QUICK GUIDE

COMPUTER LOGICAL ORGANIZATION - OVERVIEW

In the modern world of electronics, the term **Digital** is generally associated with a computer because the term **Digital** is derived from the way computers perform operation, by counting digits. For many years, the application of digital electronics was only in the computer system. But now-a-days, digital electronics is used in many other applications. Following are some of the examples in which **Digital electronics** is heavily used.

- Industrial process control
- Military system
- Television
- Communication system
- Medical equipment
- Radar
- Navigation

Signal

Signal can be defined as a physical quantity, which contains some information. It is a function of one or more than one independent variables. Signals are of two types.

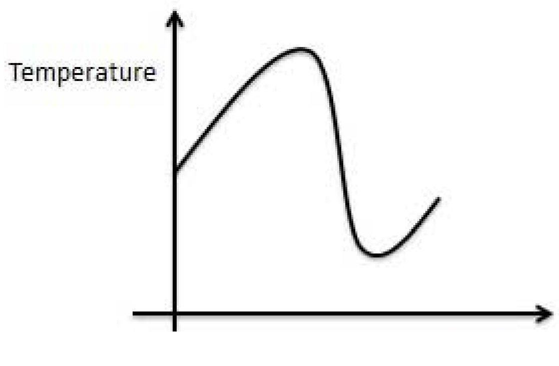
- Analog Signal
- Digital Signal

Analog Signal

An **analog signal** is defined as the signal having continuous values. Analog signal can have infinite number of different values. In real world scenario, most of the things observed in nature are analog. Examples of the analog signals are following.

- Temperature
- Pressure
- Distance
- Sound
- Voltage
- Current
- Power

Graphical representation of Analog Signal *Temperature*



Analog Signal

The circuits that process the analog signals are called as analog circuits or system. Examples of the analog system are following.

- Filter
- Amplifiers
- Television receiver
- Motor speed controller

Disadvantage of Analog Systems

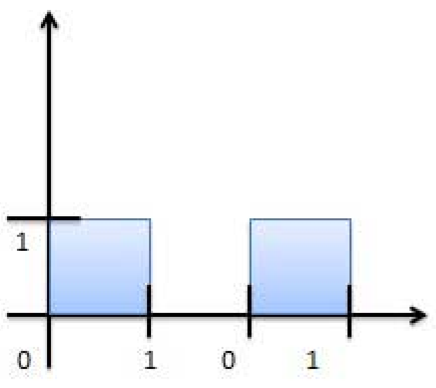
- Less accuracy
- Less versatility
- More noise effect
- More distortion
- More effect of weather

Digital Signal

A **digital signal** is defined as the signal which has only a finite number of distinct values. Digital signals are not continuous signals. In the digital electronic calculator, the input is given with the help of switches. This input is converted into electrical signal which have two discrete values or levels. One of these may be called low level and another is called high level. The signal will always be one of the two levels. This type of signal is called digital signal. Examples of the digital signal are following.

- Binary Signal
- Octal Signal
- Hexadecimal Signal

Graphical representation of the Digital Signal *Binary*



Digital Signal

The circuits that process the digital signals are called digital systems or digital circuits. Examples of the digital systems are following.

- Registers
- Flip-flop
- Counters

- Microprocessors

Advantage of Digital Systems

- More accuracy
- More versatility
- Less distortion
- Easy communicate
- Possible storage of information

Comparison of Analog and Digital Signal

S.N.	Analog Signal	Digital Signal
1	Analog signal has infinite values.	Digital signal has a finite number of values.
2	Analog signal has a continuous nature.	Digital signal has a discrete nature.
3	Analog signal is generated by transducers and signal generators.	Digital signal is generated by A to D converter.
4	Example of analog signal − sine wave, triangular waves.	Example of digital signal − binary signal.

DIGITAL NUMBER SYSTEM

A digital system can understand positional number system only where there are a few symbols called digits and these symbols represent different values depending on the position they occupy in the number.

A value of each digit in a number can be determined using

- The digit
- The position of the digit in the number
- The base of the number system $wherebaseisdefinedasthetotalnumberofdigitsavailableinthenumbersystem$.

Decimal Number System

The number system that we use in our day-to-day life is the decimal number system. Decimal number system has base 10 as it uses 10 digits from 0 to 9. In decimal number system, the successive positions to the left of the decimal point represents units, tens, hundreds, thousands and so on.

Each position represents a specific power of the base 10. For example, the decimal number 1234 consists of the digit 4 in the units position, 3 in the tens position, 2 in the hundreds position, and 1 in the thousands position, and its value can be written as

```
(1&times1000) &plus; (2&times100) &plus; (3&times10) &plus; (4&timesl)
(1&times10^3) &plus; (2&times10^2) &plus; (3&times10^1)  &plus; (4&timesl0^0)
1000 &plus; 200 &plus; 30 &plus; 1
1234
```

As a computer programmer or an IT professional, you should understand the following number systems which are frequently used in computers.

S.N.	Number System & Description
1	**Binary Number System** Base 2. Digits used: 0, 1
2	**Octal Number System** Base 8. Digits used: 0 to 7
3	**Hexa Decimal Number System** Base 16. Digits used: 0 to 9, Letters used: A- F

Binary Number System

Characteristics

- Uses two digits, 0 and 1.
- Also called base 2 number system
- Each position in a binary number represents a 0 power of the base 2. Example: 2^0
- Last position in a binary number represents an x power of the base 2. Example: 2^x where x represents the last position - 1.

Example

Binary Number: 10101_2

Calculating Decimal Equivalent −

Step	Binary Number	Decimal Number
Step 1	10101_2	$((1 \times 2^4)$ &plus; (0×2^3) &plus; (1×2^2) &plus; (0×2^1) &plus; $(1 \times 2^0))_{10}$
Step 2	10101_2	16 &plus; 0 &plus; 4 &plus; 0 &plus; 1$_{10}$
Step 3	10101_2	21_{10}

Note: 10101_2 is normally written as 10101.

Octal Number System

Characteristics

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

Example

Octal Number − 12570_8

Calculating Decimal Equivalent −

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

Note: 12570_8 is normally written as 12570.

Hexadecimal Number System

Characteristics

- Uses 10 digits and 6 letters, 0,1,2,3,4,5,6,7,8,9,A,B,C,D,E,F.
- Letters represents numbers starting from 10. A = 10, B = 11, C = 12, D = 13, E = 14, F = 15.
- Also called base 16 number system.
- Each position in a hexadecimal number represents a 0 power of the base 16. Example 16^0.
- Last position in a hexadecimal number represents an x power of the base 16. Example 16^x where x represents the last position - 1.

Example −

Hexadecimal Number: $19FDE_{16}$

Calculating Decimal Equivalent −

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

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

NUMBER SYSTEM CONVERSION

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

- Decimal to Other Base System
- Other Base System to Decimal
- Other Base System to Non-Decimal
- Shortcut method − Binary to Octal
- Shortcut method − Octal to Binary
- Shortcut method − Binary to Hexadecimal
- Shortcut method − Hexadecimal to Binary

Decimal to Other Base System

Steps

- **Step 1** − Divide the decimal number to be converted by the value of the new base.
- **Step 2** − Get the remainder from Step 1 as the rightmost digit *leastsignificantdigit* of new base number.
- **Step 3** − Divide the quotient of the previous divide by the new base.
- **Step 4** − Record the remainder from Step 3 as the next digit *totheleft* of the new base number.

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

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

Example −

Decimal Number: 29_{10}

Calculating Binary Equivalent −

Step	Operation	Result	Remainder
Step 1	29 / 2	14	1
Step 2	14 / 2	7	0
Step 3	7 / 2	3	1
Step 4	3 / 2	1	1
Step 5	1 / 2	0	1

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

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

Other Base System to Decimal System

Steps

- **Step 1** − Determine the column *positional* value of each digit *thisdependsonthepositionofthedigitandthebaseofthenumbersystem*.

- **Step 2** − Multiply the obtained column values $inStep1$ by the digits in the corresponding columns.
- **Step 3** − Sum the products calculated in Step 2. The total is the equivalent value in decimal.

Example

Binary Number − 11101_2

Calculating Decimal Equivalent −

Step	Binary Number	Decimal Number
Step 1	11101_2	$((1 \times 2^4)$ &plus; (1×2^3) &plus; (1×2^2) &plus; (0×2^1) &plus; $(1 \times 2^0))_{10}$
Step 2	11101_2	16 &plus; 8 &plus; 4 &plus; 0 &plus; 1$_{10}$
Step 3	11101_2	29_{10}

Binary Number − 11101_2 = Decimal Number − 29_{10}

Other Base System to Non-Decimal System

Steps

- **Step 1** − Convert the original number to a decimal number $base10$.
- **Step 2** − Convert the decimal number so obtained to the new base number.

Example

Octal Number − 25_8

Calculating Binary Equivalent −

Step 1 − Convert to Decimal

Step	Octal Number	Decimal Number
Step 1	25_8	$((2 \times 8^1)$ &plus; $(5 \times 8^0))_{10}$
Step 2	25_8	16 &plus; 5$_{10}$
Step 3	25_8	21_{10}

Octal Number − 25_8 = Decimal Number − 21_{10}

Step 2 − Convert Decimal to Binary

Step	Operation	Result	Remainder
Step 1	21 / 2	10	1
Step 2	10 / 2	5	0
Step 3	5 / 2	2	1

Step 4	2 / 2	1	0
Step 5	1 / 2	0	1

Decimal Number − 21_{10} = Binary Number − 10101_2

Octal Number − 25_8 = Binary Number − 10101_2

Shortcut method - Binary to Octal

Steps

- **Step 1** − Divide the binary digits into groups of three *startingfromtheright*.
- **Step 2** − Convert each group of three binary digits to one octal digit.

Example

Binary Number − 10101_2

Calculating Octal Equivalent −

Step	Binary Number	Octal Number
Step 1	10101_2	010 101
Step 2	10101_2	2_8 5_8
Step 3	10101_2	25_8

Binary Number − 10101_2 = Octal Number − 25_8

Shortcut method - Octal to Binary

Steps

- **Step 1** − Convert each octal digit to a 3 digit binary number *theoctaldigitsmaybetreatedasdecimalforthisconversion*.
- **Step 2** − Combine all the resulting binary groups *of3digitseach* into a single binary number.

Example

Octal Number − 25_8

Calculating Binary Equivalent −

Step	Octal Number	Binary Number
Step 1	25_8	2_{10} 5_{10}
Step 2	25_8	010_2 101_2
Step 3	25_8	010101_2

Octal Number − 25_8 = Binary Number − 10101_2

Shortcut method - Binary to Hexadecimal

Steps

- **Step 1** − Divide the binary digits into groups of four *startingfromtheright*.
- **Step 2** − Convert each group of four binary digits to one hexadecimal symbol.

Example

Binary Number − 10101_2

Calculating hexadecimal Equivalent −

Step	Binary Number	Hexadecimal Number
Step 1	10101_2	0001 0101
Step 2	10101_2	1_{10} 5_{10}
Step 3	10101_2	15_{16}

Binary Number − 10101_2 = Hexadecimal Number − 15_{16}

Shortcut method - Hexadecimal to Binary

Steps

- **Step 1** − Convert each hexadecimal digit to a 4 digit binary number *thehexadecimaldigitsmaybetreatedasdecimalforthisconversion*.
- **Step 2** − Combine all the resulting binary groups *of4digitseach* into a single binary number.

Example

Hexadecimal Number − 15_{16}

Calculating Binary Equivalent −

Step	Hexadecimal Number	Binary Number
Step 1	15_{16}	1_{10} 5_{10}
Step 2	15_{16}	0001_2 0101_2
Step 3	15_{16}	00010101_2

Hexadecimal Number − 15_{16} = Binary Number − 10101_2

BINARY CODES

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

Advantages of Binary Code

Following is the list of advantages that binary code offers.

- 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.

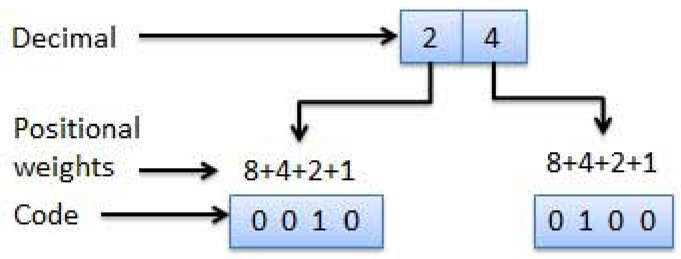
Classification of binary codes

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.

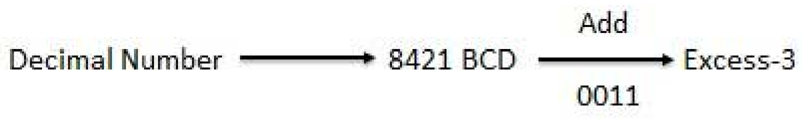


Non-Weighted Codes

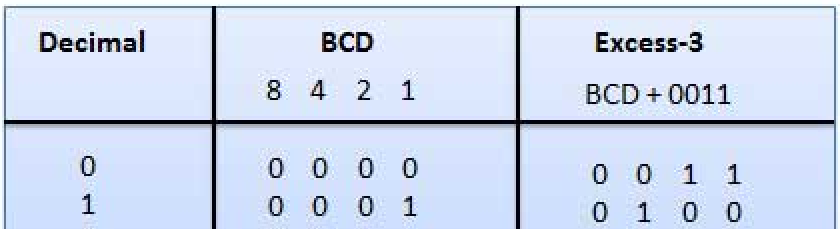
In this type of binary codes, the positional weights are not assigned. The examples of non-weighted codes are Excess-3 code and Gray code.

Excess-3 code

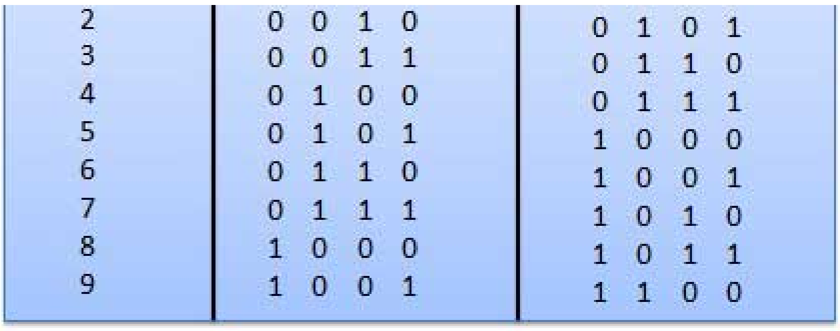
The Excess-3 code is also called as XS-3 code. It is non-weighted code used to express decimal numbers. The Excess-3 code words are derived from the 8421 BCD code words adding 0011_2 or 3_{10} to each code word in 8421. The excess-3 codes are obtained as follows −



Example



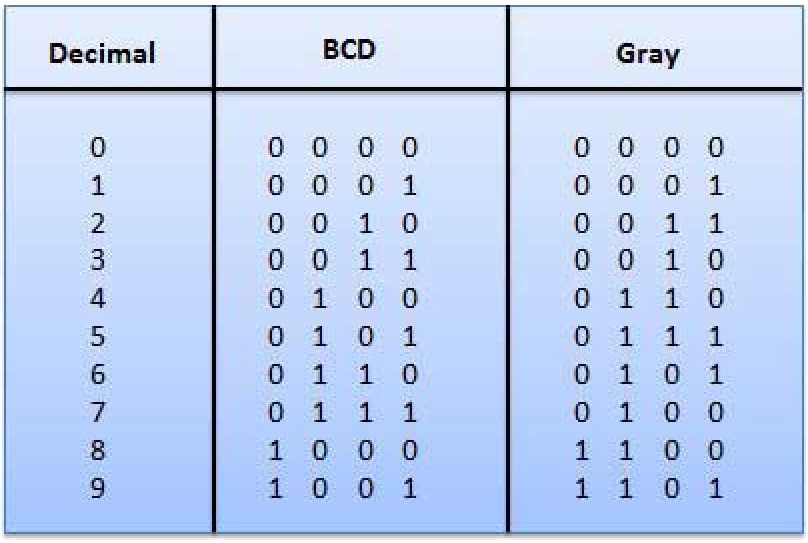
Decimal	BCD 8 4 2 1	Excess-3 BCD + 0011
0	0 0 0 0	0 0 1 1
1	0 0 0 1	0 1 0 0



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.



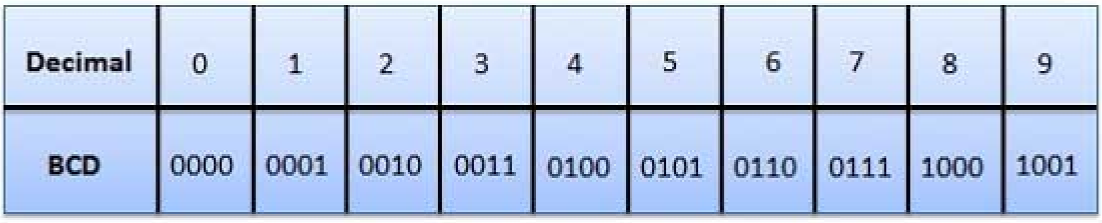
Decimal	BCD	Gray
0	0 0 0 0	0 0 0 0
1	0 0 0 1	0 0 0 1
2	0 0 1 0	0 0 1 1
3	0 0 1 1	0 0 1 0
4	0 1 0 0	0 1 1 0
5	0 1 0 1	0 1 1 1
6	0 1 1 0	0 1 0 1
7	0 1 1 1	0 1 0 0
8	1 0 0 0	1 1 0 0
9	1 0 0 1	1 1 0 1

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.

Binary Coded Decimal BCD code

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

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.

Alphanumeric codes

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

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

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

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

Error Codes

There are binary code techniques available to detect and correct data during data transmission.

Error Code	Description
Error Detection and Correction	Error detection and correction code techniques

CODES CONVERSION

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

- Binary to BCD Conversion
- BCD to Binary Conversion
- BCD to Excess-3
- Excess-3 to BCD

Binary to BCD Conversion

Steps

- **Step 1** -- Convert the binary number to decimal.
- **Step 2** -- Convert decimal number to BCD.

Example − convert 11101_2 to BCD.

