5TH SEMESTER ELECTRICAL ENGINEERING

UNIT-1BASICS OF DIGITAL ELECTRONICS

Introduction to Digital Electronics:

- Digital electronics deals with the electronic manipulation of numbers, or with the manipulation of varying quantities by means of numbers.
- Because it is convenient to do so, today's digital systems deal only with the numbers 'zero' and 'one', because they can be represented easily by 'off and 'on' within a circuit.
- This is not the limitation it might seem, for the binary system of counting can be used to represent any number that we can represent with the usual decimal (0 to 9) system that we use in everyday life.
- Digital Electronics is very important in today's life because if digital circuits compared to analog circuits are that signals represented digitally can be transmitted without degradation due to noise.

Advantages of Digital Circuits

- High accuracy and programmability
- Storage of digital data is easy
- Immune to noise
- Can be implemented in the form of integrated circuits (ICs)
- Greater reliability and flexibility

Disadvantages of Digital Circuits

- Expensive
- Operate on digital signals only
- Complex circuitry

Applications of Digital Circuits

- Mobile Phones, Calculators and Digital Computers
- Radios and communication Devices
- Signal Generator
- Smart Card
- Cathode Ray Oscilloscope (CRO)
- Analog to digital converters (ADC)
- Digital to analog converters (DAC), etc.

Number system:

- A number system is defined as a system of writing to express numbers.
- It is the mathematical notation for representing numbers of a given set by using digits or other symbols in a consistent manner.
- It provides a unique representation of every number and represents the arithmetic and algebraic structure of the figures.
- It also allows us to operate arithmetic operations like addition, subtraction and division. The value of any digit in a number can be determined by:
- The digit
- Its position in the number
- The base of the number system **Types of number system:**

There are various types of number systems in mathematics. The four most common number system types are:

- Decimal number system (Base- 10)
- Binary number system (Base- 2)
- Octal number system (Base-8)

- Hexadecimal number system (Base- 16) A number N in base or radix ‘r’ can be written as:

$(N)_b = d_{n-1}\ d_{n-2}$ -- -- -- -- $d_1\ d_0\ .\ d_{-1}\ d_{-2}$ ------------------------- d_{-m}

In the above, dn-1 to d0 is the integer part, then follows a radix point, and then d-1 to d-m is the fractional part.

D_{n-1} = Most significant bit (MSB) d_{-m}
= Least significant bit (LSB)

Decimal number system:

The base or radix of Decimal number system is 10. So, the numbers ranging from 0 to 9 are used in this number system. Mathematically, we can write it as

$1358.246 = (1 \times 10^3) + (3 \times 10^2) + (5 \times 10^1) + (8 \times 10^0) + (2 \times 10^{-1}) + (4 \times 10^{-2}) + (6 \times 10^{-3})$

Binary number system:

All digital circuits and systems use this binary number system. The base or radix of this number system is 2. So, the numbers 0 and 1 are used in this number system.
Mathematically, we can write it as

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

Octal number system:

The base or radix of octal number system is 8. So, the numbers ranging from 0 to 7 are used in this number system.
Mathematically, we can write it as

$1457.236 = (1 \times 8^3) + (4 \times 8^2) + (5 \times 8^1) + (7 \times 8^0) + (2 \times 8^{-1}) + (3 \times 8^{-2}) + (6 \times 8^{-3})$

Hexadecimal number system:

The base or radix of Hexa-decimal number system is 16. So, the numbers ranging from 0 to 9 and the letters from A to F are used in this number system. The decimal equivalent of Hexadecimal digits from A to F are 10 to 15.
Mathematically, we can write it as

$1A05.2C4 = (1 \times 16^3) + (10 \times 16^2) + (0 \times 16^1) + (5 \times 16^0) + (2 \times 16^{-1}) + (12 \times 16^{-2}) + (4 \times 16^{-3})$

Conversion from one system to another number system:

Decimal number system to other number system:

If the decimal number contains both integer part and fractional part, then convert both the parts of decimal number into another base individually. Steps for converting the decimal number into its equivalent number of any base ‘r’-

- Do division of integer part of decimal number and **successive quotients** with base ‘r’ and note down the remainders till the quotient is zero. Consider the remainders in reverse order to get the integer part of equivalent number of base ‘r’. That means, first and last remainders denote the least significant digit and most significant digit respectively.
- Do multiplication of fractional part of decimal number and **successive fractions** with base ‘r’ and note down the carry till the result is zero or the desired number of equivalent digits is obtained. Consider the normal sequence of carry in order to get the fractional part of equivalent number of base ‘r’.

Decimal to binary:

Example- $(152.25)_{10}$Step 1:

Divide the number 152 and its successive quotients with base 2.

Operation	Quotient	Remainder

152/2	76	0 (LSB)
76/2	38	0
38/2	19	0
19/2	9	1
9/2	4	1
4/2	2	0
2/2	1	0
1/2	0	1(MSB)

$(152)_{10}$ = $(10011000)_2$ Step 2:

Now, perform the multiplication of 0.27 and successive fraction with base 2.

Operation	Result	carry
0.25×2	0.50	0
0.50×2	0	1

$(0.25)_{10}$ = $(.01)_2$

Decimal to octal:

Example- $(152.25)_{10}$Step 1:

Divide the number 152 and its successive quotients with base 8.

Operation	Quotient	Remainder
152/8	19	0
19/8	2	3
2/8	0	2

$(152)_{10}$ = $(230)_8$

Step 2:

Now perform the multiplication of 0.25 and successive fraction with base 8.

Operation	Result	carry
0.25×8	**0**	**2**

$(0.25)_{10}$=$(2)_8$

So, the octal number of the decimal number 152.25 is **230.2** **Decimal to hexadecimal:**

Example- $(152.25)_{10}$ Step 1:

Divide the number 152 and its successive quotients with base 8.

Operation	Quotient	Remainder
152/16	9	8
9/16	0	9

$(152)_{10}$ = $(98)_{16}$

Step 2:

Now perform the multiplication of 0.25 and successive fraction with base 16.

Operation	Result	carry
0.25×16	0	4

$(0.25)_{10}$ = $(4)_{16}$

So, the hexadecimal number of the decimal number 152.25 is **230.4.** **Binary to other number system:**

Binary to decimal:

The process starts from multiplying the bits of binary number with its corresponding positional weights. And lastly, we add all those products.

Example- $(10110.001)_2$

$(10110.001)_2$ $=(1\times2^4)+(0\times2^3)+(1\times2^2)+(1\times2^1)+(0\times2^0)+(0\times2^{-1})+(0\times2^{-2})+(1\times2^{-3})$

$(10110.001)_2$ $=(1\times16)+(0\times8)+(1\times4)+(1\times2)+(0\times1)+(0\times1/2)+(0\times1/4)+(1\times1/8)$

$(10110.001)_2$=16+0+4+2+0+0+0+0.125

$(10110.001)_2$=$(22.125)_{10}$**Binary to octal:**

In a binary number, the pair of three bits is equal to one octal digit. Two steps to convert a binary number into an octal number which are as follows:

- In the first step, we have to make the pairs of three bits on both sides of the binary point. If there will be one or two bits left in a pair of three bits pair, we add the required number of zeros on extreme sides.
- In the second step, we write the octal digits corresponding to each pair. **Example- $(111110101011.0011)_2$**

1. Firstly, we make pairs of three bits on both sides of the binary point. 111
110 101 011.001 1
On the right side of the binary point, the last pair has only one bit. To make it a complete pair of three bits, we added two zeros on the extreme side.
111 110 101 011.001 100 2. Then, we wrote the octal digits, which correspond to each pair.
$(111110101011.0011)_2$ = $(7653.14)_8$ **Binary to hexadecimal:**

The base numbers of binary and hexadecimal are 2 and 16, respectively. In a binary number, the pair of four bits is equal to one hexadecimal digit. There are also only two steps to convert a binary number into a hexadecimal number which are as follows:

1. In the first step, we have to make the pairs of four bits on both sides of the binary point. If there will be one, two, or three bits left in a pair of four bits pair, we add the required number of zeros on extreme sides.
2. In the second step, we write the hexadecimal digits corresponding to each pair. **Example- $(10110101011.0011)_2$**

1. Firstly, we make pairs of four bits on both sides of the binary point.
111 1010 1011.0011
On the left side of the binary point, the first pair has three bits. To make it a complete pair of four bits, add one zero on the extreme side.
0111 1010 1011.0011
2. Then, we write the hexadecimal digits, which correspond to each pair.
$(011110101011.0011)_2$=$(7AB.3)_{16}$Octal to other number system:

Octal to decimal:

The process starts from multiplying the digits of octal numbers with its corresponding positional weights. And lastly, we add all those products.
Example- $(152.25)_8$ Step 1:
We multiply each digit of **152.25** with its respective positional weight, and last, we add the products of all the bits with its weight.
$(152.25)_8$ $=(1\times8^2)+(5\times8^1)+(2\times8^0)+(2\times8^{-1})+(5\times8^{-2})$
$(152.25)_8$ $=64+40+2+(2\times1/8)+(5\times1/64)$
$(152.25)_8$ $=64+40+2+0.25+0.078125$
$(152.25)_8$ $=106.328125$
So, the decimal number of the octal number 152.25 is **106.328125Octal to binary:**

The process of converting octal to binary is the reverse process of binary to octal. We write the three bits binary code of each octal number digit. **Example- $(152.25)_8$**
We write the three-bit binary digit for 1, 5, 2, and 5.
$(152.25)_8$ = $(001101010.010101)_2$
So, the binary number of the octal number 152.25 is **$(001101010.010101)_2$Octal to hexadecimal:**

For converting octal to hexadecimal, there are two steps required to perform, which are as follows:

1. In the first step, we will find the binary equivalent of number.
2. Next, we have to make the pairs of four bits on both sides of the binary point. If there will be one, two, or three bits left in a pair of four bits pair, we add the required number of

zeros on extreme sides and write the hexadecimal digits corresponding to each pair.

Example- $(152.25)_8$ Step 1:

We write the three-bit binary digit for 1, 5, 2, and 5.

$(152.25)_8$ $=(001101010.010101)_2$

So, the binary number of the octal number 152.25 is **$(001101010.010101)_2$ Step 2:**

1. Now, we make pairs of four bits on both sides of the binary point.

0 0110 1010.0101 01

On the left side of the binary point, the first pair has only one digit, and on the right side, the last pair has only two-digit. To make them complete pairs of four bits, add zeros on extreme sides.

0000 0110 1010.0101 0100

2. Now, we write the hexadecimal digits, which correspond to each pair.

$(0000\ 0110\ 1010.0101\ 0100)_2 = (6A.54)_{16}$

Hexadecimal to other number system:

Hexadecimal to decimal:

The process of converting hexadecimal to decimal is the same as binary to decimal. The process starts from multiplying the digits of hexadecimal numbers with its corresponding positional weights. And lastly, we add all those products.

Let's take an example to understand how the conversion is done from hexadecimal to decimal.

Example- $(152A.25)_{16}$ Step 1:

We multiply each digit of **152A.25** with its respective positional weight, and last we add the products of all the bits with its weight.

$(152A.25)_{16}=(1\times16^3)+(5\times16^2)+(2\times16^1)+(A\times16^0)+(2\times16^{-1})+(5\times16^{-2})$

$(152A.25)_{16}=(1\times4096)+(5\times256)+(2\times16)+(10\times1)+(2\times16^{-1})+(5\times16^{-2})$

$(152A.25)_{16}=4096+1280+32+10+(2\times1/16)+(5\times1/256)$

$(152A.25)_{16}=5418+0.125+0.125$

$(152A.25)_{16}=5418.14453125$

So, the decimal number of the hexadecimal number 152A.25 is **5418.14453125**

Hexadecimal to binary:

The process of converting hexadecimal to binary is the reverse process of binary to hexadecimal. We write the four bits binary code of each hexadecimal number digit. **Example - $(152A.25)_{16}$**

We write the four-bit binary digit for 1, 5, A, 2, and 5.

$(152A.25)_{16}= (0001\ 0101\ 0010\ 1010.0010\ 0101)_2$

So, the binary number of the hexadecimal number 152.25 is **$(1010100101010.00100101)_2$**

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

Hexadecimal to octal:

For converting hexadecimal to octal, there are two steps required to perform, which are as follows:

1. In the first step, we will find the binary equivalent of the hexadecimal number.
2. Next, we have to make the pairs of three bits on both sides of the binary point. If there will be one or two bits left in a pair of three bits pair, we add the required number of zeros on extreme sides and write the octal digits corresponding to each pair.

Example- $(152A.25)_{16}$Step 1:

We write the four-bit binary digit for 1, 5, 2, A, and 5.

$(152A.25)_{16}$ = $(0001\ 0101\ 0010\ 1010.0010\ 0101)_2$

So, the binary number of hexadecimal number 152A.25 is **$(0011010101010.010101)_2$Step 2:**

3. Then, we make pairs of three bits on both sides of the binary point.

001 010 100 101 010.001 001 010

4. Then, we write the octal digit, which corresponds to each pair.

$(001010100101010.001001010)_2 = (12452.112)_8$

So, the octal number of the hexadecimal number 152A.25 is **12452.112**

Arithmetic operation:

Two types of operation that are performed on binary data include arithmetic and logic operations. Basic arithmetic operations include addition, subtraction, multiplication and division.

Binary addition:

There are four rules for binary addition:

Input A	Input B	Sum (S) A+B	Carry (C)
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

Example-

0011010 + 001100 = 00100110

```
   1 1          carry
  0 0 1 1 0 1 0   = 26₁₀
 +0 0 0 1 1 0 0   = 12₁₀
 ---------------
  0 1 0 0 1 1 0   = 38₁₀
```

Binary subtraction:

There are four rules for binary subtraction:

Input A	Input B	Subtract (S) A-B	Borrow (B)
0	0	0	0
0	1	0	1
1	0	1	0
1	1	0	0

Example-

0011010 - 001100 = 00001110

```
      1 1       borrow
  0 0 1 1 0 1 0   = 26₁₀
 -0 0 0 1 1 0 0   = 12₁₀
 ---------------
  0 0 0 1 1 1 0   = 14₁₀
```

Binary multiplication:

There are four rules of binary multiplication.

Input A	Input B	Multiply (M) AxB
0	0	0
0	1	0
1	0	0
1	1	1

Example:

0011010 x 001100 = 100111000

```
        0 0 1 1 0 1 0   = 26₁₀
       x0 0 0 1 1 0 0   = 12₁₀
  ----------------------
        0 0 0 0 0 0 0
      0 0 0 0 0 0 0
    0 0 1 1 0 1 0
  0 0 1 1 0 1 0
  ----------------------
  0 1 0 0 1 1 1 0 0 0   = 312₁₀
```

Binary division:

There are four parts in any division: Dividend, Divisor, quotient, and remainder.

Input A	Input B	Divide (D) A/B
0	0	Not defined
0	1	0
1	0	Not defined
1	1	1

Example-

101010 / 000110 = 000111

```
              1 1 1        = 7₁₀
000110 ) 1 0 1 0 1 0       = 42₁₀
        -1 1 0             = 6₁₀
         1 0 0 1
        -1 1 0
           1 1 0
          -1 1 0
               0
```

Signed binary number representation:

- In mathematics, positive numbers (including zero) are represented as unsigned numbers.
- That is, we do not put the +ve sign in front of them to show that they are positive numbers.
- However, when dealing with negative numbers we do use a -ve sign in front of the number to show that the number is negative in value and different from a positive unsigned value, and the same is true with signed binary numbers.
- However, in digital circuits there is no provision made to put a plus or even a minus sign to a number, since digital systems operate with binary numbers that are represented in terms of "0's" and "1's".
- For signed binary numbers the most significant bit (MSB) is used as the sign bit.
- If the sign bit is "0", this means the number is positive in value.
- If the sign bit is "1", then the number is negative in value.

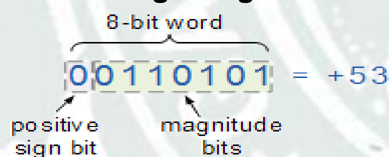
3 ways to represent negative binary number-

1. Sign magnitude
2. 1's compliment
3. 2' compliment **Sign magnitude:**

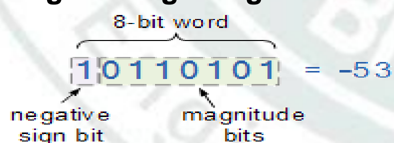
Left most digit is used to indicate the sign and the remaining digits the magnitude or value of the number.

Example-

Positive sign magnitude:



Negative sign magnitude:



The disadvantage here is that whereas before we had a full range n-bit unsigned binary number, we now have an n-1 bit signed binary number giving a reduced range of digits from:

$$-2_{(n-1)} \text{ to } +2_{(n-1)}$$

1's complement:

- The one's complement of a negative binary number is the complement of its positive counterpart.
- Thus, the one's complement of "1" is "0" and vice versa, then the one's complement of 100101002 is simply 011010112 as all the 1's are changed to 0's and the 0's to 1's.

For representing the positive numbers, there is nothing to do.

- But for representing negative numbers, we have to use 1's complement technique.
- For representing the negative number, we first have to represent it with a positive sign, and then we find the 1's complement of it.

Example- 11010.1101

For finding 1's complement of the given number, change all 0's to 1 and all 1's to 0. So, the 1's complement of the number 11010.1101 comes out 00101.0010. **2's complement:**

- 2's complement is also used to represent the signed binary numbers.
- For finding 2's complement of the binary number, we will first find the 1's complement of the binary number and then add 1 to the least significant bit of it.

Example- 110100

For finding 2's complement of the given number, change all 0's to 1 and all 1's to 0. So, the 1's complement of the number 110100 is 001011. Now add 1 to the LSB of this number, i.e., (001011)+1=001100.

Addition and subtraction using 1's complement:

There are three different cases possible when we add two binary numbers which are as follows:

Case 1: Addition of the positive number with a negative number when the positive number has a greater magnitude.

Initially, calculate the 1's complement of the given negative number. Sum up with the given positive number. If we get the end-around carry 1, it gets added to the LSB.

Example: 1101 and -1001

- First, find the 1's complement of the negative number 1001. So, for finding 1's complement, change all 0 to 1 and all 1 to 0. The 1's complement of the number 1001 is 0110.
- Now, add both the numbers, i.e., 1101 and 0110; 1101+0110=1 0011
- By adding both numbers, we get the end-around carry 1. We add this end around carry to the LSB of 0011.
  0011+1=0100

Case 2: Adding a positive value with a negative value in case the negative number has a higher magnitude.

Initially, calculate the 1's complement of the negative value. Sum it with a positive number. In this case, we did not get the end-around carry. So, take the 1's complement of the result to get the final result.

Note: The resultant is a negative value.

Example: 1101 and -1110

- First find the 1's complement of the negative number 1110. So, for finding 1's complement, we change all 0 to 1, and all 1 to 0. 1's complement of the number 1110 is 0001.
- Now, add both the numbers, i.e., 1101 and 0001; 1101+0001= 1110
- Now, find the 1's complement of the result 1110 that is the final result. So, the 1's complement of the result 1110 is 0001, and we add a negative sign before the number so that we can identify that it is a negative number.

Case 3: Addition of two negative numbers

In this case, first find the 1's complement of both the negative numbers, and then we add both these complement numbers. In this case, we always get the end-around carry, which get added to the LSB, and for getting the final result, we take the 1's complement of the result.

Note: The resultant is a negative value.

Example: -1101 and -1110 in five-bit register

- Firstly, find the 1's complement of the negative numbers 01101 and 01110. So, for finding 1's complement, we change all 0 to 1, and all 1 to 0. 1's complement of the number 01110 is 10001, and 01101 is 10010.

- Now, we add both the complement numbers, i.e., 10001 and 10010;
  10001+10010= 1 00011
- By adding both numbers, we get the end-around carry 1. We add this end-around carry to the LSB of 00011.
  00011+1=00100
- Now, find the 1's complement of the result 00100 that is the final answer. So, the 1's complement of the result 00100 is 110111, and add a negative sign before the number so that we can identify that it is a negative number.

Addition and subtraction using 2's complement:

There are three different cases possible when we add two binary numbers using 2's complement, which is as follows:

Case 1: Addition of the positive number with a negative number when the positive number has a greater magnitude.

Initially find the 2's complement of the given negative number. Sum up with the given positive number. If we get the end-around carry 1 then the number will be a positive number and the carry bit will be discarded and remaining bits are the final result.

Example: 1101 and -1001

- First, find the 2's complement of the negative number 1001. So, for finding 2's complement, change all 0 to 1 and all 1 to 0 or find the 1's complement of the number 1001. The 1's complement of the number 1001 is 0110, and add 1 to the LSB of the result 0110. So the 2's complement of number 1001 is 0110+1=0111
- Add both the numbers, i.e., 1101 and 0111;
  1101+0111=1 0100
- By adding both numbers, we get the end-around carry 1. We discard the end-around carry. So, the addition of both numbers is 0100.

Case 2: Adding of the positive value with a negative value when the negative number has a higher magnitude.

Initially, add a positive value with the 2's complement value of the negative number. Here, no end-around carry is found. So, we take the 2's complement of the result to get the final result.

Note: The resultant is a negative value.

Example: 1101 and -1110

- First, find the 2's complement of the negative number 1110. So, for finding 2's complement, add 1 to the LSB of its 1's complement value 0001.
  0001+1=0010
- Add both the numbers, i.e., 1101 and 0010;   1101+0010= 1111
- Find the 2's complement of the result 1110 that is the final result. So, the 2's complement of the result 1110 is 0001, and add a negative sign before the number so that we can identify that it is a negative number.

Case 3: Addition of two negative numbers

In this case, first, find the 2's complement of both the negative numbers, and then we will add both these complement numbers. In this case, we will always get the end-around carry, which will be added to the LSB, and forgetting the final result, we will take the2's complement of the result.

Note: The resultant is a negative value.

Example: -1101 and -1110 in five-bit register

- Firstly, find the 2's complement of the negative numbers 01101 and 01110. So, for finding 2's complement, we add 1 to the LSB of the 1's complement of these numbers. 2's complement of the number 01110 is 10010, and 01101 is 10011.
- We add both the complement numbers, i.e., 10001 and 10010; 10010+10011= 1 00101

- By adding both numbers, we get the end-around carry 1. This carry is discarded and the final result is the 2.s complement of the result 00101. So, the 2's complement of the result 00101 is 11011, and we add a negative sign before the number so that we can identify that it is a negative number.

Digital codes:

In the coding, when numbers or letters are represented by a specific group of symbols, it is said to be that number or letter is being encoded. The group of symbols is called as code. The digital data is represented, stored and transmitted as group of bits. This group of bits is also called as binary code.

Advantages of Binary Code:

- Binary codes are suitable for the computer applications.
- Binary codes are suitable for the digital communications.
- Binary codes make the analysis and designing of digital circuits if we use the binary codes.
- Since only 0 & 1 are being used, implementation becomes easy.

Classification of binary codes:

The codes are broadly categorized into following four categories.

- Weighted Codes
- Non-Weighted Codes
- Binary Coded Decimal Code
- Alphanumeric Codes
- Error Detecting Codes
- Error Correcting Codes **Weighted Codes:**

Weighted binary codes are those binary codes which obey the positional weight principle. Each position of the number represents a specific weight. Several systems of the codes are used to express the decimal digits 0 through 9. In these codes each decimal digit is represented by a group of four bits.

Decimal Digit	8421 Code	2421 Code	84-2-1 Code
0	0000	0000	0000
1	0001	0001	0111
2	0010	0010	0110
3	0011	0011	0101
4	0100	0100	0100
5	0101	1011	1011

6	0110	1100	1010
7	0111	1101	1001
8	1000	1110	1000
9	1001	1111	1111

8 4 2 1 code

- The weights of this code are 8, 4, 2 and 1.
- This code has all positive weights. So, it is a **positively weighted code**.
- This code is also called as **natural BCDBinary Coded Decimalcode**.
- In this code each decimal digit is represented by a 4-bit binary number.
- BCD is a way to express each of the decimal digits with a binary code.
- In the BCD, with four bits we can represent sixteen numbers (0000 to 1111).
- But in BCD code only first ten of these are used (0000 to 1001).
- The remaining six code combinations i.e., 1010 to 1111 are invalid in BCD.

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

Example

Let us find the BCD equivalent of the decimal number 786. This number has 3 decimal digits 7, 8 and 6. From the table, we can write the BCD 84218421 codes of 7, 8 and 6 are 0111, 1000 and 0110 respectively.

$$786_{10} = 0111\ 1000\ 0110_{BCD}$$

There are 12 bits in BCD representation, since each BCD code of decimal digit has 4 bits.

Advantages of BCD Codes

- It is very similar to decimal system.
- We need to remember binary equivalent of decimal numbers 0 to 9 only.

Disadvantages of BCD Codes

- The addition and subtraction of BCD have different rules.
- The BCD arithmetic is little more complicated.
- BCD needs more number of bits than binary to represent the decimal number. So, BCD is less efficient than binary.

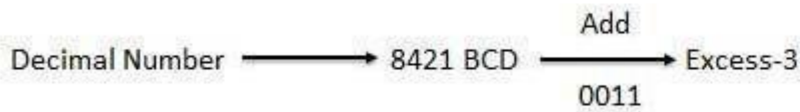
Non-weighted code:

n this type of binary codes, the positional weights are not assigned. The examples of nonweighted codes are Excess-3 code and Gray code.

Excess-3 code

- The Excess-3 code is also called as XS-3 code.
- It is non-weighted code used to express decimal numbers.

- The Excess-3 code words are derived from the 8421 BCD code words adding (0011)2 or (3)10 to each code word in 8421.
- The excess-3 codes are obtained as follows –



Decimal	BCD 8 4 2 1	Excess-3 BCD + 0011
0	0 0 0 0	0 0 1 1
1	0 0 0 1	0 1 0 0
2	0 0 1 0	0 1 0 1
3	0 0 1 1	0 1 1 0
4	0 1 0 0	0 1 1 1
5	0 1 0 1	1 0 0 0
6	0 1 1 0	1 0 0 1
7	0 1 1 1	1 0 1 0
8	1 0 0 0	1 0 1 1
9	1 0 0 1	1 1 0 0

Gray Code

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

Decimal	BCD	Gray
0	0 0 0 0	0 0 0 0
1	0 0 0 1	0 0 0 1
2	0 0 1 0	0 0 1 1
3	0 0 1 1	0 0 1 0
4	0 1 0 0	0 1 1 0
5	0 1 0 1	0 1 1 1
6	0 1 1 0	0 1 0 1
7	0 1 1 1	0 1 0 0
8	1 0 0 0	1 1 0 0
9	1 0 0 1	1 1 0 1

Binary code to Gray Code Conversion:

- Consider the given binary code and place the MSB of binary to the left of MSB.

- Compare the successive two bits starting from MSB. If the 2 bits are same, then the output is zero. Otherwise, output is one.
- Repeat the above step till the LSB of Gray code is obtained.

Example-

From the table, we know that the Gray code corresponding to binary code 1000 is 1100. Now, let us verify it by using the above procedure. Given, binary code is 1000. Step 1 − By placing same MSB to the left of MSB, the binary code will be 1000.

Step 2 − By comparing successive two bits of new binary code, we will get the gray code as 1100.

Application of Gray code:

- Gray code is popularly used in the shaft position encoders.
- A shaft position encoder produces a code word which represents the angular position of the shaft.

Alphanumeric codes:

- A binary digit or bit can represent only two symbols as it has only two states '0' or '1'.
- But this is not enough for communication between two computers because there we need many more symbols for communication.
- These symbols are required to represent 26 alphabets with capital and small letters, numbers from 0 to 9, punctuation marks and other symbols.
- The alphanumeric codes are the codes that represent numbers and alphabetic characters.
- Mostly such codes also represent other characters such as symbol and various instructions necessary for conveying information.
- An alphanumeric code should at least represent 10 digits and 26 letters of alphabet i.e. total 36 items.
- The following two alphanumeric codes are very commonly used for the data representation.

  i. American Standard Code for Information Interchange (ASCII).
  ii. Extended Binary Coded Decimal Interchange Code (EBCDIC).

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

Error detection codes:

- Error detection codes are used to detect the errors present in the received data bitstream.
- These codes contain some bits, which are included appended to the original bit stream.
- These codes detect the error, if it is occurred during transmission of the original data bitstream.
- Example − Parity code, Hamming code.

Error correction codes:

- Error correction codes are used to correct the errors present in the received data bitstream so that, we will get the original data.
- Error correction codes also use the similar strategy of error detection codes.
- Example − Hamming code.

Therefore, to detect and correct the errors, additional bits are appended to the data bits at the time of transmission. **Logic gates:**

- Logic gates play an important role in circuit design and digital systems.
- It is a building block of a digital system and an electronic circuit that always have only one output.

- These gates can have one input or more than one input, but most of the gates have two inputs.

We can classify these Logic gates into the following three categories.

1. Basic gates
2. Universal gates
3. Special gates **Basic gates:**

The basic gates are AND, OR & NOT gates.

AND gate:

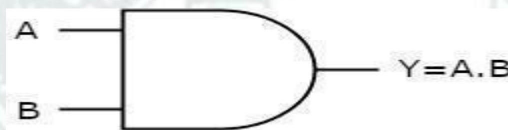
An AND gate is a digital circuit that has two or more inputs and produces an output, which is the **logical AND** of all those inputs. It is optional to represent the **Logical AND** with the symbol '.'.

The following table shows the **truth table** of 2-input AND gate.

A	B	Y = A.B
0	0	0
0	1	0
1	0	0
1	1	1

Here A, B are the inputs and Y is the output of two input AND gate. If both inputs are '1', then only the output, Y is '1'. For remaining combinations of inputs, the output, Y is '0'.

The following figure shows the **symbol** of an AND gate, which is having two inputs A, B and one output, Y.



