DIGITAL SYSTEMS

- **Number System (binary, hexa, octal, complements) codes (ASCII, UNICODE, BCD, GRAY), Error detecting and correcting code – parity and Hamming codes, Boolean algebra & Laws.**
- **Combinational circuits - SOP & POS form K-Map - encoders, Decoders, multiplexers, demultiplexers - sequential circuits - flip-flops, registers & counters.**
- **Integer representation (signed & unsigned). Half and full adder, sequential multiplier, Booth algorithm - floating point representation**

DIGITAL SYSTEMS

- Digital systems are designed to store, process, and communicate information in digital form.
- They are found in a wide range of applications, including process control, communication systems, digital instruments, and consumer products.
- The digital computer, more commonly called the computer, is an example of a typical digital system.
- A computer manipulates information in digital, or more precisely, binary form.
- A binary number has only two discrete values — zero or one.
- Each of these discrete values is represented by the OFF and ON status of an electronic switch called a transistor.
- All computers, therefore, only understand binary numbers.
- Any decimal number (base 10, with ten digits from 0 to 9) can be represented by a binary number (base 2, with digits 0 and 1).
- Number system is important from the viewpoint of understanding how data are represented before they can be processed by any digital system including a digital computer.
- There are two basic ways of representing the numerical values of the various physical quantities with which we constantly deal in our day to day lives.
- The arithmetic value which is used for representing the quantity and used in making calculations is defined as NUMBERS.
- A symbol like “4, 5, 6” which represents a number is known as numerals.
- Without numbers, counting things is not possible, date, time, money, etc. these numbers are also used for measurement and used for labeling.
- The properties of numbers make them helpful in performing arithmetic operations on them.
- These numbers can be written in numeric forms and also in words.

For example,

- 3 is written as three in words, 35 is written as thirty-five in words, etc.
- Students can write the numbers from 1 to 100 in words to learn more.
- There are different types of numbers, which we can learn.
- They are whole and natural numbers, odd and even numbers, rational and irrational numbers, etc.

ENTRI

NUMBER AND ITS TYPES

- Numbers used in mathematics are mostly decimal number systems.
- In the decimal number system, digits used are from 0 to 9 and base 10 is used.

There are many types of numbers in decimal number system, below are some of the types of numbers mentioned,

- Numbers that are represented on the right side of the zero are termed **Positive Numbers**. The value of these numbers increases on moving towards the right. Positive numbers are used for Addition between numbers. Example: 1, 2, 3, 4.

- Numbers that are represented on the left side of the zero are termed **Negative Numbers**. The value of these numbers decreases on moving towards the left. Negative numbers are used for Subtraction between numbers. Example: -1, -2, -3, -4.

- **Natural Numbers** are the most basic type of Numbers that range from 1 to infinity. These numbers are also called Positive Numbers or Counting Numbers. Natural Numbers are represented by the symbol N.

- **Whole Numbers** are basically the Natural Numbers, but they also include 'zero'. Whole numbers are represented by the symbol W.

- **Integers** are the collection of Whole Numbers plus the negative values of the Natural Numbers. Integers do not include fraction numbers i.e. they can't be written in a/b form. The range of Integers is from the Infinity at the Negative end and Infinity at the Positive end, including zero. Integers are represented by the symbol Z.

- **Rational numbers** are the numbers that can be represented in the fraction form i.e. a/b. Here, a and b both are integers and $b \neq 0$. All the fractions are rational numbers but not all the rational numbers are fractions.

- **Irrational numbers** are the numbers that can't be represented in the form of fractions i.e. they cannot be written as a/b.

- Numbers that do not have any factors other than 1 and the number itself are termed as **Prime Numbers**. All the numbers other than Prime Numbers are termed as **Composite Numbers** except 0. Zero is neither prime nor a composite number.

NUMBER SYSTEM

- A Number system is a method of showing numbers by writing, which is a mathematical way of representing the numbers of a given set, by using the numbers or symbols in a mathematical manner.
- The writing system for denoting numbers using digits or symbols in a logical manner is defined as a Number system.

- The numeral system Represents a useful set of numbers, reflects the arithmetic and algebraic structure of a number, and Provides standard representation.
- The digits from 0 to 9 can be used to form all the numbers.
- With these digits, anyone can create infinite numbers.
- For example, 156, 3907, 3456, 1298, 784859 etc.

TYPES OF NUMBER SYSTEMS

- Based on the base value and the number of allowed digits, number systems are of many types.
- The four common types of Number System are:

✓ ***Decimal Number System***
✓ ***Binary Number System***
✓ ***Octal Number System***
✓ ***Hexadecimal Number System***

➢ *Decimal Number System*

- Number system with a base value of 10 is termed a Decimal number system.
- It uses 10 digits i.e. 0-9 for the creation of numbers.
- Here, each digit in the number is at a specific place with place value a product of different powers of 10.
- Here, the place value is termed from right to left as first place value called units, second to the left as Tens, so on Hundreds, Thousands, etc
- . Here, units have the place value as 100, tens have the place value as 101, hundreds as 102, thousands as 103, and so on.

For example, 10264 has place values as,

$$(1 \times 10^4) + (0 \times 10^3) + (2 \times 10^2) + (6 \times 10^1) + (4 \times 10^0)$$

$$= 1 \times 10000 + 0 \times 1000 + 2 \times 100 + 6 \times 10 + 4 \times 1$$

$$= 10000 + 0 + 200 + 60 + 4$$

$$= 10264$$

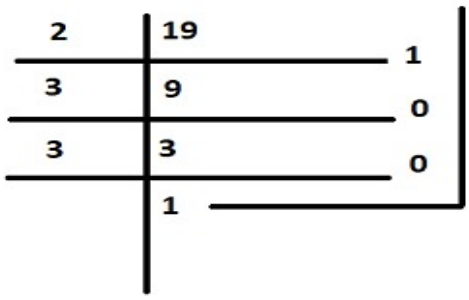
➢ *Binary Number System*

- Number System with base value 2 is termed as Binary number system.
- It uses 2 digits i.e. 0 and 1 for the creation of numbers.
- The numbers formed using these two digits are termed Binary Numbers.
- The binary number system is very useful in electronic devices and computer systems because it can be easily performed using just two states ON and OFF i.e. 0 and 1.

- Decimal Numbers 0-9 are represented in binary as: 0, 1, 10, 11, 100, 101, 110, 111, 1000, and 1001

- For example, 14 can be written as 1110, 19 can be written as 10011, 50 can be written as 110010.