Step 1 − Convert to Decimal

Binary Number − 11101_2

Calculating Decimal Equivalent −

Step	Binary Number	Decimal Number
Step 1	11101_2	$((1 \times 2^4)$ &plus; (1×2^3) &plus; (1×2^2) &plus; (0×2^1) &plus; $(1 \times 2^0))_{10}$
Step 2	11101_2	16 &plus; 8 &plus; 4 &plus; 0 &plus; 1$_{10}$
Step 3	11101_2	29_{10}

Binary Number − 11101_2 = Decimal Number − 29_{10}

Step 2 − Convert to BCD

Decimal Number − 29_{10}

Calculating BCD Equivalent. Convert each digit into groups of four binary digits equivalent.

Step	Decimal Number	Conversion
Step 1	29_{10}	0010_2 1001_2
Step 2	29_{10}	00101001_{BCD}

Result

```
(11101)₂ =  (00101001)BCD
```

BCD to Binary Conversion

Steps

- **Step 1** -- Convert the BCD number to decimal.
- **Step 2** -- Convert decimal to binary.

Example − convert 00101001_{BCD} to Binary.

Step 1 - Convert to BCD

BCD Number − 00101001_{BCD}

Calculating Decimal Equivalent. Convert each four digit into a group and get decimal equivalent for each group.

Step	BCD Number	Conversion
Step 1	00101001_{BCD}	0010_2 1001_2
Step 2	00101001_{BCD}	2_{10} 9_{10}

Step 3	00101001_{BCD}	29_{10}

BCD Number − 00101001_{BCD} = Decimal Number − 29_{10}

Step 2 - Convert to Binary

Used long division method for decimal to binary conversion.

Decimal Number − 29_{10}

Calculating Binary Equivalent −

Step	Operation	Result	Remainder
Step 1	29 / 2	14	1
Step 2	14 / 2	7	0
Step 3	7 / 2	3	1
Step 4	3 / 2	1	1
Step 5	1 / 2	0	1

As mentioned in Steps 2 and 4, the remainders have to be arranged in the reverse order so that the first remainder becomes the least significant digit LSD and the last remainder becomes the most significant digit MSD.

Decimal Number − 29_{10} = Binary Number − 11101_2

Result

```
(00101001)BCD = (11101)2
```

BCD to Excess-3

Steps

- **Step 1** -- Convert BCD to decimal.
- **Step 2** -- Add 3_{10} to this decimal number.
- **Step 3** -- Convert into binary to get excess-3 code.

Example − convert 1001_{BCD} to Excess-3.

Step 1 − Convert to decimal

1001_{BCD} = 9_{10}

Step 2 − Add 3 to decimal

9_{10} &plus; 3_{10} = 12_{10}

Step 3 − Convert to Excess-3

12_{10} = 1100_2

Result

```
(1001)BCD = (1100)XS-3
```

Excess-3 to BCD Conversion

Steps

- **Step 1** -- Subtract 0011_2 from each 4 bit of excess-3 digit to obtain the corresponding BCD code.

Example − convert 10011010_{XS-3} to BCD.

```
Given XS-3 number  = 1 0 0 1 1 0 1 0
Subtract (0011)2    = 0 0 1 1 0 0 1 1
                     --------------------
               BCD = 0 1 1 0   0 1 1 1
```

Result

```
(10011010)XS-3 = (01100111)BCD
```

COMPLEMENT ARITHMETIC

Complements are used in the digital computers in order to simplify the subtraction operation and for the logical manipulations. For each radix-r system *radixrrepresentsbaseofnumbersystem* there are two types of complements.

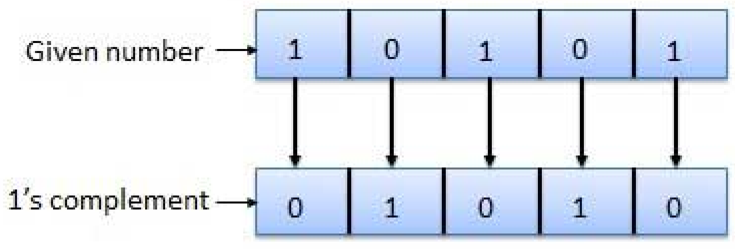
S.N.	Complement	Description
1	Radix Complement	The radix complement is referred to as the r's complement
2	Diminished Radix Complement	The diminished radix complement is referred to as the $r - 1$'s complement

Binary system complements

As the binary system has base r = 2. So the two types of complements for the binary system are 2's complement and 1's complement.

1's complement

The 1's complement of a number is found by changing all 1's to 0's and all 0's to 1's. This is called as taking complement or 1's complement. Example of 1's Complement is as follows.

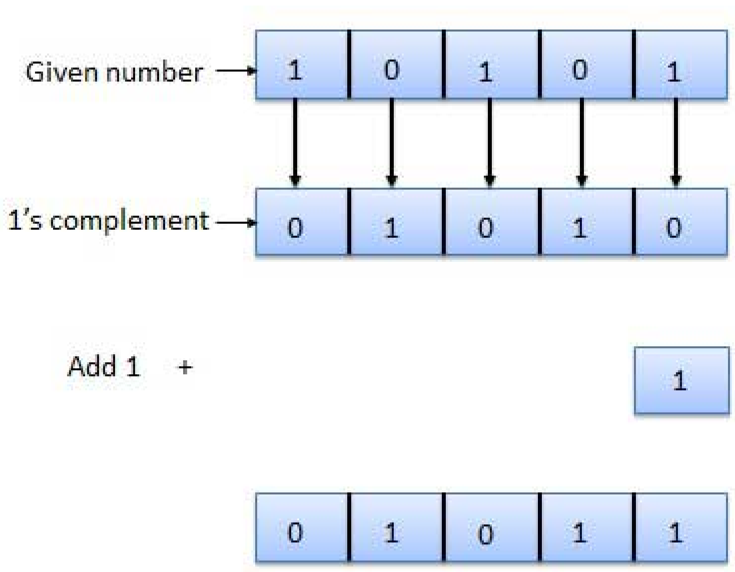


2's complement

The 2's complement of binary number is obtained by adding 1 to the Least Significant Bit *LSB* of 1's complement of the number.

2's complement = 1's complement &plus; 1

Example of 2's Complement is as follows.

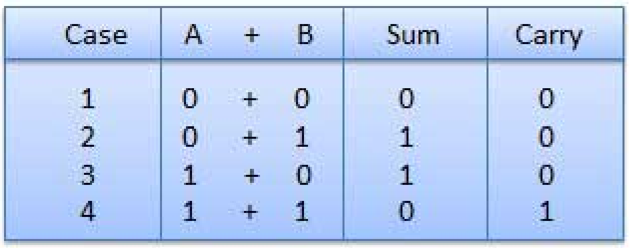


BINARY ARITHMETIC

Binary arithmetic is essential part of all the digital computers and many other digital system.

Binary Addition

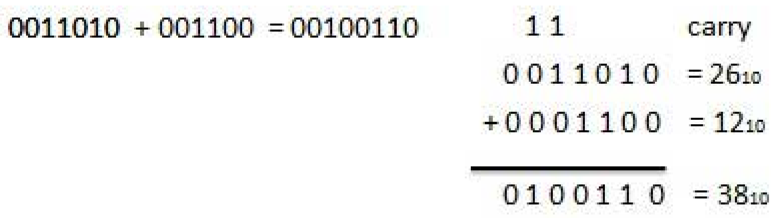
It is a key for binary subtraction, multiplication, division. There are four rules of binary addition.



Case	A	+	B	Sum	Carry
1	0	+	0	0	0
2	0	+	1	1	0
3	1	+	0	1	0
4	1	+	1	0	1

In fourth case, a binary addition is creating a sum of 1 &plus; 1 = 10 i.e. 0 is written in the given column and a carry of 1 over to the next column.

Example − Addition

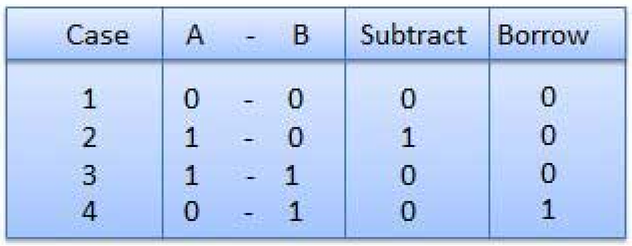


0011010 + 001100 = 00100110

```
   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

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



Case	A	-	B	Subtract	Borrow
1	0	-	0	0	0
2	1	-	0	1	0
3	1	-	1	0	0
4	0	-	1	0	1

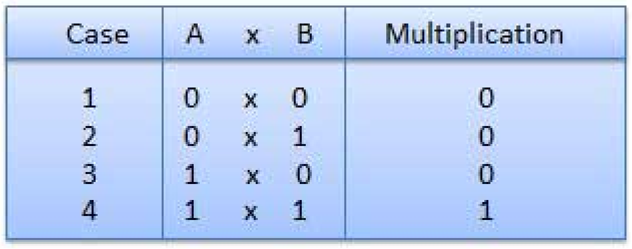
Example − Subtraction

0011010 - 001100 = 00001110

$$\begin{array}{rl} 1\;1\;\;\;\;\; & \text{borrow} \\ 0\,0\,\cancel{1}\,\cancel{1}\,0\,1\,0 & = 26_{10} \\ -0\,0\,0\,1\,1\,0\,0 & = 12_{10} \\ \hline 0\,0\,0\,1\,1\,1\,0 & = 14_{10} \end{array}$$

Binary Multiplication

Binary multiplication is similar to decimal multiplication. It is simpler than decimal multiplication because only 0s and 1s are involved. There are four rules of binary multiplication.



Case	A x B	Multiplication
1	0 x 0	0
2	0 x 1	0
3	1 x 0	0
4	1 x 1	1

Example − Multiplication

Example:

0011010 x 001100 = 100111000

$$\begin{array}{rl} 0\,0\,1\,1\,0\,1\,0 & = 26_{10} \\ \times 0\,0\,0\,1\,1\,0\,0 & = 12_{10} \\ \hline 0\,0\,0\,0\,0\,0\,0 & \\ 0\,0\,0\,0\,0\,0\,0\;\;\; & \\ 0\,0\,1\,1\,0\,1\,0\;\;\;\;\;\; & \\ 0\,0\,1\,1\,0\,1\,0\;\;\;\;\;\;\;\;\; & \\ \hline 0\,1\,0\,0\,1\,1\,1\,0\,0\,0 & = 312_{10} \end{array}$$

Binary Division

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

Example − Division

101010 / 000110 = 000111

$$\begin{array}{rl} 1\,1\,1 & = 7_{10} \\ 000110\,\overline{)\,\cancel{1}\overset{1}{0}\,1\,0\,1\,0} & = 42_{10} \\ -1\,1\,0 & = 6_{10} \\ \hline \cancel{1}\overset{1}{0}\,0\,1 & \\ -1\,1\,0 & \\ \hline 1\,1\,0 & \\ -1\,1\,0 & \\ \hline 0 & \end{array}$$

OCTAL ARITHMETIC

Octal Number System

Following are the characteristics of an octal number system.

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

Example

Octal Number – 12570_8

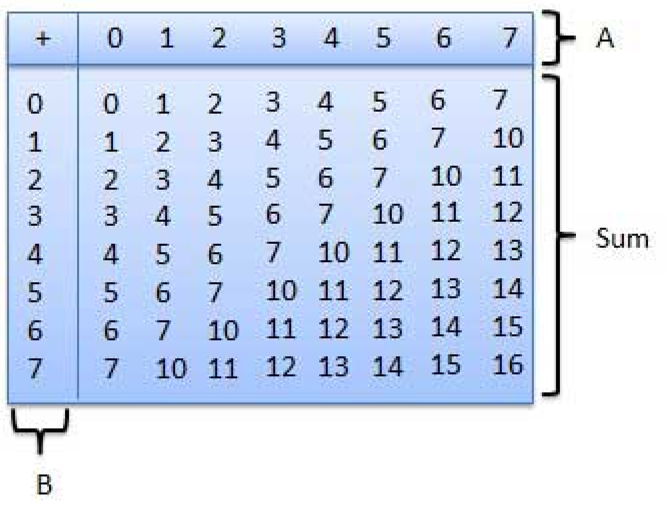
Calculating Decimal Equivalent –

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

Note – 12570_8 is normally written as 12570.

Octal Addition

Following octal addition table will help you to handle octal addition.

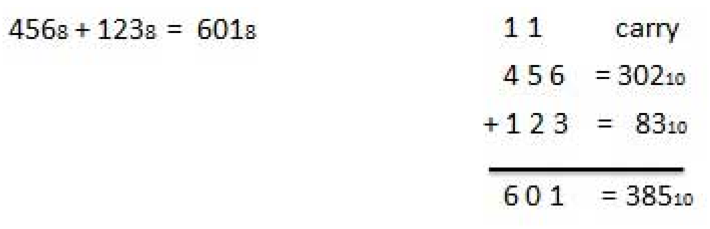


+	0	1	2	3	4	5	6	7
0	0	1	2	3	4	5	6	7
1	1	2	3	4	5	6	7	10
2	2	3	4	5	6	7	10	11
3	3	4	5	6	7	10	11	12
4	4	5	6	7	10	11	12	13
5	5	6	7	10	11	12	13	14
6	6	7	10	11	12	13	14	15
7	7	10	11	12	13	14	15	16

To use this table, simply follow the directions used in this example: Add 6_8 and 5_8. Locate 6 in the A column then locate the 5 in the B column. The point in 'sum' area where these two columns intersect is the 'sum' of two numbers.

```
6₈ &plus; 5₈ = 13₈.
```

Example – Addition



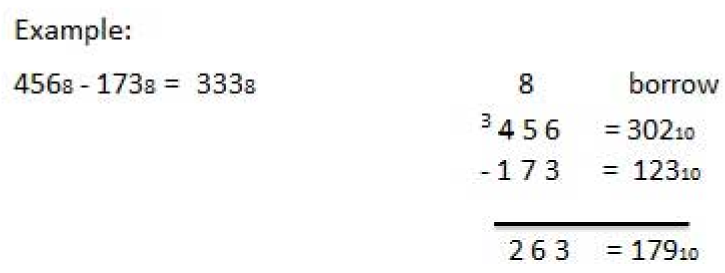
$456_8 + 123_8 = 601_8$

$$\begin{array}{rl} 1\,1 & \text{carry} \\ 4\,5\,6 & = 302_{10} \\ +1\,2\,3 & = 83_{10} \\ \hline 6\,0\,1 & = 385_{10} \end{array}$$

Octal Subtraction

The subtraction of octal numbers follows the same rules as the subtraction of numbers in any other number system. The only variation is in borrowed number. In the decimal system, you borrow a group of 10_{10}. In the binary system, you borrow a group of 2_{10}. In the octal system you borrow a group of 8_{10}.

Example − Subtraction



Example:

$456_8 - 173_8 = 333_8$

$$\begin{array}{rl} 8 & \text{borrow} \\ {}^{3}4\,5\,6 & = 302_{10} \\ -1\,7\,3 & = 123_{10} \\ \hline 2\,6\,3 & = 179_{10} \end{array}$$

HEXADECIMAL ARITHMETIC

Hexadecimal Number System

Following are the characteristics of a hexadecimal number system.

- Uses 10 digits and 6 letters, 0,1,2,3,4,5,6,7,8,9,A,B,C,D,E,F.
- Letters represents numbers starting from 10. A = 10, B = 11, C = 12, D = 13, E = 14, F = 15.
- Also called base 16 number system.
- Each position in a hexadecimal number represents a 0 power of the base 16. Example − 16^0
- Last position in a hexadecimal number represents an x power of the base 16. Example − 16^x where x represents the last position - 1.

Example

Hexadecimal Number − $19FDE_{16}$

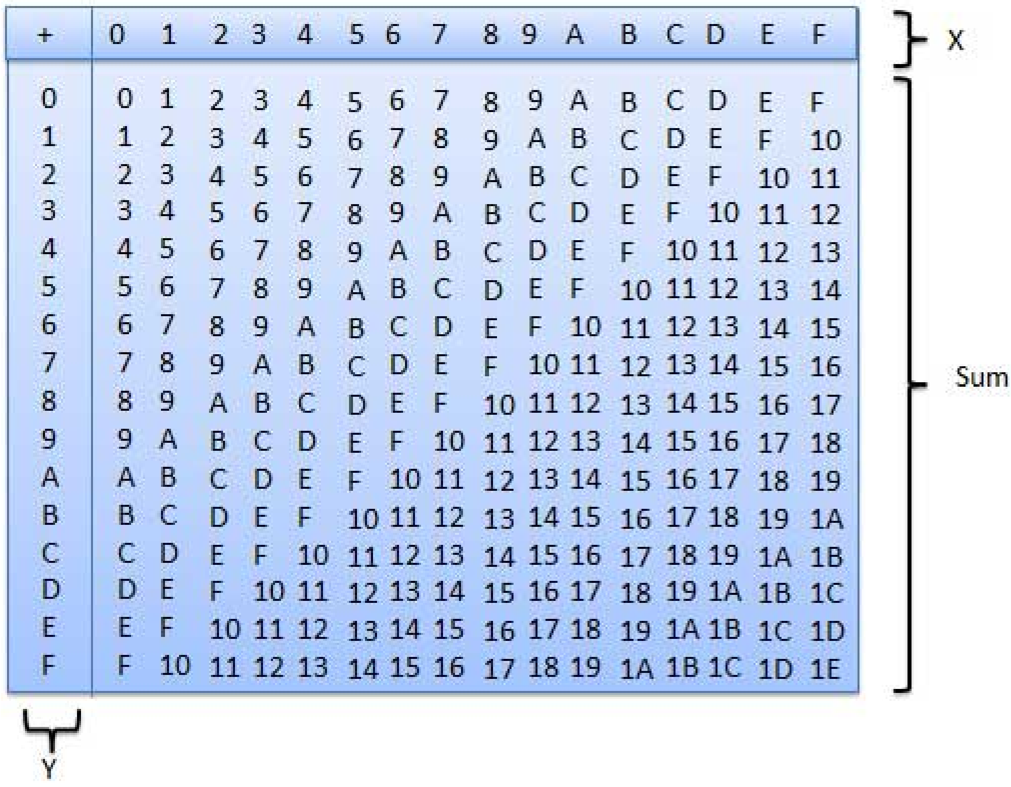
Calculating Decimal Equivalent −

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

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

Hexadecimal Addition

Following hexadecimal addition table will help you greatly to handle Hexadecimal addition.



+	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
1	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	10
2	2	3	4	5	6	7	8	9	A	B	C	D	E	F	10	11
3	3	4	5	6	7	8	9	A	B	C	D	E	F	10	11	12
4	4	5	6	7	8	9	A	B	C	D	E	F	10	11	12	13
5	5	6	7	8	9	A	B	C	D	E	F	10	11	12	13	14
6	6	7	8	9	A	B	C	D	E	F	10	11	12	13	14	15
7	7	8	9	A	B	C	D	E	F	10	11	12	13	14	15	16
8	8	9	A	B	C	D	E	F	10	11	12	13	14	15	16	17
9	9	A	B	C	D	E	F	10	11	12	13	14	15	16	17	18
A	A	B	C	D	E	F	10	11	12	13	14	15	16	17	18	19
B	B	C	D	E	F	10	11	12	13	14	15	16	17	18	19	1A
C	C	D	E	F	10	11	12	13	14	15	16	17	18	19	1A	1B
D	D	E	F	10	11	12	13	14	15	16	17	18	19	1A	1B	1C
E	E	F	10	11	12	13	14	15	16	17	18	19	1A	1B	1C	1D
F	F	10	11	12	13	14	15	16	17	18	19	1A	1B	1C	1D	1E

To use this table, simply follow the directions used in this example − Add A_{16} and 5_{16}. Locate A in the X column then locate the 5 in the Y column. The point in 'sum' area where these two columns intersect is the sum of two numbers.

