PLDs. The Programmable Array Logic (PAL) is also a type of PLD used to design and implement a variety of custom logic functions. These programmable array logic devices allow digital designers to develop complex logic structures with high flexibility and efficiency. Construction-wise, a PAL device consists of an array of programmable AND gates connected to a fixed array of OR gates. This array structure helps to implement various logic functions by interconnecting the input lines, AND gates and OR gates.

Block Diagram of PAL

Similar to PLA, the Programmable Array Logic (PAL) is also a type of fixed architecture logic device having an array of programmable AND gates and an array of fixed OR gates as shown in the following figure −



From this block diagram, it can be seen that a PAL consists of the following three main components −

- Input Buffers
- AND Gate Array
- OR Gate Array

These components are connected together through a programmed connection indicated by "X". In practice, these programmed connections can be made through EPROM cells or other programming technologies.

Combinational Logic Design Using PAL Devices

We can design combinational logic circuits using Programmable Array Logic (PAL) devices. While designing combinational logic using PAL, it is important to note that the Boolean expression in the sum products form must be simplified to fit into each section of the PAL.

Because the array of OR gates is fixed, the number of product terms fed to each OR gate cannot be changed. If there is a situation when the number of product terms are more, then the Boolean function must be implemented for that section.

Let us understand the combinational logic design using PAL devices with the help of examples.

Example

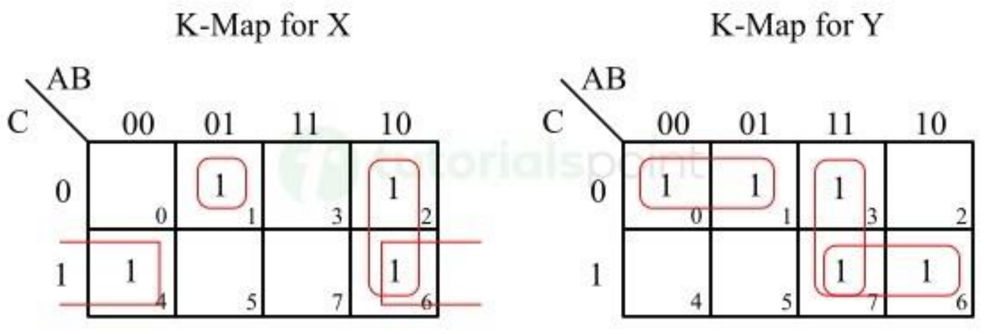
Consider a combinational logic circuit which has 3 inputs and 2 outputs. The logic functions for the outputs are given below. Implement this circuit using PAL.

$X(A,B,C)=\sum m(1,2,4,6)$

$Y(A,B,C)=\sum m(0,1,3,6,7)$

Solution

Obtaining the Boolean expressions for the given logic functions,



From these K-maps, we get,

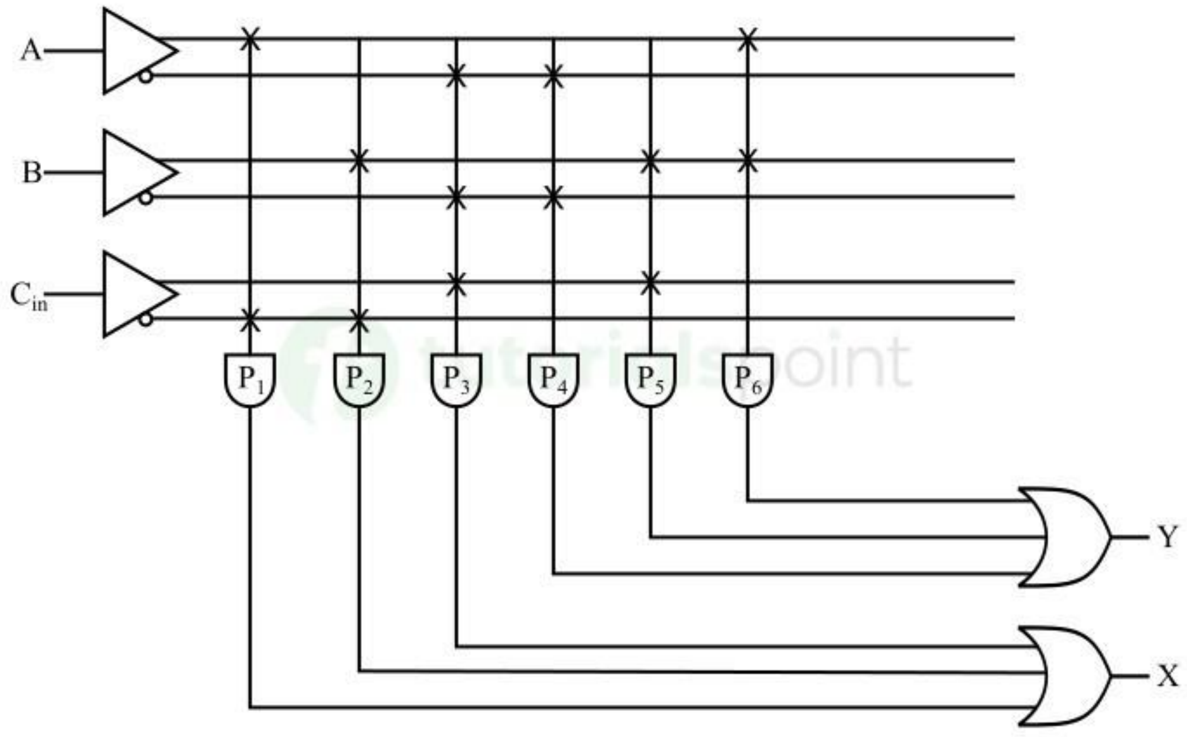
X=A$\bar{C}\bar{C}$ +B$\bar{C}$ +$\overline{ACB}$

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

Now, prepare the PAL program table for these output functions, which is given below −

Product Terms		AND Gate Inputs		
		A	B	C
1	$A\bar{B}$	1	-	0
2	$B\bar{C}$	-	1	0
3	$\overline{ABC}$	0	0	1
4	$\bar{A}\bar{B}$	0	0	-
5	BC	-	1	1
6	AB	1	1	-

Now, let's implement the PAL logic circuit as per this table. This circuit diagram is shown in the following figure −



This is how we can implement a logic function using Programmable Array Logic (PAL).

Applications of PAL:

Programmable Array Logics (PALs) are extensively used in a variety of applications in the field of digital electronics. Some common applications of PALs are listed below −

- PALs are used in embedded systems for implementing control logics, providing interfacing between different components, sensors, and other subsystems, and performing various signal processing tasks like filtering, modulation, demodulation, signal conditioning, etc.
- In communication systems, PALs are employed for implementing encoding and decoding algorithms, protocol processing, error detection and correction, multiplexing and demultiplexing, etc.
- PALs are also used in the field of automotive electronics for implementing control logics for managing engine functions, fuel injection, emission control system, anti-lock braking system, audio system, navigation, and driver assistance system.

- In industrial automation and robotics, the PALs play an important role, as they help to develop logic functions for controlling and monitoring industrial processes, sensors, and other components.
- PALs are also used in consumer electronics like washing machines, microwave ovens, home automation systems, etc. to implement their control functions.

UNIT-V:8085 MICROPROCESSOR: Introduction to microprocessor – pin configuration of 8085 – Flags – Registers (General and special purpose) – interrupts and its priority – instruction set of 8085 – addressing modes of 8085 – Assembly language programming using 8085 – programs for addition, subtraction, multiplication and division (8-Bit only).

5.1. Introduction to microprocessor:

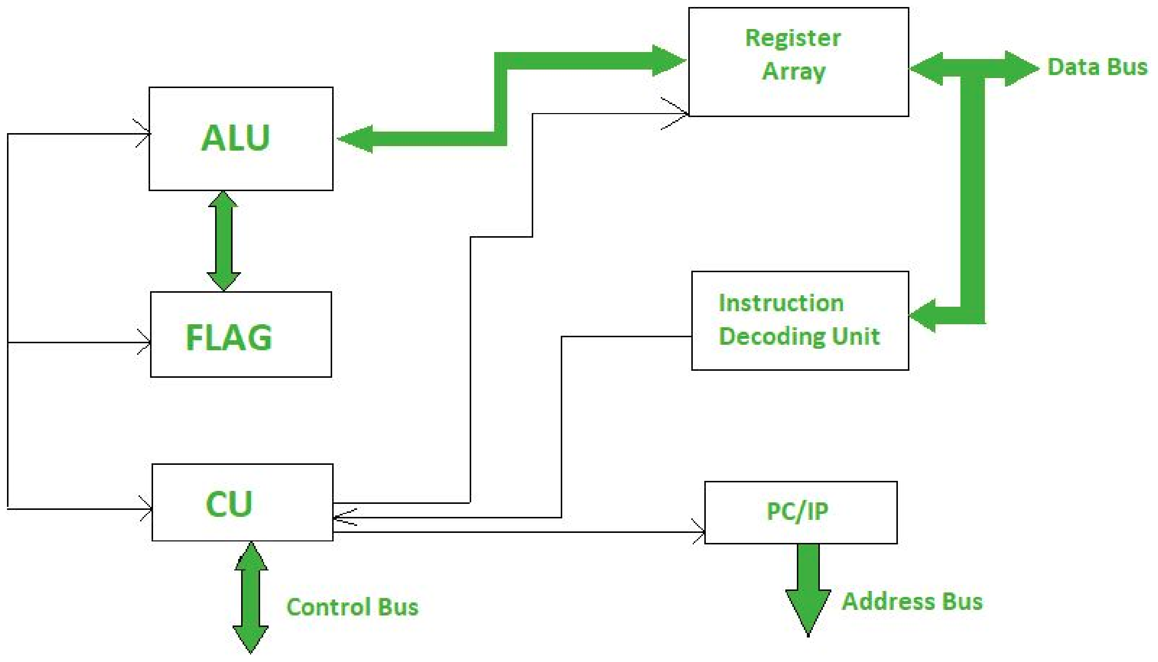
A Microprocessor is is a programmable device that takes in input, performs some arithmetic and logical operations over it and produces the desired output. In simple words, a Microprocessor is a digital device on a chip that can fetch instructions from memory, decode and execute them, and give results. It is an important part of a computer architecture without which you will not be able to perform anything on your computer.

Block Diagram of a Microprocessor

A Microprocessor takes a bunch of instructions in machine language and executes them, telling the processor what it has to do. The microprocessor performs three basic things while executing the instruction:

- It performs some basic operations like addition, subtraction, multiplication, division, and some logical operations using its Arithmetic and Logical Unit (ALU). New Microprocessors also perform operations on floating-point numbers.
- Data in microprocessors can move from one location to another.
- It has a Program Counter (PC) register that stores the address of the next instruction based on the value of the PC, Microprocessor jumps from one location to another and makes decisions.