Example of 19 in the binary system



Here 19 can be written as 10011

ADVANTAGES

- Logic operations are the backbone of any digital computer, although solving a problem on computer could involve an arithmetic operation too.
- The introduction of the mathematics of logic by George Boole laid the foundation for the modern digital computer.
- He reduced the mathematics of logic to a binary notation of '0' and '1'.

- Another advantage of this number system was that all kind of data could be conveniently represented in terms of 0s and 1s.

- Also basic electronic devices used for hardware implementation could be conveniently and efficiently operated in two distinctly different modes.

➢ *Octal Number System*

- Octal Number System is one in which the base value is 8.
- It uses 8 digits i.e. 0-7 for the creation of Octal Numbers.
- Octal Numbers can be converted to Decimal values by multiplying each digit with the place value and then adding the result.
- Here the place values are 80, 81, and 82.
- Octal Numbers are useful for the representation of UTF8 Numbers.

Example,

$(135)_{10}$ can be written as $(207)_8$

$(215)_{10}$ can be written as $(327)_8$

➢ *Hexadecimal Number System*

- Number System with base value 16 is termed as Hexadecimal Number System.
- It uses 16 digits for the creation of its numbers.
- Digits from 0-9 are taken like the digits in the decimal number system but the digits from 10-15 are represented as A-F i.e.
- 10 is represented as A, 11 as B, 12 as C, 13 as D, 14 as E, and 15 as F.
- Hexadecimal Numbers are useful for handling memory address locations.
- The hexadecimal number system provides a condensed way of representing large binary numbers stored and processed.

Examples,

$(255)_{10}$ can be written as $(FF)_{16}$

$(1096)_{10}$ can be written as $(448)_{16}$

$(4090)_{10}$ can be written as $(FFA)_{16}$

Hexadecimal	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
Decimal	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

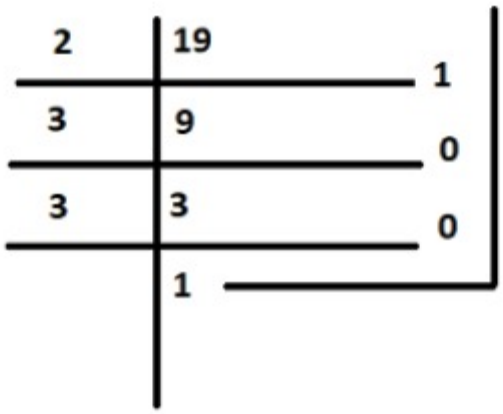
Sample Problems

Question 1: Convert (18)10 as a binary number?

Solution:

Question 2: Convert 325_8 into a decimal?

Solution:



$325_8 = 3 \times 8^2 + 2 \times 8^1 + 5 \times 8^0$

$= 3 \times 64 + 2 \times 8 + 5 \times 1$

$= 192 + 16 + 5$

$= 213_{10}$

Question 3: Convert (2056)16 into an octal number?

Solution:

Here $(2056)_{16}$ is in hexadecimal form

First we will convert into decimal form from hexadecimal.

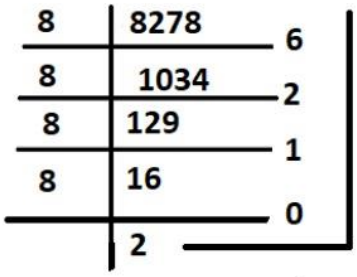
$$(2056)_{16} = 2 \times 16^3 + 0 \times 16^2 + 5 \times 16^1 + 6 \times 16^0$$

$$= 2 \times 4096 + 0 + 80 + 6$$

$$= 8192 + 0 + 80 + 6$$

$$= (8278)_{10}$$

Now convert this decimal number into octal number by dividing it by 8



So will take the value of remainder from 20126

$(8278)_{10} = (20126)_8$

Therefore, $(2056)_{16} = (20126)_8$

Question 4: Convert (101110)2 into octal number.

Solution:

Given $(101110)_2$ a binary number, to convert it into octal number

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

Using the above table we can write given number as,

101 110 i.e.

101 = 5

110 = 6

So $(101110)_2$ in octal number is $(56)_8$

BINARY REPRESENTATION

- Binary is a base-2 number system that uses two states 0 and 1 to represent a number.
- We can also call it to be a true state and a false state.
- A binary number is built the same way as we build the normal decimal number.

- For example, a decimal number 45 can be represented as 4*10^1+5*10^0 = 40+5

- Now in binary 45 is represented as 101101.

- As we have powers of 10 in decimal number similarly there are powers of 2 in binary numbers.

- Hence 45 which is 101101 in binary can be represented as:

  2^0*1+2^1*0+2^2*1+2^3*1+2^4*0+2^5*1 = 45

- The binary number is traversed from left to right.

Sign and Magnitude representation –

- There are many ways for representing negative integers.
- One of the ways is sign-magnitude.
- This system uses one bit to indicate the sign.
- Mathematical numbers are generally made up of a sign and a value.
- The sign indicates whether the number is positive, (+) or negative, (–) while the value indicates the size of the number.
- For example 13, +256 or -574.
- Presenting numbers in this way is called sign-magnitude representation since the left most digits can be used to indicate the sign and the remaining digits the magnitude or value of the number.

- Sign-magnitude notation is the simplest and one of the most common methods of representing positive and negative numbers.
- Thus negative numbers are obtained simply by changing the sign of the corresponding positive number, for example, +2 and -2, +10 and -10, etc.
- Similarly adding a 1 to the front of a binary number is negative and a 0 makes it positive.

- For example 0101101 represent +45 and 1101101 represents -45 if 6 digits of a binary number are considered and the leftmost digit represents the sign.

- But a problem with the sign-magnitude method is that it can result in the possibility of two different bit patterns having the same binary value.

- For example, +0 and -0 would be 0000 and 1000 respectively as a signed 4-bit binary number.

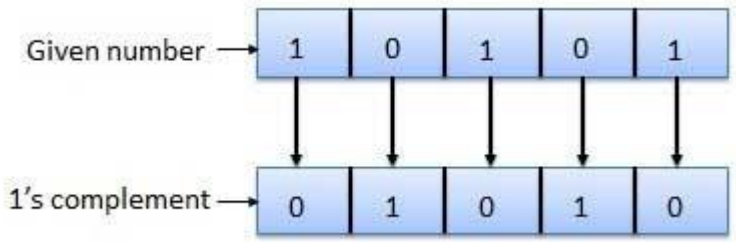
- So using this method there can be two representations for zero, a positive zero 0000 and also a negative zero 1000 which can cause big complications for computers and digital systems.