```
A16 &plus; 516 = F16.
```

Example − Addition

$4A6_{16} + 1B3_{16} = 659_{16}$

```
 1        carry
 4 A 6   = 1190₁₀
+1 B 3   =  435₁₀
────────
 6 5 9   = 1625₁₀
```

Hexadecimal Subtraction

The subtraction of hexadecimal numbers follow the same rules as the subtraction of numbers in any other number system. The only variation is in borrowed number. In the decimal system, you borrow a group of 10_{10}. In the binary system, you borrow a group of 2_{10}. In the hexadecimal system you borrow a group of 16_{10}.

Example - Subtraction

$4A6_{16} - 1B3_{16} = 2F3_{16}$

```
  16      borrow
 ³4 A 6   = 1190₁₀
 -1 B 3   =  435₁₀
```

2 F 3 $= 755_{10}$

BOOLEAN ALGEBRA

Boolean Algebra is used to analyze 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.

Rule in Boolean Algebra

Following are the important rules used in Boolean algebra.

- Variable used can have only two values. Binary 1 for HIGH and Binary 0 for LOW.
- Complement of a variable is represented by an overbar $-$. Thus, complement of variable B is represented as $\overline{B}$ = 0.
- ORing of the variables is represented by a plus $+$ sign between them. For example ORing of A, B, C is represented as A &plus; B &plus; C.
- Logical ANDing of the two or more variable is represented by writing a dot between them such as A.B.C. Sometime the dot may be omitted like ABC.

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.

$$(i)\ A + 0 = A \qquad (ii)\ A + 1 = 1$$
$$(iii)\ A + A = A \qquad (iv)\ A + \overline{A} = 1$$

INVERSION law

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

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

Important Boolean Theorems

Following are few important boolean Theorems.

Boolean function/theorems	Description
Boolean Functions	Boolean Functions and Expressions, K-Map and NAND Gates realization
De Morgan's Theorems	De Morgan's Theorem 1 and Theorem 2

LOGIC GATES

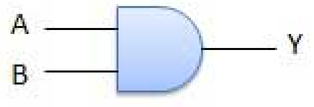
Logic gates are the basic building blocks of any digital system. It is an electronic circuit having one or more than one input and only one output. The relationship between the input and the output is based on a **certain logic**. Based on this, logic gates are named as AND gate, OR gate, NOT gate etc.

AND Gate

A circuit which performs an AND operation is shown in figure. It has n input $n >= 2$ and one output.

$$Y = A \text{ AND } B \text{ AND } C \ldots\ldots N$$
$$Y = A.B.C \ldots\ldots N$$
$$Y = ABC \ldots\ldots N$$

Logic diagram



Truth Table

Inputs		Output
A	B	AB
0	0	0
0	1	0
1	0	0

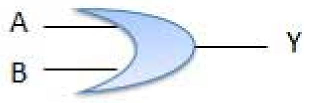
1	1	1

OR Gate

A circuit which performs an OR operation is shown in figure. It has n input $n >= 2$ and one output.

Y	=	A OR B OR C .... N
Y	=	A + B + C .... N

Logic diagram



Truth Table

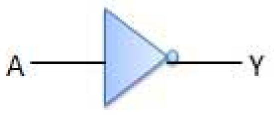
Inputs		Output
A	B	A + B
0	0	0
0	1	1
1	0	1
1	1	1

NOT Gate

NOT gate is also known as **Inverter**. It has one input A and one output Y.

Y	=	NOT A
Y	=	$\overline{A}$

Logic diagram



Truth Table

Inputs	Output
A	B
0	1
1	0

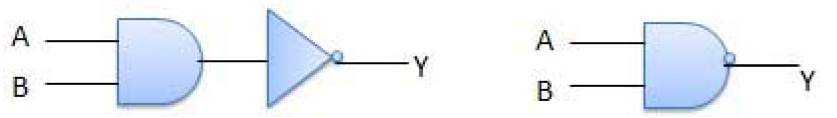
NAND Gate

A NOT-AND operation is known as NAND operation. It has n input $n >= 2$ and one output.

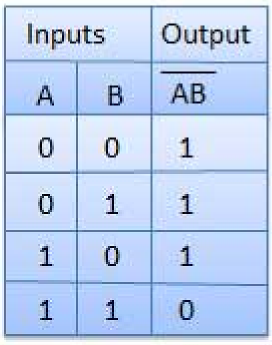
Y	=	A NOT AND B NOT AND C .... N

Y = A NAND B NAND C .... N

Logic diagram



Truth Table



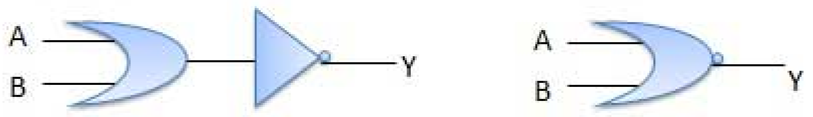
Inputs		Output
A	B	$\overline{AB}$
0	0	1
0	1	1
1	0	1
1	1	0

NOR Gate

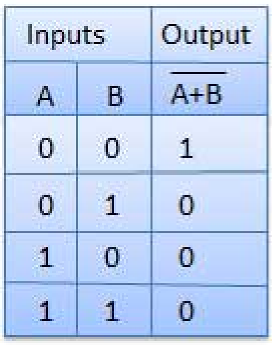
A NOT-OR operation is known as NOR operation. It has n input $n >= 2$ and one output.

Y = A NOT OR B NOT OR C .... N
Y = A NOR B NOR C .... N

Logic diagram



Truth Table



Inputs		Output
A	B	$\overline{A+B}$
0	0	1
0	1	0
1	0	0
1	1	0

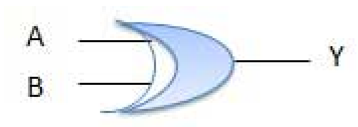
XOR Gate

XOR or Ex-OR gate is a special type of gate. It can be used in the half adder, full adder and subtractor. The exclusive-OR gate is abbreviated as EX-OR gate or sometime as X-OR gate. It has n input $n >= 2$ and one output.

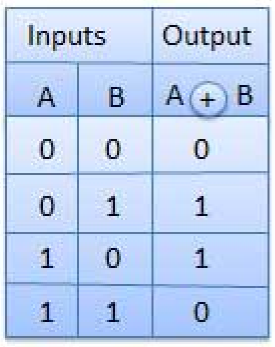
Y = A XOR B XOR C .... N
Y = $A \oplus B \oplus C$ .... N

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

Logic diagram



Truth Table



Inputs		Output
A	B	$A \oplus B$
0	0	0
0	1	1
1	0	1
1	1	0

XNOR Gate

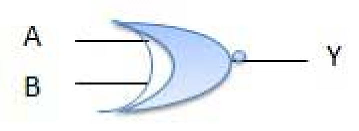
XNOR gate is a special type of gate. It can be used in the half adder, full adder and subtractor. The exclusive-NOR gate is abbreviated as EX-NOR gate or sometime as X-NOR gate. It has n input $n >= 2$ and one output.

$$Y = A \text{ XOR } B \text{ XOR } C \ldots\ldots N$$

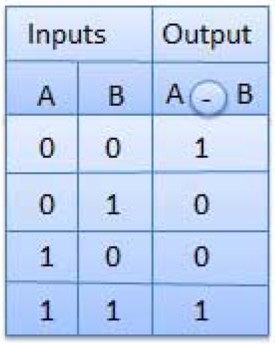
$$Y = A \odot B \odot C \ldots\ldots N$$

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

Logic diagram



Truth Table



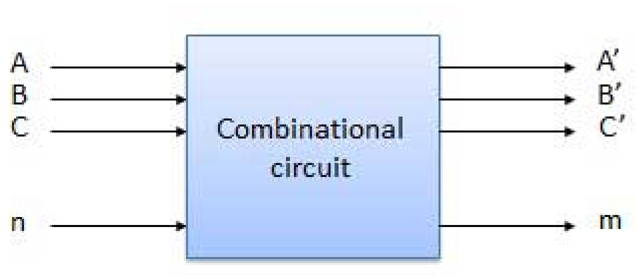
Inputs		Output
A	B	$A \odot B$
0	0	1
0	1	0
1	0	0
1	1	1

COMBINATIONAL CIRCUITS

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

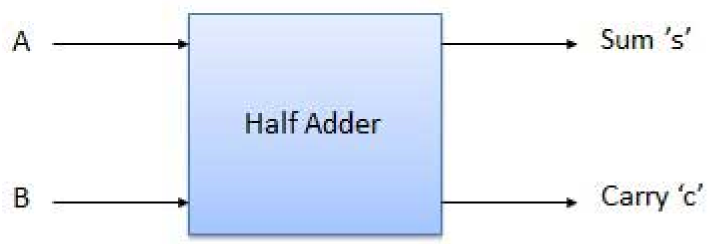


We're going to elaborate few important combinational circuits as follows.

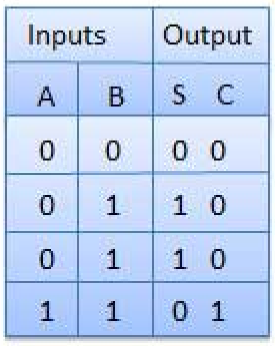
Half Adder

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

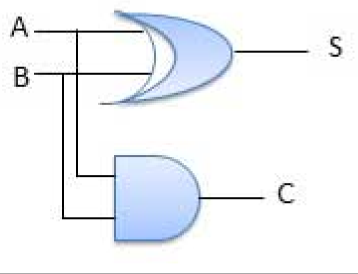


Truth Table



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

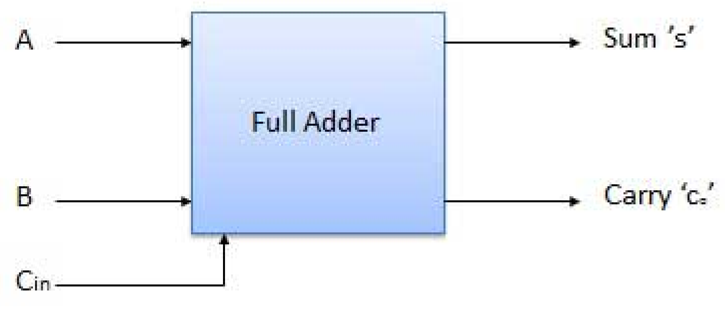
Circuit Diagram



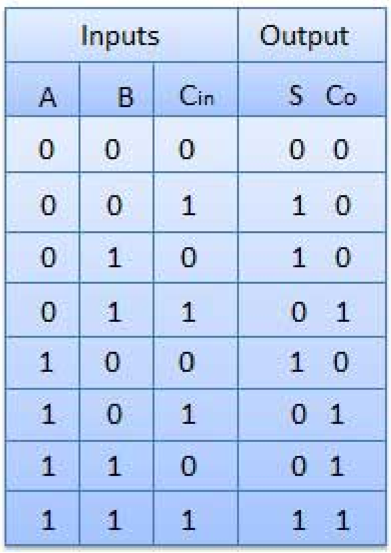
Full Adder

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

Block diagram

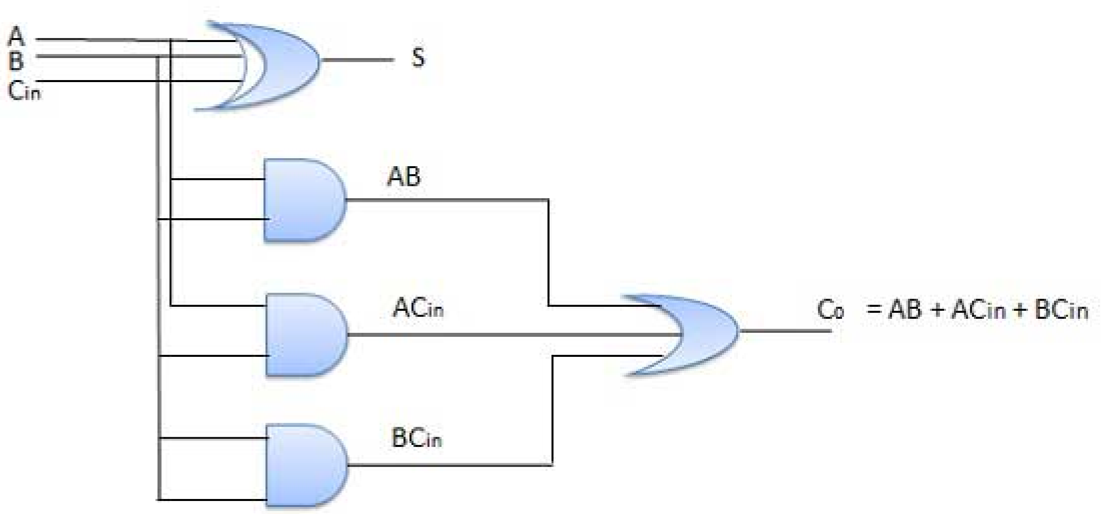


Truth Table



Inputs			Output	
A	B	C_{in}	S	C_o
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



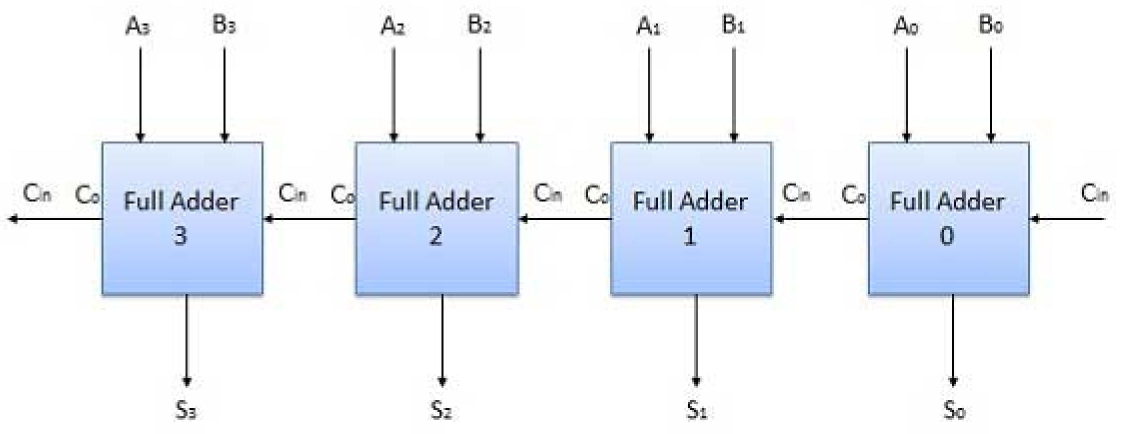
N-Bit Parallel Adder

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



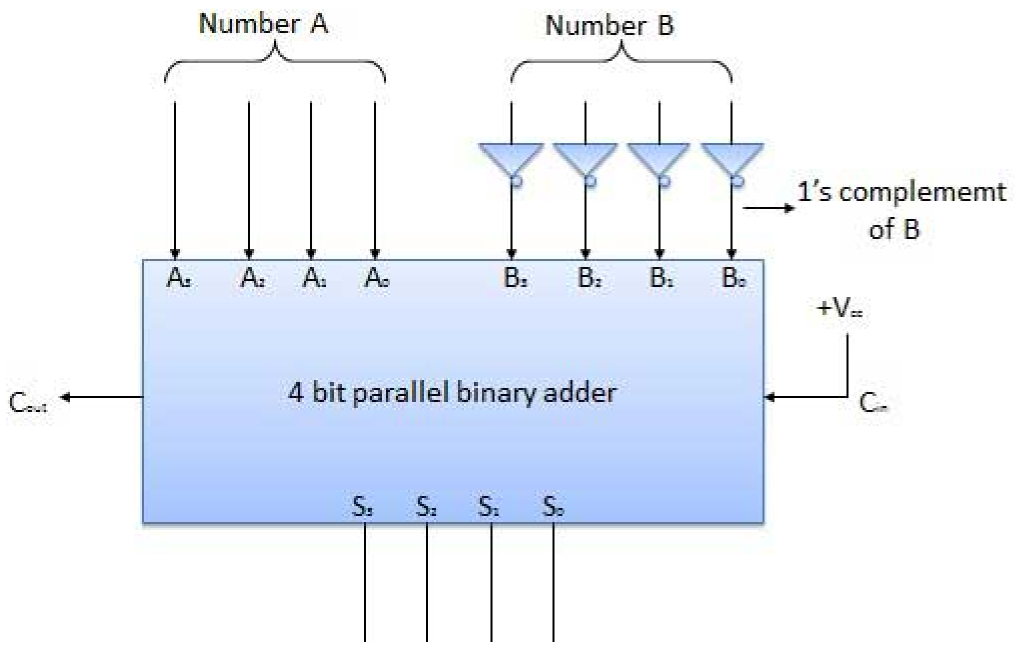
N-Bit Parallel Subtractor

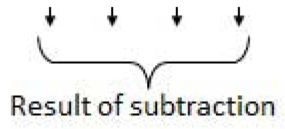
The subtraction can be carried out by taking the 1's or 2's complement of the number to be subtracted. For example we can perform the subtraction $A - B$ by adding either 1's or 2's complement of B to A. That means we can use a binary adder to perform the binary subtraction.

4 Bit Parallel Subtractor

The number to be subtracted B is first passed through inverters to obtain its 1's complement. The 4-bit adder then adds A and 2's complement of B to produce the subtraction. S_3 S_2 S_1 S_0 represents the result of binary subtraction $A - B$ and carry output C_{out} represents the polarity of the result. If A > B then Cout = 0 and the result of binary form $A - B$ then C_{out} = 1 and the result is in the 2's complement form.

Block diagram

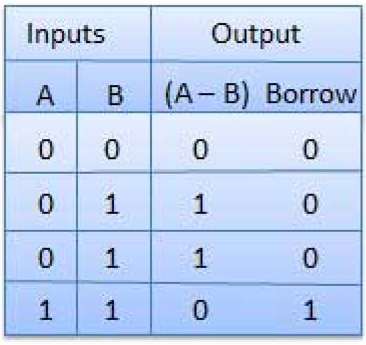




Half Subtractors

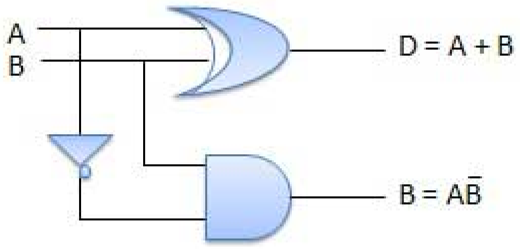
Half subtractor is a combination circuit with two inputs and two outputs *differenceandborrow*. It produces the difference between the two binary bits at the input and also produces an output *Borrow* to indicate if a 1 has been borrowed. In the subtraction $A - B$, A is called as Minuend bit and B is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

Truth Table



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

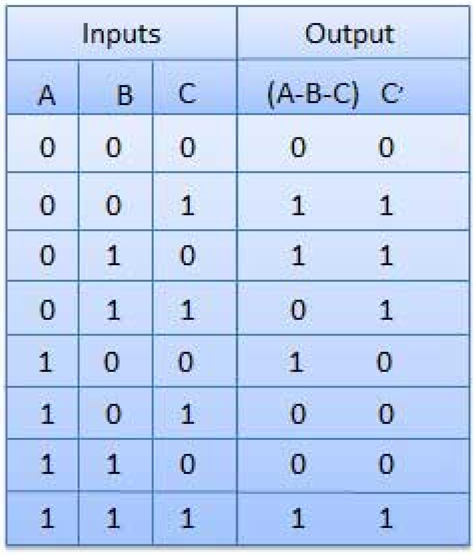
Circuit Diagram



Full Subtractors

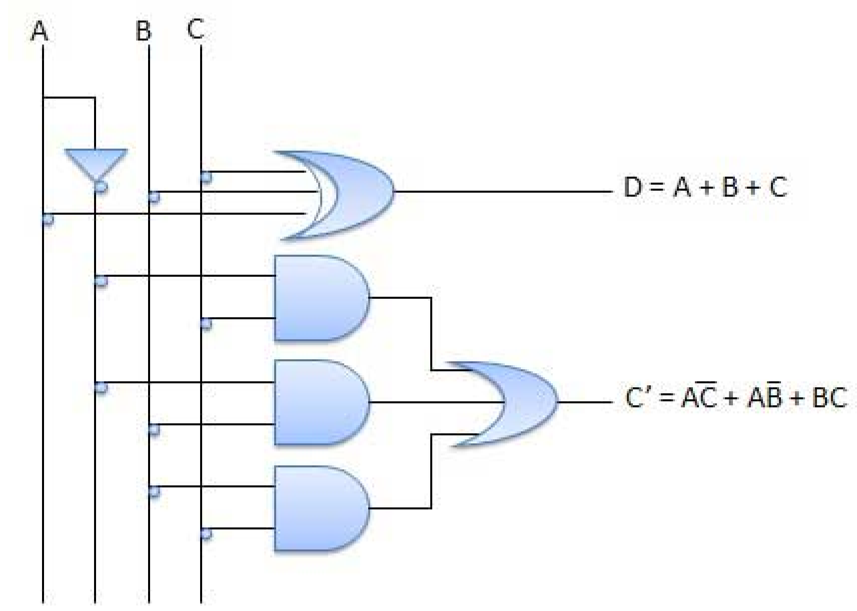
The disadvantage of a half subtractor is overcome by full subtractor. The full subtractor is a combinational circuit with three inputs A,B,C and two output D and C'. A is the 'minuend', B is 'subtrahend', C is the 'borrow' produced by the previous stage, D is the difference output and C' is the borrow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

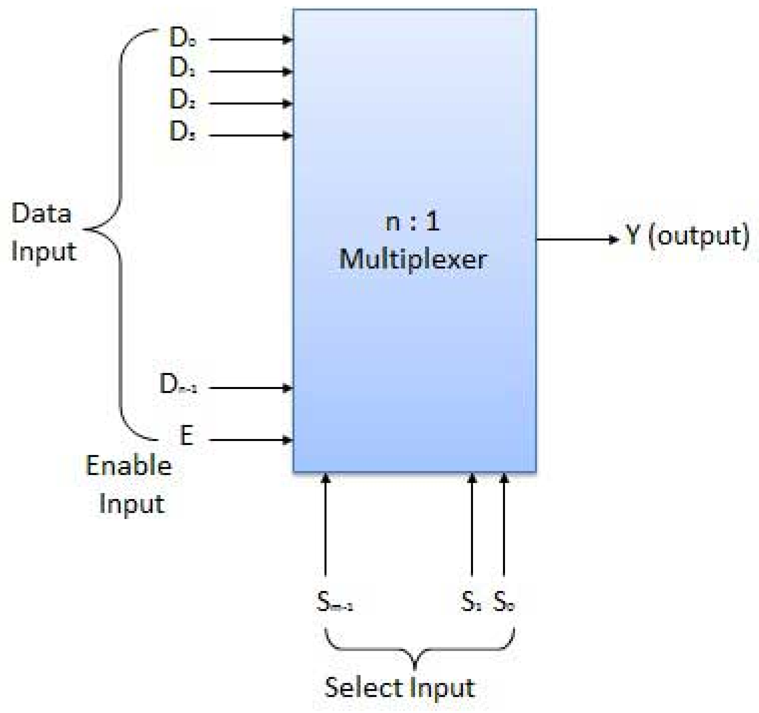