A typical Microprocessor structure looks like this.



Clock Speed of different Microprocessor:

- 16-bit Microprocessor

8086: 4.7MHz, 8MHz, 10MHz

8088: more than 5MHz

80186/80188: 6MHz

80286: 8MHz

- 32-bit Microprocessor

INTEL 80386: 16MHz to 33MHz

INTEL 80486: 16MHz to 100MHz

PENTIUM: 66MHz

- 64-bit Microprocessor

INTEL CORE-2: 1.2GHz to 3GHz

INTEL i7: 2.66GHz to 3.33GHz

INTEL i5: 2.4GHz to 3.6GHz

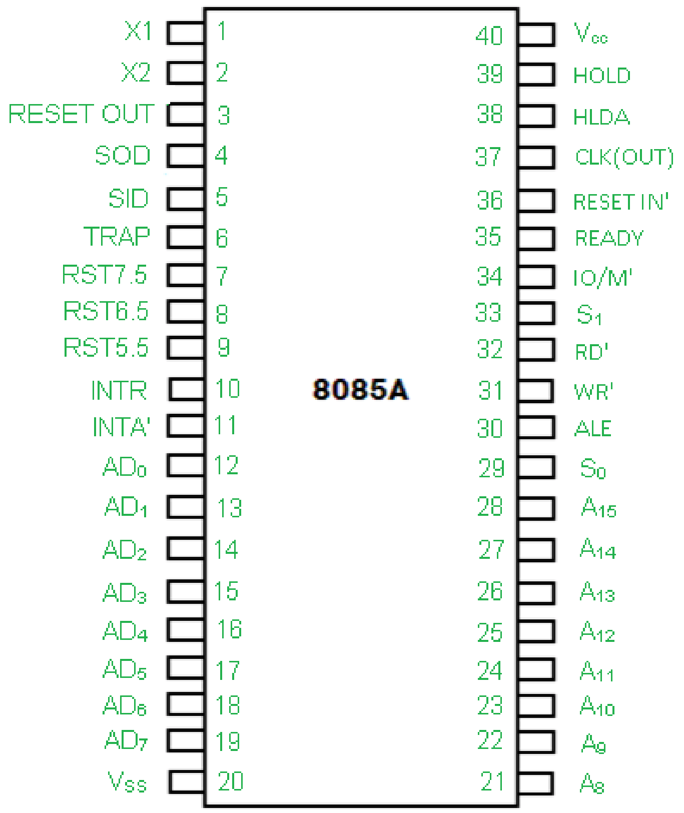
INTEL i3: 2.93GHz to 3.33GHz

We do not have any 128-bit Microprocessor at work at present one of the reasons for this is that we are a long way from exhausting the 64-bit address space itself, we use it at a constant rate of roughly 2 bits every 3 years. At present we have only used 48 bits of 64 bits so why require 128-bit address space. Also, 128-bit Microprocessor would be much slower than the 64 bit Microprocessor.

5.2. Pin configuration of 8085:

The 8085 microprocessor is a popular 8-bit microprocessor developed by Intel. It has 40 pins, each with a specific function for interfacing with memory, input/output devices, and other components.

Pin diagram of 8085 microprocessor is shown below:



Pin Descriptions:

1. Address Bus and Data Bus

Address Bus (A8 to A15): The address bus is unidirectional, i.e., bits flow in one direction from the microprocessor unit to the peripheral devices and uses the higher order address bus.

Address Data Bus (AD0 to AD7): These are bi-directional data pins used to transfer data between the microprocessor and memory or I/O devices. The microprocessor is an 8-bit processor, so it uses 8 data lines. These pins serve the dual purpose of transmitting lower order address and data byte. During 1st clock cycle, these pins act as lower half of address. In remaining clock cycles, these pins act as data bus.

2. Control and Status Signals

ALE - It is an Address Latch Enable signal. It goes high during first T state of a machine cycle and enables the lower 8-bits of the address, if its value is 1 otherwise data bus is activated.

IO/M' - It is a status signal which determines whether the address is for input-output or memory. When it is high(1) the address on the address bus is for input-output devices. When it is low(0) the address on the address bus is for the memory.

SO, S1 - These are status signals. They distinguish the various types of operations such as halt, reading, instruction fetching or writing.

IO/M'	**S1**	**S0**	**Data Bus Status**
0	1	1	Opcode fetch
0	1	0	Memory read
0	0	1	Memory write
1	1	0	I/O read

IO/M'	S1	S0	Data Bus Status
1	0	1	I/O write
1	1	1	Interrupt acknowledge
0	0	0	Halt

RD' - It is a signal to control READ operation. When it is low the selected memory or input-output device is read.

WR' - It is a signal to control WRITE operation. When it goes low the data on the data bus is written into the selected memory or I/O location.

READY - It senses whether a peripheral is ready to transfer data or not. If READY is high(1) the peripheral is ready. If it is low(0) the microprocessor waits till it goes high. It is useful for interfacing low speed devices.

3. Power Supply and Clock Frequency

Vcc - +5v power supply

Vss - Ground Reference

XI, X2 - A crystal is connected at these two pins. The frequency is internally divided by two, therefore, to operate a system at 3MHZ the crystal should have frequency of 6MHZ.

CLK (OUT) - This signal can be used as the system clock for other devices.

4. Interrupts and Peripheral Initiated Signals:

The 8085 has five interrupt signals that can be used to interrupt a program execution.

(i) INTR

(ii) RST 7.5

(iii) RST 6.5

(iv) RST 5.5

(v) TRAP

The microprocessor acknowledges Interrupt Request by INTA' signal. In addition to Interrupts, there are three externally initiated signals namely RESET, HOLD and READY. To respond to HOLD request, it has one signal called HLDA.

1. INTR (Interrupt Request): This pin is used to request an interrupt from an external device.
2. RST7.5, RST6.5, RST5.5: These are vectored interrupts with different priority levels.
3. TRAP: This is a non-maskable interrupt and has the highest priority.
4. INTA (Interrupt Acknowledge): This pin is used to acknowledge interrupts from external devices.

5. Reset Signals

RESET IN - When the signal on this pin is low(0), the program-counter is set to zero, the microprocessor unit is reset.

RESET OUT - This signal indicates that the MPU is being reset. The signal can be used to reset other devices.

6. DMA Signals

HOLD - It indicates that another device is requesting the use of the address and data bus. Having received HOLD request the microprocessor relinquishes the use of the buses as soon as the current machine cycle is completed. Internal processing may continue. After the removal of the HOLD signal the processor regains the bus.

HLDA - It is a signal which indicates that the hold request has been received after the removal of a HOLD request, the HLDA goes low.

7. Serial I/O Ports

Serial transmission in 8085 is implemented by the two signals.

SID and SOD - SID is a data line for serial input whereas SOD is a data line for serial output.

Features of 8085 Microprocessor

1. 8-bit Processor: Processes 8 bits of data at a time.
2. 16-bit Address Bus: Can address up to 64 KB of memory.
3. 5 MHz Clock Speed: Provides a decent processing speed for basic tasks.
4. Low Power Consumption: Ideal for embedded systems with minimal power needs.
5. Interrupt System: Features 5 interrupt pins for external signal response.

Limitations of 8085 Microprocessor

1. Limited Memory Addressing: Can address only up to 64KB of memory due to its 16-bit address bus.
2. No Multiprocessing Support: Cannot connect multiple processors for parallel processing.
3. Limited Instruction Set: Supports only 50 instructions, restricting its functionality for complex tasks.
4. No Direct Memory Access: Lacks DMA (Direct Memory Access) support, which slows data transfer between memory and peripherals.
5. Slower Processing: With a clock speed of up to 5 MHz, it is slower compared to modern processors.

5.3. Flags:

The Intel 8085 microprocessor contains five flip-flops to serve as a status flags. The flipflops are reset or set according to the conditions which arise during an arithmetic or logical operation.

a. Carry Flag (CS)

b. Parity Flag (P)

c. Auxiliary Carry Flag (AC)

d. Zero Flag(Z)

e. Sign Flag(S)

D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
S	Z	X	AC	X	P	X	CS

a) Carry Flag (CS)

Carry is generated when performing n bit operations and the result is more than n bits, then this flag becomes set i.e. 1, otherwise it becomes reset i.e. 0.

During subtraction (A-B), if A>B it becomes reset and if (A<B) it becomes set.

Carry flag is also called borrow flag.

77

1-carry out from MSB bit on addition or borrow into MSB bit on subtraction

0-no carry out or borrow into MSB bit

Example:

MVI A 30 (load 30H in register A)

MVI B 40 (load 40H in register B)

SUB B (A = A – B)

These set of instructions will set the carry flag to 1 as 30 – 40 generates a carry/borrow.

MVI A 40 (load 40H in register A)

MVI B 30 (load 30H in register B)

SUB B (A = A – B)

These set of instructions will reset the sign flag to 0 as 40 – 30 does not generate any carry/borrow.

b) Parity Flag (P)

If after any arithmetic or logical operation the result has even parity, an even number of 1 bits, the parity register becomes set i.e. 1, otherwise it becomes reset i.e. 0.

1-accumulator has even number of 1 bits

0-accumulator has odd parity

Example:

MVI A 05 (load 05H in register A)

This instruction will set the parity flag to 1 as the BCD code of 05H is 00000101, which contains even number of ones i.e. 2.

c) Auxiliary Carry Flag (AC)

This flag is used in BCD number system(0-9). If after any arithmetic or logical operation D(3) generates any carry and passes on to B(4) this flag becomes set i.e. 1, otherwise it becomes reset i.e. 0. This is the only flag register which is not accessible by the programmer1-carry out from bit 3 on addition or borrow into bit 3 on subtraction 0-otherwise

Example:

MOV A 2B (load 2BH in register A)

MOV B 39 (load 39H in register B)

ADD B (A = A + B)

78

These set of instructions will set the auxiliary carry flag to 1, as on adding 2B and 39, addition of lower order nibbles B and 9 will generate a carry.

d) Zero Flag(Z)

After any arithmetical or logical operation if the result is 0 (00)H, the zero flag becomes set i.e. 1, otherwise it becomes reset i.e. 0.