The two complement notations used to represent signed magnitude numbers are:

- ✓ **One's complement**
- ✓ **Two's complement**

✓ **<u>One's complement</u>**

- 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.



1's Complement Table

Binary Number	1's Complement
0000	1111
0001	1110
0010	1101
0011	1100
0100	1011
0101	1010
0110	1001
0111	1000
1000	0111
1001	0110
1010	0101
1011	0100
1100	0011
1101	0010
1110	0001

1111	0000

Use of 1's complement

- 1's complement plays an important role in representing the signed binary numbers.
- The main use of 1's complement is to represent a signed binary number.
- Apart from this, it is also used to perform various arithmetic operations such as addition and subtraction.

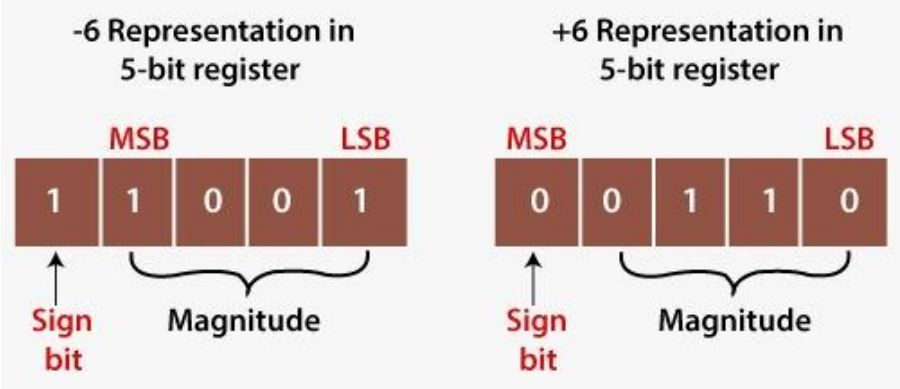
- In signed binary number representation, we can represent both positive and negative numbers.
- For representing the positive numbers, there is nothing to do.
- But for representing negative numbers, we have to use 1's complement technique.
- For representing the negative number, we first have to represent it with a positive sign, and then we find the 1's complement of it.

- Let's take an example of a positive and negative number and see how these numbers are represented.

Example 1: +6 and -6

- The number +6 is represented as same as the binary number.
- For representing both numbers, we will take the 5-bit register.
- So the +6 is represented in the 5-bit register as 0 0110.

- The -6 is represented in the 5-bit register in the following way:
  1. +6=0 0110
  2. Find the 1's complement of the number 0 0110, i.e., 1 1001. Here, MSB denotes that a number is a negative number.



- Here, MSB refers to Most Significant Bit, and LSB denotes the Least Significant Bit.

Example 2: +120 and -120

- The number +120 is represented as same as the binary number.
- For representing both numbers, take the 8-bit register.
- So the +120 is represented in the 8-bit register as 0 1111000.

- The -120 is represented in the 8-bit register in the following way:

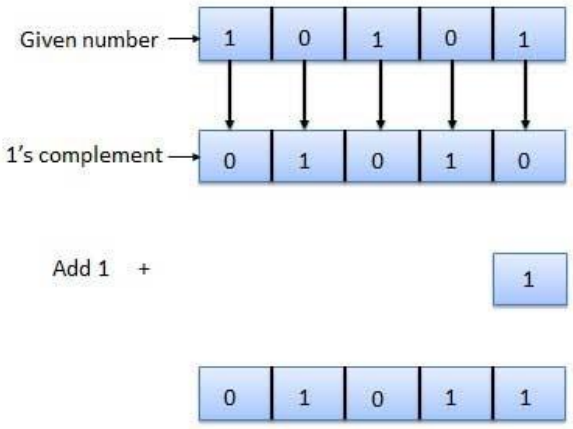
1. +120=0 1111000
2. Now, find the 1's complement of the number 0 1111000, i.e., 1 0000111. Here, the MSB denotes the number is the negative number.

✓ **Two'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 + 1

- Example of 2's Complement is as follows.



2's Complement Table

Binary Number	1's Complement	2's complement
0000	1111	0000
0001	1110	1111
0010	1101	1110
0100	1011	1100
0101	1010	1011
0110	1001	1010

Use of 2's complement

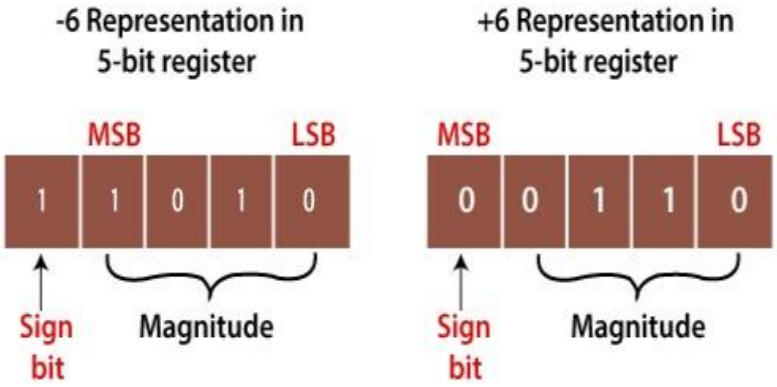
- 2's complement is used for representing signed numbers and performing arithmetic operations such as subtraction, addition, etc.
- The positive number is simply represented as a magnitude form. So there is nothing to do for representing positive numbers.
- But if we represent the negative number, then we have to choose either 1's complement or 2's complement technique.
- The 1's complement is an ambiguous technique, and 2's complement is an unambiguous technique.
- Let's see an example to understand how we can calculate the 2's complement in signed binary number representation.

Example 1: +6 and -6

- The number +6 is represented as same as the binary number. For representing both numbers, take the 5-bit register.
- So the +6 is represented in the 5-bit register as 0 0110.

- The -6 is represented in the 5-bit register in the following way:

1. +6=0 0110
2. Now, find the 1's complement of the number 0 0110, i.e. 1 1001.
3. Now, add 1 to its LSB. When we add 1 to the LSB of 11001, the newly generated number comes out 11010. Here, the sign bit is one which means the number is the negative number.



Example 2: +120 and -120

- The number +120 is represented as same as the binary number. For representing both numbers, take the 8-bit register.

- So the +120 is represented in the 8-bit register as 0 1111000.