Circuit Diagram



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

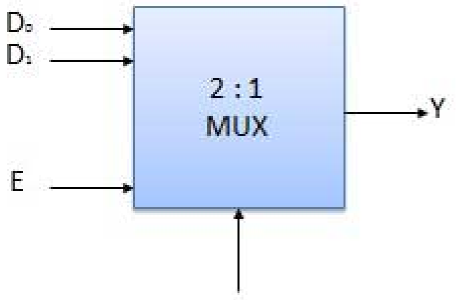


Multiplexers come in multiple variations

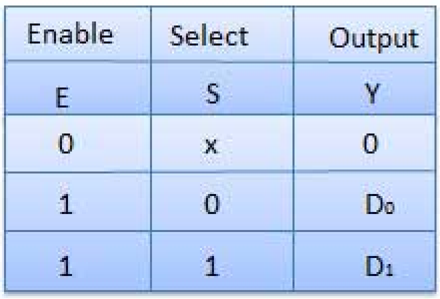
- 2 : 1 multiplexer
- 4 : 1 multiplexer

- 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

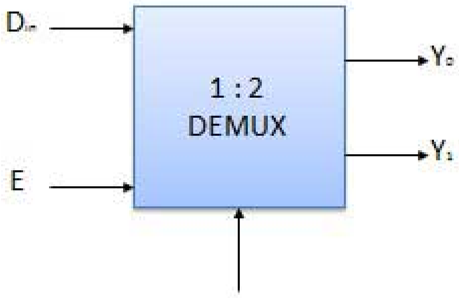
Demultiplexers

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.

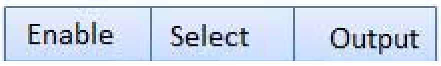
Demultiplexers comes in multiple variations.

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

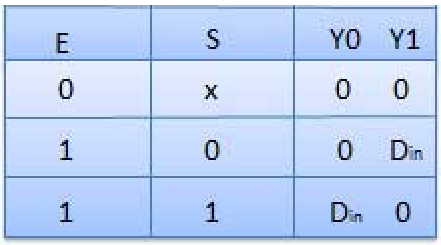
Block diagram



Truth Table



Enable	Select	Output



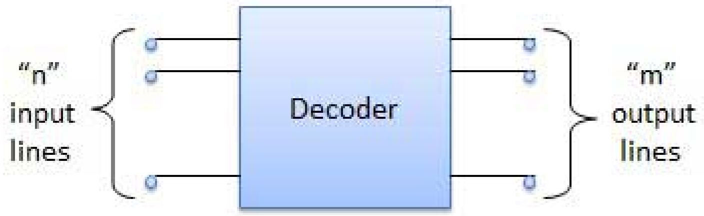
E	S	Y0	Y1
0	x	0	0
1	0	0	D_{in}
1	1	D_{in}	0

x = Don't care

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



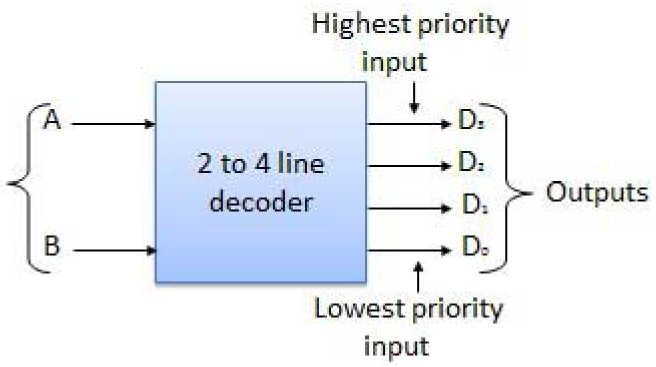
Examples of Decoders are following.

- Code converters
- BCD to seven segment decoders
- Nixie tube decoders
- Relay actuator

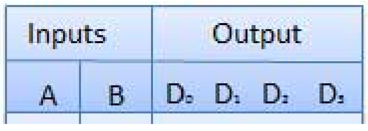
2 to 4 Line Decoder

The block diagram of 2 to 4 line decoder is shown in the fig. A and B are the two inputs where D through D are the four outputs. Truth table explains the operations of a decoder. It shows that each output is 1 for only a specific combination of inputs.

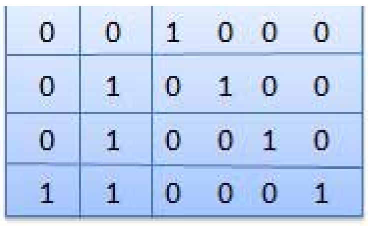
Block diagram



Truth Table

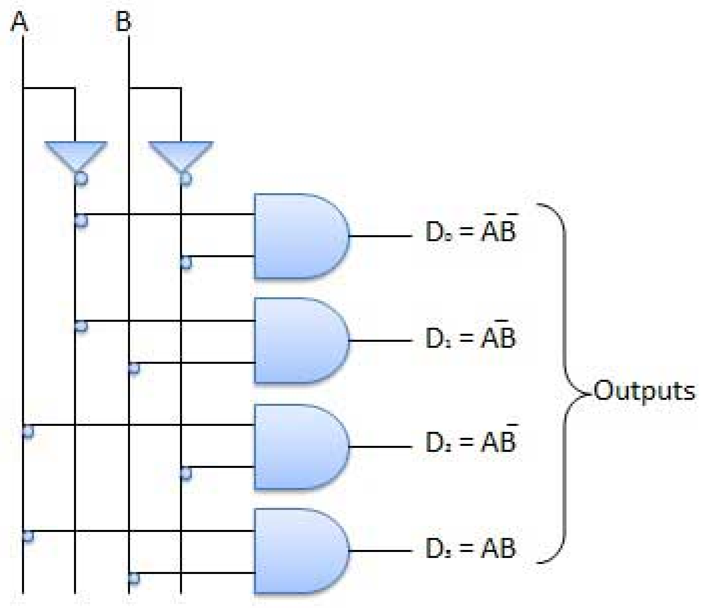


Inputs		Output			
A	B	D_0	D_1	D_2	D_3



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

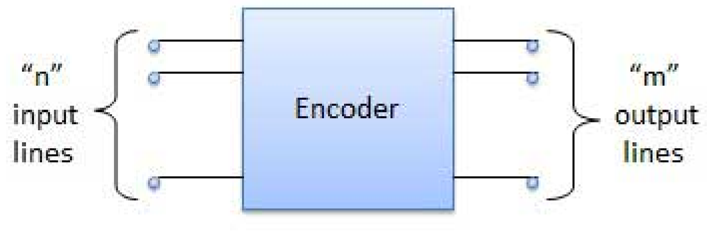
Logic Circuit



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



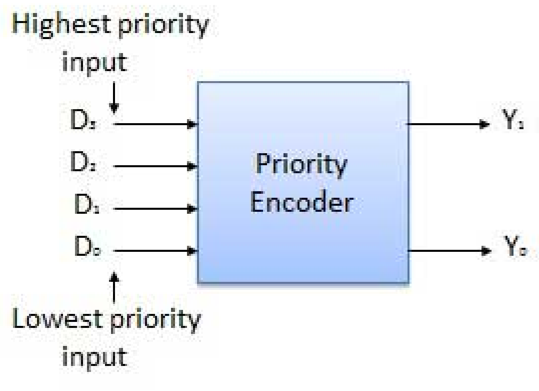
Examples of Encoders are following.

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

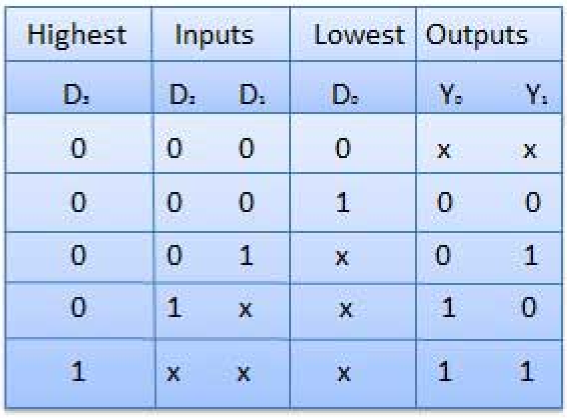
Priority Encoder

This is a special type of encoder. Priority is given to the input lines. If two or more input line are 1 at the same time, then the input line with highest priority will be considered. There are four input D_0, D_1, D_2, D_3 and two output Y_0, Y_1. Out of the four input D_3 has the highest priority and D_0 has the lowest priority. That means if $D_3 = 1$ then $Y_1\ Y_1 = 11$ irrespective of the other inputs. Similarly if $D_3 = 0$ and $D_2 = 1$ then $Y_1\ Y_0 = 10$ irrespective of the other inputs.

Block diagram

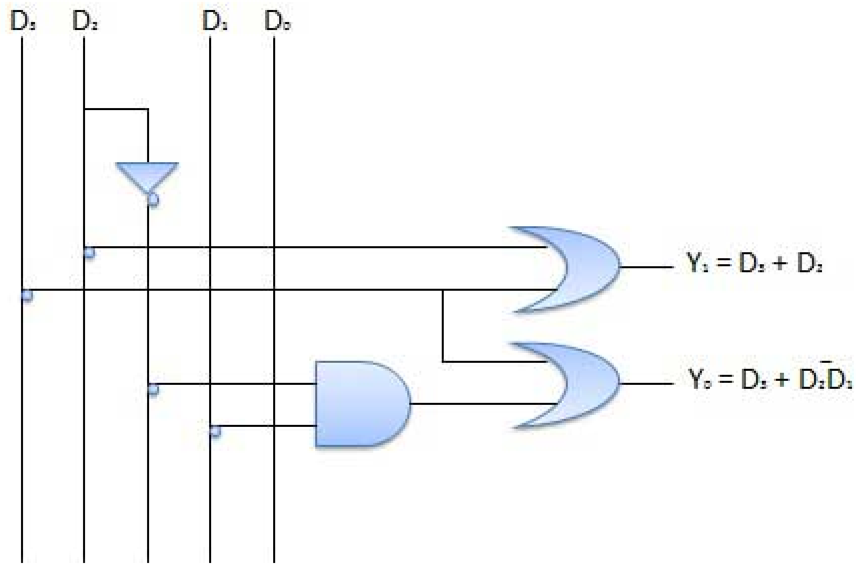


Truth Table



Highest	Inputs		Lowest	Outputs	
D_3	D_2	D_1	D_0	Y_1	Y_0
0	0	0	0	x	x
0	0	0	1	0	0
0	0	1	x	0	1
0	1	x	x	1	0
1	x	x	x	1	1

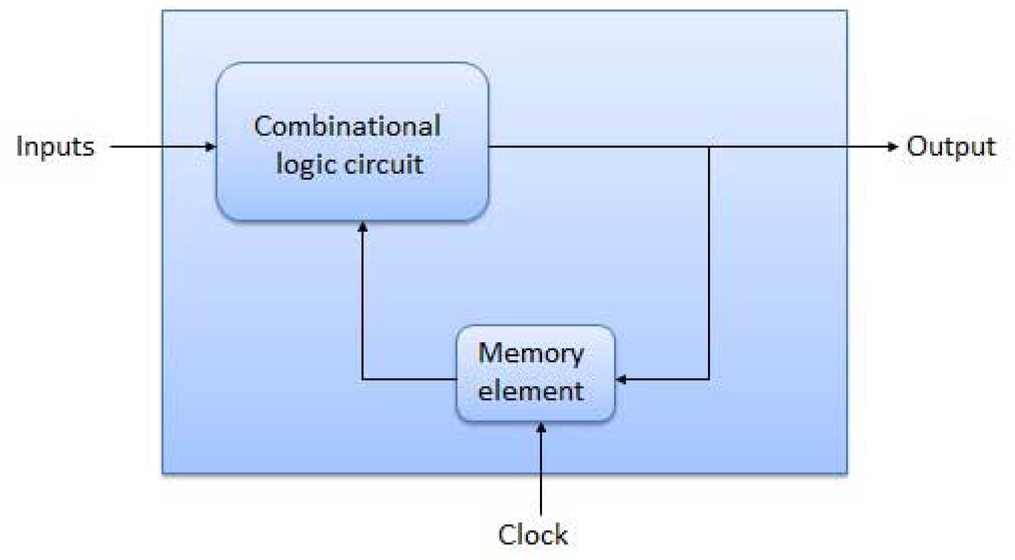
Logic Circuit



SEQUENTIAL CIRCUITS

The combinational circuit does not use any memory. Hence the previous state of input does not have any effect on the present state of the circuit. But sequential circuit has memory so output can vary based on input. This type of circuits uses previous input, output, clock and a memory element.

Block diagram



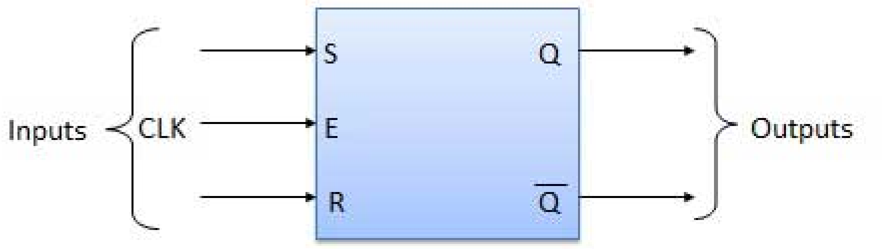
Flip Flop

Flip flop is a sequential circuit which generally samples its inputs and changes its outputs only at particular instants of time and not continuously. Flip flop is said to be edge sensitive or edge triggered rather than being level triggered like latches.

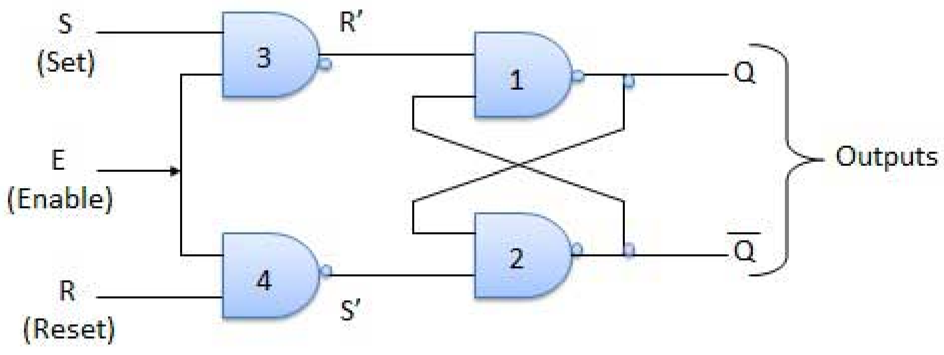
S-R Flip Flop

It is basically S-R latch using NAND gates with an additional **enable** input. It is also called as level triggered SR-FF. For this, circuit in output will take place if and only if the enable input E is made active. In short this circuit will operate as an S-R latch if E = 1 but there is no change in the output if E = 0.

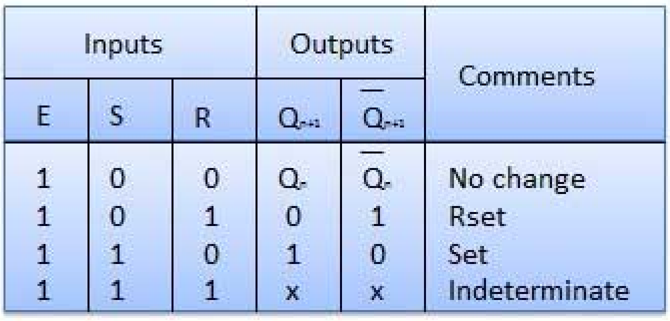
Block Diagram



Circuit Diagram



Truth Table



Inputs			Outputs		Comments
E	S	R	Q_{n+1}	$\overline{Q}_{n+1}$	
1	0	0	Q_n	$\overline{Q}_n$	No change
1	0	1	0	1	Rset
1	1	0	1	0	Set
1	1	1	x	x	Indeterminate

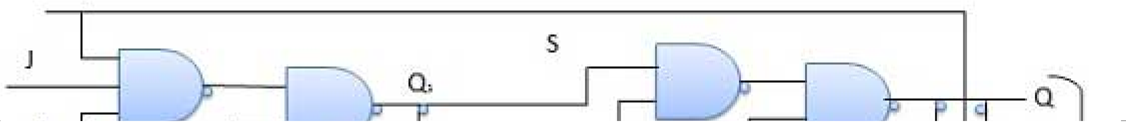
Operation

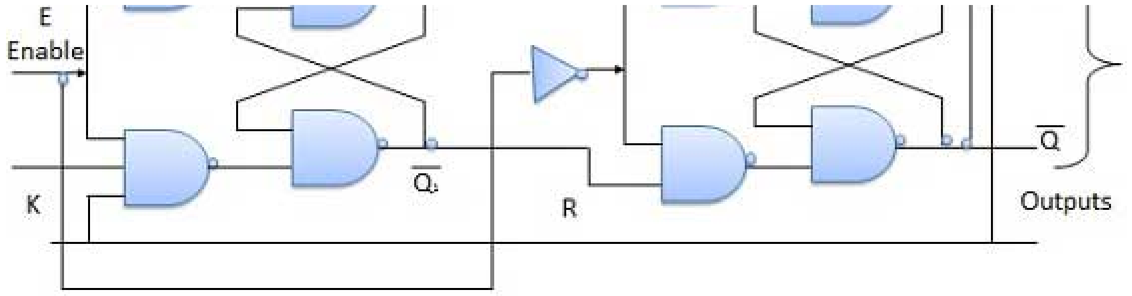
S.N.	Condition	Operation
1	**S = R = 0 : No change**	If S = R = 0 then output of NAND gates 3 and 4 are forced to become 1. Hence R' and S' both will be equal to 1. Since S' and R' are the input of the basic S-R latch using NAND gates, there will be no change in the state of outputs.
2	**S = 0, R = 1, E = 1**	Since S = 0, output of NAND-3 i.e. R' = 1 and E = 1 the output of NAND-4 i.e. S' = 0. Hence $Q_{n\&plus;1}$ = 0 and $Q_{n\&plus;1}$ bar = 1. This is reset condition.
3	**S = 1, R = 0, E = 1**	Output of NAND-3 i.e. R' = 0 and output of NAND-4 i.e. S' = 1. Hence output of S-R NAND latch is $Q_{n\&plus;1}$ = 1 and $Q_{n\&plus;1}$ bar = 0. This is the reset condition.
4	**S = 1, R = 1, E = 1**	As S = 1, R = 1 and E = 1, the output of NAND gates 3 and 4 both are 0 i.e. S' = R' = 0. Hence the **Race** condition will occur in the basic NAND latch.

Master Slave JK Flip Flop

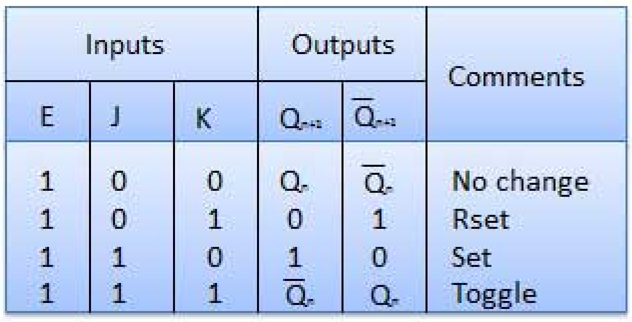
Master slave JK FF is a cascade of two S-R FF with feedback from the output of second to input of first. Master is a positive level triggered. But due to the presence of the inverter in the clock line, the slave will respond to the negative level. Hence when the clock = 1 *positivelevel* the master is active and the slave is inactive. Whereas when clock = 0 *lowlevel* the slave is active and master is inactive.

Circuit Diagram





Truth Table



Inputs			Outputs		Comments
E	J	K	Q_{n+1}	$\overline{Q}_{n+1}$	
1	0	0	Q_n	$\overline{Q}_n$	No change
1	0	1	0	1	Rset
1	1	0	1	0	Set
1	1	1	$\overline{Q}_n$	Q_n	Toggle

Operation

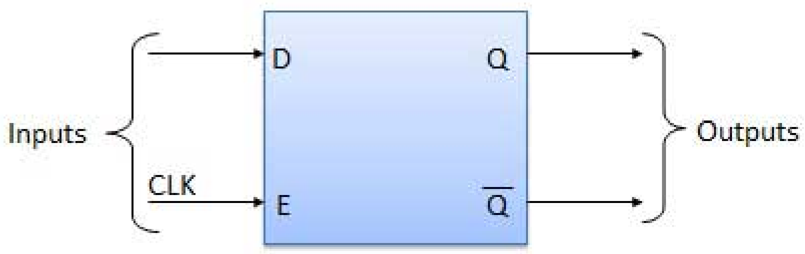
S.N.	Condition	Operation
1	**J = K = 0** *Nochange*	When clock = 0, the slave becomes active and master is inactive. But since the S and R inputs have not changed, the slave outputs will also remain unchanged. Therefore outputs will not change if J = K =0.
2	**J = 0 and K = 1** *Reset*	Clock = 1 − Master active, slave inactive. Therefore outputs of the master become Q_1 = 0 and Q_1 bar = 1. That means S = 0 and R =1. Clock = 0 − Slave active, master inactive. Therefore outputs of the slave become Q = 0 and Q bar = 1. Again clock = 1 − Master active, slave inactive. Therefore even with the changed outputs Q = 0 and Q bar = 1 fed back to master, its output will be Q1 = 0 and Q1 bar = 1. That means S = 0 and R = 1. Hence with clock = 0 and slave becoming active the outputs of slave will remain Q = 0 and Q bar = 1. Thus we get a stable output from the Master slave.
3	**J = 1 and K = 0** *Set*	Clock = 1 − Master active, slave inactive. Therefore outputs of the master become Q_1 = 1 and Q_1 bar = 0. That means S = 1 and R =0. Clock = 0 − Slave active, master inactive. Therefore outputs of the slave become Q = 1 and Q bar = 0. Again clock = 1 − then it can be shown that the outputs of the slave are stabilized to Q = 1 and Q bar = 0.

4	**J = K = 1** *Toggle*	Clock = 1 − Master active, slave inactive. Outputs of master will toggle. So S and R also will be inverted. Clock = 0 − Slave active, master inactive. Outputs of slave will toggle. These changed output are returned back to the master inputs. But since clock = 0, the master is still inactive. So it does not respond to these changed outputs. This avoids the multiple toggling which leads to the race around condition. The master slave flip flop will avoid the race around condition.

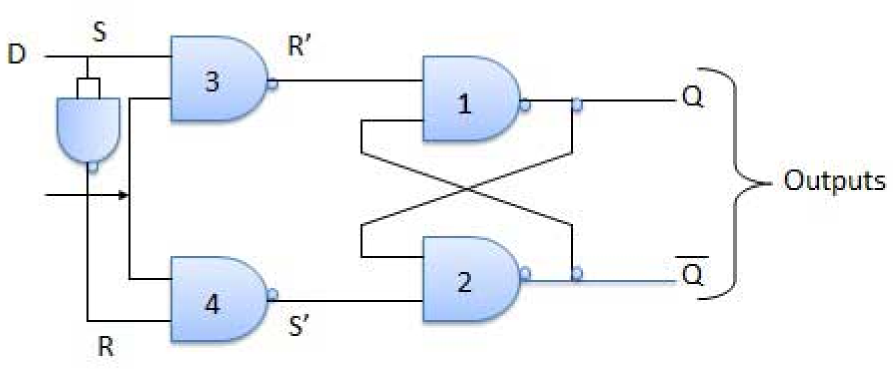
Delay Flip Flop / D Flip Flop