00H zero flag is 1.

from 01H to FFH zero flag is 0

1- zero result

0- non-zero result

Example:

MVI A 10 (load 10H in register A)

SUB A (A = A – A)

These set of instructions will set the zero flag to 1 as 10H – 10H is 00H

e) Sign Flag(S)

After any operation if the MSB (B(7)) of the result is 1, it indicates the number is negative and the sign flag becomes set, i.e. 1. If the MSB is 0, it indicates the number is positive and the sign flag becomes reset i.e. 0.

from 00H to 7F, sign flag is 0

from 80H to FF, sign flag is 1

1- MSB is 1 (negative)

0- MSB is 0 (positive)

Example:

MVI A 30 (load 30H in register A)

MVI B 40 (load 40H in register B)

SUB B (A = A – B)

These set of instructions will set the sign flag to 1 as 30 – 40 is a negative number.

MVI A 40 (load 40H in register A)

MVI B 30 (load 30H in register B)

SUB B (A = A – B)

These set of instructions will reset the sign flag to 0 as 40 – 30 is a positive number.

5.4. Registers (General and special purpose):

In the 8085 microprocessor, registers refer to the high-speed internal storage locations or memory elements that hold binary data or instructions temporarily during the execution of a program. The 8085 microprocessor features various general purpose, special purpose, and combined purpose registers to efficiently carry out arithmetic, logical, and data transfer operations. The 8085 microprocessor has eight addressable 8-bit registers: A, B, C, D, E, H, L, F, and two 16-bit registers, PC and SP.

These registers can be classified as follows:

- General Purpose Registers: B, C, D, E, H, L
- Special Purpose Registers:

1. Accumulator (A)
2. Flag Register (F)
3. Instruction Register

General Purpose Registers

The 8085 microprocessor features 590 general-purpose 8-bit registers namely B, C, D, E, H, and L that can store 8-bit data/operand. These registers serve as temporary high-speed memory during arithmetic/logical operations. They can also be combined into 480 register pairs like BC, DE, and HL to efficiently handle 16-bit data/addresses. The H-L register pair is commonly used as an index register to access memory locations in memory addressing modes.

- Register B: Used for various data manipulation tasks.
- Register C: Often paired with Register B for 16-bit operations.
- Register D: Another versatile register for general-purpose use.
- Register E: Typically paired with Register D.
- Register H: Used in conjunction with Register L.
- Register L: Completes the pair with Register H for addressing.

Special Purpose Registers

Some 210 special purpose registers in 8085 include 16-bit Program Counter (PC) and Stack Pointer (SP) registers along with 8-bit Accumulator, Flags, Instruction, and Memory Address registers. Special purpose registers in 8085 are designed for specific tasks related to the control and operation of the microprocessor. These registers include the accumulator, flag register, stack pointer, and program counter.

Special Purpose Registers in 8085

- Accumulator (A): A crucial register used in arithmetic and logic operations, as well as data transfer.

- Flag Register (F): Holds the status flags that indicate the outcome of arithmetic and logic operations.

The flag register in the 8085 microprocessor is an 8-bit register that indicates the status of the microprocessor after arithmetic and logic operations. It contains five flags:

- **Sign (S):** Indicates the sign of the result.
- **Zero (Z):** Set if the result is zero.
- **Auxiliary Carry (AC):** Indicates a carry-out of the lower nibble.
- **Parity (P):** Set if the number of set bits in the result is even.
- **Carry (CY):** Indicates a carry-out of the most significant bit.
- **Instruction Register:** Holds the current instruction being executed. Instruction Register in 8085
  The instruction register in 8085 holds the opcode of the current instruction that the microprocessor is executing, enabling the control unit to decode and execute the instruction.

5.5. Interrupts and its priority:

In the 8085 microprocessor, interrupt is a process in which control of the program transfers from the main program to the starting location defined by interrupt. It is a process by which some external device or peripheral informs microprocessor to become ready for data communication by accepting the made request. Hence, it is a signal that temporarily suspends the normal execution of a program and redirects the control to a specific interrupt service routine (ISR). Interrupts allow the microprocessor to respond to external events, such as user input, system events, or hardware signals without the need for constant polling.

Types of Interrupt Signals in the 8085 Microprocessor

There are five interrupt signals in the 8085 microprocessor:

1. TRAP: The TRAP interrupt is a non-maskable interrupt that is generated by an external device, such as a power failure or a hardware malfunction. The TRAP interrupt has the highest priority and cannot be disabled.

2. RST 7.5: The RST 7.5 interrupt is a maskable interrupt that is generated by a software instruction. It has the second highest priority.
3. RST 6.5: The RST 6.5 interrupt is a maskable interrupt that is generated by a software instruction. It has the third highest priority.
4. RST 5.5: The RST 5.5 interrupt is a maskable interrupt that is generated by a software instruction. It has the fourth highest priority.
5. INTR: The INTR interrupt is a maskable interrupt that is generated by an external device, such as a keyboard or a mouse. It has the lowest priority and can be disabled.

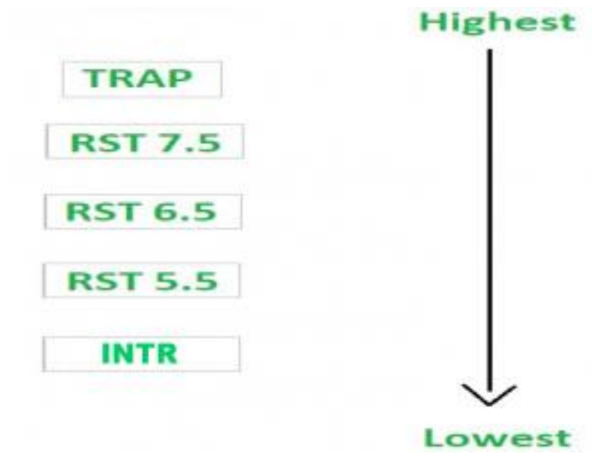
When microprocessor receives any interrupt signal from peripheral(s) which are requesting its services, it stops its current execution and program control is transferred to a sub-routine by generating CALL signal and after executing sub-routine by generating RET signal again program control is transferred to main program from where it had stopped. When microprocessor receives interrupt signals, it sends an acknowledgement (INTA) to the peripheral which is requesting for its service.

Interrupts can be classified into various categories based on different parameters:

- **Hardware and Software Interrupts**: When microprocessors receive interrupt signals through pins (hardware) of microprocessor, they are known as *Hardware Interrupts*. There are 5 Hardware Interrupts in 8085 microprocessor. They are: *INTR, RST 7.5, RST 6.5, RST 5.5, TRAP Software Interrupts* are those which are inserted in between the program which means these are mnemonics of microprocessor. There are 8 software interrupts in 8085 microprocessor. They are: *RST 0, RST 1, RST 2, RST 3, RST 4, RST 5, RST 6, RST 7.*
- **Vectored and Non-Vectored Interrupts**: Vectored Interrupts are those which have fixed vector address (starting address of sub-routine) and after executing these, program control is transferred to that address. Vector Addresses are calculated by the formula 8 * TYPE

Maskable and Non-Maskable Interrupts: Maskable Interrupts are those which can be disabled or ignored by the microprocessor. These interrupts are either edge-triggered or level-triggered, so they can be disabled. Non-Maskable Interrupts are those which cannot be disabled or ignored by microprocessor. It consists of both level as well as edge triggering and is used in critical power failure conditions.

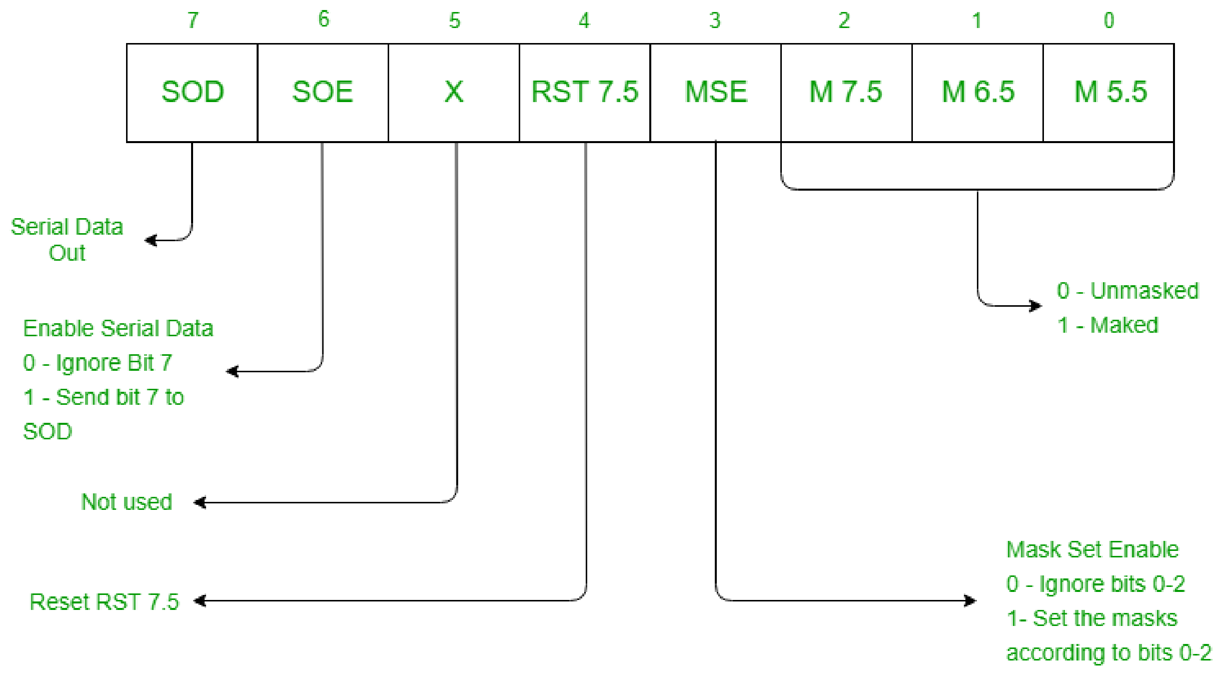
Priority of Interrupts: When microprocessor receives multiple interrupt requests simultaneously, it will execute the interrupt service request (ISR) according to the priority of the interrupts.