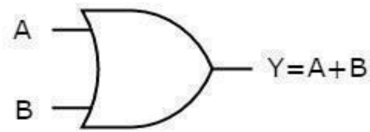
OR gate:

An OR gate is a digital circuit that has two or more inputs and produces an output, which is the logical OR of all those inputs. This **logical OR** is represented with the symbol '+'. The following table shows the **truth table** of 2-input OR gate.

A	B	Y = A+ B
0	0	0
0	1	1
1	0	1
1	1	1

Here A, B are the inputs and Y is the output of two input OR gate. If both inputs are '0', then only the output, Y is '0'. For remaining combinations of inputs, the output, Y is '1'.

The following figure shows the **symbol** of an OR gate, which is having two inputs A, B and one output, Y.



This OR gate produces an output Y, which is the **logical OR** of two inputs A, B.

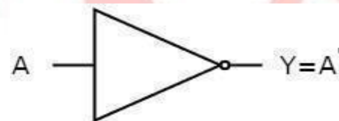
NOT gate:

A NOT gate is a digital circuit that has single input and single output. The output of NOT gate is the **logical inversion** of input. Hence, the NOT gate is also called as inverter. The following table shows the **truth table** of NOT gate.

A	Y = A'
0	1
1	0

Here A and Y are the input and output of NOT gate respectively. If the input, A is '0', then the output, Y is '1'. Similarly, if the input, A is '1', then the output, Y is '0'.

The following figure shows the **symbol** of NOT gate, which is having one input, A and one output, Y.



This NOT gate produces an output Y, which is the **complement** of input, A.

Universal gates

NAND & NOR gates are called as **universal gates**.

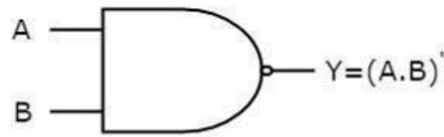
NAND gate

NAND gate is a digital circuit that has two or more inputs and produces an output, which is the **inversion of logical AND** of all those inputs.

The following table shows the **truth table** of 2-input NAND gate.

A	B	Y = (A.B)'
0	0	1
0	1	1
1	0	1
1	1	0

The following image shows the **symbol** of NAND gate, which is having two inputs A, B and one output, Y.



NAND gate operation is same as that of AND gate followed by an inverter. That's why the NAND gate symbol is represented like that.

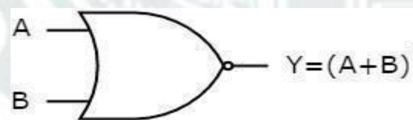
NOR gate:

NOR gate is a digital circuit that has two or more inputs and produces an output, which is the **inversion of logical OR** of all those inputs.

The following table shows the **truth table** of 2-input NOR gate

A	**B**	**Y = (**A+B)'
0	0	1
0	1	0
1	0	0
1	1	0

The following figure shows the **symbol** of NOR gate, which is having two inputs A, B and one output, Y.



NOR gate operation is same as that of OR gate followed by an inverter. That's why the NOR gate symbol is represented like that.

Special Gates

Ex-OR & Ex-NOR gates are called as special gates. Because, these two gates are special cases of OR & NOR gates.

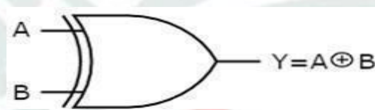
Ex-OR gate:

The full form of Ex-OR gate is **Exclusive-OR** gate. Its function is same as that of OR gate except for some cases, when the inputs having even number of ones. The following table shows the **truth table** of 2-input Ex-OR gate.

A	**B**	**Y = A⊕B**

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

The output of Ex-OR gate is ‘1’, when only one of the two inputs is ‘1’. And it is zero, when both inputs are same.
Below figure shows the **symbol** of Ex-OR gate, which is having two inputs A, B and one output, Y.



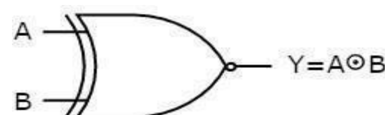
The output of Ex-OR gate is ‘1’, when odd number of ones present at the inputs. Hence, the output of Ex-OR gate is also called as an **odd function**.

Ex-NOR gate:

The full form of Ex-NOR gate is **Exclusive-NOR** gate. Its function is same as that of NOR gate except for some cases, when the inputs having even number of ones.
The following table shows the **truth table** of 2-input Ex-NOR gate.

A	B	Y = A⊙B
0	0	1
0	1	0
1	0	0
1	1	1

The output of Ex-NOR gate is ‘1’, when both inputs are same. And it is zero, when both the inputs are different.
The following figure shows the **symbol** of Ex-NOR gate, which is having two inputs A, B and one output, Y.

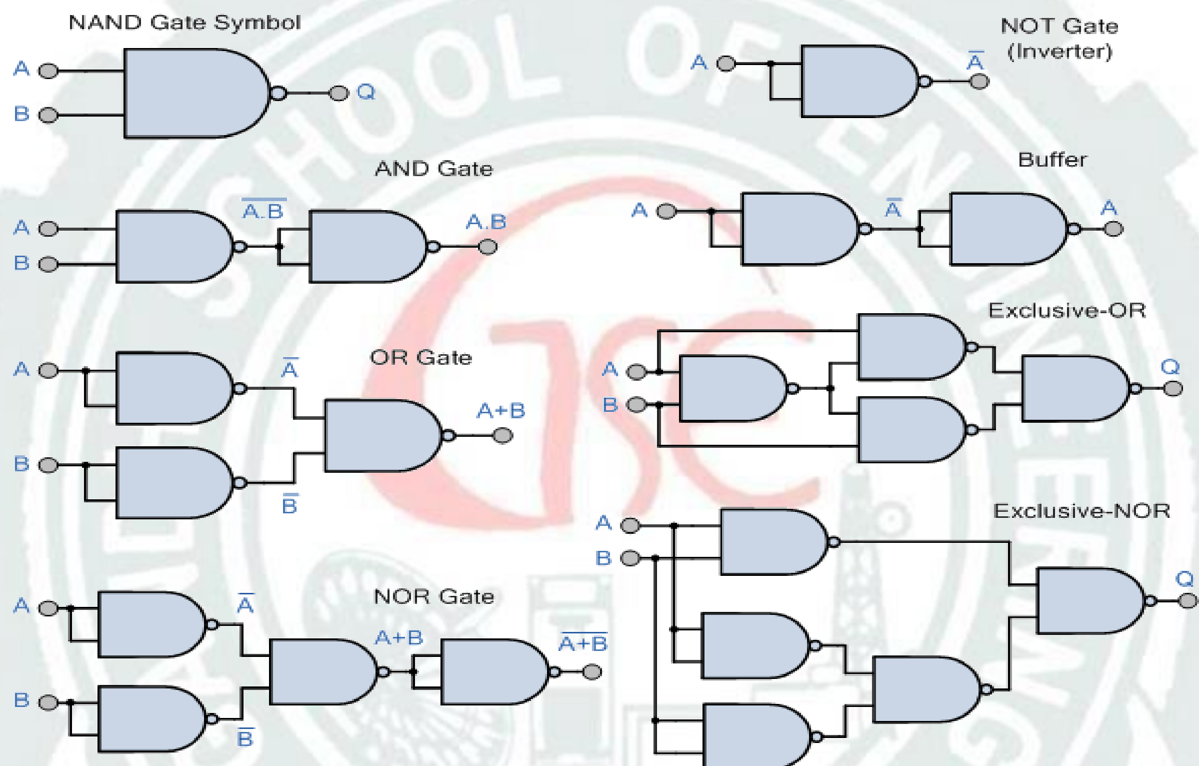


The output of Ex-NOR gate is ‘1’, when even number of ones present at the inputs. Hence, the output of Ex-NOR gate is also called as an **even function**.

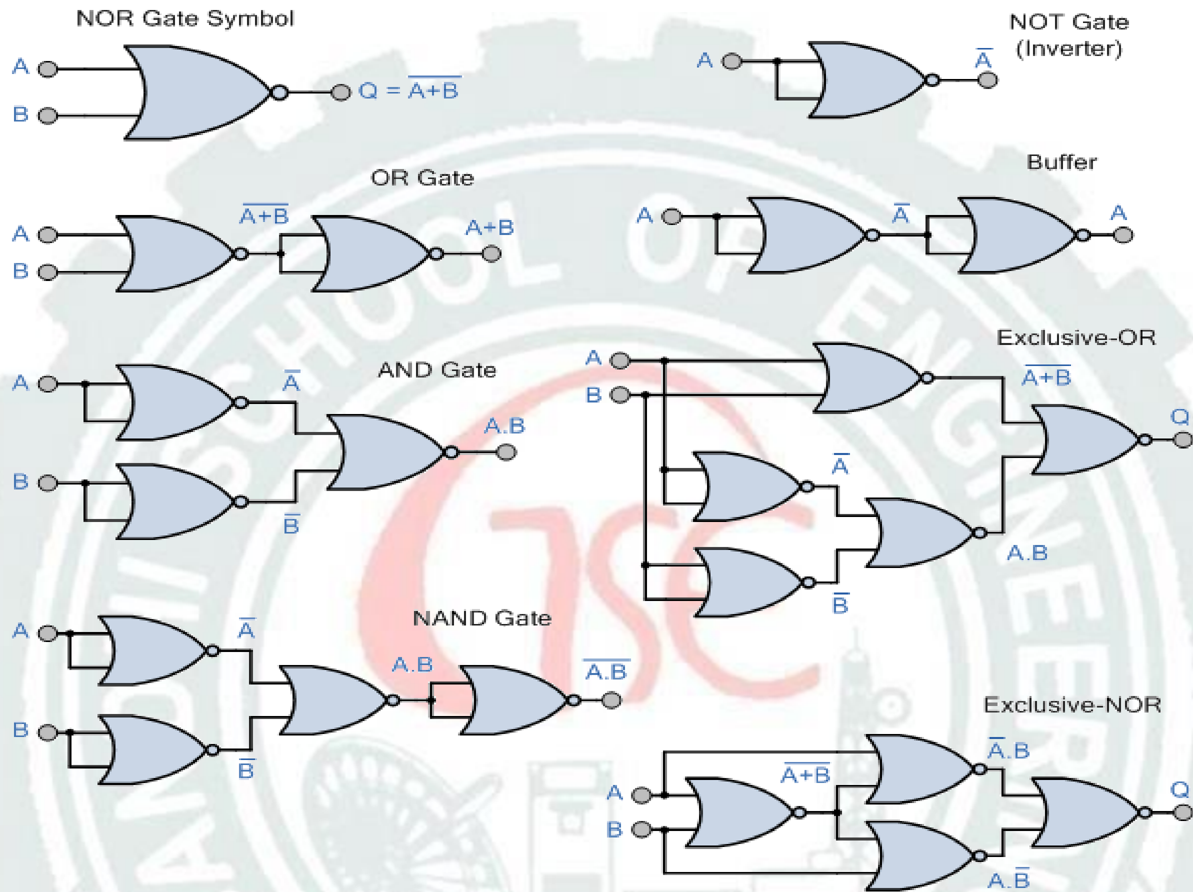
From the above truth tables of Ex-OR & Ex-NOR logic gates, we can easily notice that the Ex-NOR operation is just the logical inversion of Ex-OR operation.

Universal gates and its realization:

We can realise all of the other gates by using just one single type of universal logic gate, the NAND (NOT AND) or the NOR (NOT OR) gate, thereby reducing the number of different types of logic gates required, and also the cost. Thus, the NAND and the NOR gates are commonly referred to as Universal Logic Gates. **Implementation of logic gates using NAND gate only:**



Implementation of logic gates using NOR gate only:



Boolean Algebra:

Boolean Algebra is used to analyse and simplify the digital (logic) circuits. It uses only the binary numbers i.e., 0 and 1. It is also called as Binary Algebra or logical Algebra. Boolean algebra was invented by George Boole in 1854.

Boolean Laws

There are six types of Boolean Laws.

Commutative law

Any binary operation which satisfies the following expression is referred to as commutative operation.

(i) $A.B = B.A$ (ii) $A + B = B + A$

Commutative law states that changing the sequence of the variables does not have any effect on the output of a logic circuit.

Associative law

This law states that the order in which the logic operations are performed is irrelevant as their effect is the same.

(i) $(A.B).C = A.(B.C)$ (ii) $(A + B) + C = A + (B + C)$

Distributive law

Distributive law states the following condition.

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

AND law

These laws use the AND operation. Therefore, they are called as **AND** laws.

(i) $A.0 = 0$ (ii) $A.1 = A$

(iii) $A.A = A$ (iv) $A.\overline{A} = 0$

OR law

These laws use the OR operation. Therefore, they are called as OR laws.

(i) $A + 0 = A$ (ii) $A + 1 = 1$

(iii) $A + A = A$ (iv) $A + \overline{A} = 1$

INVERSION law

This law uses the NOT operation. The inversion law states that double inversion of a variable results in the original variable itself.

$\overline{\overline{A}} = A$

Boolean Function:

Boolean algebra deals with binary variables and logic operation. A **Boolean Function** is described by an algebraic expression called **Boolean expression** which consists of binary variables, the constants 0 and 1, and the logic operation symbols. Consider the following example.

F (A, B, C, D)	=	$A + \overline{BC} + ADC$	Equation No. 1
Boolean Function		Boolean Expression	

Here the left side of the equation represents the output Y. So we can state equation no. 1

$Y = A + \overline{BC} + ADC$

Truth Table Formation

A truth table represents a table having all combinations of inputs and their corresponding result.

It is possible to convert the switching equation into a truth table. For example, consider the following switching equation.

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

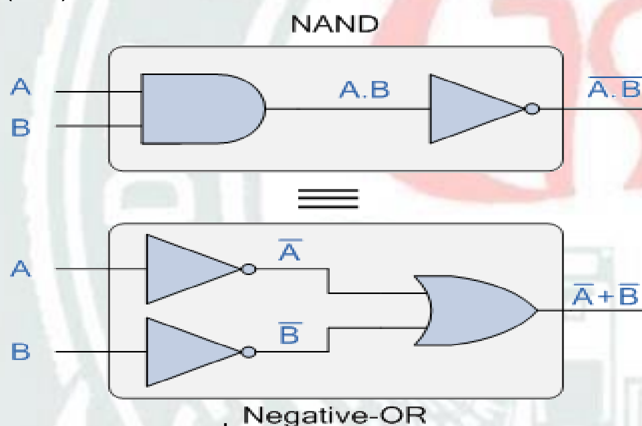
The output will be high (1) if A = 1 or BC = 1 or both are 1. The truth table for this equation is shown by Table (a). The number of rows in the truth table is 2^n where n is the number of input variables (n=3 for the given equation). Hence there are $2^3 = 8$ possible input combination of inputs.

Inputs			Output
A	B	C	F
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	1

De Morgan's Theorem:

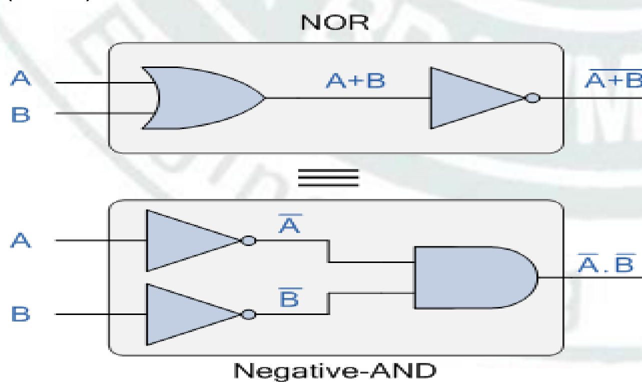
De Morgan’s 1st theorem states that the complement of the product of all the terms is equal to the sum of the complement of each term.

(A.B)’ = A’ + B’



De Morgan’s 2nd theorem states that the complement of the sum of all the terms is equal to the product of the complement of each term.

(A + B)’ = A’. B’



Duality Theorem:

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

Group1	Group2
x + 0 = x	x.1 = x
x + 1 = 1	x.0 = 0
x + x = x	x.x = x
x + x’ = 1	x.x’ = 0
x + y = y + x	x.y = y.x

Example-1:
Given Boolean function, f = p’qr + pq’r + pqr’ + pqr.

Step 1 − Use the **Boolean postulate**, x + x = x. That means, the Logical OR operation with any Boolean variable ‘n’ times will be equal to the same variable. So, we can write the last term pqr two more times.

⇒ f = p’qr + pq’r + pqr’ + pqr + pqr + pqr

Step 2 − Use **Distributive law** for 1st and 4th terms, 2nd and 5th terms, 3rd and 6th terms.

⇒ f = qrp+pp+p + prq+qq+q + pqr+rr+r

Step 3 − Use **Boolean postulate**, x + x’ = 1 for simplifying the terms present in eachparenthesis.

⇒ f = qr11 + pr11 + pq11

Step 4 − Use **Boolean postulate**, x.1 = x for simplifying the above three terms.

⇒ f = qr + pr + pq f =

pq + qr + pr Therefore, the simplified Boolean function is **f =**

pq + qr + pr.

Example-2:
Let us find the complement **of the Boolean function, f = p’q + pq’.** The complement of Boolean function is f’ = pq+pqpq+pq’.

Step 1 − Use DeMorgan’s theorem, x+yx+y’ = x’.y’. ⇒ f’
= pqpq’.pqpq’

Step 2 − Use DeMorgan’s theorem, x.yx.y’ = x’ + y’

⇒ f' = {pp' + q'}.{p' + qq'}

Step3 - Use the Boolean postulate, xx'=x. ⇒ f' = {p + q'}.{p' + q} ⇒ f' = pp' + pq + p'q' + qq'

Step 4 − Use the Boolean postulate, xx'=0.

⇒ f = 0 + pq + p'q' + 0 ⇒

f = pq + p'q'

Therefore, the **complement** of Boolean function, p'q + pq' is **pq + p'q'**.

SOP and POS form:

Sum of Product (SOP):

- The Sum of Product expression is equivalent to the logical AND function which Sums two or more Products to produce an output.
- We will get four Boolean product terms by combining two variables x and y with logical AND operation.
- These Boolean product terms are called as **min terms** or **standard product terms**.
- If the binary variable is '0', then it is represented as complement of variable and '1' as normal form in min term. The min terms are x'y', x'y, xy' and xy.

Product of Sum (POS):

- The Product of Sum expression is equivalent to the logical OR-AND function which gives the AND Product of two or more OR Sums to produce an output.
- We will get four Boolean sum terms by combining two variables x and y with logical OR operation.
- These Boolean sum terms are called as **Max terms** or **standard sum terms**. If the binary variable is '1', then it is represented as complement of variable and '0' as normal form in Max term. The Max terms are x + y, x + y', x' + y and x' + y'.

x	y	Min terms	Max terms
0	0	m_0=x'y'	M_0=x + y
0	1	m_1=x'y	M_1=x + y'
1	0	m_2=xy'	M_2=x' + y
1	1	m_3=xy	M_3=x' + y'

Canonical SOP and POS forms:

- A truth table consists of a set of inputs and outputs.
- If there are 'n' input variables, then there will be 2^n possible combinations with zeros and ones.

- So, the value of each output variable depends on the combination of input variables.
   - So, each output variable will have ‘1’ for some combination of input variables and ‘0’ for some other combination of input variables.

Therefore, we can express each output variable in following two ways.

- Canonical SOP form
- Canonical POS form **Canonical SOP form:**
- Canonical SOP form means Canonical Sum of Products form.
- In this form, each product term contains all literals.
- So, these product terms are nothing but the min terms. Hence, canonical SOP form is also called as **sum of min terms** form.
- First, identify the min terms for which, the output variable is one and then do the logical OR of those min terms in order to get the Boolean expression function corresponding to that output variable. This Boolean function will be in the form of sum of min terms.
- Follow the same procedure for other output variables also, if there is more than one output variable. **Example**

Consider the following **truth table**.

Inputs		Ou put	
p	**Q**	**r**	**f**
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

- Here, the output f is ‘1’ for four combinations of inputs.
- The corresponding min terms are p’qr, pq’r, pqr’, pqr.
- By doing logical OR of these four min terms, we will get the Boolean function of output f.

Therefore, the Boolean function of output is, f
= p’qr + pq’r + pqr’ + pqr.

This is the **canonical SOP form** of output, f. We can also represent this function in following two notations.

$$f = m3+m5+m6+m7 \quad f=\sum m(3,5,6,7)$$

In one equation, we represented the function as sum of respective min terms. In other equation, we used the symbol for summation of those min terms. **Canonical POS form:**

- Canonical POS form means Canonical Product of Sums form.

- In this form, each sum term contains all literals. So, these sum terms are nothing but the Max terms. Hence, canonical POS form is also called as **product of Max terms** form.
- First, identify the Max terms for which, the output variable is zero and then do the logical AND of those Max terms in order to get the Boolean expression function corresponding to that output variable. This Boolean function will be in the form of product of Max terms.
- Follow the same procedure for other output variables also, if there is more than one output variable.

Example

- Consider the same truth table of previous example.
- Here, the output f is '0' for four combinations of inputs.
- The corresponding Max terms are p + q + r, p + q + r', p + q' + r, p' + q + r.
- By doing logical AND of these four Max terms, we will get the Boolean function of output f.

Therefore, the Boolean function of output is, f
= (p+q+r). (p+q+r). (p+q+r). (p+q+r).
This is the **canonical POS form** of output, f.
We can also represent this function in following two notations. f=M0.M1.M2.M4

f=∏M(0,1,2,4)

In one equation, we represented the function as product of respective Max terms. In other equation, we used the symbol for multiplication of those Max terms. The Boolean function,
f = (p+q+r). (p+q+r). (p+q+r). (p+q+r)
is the dual of the Boolean function, f
= p'qr + pq'r + pqr' + pqr.
Therefore, both canonical SOP and canonical POS forms are **Dual** to each other. Functionally, these two forms are same. Based on the requirement, we can use one of these two forms.

Standard SOP and POS forms

We discussed two canonical forms of representing the Boolean outputs. Similarly, there are two standard forms of representing the Boolean outputs. These are the simplified version of canonical forms.

- Standard SOP form
- Standard POS form

The main **advantage** of standard forms is that the number of inputs applied to logic gates can be minimized. Sometimes, there will be reduction in the total number of logic gates required.

Standard SOP form:

Standard SOP form means **Standard Sum of Products** form. In this form, each product term need not contain all literals. So, the product terms may or may not be the min terms. Therefore, the Standard SOP form is the simplified form of canonical SOP form.
We will get Standard SOP form of output variable in two steps.

- Get the canonical SOP form of output variable
- Simplify the above Boolean function, which is in canonical SOP form.

Follow the same procedure for other output variables also, if there is more than one output variable. Sometimes, it may not possible to simplify the canonical SOP form. In that case, both canonical and standard SOP forms are same.

Example

Convert the following Boolean function into Standard SOP form. f

= p'qr + pq'r + pqr' + pqr

The given Boolean function is in canonical SOP form. Now, we have to simplify this Boolean function in order to get standard SOP form.

Step 1 − Use the **Boolean postulate**, x + x = x. That means, the Logical OR operation with any Boolean variable 'n' times will be equal to the same variable. So, we can write the last term pqr two more times.

$\Rightarrow$ f = p'qr + pq'r + pqr' + pqr + pqr + pqr

Step 2 − Use **Distributive law** for 1st and 4th terms, 2nd and 5th terms, 3rd and 6th terms.f$\Rightarrow$

= qr (p+p) + pr (q+q) + pq (r+r)

Step 3 − Use **Boolean postulate**, x + x' = 1 for simplifying the terms present in each parenthesis. f$\Rightarrow$= qr 1 + pr 1 + pq 1

Step 4 − Use **Boolean postulate**, x.1 = x for simplifying above three terms.

$\Rightarrow$ f = qr + pr + pq$\Rightarrow$

f = pq + qr + pr

This is the simplified Boolean function. Therefore, the **standardSOP form** corresponding to given canonical SOP form is **f = pq + qr + prStandard POS form:**

Standard POS form means **Standard Product of Sums** form. In this form, each sum term need not contain all literals. So, the sum terms may or may not be the Max terms. Therefore, the Standard POS form is the simplified form of canonical POS form.

We will get Standard POS form of output variable in two steps.

- Get the canonical POS form of output variable
- Simplify the above Boolean function, which is in canonical POS form.

Follow the same procedure for other output variables also, if there is more than one output variable. Sometimes, it may not possible to simplify the canonical POS form. In that case, both canonical and standard POS forms are same.

Example

Convert the following Boolean function into Standard POS form.

f = (p+q+r). (p+q+r). (p+q+r). (p+q+r)

The given Boolean function is in canonical POS form. Now, we have to simplify this Boolean function in order to get standard POS form.

Step 1 − Use the **Boolean postulate**, x.x = x. That means, the Logical AND operation with any Boolean variable 'n' times will be equal to the same variable. So, we can write the first term p+q+r two more times.

$\Rightarrow$ f = (p+q+r). (p+q+r). (p+q+r). (p+q+r). (p+q+r). (p+q+r)

Step 2 − Use **Distributive law**, x + y.z = (x+y). (x+z) for 1st and 4th parenthesis, 2nd and 5th parenthesis, 3rd and 6th parenthesis.

$\Rightarrow$ f = (p+q+rr). (p+r+qq). (q+r+pp)

Step 3 − Use **Boolean postulate**, x.x'=0 for simplifying the terms present in each parenthesis. f

$\Rightarrow$ = (p+q+0). (p+r+0). (q+r+0)

Step 4 − Use **Boolean postulate**, x + 0 = x for simplifying the terms present in each parenthesis f$\Rightarrow$= (p+q). (p+r). (q+r) f =$\Rightarrow$(p+q). (q+r). (p+r)

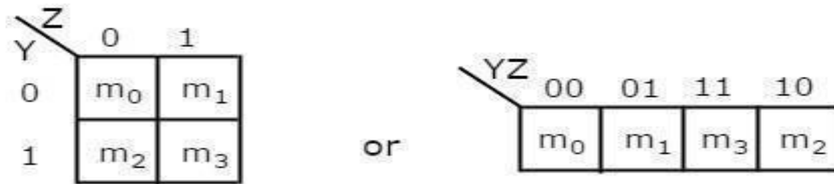
This is the simplified Boolean function. Therefore, the **standard POS form** corresponding to given canonical POS form is **f = (**p+q)**. (**q+r)**. (**p+r). This is the **dual** of the Boolean function, f = pq + qr + pr.

Therefore, both Standard SOP and Standard POS forms are Dual to each other. **Karnaugh map:**

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

2- Variable K-Map:

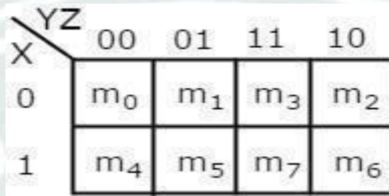
The number of cells in 2 variable K-map is four, since the number of variables is two.



- There is only one possibility of grouping 4 adjacent min terms.
- The possible combinations of grouping 2 adjacent min terms are {(m_0, m_1), (m_2, m_3), (m_0, m_2) and (m_1, m_3)}. **3-Variable**

K-Map:

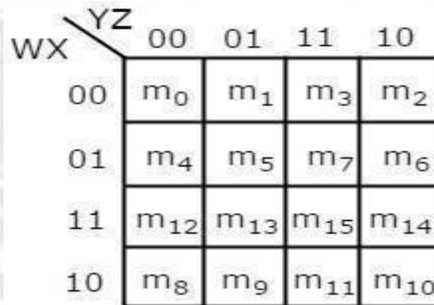
The number of cells in 3 variable K-map is eight, since the number of variables is three.

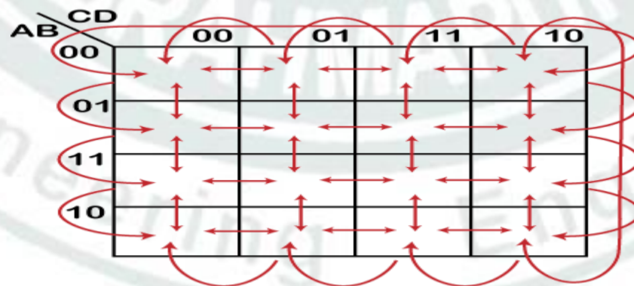


- There is only one possibility of grouping 8 adjacent min terms.
- The possible combinations of grouping 4 adjacent min terms are {(m_0, m_1, m_3, m_2), (m_4, m_5, m_7, m_6), (m_0, m_1, m_4, m_5), (m_1, m_3, m_5, m_7), (m_3, m_2, m_7, m_6) and (m_2, m_0, m_6, m_4)}.
- The possible combinations of grouping 2 adjacent min terms are {(m_0, m_1), (m_1, m_3), (m_3, m_2), (m_2, m_0), (m_4, m_5), (m_5, m_7), (m_7, m_6), (m_6, m_4), (m_0, m_4), (m_1, m_5), (m_3, m_7) and (m_2, m_6)}.
- If x=0, then 3 variable K-map becomes 2 variable K-map.

4- Variable K-Map:

The number of cells in 4 variable K-map is sixteen, since the number of variables is four.





- There is only one possibility of grouping 16 adjacent min terms.
- Let R_1, R_2, R_3 and R_4 represents the min terms of first row, second row, third row and fourth row respectively. Similarly, C_1, C_2, C_3 and C_4 represents the min terms of first column, second column, third column and fourth column respectively. The possible

combinations of grouping 8 adjacent min terms are {(R_1, R_2), (R_2, R_3), (R_3, R_4), (R_4, R_1), (C_1, C_2), (C_2, C_3), (C_3, C_4), (C_4, C_1)}.

- If w=0, then 4 variable K-map becomes 3 variable K-map.

Example- **f(**W,X,Y,Z)**=** **∑m(2,6,8,9,10,11,14,15)**
using K-map.
The given Boolean function is in sum of products form. It is having 4 variables W, X, Y & Z. So, we require **4 variable K-map**.