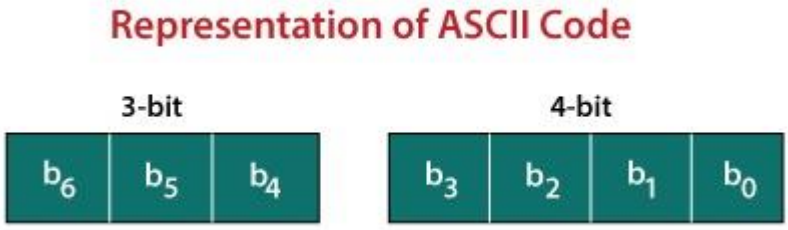
- The -120 is represented in the 8-bit register in the following way:

1. +120=0 1111000
2. Now, find the 1's complement of the number 0 1111000, i.e. 1 0000111. Here, the MSB denotes the number is the negative number.

3. Now, add 1 to its LSB. When we add 1 to the LSB of 1 0000111, the newly generated number comes out 1 0001000. Here, the sign bit is one, which means the number is the negative number.

ASCII Code

- The ASCII stands for American Standard Code for Information Interchange.
- The ASCII code is an alphanumeric code used for data communication in digital computers.
- The ASCII is a 7-bit code capable of representing 27 or 128 number of different characters.
- The ASCII code is made up of a three-bit group, which is followed by a four-bit code.



- The ASCII Code is a 7 or 8-bit alphanumeric code.
- This code can represent 127 unique characters.
- The ASCII code starts from 00h to 7Fh.
- In this, the code from 00h to 1Fh is used for control characters, and the code from 20h to 7Fh is used for graphic symbols.
- The 8-bit code holds ASCII, which supports 256 symbols where math and graphic symbols are added.
- The range of the extended ASCII is 80h to FFh.
- The ASCII characters are classified into the following groups:

1. Control Characters

- The non-printable characters used for sending commands to the PC or printer are known as control characters.
- We can set tabs, and line breaks functionality by this code.
- The control characters are based on telex technology.
- Nowadays, it's not so much popular in use.
- The character from 0 to 31 and 127 comes under control characters.

2. **Special Characters**

- All printable characters that are neither numbers nor letters come under the special characters.
- These characters contain technical, punctuation, and mathematical characters with space also.
- The character from 32 to 47, 58 to 64, 91 to 96, and 123 to 126 comes under this category.

3. **Numbers Characters**

- This category of ASCII code contains ten Arabic numerals from 0 to 9.

4. **Letters Characters**

- In this category, two groups of letters are contained, i.e., the group of uppercase letters and the group of lowercase letters.
- The range from 65 to 90 and 97 to 122 comes under this category.

ASCII Table

The values are typically represented in ASCII code tables in decimal, binary, and hexadecimal form.

Binary	Hexa decimal	Decimal	ASCII Symbol	Description	Group
0000000	0	0	NUL	The null character encourage the device to do nothing	Control Character
0000001	1	1	SOH	The symbol SOH (Starts of heading) Initiates the header.	Control Character
0000010	2	2	STX	The symbol STX (Start of Text) ends the header and marks the beginning of a message.	Control Character
0000011	3	3	ETX	The symbol ETX (End of Text) indicates the end of the message.	Control Character
0000100	4	4	EOT	The EOT(end of text) symbol marks the end	Control Character

				of a completes transmission	
0000101	5	5	ENQ	The ENQ(Enquiry) symbol is a request that requires a response	Control Character
0000110	6	6	ACK	The ACK (Acknowledge) symbol is a positive answer to the request.	Control Character
0000111	7	7	BEL	The BEL (Bell) symbol triggers a beep.	Control Character
0001000	8	8	BS	Lets the cursor move back one step (Backspace)	Control Character
0001001	9	9	TAB (HT)	A horizontal tab that moves the cursor within a row to the next predefined position (Horizontal Tab)	Control Character
0001010	A	10	LF	Causes the cursor to jump to the next line (Line Feed)	Control Character
0001011	B	11	VT	The vertical tab lets the cursor jump to a predefined line (Vertical Tab)	Control Character
0001100	C	12	FF	Requests a page break (Form Feed)	Control Character
0001101	D	13	CR	Moves the cursor back to the first position of the line	Control Character

				(Carriage Return)	
0001110	E	14	SO	Switches to a special presentation (Shift Out)	Control Character
0001111	F	15	SI	Switches the display back to the normal state (Shift In)	Control Character
0010000	10	16	DLE	Changes the meaning of the following characters (Data Link Escape)	Control Character
0001011	B	11	VT	The vertical tab lets the cursor jump to a predefined line (Vertical Tab)	Control Character
0001100	C	12	FF	Requests a page break (Form Feed)	Control Character
0001101	D	13	CR	Moves the cursor back to the first position of the line (Carriage Return)	Control Character
0001110	E	14	SO	Switches to a special presentation (Shift Out)	Control Character
0001111	F	15	SI	Switches the display back to the normal state (Shift In)	Control Character
0010000	10	16	DLE	Changes the meaning of the following characters (Data Link Escape)	Control Character
0010001	11	17	DC1	Control characters assigned depending on the device	Control Character

				used (Device Control)	
0010010	12	18	DC2	Control characters assigned depending on the device used (Device Control)	Control Character
0010011	13	19	DC3	Control characters assigned depending on the device used (Device Control)	Control Character
0011010	1A	26	SUB	Replacement for a faulty sign (Substitute)	Control Character
0011011	1B	27	ESC	Initiates an escape sequence and thus gives the following characters a special meaning (Escape)	Control Character
0011100	1C	28	FS	File separator.	Control Character
0011101	1D	29	GS	Group separator.	Control Character
0011110	1E	30	RS	Record separator.	Control Character
0011111	1F	31	US	Unit separator.	Control Character
0100000	20	32	SP	Blank space	Special Character
0100001	21	33	!	Exclamation mark	Special Character
0100010	22	34		Only quotes above	Special Character
0100011	23	35	#	Pound sign	Special Character
0100100	24	36	$	Dollar sign	Special Character
0100101	25	37	%	Percentage sign	Special Character
0100110	26	38	&	Commercial and	Special Character