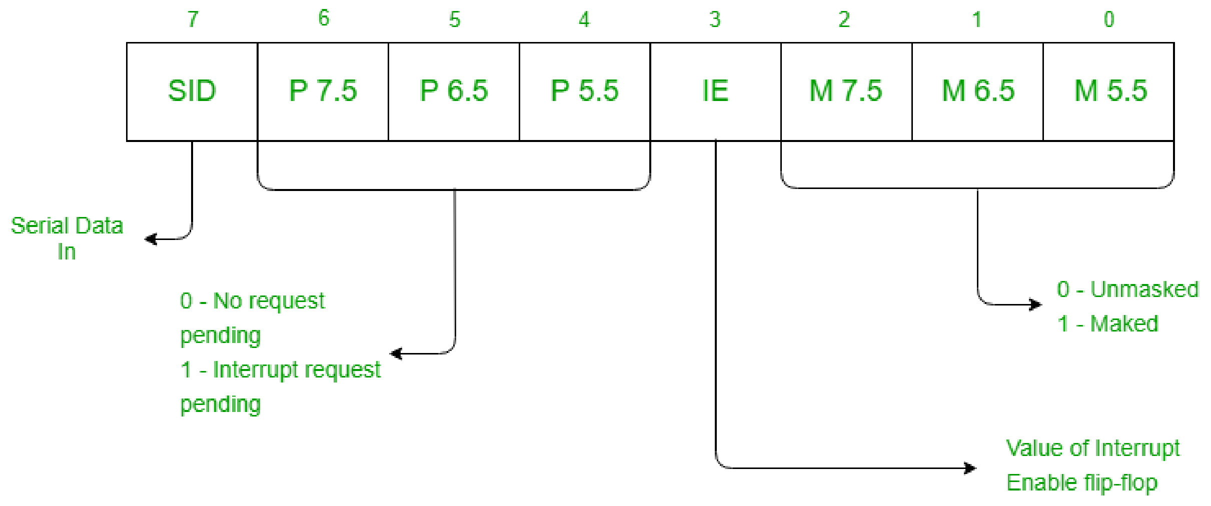
Instruction for Interrupts

- **Enable Interrupt (EI):** The interrupt enable flip-flop is set and all interrupts are enabled following the execution of next instruction followed by EI. No flags are affected. After a system reset, the interrupt enable flip-flop is reset, thus disabling the interrupts. This instruction is necessary to enable the interrupts again (except TRAP).
- **Disable Interrupt (DI):** This instruction is used to reset the value of enable flip-flop hence disabling all the interrupts. No flags are affected by this instruction.
- **Set Interrupt Mask (SIM):** It is used to implement the hardware interrupts (RST 7.5, RST 6.5, RST 5.5) by setting various bits to form masks or generate output data via the Serial Output Data (SOD) line. First the required value is loaded in accumulator then

SIM will take the bit pattern from i



- **Read Interrupt Mask (RIM):** This instruction is used to read the status of the hardware interrupts (RST 7.5, RST 6.5, RST 5.5) by loading into the A register a byte which defines the condition of the mask bits for the interrupts. It also reads the condition of SID (Serial Input Data) bit on the microprocessor.



Uses of Interrupts in 8085 microprocessor

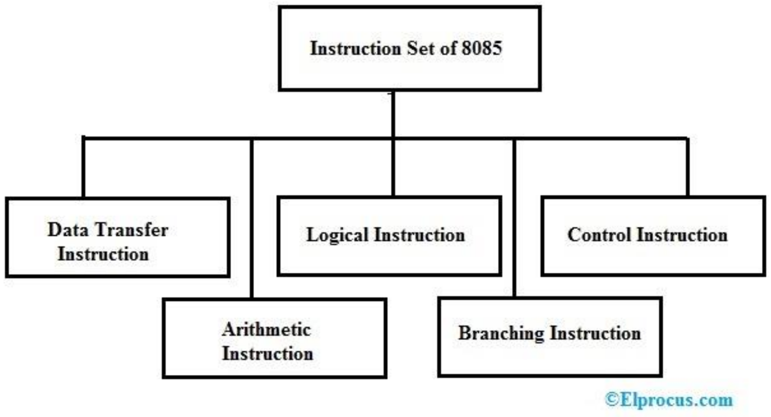
- Interrupts in the 8085 microprocessor are used for various purposes, including:
- **Real-time processing**: Interrupts allow the microprocessor to respond quickly to external events in real-time, such as user input or hardware signals. This is particularly useful in applications such as control systems and data acquisition systems where time-critical operations are required.
- **Multi-tasking:** Interrupts enable the microprocessor to perform multiple tasks simultaneously by temporarily suspending the current task and executing the ISR for the interrupting event. This allows the microprocessor to switch between different tasks and maximize the utilization of system resources.
- **Input/output operations**: Interrupts can be used for handling input/output operations, such as data transfer between the microprocessor and external devices. This allows the microprocessor to perform other tasks while waiting for input/output operations to complete.
- **Error handling:** Interrupts can be used for error handling and exception handling, such as detecting and recovering from hardware or software errors.
- **Power management**: Interrupts can be used for power management, such as putting the microprocessor into a low-power mode when it is not needed and waking it up when an interrupt occurs.

5.6. Instruction set of 8085:

A binary command that is used to perform a function in the microprocessor over provided data is known as instruction. A set of instructions is known as an instruction set that decides the microprocessor function. Every instruction includes two parts like Opcode & the Operand where Opcode is used to specify the function to be executed & operand gives the data to be functioned on.

Classification of Instruction Set of 8085

The instruction set of 8085 microprocessor is classified into five types which include the following.



- Data Transfer Instruction
- Arithmetic Instruction
- Logical Instruction
- Branching Instruction
- Control Instruction

Data Transfer Instruction:

An instruction that is used to transfer the data from one register to another is known as data transfer instruction. So, the data transfer can be done from source to destination without changing the source contents. Data transfer mainly occurs from one register to another register, from memory location to register, register to memory, and between an I/O device & accumulator. Following are the list of Data Transfer Instruction

Mov r, M

This data transfer instruction is used to transfer data present within the memory (M) to the register (r). But, the memory location address must be there within the HL register.

Example: MOV r, 1020H

Mov M, Data

This type of instruction specifies the data transfer immediately to a location of memory. This memory location address can be specified at the H-L registers.

MVI r, Data (Move Immediate)

In this type of instruction, the transmission of data can be done immediately toward the particular register.

LDA address (Load Accumulator)

LDA is a load accumulator instruction that is mainly used for copying the data available in the address of memory indicated as the instruction's operand to the accumulator. Particularly, in this case, the available data in the 16-bit address memory is transferred toward the accumulator.

LDAX (LoaD Accumulator by eXtended Register Pair)

It is a load accumulator from an address in the register pair. In this type of data transfer instruction, the register holds the address of the data that needs to be loaded to the accumulator.

LHLD (Load H & L Registers Direct)

LHLD instruction is a direct load instruction, where it loads the H-L register with the data from the memory. In this type of instruction, the data which is available in the address specified is copied to the L register first and then the available data within the next memory location will be loaded in the H register.

STA Address (Store Accumulator Contents in Memory)

STA stands for stored accumulator direct instruction. Once this instruction is accepted, then the available data within the accumulator can be transferred to the address of memory indicated within the operand.

STAX Register (Store Accumulator by Extended Register)

It is a stored accumulator indirect instruction. In this instruction, the register is available as the operand that holds a memory address. Thus, the accumulator data can be copied to that specific memory location.

XCHG (Exchange)

This type of data transfer instruction can be used to exchange the data available within two registers.

SPHL (Stack Pointer HL Register)

In this data transfer instruction, the data of H &L can be moved to the stack pointer.

PCHL (Program Counter with HL Data)

Similar to SPHL instruction, this PCHL instruction simply copies the H-L register's data into the SP by loading the high order bytes at H & low order bytes at L.

PUSH

In this type of instruction, the stack can be loaded with the available data within the register provided in the operand. Initially, the stack pointer gets decreased & high order bytes are copied to the stack. Further stack pointer gets decreased to load the low order register bytes.

POP

This instruction indicates the data transfer from the top of the stack to the register provided as the operand.

OUT

In this type of data transfer instruction, the data available at the accumulator can be copied toward the I/O port. An 8-bit port address at the operand is present.

IN

This type of instruction is used to load the data available at the I/O port to the accumulator. The operand simply holds the port address from where the data can be copied.

Arithmetic Instruction of 8085

The arithmetic instructions perform different operations like addition, subtraction, increment & decrement on the data within memory & register in the 8085 microprocessor.

ADD r

This arithmetic instruction adds the data which is available in the register to the data available within the accumulator & the final result will be stored in the accumulator.

Example: ADD C

ADD M

This type of instruction is mainly used to add the date in the memory address data denoted at the operand to the data available at the accumulator. So the addition result will be stored within the accumulator.

Example: ADD 28H

ADI Data (Add Immediate)

In this instruction, the 8-bit data is specified as an operand is added immediately to the data available at the accumulator & the result is stored at the accumulator.

ACI Data (Add with Carry Immediate)

This type of instruction simply adds the 8-bit data available at the operand & carries the flag by the data available at the accumulator. After every addition, the flag reproduces the output of the addition.

ADC r (Add with Carry)

In this type of instruction, the data present at the register can be added to the data available at the accumulator with the carry bit & output is simply reflected at the accumulator.

AMC M

This type of instruction is mainly used to add the available data at the location of memory whose address is denoted within the operand specified & the carry bit with the data available within the accumulator. So the output of addition can be stored within the accumulator.

Example: AMC 25H

SUB r

This type of instruction is used to subtract the available data at the register given at the operand from the data present in the accumulator. The final result will be stored at the accumulator.

SUB M

This instruction is used to subtract the available data at the location of memory whose address is provided by the H-L register from the data present at the accumulator.

SUI Data (Subtract Immediate from Accumulator)

This type of instruction is mainly used to instantly subtract the data available as operand within the instruction from the available data at the accumulator. After every subtraction, the flag can be changed to show the result of subtraction.

SBI Data (Subtract with Borrow Immediate from Accumulator)

This type of instruction helps subtract the 8-bit data provided as the operand & the borrow bit from the available data at the accumulator, and the result will be stored within the accumulator.

SBB r

This instruction is used to subtract the data present at the register & the borrow bit from the data present at the accumulator.

SBB M (Subtraction with Borrow)

This instruction is used to specify the subtraction of data available at the memory location, whose address is available at the H-L register & the borrow bit from the data present at the accumulator.

Example: SBB 1000H

INX r (Increment Extended Register)

This type of instruction is used to increase the data by 1 which is available at the register provided at the operand. The result will be stored at the same register.

DCX r (Decrement Extended Register)

This type of instruction decreases the data available at the register by 1 & the result will be stored in the same register.

DCR M (Decrement Register)

In an instruction, sometimes the operand holds a location of memory. The memory location address is available at the H-L pair. Thus the data available at that specific location will be decreased by 1.

DAA (Decimal Adjust Accumulator)

DAA is a decimal adjust accumulator, used to break the binary number from 8-bit to two 4-bit binary-coded decimal numbers.