Delay Flip Flop or D Flip Flop is the simple gated S-R latch with a NAND inverter connected between S and R inputs. It has only one input. The input data is appearing at the output after some time. Due to this data delay between i/p and o/p, it is called delay flip flop. S and R will be the complements of each other due to NAND inverter. Hence S = R = 0 or S = R = 1, these input condition will never appear. This problem is avoid by SR = 00 and SR = 1 conditions.

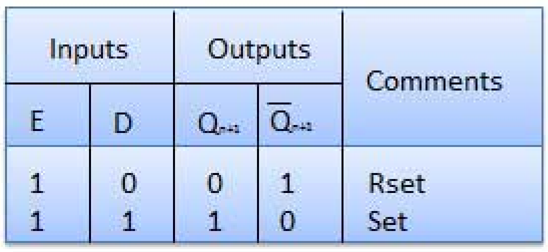
Block Diagram



Circuit Diagram



Truth Table



Inputs		Outputs		Comments
E	D	Q_{n+1}	$\overline{Q}_{n+1}$	
1	0	0	1	Rset
1	1	1	0	Set

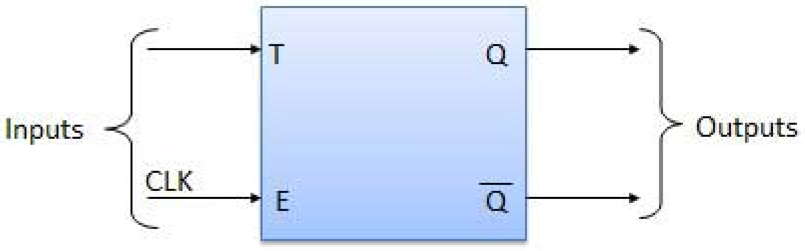
Operation

S.N.	Condition	Operation
1	**E = 0**	Latch is disabled. Hence no change in output.
2	**E = 1 and D = 0**	If E = 1 and D = 0 then S = 0 and R = 1. Hence irrespective of the present state, the next state is $Q_{n\&plus;1}$ = 0 and $Q_{n\&plus;1}$ bar = 1. This is the reset condition.
3	**E = 1 and D = 1**	If E = 1 and D = 1, then S = 1 and R = 0. This will set the latch and $Q_{n\&plus;1}$ = 1 and $Q_{n\&plus;1}$ bar = 0 irrespective of the present state.

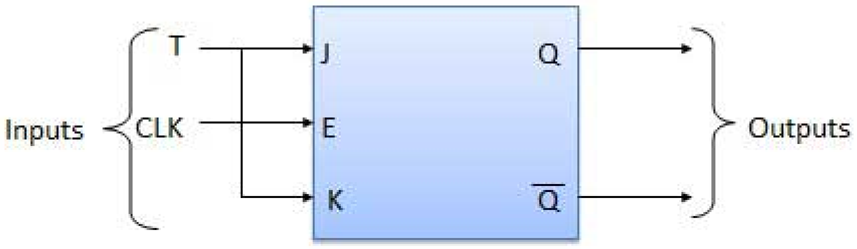
Toggle Flip Flop / T Flip Flop

Toggle flip flop is basically a JK flip flop with J and K terminals permanently connected together. It has only input denoted by **T** as shown in the Symbol Diagram. The symbol for positive edge triggered T flip flop is shown in the Block Diagram.

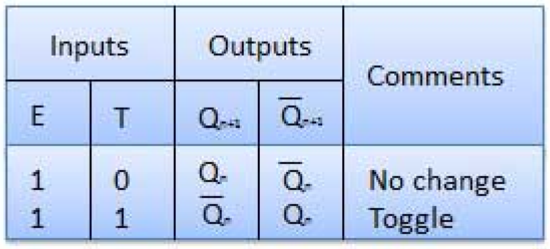
Symbol Diagram



Block Diagram



Truth Table



Inputs		Outputs		Comments
E	T	Q_{n+1}	$\overline{Q}_{n+1}$	
1	0	Q_n	$\overline{Q}_n$	No change
1	1	$\overline{Q}_n$	Q_n	Toggle

Operation

S.N.	Condition	Operation

1	**T = 0, J = K = 0**	The output Q and Q bar won't change
2	**T = 1, J = K = 1**	Output will toggle corresponding to every leading edge of clock signal.

DIGITAL REGISTERS

Flip-flop is a 1 bit memory cell which can be used for storing the digital data. To increase the storage capacity in terms of number of bits, we have to use a group of flip-flop. Such a group of flip-flop is known as a **Register**. The **n-bit register** will consist of **n** number of flip-flop and it is capable of storing an **n-bit** word.

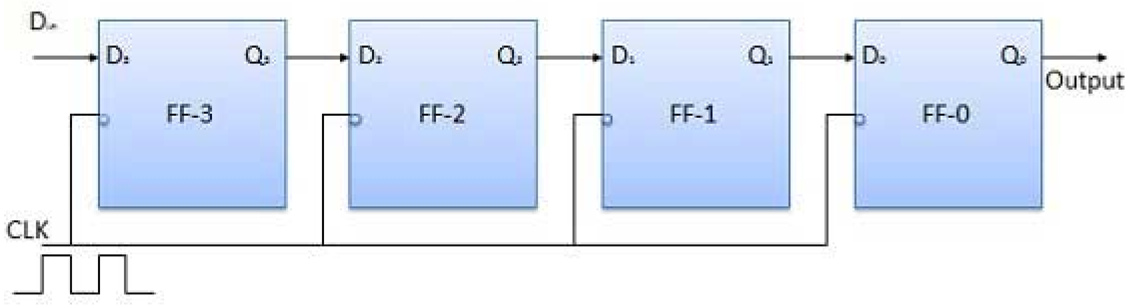
The binary data in a register can be moved within the register from one flip-flop to another. The registers that allow such data transfers are called as **shift registers**. There are four mode of operations of a shift register.

- Serial Input Serial Output
- Serial Input Parallel Output
- Parallel Input Serial Output
- Parallel Input Parallel Output

Serial Input Serial Output

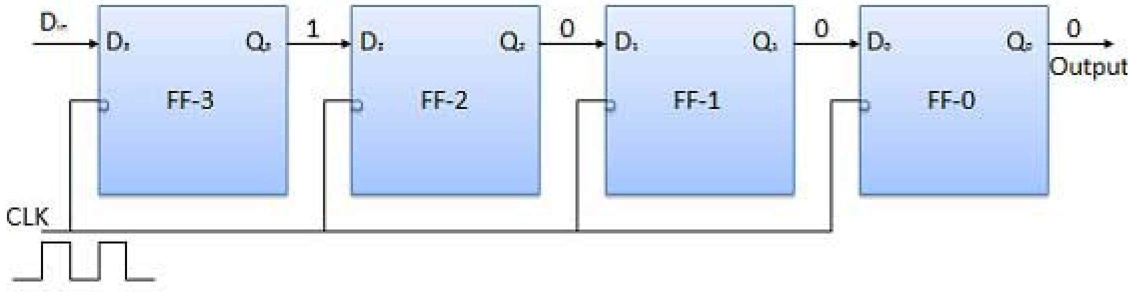
Let all the flip-flop be initially in the reset condition i.e. $Q_3 = Q_2 = Q_1 = Q_0 = 0$. If an entry of a four bit binary number 1 1 1 1 is made into the register, this number should be applied to $\mathbf{D_{in}}$ bit with the LSB bit applied first. The D input of FF-3 i.e. D_3 is connected to serial data input $\mathbf{D_{in}}$. Output of FF-3 i.e. Q_3 is connected to the input of the next flip-flop i.e. D_2 and so on.

Block Diagram

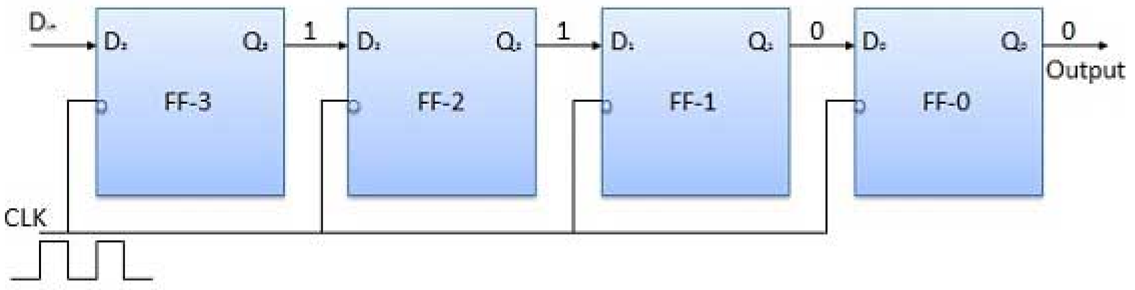


Operation

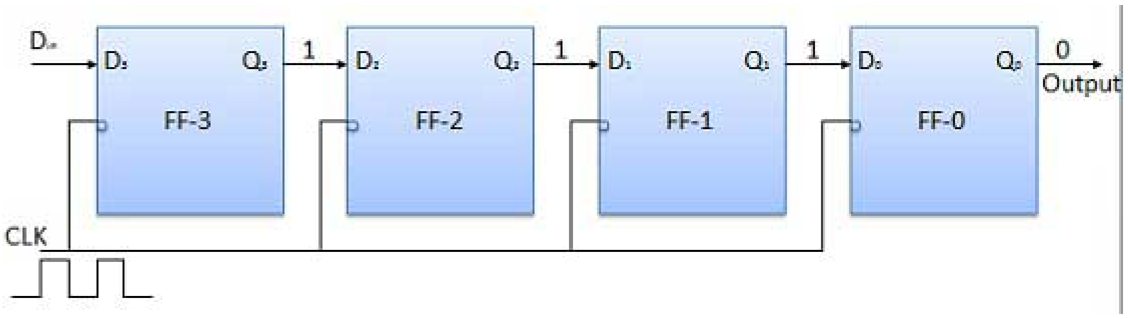
Before application of clock signal, let $Q_3\ Q_2\ Q_1\ Q_0 = 0000$ and apply LSB bit of the number to be entered to D_{in}. So $D_{in} = D_3 = 1$. Apply the clock. On the first falling edge of clock, the FF-3 is set, and stored word in the register is $Q_3\ Q_2\ Q_1\ Q_0 = 1000$.



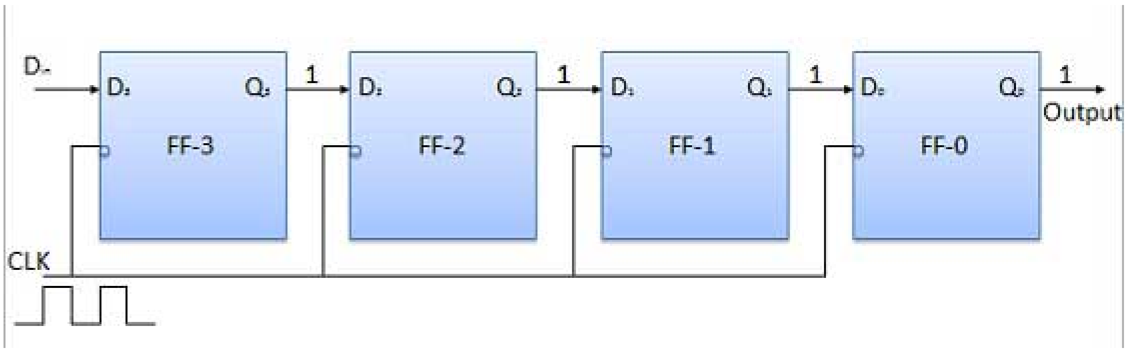
Apply the next bit to D_{in}. So $D_{in} = 1$. As soon as the next negative edge of the clock hits, FF-2 will set and the stored word change to $Q_3\ Q_2\ Q_1\ Q_0 = 1100$.



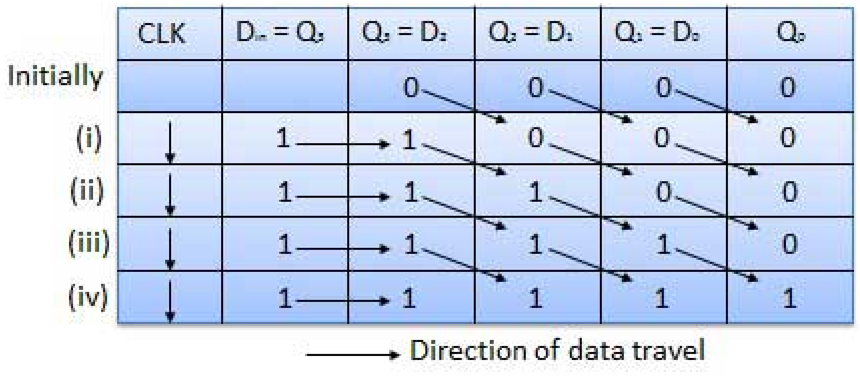
Apply the next bit to be stored i.e. 1 to D_{in}. Apply the clock pulse. As soon as the third negative clock edge hits, FF-1 will be set and output will be modified to Q_3 Q_2 Q_1 Q_0 = 1110.



Similarly with D_{in} = 1 and with the fourth negative clock edge arriving, the stored word in the register is Q_3 Q_2 Q_1 Q_0 = 1111.



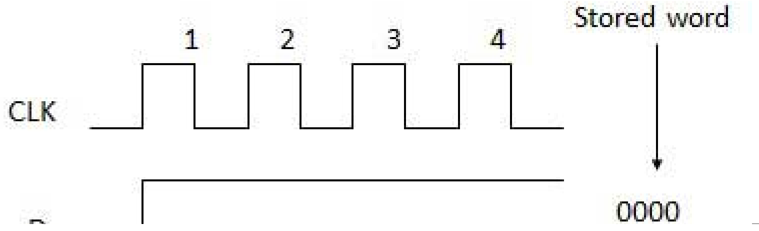
Truth Table

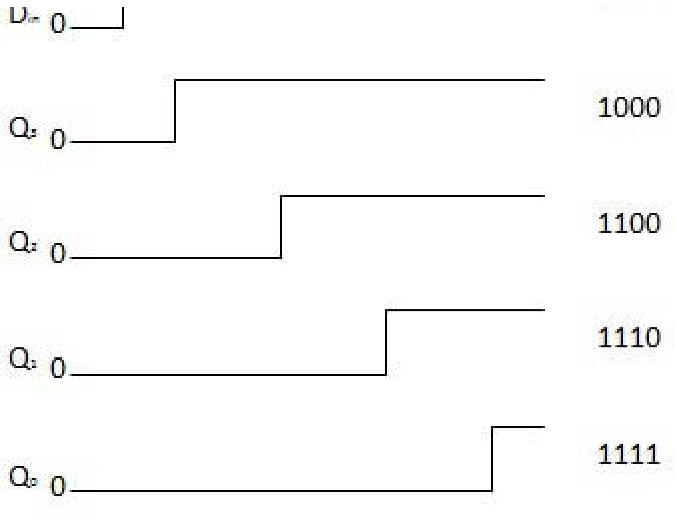


	CLK	$D_{in} = Q_3$	$Q_3 = D_2$	$Q_2 = D_1$	$Q_1 = D_0$	Q_0
Initially			0	0	0	0
(i)	↓	1	1	0	0	0
(ii)	↓	1	1	1	0	0
(iii)	↓	1	1	1	1	0
(iv)	↓	1	1	1	1	1

→ Direction of data travel

Waveforms

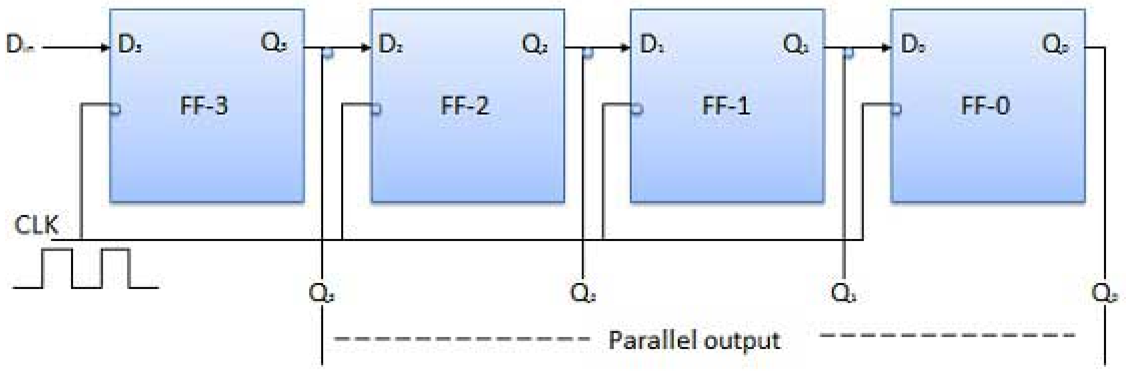




Serial Input Parallel Output

- In such types of operations, the data is entered serially and taken out in parallel fashion.
- Data is loaded bit by bit. The outputs are disabled as long as the data is loading.
- As soon as the data loading gets completed, all the flip-flops contain their required data, the outputs are enabled so that all the loaded data is made available over all the output lines at the same time.
- 4 clock cycles are required to load a four bit word. Hence the speed of operation of SIPO mode is same as that of SISO mode.

Block Diagram



Parallel Input Serial Output *PISO*

- Data bits are entered in parallel fashion.
- The circuit shown below is a four bit parallel input serial output register.
- Output of previous Flip Flop is connected to the input of the next one via a combinational circuit.
- The binary input word B_0, B_1, B_2, B_3 is applied though the same combinational circuit.
- There are two modes in which this circuit can work namely - shift mode or load mode.

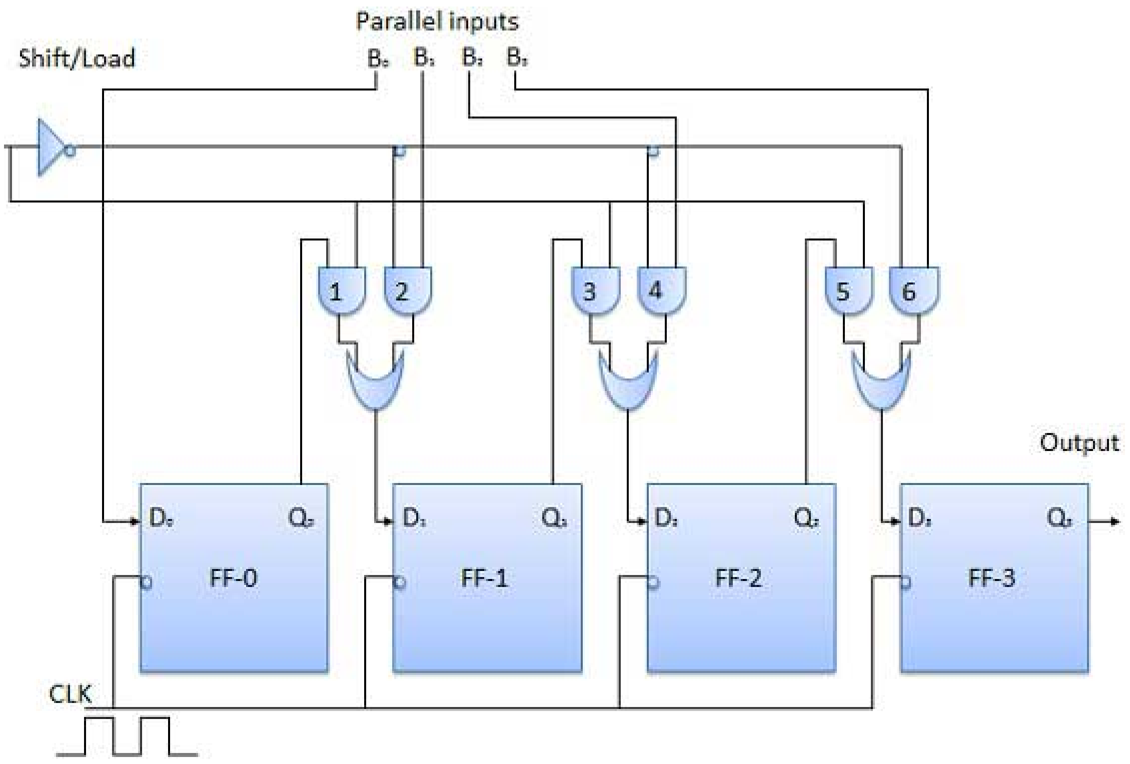
Load mode

When the shift/load bar line is low 0, the AND gate 2, 4 and 6 become active they will pass B_1, B_2, B_3 bits to the corresponding flip-flops. On the low going edge of clock, the binary input B_0, B_1, B_2, B_3 will get loaded into the corresponding flip-flops. Thus parallel loading takes place.

Shift mode

When the shift/load bar line is low 1, the AND gate 2, 4 and 6 become inactive. Hence the parallel loading of the data becomes impossible. But the AND gate 1,3 and 5 become active. Therefore the shifting of data from left to right bit by bit on application of clock pulses. Thus the parallel in serial out operation takes place.

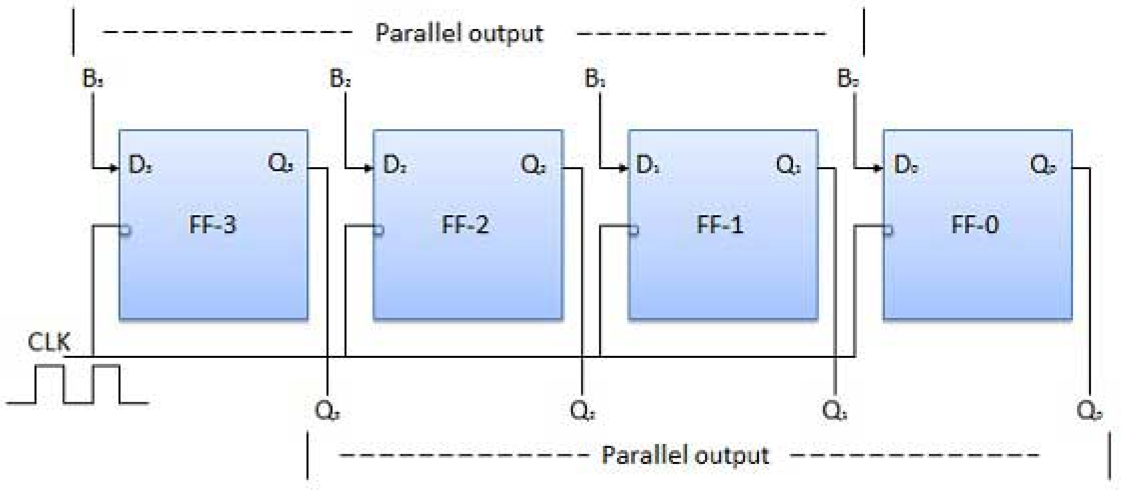
Block Diagram



Parallel Input Parallel Output *PIPO*

In this mode, the 4 bit binary input B_0, B_1, B_2, B_3 is applied to the data inputs D_0, D_1, D_2, D_3 respectively of the four flip-flops. As soon as a negative clock edge is applied, the input binary bits will be loaded into the flip-flops simultaneously. The loaded bits will appear simultaneously to the output side. Only clock pulse is essential to load all the bits.

Block Diagram

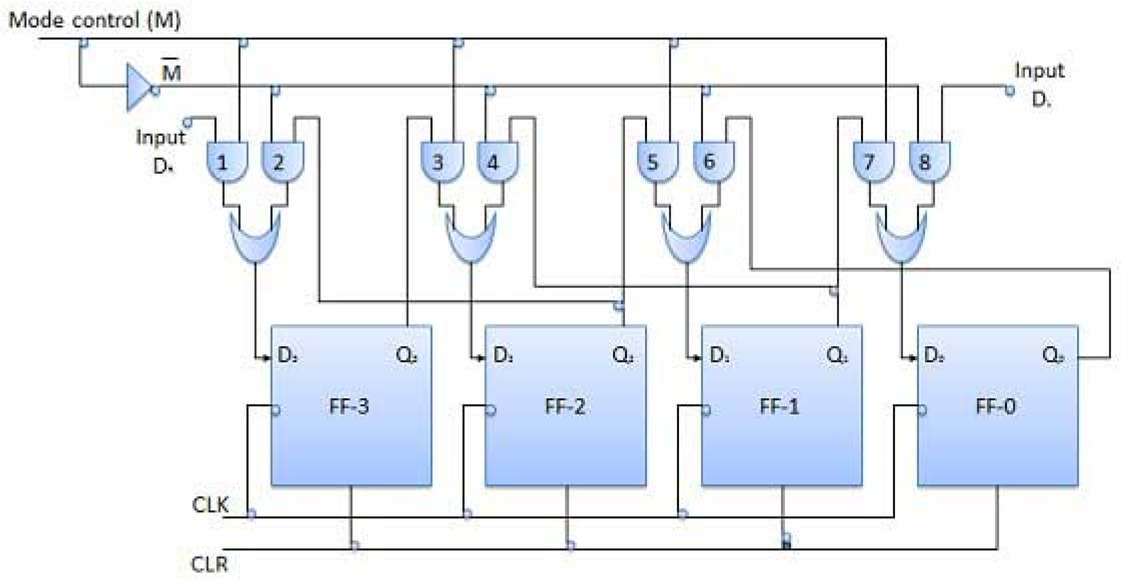


Bidirectional Shift Register

- If a binary number is shifted left by one position then it is equivalent to multiplying the original number by 2. Similarly if a binary number is shifted right by one position then it is equivalent to dividing the original number by 2.

- Hence if we want to use the shift register to multiply and divide the given binary number, then we should be able to move the data in either left or right direction.
- Such a register is called bi-directional register. A four bit bi-directional shift register is shown in fig.
- There are two serial inputs namely the serial right shift data input DR, and the serial left shift data input DL along with a mode select input M.

Block Diagram



Operation

S.N.	Condition	Operation
1	**With M = 1 − Shift right operation**	If M = 1, then the AND gates 1, 3, 5 and 7 are enabled whereas the remaining AND gates 2, 4, 6 and 8 will be disabled. The data at D_R is shifted to right bit by bit from FF-3 to FF-0 on the application of clock pulses. Thus with M = 1 we get the serial right shift operation.