0100111	27	39		Apostrophe	Special Character
0101000	28	40	(	Left bracket	Special Character
0101001	29	41	)	Right bracket	Special Character
0101010	2A	42	*	Asterisk	Special Character
0101011	2B	43	+	Plus symbol	Special Character
0101100	2C	44	,	Comma	Special Character
0101101	2D	45	-	Dash	Special Character
0101110	2E	46	.	Full stop	Special Character
0101111	2F	47	/	Forward slash	Special Character
0110000	30	48	0		Numbers
0110001	31	49	1		Numbers
0110010	32	50	2		Numbers
0110011	33	51	3		Numbers
0110100	34	52	4		Numbers
0110101	35	53	5		Numbers
0110110	36	54	6		Numbers
0110111	37	55	7		Numbers
0111000	38	56	8		Numbers
0111001	39	57	9		Numbers
0111010	3A	58	:	Colon	Special characters
0111011	3B	59	;	Semicolon	Special characters
0111100	3C	60	<	Small than bracket	Special characters
0111101	3D	61	=	Equals sign	Special characters
0111110	3E	62	>	Bigger than symbol	Special characters
0111111	3F	63	?	Question mark	Special characters
1000000	40	64	@	At symbol	Special characters
1000001	41	65	A		Capital letters
1100110	66	102	F		Lowercase letters
1100111	67	103	G		Lowercase letters
1101000	68	104	H		Lowercase letters

1101001	69	105	I		Lowercase letters
1101010	6A	106	J		Lowercase letters
1101011	6B	107	K		Lowercase letters
1101100	6C	108	L		Lowercase letters
1101101	6D	109	M		Lowercase letters
1101110	6E	110	N		Lowercase letters
1101111	6F	111	O		Lowercase letters
1110000	70	112	P		Lowercase letters
1110001	71	113	Q		Lowercase letters
1110010	72	114	R		Lowercase letters
1110011	73	115	S		Lowercase letters
1110100	74	116	T		Lowercase letters
1110101	75	117	U		Lowercase letters
1110110	76	118	v		Lowercase letters
1110111	77	119	w		Lowercase letters
1111000	78	120	x		Lowercase letters
1111001	79	121	y		Lowercase letters
1111010	7A	122	z		Lowercase letters
1111011	7B	123	{	Left curly bracket	Special characters
1111100	7C	124	l	Vertical line	Special characters
1111101	7D	125	}	Right curly brackets	Special characters
1111110	7E	126	~	Tilde	Special characters
1111111	7F	127	DEL	The DEL (Delete) symbol deletes a character. This is a control character that consists of the same number	Control c

				in all positions.	
1111000	78	120	x		Lowercase letters
1111001	79	121	y		Lowercase letters
1111010	7A	122	z		Lowercase letters
1111011	7B	123	{	Left curly bracket	Special characters
1111100	7C	124	l	Vertical line	Special characters
1111101	7D	125	}	Right curly brackets	Special characters
1111110	7E	126	~	Tilde	Special characters
1111111	7F	127	DEL	The DEL (Delete) symbol deletes a character. This is a control character that consists of the same number in all positions.	Control character

Example 1:

(1001010110000111101101100001101010011100001101111110100 1101110111010010000000110001011001001100 11)$_2$

Step 1: In the first step, we make the groups of 7-bits because the ASCII code is 7 bit.

1001010 1100001 1110110 1100001 1010100 1110000 1101111 1101001 1101110 1110100 1000000 0110001 0110010 0110011

Step 2: Then, we find the equivalent decimal number of the binary digits either from the ASCII table or 64 32 16 8 4 2 1 scheme.

Binary	Decimal
64 32 16 8 4 2 1 1 0 0 1 0 1 0	64+8+2=74
64 32 16 8 4 2 1 1 1 0 0 0 0 1	64+32+1=94
64 32 16 8 4 2 1 1 1 1 0 1 1 0	64+32+16+4+2=118
64 32 16 8 4 2 1 1 1 0 0 0 0 1	64+32+1=97
64 32 16 8 4 2 1	64+16+4=84

1 0 1 0 1 0 0	
64 32 16 8 4 2 1 1 1 1 0 0 0 0	64+32+16=112
64 32 16 8 4 2 1 1 1 0 1 1 1 1	64+32+8+4+2+1=111
64 32 16 8 4 2 1 1 1 0 1 0 0 1	64+32+8+1=105
64 32 16 8 4 2 1 1 1 0 1 1 1 0	64+32+8+4+2=110
64 32 16 8 4 2 1 1 1 1 0 1 0 0	64+32+16+4=116
64 32 16 8 4 2 1 1 0 0 0 0 0 0	64
64 32 16 8 4 2 1 0 1 1 0 0 0 1	32+16+1=49
64 32 16 8 4 2 1 0 1 1 0 0 1 0	32+16+2=50
64 32 16 8 4 2 1 0 1 1 0 0 1 1	32+16+2+1=51

Step 3: Last, we find the equivalent symbol of the decimal number from the ASCII table.

Decimal	**Symbol**
74	J
94	a
118	v
97	a
84	T
112	p
111	o
111	o
105	i
110	n
116	t
64	@
49	1
50	2
51	3

UNICODE

- Unicode is a universal encoding system to provide a comprehensive character set and was created by the Unicode Consortium (a group of multilingual software manufacturers).
- Unicode simplifies software localization and improves multilingual text processing.
- It overcomes the difficulty inherent in ASCII and extended ASCII.

- Unicode has standardizes script behavior which allows any combination of characters, drawn from any combination of scripts and languages, to co-exist in a single document.

- Unicode defines multiple encodings of its single character set: UTF-7, UTF-8, UTF-16, and UTF-32.

- Conversion of data among these encodings is lossless.

- Unicode was originally a 2-byte character set.

- Unicode version 3, however, is a 4-byte code and is fully compatible with ASCII and extended ASCII.

- These all support encoding the same set of characters.

  ✓ UTF-8 uses anywhere from 1 to 4 bytes per character depending on character, but ASCII take only 1 byte and 4 bytes for unusual ones.
  ✓ UTF-16 uses 2 bytes for most characters, while very unusual characters take 4.
  ✓ UTF-32 uses 4 bytes per character. We can calculate the number of characters in a UTF-32 string by only counting bytes.

- The notation uses hexadecimal digits in format as follows.

U-XXXXXXXX –

- The numbering goes from U-00000000 to U-FFFFFFFF.
- Unicode divides the available space codes into planes.
- A plane is a continuous group of 65,536 code points.
- The most significant 16 bits define the plane (i.e. number of planes = 65,535) and each plane can define up to 65,536 characters or symbols.

Types of Plane –

1. **Basic multilingual plane (BMP)** – Plane 0000, the basic multilingual plane is designed to be compatible with the previous 16-bit Unicode. The most significant 16-bits in this plane are all zeroes. It mostly defines character sets in different languages with the exception of some control and special characters. It is represented as U+XXXX where XXXX is the least significant 16-bits, eig.,: U+0900 to U+09FF reserved for Devanagari, Bengali U+2200 to U+22FF reserved for a mathematical operation etc.