WX \ YZ	00	01	11	10
00				1
01				1
11			1	1
10	1	1	1	1

The **4 variable K-map** with three **groupings** is

WX \ YZ	00	01	11	10	
00				1	YZ'
01				1	
11			1	1	WY
10 (WX')	1	1	1	1	

Therefore, the **simplified Boolean function** is **f = WX' + WY + YZ'**

Example- f(X,Y,Z)=∏M(0,1,2,4)
using K-map.
The given Boolean function is in product of Max terms form. It is having 3 variables X, Y & Z. So, we require 3 variable K-map.

X \ YZ	00	01	11	10
0	0	0		0
1	0			

The **3 variable K-map** with three **groupings** is

X \ YZ	00	01	11	10
0	0	0		0
1	0			

X+Y, Z+X, Y+Z

Therefore, the **simplified Boolean function** is

f = (X+Y)**. (**Y+Z)**. (**Z+X)

Don't care condition:

- The "Don't care" condition says that we can use the blank cells of a K-map to make a group of the variables.

- To make a group of cells, we can use the "don't care" cells as either 0 or 1, and if required, we can also ignore that cell.
- We mainly use the "don't care" cell to make a large group of cells.
- The cross(X) symbol is used to represent the "don't care" cell in K-map.
- This cross symbol represents an invalid combination.
- The "don't care" in excess-3 code are 0000, 0001, 0010, 1101, 1110, and 1111 because they are invalid combinations.
- Apart from this, the 4-bit BCD to Excess-3 code, the "don't care" are 1010, 1011, 1100, 1101, 1110, and 1111.

Example 1: Minimize f = ∑m(1,5,6,12,13,14) + d(4) in SOP minimal form

AB \ CD	00	01	11	10
00		1		
01	X	1		1
11	1	1		1
10				

So, the minimized SOP form of the function is:
f = BC' + BD' + A'C'D **Example-2:**
Minimize the following function in SOP minimal form using K-Maps: F(A, B, C, D) = ∑m(1, 2, 6, 7, 8, 13, 14, 15) + d(3, 5, 12)

AB \ CD	00	01	11	10
00		1	X	1
01		X	1	1
11	X	1	1	1
10	1			

F = AC'D' + A'D + A'C + AB

UNIT-2Combinational logic circuit

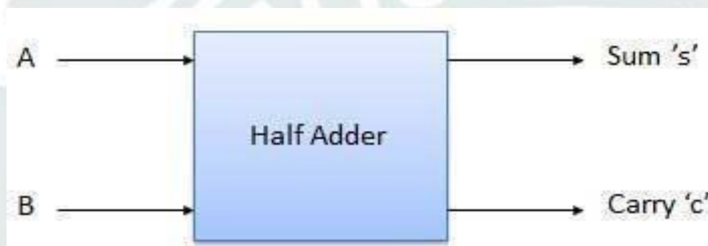
Combinational circuit is a circuit in which we combine the different gates in the circuit, for example encoder, decoder, multiplexer and demultiplexer. Some of the characteristics of combinational circuits are following −

- The output of combinational circuit at any instant of time, depends only on the levels present at input terminals.
- The combinational circuit do not use any memory. The previous state of input does not have any effect on the present state of the circuit.
- A combinational circuit can have an n number of inputs and m number of outputs.

Block Diagram:

Half Adder

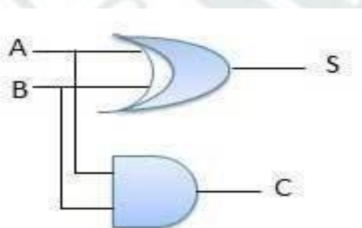
Half adder is a combinational logic circuit with two inputs and two outputs. The half adder circuit is designed to add two single bit binary number A and B. It is the basic building block for addition of two **single** bit numbers. This circuit has two outputs **carry** and **sum**. Block diagram



Truth Table

Inputs		Output	
A	B	S	C
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

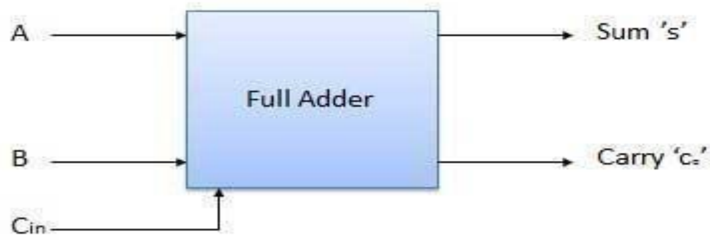
Circuit Diagram



Full Adder

Full adder is developed to overcome the drawback of Half Adder circuit. It can add two onebit numbers A and B, and carry c. The full adder is a three input and two output combinational circuit.

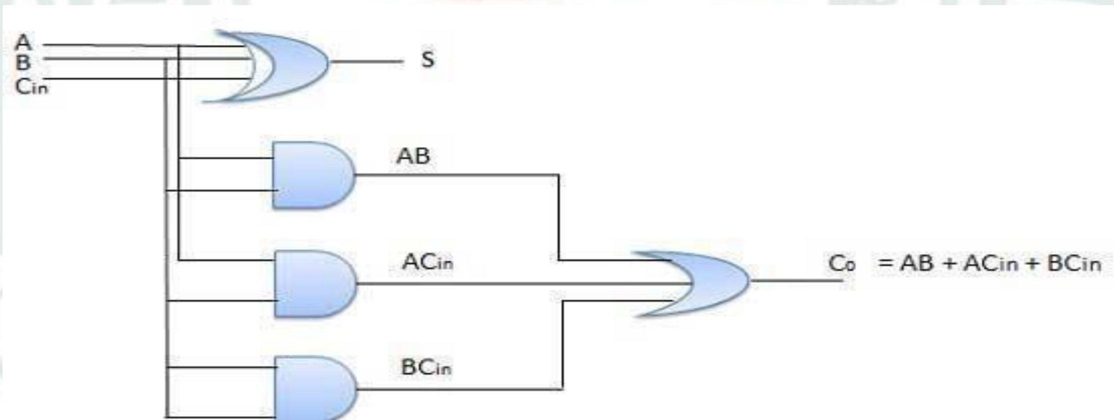
Block diagram



Truth Table

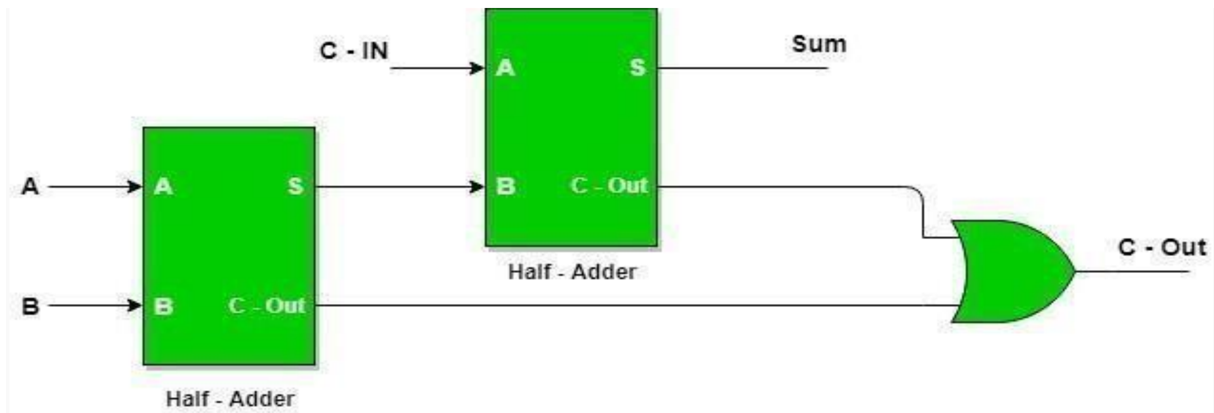
Inputs			Output	
A	B	Cin	S	Co
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

Circuit Diagram



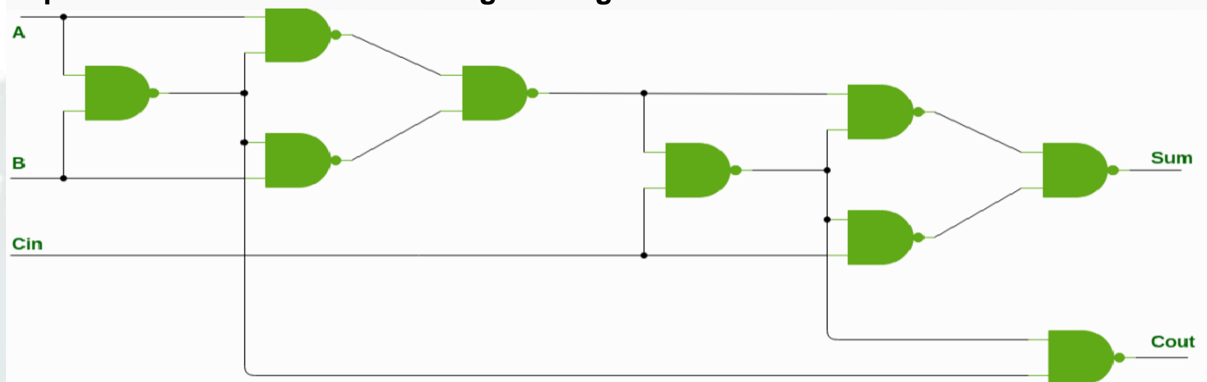
Implementation of Full Adder using Half Adders

2 Half Adders and a 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:

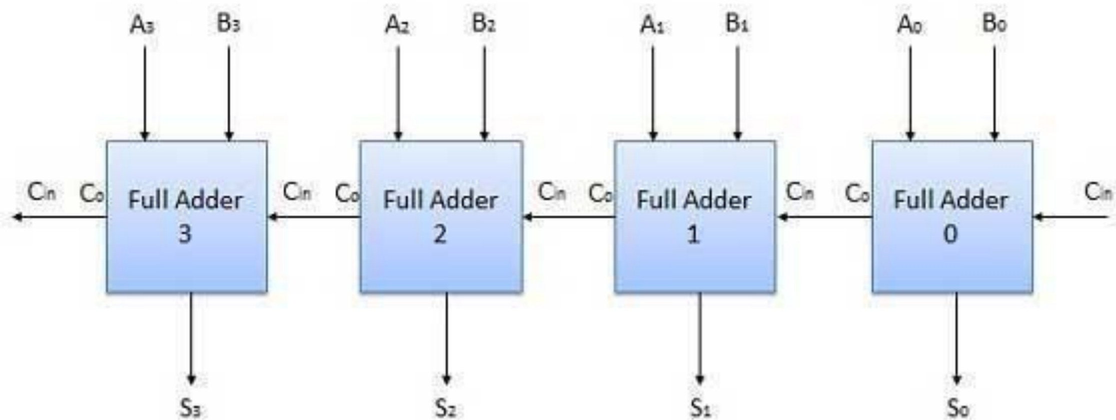


The Full Adder is capable of adding only two single digit binary number along with a carry input. But in practical we need to add binary numbers which are much longer than just one bit. To add two n-bit binary numbers we need to use the n-bit parallel adder. It uses a number of full adders in cascade. The carry output of the previous full adder is connected to carry input of the next full adder.

4 Bit Parallel Adder

In the block diagram, A_0 and B_0 represent the LSB of the four bit words A and B. Hence Full Adder-0 is the lowest stage. Hence its C_{in} has been permanently made 0. The rest of the connections are exactly same as those of n-bit parallel adder is shown in fig. The four-bit parallel adder is a very common logic circuit.

Block diagram



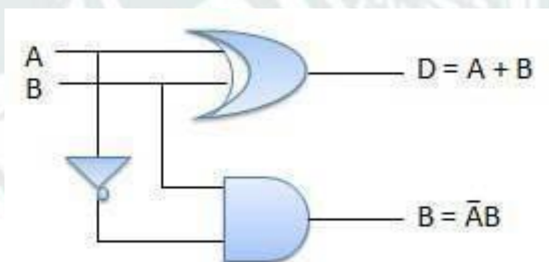
Half Subtractors

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

Truth Table

Inputs		Output	
A	B	(A – B)	Borrow
0	0	0	0
0	1	1	1
1	0	1	0
1	1	0	0

Circuit Diagram



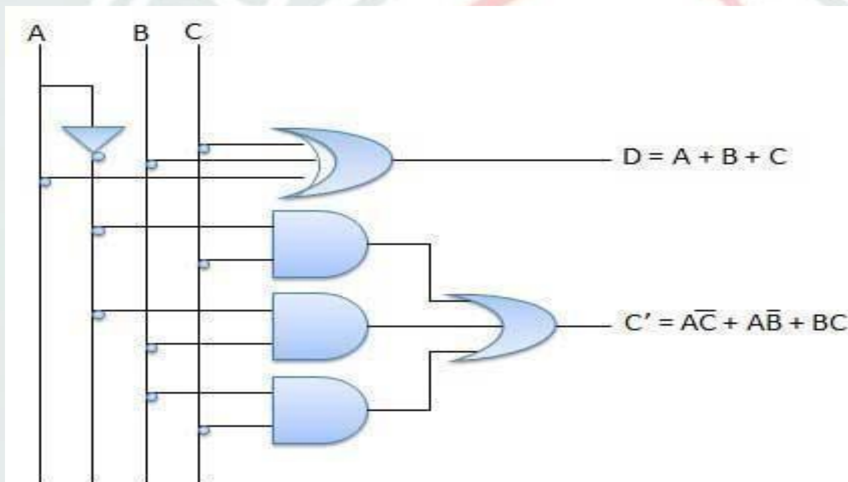
Full Subtractors

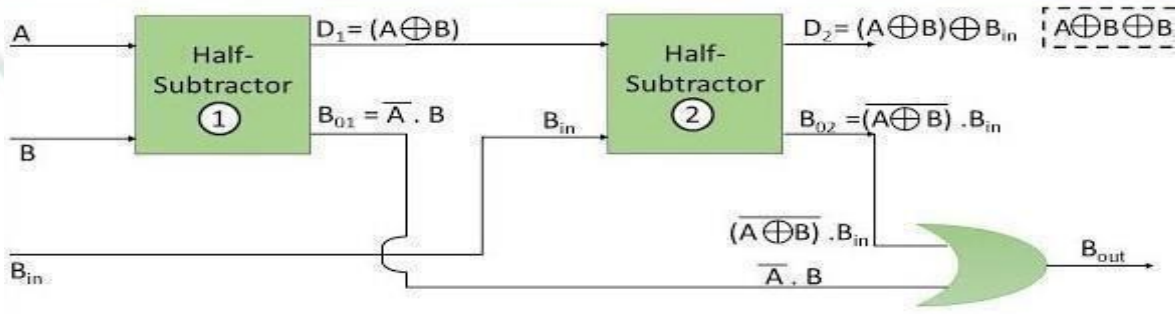
The disadvantage of a half subtractor is overcome by full subtractor. The full subtractor is a combinational circuit with three inputs A,B,C and two output D and C'. A is the 'minuend', B is 'subtrahend', C is the 'borrow' produced by the previous stage, D is the difference output and C' is the borrow output.

Truth Table

Inputs			Output	
A	B	C	(A-B-C)	C'
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

Circuit Diagram



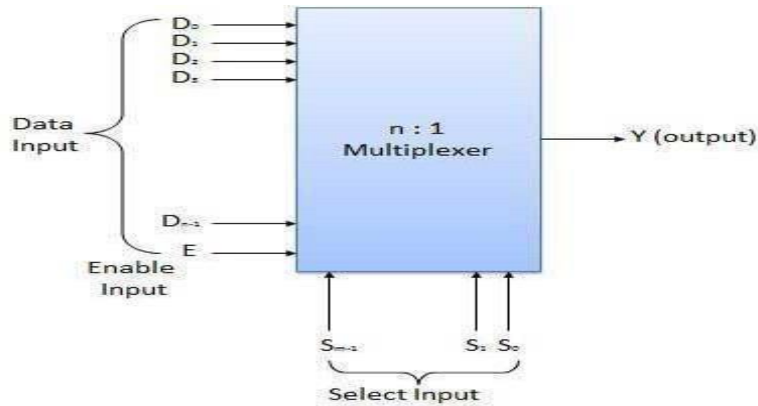


Multiplexers

Multiplexer is a special type of combinational circuit. There are n-data inputs, one output and m select inputs with 2m = n. It is a digital circuit which selects one of the n data inputs and routes it to the output. The selection of one of the n inputs is done by the selected inputs. Depending on the digital code applied at the selected inputs, one out of n data sources is selected and transmitted to the single output Y. E is called the strobe or enable input which is

useful for the cascading. It is generally an active low terminal that means it will perform the required operation when it is low.

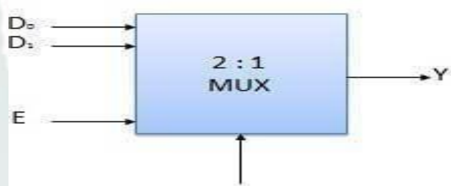
Block diagram



Multiplexers come in multiple variations

- 2 : 1 multiplexer
- 4 : 1 multiplexer
- 16 : 1 multiplexer
- 32 : 1 multiplexer

Block Diagram



Truth Table

Enable	Select	Output
E	S	Y
0	x	0
1	0	D_0
1	1	D_1

x = Don't care

Demultiplexers

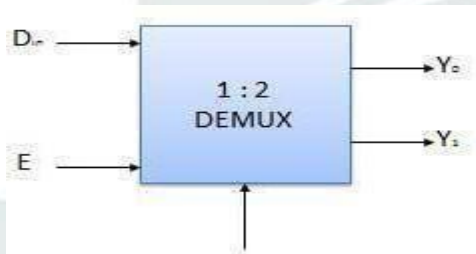
A demultiplexer performs the reverse operation of a multiplexer i.e. it receives one input and distributes it over several outputs. It has only one input, n outputs, m select input. At a time

only one output line is selected by the select lines and the input is transmitted to the selected output line. A de-multiplexer is equivalent to a single pole multiple way switch as shown in fig.

Demultiplexers comes in multiple variations.

- 1 : 2 demultiplexer
- 1 : 4 demultiplexer
- 1 : 16 demultiplexer
- 1 : 32 demultiplexer

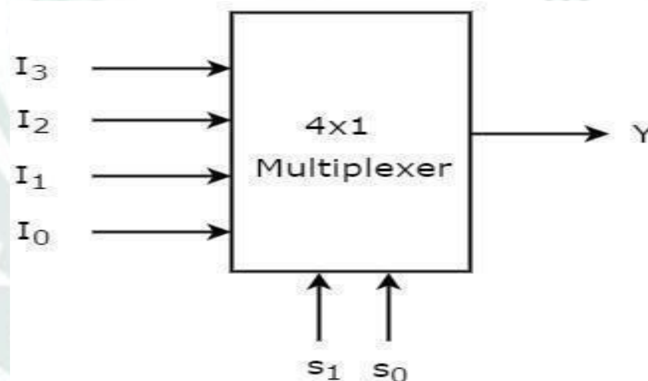
Block diagram



Truth Table

4:1 Multiplexer

4:1 Multiplexer has four data inputs I_3, I_2, I_1& I_0, two selection lines s_1& s_0 and one output Y. The **block diagram** of 4x1 Multiplexer is shown in the following figure.



One of these 4 inputs will be connected to the output based on the combination of inputs present at these two selection lines.

Truth table of 4:1 Multiplexer is shown below.

Selection Lines		Output
S_1	**S_0**	**Y**
0	0	I_0

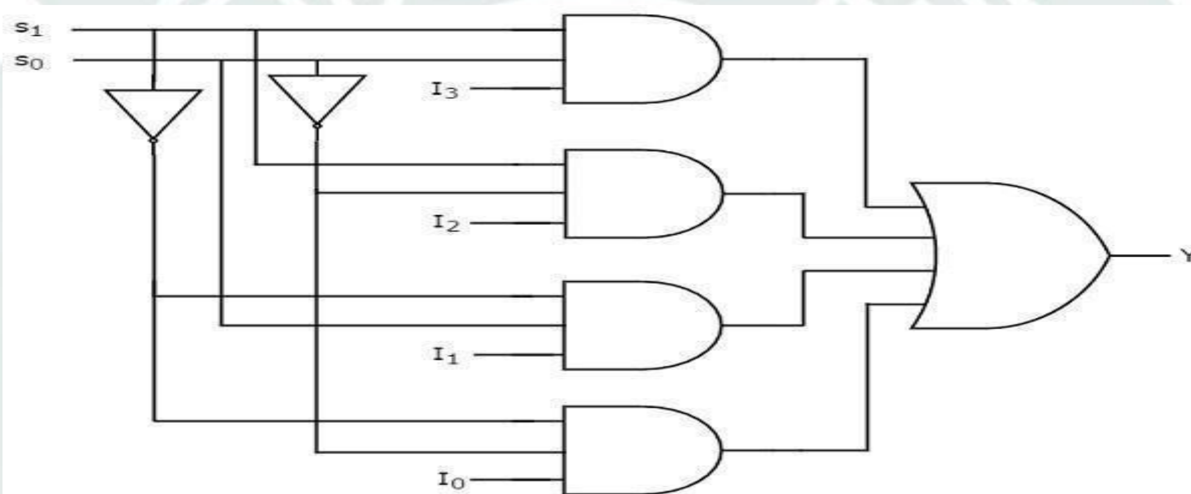
0	1	I_1
1	0	I_2
1	1	I_3

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

$$Y = S1'S0'I_0 + S1'S0I_1 + S1S0'I_2 + S1S0I_3$$

We can implement this Boolean function using Inverters, AND gates & OR gate.

The **circuit diagram** of 4:1 multiplexer is shown in the following figure.



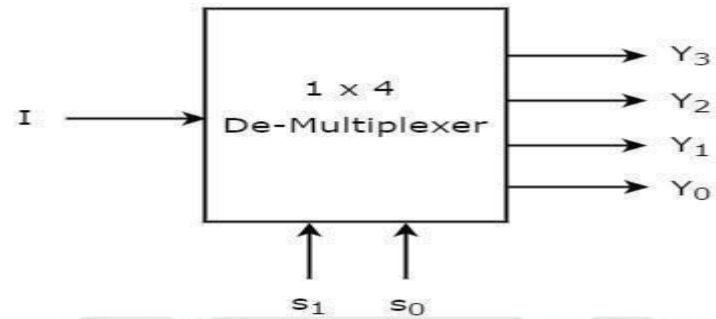
We can easily understand the operation of the above circuit. Similarly, you can implement 8x1 Multiplexer and 16x1 multiplexer by following the same procedure.

Enable	Select	Output	
E	S	Y0	Y1
0	x	0	0
1	0	0	D_{in}
1	1	D_{in}	0

x = Don't care

1:4 De-Multiplexer

1:4 De-Multiplexer has one input I, two selection lines, s_1& s_0 and four outputs Y_3, Y_2, Y_1&Y_0. The **block diagram** of 1:4 De-Multiplexer is shown in the following figure.



The single input 'I' will be connected to one of the four outputs, Y_3 to Y_0 based on the values of selection lines s_1& s0. The **Truth table** of 1x4 De-Multiplexer is shown below.

Selection Inputs		**Outputs**			
S_1	S_0	Y_3	Y_2	Y_1	Y_0
0	0	0	0	0	I
0	1	0	0	I	0
1	0	0	I	0	0
1	1	I	0	0	0

From the above Truth table, we can directly write the **Boolean functions** for each output as

$$Y3 = s_1 s_0 I$$

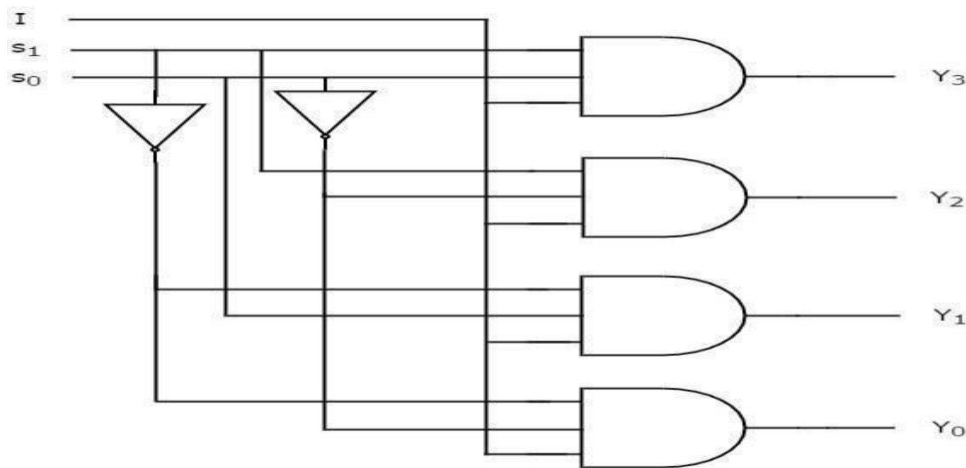
$$Y2 = s1s0'I$$

$$Y1 = s1's0I$$

$$Y0 = s1's0'I$$

We can implement these Boolean functions using Inverters & 3-input AND gates.

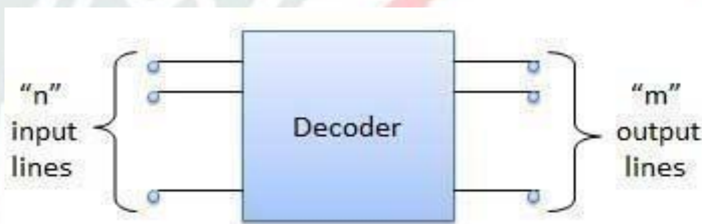
The **circuit diagram** of 1:4 De-Multiplexer is shown in the following figure.



Decoder

A decoder is a combinational circuit. It has n input and to a maximum m = 2n outputs. Decoder is identical to a demultiplexer without any data input. It performs operations which are exactly opposite to those of an encoder.

Block diagram



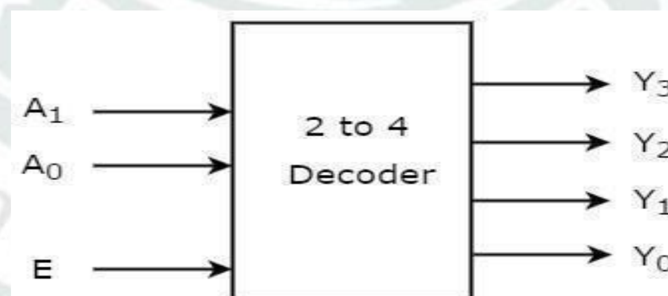
Examples of Decoders are following.

- Code converters
- BCD to seven segment decoders

2 to 4 Decoder

Let 2 to 4 Decoder has two inputs A_1& A_0 and four outputs Y_3, Y_2, Y_1& Y_0.

The **block diagram** of 2 to 4 decoder is shown in the following figure.



One of these four outputs will be '1' for each combination of inputs when enable, E is '1'.

The **Truth table** of 2 to 4 decoder is shown below.

Enable	Inputs	Outputs

E	A_1	A_0	Y_3	Y_2	Y_1	Y_0
0	x	x	0	0	0	0
1	0	0	0	0	0	1
1	0	1	0	0	1	0
1	1	0	0	1	0	0
1	1	1	1	0	0	0

From Truth table, we can write the **Boolean functions** for each output as

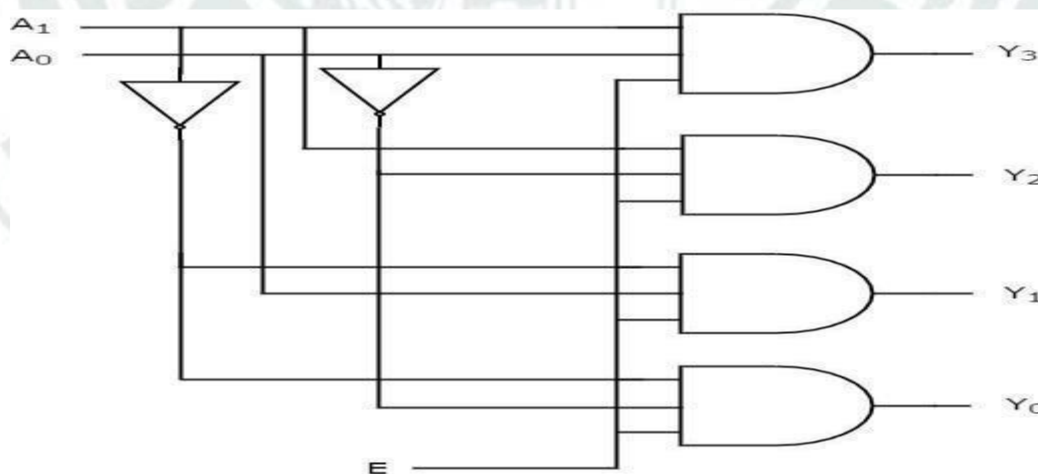
$$Y3 = E.A1.A0$$

$$Y2 = E.A1.A0'$$

$$Y1 = E.A1'.A0$$

$$Y0 = E.A1'.A0'$$

Each output is having one product term. So, there are four product terms in total. We can implement these four product terms by using four AND gates having three inputs each & two inverters. The **circuit diagram** of 2 to 4 decoder is shown in the following figure.

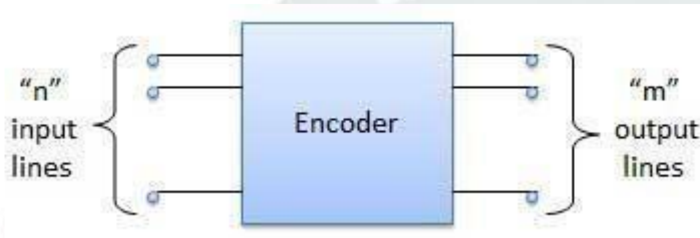


Therefore, the outputs of 2 to 4 decoder are nothing but the **min terms** of two input variables A_1& A_0, when enable, E is equal to one. If enable, E is zero, then all the outputs of decoder will be equal to zero.