Logical Instruction:

Logical instructions are mainly used to perform different operations like logical or Boolean over the data available in either memory or register. These instructions will modify the flag bits based on the operation executed.

CMP R/M (Compare the Register/Memory with the Accumulator)

This instruction is used to compare the data at the accumulator with the data present at the register or memory which is given as operand. According to the result obtained by the comparison, the flags are set. While the data that is compared remains unchanged.

CPI Data (Compare immediate through the Accumulator)

This type of instruction compares the 8-bit data provided as operand within the instruction by the data available within the accumulator. This result is shown through the flags.

ANA R/M (Logical AND register or memory with the accumulator)

This instruction executes the AND operation of the data available within the accumulator to the data available in the memory or register. After the operation of AND, S, P, Z will be changed to show the outcome of the comparison.

ANI data (And Immediate with Accumulator)

This instruction executes AND operation for the immediate 8-bit data provided as operand by the data available in the accumulator.

ORA R/M (OR Accumulator Register or Memory)

This instruction is used to perform OR operation of the data available within the accumulator by the data available in the memory location or register.

ORI data (OR Immediate Data)

The 8-bit data provided as an operand is ORed logically with the data within the accumulator. So, the output of this instruction can be saved within the accumulator.

XRA R/M (Exclusive OR Immediate with Accumulator)

This instruction is used to execute XOR operation through data available at the accumulator & the data present at the memory or register.

XRI data (Exclusive OR Accumulator)

This type of instruction is used to execute the XOR operation of the 8-bit data specified as operand & the data present at the accumulator. The output will be stored at the accumulator.

RLC (Rotate Left Accumulator)

This instruction holds significance when there exists a need to rotate the bits present in the accumulator. Basically, for an 8-bit value, each bit is rotated or shifted left by one position. Also, the rotation of the last bit of the sequence i.e., D7, sets the CY flag.

RRC (Right Rotate Accumulator)

This instruction is used to rotate the bit toward the right with one position. So, in this case, D0 sets the CY flag.

RAL (Rotate Accumulator Left)

This type of instruction is used to rotate the bits toward the left with one of the data available within the accumulator through the carry flag. Here, D7 can be shifted to hold the flag & the bit within the carry flag can be shifted to D0.

RAR (Rotate Accumulator Right)

This type of instruction is mainly used to rotate the data bits to the right which are available within the accumulator by the carry flag. Here, D0 can be shifted to hold the flag & the carry bit can be moved to the D7 position.

STC (Set the Carry Flag)

This type of instruction is used to set the carry flag (CF) to 1 by not affecting any other flags.

CMA (Complement the Accumulator)

This type of instruction generates the complement of data at the accumulator. So, this function does not change any of the flags.

CMC (Complement the Carry Flag)

This type of instruction is used to complement the data available at the carry flag (CF). So this instruction does not affect any other flag.

Branching Instruction:

These types of instructions are mainly used to transfer or switch the microprocessor from one location to another. So, it simply changes the general sequential flow.

JMP address (Jump unconditionally)

This type of instruction is mainly used to transfer the series of the current program to that location of memory whose 16-bit address can be simply specified within the operand of the instruction.

Jx Address

This is a conditional branching type instruction, where the series of current programs can be transferred to that specific location whose address can be provided at the operand. However this transferring mainly depends on the specified PSX flag.

CALL address

This instruction shifts the control of a series of current programs toward the memory address available at the operand. However the PC gets decreased before transferring,

RET (Return from the Subroutine)

This type of instruction can cause the unconditional return of the sub-routine to the actual program.

RST(Restart Instruction)

This type of instruction is mainly used to transfer the series from the main program to the interrupt service routine. Mostly, the transfer can be performed above one of the 8-bits which are indicated within the operand.

Control Instruction:

These instructions are mainly used to control the microprocessor operations. These instructions are discussed below.

NOP (No operation)

NOP stands for no operation. Once the 8085 microprocessor gets this instruction, then it does not perform any operation based on execution.

DI (Disable Interrupts)

DI is the disabling of the interrupt that is generated within the microprocessor. Interrupt resetting will allow disabling all the interrupts apart from TRAP.

EI (Enable Interrupts)

This type of instruction is mainly used to allow the interrupt. Once the interrupt enable pin is set then leads to enabling the interrupts within the system.

HLT (Halt & Enter Wait State)

Once the HLT interrupt is decoded through the microprocessor, it stops the current operation and waits for further instruction. To escape from the halt condition either a reset or an interrupt is necessary.

SIM (Set Interrupt Mask)

SIM is the set interrupt mask, which is used to execute the hardware interrupts programming & serial output.

RIM (Read Interrupt Mask)

RIM is the read interrupt mask that is used to situate the preferred data at the accumulator based on the serial input & interrupt.

5.7. Addressing modes of 8085:

The 8085 microprocessor supports various addressing modes that define how operands (data) are specified and accessed from memory or registers, enabling efficient instruction execution. These addressing modes provide flexibility, allowing programmers to optimize performance, reduce memory usage, and simplify code complexity. By combining appropriate addressing modes with memory and performance optimizations, programmers can enhance the efficiency of their programs.

- An addressing mode specifies how an instruction accesses data, whether directly, indirectly, or immediately, influencing how operands are fetched or manipulated.

- Proper use of addressing modes in the 8085 helps optimize performance and memory usage while simplifying programming tasks.

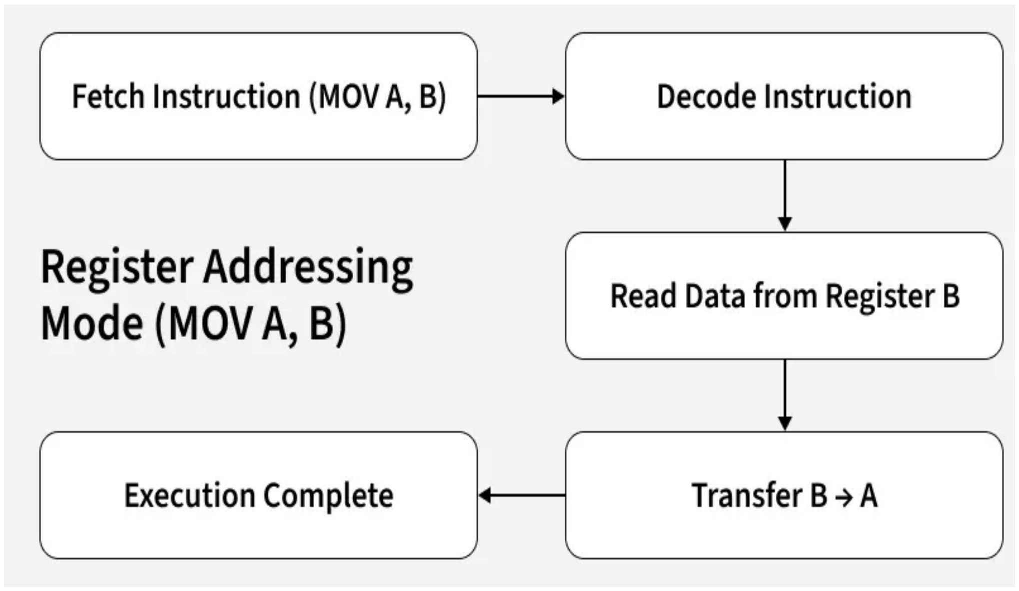
Types of Addressing Modes:

1. Register Addressing Mode:

Feature: Operand is stored and operated on, within CPU registers, making execution very fast since no memory access is required.
Description: In this mode, the operands are stored in the microprocessor's registers. The instruction specifies the register where the data is stored or to which it should be transferred.

Examples:

- *MOV A, B → Moves contents of register B to A.*
- *ADD B → Adds contents of B to A and stores the result in A.*
- *INR A → Increments the contents of A by 1.*


Fetch Instruction (MOV A, B)
Decode Instruction
Register Addressing Mode (MOV A, B)
Read Data from Register B
Transfer B → A
Execution Complete

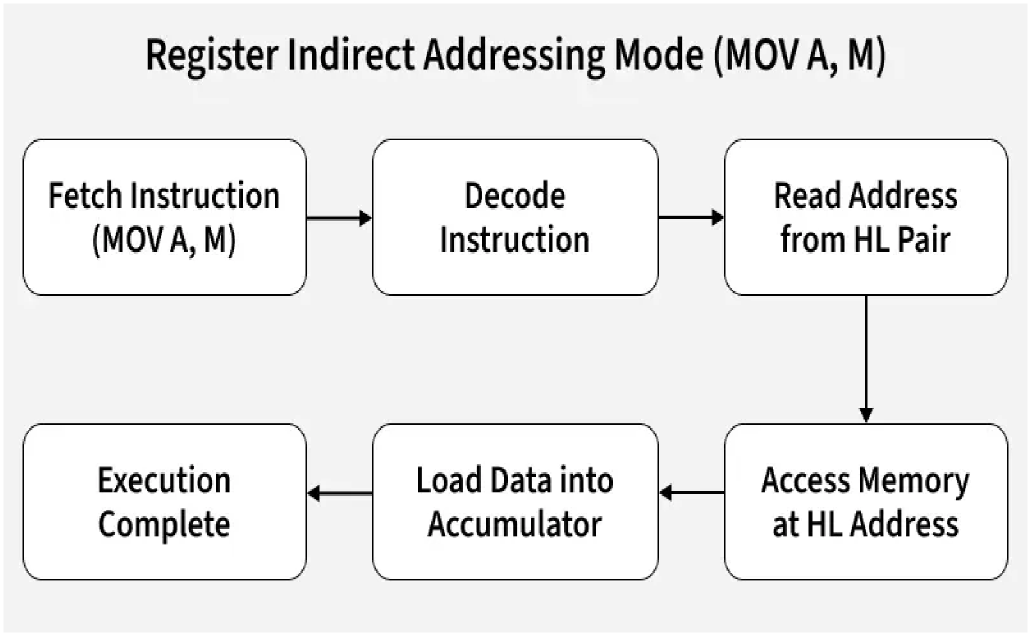
2. Register Indirect Addressing Mode:

Feature: Operand's memory address is specified indirectly through a register pair, allowing flexible access to memory.
Description: Here, the address of the operand is stored in a register pair, and the instruction

indirectly refers to the data through this pair. The actual operand resides in the memory location pointed to by the register pair.