2. **Supplementary multilingual plane (SMP)** – Plane 0001, the supplementary multilingual plane, is designed to provide more codes for those multilingual characters that are excluded in the BMP. Example: 10140-1018F is reserved for Ancient Greek Numbers.
3. **Supplementary ideography plane (SIP)** – Plane 0002, the supplementary ideography plane, is designed to provide codes for ideographic symbols, symbols that provide an idea in contrast to a sound, e.g., 20000-2A6DF are reserved for CJK Unified Extension B
4. **Supplementary special plane (SSP)** – 000E, the supplementary special plane, is used for special characters, e.g., E0000-E007F is reserved for tags.
5. **Private use planes (PUPs)** – Planes 000F and 0010, private use planes are for private use. They are used by fonts internally to refer to auxiliary glyphs.

BCD OR BINARY CODED DECIMAL

- Binary Coded Decimal, or BCD, is another process for converting decimal numbers into their binary equivalents.
- It is a form of binary encoding where each digit in a decimal number is represented in the form of bits.
- This encoding can be done in either 4-bit or 8-bit (usually 4-bit is preferred).
- It is a fast and efficient system that converts the decimal numbers into binary numbers as compared to the existing binary system.
- These are generally used in digital displays where is the manipulation of data is quite a task.
- Thus BCD plays an important role here because the manipulation is done treating each digit as a separate single sub-circuit.
- The BCD equivalent of a decimal number is written by replacing each decimal digit in the integer and fractional parts with its four bit binary equivalent.
- The BCD code is more precisely known as 8421 BCD code, with 8, 4, 2 and 1 representing the weights of different bits in the four-bit groups, Starting from MSB and proceeding towards LSB.
- This feature makes it a weighted code, which means that each bit in the four bit group representing a given decimal digit has an assigned weight.
- Many decimal values have an infinite place-value representation in binary but have a finite place-value in binary-coded decimal.
- For example, 0.2 in binary is .001100… and in BCD is 0.0010.
- It avoids fractional errors and is also used in huge financial calculations.

Consider the following truth table and focus on how these are represented.

Truth Table for Binary Coded Decimal

DECIMAL NUMBER	BCD
0	0000
1	0001
2	0010

3	0011
4	0100
5	0101
6	0110

- In the BCD numbering system, the given decimal number is segregated into chunks of four bits for each decimal digit within the number.
- Each decimal digit is converted into its direct binary form (usually represented in 4-bits).

For example:

1. Convert (123)10 in BCD

From the truth table above,
1 -> 0001
2 -> 0010
3 -> 0011
Thus, BCD becomes -> 0001 0010 0011

2. Convert (324)10 in BCD

(324)10 -> 0011 0010 0100 (BCD)

Again from the truth table above,
3 -> 0011
2 -> 0010
4 -> 0100
Thus, BCD becomes -> 0011 0010 0100

This is how decimal numbers are converted to their equivalent BCDs.

- It is noticeable that the BCD is nothing more than a binary representation of each digit of a decimal number.
- It cannot be ignored that the BCD representation of the given decimal number uses extra bits, which makes it heavy-weighted.

GRAY CODE

- The reflected binary code or Gray code is an ordering of the binary numeral system such that two successive values differ in only one bit (binary digit).
- Gray codes are very useful in the normal sequence of binary numbers generated by the hardware that may cause an error or ambiguity during the transition from one number to the next.
- So, the Gray code can eliminate this problem easily since only one bit changes its value during any transition between two numbers.
- Gray code is not weighted that means it does not depend on positional value of digit.

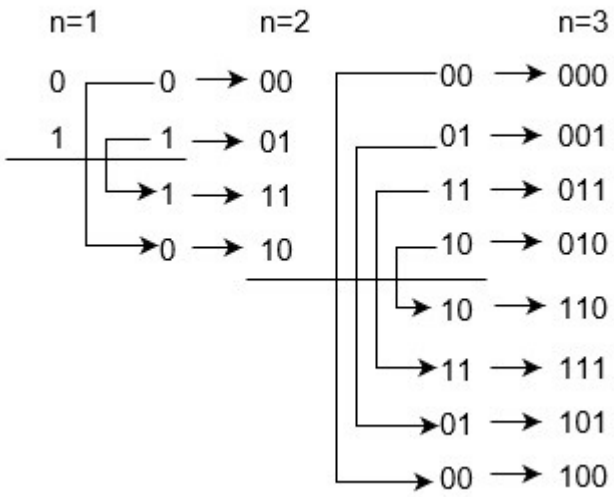
- This cyclic variable code that means every transition from one value to the next value involves only one bit change.
- Gray code also known as reflected binary code, because the first (n/2) values compare with those of the last (n/2) values, but in reverse order.

Constructing an n-bit Gray code

- N-bit Gray code can be generated recursively using reflect and prefix method which is explained as following below.

✓ Generate code for n=1: 0 and 1 code.
✓ Take previous code in sequence: 0 and 1.
✓ Add reversed codes in the following list: 0, 1, 1 and 0.
✓ Now add prefix 0 for original previous code and prefix 1 for new generated code: 00, 01, 11, and 10.

- Therefore, Gray code 0 and 1 are for Binary number 0 and 1 respectively.
- Gray codes: 00. 01, 11, and 10 are for Binary numbers: 00, 01, 10, and 11 respectively.
- Similarly you can construct Gray code for 3 bit binary numbers:



- Therefore, Gray codes are as following below,

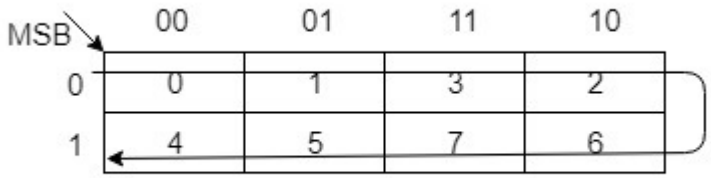
For n = 1 bit		**For n = 2 bit**		**For n = 3 bit**	
Binary	Gray	Binary	Gray	Binary	Gray
0	1	00	00	000	000
0	1	01	01	001	001
		10	11	010	011
		11	10	011	010
				100	110
				101	111
				110	101
				111	100

		001	001

- Iterative method of generating $G_{(n+1)}$ from G_n are given below.
- This is simpler method to contract Gray code of n-bit Binary numbers.
- Each bit is inverted if the next higher bit of the input value is set to one.
- The nth Gray code is obtained by computing n⊕ (floor (n/2)).