Similarly, 3 to 8 decoder produces eight min terms of three input variables A_2, A_1& A_0 and 4 to 16 decoder produces sixteen min terms of four input variables A_3, A_2, A_1& A_0.

Encoder

Encoder is a combinational circuit which is designed to perform the inverse operation of the decoder. An encoder has n number of input lines and m number of output lines. An encoder produces an m bit binary code corresponding to the digital input number. The encoder accepts an n input digital word and converts it into an m bit another digital word. Block diagram



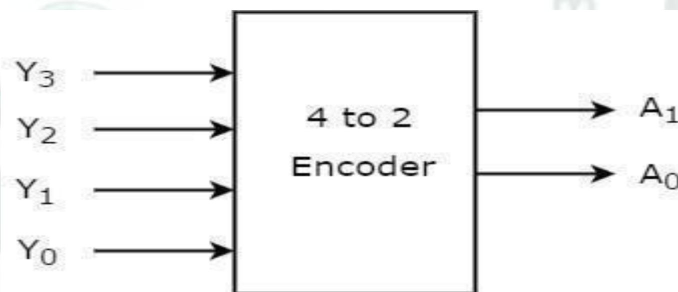
Examples of Encoders are following.

- Priority encoders
- Decimal to BCD encoder
- Octal to binary encoder
- Hexadecimal to binary encoder

4 to 2 Encoder

Let 4 to 2 Encoder has four inputs Y_3, Y_2, Y_1& Y_0 and two outputs A_1& A_0.

The **block diagram** of 4 to 2 Encoder is shown in the following figure.



At any time, only one of these 4 inputs can be '1' in order to get the respective binary code at the output. The **Truth table** of 4 to 2 encoder is shown below.

Inputs				**Outputs**	
Y_3	Y_2	Y_1	Y_0	A_1	A_0
0	0	0	1	0	0

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

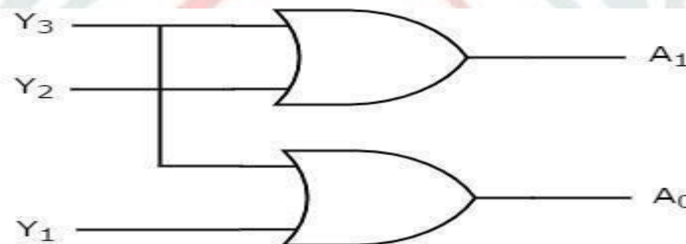
From Truth table, we can write the **Boolean functions** for each output as

$$A1 = Y3 + Y2$$

$$A0 = Y3 + Y1$$

We can implement the above two Boolean functions by using two input OR gates.

The **circuit diagram** of 4 to 2 encoder is shown in the following figure.

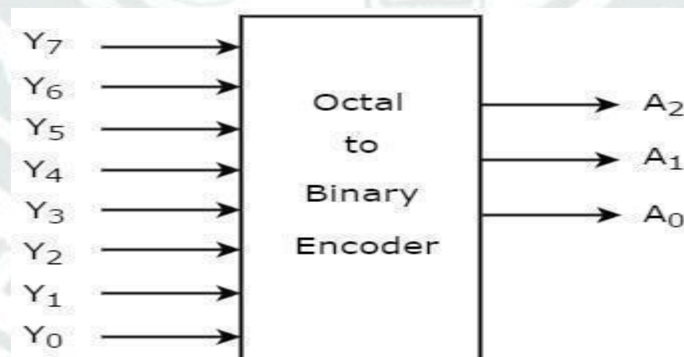


The above circuit diagram contains two OR gates. These OR gates encode the four inputs with two bits

Octal to Binary Encoder

Octal to binary Encoder has eight inputs, Y_7 to Y_0 and three outputs A_2, A_1& A_0. Octal to binary encoder is nothing but 8 to 3 encoder.

The **block diagram** of octal to binary Encoder is shown in the following figure.



At any time, only one of these eight inputs can be '1' in order to get the respective binary code. The **Truth table** of octal to binary encoder is shown below.

Inputs	Outputs

Y_7	Y_6	Y_5	Y_4	Y_3	Y_2	Y_1	Y_0	A_2	A_1	A_0
0	0	0	0	0	0	0	1	0	0	0
0	0	0	0	0	0	1	0	0	0	1
0	0	0	0	0	1	0	0	0	1	0
0	0	0	0	1	0	0	0	0	1	1
0	0	0	1	0	0	0	0	1	0	0
0	0	1	0	0	0	0	0	1	0	1
0	1	0	0	0	0	0	0	1	1	0
1	0	0	0	0	0	0	0	1	1	1

From Truth table, we can write the **Boolean functions** for each output as

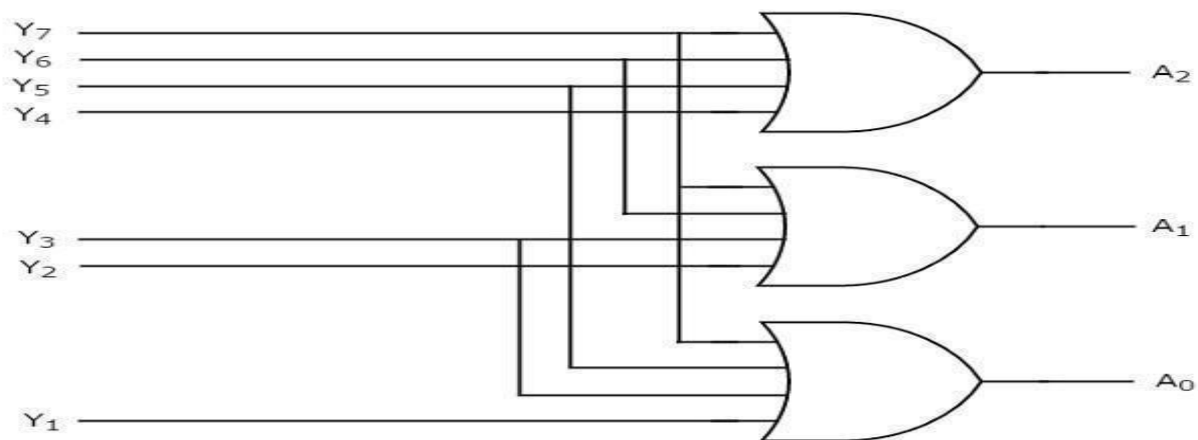
$$A2 = Y7 + Y6 + Y5 + Y4$$

$$A1 = Y7 + Y6 + Y3 + Y2$$

$$A0 = Y7 + Y5 + Y3 + Y1$$

We can implement the above Boolean functions by using four input OR gates.

The **circuit diagram** of octal to binary encoder is shown in the following figure.

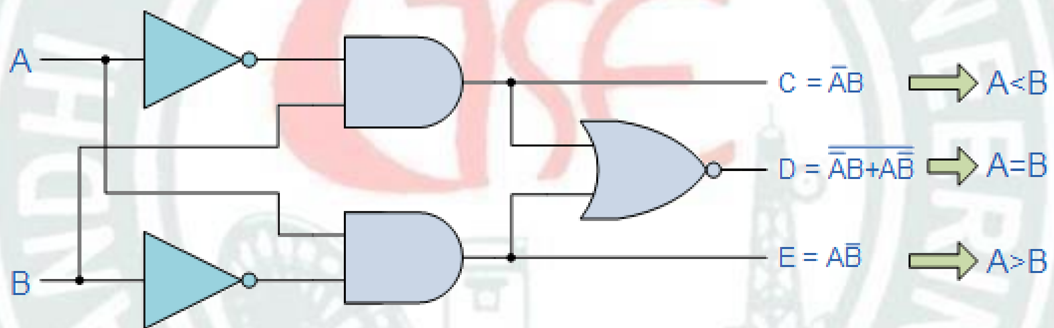


The above circuit diagram contains three 4-input OR gates. These OR gates encode the eight inputs with three bits.

Digital comparator

The Digital Comparator is another very useful combinational logic circuit used to compare the value of two binary digits.

1- bit Digital Comparator Circuit



Then the operation of a 1-bit digital comparator is given in the following Truth Table. **Digital Comparator Truth Table**

Inputs		Outputs		
B	A	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

2-bit Magnitude Comparator

A comparator that compares two binary numbers (each number having 2 bits) and produces three outputs based on the relative magnitudes of given binary bits is called a 2-bit magnitude comparator.

Truth Table

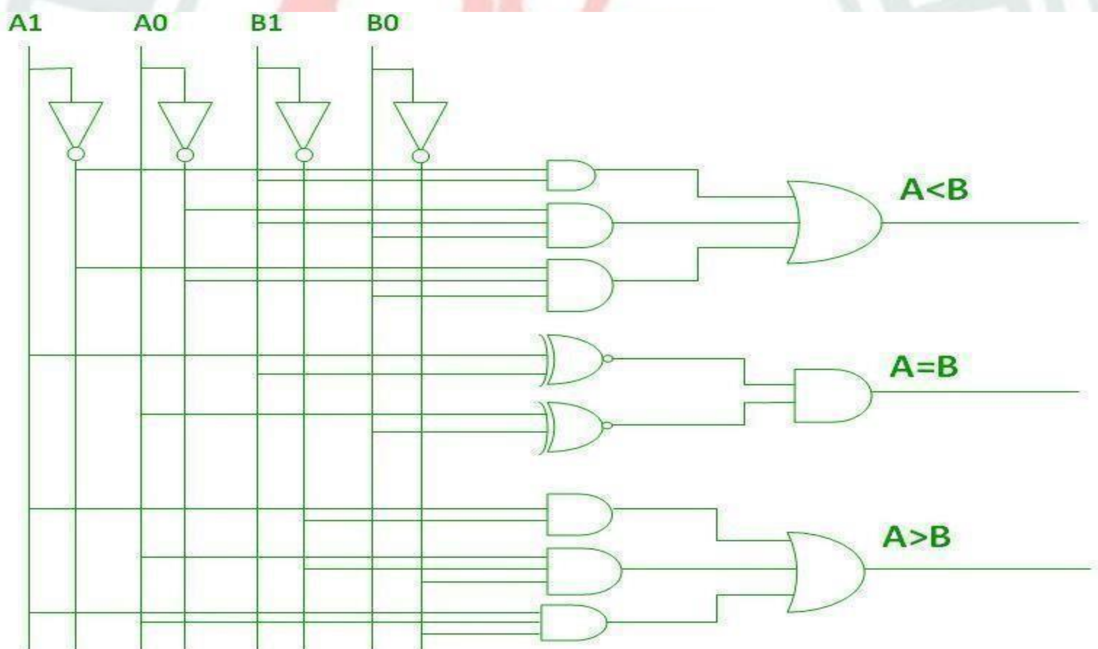
A1	A0	B1	B0	A<B	A=B	A>B
0	0	0	0	0	1	0
0	0	0	1	1	0	0
0	0	1	0	1	0	0
0	0	1	1	1	0	0
0	1	0	0	0	0	1
0	1	0	1	0	1	0
0	1	1	0	1	0	0
0	1	1	1	1	0	0
1	0	0	0	0	0	1
1	0	0	1	0	0	1

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

The truth table derives the expressions of A<B, A>B, and A=B as below

$$\boldsymbol{A < B - A1'B1' + A0'B1B0 + A1'A0'B0}$$
$$\boldsymbol{A > B - A1B1' + A0B1'B0' + A1A0B0'}$$
$$\boldsymbol{A = B - (A0\ Ex - Nor\ B0)\ (A1\ Ex - Nor\ B1)}$$

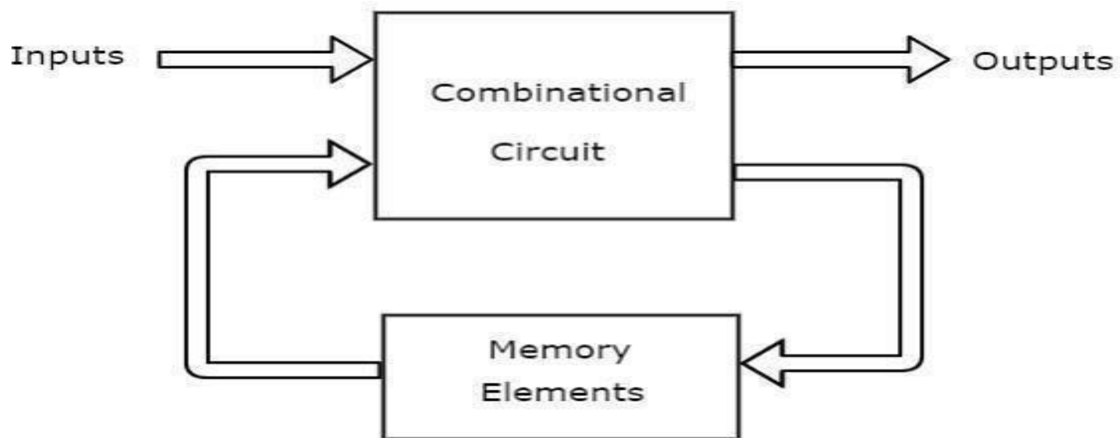
With these expressions, the Circuit diagram can be as follows



UNIT-3

Sequential Logic Circuits

Sequential circuit contains a set of inputs and outputs. The outputs of sequential circuit depend not only on the combination of present inputs but also on the previous outputs. Previous output is nothing but the **present state**. Therefore, sequential circuits contain combinational circuits along with memory storage elements. Some sequential circuits may not contain combinational circuits, but only memory elements.



Following table shows the **differences** between combinational circuits and sequential circuits.

Combinational Circuits	Sequential Circuits
Outputs depend only on present inputs.	Outputs depend on both present inputs and present state.
Feedback path is not present.	Feedback path is present.
Memory elements are not required.	Memory elements are required.
Clock signal is not required.	Clock signal is required.
Easy to design.	Difficult to design.

Types of Sequential Circuits

Following are the two types of sequential circuits −

- Asynchronous sequential circuits
- Synchronous sequential circuits

Asynchronous sequential circuits

If some or all the outputs of a sequential circuit do not change affect with respect to active transition of clock signal, then that sequential circuit is called as **Asynchronous sequential circuit**. That means, all the outputs of asynchronous sequential circuits do not change affect at the same time. Therefore, most of the outputs of asynchronous sequential circuits are **not in synchronous** with either only positive edges or only negative edges of clock signal.

Synchronous sequential circuits

If all the outputs of a sequential circuit change affect with respect to active transition of clock signal, then that sequential circuit is called as **Synchronous sequential circuit**. That means, all the outputs of synchronous sequential circuits change affect at the same time. Therefore, the outputs of synchronous sequential circuits are in synchronous with either only positive edges or only negative edges of clock signal.

Clock Signal and Triggering

Clock signal

Clock signal is a periodic signal and its ON time and OFF time need not be the same. We can represent the clock signal as a **square wave**, when both its ON time and OFF time are same. This clock signal is shown in the following figure.

Types of Triggering

Following are the two possible types of triggering that are used in sequential circuits.

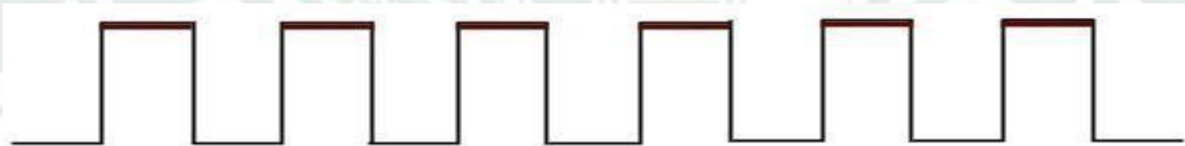
- Level triggering
- Edge triggering

Level triggering

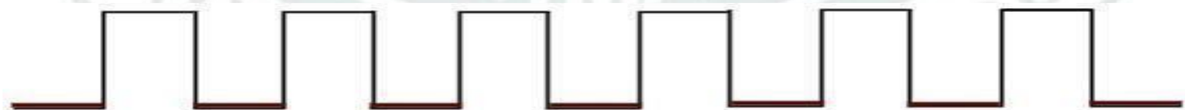
There are two levels, namely logic High and logic Low in clock signal. Following are the two **types of level triggering**.

- Positive level triggering
- Negative level triggering

If the sequential circuit is operated with the clock signal when it is in **Logic High**, then that type of triggering is known as **Positive level triggering**. It is highlighted in below figure.



If the sequential circuit is operated with the clock signal when it is in **Logic Low**, then that type of triggering is known as **Negative level triggering**. It is highlighted in the following figure.



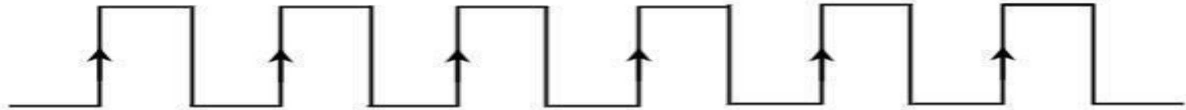
Edge triggering

There are two types of transitions that occur in clock signal. That means, the clock signal transitions either from Logic Low to Logic High or Logic High to Logic Low.

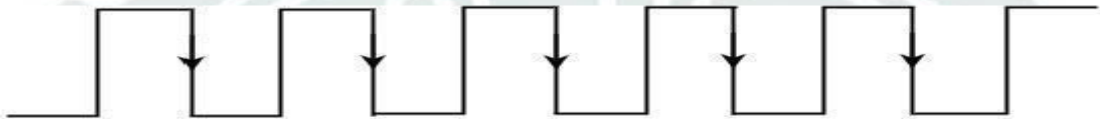
Following are the two **types of edge triggering** based on the transitions of clock signal.

- Positive edge triggering
- Negative edge triggering

If the sequential circuit is operated with the clock signal that is transitioning from Logic Low to Logic High, then that type of triggering is known as **Positive edge triggering**. It is also called as rising edge triggering. It is shown in the following figure.



If the sequential circuit is operated with the clock signal that is transitioning from Logic High to Logic Low, then that type of triggering is known as **Negative edge triggering**. It is also called as falling edge triggering. It is shown in the following figure.



There are two types of memory elements based on the type of triggering that is suitable to operate it.

- Latches
- Flip-flops

Latches operate with enable signal, which is **level sensitive**. Whereas, flip-flops are edge sensitive.

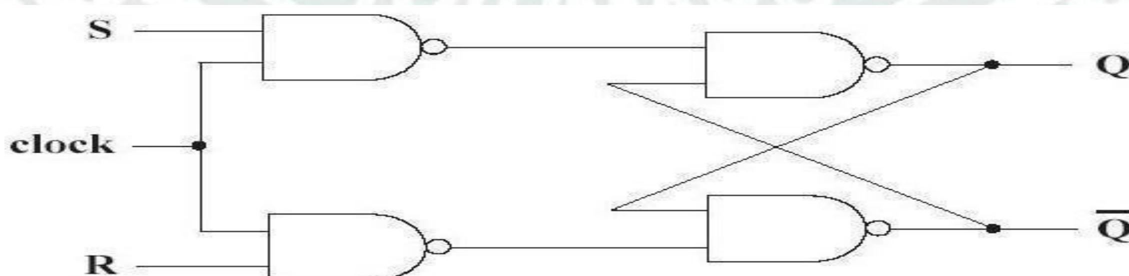
There are 4 types of flip flops:

- SR Flip-Flop
- D Flip-Flop
- JK Flip-Flop
- T Flip-Flop

SR Flip-Flop

SR flip-flop operates with only positive clock transitions or negative clock transitions. Whereas, SR latch operates with enable signal.

The **circuit diagram** of SR flip-flop is shown in the following figure.



This circuit has two inputs S & R and two outputs Q& Q'. The operation of SR flipflop is similar to SR Latch. But, this flip-flop affects the outputs only when positive transition of the clock signal is applied instead of active enable.

The following table shows the **state table** of SR flip-flop.

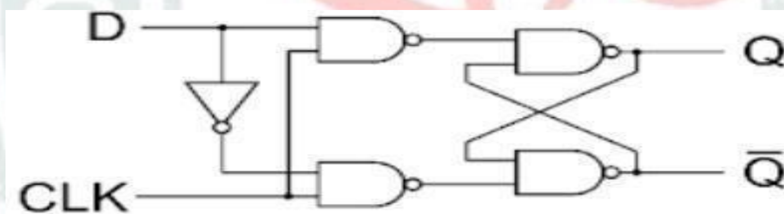
S	R	Qt+1
0	0	Q
0	1	0
1	0	1
1	1	-

Here, Q & Qt+1 are present state & next state respectively. So, SR flip-flop can be used for one of these three functions such as Hold, Reset & Set based on the input conditions, when positive transition of clock signal is applied.

D Flip-Flop

D flip-flop operates with only positive clock transitions or negative clock transitions. Whereas, D latch operates with enable signal. That means, the output of D flip-flop is insensitive to the changes in the input, D except for active transition of the clock signal.

The **circuit diagram** of D flip-flop is shown in the following figure.



This circuit has single input D and two outputs Q & Q'. The operation of D flip-flop is similar to D Latch. But, this flip-flop affects the outputs only when positive transition of the clock signal is applied instead of active enable.

The following table shows the **state table** of D flip-flop.

D	Qt + 1
0	0
1	1

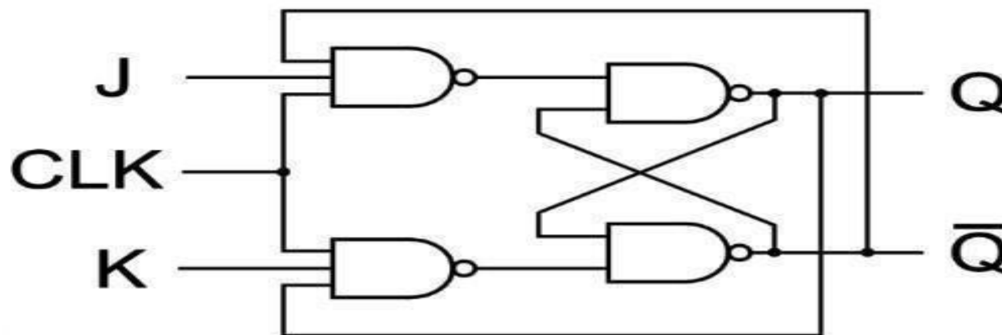
Therefore, D flip-flop always Hold the information, which is available on data input, D of earlier positive transition of clock signal.

D flip-flops can be used in registers, **shift registers** and some of the counters.

JK Flip-Flop

JK flip-flop is the modified version of SR flip-flop. It operates with only positive clock transitions or negative clock transitions.

The **circuit diagram** of JK flip-flop is shown in the following figure.



This circuit has two inputs J & K and two outputs Q & Q’. The operation of JK flip-flop is similar to SR flip-flop.

The following table shows the **state table** of JK flip-flop.

J	K	**Q**t+1
0	0	Q
0	1	0
1	0	1
1	1	Q'

Here, Q & Qt+1 are present state & next state respectively. So, JK flip-flop can be used for one of these four functions such as Hold, Reset, Set & Complement of present state based on the input conditions, when positive transition of clock signal is applied.

Master-Slave JK Flip Flop

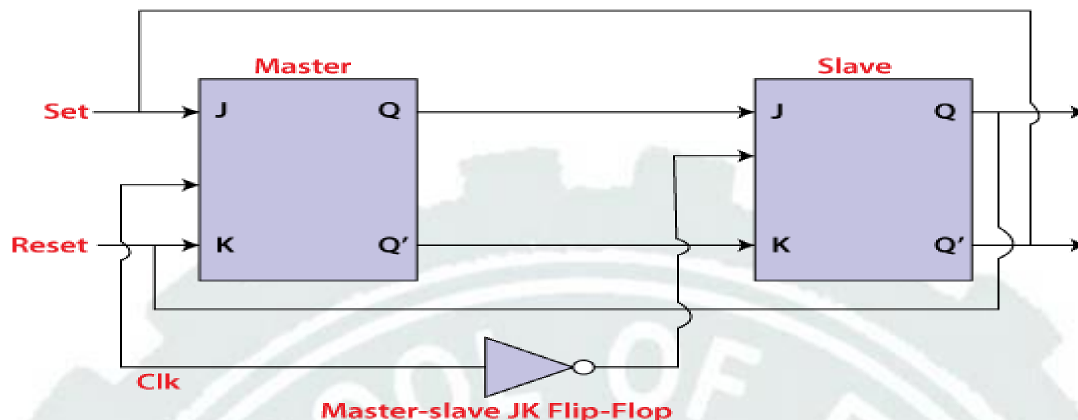
In "JK Flip Flop", when both the inputs and CLK set to 1 for a long time, then Q output toggle until the CLK is 1. Thus, the uncertain or unreliable output produces. This problem is referred to as a **race-round condition** in JK flip-flop and avoided by ensuring that the CLK set to 1 only for a very short time.

Explanation

The master-slave flip flop is constructed by combining two J K flip flop. These flip flops are connected in a series configuration. In these two flip flops, the 1st flip flop work as "master", called the master flip flop, and the 2nd work as a "slave", called slave flip flop.

In "master-slave flip flop", apart from these two flip flops, an inverter or NOT gate is also used. For passing the inverted clock pulse to the "slave" flip flop, the inverter is connected to the

clock's pulse. In simple words, when CP set to false for "master", then CP is set to true for "slave", and when CP set to true for "master", then CP is set to false for "slave".

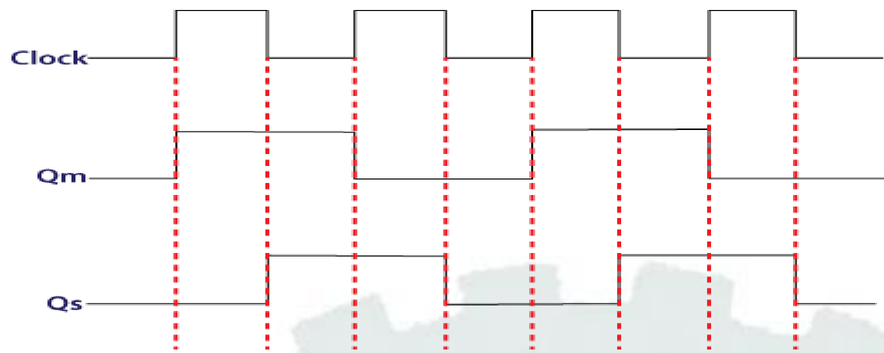


Master-slave JK Flip-Flop

Working:

- When the clock pulse is true, the slave flip flop will be in the isolated state, and the system's state may be affected by the J and K inputs. The "slave" remains isolated until the CP is 1. When the CP set to 0, the master flip-flop passes the information to the slave flip flop to obtain the output.
- The master flip flop responds first from the slave because the master flip flop is the positive level trigger, and the slave flip flop is the negative level trigger.
- The output Q'=1 of the master flip flop is passed to the slave flip flop as an input K when the input J set to 0 and K set to 1. The clock forces the slave flip flop to work as reset, and then the slave copies the master flip flop.
- When J=1, and K=0, the output Q=1 is passed to the J input of the slave. The clock's negative transition sets the slave and copies the master.
- The master flip flop toggles on the clock's positive transition when the inputs J and K set to 1. At that time, the slave flip flop toggles on the clock's negative transition.
- The flip flop will be disabled, and Q remains unchanged when both the inputs of the JK flip flop set to 0.

Timing Diagram of a Master Flip Flop:

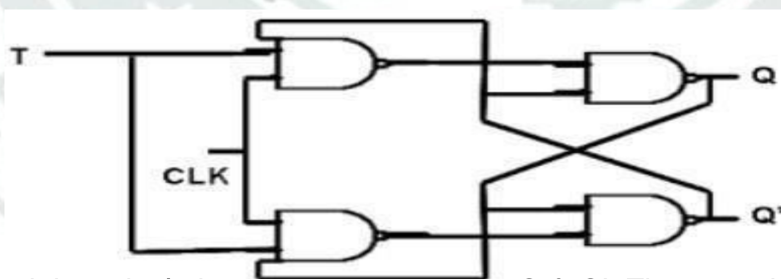


- When the clock pulse set to 1, the output of the master flip flop will be one until the clock input remains 0.
- When the clock pulse becomes high again, then the master's output is 0, which will be set to 1 when the clock becomes one again.
- The master flip flop is operational when the clock pulse is 1. The slave's output remains 0 until the clock is not set to 0 because the slave flip flop is not operational.
- The slave flip flop is operational when the clock pulse is 0. The output of the master remains one until the clock is not set to 0 again.
- Toggling occurs during the entire process because the output changes once in the cycle.

T Flip-Flop

T flip-flop is the simplified version of JK flip-flop. It is obtained by connecting the same input 'T' to both inputs of JK flip-flop. It operates with only positive clock transitions or negative clock transitions.

The **circuit diagram** of T flip-flop is shown in the following figure.



This circuit has single input T and two outputs Q & Q'. The operation of T flip-flop is same as that of JK flip-flop. Here, we considered the inputs of JK flip-flop as **J = T** and **K = T** in order to utilize the modified JK flip-flop for 2 combinations of inputs. So, we eliminated the other two combinations of J & K, for which those two values are complement to each other in T flipflop.

The following table shows the **state table** of T flip-flop.

D	**Q**t+1

0	Q
1	Q'

Here, Q & Qt+1 are present state & next state respectively. So, T flip-flop can be used for one of these two functions such as Hold, & Complement of present state based on the input conditions, when positive transition of clock signal is applied.
The output of T flip-flop always toggles for every positive transition of the clock signal, when input T remains at logic High 11. Hence, T flip-flop can be used in **counters**.