2	**With M = 0 − Shift left operation**	When the mode control M is connected to 0 then the AND gates 2, 4, 6 and 8 are enabled while 1, 3, 5 and 7 are disabled. The data at D_L is shifted left bit by bit from FF-0 to FF-3 on the application of clock pulses. Thus with M = 0 we get the serial right shift operation.

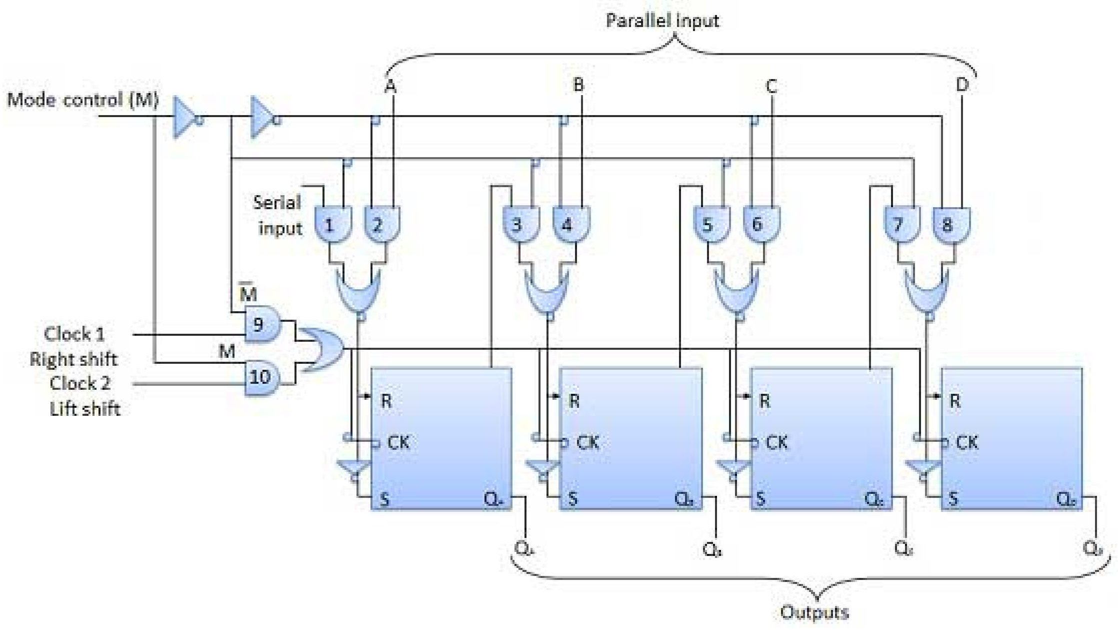
Universal Shift Register

A shift register which can shift the data in only one direction is called a uni-directional shift register. A shift register which can shift the data in both directions is called a bi-directional shift register. Applying the same logic, a shift register which can shift the data in both directions as well as load it parallely, is known as a universal shift register. The shift register is capable of performing the following operation −

- Parallel loading
- Lift shifting
- Right shifting

The mode control input is connected to logic 1 for parallel loading operation whereas it is connected to 0 for serial shifting. With mode control pin connected to ground, the universal shift register acts as a bi-directional register. For serial left operation, the input is applied to the serial input which goes to AND gate-1 shown in figure. Whereas for the shift right operation, the serial input is applied to D input.

Block Diagram



DIGITAL COUNTERS

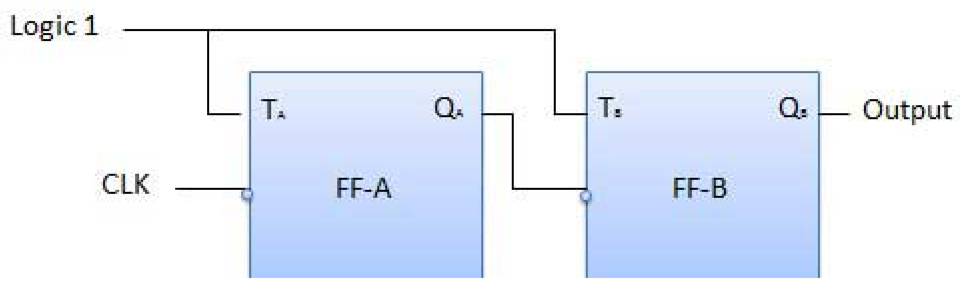
Counter is a sequential circuit. A digital circuit which is used for a counting pulses is known counter. Counter is the widest application of flip-flops. It is a group of flip-flops with a clock signal applied. Counters are of two types.

- Asynchronous or ripple counters.
- Synchronous counters.

Asynchronous or ripple counters

The logic diagram of a 2-bit ripple up counter is shown in figure. The toggle T flip-flop are being used. But we can use the JK flip-flop also with J and K connected permanently to logic 1. External clock is applied to the clock input of flip-flop A and Q_A output is applied to the clock input of the next flip-flop i.e. FF-B.

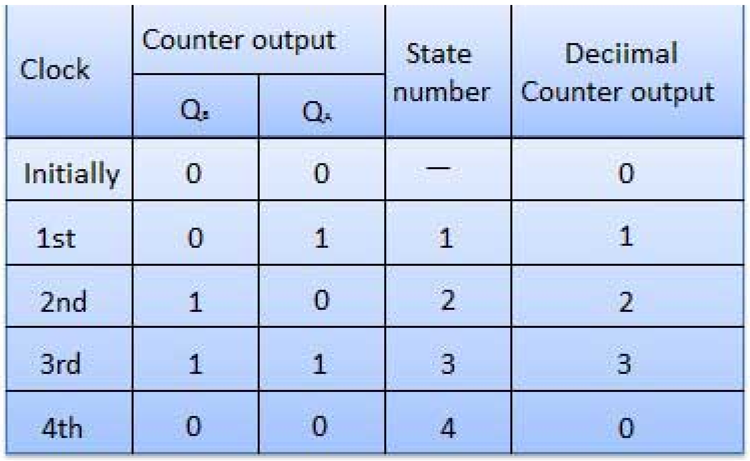
Logical Diagram



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 be equal to 1. Q_A is connected to clock input of FF-B. Since Q_A has changed from 0 to 1, it is treated as the positive clock edge by FF-B. There is no change in Q_B because FF-B is a negative edge triggered FF. 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 = 0. The change in Q_A acts as a negative clock edge for FF-B. So it will also toggle, and Q_B will be 1. Q_BQ_A = 10 after the second clock pulse.
4	**After 3rd negative clock edge**	On the arrival of 3rd negative clock edge, FF-A toggles again and Q_A become 1 from 0. Since this is a positive going change, FF-B does not respond to it and remains inactive. So Q_B does not change and continues to be equal to 1. Q_BQ_A = 11 after the third clock pulse.
5	**After 4th negative clock edge**	On the arrival of 4th negative clock edge, FF-A toggles again and Q_A becomes 1 from 0. This negative change in Q_A acts as clock pulse for FF-B. Hence it toggles to change Q_B from 1 to 0. Q_BQ_A = 00 after the fourth clock pulse.

Truth Table



Clock	Counter output		State number	Deciimal Counter output
	Q_B	Q_A		
Initially	0	0	—	0
1st	0	1	1	1
2nd	1	0	2	2
3rd	1	1	3	3
4th	0	0	4	0

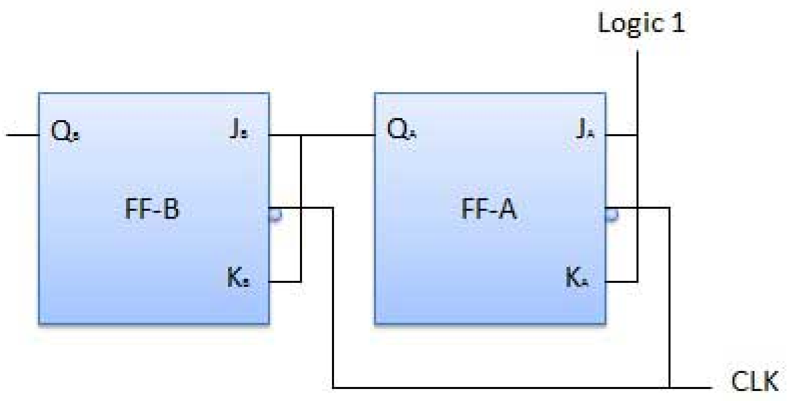
Synchronous counters

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.

Logical Diagram



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.

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 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 $Q = Qbar$ output of the previous FF.

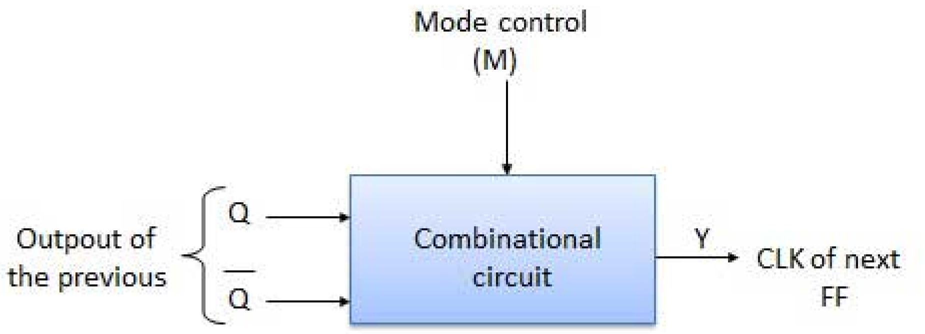
- **UP counting mode $\boldsymbol{M = 0}$** − The Q output of the preceding FF is connected to the clock of the next stage if up counting is to be achieved. For this mode, the mode select input M is at logic 0 $M = 0$.
- **DOWN counting mode $\boldsymbol{M = 1}$** − If M = 1, then the Q bar output of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

Example

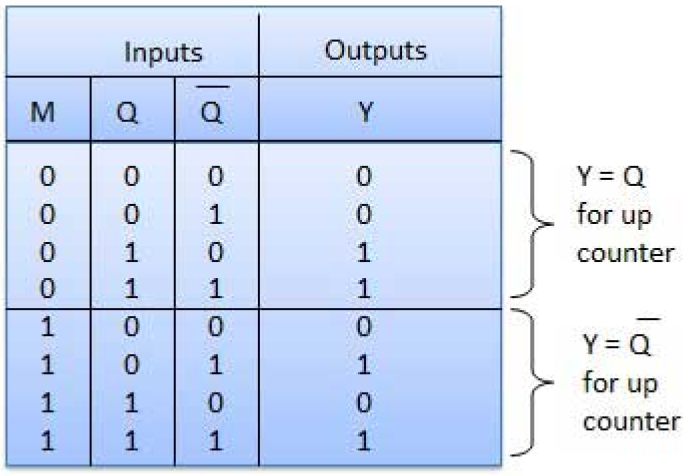
3-bit binary up/down ripple counter.

- 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.

Block Diagram



Truth Table



Inputs			Outputs
M	Q	$\overline{Q}$	Y
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	1

Y = Q for up counter

Y = $\overline{Q}$ for up counter

Operation

S.N.	Condition	Operation
1	**Case 1 − With M = 0** *Upcountingmode*	If M = 0 and M bar = 1, then the AND gates 1 and 3 in fig. will be enabled whereas the AND gates 2 and 4 will be disabled.

		Hence Q_A gets connected to the clock input of FF-B and Q_B gets connected to the clock input of FF-C. These connections are same as those for the normal up counter. Thus with M = 0 the circuit work as an up counter.
2	**Case 2: With M = 1** $Downcountingmode$	If M = 1, then AND gates 2 and 4 in fig. are enabled whereas the AND gates 1 and 3 are disabled. Hence Q_A bar gets connected to the clock input of FF-B and Q_B bar gets connected to the clock input of FF-C. These connections will produce a down counter. Thus with M = 1 the circuit works as a down counter.

Modulus Counter $MOD - NCounter$

The 2-bit ripple counter is called as MOD-4 counter and 3-bit ripple counter is called as MOD-8 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.

MEMORY DEVICES

A memory is just like a human brain. It is used to store data and instruction. Computer memory is the storage space in computer where data is to be processed and instructions required for processing are stored.

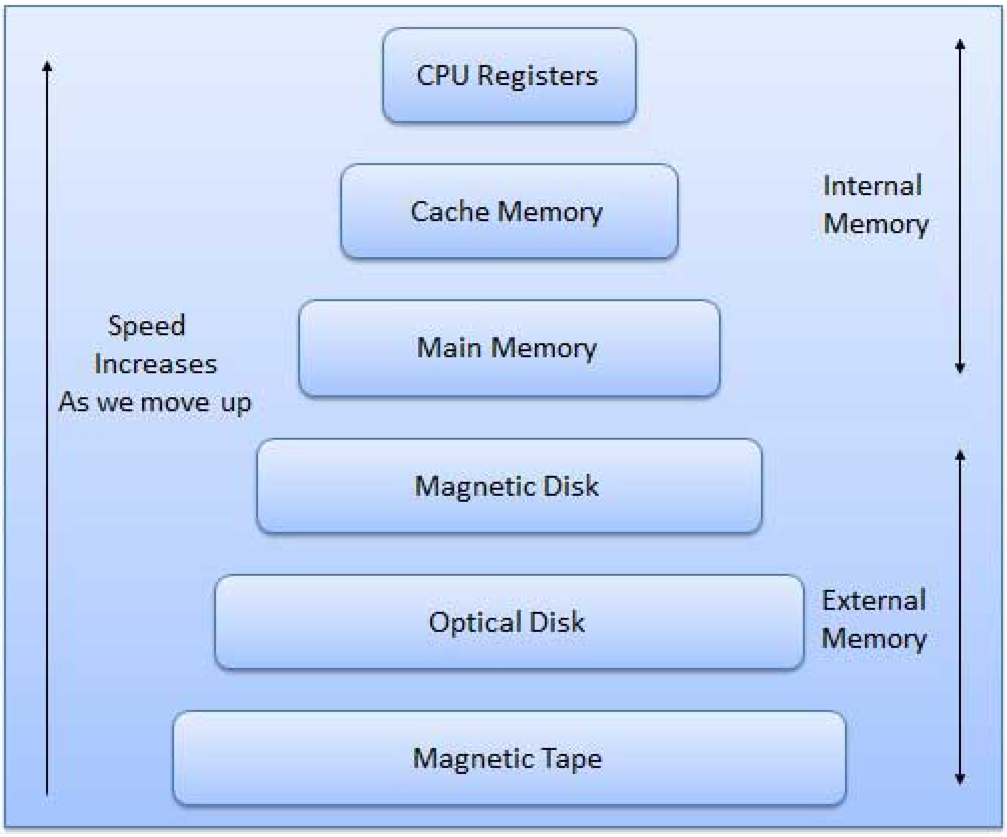
The memory is divided into large number of small parts. Each part is called a cell. Each location or cell has a unique address which varies from zero to memory size minus one.

For example if computer has 64k words, then this memory unit has 64 * 1024 = 65536 memory location. The address of these locations varies from 0 to 65535.

Memory is primarily of two types

- **Internal Memory** − cache memory and primary/main memory

- **External Memory** − magnetic disk / optical disk etc.



Characteristics of Memory Hierarchy are following when we go from top to bottom.

- Capacity in terms of storage increases.
- Cost per bit of storage decreases.
- Frequency of access of the memory by the CPU decreases.
- Access time by the CPU increases.

RAM

A RAM constitutes the internal memory of the CPU for storing data, program and program result. It is read/write memory. It is called random access memory *RAM*.

Since access time in RAM is independent of the address to the word that is, each storage location inside the memory is as easy to reach as other location & takes the same amount of time. We can reach into the memory at random & extremely fast but can also be quite expensive.

RAM is volatile, i.e. data stored in it is lost when we switch off the computer or if there is a power failure. Hence, a backup uninterruptible power system *UPS* is often used with computers. RAM is small, both in terms of its physical size and in the amount of data it can hold.

RAM is of two types

- Static RAM *SRAM*
- Dynamic RAM *DRAM*

Static RAM *SRAM*

The word **static** indicates that the memory retains its contents as long as power remains applied. However, data is lost when the power gets down due to volatile nature. SRAM chips use a matrix of 6-transistors and no capacitors. Transistors do not require power to prevent leakage, so SRAM need not have to be refreshed on a regular basis.

Because of the extra space in the matrix, SRAM uses more chips than DRAM for the same amount of storage space, thus making the manufacturing costs higher.