Examples:

- *MOV A, M → Moves data from memory (address in HL pair) to A.*
- *LDAX B → Loads accumulator with data from memory pointed by BC pair.*
- *STAX B → Stores accumulator contents in memory pointed by BC pair.*

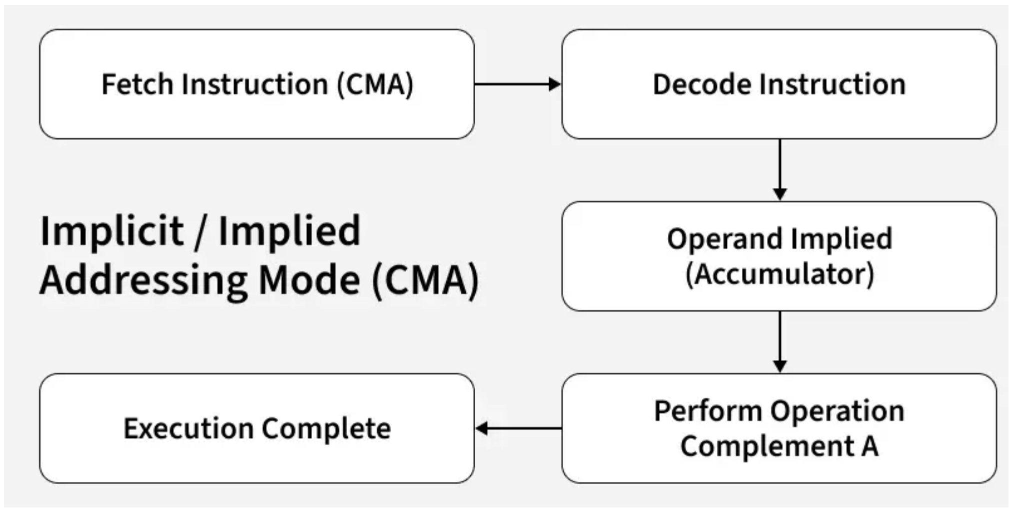

Register Indirect Addressing Mode (MOV A, M)
Fetch Instruction (MOV A, M)
Decode Instruction
Read Address from HL Pair
Access Memory at HL Address
Load Data into Accumulator
Execution Complete

3. Implied / Implicit Addressing Mode:

Feature: Operand is implied or predefined in the instruction, usually involving the accumulator. Description: In this mode, the operand is not explicitly mentioned in the instruction. It is inherently defined by the instruction itself.

Examples:

- *CMA → Complements the contents of the accumulator.*
- *RRC → Rotates accumulator right by one bit.*
- *RLC → Rotates accumulator left by one bit.*


Fetch Instruction (CMA)
Decode Instruction
Implicit / Implied Addressing Mode (CMA)
Operand Implied (Accumulator)
Perform Operation Complement A
Execution Complete

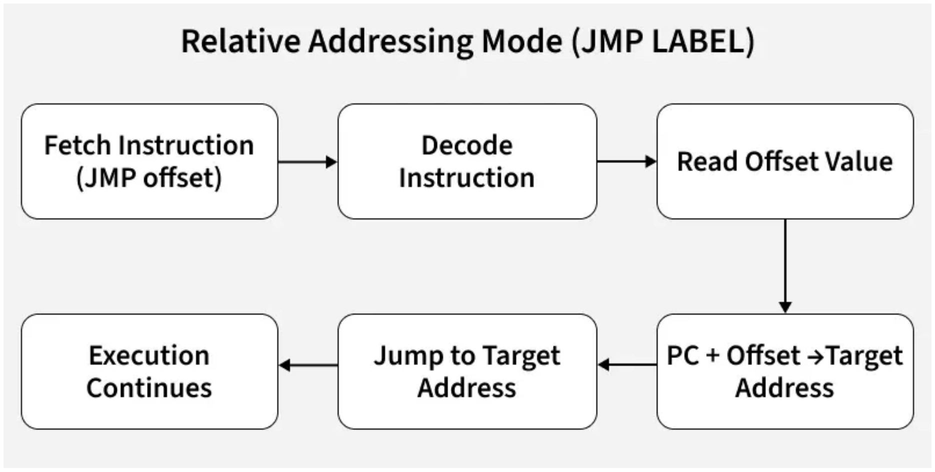
4. Relative Addressing Mode:

Feature: Operand is an offset relative to the program counter, commonly used for branching or looping.

Description: In this mode, the effective address is determined by adding a constant value (offset) to the contents of the program counter (PC).

Examples:

- *MOV R0,#05H*
- *AGAIN: MVI A,#55H*
- *ADD A,R0*
- *JMP AGAIN*


Relative Addressing Mode (JMP LABEL)
Fetch Instruction (JMP offset)
Decode Instruction
Read Offset Value
PC + Offset →Target Address
Jump to Target Address
Execution Continues

Here, JMP AGAIN uses Relative Addressing Mode by jumping to a location relative to the current PC value, enabling loops or conditional branches.

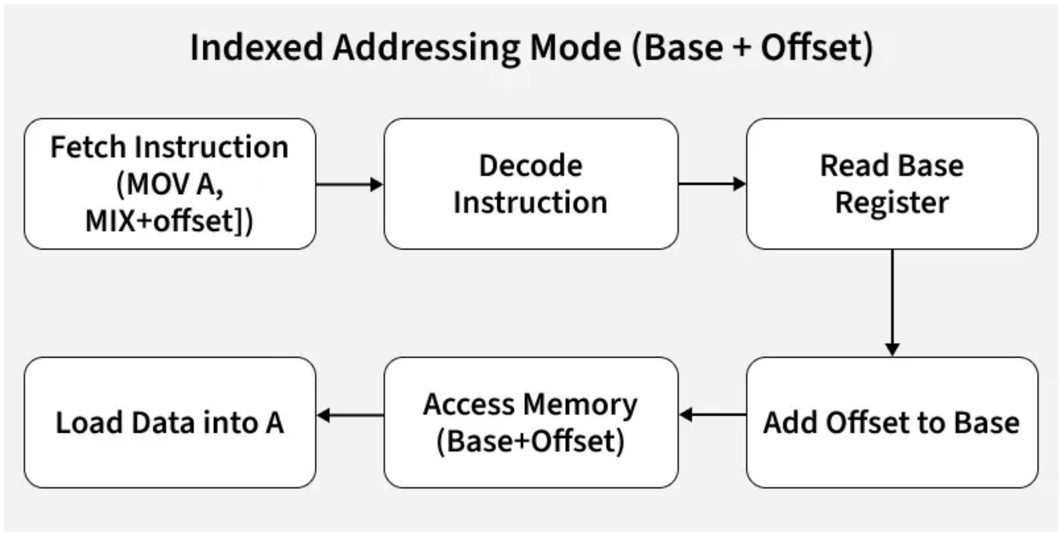
5. Indexed Addressing Mode:

Feature: Effective address = base address + offset, making it useful for accessing arrays or tables.

Description: This mode is typically used to access sequential data structures in memory such as arrays or lookup tables. The base address is stored in a register, and an offset is added to it to calculate the final memory address of the operand.

Example:

- *Suppose HL register pair holds the base address of an array.*
- *An offset is added to HL to access a specific element.*
- *This allows easy traversal of arrays or tables without manually calculating each address.*


Indexed Addressing Mode (Base + Offset)
Fetch Instruction (MOV A, MIX+offset])
Decode Instruction
Read Base Register
Add Offset to Base
Access Memory (Base+Offset)
Load Data into A

6. Memory-Mapped I/O Addressing Mode

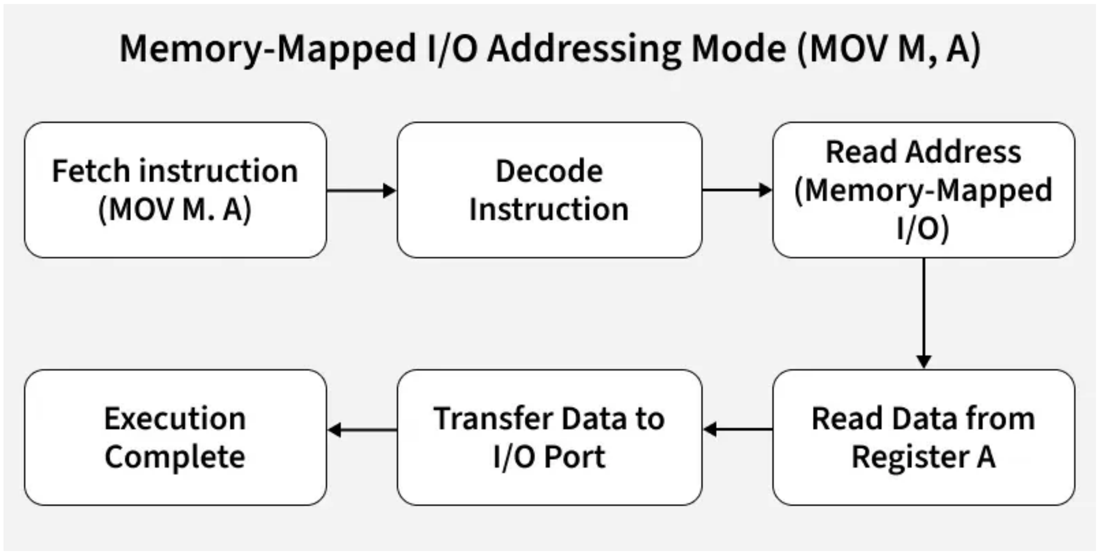
Feature: I/O devices are accessed through memory addresses instead of special I/O instructions.

Description: In this mode, I/O devices are treated as part of the memory. Each device is assigned a unique memory address, and the same instructions used for memory (like LDA, STA, MOV M,

A) can also be used to read from or write to an I/O device. This simplifies communication with peripherals.