UNIT-4

Registers, memories and PLD

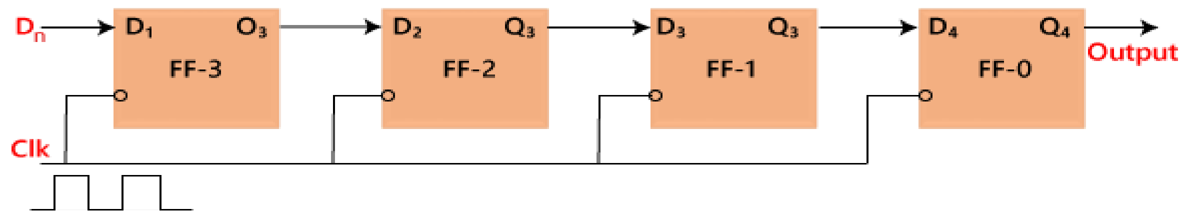
Shift register

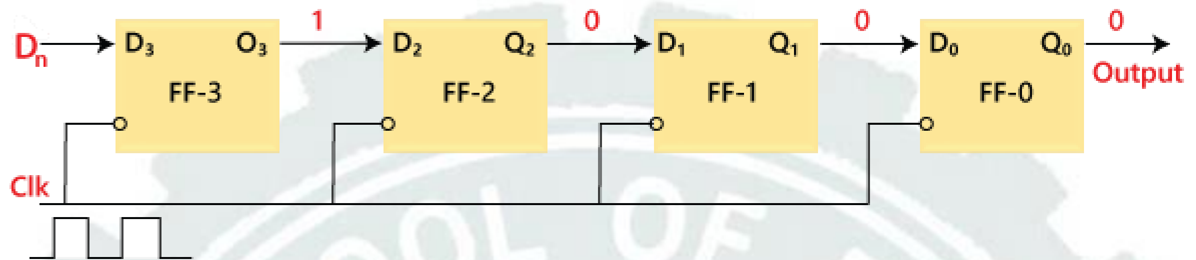
- Flip flops can be used to store a single bit of binary data (1or 0).
- However, in order to store multiple bits of data, we need multiple flip flops. N flip flops are to be connected in an order to store n bits of data.
- A **Register** is a device which is used to store such information. It is a group of flip flops connected in series used to store multiple bits of data.
- The information stored within these registers can be transferred with the help of **shift registers**.
- Shift Register is a group of flip flops used to store multiple bits of data. The bits stored in such registers can be made to move within the registers and in/out of the registers by applying clock pulses.
- An n-bit shift register can be formed by connecting n flip-flops where each flip flop stores a single bit of data.

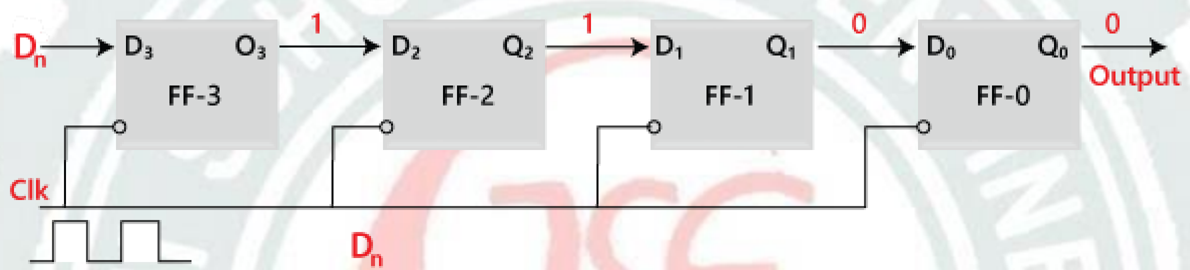
Shift registers are basically of 4 types. These are:

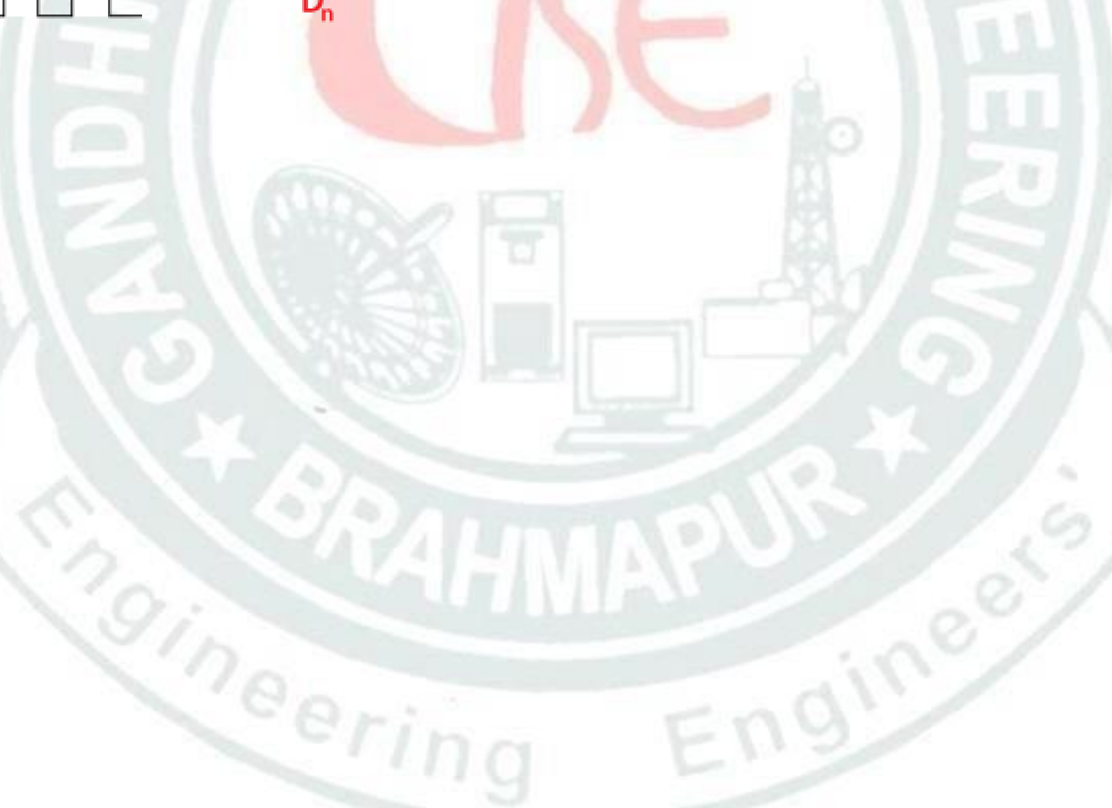
1. Serial In Serial Out shift register
2. Serial In parallel Out shift register
3. Parallel In Serial Out shift register
4. Parallel In parallel Out shift register

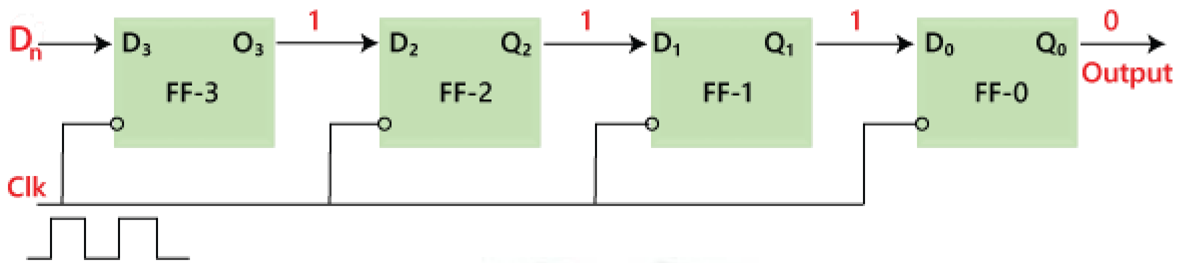
Serial-In Serial-Out Shift Register (SISO) –
The shift register, which allows serial input (one bit after the other through a single data line) and produces a serial output is known as Serial-In Serial-Out shift register. Since there is only one output, the data leaves the shift register one bit at a time in a serial pattern, thus the name Serial-In Serial-Out Shift Register. The circuit consists of four D flip-flops which are connected in a serial manner. All these flip-flops are synchronous with each other since the same clock signal is applied to each flip flop. The main use of a SISO is to act as a delay element.

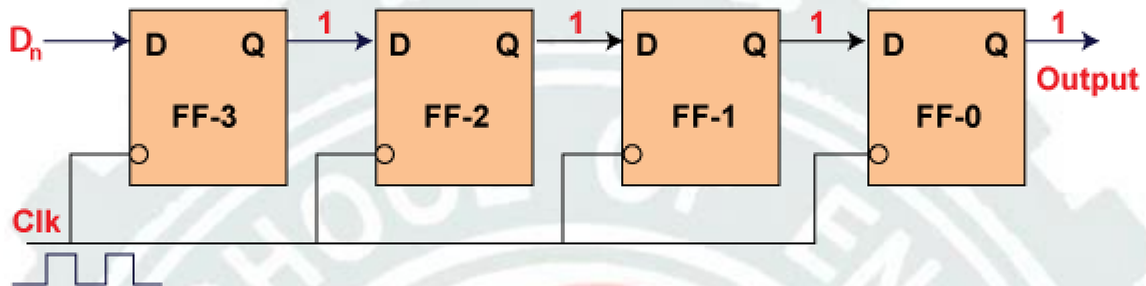

Dn
D1
O3
FF-3
D2
Q3
FF-2
D3
Q3
FF-1
D4
Q4
FF-0
Output
Clk


Dn
D3
O3
FF-3
1
D2
Q2
FF-2
0
D1
Q1
FF-1
0
D0
Q0
FF-0
0
Output
Clk


Dn
D3
O3
FF-3
1
D2
Q2
FF-2
1
D1
Q1
FF-1
0
D0
Q0
FF-0
0
Output
Clk
Dn

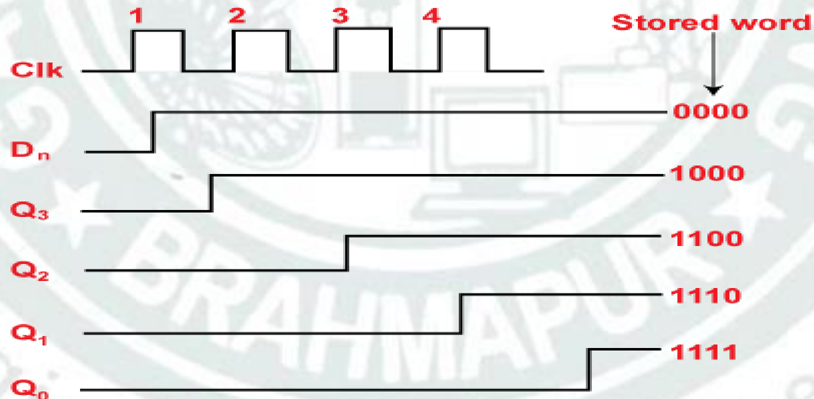






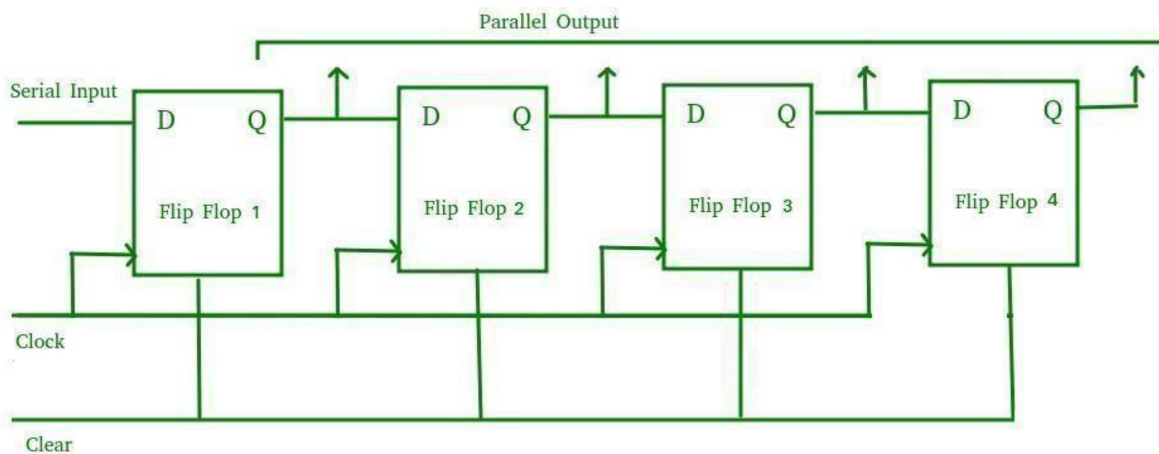
	Clk	$D_n = Q_3$	$Q_3 = D_2$	$Q_2 = D_1$	$Q_1 = D_0$	Q_0
Initially			0	0	0	0
(1)	↓	1	1	0	0	0
(2)	↓	1	1	1	0	0
(3)	↓	1	1	1	1	0
(4)	↓	1	1	1	1	1

→ Direction of data travel



Serial-In Parallel-Out shift Register (SIPO) –

The shift register, which allows serial input (one bit after the other through a single data line) and produces a parallel output is known as Serial-In Parallel-Out shift register. The circuit consists of four D flip-flops which are connected. The clear (CLR) signal is connected in addition to the clock signal to all the 4 flip flops in order to RESET them. The output of the first flip flop is connected to the input of the next flip flop and so on. All these flip-flops are synchronous with each other since the same clock signal is applied to each flip flop. They are used in communication lines where demultiplexing of a data line into several parallel lines is required because the main use of the SIPO register is to convert serial data into parallel data.



Parallel IN Serial OUT (PISO)

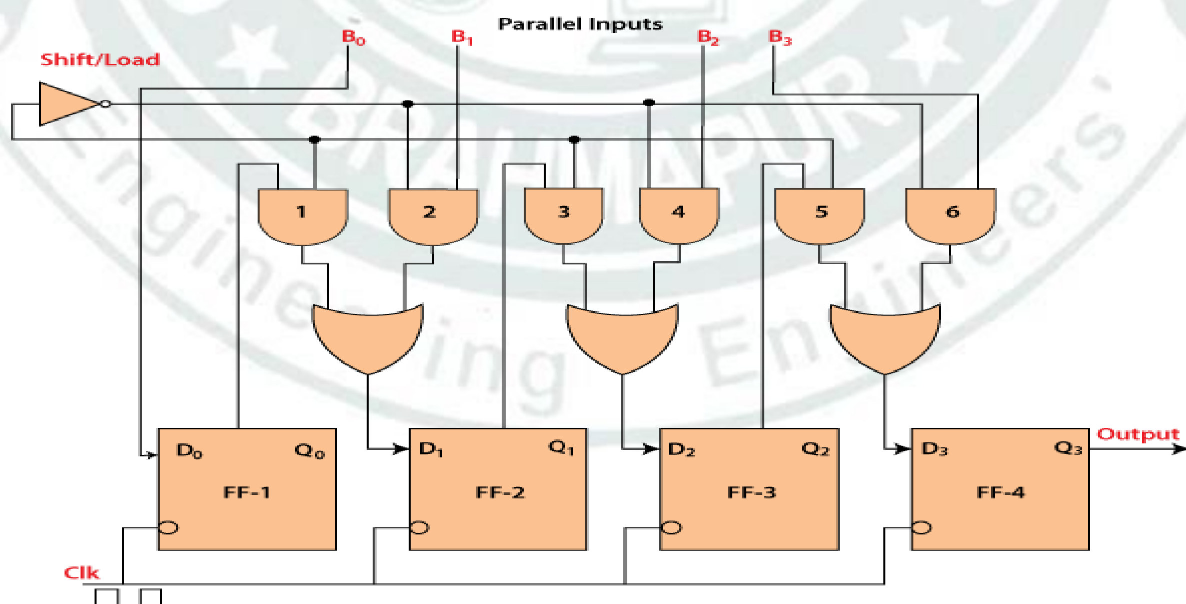
In the **"Parallel IN Serial OUT"** register, the data is entered in a parallel way, and the outcome comes serially. A four-bit **"Parallel IN Serial OUT"** register is designed below. The input of the flip flop is the output of the previous Flip Flop. The input and outputs are connected through the combinational circuit. Through this combinational circuit, the binary input B_0, B_1, B_2, B_3 are passed. The **shift mode** and the **load mode** are the two modes in which the **"PISO"** circuit works.

Load mode

The bits B_0, B_1, B_2, and B_3 are passed to the corresponding flip flops when the second, fourth, and sixth "AND" gates are active. These gates are active when the shift or load bar line set to 0. The binary inputs B0, B1, B2, and B3 will be loaded into the respective flip-flops when the edge of the clock is low. Thus, parallel loading occurs.

Shift mode

The second, fourth, and sixth gates are inactive when the load and shift line set to 0. So, we are not able to load data in a parallel way. At this time, the first, third, and fifth gates will be activated, and the shifting of the data will be left to the right bit. In this way, the **"Parallel IN Serial OUT"** operation occurs.

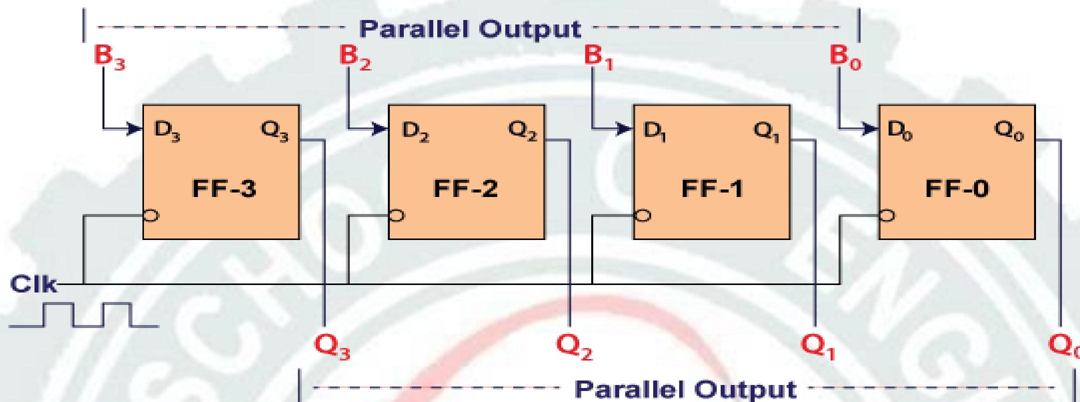


A Parallel in Serial out (PISO) shift register us used to convert parallel data to serial data.

Parallel IN Parallel OUT (PIPO)

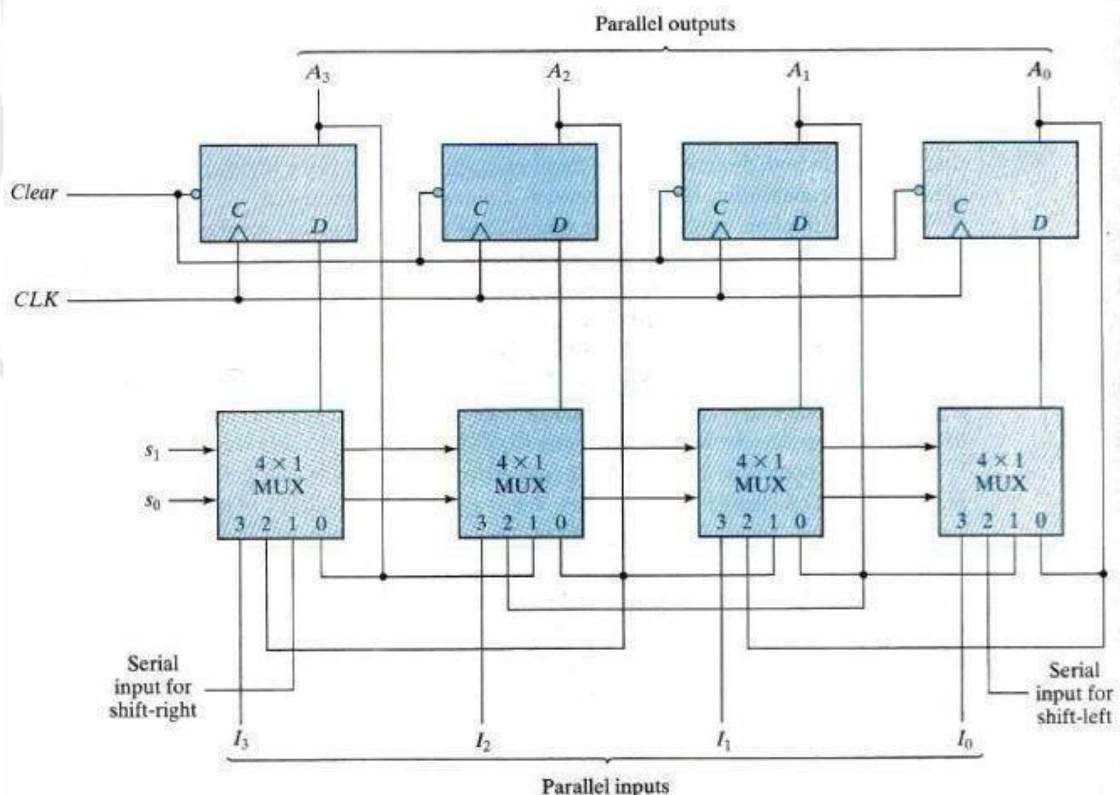
In **"Parallel IN Parallel OUT"**, the inputs and the outputs come in a parallel way in the register.

The inputs A_0, A_1, A_2, and A_3, are directly passed to the data inputs D_0, D_1, D_2, and D_3 of the respective flip flop. The bits of the binary input is loaded to the flip flops when the negative clock edge is applied. The clock pulse is required for loading all the bits. At the output side, the loaded bits appear.



Universal shift register

A Universal shift register is a register which has both the right shift and left shift with parallel load capabilities. Universal shift registers are used as memory elements in computers. A Unidirectional shift register is capable of shifting in only one direction. A bidirectional shift register is capable of shifting in both the directions. The Universal shift register is a combination design of **bidirectional** shift register and a **unidirectional** shift register with parallel load provision



Basic connections –

1. The first input (zeroth pin of multiplexer) is connected to the output pin of the corresponding flip-flop.
2. The second input (first pin of multiplexer) is connected to the output of the veryprevious flip flop which facilitates the right shift.
3. The third input (second pin of multiplexer) is connected to the output of the very-next flip-flop which facilitates the left shift.
4. The fourth input (third pin of multiplexer) is connected to the individual bits of the input data which facilitates parallel loading.

The working of the Universal shift register depends on the inputs given to the select lines.

The register operations performed for the various inputs of select lines are as follows:

S1	S0	Operation
0	0	No change
0	1	Shift right
1	0	Shift left
1	1	Parallel load

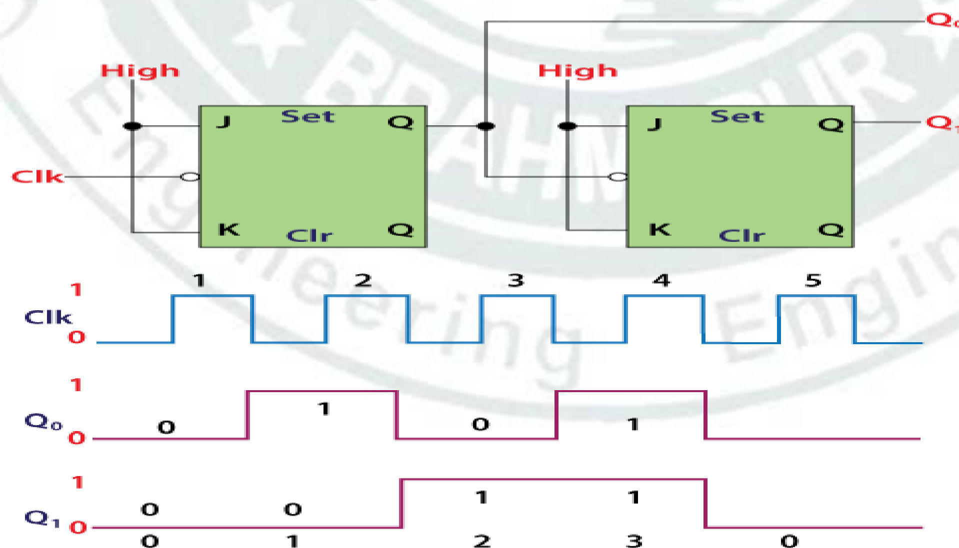
Counters

Counter is a sequential circuit. A digital circuit which is used for a counting pulses is known counter. Counter is the widest application of flip-flops. It is a group of flip-flops with a clock signal applied. Counters are of two types.

- Asynchronous or ripple counters. Synchronous counters.

Asynchronous or ripple counters

The **Asynchronous counter** is also known as the **ripple counter**. Below is a diagram of the 2-bit **Asynchronous counter** in which we used two T flip-flops or two JK flip flopby setting both of the inputs to 1 permanently. The external clock pass to the clock input of the first flip flop, i.e., FF-A and its output, i.e., is passed to clock input of the next flip flop, i.e., FF-B.



Operation:

1. **Condition 1:** When both the flip flops are in reset condition.
   Operation: The outputs of both flip flops, i.e., Q_A Q_B, will be 0.
2. **Condition 2:** When the first negative clock edge passes.
   Operation: The first flip flop will toggle, and the output of this flip flop will change from 0 to 1. The output of this flip flop will be taken by the clock input of the next flip flop. This output will be taken as a positive edge clock by the second flip flop. This input will not change the second flip flop's output state because it is the negative edge triggered flip flop. So, Q_A = 1 and Q_B = 0
3. **Condition 3:** When the second negative clock edge is applied.
   Operation: The first flip flop will toggle again, and the output of this flip flop will change from 1 to 0. This output will be taken as a negative edge clock by the second flip flop. This input will change the second flip flop's output state because it is the negative edge triggered flip flop. So, Q_A = 0 and Q_B = 1.
4. **Condition 4:** When the third negative clock edge is applied.
   Operation: The first flip flop will toggle again, and the output of this flip flop will change from 0 to 1. This output will be taken as a positive edge clock by the second flip flop. This input will not change the second flip flop's output state because it is the negative edge triggered flip flop. So, Q_A = 1 and Q_B = 1
5. **Condition 5:** When the fourth negative clock edge is applied.
   Operation: The first flip flop will toggle again, and the output of this flip flop will change from 1 to 0. This output will be taken as a negative edge clock by the second flip flop. This input will change the output state of the second flip flop. So, Q_A = 0 and Q_B = 0

Classification of counters

Depending on the way in which the counting progresses, the synchronous or asynchronous counters are classified as follows −

- Up counters
- Down counters
- Up/Down counters

UP/DOWN Counter

Up counter and down counter is combined together to obtain an UP/DOWN counter. A mode control (M) input is also provided to select either up or down mode. A combinational circuit is required to be designed and used between each pair of flip-flop in order to achieve the up/down operation.

- Type of up/down counters
- UP/DOWN ripple counters
- UP/DOWN synchronous counter

UP/DOWN Counter

Up counter and down counter is combined together to obtain an UP/DOWN counter. A mode control (M) input is also provided to select either up or down mode. A combinational circuit is

required to be designed and used between each pair of flip-flop in order to achieve the up/down operation.

- Type of up/down counters
- UP/DOWN ripple counters
- UP/DOWN synchronous counter

UP/DOWN Ripple Counters

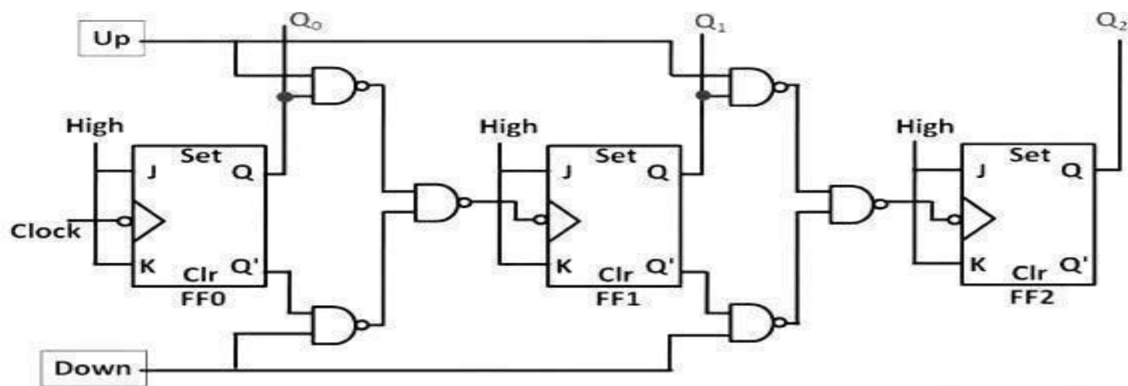
In the UP/DOWN ripple counter all the FFs operate in the toggle mode. So, either T flip-flops or JK flip-flops are to be used. The LSB flip-flop receives clock directly. But the clock to every other FF is obtained from (Q = Q bar) output of the previous FF.

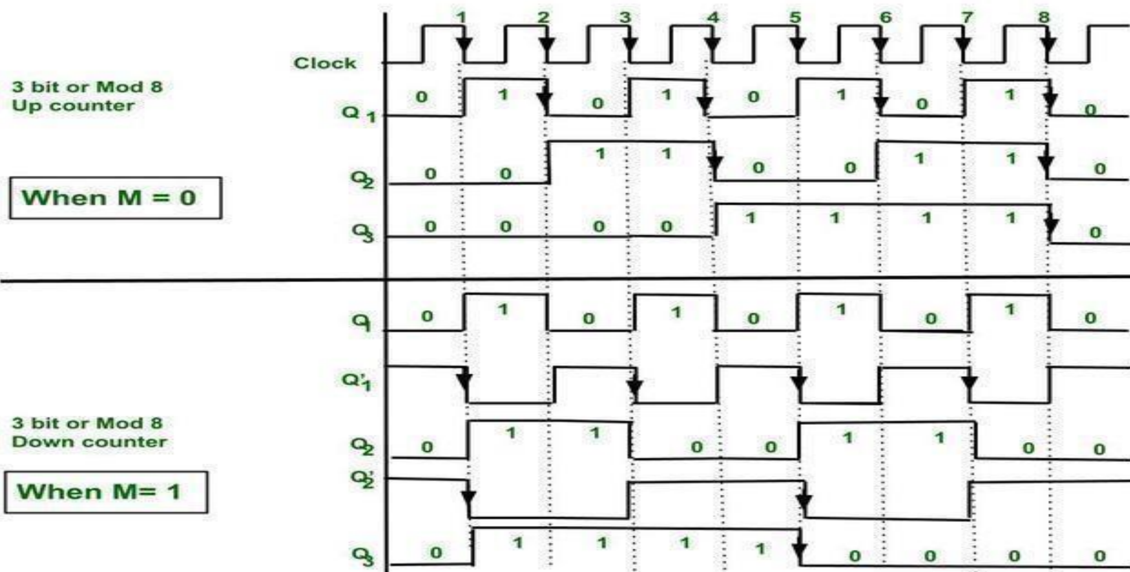
- **UP counting mode (M=0)** − The Q output of the preceding FF is connected to the clock of the next stage if up counting is to be achieved. For this mode, the mode select input M is at logic 0 (M=0).
- **DOWN counting mode (M=1)** − If M = 1, then the Q bar output of the preceding FF is connected to the next FF. This will operate the counter in the counting mode.