Static RAM is used as cache memory needs to be very fast and small.

Dynamic RAM *DRAM*

DRAM, unlike SRAM, must be continually **refreshed** in order for it to maintain the data. This is done by placing the memory on a refresh circuit that rewrites the data several hundred times per second. DRAM is used for most system memory because it is cheap and small. All DRAMs are made up of memory cells. These cells are composed of one capacitor and one transistor.

ROM

ROM stands for Read Only Memory. The memory from which we can only read but cannot write on it. This type of memory is non-volatile. The information is stored permanently in such memories during manufacture.

A ROM, stores such instruction as are required to start computer when electricity is first turned on, this operation is referred to as bootstrap. ROM chip are not only used in the computer but also in other electronic items like washing machine and microwave oven.

Following are the various types of ROM −

MROM *MaskedROM*

The very first ROMs were hard-wired devices that contained a pre-programmed set of data or instructions. These kind of ROMs are known as masked ROMs. It is inexpensive ROM.

PROM *ProgrammableReadOnlyMemory*

PROM is read-only memory that can be modified only once by a user. The user buys a blank PROM and enters the desired contents using a PROM programmer. Inside the PROM chip there are small fuses which are burnt open during programming. It can be programmed only once and is not erasable.

EPROM *ErasableandProgrammableReadOnlyMemory*

The EPROM can be erased by exposing it to ultra-violet light for a duration of upto 40 minutes. Usually, an EPROM eraser achieves this function. During programming an electrical charge is trapped in an insulated gate region. The charge is retained for more than ten years because the charge has no leakage path. For erasing this charge, ultra-violet light is passed through a quartz crystal window *lid*. This exposure to ultra-violet light dissipates the charge. During normal use the quartz lid is sealed with a sticker.

EEPROM *ElectricallyErasableandProgrammableReadOnlyMemory*

The EEPROM is programmed and erased electrically. It can be erased and reprogrammed about ten thousand times. Both erasing and programming take about 4 to 10 ms *millisecond*. In EEPROM, any location can be selectively erased and programmed. EEPROMs can be erased one byte at a time, rather than erasing the entire chip. Hence, the process of re-programming is flexible but slow.

Serial Access Memory

Sequential access means the system must search the storage device from the beginning of the memory address until it finds the required piece of data. Memory device which supports such access is called a Sequential Access Memory or Serial Access Memory. Magnetic tape is an example of serial access memory.

Direct Access Memory

Direct access memory or Random Access Memory, refers to conditions in which a system can go directly to the information that the user wants. Memory device which supports such access is called a Direct Access Memory. Magnetic disks, optical disks are examples of direct access memory.

Cache Memory

Cache memory is a very high speed semiconductor memory which can speed up CPU. It acts as a buffer between the CPU and main memory. It is used to hold those parts of data and program which are most frequently used by CPU. The parts of data and programs, are transferred from disk to cache memory by operating system, from where CPU can access them.

Advantages

- Cache memory is faster than main memory.
- It consumes less access time as compared to main memory.
- It stores the program that can be executed within a short period of time.
- It stores data for temporary use.

Disadvantages

- Cache memory has limited capacity.
- It is very expensive.

Virtual memory is a technique that allows the execution of processes which are not completely available in memory. The main visible advantage of this scheme is that programs can be larger than physical memory. Virtual memory is the separation of user logical memory from physical memory.

This separation allows an extremely large virtual memory to be provided for programmers when only a smaller physical memory is available. Following are the situations, when entire program is not required to be loaded fully in main memory.

- User written error handling routines are used only when an error occurred in the data or computation.
- Certain options and features of a program may be used rarely.
- Many tables are assigned a fixed amount of address space even though only a small amount of the table is actually used.
- The ability to execute a program that is only partially in memory would counter many benefits.
- Less number of I/O would be needed to load or swap each user program into memory.
- A program would no longer be constrained by the amount of physical memory that is available.
- Each user program could take less physical memory, more programs could be run the same time, with a corresponding increase in CPU utilization and throughput.

Auxiliary Memory

Auxiliary memory is much larger in size than main memory but is slower. It normally stores system programs, instruction and data files. It is also known as secondary memory. It can also be used as an overflow/virtual memory in case the main memory capacity has been exceeded. Secondary memories cannot be accessed directly by a processor. First the data/information of auxiliary memory is transferred to the main memory and then that information can be accessed by the CPU. Characteristics of Auxiliary Memory are following −

- **Non-volatile memory** − Data is not lost when power is cut off.
- **Reusable** − The data stays in the secondary storage on permanent basis until it is not overwritten or deleted by the user.

- **Reliable** − Data in secondary storage is safe because of high physical stability of secondary storage device.
- **Convenience** − With the help of a computer software, authorised people can locate and access the data quickly.
- **Capacity** − Secondary storage can store large volumes of data in sets of multiple disks.
- **Cost** − It is much lesser expensive to store data on a tape or disk than primary memory.

CPU ARCHITECTURE

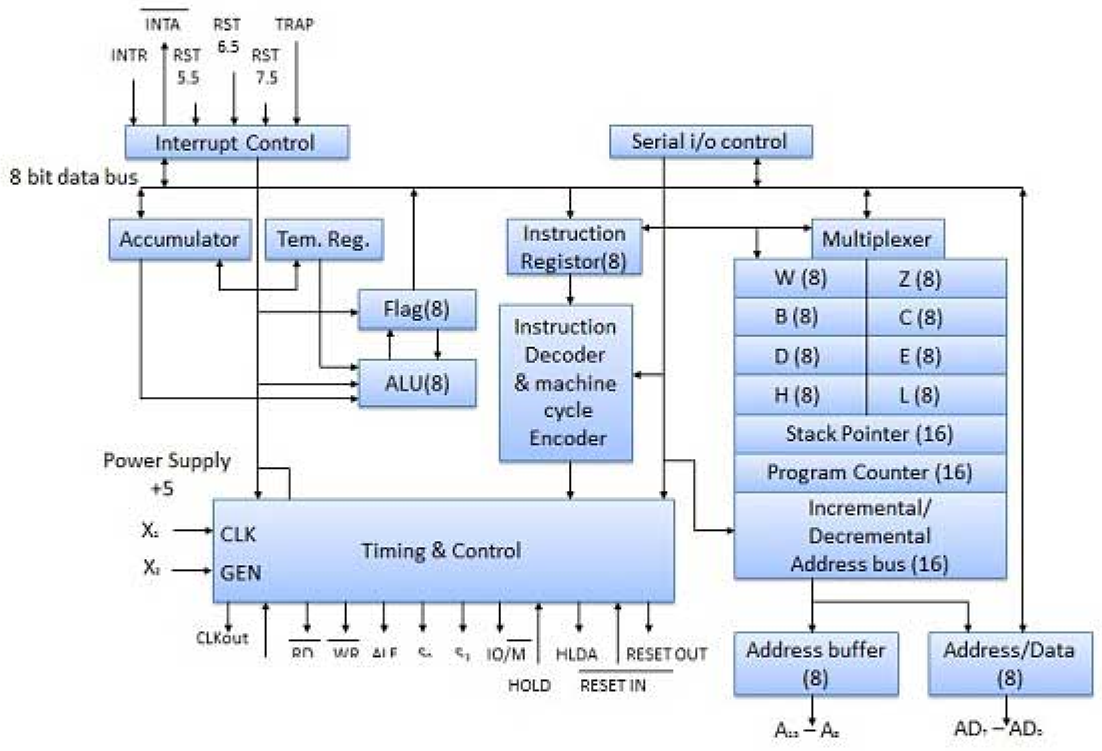
Microprocessing unit is synonymous to central processing unit, CPU used in traditional computer. Microprocessor *MPU* acts as a device or a group of devices which do the following tasks.

- communicate with peripherals devices
- provide timing signal
- direct data flow
- perform computer tasks as specified by the instructions in memory

8085 Microprocessor

The 8085 microprocessor is an 8-bit general purpose microprocessor which is capable to address 64k of memory. This processor has forty pins, requires &plus;5 V single power supply and a 3-MHz single-phase clock.

Block Diagram



ALU

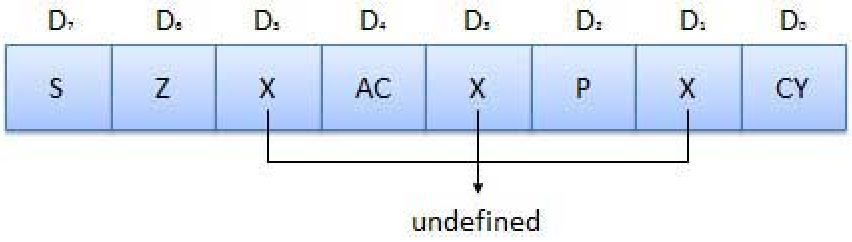
The ALU perform the computing function of microprocessor. It includes the accumulator, temporary register, arithmetic & logic circuit & and five flags. Result is stored in accumulator & flags.

Block Diagram

Accumulator

It is an 8-bit register that is part of ALU. This register is used to store 8-bit data & in performing arithmetic & logic operation. The result of operation is stored in accumulator.

Diagram



Flags

Flags are programmable. They can be used to store and transfer the data from the registers by using instruction. The ALU includes five flip-flops that are set and reset according to data condition in accumulator and other registers.