Examples:

- *MOV M, A → Writes data from accumulator to the I/O device (memory-mapped address).*
- *MOV A, M → Reads data from the I/O device into accumulator.*


Memory-Mapped I/O Addressing Mode (MOV M, A)
Fetch instruction (MOV M. A)
Decode Instruction
Read Address (Memory-Mapped I/O)
Read Data from Register A
Transfer Data to I/O Port
Execution Complete

Usage of Addressing Modes

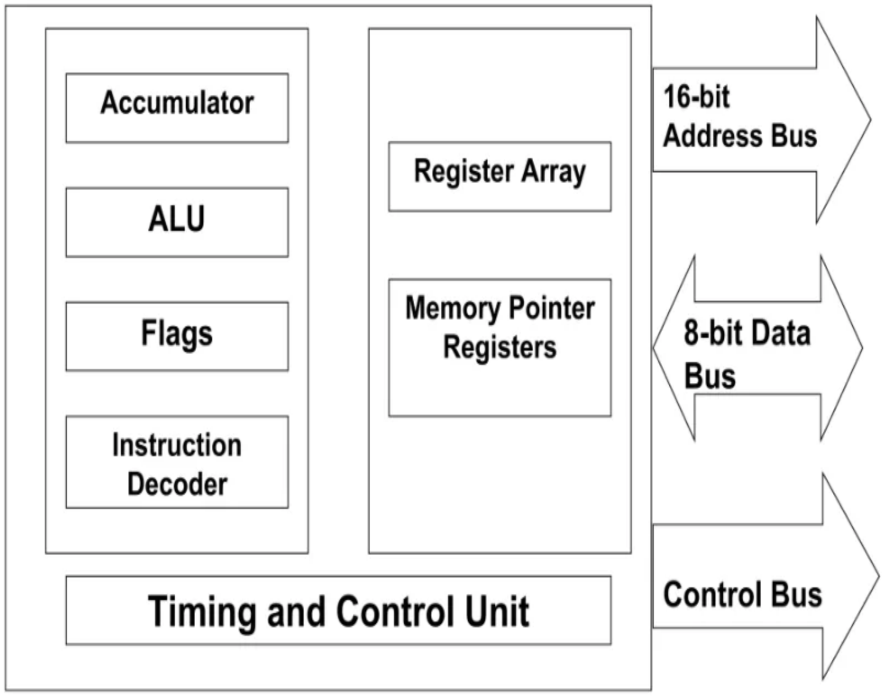
- **Flexibility**: Addressing modes let programmers choose the best way to access data based on its type and size, adapting to different tasks easily.
- **Memory Optimization**: They enable efficient use of memory by minimizing address storage and reducing memory accesses, especially with indirect and indexed addressing.
- **Performance Optimization**: By lowering the number of memory accesses, addressing modes speed up program execution and improve processor efficiency.
- **Reduced Code Size**: Using efficient addressing modes helps write more compact code with fewer instructions.

5.8. Assembly language programming using 8085:

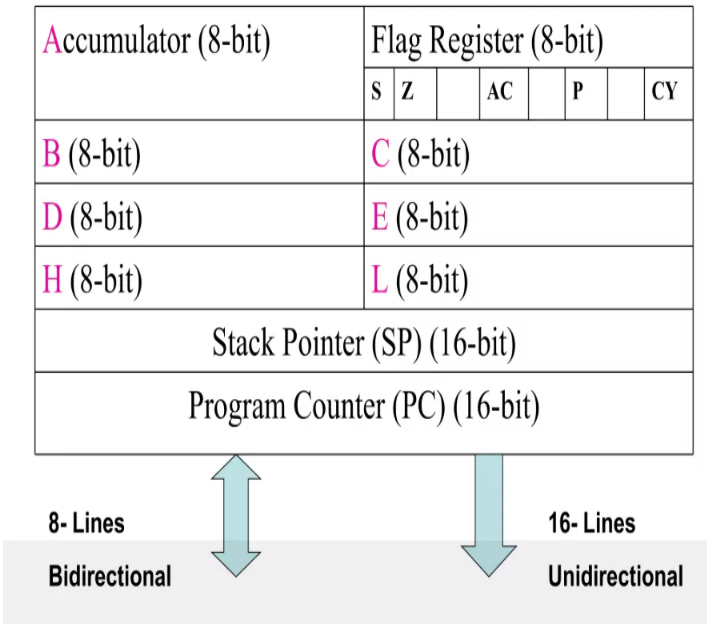
In the Assembly language programming 8085, the 8085 microprocessor's assembly programming language involves writing low-level scripts that control the hardware directly. This form of programming is vital for grasping how microprocessors function, enhancing performance, and executing important real-time applications. The 8085 microprocessor, featuring a straightforward architecture and instruction set, serves as a great foundation for understanding assembly language programming.

Programming Model of 8085

- A microprocessor cannot comprehend an assembly language program.
- A program called Assembler is used to translate programs written in Assembly language into machine language.



Understanding the above model.



Let’s examine the 8085 microprocessor’s programming in the Assembly language programming 8085.

In the Assembly language programming 8085, Instruction sets are sets of codes used to carry out specific tasks. It is divided into five groups.

- **Instructions for Control**: The table that follows lists the control instructions along with their definitions.
- **Reasonable Directions**: The table that follows lists logical instructions along with their definitions.
- **Instructions for Branching:** The table that follows lists the branching instructions along with their definitions.
- **Instructions in Arithmetic**: The table that follows lists arithmetic instructions along with their definitions.
- **Instructions for Data Transfer**: The table that follows lists data-transfer commands along with their definitions.

Demo Programs for 8085:

In the Assembly language programming 8085. Using the instructions above, let's now examine some program demonstrations.

Problem 1: Create an assembly language program that adds two 8-bit values that are kept in the 8085 microprocessor at addresses 2050 and 2051. The program's start address is assumed to be 2000.

The algorithm

- Put the initial integer into the accumulator from memory location 2050.
- Transfer the accumulator's contents to register H.
- It is necessary to load the second number from memory location 2051 into the accumulator.
- The contents of register H and the accumulator are then added using the "ADD" instruction, and the result is stored at 3050.
- The generated carry is retrieved using the "ADC" command and then stored at memory location 3051.

Program:

Memory Address	Mnemonics	Comment
2000	LDA 2050	A ← [2050]
2003	MOV H, A	H ← A
2004	LDA 2051	A ← [2051]
2007	ADD H	A ← A + H
2008	MOV L, A	L ← A
2009	MVI A 00	A ← 00
200B	ADC A	A ← A + A + carry
200C	MOV H, A	H ← A

Memory Address	Mnemonics	Comment
200D	SHLD 3050	H → 3051, L → 3050
2010	HLT	

Explanation of the Above Program:

1. LDA 2050 explains that it transfers the data from the 2050 memory location to the accumulator. (Opcode: 3A 50 20)
2. MOV H, A duplicates the contents of the Accumulator in order to register H to A. (Opcode: 67)
3. LDA 2051 transfers the contents of the 2051 memory location to the accumulator. (Opcode: 3A 51 20)
4. The contents of the H register (F9) and A (Accumulator) are added by using ADD H. The outcome is kept in A itself. All arithmetic instructions by default use A as an operand, and the result is saved as well (Opcode: 84).
5. The contents of A (34) are copied to L via MOV L, A (Opcode: 6F).
6. MVI Data (i.e., 00) is moved immediately to A by A 00 (Opcode: 3E 00).
7. The contents of the given register (A), the contents of A(00), and carry (1) are added by ADC A. A is by default an operand and saves the result because ADC is also an arithmetic operation (Opcode: 8F).
8. MOV H, A transfers A (01)'s contents to H (Opcode: 67).
9. SHLD 3050 is used to move the contents of the L register (34) to the 3050 memory location and the contents of the H register (01) to the 3051 memory address (Opcode: 22 50 30).
10. HLT ends the program's execution and prevents it from continuing (Opcode: 76).

Functional Units of 8085 Microprocessors

The following functional units make up 8085 in the Assembly language programming 8085 are:

The Accumulator

Arithmetic, logical, I/O, and LOAD/STORE operations are all carried out by this 8-bit register. It is linked to the ALU and internal data bus.

Arithmetic Unit and Logic Unit

As the name implies, it works with 8-bit data to carry out arithmetic and logical operations such as addition, subtraction, AND, OR, etc.

General purposes Register

The 8085 CPU has six general-purpose registers: B, C, D, E, H, and L. 8-bit data can be stored in each register.

These registers have the ability to store 16-bit data in pairs, and their pairing combinations are similar to B-C, D-E, and H-L.

Counter for the program

The memory address location of the subsequent instruction to be executed is stored in this 16-bit register. The microprocessor advances the program each time an instruction is executed, pointing the program counter to the memory address of the next instruction that has to be executed.

Stack pointer

Additionally, it is a 16-bit register that functions similarly to a stack and is always increased or decreased by two when push and pop operations are performed.

Temporary register

In the Assembly language programming 8085, the temporary data for arithmetic and logical operations is stored in this 8-bit register.

Flag Register

Depending on the value stored in the accumulator, the 8-bit register's five 1-bit flip-flops can contain either 0 or 1.

These are the five flip-flops in the collection.

- Sign (S)
- Zero (Z)
- Auxiliary Carry (AC)
- Parity (P)
- Carry (C)

Decoder and instruction register

In the Assembly language programming 8085, the 8085 register has eight bits. An instruction is saved in the instruction register once it has been fetched from memory. An instruction decoder decodes the information stored in the instruction register.

Control of interruptions

In the Assembly language programming 8085, As the name suggests, it controls disruptions during a process. Every time an interrupt occurs while a microprocessor is running a main program, it switches control from the main program to handle the incoming request. When the request is complete, the control goes back to the main program.

The 8085 microprocessor has five interrupt signals: TRAP, RST 5.5, RST 7.5, RST 6.5, and INTR.

Control of serial input/output

These two instructions—SID (Serial input data) and SOD (Serial output data)—are used to regulate serial data communication.

Data bus and address bus

The data is transported to storage via a data bus. It is bidirectional, while the address bus is unidirectional and transports the location to the storage location. It is employed to address I/O devices and transfer data.

The address-data buffer and the address buffer

To connect with the CPU, the information kept in the program counter and stack pointer is loaded into the address buffer and address-data buffer. These buses connect the memory and I/O chips, allowing the CPU to exchange the needed data with them.