Example

3-bit binary up/down ripple counter.

- 3-bit − hence three FFs are required.
- UP/DOWN − So a mode control input is essential.
- For a ripple up counter, the Q output of preceding FF is connected to the clock input of the next one.
- For a ripple up counter, the Q output of preceding FF is connected to the clock input of the next one.
- For a ripple down counter, the Q bar output of preceding FF is connected to the clock input of the next one.
- Let the selection of Q and Q bar output of the preceding FF be controlled by the mode control input M such that, If M = 0, UP counting. So, connect Q to CLK. If M = 1, DOWN counting. So, connect Q bar to CLK.





Modulus Counter (MOD-N Counter)

The 2-bit ripple counter is called as MOD-4 counter and 3-bit ripple counter is called as MOD8 counter. So, in general, an n-bit ripple counter is called as modulo-N counter. Where, MOD number = 2^n.

Type of modulus

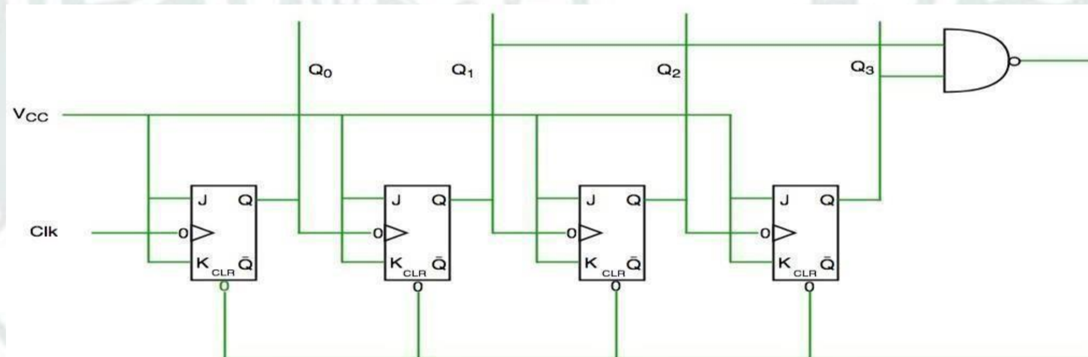
- 2-bit up or down (MOD-4)
- 3-bit up or down (MOD-8)
- 4-bit up or down (MOD-16) **Application of counters**
- Frequency counters
- Digital clock
- Time measurement
- A to D converter
- Frequency divider circuits
- Digital triangular wave generator.

Decade counter

A decade counter counts ten different states and then reset to its initial states. A simple decade counter will count from 0 to 9 but we can also make the decade counters which can go through any ten states between 0 to 15 (for 4 bit counter).

Clock
pulse Q3 Q2 Q1 Q0

0	0	0	0	0
1	0	0	0	1
2	0	0	1	0
3	0	0	1	1
4	0	1	0	0
5	0	1	0	1
6	0	1	1	0
7	0	1	1	1
8	1	0	0	0
9	1	0	0	1
10	0	0	0	0



We see from circuit diagram that we have used NAND gate for Q3 and Q1 and feeding this to clear input line because binary representation of 10 is—1010

And we see Q3 and Q1 are 1 here, if we give NAND of these two bits to clear input then counter will be clear at 10 and again start from beginning.

Synchronous counters

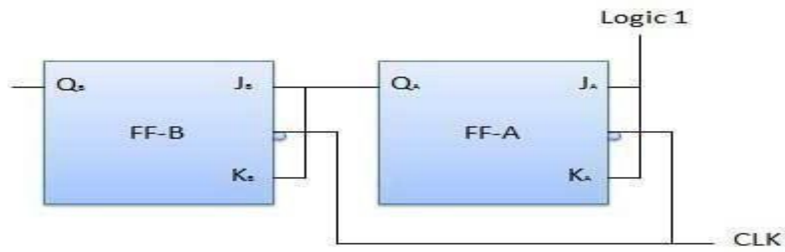
If the "clock" pulses are applied to all the flip-flops in a counter simultaneously, then such a counter is called as synchronous counter.

2-bit Synchronous up counter

The J_A and K_A inputs of FF-A are tied to logic 1. So, FF-A will work as a toggle flip-flop. The J_B and K_B inputs are connected to Q_A.

Operation

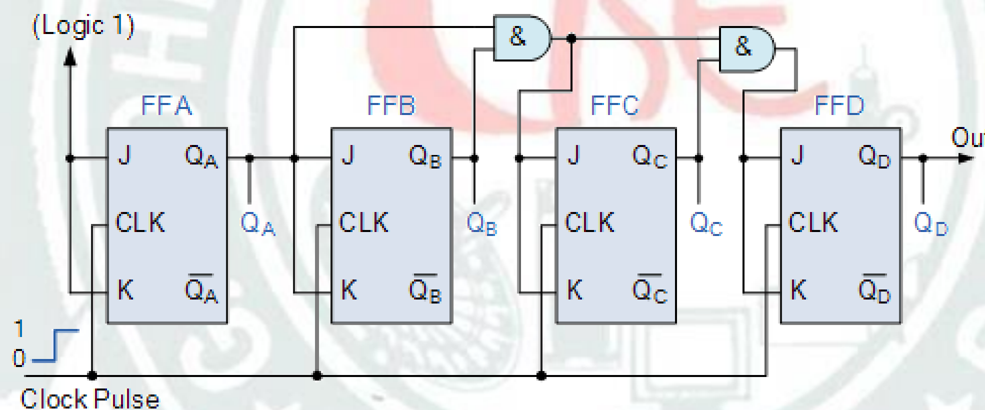
S.N.	Condition	Operation
1	**Initially let both the FFs be in the reset state**	Q_BQ_A = 00 initially.
2	**After 1st negative clock edge**	As soon as the first negative clock edge is applied, FF-A will toggle and Q_A will change from 0 to 1. But at the instant of application of negative clock edge, Q_A , J_B = K_B = 0. Hence FF-B will not change its state. So Q_B will remain 0. Q_BQ_A = 01 after the first clock pulse.
3	**After 2nd negative clock edge**	On the arrival of second negative clock edge, FF-A toggles again and Q_A changes from 1 to 0. But at this instant Q_A was 1. So J_B = K_B= 1 and FF-B will toggle. Hence Q_B changes from 0 to 1. Q_BQ_A = 10 after the second clock pulse.
4	**After 3rd negative clock edge**	On application of the third falling clock edge, FF-A will toggle from 0 to 1 but there is no change of state for FF-B. Q_BQ_A = 11 after the third clock pulse.
5	**After 4th negative clock edge**	On application of the next clock pulse, Q_A will change from 1 to 0 as Q_B will also change from 1 to 0. Q_BQ_A = 00 after the fourth clock pulse.

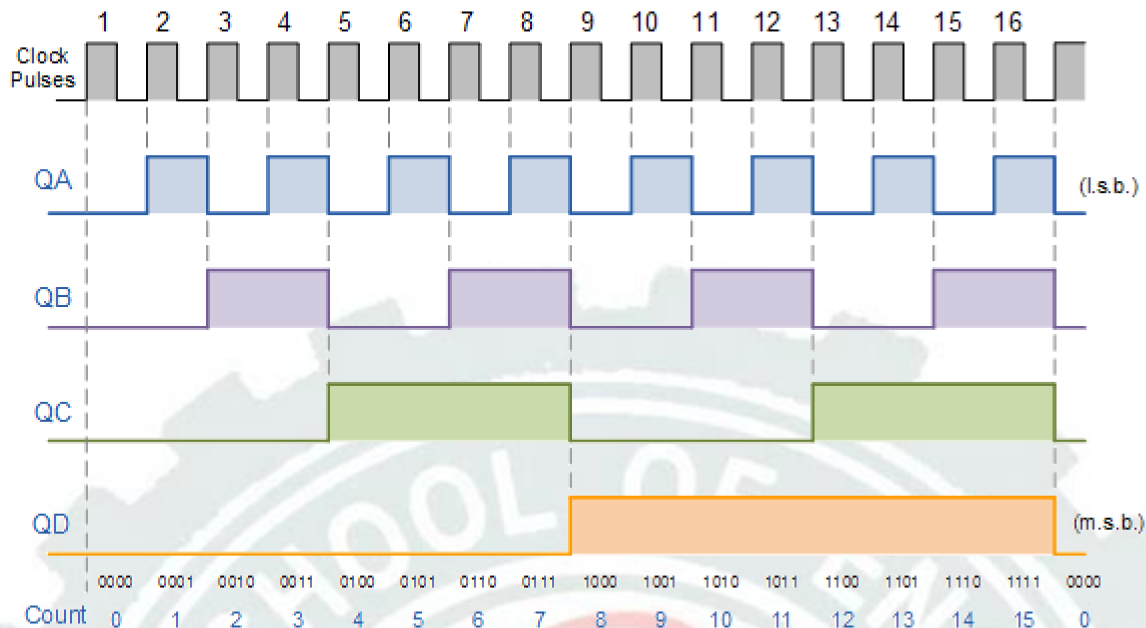


GANDHI SCHOOL OF ENGINEERING
BRAHMAPUR
Engineering Engineers'

4-bit synchronous counter

- The external clock pulses (pulses to be counted) are fed directly to each of the J-K flipflops in the counter chain and that both the J and K inputs are all tied together in toggle mode, but only in the first flip-flop, flip-flop FFA (LSB) are they connected HIGH, logic
  "1" allowing the flip-flop to toggle on every clock pulse.
- Then the synchronous counter follows a predetermined sequence of states in response to the common clock signal, advancing one state for each pulse.
- The J and K inputs of flip-flop FFB are connected directly to the output Q_A of flipflop FFA, but the J and K inputs of flip-flops FFC and FFD are driven from separate AND gates which are also supplied with signals from the input and output of the previous stage.
- These additional AND gates generate the required logic for the JK inputs of the next stage.
- If we enable each JK flip-flop to toggle based on whether or not all preceding flip-flop outputs (Q) are "HIGH" we can obtain the same counting sequence as with the asynchronous circuit but without the ripple effect, since each flip-flop in this circuit will be clocked at exactly the same time.
- Then as there is no inherent propagation delay in synchronous counters, because all the counter stages are triggered in parallel at the same time, the maximum operating frequency of this type of frequency counter is much higher than that for a similar asynchronous counter circuit.



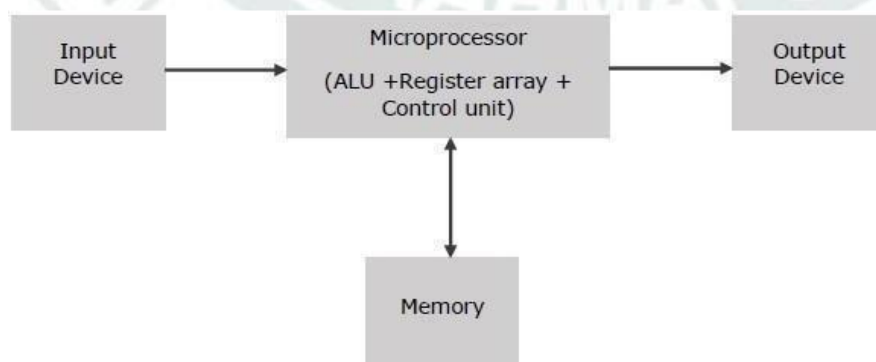


UNIT-4

8085 Microprocessor

- A Microprocessor is a multipurpose, programmable, clock-driven, register-based electronic device that reads binary instructions from a storage device called memory, accepts binary data as input and processes data according to those instructions and provide results as output.
- Microprocessor is a controlling unit of a micro-computer, fabricated on a small chip capable of performing ALU (Arithmetic Logical Unit) operations and communicating with the other devices connected to it.
- Microprocessor consists of an ALU, register array, and a control unit.
- ALU performs arithmetical and logical operations on the data received from the memory or an input device.
- Register array consists of registers identified by letters like B, C, D, E, H, L and accumulator.
- The control unit controls the flow of data and instructions within the computer.

Block Diagram of a Basic Microcomputer



How does a Microprocessor Work?

The microprocessor follows a sequence: Fetch, Decode, and then Execute.

- Initially, the instructions are stored in the memory in a sequential order.
- The microprocessor fetches those instructions from the memory, then decodes it and executes those instructions till STOP instruction is reached.
- Later, it sends the result in binary to the output port. Between these processes, the register stores the temporarily data and ALU performs the computing functions.

Classification of Microprocessor

Microprocessor is classified into two categories-

RISC & CISC

RISC Processor

- RISC stands for Reduced Instruction Set Computer.
- It is designed to reduce the execution time by simplifying the instruction set of the computer.
- Using RISC processors, each instruction requires only one clock cycle to execute results in uniform execution time.
- This reduces the efficiency as there are more lines of code, hence more RAM is needed to store the instructions.
- The compiler also has to work more to convert high-level language instructions into machine code.

Characteristics of RISC

The major characteristics of a RISC processor are as follows −

- It consists of simple instructions.
- It supports various data-type formats.
- It utilizes simple addressing modes and fixed length instructions for pipelining.
- It supports register to use in any context. One cycle execution time.
- “LOAD” and “STORE” instructions are used to access the memory location. It consists of larger number of registers.
- It consists of less number of transistors.

CISC Processor

- CISC stands for Complex Instruction Set Computer.
- It is designed to minimize the number of instructions per program, ignoring the number of cycles per instruction.
- The emphasis is on building complex instructions directly into the hardware.
- The compiler has to do very little work to translate a high-level language into assembly level language/machine code because the length of the code is relatively short, so very little RAM is required to store the instructions.

Characteristics of CISC

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

8085 Microprocessor

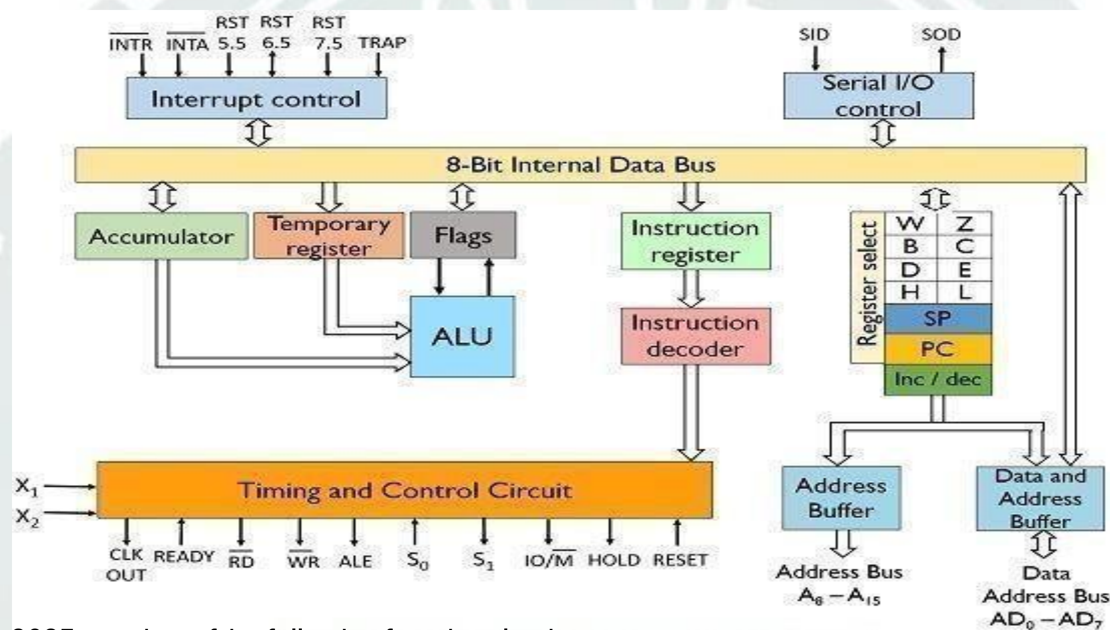
It is an 8-bit microprocessor designed by Intel in 1977 using NMOS technology.

It has the following configuration −

8-bit data bus

- 16-bit address bus, which can address upto 64KB
- A 16-bit program counter
- A 16-bit stack pointer
- Six 8-bit registers arranged in pairs: BC, DE, HL
- Requires +5V supply to operate at 3 MHZ single phase clock It is used in washing machines, microwave ovens, mobile phones, etc.

8085 Microprocessor – Functional Units



8085 consists of the following functional units −

Accumulator

It is an 8-bit register used to perform arithmetic, logical, I/O & LOAD/STORE operations. It is connected to internal data bus & ALU. **Arithmetic and logic unit**

As the name suggests, it performs arithmetic and logical operations like Addition, Subtraction, AND, OR, etc. on 8-bit data.

General purpose register

There are 6 general purpose registers in 8085 processor, i.e., B, C, D, E, H & L. Each register can hold 8-bit data.

These registers can work in pair to hold 16-bit data and their pairing combination is like B-C, D-E & H-L.

Program counter

It is a 16-bit register used to store the memory address location of the next instruction to be executed. Microprocessor increments the program whenever an instruction is being executed, so that the program counter points to the memory address of the next instruction that is going to be executed.

Stack pointer

It is also a 16-bit register works like stack, which is always incremented/decremented by 2 during push & pop operations. **Temporary register**

It is an 8-bit register, which holds the temporary data of arithmetic and logical operations. **Flag register**

It is an 8-bit register having five 1-bit flip-flops, which holds either 0 or 1 depending upon the result stored in the accumulator.

These are the set of 5 flip-flops − Sign
(S)- set to 1 if result is negative.
Zero (Z)- set to 1 if result is zero.
Auxiliary Carry (AC)- set to 1 if carry arises from 3rd bit to 4th bit.
Parity (P)- set to 1 if result has even no. of 1.
Carry (CS)- set to 1 if carry arises after arithmetic and logical operation. Its bit position is shown in the following table –

B7	B6	B5	B4	B3	B2	B1	B0
S	Z	X	AC	X	P	X	CS

Instruction register and decoder

It is an 8-bit register. When an instruction is fetched from memory then it is stored in the Instruction register. Instruction decoder decodes the information present in the Instruction register.

Timing and control unit

It provides timing and control signal to the microprocessor to perform operations. Following are the timing and control signals, which control external and internal circuits − Control Signals: READY, RD', WR', ALE
Status Signals: S0, S1, IO/M'
DMA Signals: HOLD, HLDA
RESET Signals: RESET IN, RESET OUT

Interrupt control

As the name suggests it controls the interrupts during a process. When a microprocessor is executing a main program and whenever an interrupt occurs, the microprocessor shifts the control from the main program to process the incoming request. After the request is completed, the control goes back to the main program.

There are 5 interrupt signals in 8085 Microprocessor: INTR, RST 7.5, RST 6.5, RST 5.5, TRAP.

Serial Input/output control

It controls the serial data communication by using these two instructions: SID (Serial input data) and SOD (Serial output data).

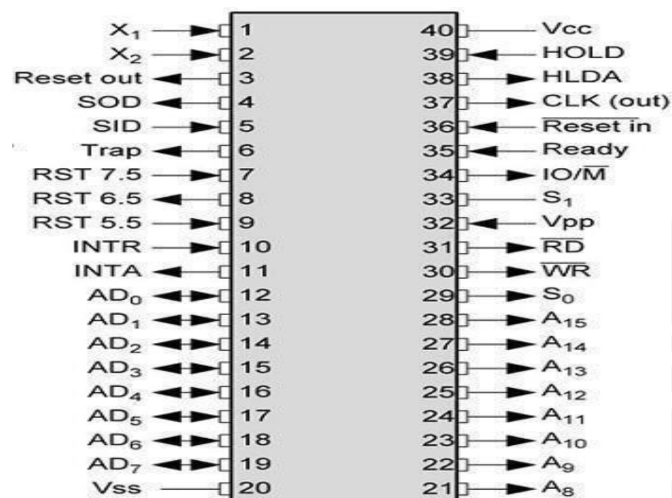
Address buffer and address-data buffer

The content stored in the stack pointer and program counter is loaded into the address buffer and address-data buffer to communicate with the CPU. The memory and I/O chips are connected to these buses; the CPU can exchange the desired data with the memory and I/O chips.

Address bus and data bus

Data bus carries the data to be stored. It is bidirectional, whereas address bus carries the location to where it should be stored and it is unidirectional. It is used to transfer the data & Address I/O devices.

Pin diagram and description



The pins of 8085 microprocessor can be classified into seven groups −

Address bus

A15-A8, it carries the most significant 8-bits of memory/IO address.

Data bus

AD7-AD0, it carries the least significant 8-bit address and data bus.

Control and status signals

These signals are used to identify the nature of operation. There are 3 control signal and 3 status signals.

Three **control signals** are RD, WR & ALE.

RD − This signal indicates that the selected IO or memory device is to be read and is ready for accepting data available on the data bus.

WR − This signal indicates that the data on the data bus is to be written into a selected memory or IO location.

ALE − It is a positive going pulse generated when a new operation is started by the microprocessor. When the pulse goes high, it indicates address. When the pulse goes down it indicates data.

Three **status signals** are IO/M, S0 & S1.

IO/M

This signal is used to differentiate between IO and Memory operations, i.e. when it is high indicates IO operation and when it is low then it indicates memory operation. **S1 & S0**

These signals are used to identify the type of current operation.

Power supply

There are 2 power supply signals − VCC & VSS. **VCC** indicates +5v power supply and **VSS** indicates ground signal.

Clock signals

There are 3 **clock signals**, i.e. X1, X2, CLK OUT.

X1, X2 − A crystal (RC, LC N/W) is connected at these two pins and is used to set frequency of the internal clock generator. This frequency is internally divided by 2.

CLK OUT − This signal is used as the system clock for devices connected with the microprocessor.

Interrupts & externally initiated signals

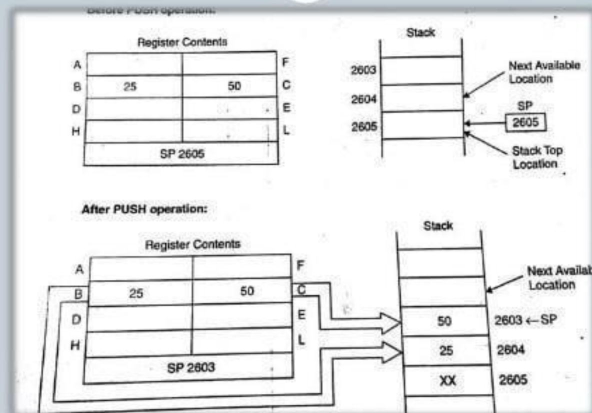
Interrupts are the signals generated by external devices to request the microprocessor to perform a task. There are 5 interrupt signals, i.e., TRAP, RST 7.5, RST 6.5, RST 5.5, and INTR. We will discuss interrupts in detail in interrupts section.

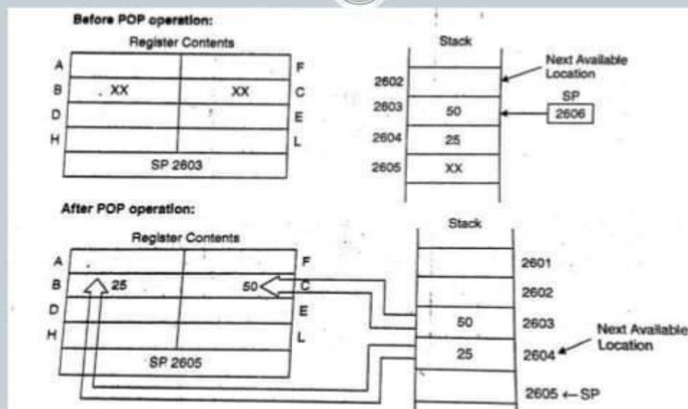
INTA− It is an interrupt acknowledgment signal.

RESET IN − This signal is used to reset the microprocessor by setting the program counter to zero.
RESET OUT − This signal is used to reset all the connected devices when the microprocessor is reset.
READY − This signal indicates that the device is ready to send or receive data. If READY is low, then the CPU has to wait for READY to go high.
HOLD − This signal indicates that another master is requesting the use of the address and data buses.
HLDA(HOLD Acknowledge) − It indicates that the CPU has received the HOLD request andit will relinquish the bus in the next clock cycle. HLDA is set to low after the HOLD signal is removed.
Serial I/O signals
There are 2 serial signals, i.e., SID and SOD and these signals are used for serial communication.
SOD (Serial output data line) − The output SOD is set/reset as specified by the SIM instruction.
SID (Serial input data line) − The data on this line is loaded into accumulator whenever a RIM instruction is executed.

Stack, stack top and stack pointer

- The stack is a LIFO (last in, first out) data structure implemented in the RAM area and is used to store addresses and data when the microprocessor branches to a subroutine.
- Then the return address used to get pushed on this stack.
- Also, to swap values of two registers and register pairs we use the stack as well.
- The Stack Pointer register will hold the address of the top location of the stack.
- On a stack, we can perform two operations.
- PUSH and POP.
- In case of PUSH operation, the SP register gets decreased by 2 and new data item used to insert on to the top of the stack.
- In case of POP operation, the data item will have to be deleted from the top of the stack and the SP register will get increased by the value of 2.





Interrupts

- 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:

1. 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.

2. Vectored and Non-Vectored Interrupts –

Vectored Interrupts are those which have fixed vector address (starting address of subroutine) and after executing these, program control is transferred to that address. Vector Addresses are calculated by the formula 8 * TYPE

INTERRUPT	VECTOR ADDRESS
TRAP (RST 4.5)	24 H
RST 5.5	2C H
RST 6.5	34 H
RST 7.5	3C H

For Software interrupts vector addresses are given by:

INTERRUPT	VECTOR ADDRESS
RST 0	00 H
RST 1	08 H
RST 2	10 H
RST 3	18 H
RST 4	20 H
RST 5	28 H
RST 6	30 H
RST 7	38 H

Non-Vectored Interrupts are those in which vector address is not predefined. The interrupting device gives the address of sub-routine for these interrupts. INTR is the only nonvectored interrupt in 8085 Microprocessor.

3. 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. INTR, RST 7.5, RST 6.5, RST 5.5 are maskable interrupts in 8085 Microprocessor.

Non-Maskable Interrupts are those which cannot be disabled or ignored by microprocessor. TRAP is a non-maskable interrupt. It consists of both level as well as edge triggering and is used in critical power failure conditions.

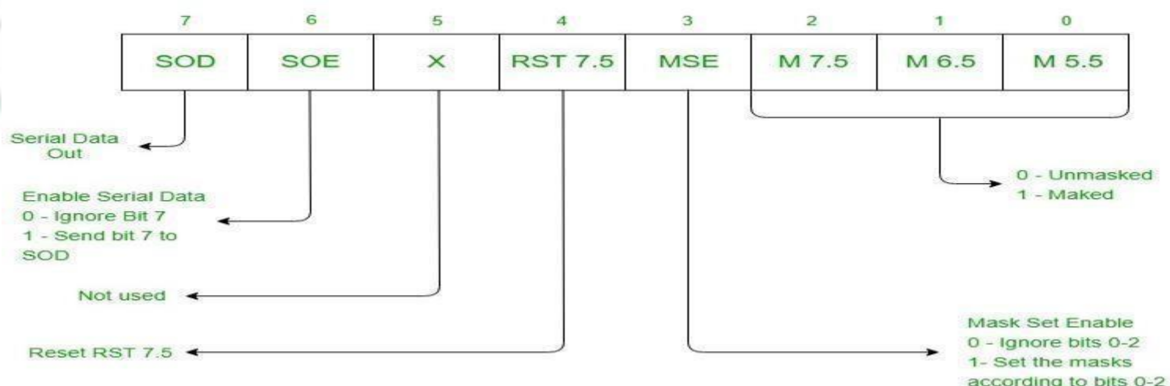
Priority of Interrupts –

When microprocessor receives multiple interrupt requests simultaneously, it will execute the interrupt service request (ISR) according to the priority of the interrupts.

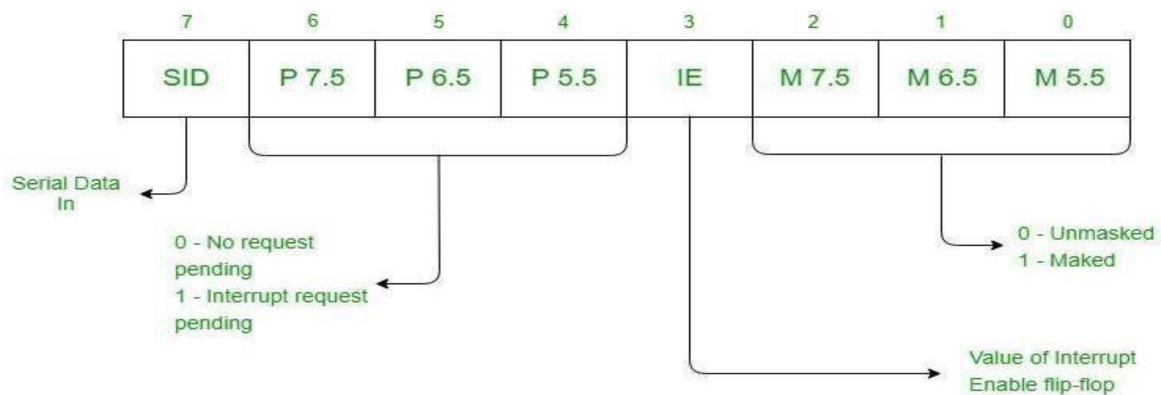
Instruction for Interrupts –

i. 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). **ii. 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.

iii. 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 it.



iv. 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.