❖ G_n is unique numbers for the permutation from 0 to (2^n-1).
❖ G_n is embedded as the first half of $G_{(n+1)}$ and second half as the reverse order of $G_{(n+1)}$.
❖ Prefix 0 in each digit of first half and 1 in each digit of second half.

- The hamming distance of two neighbors Gray codes is always 1 and also first Gray code and last Gray code also has Hamming distance is always 1, so it is also called Cyclic codes.
- You can construct Gray codes using other methods but they may not be performed in parallel like given above method.
- For example, 3 bit Gray codes can be contracted using K-map which is given as following below:



Decimal	Binary	Gray Code
0	000	000
1	001	001
2	010	011
3	011	010
4	100	110
5	101	111
6	110	101

Types of Gray Codes

- There are also other types of Gray codes, like Beckett-Gray code, Single track Gray codes etc.

❖ N-ary Gray code, where non-Boolean values are included like sequences of 1, 2, 3.
❖ Two dimensional (n,k) Gray codes are used for error correction.
❖ Balanced Gray codes have equal transition counts.

Uses of Gray codes

- Gray codes are used in rotary and optical encoders, Karnaugh maps, and error detection.

ERROR DETECTION AND CORRECTION CODE

- We know that the bits 0 and 1 corresponding to two different range of analog voltages.
- So, during transmission of binary data from one system to the other, the noise may also be added.
- Due to this, there may be errors in the received data at other system.
- That means a bit 0 may change to 1 or a bit 1 may change to 0.
- We can't avoid the interference of noise.
- But, we can get back the original data first by detecting whether any errors present and then correcting those errors.
- For this purpose, we can use the following codes.

  - **Error detection codes**
  - **Error correction codes**

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

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

- Therefore, to detect and correct the errors, additional bits are appended to the data bits at the time of transmission.

Parity Code

It is easy to include append one parity bit either to the left of MSB or to the right of LSB of original bit stream. There are two types of parity codes, namely even parity code and odd parity code based on the type of parity being chosen.

Even Parity Code

The value of even parity bit should be zero, if even number of ones present in the binary code. Otherwise, it should be one. So that, even number of ones present in even parity code. Even parity code contains the data bits and even parity bit.

- The following table shows the even parity codes corresponding to each 3-bit binary code.
- Here, the even parity bit is included to the right of LSB of binary code.

Binary Code	Even Parity bit	Even Parity Code

000	0	0000
001	1	0011
010	1	0101
011	0	0110
100	1	1001
101	0	1010
110	0	1100

- Here, the number of bits present in the even parity codes is 4.
- So, the possible even number of ones in these even parity codes are 0, 2 & 4.

✓ If the other system receives one of these even parity codes, then there is no error in the received data. The bits other than even parity bit are same as that of binary code.

✓ If the other system receives other than even parity codes, then there will be an errors in the received data. In this case, we can't predict the original binary code because we don't know the bit positions of error.

- Therefore, even parity bit is useful only for detection of error in the received parity code. But, it is not sufficient to correct the error.

Odd Parity Code

- The value of odd parity bit should be zero, if odd number of ones present in the binary code.
- Otherwise, it should be one.
- So that, odd number of ones present in odd parity code. Odd parity code contains the data bits and odd parity bit.

- The following table shows the odd parity codes corresponding to each 3-bit binary code.
- Here, the odd parity bit is included to the right of LSB of binary code.

Binary Code	**Odd Parity bit**	**Odd Parity Code**
000	1	0001
001	0	0010
010	0	0100
011	1	0111
100	0	1000
101	1	1011
110	1	1101

- Here, the number of bits present in the odd parity codes is 4.
- So, the possible odd number of ones in these odd parity codes is 1 & 3.

✓ If the other system receives one of these odd parity codes, then there is no error in the received data. The bits other than odd parity bit are same as that of binary code.

✓ If the other system receives other than odd parity codes, then there is an errors in the received data. In this case, we can't predict the original binary code because we don't know the bit positions of error.

- Therefore, odd parity bit is useful only for detection of error in the received parity code. But, it is not sufficient to correct the error.

HAMMING CODE

- Hamming code is useful for both detection and correction of error present in the received data.
- This code uses multiple parity bits and we have to place these parity bits in the positions of powers of 2.
- The minimum value of 'k' for which the following relation is correct valid is nothing but the required number of parity bits.

$$2^k \geq n+k+1$$

Where,

'n' is the number of bits in the binary code information
'k' is the number of parity bits

Therefore, the number of bits in the Hamming code is equal to n + k.

- Let the Hamming code is $b_n+kb_{n+k-1}.....b_3b_2b_1$ & parity bits $p_k,p_{k-1},....p_1$
- We can place the 'k' parity bits in powers of 2 positions only. In remaining bit positions, we can place the 'n' bits of binary code.
- Based on requirement, we can use either even parity or odd parity while forming a Hamming code.
- But, the same parity technique should be used in order to find whether any error present in the received data.

- Follow this procedure for finding parity bits.

  - Find the value of p_1, based on the number of ones present in bit positions b_3, b_5, b_7 and so on. All these bit positions suffixes in their equivalent binary have '1' in the place value of 2^0.

  - Find the value of p_2, based on the number of ones present in bit positions b_3, b_6, b_7 and so on. All these bit positions suffixes in their equivalent binary have '1' in the place value of 2^1.

  - Find the value of p3, based on the number of ones present in bit positions b_5, b_6, b_7 and so on. All these bit positions suffixes in their equivalent binary have '1' in the place value of 2^2.

  - Similarly, find other values of parity bits.

- Follow this procedure for finding check bits.

- Find the value of c_1, based on the number of ones present in bit positions b_1, b_3, b_5, b_7 and so on. All these bit positions suffixes in their equivalent binary have '1' in the place value of 2^0.

- Find the value of c_2, based on the number of ones present in bit positions b_2, b_3, b_6, b_7 and so on. All these bit positions suffixes in their equivalent binary have '1' in the place value of 2^1.

- Find the value of c_3, based on the number of ones present in bit positions b_4, b_5, b_6, b_7 and so on. All these bit positions suffixes in their equivalent binary have '1' in the place value of 2^2.

- Similarly, find other values of check bits.

- The decimal equivalent of the check bits in the received data gives the value of bit position, where the error is present.
- Just complement the value present in that bit position.
- Therefore, we will get the original binary code after removing parity bits.

Example 1

Let us find the Hamming code for binary code, $d_4d_3d_2d_1 = 1000$. Consider even parity bits.