5.9. Programs for addition, subtraction, multiplication and division (8-Bit only):

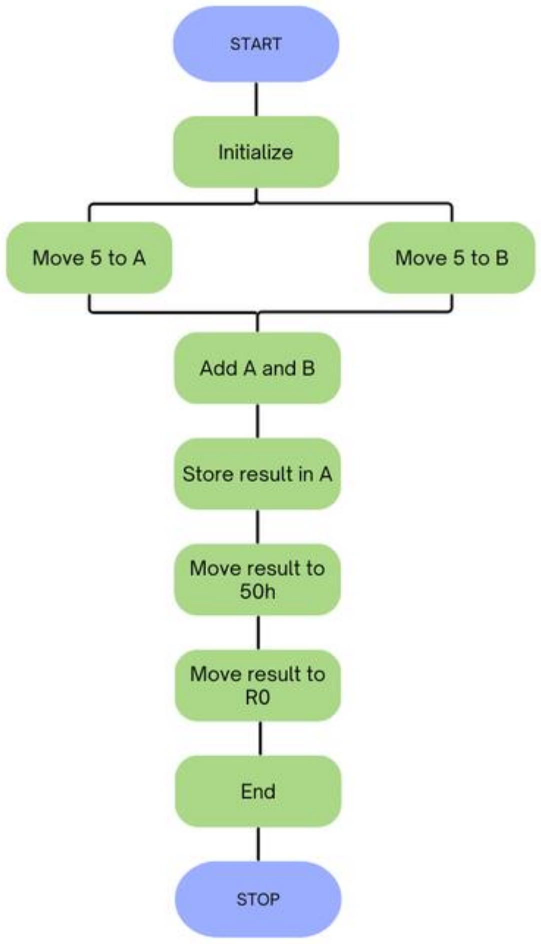
5.5.1.Program for 8-bit Addition:

```
;program for 8-bit addition in assembly language programming
org 0000
mov a,#5
mov b,#5
add a,b
mov 50h,a
mov r0,50h
end
```

Program Description

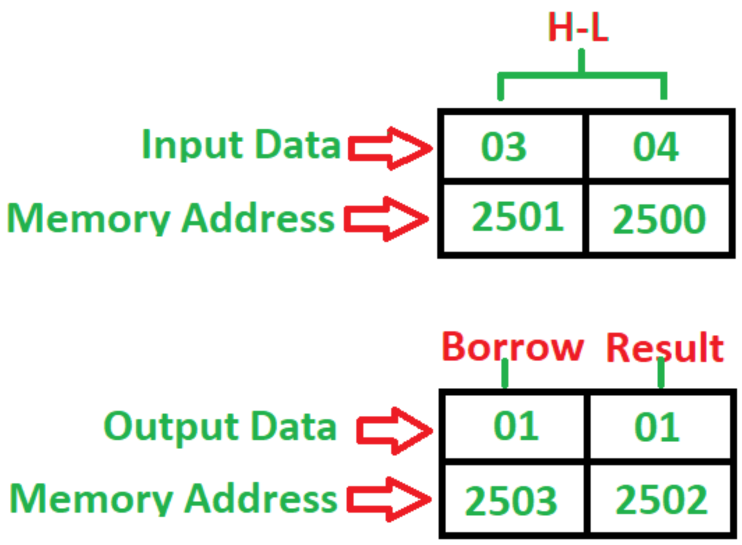
- ORG 0000H means it set the statement at memory address 0000H.
- mov a,#5 using immediate Addressing mode we are transferring hexadecimal '5' to accumulator.
- mov b,#5 Same as above we are transferring hexadecimal 5 to b.
- add a,b here add opcode will adds the data of a and b
- mov 50h,a this line uses to store the output to the memory address 50h from accumulator.
- mov r0,50h this line uses to store the output to the register from memory 50h.
- end This 'end' opcode to stops the program
- To Write comments in Assembly language we use a ";".

Flowchart of program



5.9.2. Program for 8-bit Subtraction:

Problem : Write a program to subtract two 8-bit numbers with or without borrow where first number is at 2500 memory address and second number is at 2501 memory address and store the result into 2502 and borrow into 2503 memory address.



Example

Algorithm for 8085 program to subtract two 8-bit numbers with or without borrow

1. Load 00 in a register C (for borrow)
2. Load two 8-bit number from memory into registers
3. Move one number to accumulator
4. Subtract the second number with accumulator
5. If borrow is not equal to 1, go to step 7
6. Increment register for borrow by 1
7. Store accumulator content in memory
8. Move content of register into accumulator
9. Store content of accumulator in other memory location
10. Stop

Program

Memory	Mnemonics	Operands	Comment

Memory	Mnemonics	Operands	Comment
2000	MVI	C, 00	[C] <- 00
2002	LHLD	2500	[H-L] <- [2500]
2005	MOV	A, H	[A] <- [H]
2006	SUB	L	[A] <- [A] - [L]
2007	JNC	200B	Jump If no borrow
200A	INR	C	[C] <- [C] + 1
200B	STA	2502	[A] -> [2502], Result
200E	MOV	A, C	[A] <- [C]
2010	STA	2503	[A] -> [2503], Borrow
2013	HLT		Stop

Explanation

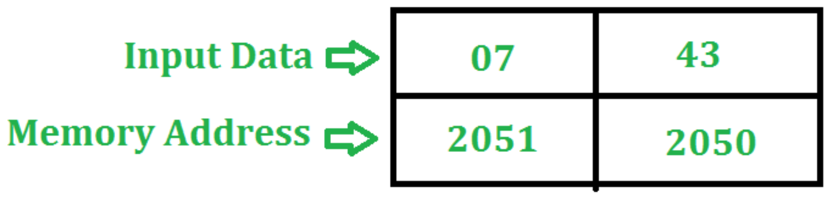
Registers A, H, L, C are used for general purpose:

1. MOV is used to transfer the data from memory to accumulator (1 Byte)
2. LHLD is used to load register pair directly using 16-bit address (3 Byte instruction)
3. MVI is used to move data immediately into any of registers (2 Byte)
4. STA is used to store the content of accumulator into memory(3 Byte instruction)
5. INR is used to increase register by 1 (1 Byte instruction)
6. JNC is used to jump if no borrow (3 Byte instruction)
7. SUB is used to subtract two numbers where one number is in accumulator(1 Byte)
8. HLT is used to halt the program

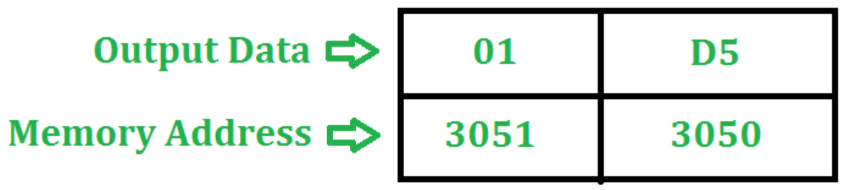
5.9.3. Program for 8-bit Multiplication:

Problem - Multiply two 8 bit numbers stored at address 2050 and 2051. Result is stored at address 3050 and 3051. Starting address of program is taken as 2000.

Example -



Input Data ⇨	07	43
Memory Address ⇨	2051	2050



Output Data ⇨	01	D5
Memory Address ⇨	3051	3050

Algorithm -

1. We are taking adding the number 43 seven(7) times in this example.
2. As the multiplication of two 8 bit numbers can be maximum of 16 bits so we need register pair to store the result.

Program

Memory Address	Mnemonics	Comment
2000	LHLD 2050	H←2051, L←2050
2003	XCHG	H↔D, L↔E
2004	MOV C, D	C←D
2005	MVI D 00	D←00
2007	LXI H 0000	H←00, L←00
200A	DAD D	HL←HL+DE
200B	DCR C	C←C-1
200C	JNZ 200A	If Zero Flag !=0, goto 200A

Memory Address	Mnemonics	Comment
200F	SHLD 3050	H→3051, L→3050
2012	HLT	

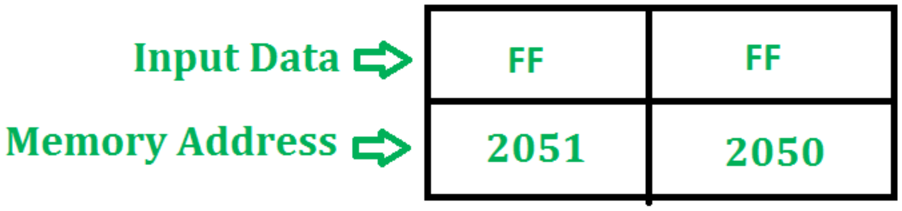
Explanation - Registers used: A, H, L, C, D, E

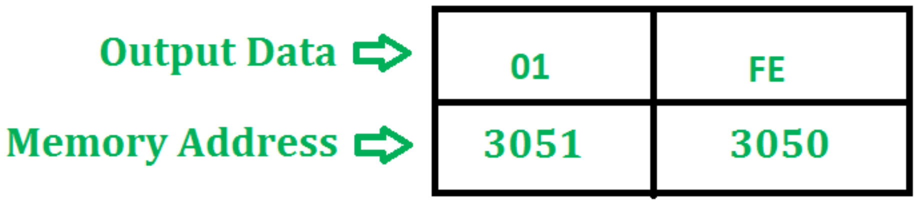
1. LHLD 2050 loads content of 2051 in H and content of 2050 in L
2. XCHG exchanges contents of H with D and contents of L with E
3. MOV C, D copies content of D in C
4. MVI D 00 assigns 00 to D
5. LXI H 0000 assigns 00 to H and 00 to L
6. DAD D adds HL and DE and assigns the result to HL
7. DCR C decrements C by 1
8. JNZ 200A jumps program counter to 200A if zero flag != 0 (Not equal to 0)
9. SHLD stores value of H at memory location 3051 and L at 3050
10. HLT stops executing the program and halts any further execution.

5.9.4. Program for 8-bit Division:

Problem - Write 8085 program to divide two 8 bit numbers.

Example -





Algorithm -

1. Start the program by loading the HL pair registers with address of memory location.
2. Move the data to B Register.
3. Load the second data into accumulator.
4. Compare the two numbers to check carry.
5. Subtract two numbers.
6. Increment the value of carry.
7. Check whether the repeated subtraction is over.
8. Then store the results(quotient and remainder) in given memory location.
9. Terminate the program.

Program

ADDRESS	MNEMONICS	COMMENT
2000	LXI H, 2050	

ADDRESS	MNEMONICS	COMMENT
2003	MOV B, M	B<-M
2004	MVI C, 00	C<-00H
2006	INX H	
2007	MOV A, M	A<-M
2008	CMP B	
2009	JC 2011	check for carry
200C	SUB B	A<-A-B
200D	INR C	C<-C+1
200E	JMP 2008	
2011	STA 3050	3050<-A
2014	MOV A, C	A<-C
2015	STA 3051	3051<-A

ADDRESS	MNEMONICS	COMMENT
2018	HLT	terminate the program

Explanation - Registers A, H, L, C, B are used for general purpose.

1. LXI H, 2050 will load the HL pair register with the address 2050 of memory location.
2. MOV B, M copies the content of memory into register B.
3. MVI C, 00 assign 00 to C.
4. INX H increment register pair HL.
5. MOV A, M copies the content of memory into accumulator.
6. CMP B compares the content of accumulator and register B.
7. JC 2011 jump to address 2011 if carry flag is set.
8. SUB B subtract the content of accumulator with register B and store the result in accumulator.
9. INR C increment the register C.
10. JMP 2008 control will shift to memory address 2008.
11. STA 3050 stores the remainder at memory location 3050.
12. MOV A, C copies the content of register into accumulator.
13. STA 3051 stores the quotient at memory location 3051.

14. HLT stops executing the program and halts any further execution.