Opcodes and operands

- Instruction is divided into two parts: opcodes and operands.
- The opcode is the instruction that is executed by the CPU and the operand is the data or memory location used to execute that instruction.
- An operand (written using hexadecimal notation) provides the data itself, or the location where the data to be processed is stored.
- Some instructions do not require an operand and some may require more than one operand.

Instruction size

- The 8085 instruction set is classified into 3 categories by considering the length of the instructions.
- Three types of instruction are: 1-byte instruction, 2-byte instruction, and 3-byte instruction.

1. One-byte instructions –
In 1-byte instruction, the opcode and the operand of an instruction are represented in one byte.
Example- MOV A,B

2. Two-byte instructions –
Two-byte instruction is the type of instruction in which the first 8 bits indicates the opcode and the next 8 bits indicates the operand.
Example- MVI A,34H
3. Three-byte instructions –
Three-byte instruction is the type of instruction in which the first 8 bits indicates the opcode and the next two bytes specify the 16-bit address. The low-order address is represented in second byte and the high-order address is represented in the third byte. Example- LDA 2000H

Instruction set of 8085 Microprocessor

8085 instruction set is classified in 5 groups- Data transfer group
Arithmetic group
Logical group
Branch control group
Machine control group

Data transfer group

Data transfer instructions are the instructions which transfers data in the microprocessor. They are also called copy instructions.

Opcode	Operand	Explanation	Example
MOV	R1,R2	Move the data from R2 to R1	MOV A,B
MOV	R,M	Move data from memory location to R	MOV B,M
MVI	R,8-bit data	Move the immediate 8-bit data to R	MVI C,34H
MVI	M,8-bit data	Move the immediate 8-bit data to memory location	MVI M,23H
LDA	16-bit address	Load the data from 16-bit address to ACC	LDA 2000H
STA	16-bit address	Store the data of ACC to 16-bit address	STA 2500H
LHLD	16-bit address	Directly loads at H & L registers	LHLD 2050
SHLD	16-bit address	directly stores from H & L registers	SHLD 2050
LXI	rp, 16-bit data	loads the specified register pair with data	LXI H, 3050
XCHG		exchanges H with D, and L with E	XCHG
PUSH	rp	pushes rp to the stack	PUSH H
POP	rp	pops the stack to rp	POP H
IN	8-bit port address	inputs contents of the specified port to A	IN 01
OUT	8-bit port address	outputs contents of A to the specified port	OUT 02

Arithmetic group

Arithmetic Instructions are the instructions which perform basic arithmetic operations such as addition, subtraction and a few more. In 8085 Microprocessor, the destination operand is generally the accumulator. In 8085 Microprocessor, the destination operand is generally the accumulator.

Opcode	Operand	Explanation	Example
ADD	R	A = A + R	ADD B
ADD	M	A = A + M	ADD M
ADI	8-bit data	A = A + 8-bit data	ADI 50
ADC	R	A = A + R + prev. carry	ADC B
ADC	M	A = A + Mc + prev. carry	ADC M
ACI	8-bit data	A = A + 8-bit data + prev. carry	ACI 50
SUB	R	A = A - R	SUB B
SUB	M	A = A - M	SUB M
SUI	8-bit data	A = A - 8-bit data	SUI 50
SBB	R	A = A - R - prev. carry	SBB B
SBB	M	A = A - M -prev. carry	SBB M
SBI	8-bit data	A = A - 8-bit data - prev. carry	SBI 50
INR	R	R = R + 1	INR B
INR	M	M = M + 1	INR M
INX	r.p.	r.p. = r.p. + 1	INX H

DCR	R	R = R - 1	DCR B
DCR	M	M = M - 1	DCR M
DCX	r.p.	r.p. = r.p. - 1	DCX H
DAD	r.p.	HL = HL + r.p.	DAD H

Logical group

Logical instructions are the instructions which perform basic logical operations such as AND, OR, etc. In 8085 Microprocessor, the destination operand is always the accumulator. Here logical operation works on a bitwise level.

Opcode	Operand	Explanation	Example
ANA	R	A = A AND R	ANA B
ANA	M	A = A AND M	ANA M
ANI	8-bit data	A = A AND 8-bit data	ANI 50
ORA	R	A = A OR R	ORA B
ORA	M	A = A OR M	ORA M
ORI	8-bit data	A = A OR 8-bit data	ORI 50
XRA	R	A = A XOR R	XRA B
XRA	M	A = A XOR M	XRA M
XRI	8-bit data	A = A XOR 8-bit data	XRI 50
CMA		A = 1's compliment of A	CMA
CMP	R	Compares R with A and triggers the flag register	CMP B
CMP	M	Compares Mc with A and triggers the flag register	CMP M
CPI	8-bit data	Compares 8-bit data with A and triggers the flag register	CPI 50
RRC		Rotate accumulator right without carry	RRC
RLC		Rotate accumulator left without carry	RLC
RAR		Rotate accumulator right with carry	RAR
RAL		Rotate accumulator left with carry	RAR
CMC		Compliments the carry flag	CMC
STC		Sets the carry flag	STC

Branch group

Branching instructions refer to the act of switching execution to a different instruction sequence as a result of executing a branch instruction. The three types of branching instructions are:

1. Jump (unconditional and conditional)
2. Call (unconditional and conditional)
3. Return (unconditional and conditional)

1. Jump Instructions – The jump instruction transfers the program sequence to the memory address given in the operand based on the specified flag. Jump instructions are 2 types: Unconditional Jump Instructions and Conditional Jump Instructions.

(a) Unconditional Jump Instructions: Transfers the program sequence to the described memory address.

JMP 16-bit address Jumps to the address

Example- JMP 2050

(b) Conditional Jump Instructions: Transfers the program sequence to the described memory address only if the condition in satisfied.

Opcode	Operand	Explanation	Example
JC	Address	Jumps to the address if carry flag is 1	JC 2050
JNC	Address	Jumps to the address if carry flag is 0	JNC 2050

JZ	Address	Jumps to the address if zero flag is 1	JZ 2050
JNZ	Address	Jumps to the address if zero flag is 0	JNZ 2050
JPE	Address	Jumps to the address if parity flag is 1	JPE 2050
JPO	Address	Jumps to the address if parity flag is 0	JPO 2050
JM	Address	Jumps to the address if sign flag is 1	JM 2050
JP	Address	Jumps to the address if sign flag 0	JP 2050
JC	Address	Jumps to the address if carry flag is 1	JC 2050
JNC	Address	Jumps to the address if carry flag is 0	JNC 2050
JZ	Address	Jumps to the address if zero flag is 1	JZ 2050
JNZ	Address	Jumps to the address if zero flag is 0	JNZ 2050
JPE	Address	Jumps to the address if parity flag is 1	JPE 2050
JPO	Address	Jumps to the address if parity flag is 0	JPO 2050
JM	Address	Jumps to the address if sign flag is 1	JM 2050
JP	Address	Jumps to the address if sign flag 0	JP 2050

2. Call Instructions – The call instruction transfers the program sequence to the memory address given in the operand. Before transferring, the address of the next instruction after CALL is pushed onto the stack. Call instructions are 2 types: Unconditional Call Instructions and Conditional Call Instructions.

(a) Unconditional Call Instructions: It transfers the program sequence to the memory address given in the operand.

CALL 16-address   Unconditionally calls

Example- CALL 2050

(b) Conditional Call Instructions: Only if the condition is satisfied, the instructions executes.

Opcode	Operand	Explanation	Example
CC	Address	Call if carry flag is 1	CC 2050
CNC	Address	Call if carry flag is 0	CNC 2050
CZ	Address	Calls if zero flag is 1	CZ 2050
CNZ	Address	Calls if zero flag is 0	CNZ 2050
CPE	Address	Calls if parity flag is 1	CPE 2050
CPO	Address	Calls if parity flag is 0	CPO 2050
CM	Address	Calls if sign flag is 1	CM 2050
CP	Address	Calls if sign flag is 0	CP 2050

3. Return Instructions – The return instruction transfers the program sequence from the subroutine to the calling program. Return instructions are 2 types: Unconditional Jump Instructions and Conditional Jump Instructions.

(a) Unconditional Return Instruction: The program sequence is transferred unconditionally from the subroutine to the calling program. RET   Return from the subroutine unconditionally

(b) Conditional Return Instruction: The program sequence is transferred unconditionally from the subroutine to the calling program only is the condition is satisfied.

Opcode	Operand	Explanation	Example
RC		Return from the subroutine if carry flag is 1	RC

RNC		Return from the subroutine if carry flag is 0	RNC
RZ		Return from the subroutine if zero flag is 1	RZ
RNZ		Return from the subroutine if zero flag is 0	RNZ
RPE		Return from the subroutine if parity flag is 1	RPE
RPO		Return from the subroutine if parity flag is 0	RPO
RM		Returns from the subroutine if sign flag is 1	RM
RP		Returns from the subroutine if sign flag is 0	RP

Machine control group

Opcode	Operand	Meaning	Explanation
NOP		No operation	No operation is performed, i.e., the instruction is fetched and decoded.
HLT		Halt and enter wait state	The CPU finishes executing the current instruction and stops further execution. An interrupt or reset is necessary to exit from the halt state.
DI		Disable interrupts	The interrupt enable flip-flop is reset and all the interrupts are disabled except TRAP.
EI		Enable interrupts	The interrupt enable flip-flop is set and all the interrupts are enabled.
RIM		Read interrupt mask	This instruction is used to read the status of interrupts 7.5, 6.5, 5.5 and read serial data input bit.
SIM		Set interrupt mask	This instruction is used to implement the interrupts 7.5, 6.5, 5.5, and serial data output.

Addressing modes

The term addressing modes refers to the way in which the operand of an instruction is specified.

Types of addressing modes –

In 8085 microprocessor there are 5 types of addressing modes:

Immediate Addressing Mode –

In immediate addressing mode the source operand is always data. If the data is 8-bit, then the instruction will be of 2 bytes, if the data is of 16-bit then the instruction will be of 3 bytes.

Examples:

MVI B,45 (move the data 45H immediately to register B)

LXI H,3050 (load the H-L pair with the operand 3050H immediately)

JMP address (jump to the operand address immediately)

Register Addressing Mode –

In register addressing mode, the data to be operated is available inside the register(s) and register(s) is(are) operands. Therefore, the operation is performed within various registers of the microprocessor.

Examples:

MOV A, B (move the contents of register B to register A)

ADD B (add contents of registers A and B and store the result in register A) INR A (increment the contents of register A by one)

Direct Addressing Mode –

In direct addressing mode, the data to be operated is available inside a memory location and that memory location is directly specified as an operand. The operand is directly available in the instruction itself.

Examples:

LDA 2050 (load the contents of memory location into accumulator A)

LHLD address (load contents of 16-bit memory location into H-L register pair)

IN 35 (read the data from port whose address is 35)

Register Indirect Addressing Mode –

In register indirect addressing mode, the data to be operated is available inside a memory location and that memory location is indirectly specified by a register pair.

Examples:

MOV A, M (move the contents of the memory location pointed by the H-L pair to the accumulator)

LDAX B (move contents of B-C register to the accumulator)

LHLD 9570 (load immediate the H-L pair with the data of the location 9570)

Implied/Implicit Addressing Mode –

In implied/implicit addressing mode the operand is hidden and the data to be operated is available in the instruction itself. Examples:

CMA (finds and stores the 1's complement of the contents of accumulator A in A)

RRC (rotate accumulator A right by one bit)

RLC (rotate accumulator A left by one bit)

Instruction cycle of 8085 Microprocessor

Time required to execute and fetch an entire instruction is called instruction cycle. It consists:

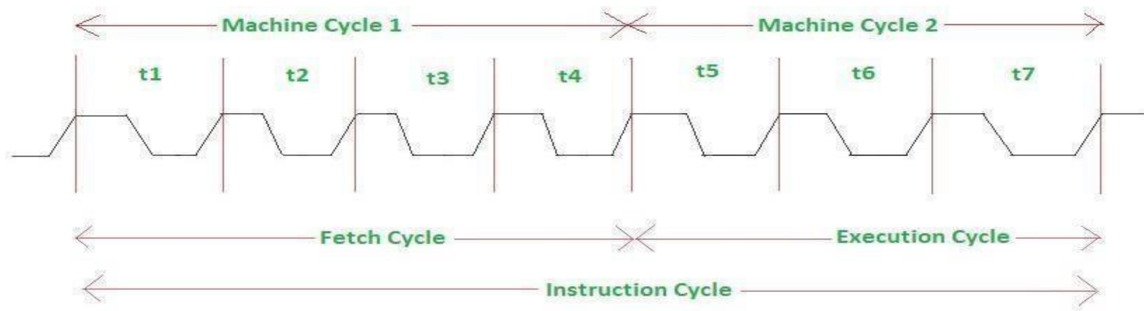
Fetch cycle – The next instruction is fetched by the address stored in program counter (PC) and then stored in the instruction register.

Decode instruction – Decoder interprets the encoded instruction from instruction register.

Execution cycle – consists memory read (MR), memory write (MW), input output read (IOR) and input output write (IOW)

The time required by the microprocessor to complete an operation of accessing memory or input/output devices is called **machine cycle**. One time period of frequency of microprocessor is called **t-state**. A t-state is measured from the falling edge of one clock pulse to the falling edge of the next clock pulse.

Fetch cycle takes four t-states and execution cycle takes three t-states.

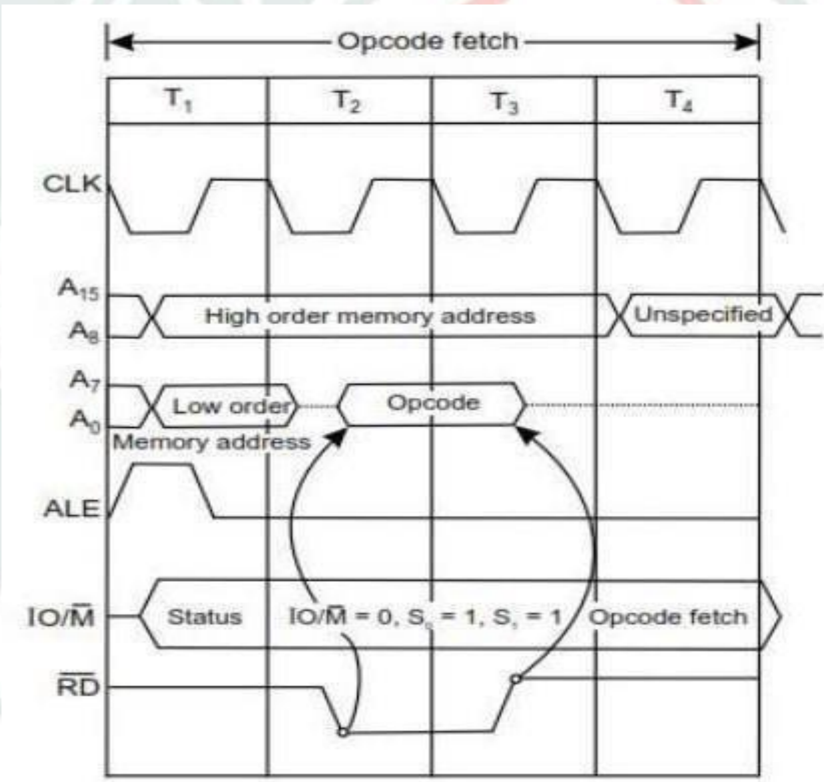


Instruction cycle in 8085 microprocessor

Timing diagram

Timing Diagram is a graphical representation. It represents the execution time taken by each instruction in a graphical format. The execution time is represented in T-states.

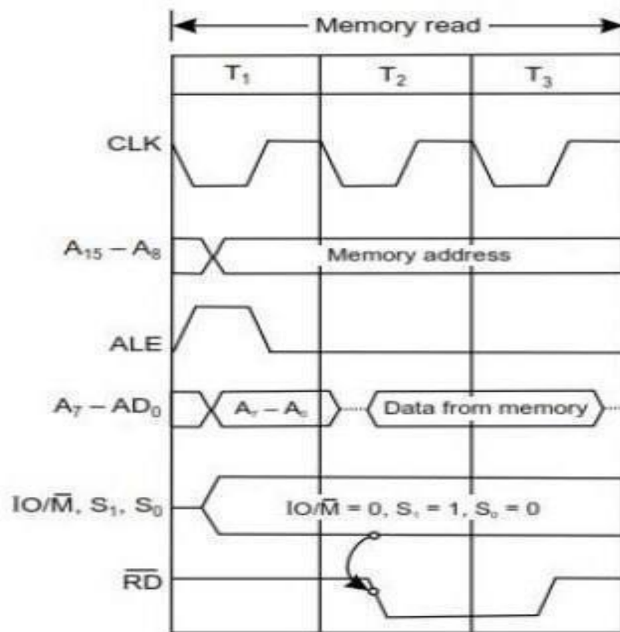
Opcode fetch cycle



- Each instruction of the processor has one byte opcode.
- The opcodes are stored in memory. So, the processor executes the opcode fetch machine cycle to fetch the opcode from memory.
- Hence, every instruction starts with opcode fetch machine cycle.
- The time taken by the processor to execute the opcode fetch cycle is 4T.
- In this time, the first, 3 T-states are used for fetching the opcode from memory and the remaining T-states are used for internal operations by the processor.

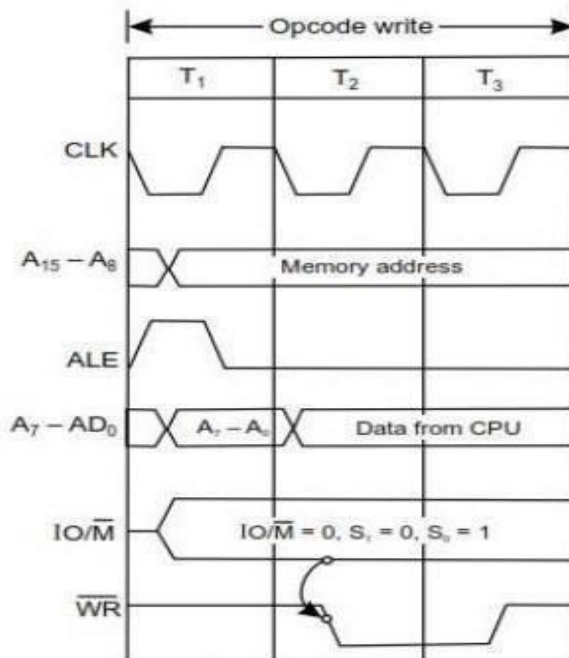
Memory read cycle

- The memory read machine cycle is executed by the processor to read a data byte from memory.
- The processor takes 3T states to execute this cycle.
- The instructions which have more than one byte word size will use the machine cycle after the opcode fetch machine cycle.



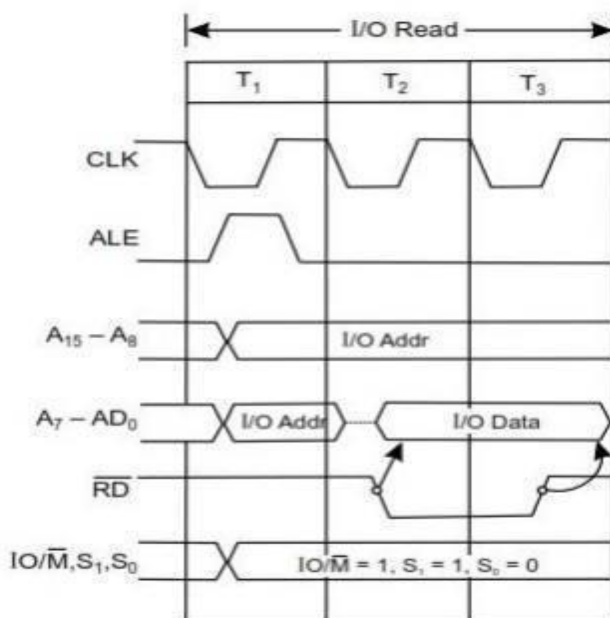
Memory write cycle

- The memory write machine cycle is executed by the processor to write a data byte in a memory location.
- The processor takes, 3T states to execute this machine cycle.



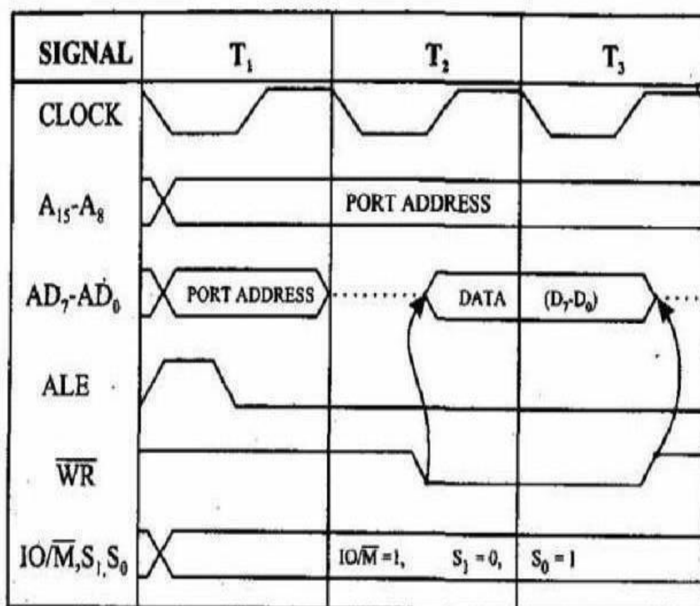
I/O read cycle

- The I/O Read cycle is executed by the processor to read a data byte from I/O port or from the peripheral, which is I/O, mapped in the system.
- The processor takes 3T states to execute this machine cycle.
- The IN instruction uses this machine cycle during the execution.



I/O write cycle

- The I/O Read cycle is executed by the processor to write a data byte from system to I/O port or peripheral, which is I/O mapped.
- The processor takes 3T states to execute this machine cycle.
- The OUT instruction uses this machine cycle during the execution.

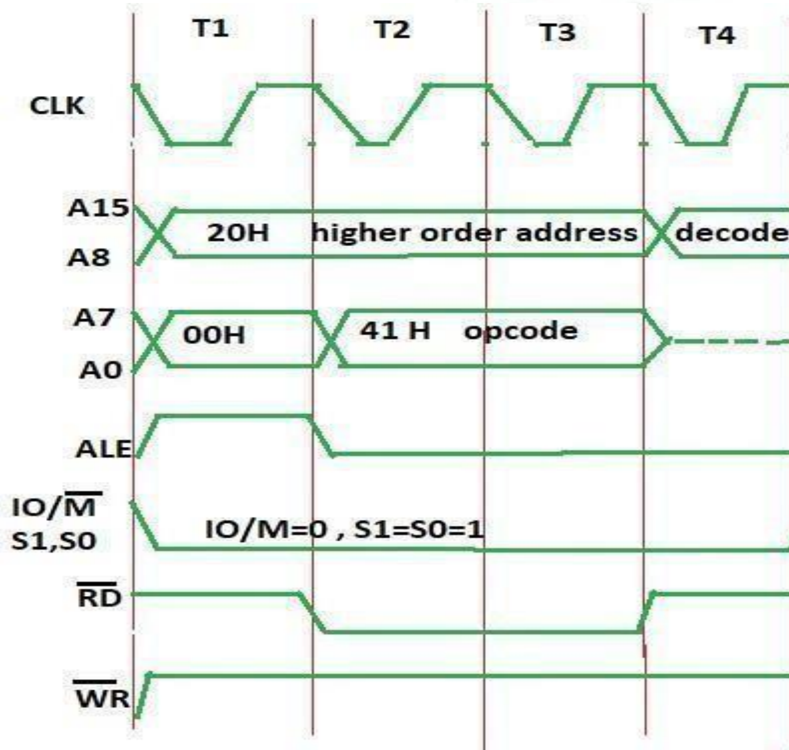


Example-1

The instruction MOV B, C is of 1 byte; therefore, the complete instruction will be stored in a single memory address.

2000 MOV B,C

Only opcode fetching is required for this instruction and thus we need 4 T states for the timing diagram. For the opcode fetch the IO/M (low active) = 0, S1 = 1 and S0 = 1.



In Opcode fetch (t1-t4 T-states):

1. 00 – lower bit of address where opcode is stored, i.e., 00 2.
20 – higher bit of address where opcode is stored, i.e., 20.
3. ALE – provides signal for multiplexed address and data bus. Only in t1 it used as address bus to fetch lower bit of address otherwise it will be used as data bus.
4. RD (low active) – signal is 1 in t1 & t4 as no data is read by microprocessor. Signal is 0 in t2 & t3 because here the data is read by microprocessor.
5. WR (low active) – signal is 1 throughout, no data is written by microprocessor.
6. IO/M (low active) – signal is 1 in throughout because the operation is performing on memory.
7. S0 and S1 – both are 1 in case of opcode fetching.

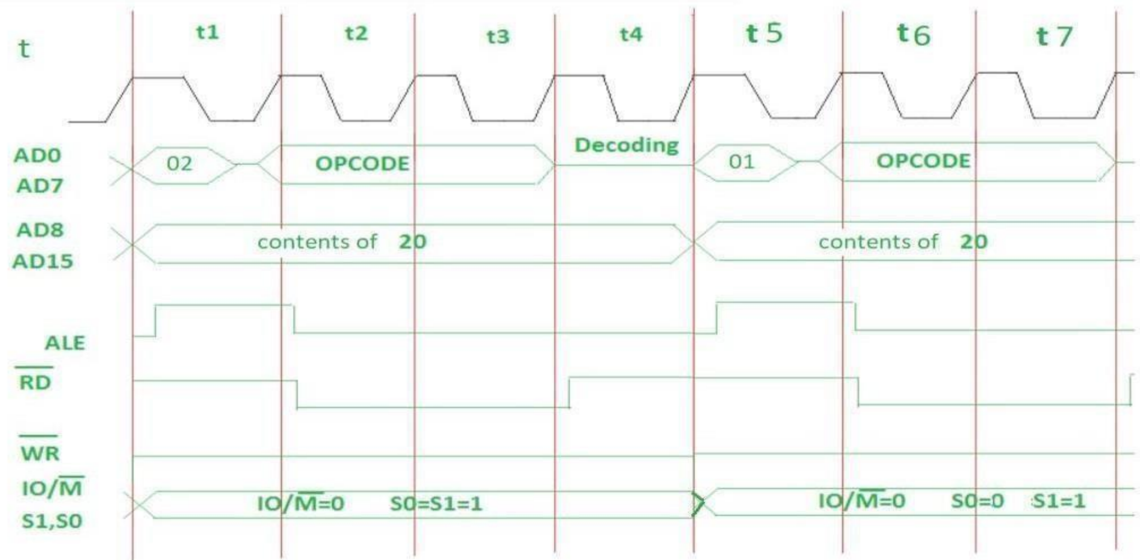
Example-2

MVI B, 45

2000: Opcode

2001: 45

- The opcode fetch will be same in all the instructions.
- Only the read instruction of the opcode needs to be added in the successive T states.
- For the opcode fetch the IO/M (low active) = 0, S1 = 1 and S0 = 1. Also, 4 T states will be required to fetch the opcode from memory.
- For the opcode read the IO/M (low active) = 0, S1 = 1 and S0 = 0. Also, only 3 T states will be required to read data from memory.



In Opcode fetch (t1-t4 T-states) –

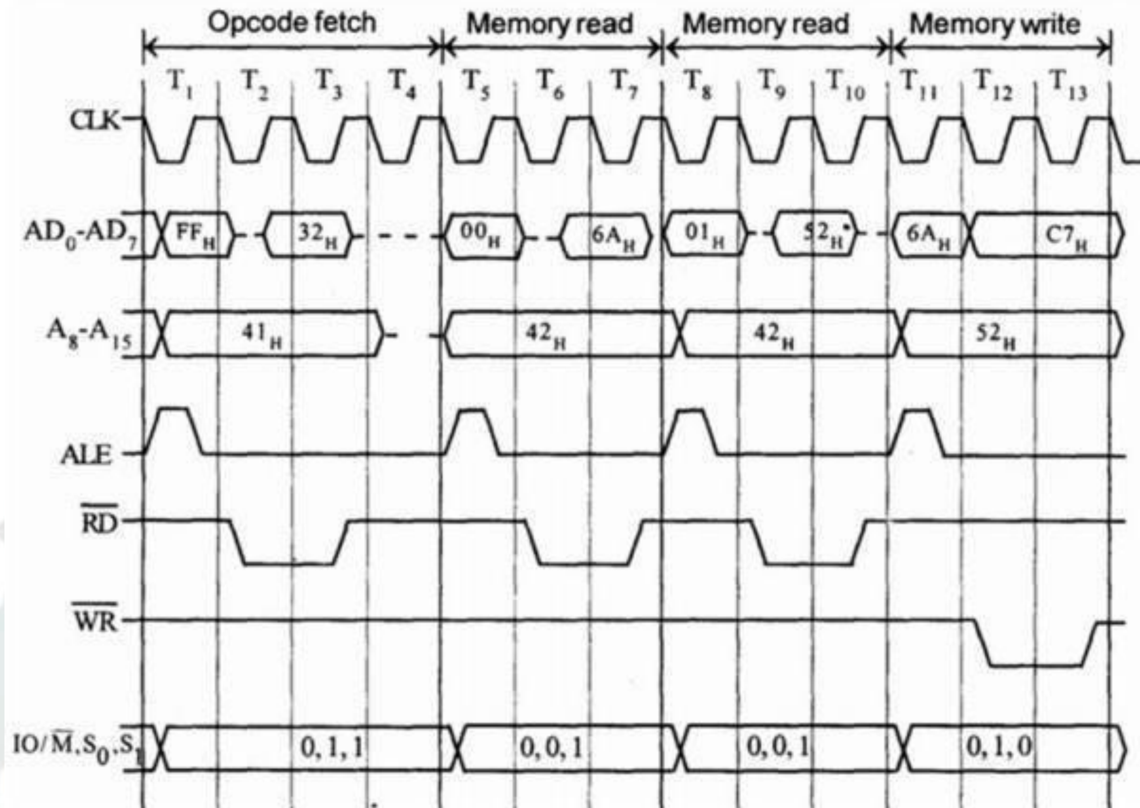
1. 00 – lower bit of address where opcode is stored.
2. 20 – higher bit of address where opcode is stored.
3. ALE – Provides signal for multiplexed address and data bus. Only in t1 it used as address bus to fetch lower bit of address otherwise it will be used as data bus.
4. RD (low active) – Signal is 1 in t1 & t4, no data is read by microprocessor. Signal is 0 in t2 & t3, data is read by microprocessor.
5. WR (low active) – Signal is 1 throughout, no data is written by microprocessor.
6. IO/M (low active) – Signal is 0 in throughout, operation is performing on memory.
7. S0 and S1 – Signal is 1 in t1 to t4 states, as to fetch the opcode from the memory.