- **S** *Sign* **flag** − After the execution of an arithmetic operation, if bit D_7 of the result is 1, the sign flag is set. It is used to signed number. In a given byte, if D_7 is 1 means negative number. If it is zero means it is a positive number.
- **Z** *Zero* **flag** − The zero flag is set if ALU operation result is 0.
- **AC** *AuxiliaryCarry* **flag** − In arithmetic operation, when carry is generated by digit D3 and passed on to digit D_4, the AC flag is set. This flag is used only internally BCD operation.
- **P** *Parity* **flag** − After arithmetic or logic operation, if result has even number of 1s, the flag is set. If it has odd number of 1s, flag is reset.
- **C** *Carry* **flag** − If arithmetic operation result is in a carry, the carry flag is set, otherwise it is reset.

Register section

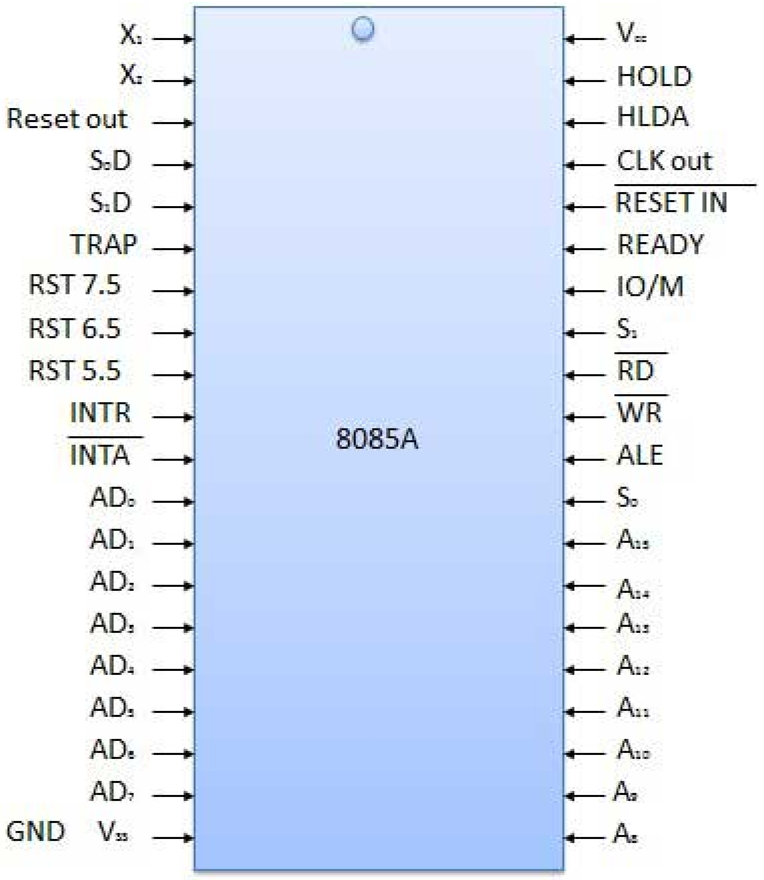
It is basically a storage device and transfers data from registers by using instructions.

- **Stack Pointer** *SP* − The stack pointer is also a 16-bit register which is used as a memory pointer. It points to a memory location in Read/Write memory known as stack. In between execution of program, sometime data to be stored in stack. The beginning of the stack is defined by loading a 16-bit address in the stack pointer.
- **Program Counter** *PC* − This 16-bit register deals with fourth operation to sequence the execution of instruction. This register is also a memory pointer. Memory location have 16-bit address. It is used to store the execution address. The function of the program counter is to point to memory address from which next byte is to be fetched.
- **Storage registers** − These registers store 8-bit data during a program execution. These registers are identified as B, C, D, E, H, L. They can be combined as register pair BC, DE and HL to perform some 16 bit operations.

Time and Control Section

This unit is responsible to synchronize Microprocessor operation as per the clock pulse and to generate the control signals which are necessary for smooth communication between Microprocessor and peripherals devices. The RD bar and WR bar signals are synchronous pulses which indicates whether data is available on the data bus or not. The control unit is responsible to control the flow of data between microprocessor, memory and peripheral devices.

PIN diagram



All the signal can be classified into six groups

S.N.	Group	Description
1	**Address bus**	The 8085 microprocessor has 8 signal line, A_{15} - A_8 which are uni directional and used as a high order address bus.
2	**Data bus**	The signal line AD7 - AD0 are bi-directional for dual purpose. They are used as low order address bus as well as data bus. Control Signal **RD bar** − It is a read control signal *activelow*. If it is active then memory read the data. **WR bar** − It is write control signal *activelow*. It is active when written into selected memory. Status signal **ALU** ***AddressLatchEnable*** − When ALU is high. 8085 microprocessor use address bus. When

3	**Control signal and Status signal**	ALU is low. 8085 microprocessor is use data bus. **IO/M bar** − This is a status signal used to differentiate between i/o and memory operations. When it is high, it indicate an i/o operation and when it is low, it indicate memory operation. **S_1 and S_0** − These status signals, similar to i/o and memory bar, can identify various operations, but they are rarely used in small system.
4	**Power supply and frequency signal**	**V_{cc}** − &plus;5v power supply. **V_{ss}** − ground reference. **X, X** − A crystal is connected at these two pins. The frequency is internally divided by two operate system at 3-MHz, the crystal should have a frequency of 6-MHz. **CLK out** − This signal can be used as the system clock for other devices.
5	**Externally initiated signal**	**INTR** i/p − Interrupt request. **INTA bar** o/p − It is used as acknowledge interrupt. **TRAP** i/p − This is non maskable interrupt and has highest priority. **HOLD** i/p − It is used to hold the executing program. **HLDA** o/p − Hold acknowledge. **READY** i/p − This signal is used to delay the microprocessor read or write cycle until a slow responding peripheral is ready to accept or send data. **RESET IN bar** − When the signal on this pin goes low, the program counter is set to zero, the bus are tri-stated, & MPU is reset. **RESET OUT** − This signal indicate that MPU is being reset. The signal can be used to reset other devices. **RST 7.5, RST 6.5, RST 5.5** *Requestinterrupt* − It is used to transfer the program control to specific memory location. They have higher priority than INTR interrupt.
6	**Serial I/O ports**	The 8085 microprocessor has two signals to implement the serial transmission serial input data and serial output data.

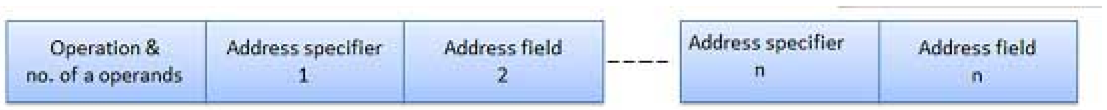
Instruction Format

Each instruction is represented by a sequence of bits within the computer. The instruction is divided into group of bits called field. The way instruction is expressed is known as instruction format. It is usually represented in the form of rectangular box. The instruction format may be of the following types.

Variable Instruction Formats

These are the instruction formats in which the instruction length varies on the basis of opcode & address specifiers. For Example, VAX instruction vary between 1 and 53 bytes while X86 instruction vary between 1 and 17 bytes.

Format



Advantage

These formats have good code density.

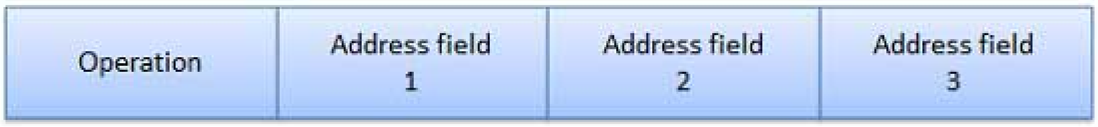
Drawback

These instruction formats are very difficult to decode and pipeline.

Fixed Instruction Formats

In this type of instruction format, all instructions are of same size. For Example, MIPS, Power PC, Alpha, ARM.

Format



Advantage

They are easy to decode & pipeline.

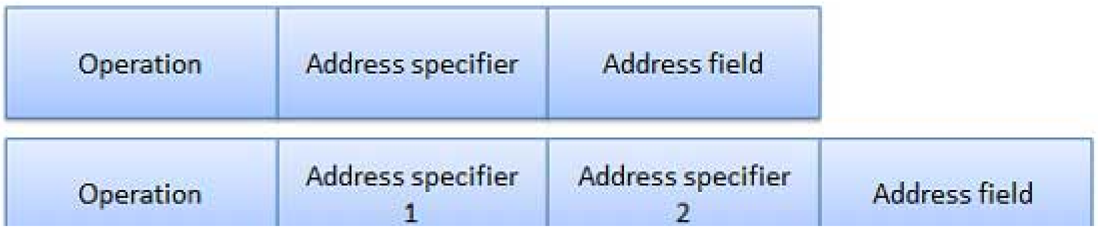
Drawback

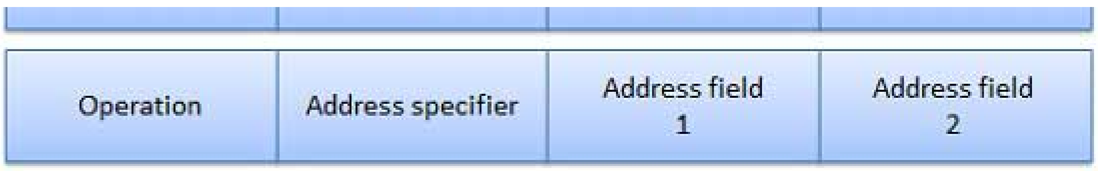
They don't have good code density.

Hybrid Instruction Formats

In this type of instruction formats, we have multiple format length specified by opcode. For example, IBM 360/70, MIPS 16, Thumb.

Format





Advantage

These compromise between code density & instruction of these type are very easy to decode.

Addressing Modes

Addressing mode provides different ways for accessing an address to given data to a processor. Operated data is stored in the memory location, each instruction required certain data on which it has to operate. There are various techniques to specify address of data. These techniques are called Addressing Modes.

- **Direct addressing mode** − In the direct addressing mode, address of the operand is given in the instruction and data is available in the memory location which is provided in instruction. We will move this data in desired location.
- **Indirect addressing mode** − In the indirect addressing mode, the instruction specifies a register which contain the address of the operand. Both internal RAM and external RAM can be accessed via indirect addressing mode.
- **Immediate addressing mode** − In the immediate addressing mode, direct data is given in the operand which move the data in accumulator. It is very fast.
- **Relative addressing mode** − In the relative address mode, the effective address is determined by the index mode by using the program counter in stead of general purpose processor register. This mode is called relative address mode.
- **Index addressing mode** − In the index address mode, the effective address of the operand is generated by adding a content value to the contents of the register. This mode is called index address mode.