In Opcode read (t5-t7 T-states) –

1. 01 – lower bit of address where data is stored.
2. 320 – higher bit of address where data is stored.
3. ALE – Provides signal for multiplexed address and data bus. Only in t5 it used as address bus to fetch lower bit of address otherwise it will be used as data bus.
4. RD (low active) – Signal is 1 in t5 as no data is read by microprocessor. Signal is 0 in t6 & t7 as data is read by microprocessor.
5. WR (low active) – Signal is 1 throughout, no data is written by microprocessor.
6. IO/M (low active) – Signal is 0 in throughout, operation is performing on memory.
7. S0 – Signal is 0 in throughout, operation is performing on memory to read data 45.
8. S1 – Signal is 1 throughout, operation is performing on memory to read data 45.

Example-3

41FF STA 526AH



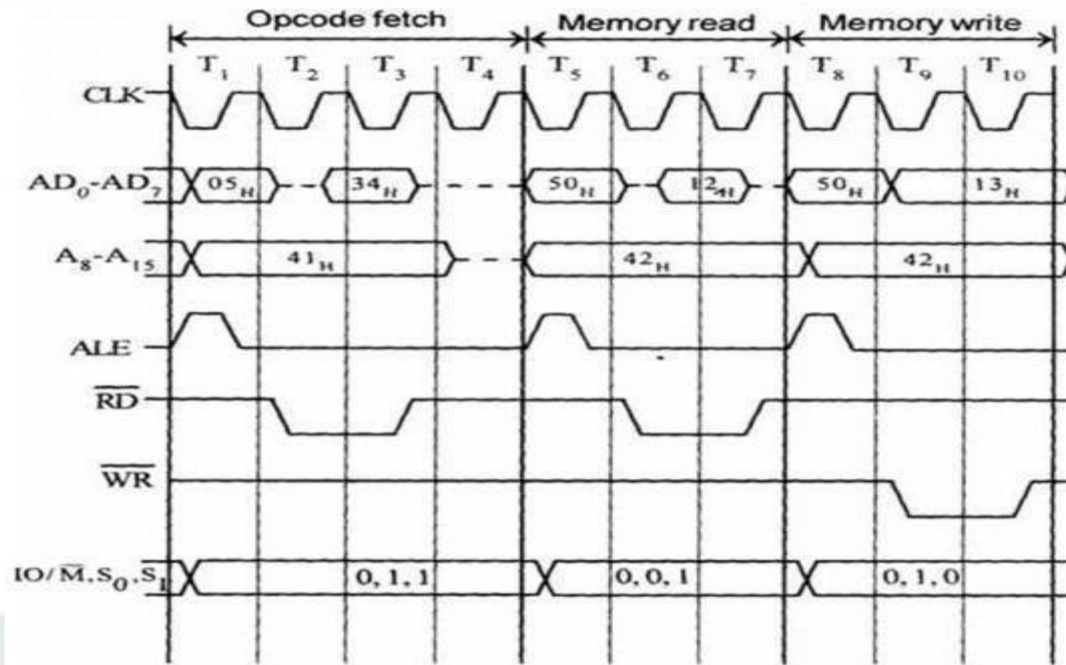
 STA means Store Accumulator -The contents of the accumulator is stored in the specified address (526A).

- The opcode of the STA instruction is said to be 32H. It is fetched from the memory 41FFH
- Then the lower order memory address is read (6A). - Memory Read Machine Cycle
- Read the higher order memory address (52).- Memory Read Machine Cycle
- The combination of both the addresses are considered and the content from accumulator is written in 526A. - Memory Write Machine Cycle
- Assume the memory address for the instruction and let the content of accumulator is C7H. So, C7H from accumulator is now stored in 526A.

Example-4

4105 INR M

- Fetching the Opcode 34H from the memory 4105H. (OF cycle)
- Let the memory address (M) be 4250H. (MR cycle -To read Memory address and data)
   Let the content of that memory is 12H.
- Increment the memory content from 12H to 13H. (MW machine cycle)



Counter and time delay

When the delay subroutine is executed, the microprocessor does not execute other tasks. For the delay we are using the instruction execution times. executing some instructions in a loop, the delay is generated. There are some methods of generating delays. These methods are as follows.

- Using NOP instructions
- Using 8-bit register as counter
- Using 16-bit register pair as counter. **Using NOT instructions:**
- One of the main usage of NOP instruction is in delay generation.
- The NOP instruction is taking four clock pulses to be fetching, decoding and executing.
- In the 8085 MPU the internal clock frequency is 3MHz.
- So, from that we can easily determine that each clock period is 1/3 of a microsecond.
- So, the NOP will be executed in 1/3 * 4 = 1.333µs. **Using 8-bit register as counter:**
- Counter is another approach to generate a time delay.
- In this case the program size is smaller.
- So, in this approach we can generate more time delay in less space.
- The following program will demonstrate the time delay using 8-bit counter. MVI B,FFH

LOOP: DCR B

JNZ LOOP

RET

- Here the first instruction will be executed once, it will take 7 T-states.
- DCR C instruction takes 4 T-states.
- This will be executed 255 (FF) times.
- The JNZ instruction takes 10 T-states when it jumps (It jumps 254 times), otherwise it will take 7 T-States.
- And the RET instruction takes 10 T-States.
  7 + ((4*255) + (10*254)) + 7 + 10 = 3584.
- So, the time delay will be 3584 * 1/3µs = 1194.66µs.

- So, when we need some small delay, then we can use this technique with some other values in the place of FF.

This technique can also be done using some nested loops to get larger delays. The following code is showing how we can get some delay with one loop into some other loops.

```
MVI   B,FFH
L1: MVI C,FFH
L2: DCR C
    JNZ L2
    DCR B
    JNZ L1
    RET
```

From this block, if we calculate the delay, it will be nearly 305μs delay. It extends the time of delay.

Using 16-bit register-pair as counter:

- Instead of using 8-bit counter, we can do that kind of task using 16-bit register pair.
- Using this method more time delay can be generated.
- This method can be used to get more than 0.5 seconds delay.

Program	Time (T-States)
LXI B,FFFFH	10
LOOP: DCX B	6
MOV A,B	4
ORA C	4
JNZ LOOP	10 (For Jump), 7(Skip)
RET	10

From that table, if we calculate the time delay:

10 + (6 + 4 + 4 + 10) * 65535H - 3 + 10 = 17 + 24 * 65535H = 1572857.
So, the time delay will be 1572857 * 1/3μs = 0.52428s. Here we are getting nearly 0.5s delay.

Assembly language program

Example-1

Write an assembly language program to add two 8-bit numbers 45H and 32H in 8085 Microprocessor and store the result in 2050H. The starting address of the program is taken as 2000.

Program address	Mnemonics	Operands	comments
2000	MVI	A,45	Load 1st data 45H in ACC
2002	MVI	B,32	Load 2nd data 32H in B
2004	ADD	B	A+B=A
2005	STA	2050	Store the result in 2050H
2008	HLT		Stop the program

O/P address **Result**

2050H 77H

Example-2

Write an ALP to add 2 8-bit numbers stored in memory location 2050H and 2051H. Result can be 8/16 bit and store it in 2052H and 2053H.

Program address	**Label**	**Mnemonics**	**Operands**	**Comments**
2000		MVI	C,00	Initialize the carry
2002		LXI	H,2050	Get the 1st data
2005		MOV	A,M	Load 1st data in ACC
2006		INX	H	Get 2nd data
2007		ADD	M	Add both data
2008		JNC	LOOP	If no carry, jump to LOOP
200B		INR	C	If carry, increment register C
200C	LOOP	STA	2052	Store the sum in 2052
2010		MOV	A,C	Move carry to ACC
2011		STA	2053	Store carry in 5053
2014		HLT		Stop the program

Without carry

I/P address	**Data**
2050	53
2051	27

With carry

I/P address	**Data**
2050	D9
2051	62

O/P address	Result	O/P address	Result
2052	7A	2053	3B
2053	00	2053	01

Basic Interfacing concept

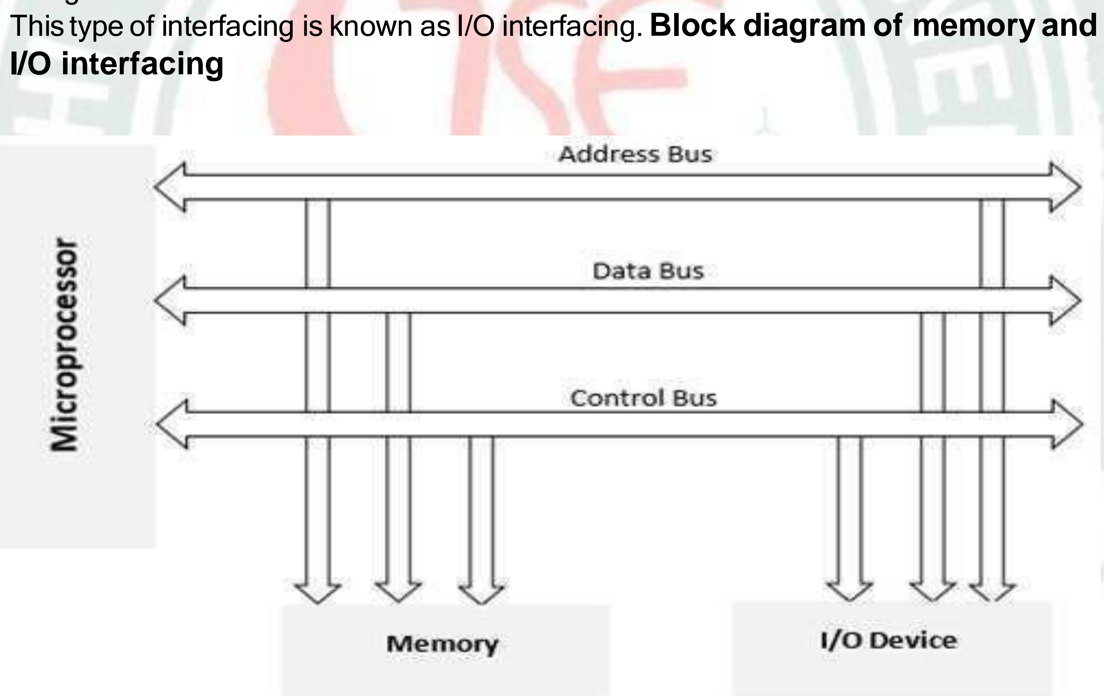
Interface is the path for communication between two components. Interfacing is of two types, memory interfacing and I/O interfacing.

Memory interfacing

- Memory interfacing is used to provide more memory space to accommodate complex programs for more complicated systems.
- Types of memories which are most commonly used to interface with 8085 are RAM, ROM, and EEPROM.
- 8085 can access 64kB of external memory.
- It can be explained as- total number of address lines in 8085 are 16, therefore it can access 2^16 = 65535 locations i.e., 64kB

I/O interfacing

- There are various communication devices like the keyboard, mouse, printer, etc.
- So, we need to interface the keyboard and other devices with the microprocessor by using latches and buffers.
- This type of interfacing is known as I/O interfacing. **Block diagram of memory and I/O interfacing**



Memory mapped I/O and I/O mapped I/O In
Memory Mapped Input Output −

- We allocate a memory address to an Input-Output device.
- Any instructions related to memory can be accessed by this Input-Output device. The Input-Output device data are also given to the Arithmetic Logical Unit.

Input-Output Mapped Input Output −

- We give an Input-Output address to an Input-Output device
- Only IN and OUT instructions are accessed by such devices.

- The ALU operations are not directly applicable to such Input-Output data. So as a summary we can mention that −
- I/O is any general-purpose port used by processor/controller to handle peripherals connected to it.
- I/O mapped I/Os have a separate address space from the memory. So, total addressed capacity is the number of I/Os connected and a memory connected. Separate I/Orelated instructions are used to access I/Os. A separate signal is used for addressing an I/O device.
- Memory-mapped I/Os share the memory space with external memory. So, total addressed capacity is memory connected only. This is underutilisation of resources if your processor supports I/O-mapped I/O. In this case, instructions used to access I/Os are the same as that used for memory.
- Let's take an example of the 8085 processor. It has 16 address lines i.e., addressing capacity of 64 KB memory. It supports I/O-mapped I/Os. It can address up to 256 I/Os.
- If we connect I/Os to it an I/O-mapped I/O then, it can address 256 I/Os + 64 KB memory. And special instructions IN and OUT are used to access the peripherals. Here we fully utilize the addressing capacity of the processor.
- If the peripherals are connected in memory mapped fashion, then total devices it can address is only 64K. This is underutilisation of the resource. And only memoryaccessing instructions like MVI, MOV, LOAD, SAVE are used to access the I/O devices.

UNIT-5 Interfacing and support chips

8255 Programable Peripheral Interface (PPI)

- PPI 8255 is a general purpose programmable I/O device designed to interface the CPU with its outside world such as ADC, DAC, keyboard etc.
- We can program it according to the given condition. It can be used with almost any microprocessor.
- It consists of three 8-bit bidirectional I/O ports (24 I/O lines) which can be configured as per the requirement.

Ports of 8255A

8255A has three ports, i.e., PORT A, PORT B, and PORT C.

- Port A (PA0-PA7) contains one 8-bit output latch/buffer and one 8-bit input buffer.
- Port B (PB0-PB7) is similar to PORT A.
- Port C can be split into two parts, i.e., PORT C lower (PC0-PC3) and PORT C upper (PC7-PC4) by the control word.

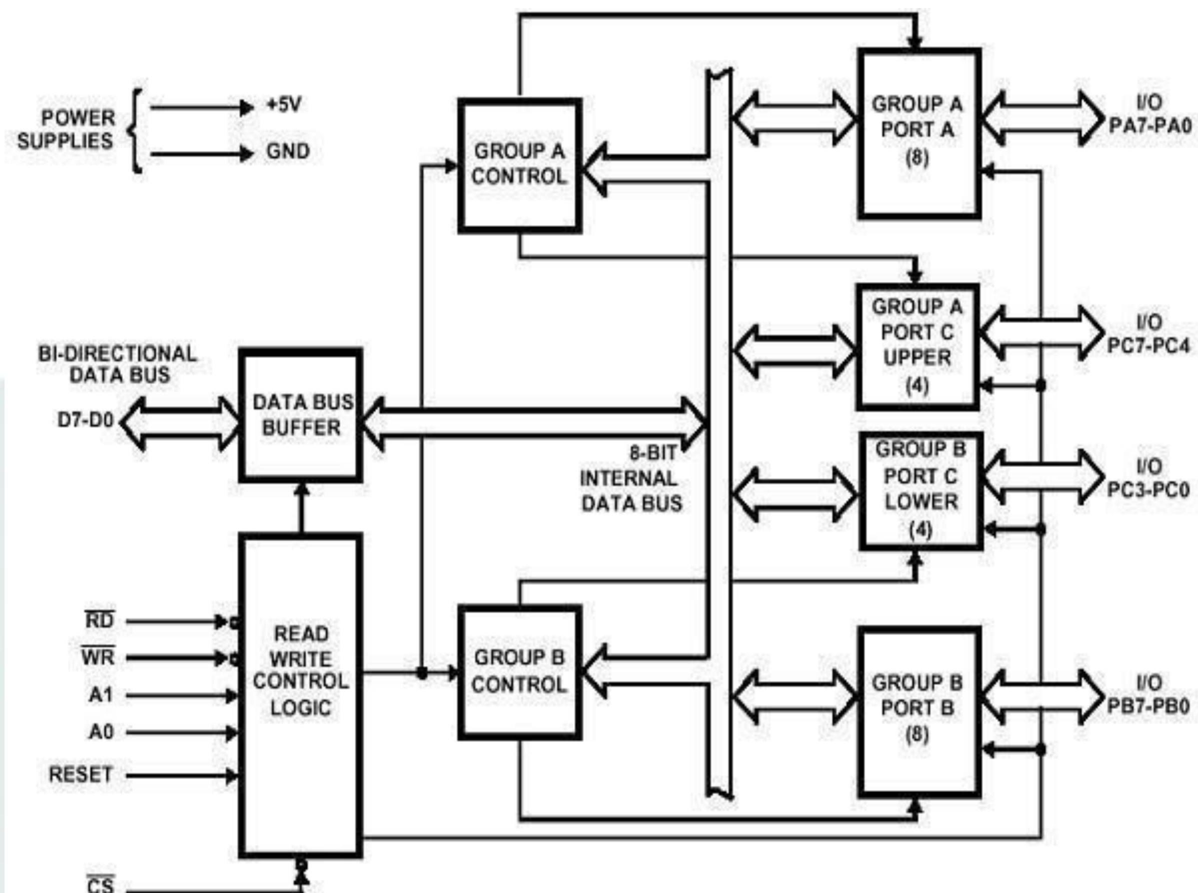
These three ports are further divided into two groups, i.e., Group A includes PORT A and upper PORT C. Group B includes PORT B and lower PORT C. These two groups can be programmed in three different modes, i.e., the first mode is named as mode 0, the second mode is named as Mode 1 and the third mode is named as Mode 2.

Features of 8255A

The prominent features of 8255A are as follows − It consists of 3 8-bit I/O ports i.e., PA, PB, and PC.

- Address/data bus must be externally demultiplexed.
- It is TTL compatible.
- It has improved DC driving capability.

8255 Architecture



Control group A

Control group A consist of port A and port C upper.

Control group B

Control group B consists of port C lower and port B.

Data Bus Buffer

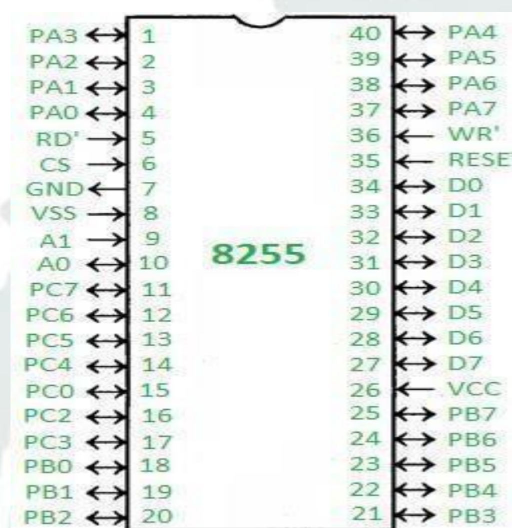
- It is a tri-state 8-bit buffer, which is used to interface the microprocessor to the system data bus.
- Data is transmitted or received by the buffer as per the instructions by the CPU.
- Control words and status information is also transferred using this bus.

Read/Write Control Logic

- This block is responsible for controlling the internal/external transfer of data/control/status word.
- It accepts the input from the CPU address and control buses, and in turn issues command to both the control groups.
- Depending upon the value if CS', A1 and A0 we can select different ports in different modes as input-output function or BSR.
- This is done by writing a suitable word in control register (control word D0-D7).

CS’	A1	A0	Selection
0	0	0	PORT A
0	0	1	PORT B
0	1	0	PORT C
0	1	1	Control Register
1	X	X	No Selection

Pin diagram



CS

It stands for Chip Select. A LOW on this input selects the chip and enables the communication between the 8255A and the CPU. It is connected to the decoded address, and A0 & A1 are connected to the microprocessor address lines.

WR

It stands for write. This control signal enables the write operation. When this signal goes low, the microprocessor writes into a selected I/O port or control register.

RESET

This is an active high signal. It clears the control register and sets all ports in the input mode

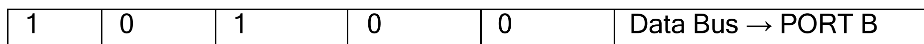
RD

It stands for Read. This control signal enables the Read operation. When the signal is low, the microprocessor reads the data from the selected I/O port of the 8255.

A0 and A1

These input signals work with RD, WR, and one of the control signals. Following is the table showing their various signals with their result.

A_1	A_0	RD	WR	CS	Result
0	0	0	1	0	**<u>Input Operation</u>** PORT A → Data Bus
0	1	0	1	0	PORT B → Data Bus
1	0	0	1	0	PORT C → Data Bus
0	0	1	0	0	**<u>Output Operation</u>** Data Bus → PORT A
0	1	1	0	0	Data Bus → PORT A

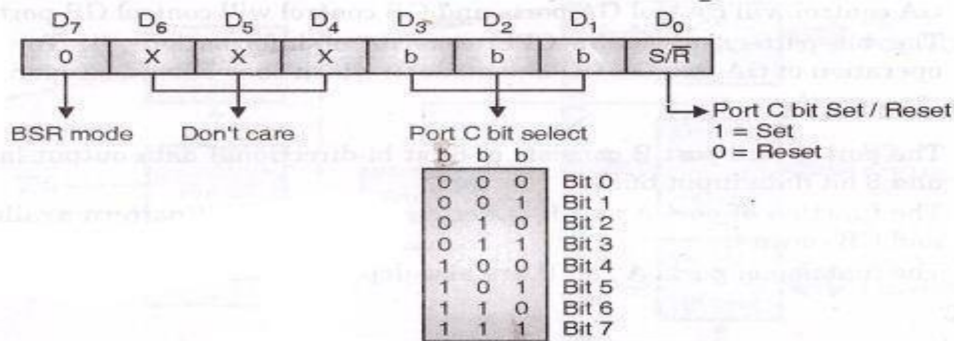


1	0	1	0	0	Data Bus → PORT B

Operating Modes

1. BSR (bit set-reset) mode-

If MSB of control word (D7) is 0, PPI works in BSR mode. In this mode only port C bits are used for set or reset.



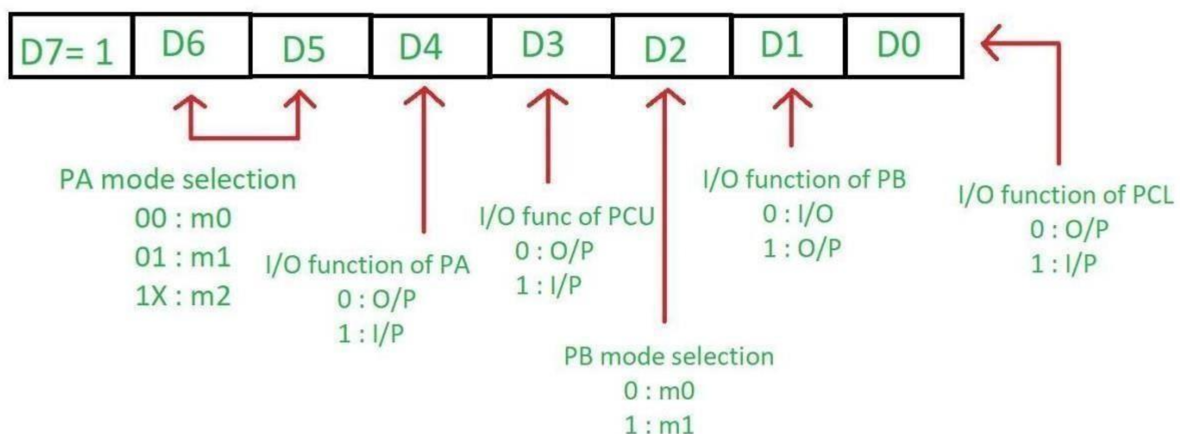
BSR control word format

2. I/O mode-

Mode 0 – In this mode, Port A and B is used as two 8-bit ports and Port C as two 4-bit ports. Each port can be programmed in either input mode or output mode where outputs are latched and inputs are not latched. Ports do not have interrupt capability.

Mode 1 – In this mode, Port A and B is used as 8-bit I/O ports. They can be configured as either input or output ports. Each port uses three lines from port C as handshake signals. Inputs and outputs are latched.

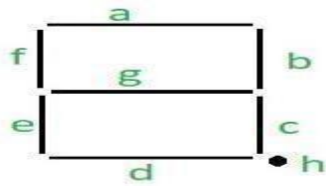
Mode 2 – In this mode, Port A can be configured as the bidirectional port and Port B either in Mode 0 or Mode 1. Port A uses five signals from Port C as handshake signals for data transfer. The remaining three signals from Port C can be used either as simple I/O or as handshake for port B.



Seven segment LED display

A seven-segment LED is a kind of LED (Light Emitting Diode) consisting of 7 small LEDs it usually comes with the microprocessor's as we commonly need to interface them with microprocessors like 8085.

Structure of Seven Segments LED:



- It can be used to represent numbers from 0 to 8 with a decimal point.
- We have eight segments in a Seven Segment LED display consisting of 7 segments which include '.'.
- The seven segments are denoted as "a, b, c, d, e, f, g, h" respectively, and '.' is represented by "h".

Interfacing Seven Segment Display with 8085:

We will see a program to Interfacing Seven Segment Display with 8085 using 8255.
Note logic needed for activation –
Common Anode – 0 will make an LED glow.
Common Cathode – 1 will make an LED glow.
Common Anode Method:
Here we are using a common anode display therefore 0 logic is needed to activate the segment. Suppose to display number 9 at the seven-segment display, therefore the segments F, G, B, A, C, and D have to be activated.
The instructions to execute it is given as,

```
    MVI  A,99
OUT 00
```

- First, we are storing the 99H in the accumulator i.e., 10010000 by using MVI instruction.
- By OUT instruction we are sending the data stored in the accumulator to the port 00H. Common Cathode Method:

Here we are using common cathode 1 logic is needed to activate the signal. Suppose to display number 9 at the seven-segment display, therefore the segments F, G, B, A, C, and D have to be activated.
The instructions to execute it is given as,

```
    MVI  A,6F
OUT 00
```

- First, we are storing the 6FH in the accumulator i.e., 01101111 by using MVI instruction.
- By OUT instruction we are sending the data stored in the accumulator to the port 00H.

Traffic light controller

The traffic lights are interfaced to Microprocessor system through buffer and ports of programmable peripheral Interface 8255. So the traffic lights can be automatically switched ON/OFF in desired sequence. The Interface board has been designed to work with parallel port of Microprocessor system.

Working Program

Design of a microprocessor system to control traffic lights. The traffic should be controlled in the following manner.

1) Allow traffic from W to E and E to W transition for 20 seconds.
2) Give transition period of 5 seconds (Yellow bulbs ON)
3) Allow traffic from N to 5 and 5 to N for 20 seconds
4) Give transition period of 5 seconds (Yellow bulbs ON) 5) Repeat the process.

Source Program:

```
            MVI A, 80H:            Initialize 8255, port A and port B
             OUT 83H (CR):         in output mode
START: MVI A, 09H
            OUT     80H     (PA):  Send data on PA to glow R1 and R2
MVI A, 24H
            OUT 81H (PB):                 Send data on PB to glow G3 and G4
            MVI C, 28H:            Load multiplier count (40ıo) for delay
            CALL DELAY:                 Call delay subroutine
            MVI A, 12H
           OUT (81H) PA:           Send data on Port A to glow Y1 and Y2
           OUT (81H) PB:           Send data on port B to glow Y3 and Y4
           MVI C, 0AH:             Load multiplier count (10ıo) for delay
           CALL: DELAY:                 Call delay subroutine
           MVI A, 24H
           OUT (80H) PA: MVI       Send data on port A to glow G1 and G2
           A, 09H
           OUT (81H) PB:           Send data on port B to glow R3 and R4
           MVI C, 28H:             Load multiplier count (40ıo) for delay
           CALL DELAY: MVI              Call delay subroutine
           A, 12H
           OUT PA:                 Send data on port A to glow Y1 and Y2
           OUT PB:                 Send data on port B to glow Y3 and Y4
           MVI C, 0AH:             Load multiplier count (10ıo) for delay
           CALL DELAY:                  Call delay subroutine
           JMP START
Delay Subroutine:
DELAY:  LXI D, Count:                     Load count to give 0.5 sec delay
BACK:  DCX D: MOV A, D                  Decrement counter

           ORA E:                       Check whether count is 0
           JNZ BACK:               If not zero, repeat
           DCR   C:                Check if multiplier zero, otherwise repeat
           JNZ DELAY
           RET:                    Return to main program
```

Square wave generator

- With 00H as i/p to DAC, analog o/p is -5V, and with FFH as i/p, analog o/p is +5V.
- I/P 00H and FFH at regular intervals generate square wave. The frequency can be varied by varying the time delay.

Algorithm

Initialize the control word of 8255 to operate in I/O mode for port A and B & C to operate in o/p mode.

Program

```
         MVI A,80
         OUT CWR      initialize the control word
LOOP: MVI A,00
```

```
        OUT  PA
        CALL DELAY
        MVI A,FF
        OUT  PA
        CALL DELAY
        JMP LOOP
DELAY: MVI C,85
BACK: DCR C
        JNZ BACK
        RET
```