The number of bits in the given binary code is n=4.

We can find the required number of parity bits by using the following mathematical relation.

$$\mathbf{2^k \geq n+k+1}$$

Substitute, n=4 in the above mathematical relation.

$$\mathbf{\Rightarrow 2^k \geq 4+k+1}$$
$$\mathbf{\Rightarrow 2^k \geq 5+k}$$

- The minimum value of k that satisfied the above relation is 3.
- Hence, we require 3 parity bits p_1, p_2, and p_3.
- Therefore, the number of bits in Hamming code will be 7, since there are 4 bits in binary code and 3 parity bits.
- We have to place the parity bits and bits of binary code in the Hamming code as shown below.

- The **7-bit Hamming code** is $b_7b_6b_5b_4b_3b_2b_1=d_4d_3d_2p_3d_1p_2bp_1$

- By substituting the bits of binary code, the Hamming code will be $b_7b_6b_5b_4b_3b_2b_1=100p_3Op_2p_1$. Now, let us find the parity bits.

$$\mathbf{p_1=b_7\oplus b_5\oplus b_3=1\oplus 0\oplus 0=1}$$
$$\mathbf{p_2=b_7\oplus b_6\oplus b_3=1\oplus 0\oplus 0=1}$$
$$\mathbf{p_3=b_7\oplus b_6\oplus b_5=1\oplus 0\oplus 0=1}$$

- By substituting these parity bits, the Hamming code will be $b_7b_6b_5b_4b_3b_2b_1=1001011$.

Example 2

- In the above example, we got the Hamming code as $b_7b_6b_5b_4b_3b_2b_1=1001011$.
- Now, let us find the error position when the code received is $b_7b_6b_5b_4b_3b_2b_1=1001111$.

Now, let us find the check bits.

$$\mathbf{c_1=b_7\oplus b_5\oplus b_3\oplus b_1=1\oplus 0\oplus 1\oplus 1=1}$$
$$\mathbf{c_2=b_7\oplus b_6\oplus b_3\oplus b_2=1\oplus 0\oplus 1\oplus 1=1}$$
$$\mathbf{c_3=b_7\oplus b_6\oplus b_5\oplus b_4=1\oplus 0\oplus 0\oplus 1=0}$$

- The decimal value of check bits gives the position of error in received Hamming code.

$$\mathbf{c_3c_2c_1=(011)_2=(3)_{10}}$$

- Therefore, the error present in third bit (b_3) of hamming code. Just complement the value present in that bit and remove parity bits in order to get the original binary code.

BOOLEAN ALGEBRA

- Boolean algebra is an algebra, which deals with binary numbers & binary variables.
- Hence, it is also called as Binary Algebra or logical Algebra.
- A mathematician, named George Boole had developed this algebra in 1854.
- The variables used in this algebra are also called as Boolean variables.
- The range of voltages corresponding to Logic 'High' is represented with '1' and the range of voltages corresponding to logic 'Low' is represented with '0'.

Postulates and Basic Laws of Boolean Algebra

- In this section, let us discuss about the Boolean postulates and basic laws that are used in Boolean algebra.

- These are useful in minimizing Boolean functions.

Boolean Postulates

- Consider the binary numbers 0 and 1, Boolean variable x and its complement x′.
- Either the Boolean variable or complement of it is known as literal.
- The four possible logical OR operations among these literals and binary numbers are shown below.

x + 0 = x

x + 1 = 1

x + x = x

x + x' = 1

- Similarly, the four possible logical AND operations among those literals and binary numbers are shown below.

x.1 = x

x.0 = 0

x.x = x

x.x' = 0

- These are the simple Boolean postulates.
- We can verify these postulates easily, by substituting the Boolean variable with '0' or '1'.

Note− The complement of complement of any Boolean variable is equal to the variable itself. i.e., x′′=x.

Basic Laws of Boolean Algebra

Following are the three basic laws of Boolean Algebra.

- ✓ **Commutative law**
- ✓ **Associative law**
- ✓ **Distributive law**

Commutative Law

- If any logical operation of two Boolean variables gives the same result irrespective of the order of those two variables, then that logical operation is said to be Commutative.
- The logical OR & logical AND operations of two Boolean variables x & y are shown below

$$x + y = y + x$$

$$x.y = y.x$$

- The symbol '+' indicates logical OR operation.
- Similarly, the symbol '.' indicates logical AND operation and it is optional to represent.
- Commutative law obeys for logical OR & logical AND operations.

Associative Law

- If a logical operation of any two Boolean variables is performed first and then the same operation is performed with the remaining variable gives the same result, then that logical operation is said to be Associative.
- The logical OR & logical AND operations of three Boolean variables x, y & z are shown below.

$$x + y+z = x+y + z$$

$$x.y.z = x.y.z$$

- Associative law obeys for logical OR & logical AND operations.

Distributive Law

- If any logical operation can be distributed to all the terms present in the Boolean function, then that logical operation is said to be Distributive.
- The distribution of logical OR & logical AND operations of three Boolean variables x, y & z are shown below.

$$x.y+z = x.y + x.z$$

$$x + y.z = x+y.x+z$$

- Distributive law obeys for logical OR and logical AND operations.

- These are the Basic laws of Boolean algebra. We can verify these laws easily, by substituting the Boolean variables with '0' or '1'.

Theorems of Boolean Algebra

- The following two theorems are used in Boolean algebra.

✓ **Duality theorem**
✓ **DeMorgan's theorem**

Duality Theorem

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

- Let us make the Boolean equations relations that we discussed in the section of Boolean postulates and basic laws into two groups.

- The following table shows these two groups.

Group1	Group2
x + 0 = x	x.1 = x
x + 1 = 1	x.0 = 0
x + x = x	x.x = x
x + x' = 1	x.x' = 0
x + y = y + x	x.y = y.x
x + y+z = x+y + z	x.y.z = x.y.z
x.y+z = x.y + x.z	x + y.z = x+y.x+z

- In each row, there are two Boolean equations and they are dual to each other.
- We can verify all these Boolean equations of Group1 and Group2 by using duality theorem.

DeMorgan's Theorem

- This theorem is useful in finding the complement of Boolean function.
- It states that the complement of logical OR of at least two Boolean variables is equal to the logical AND of each complemented variable.

DeMorgan's theorem with 2 Boolean variables x and y can be represented as

$$x+y' = x'.y'$$

The dual of the above Boolean function is

$$x.y' = x' + y'$$

- Therefore, the complement of logical AND of two Boolean variables is equal to the logical OR of each complemented variable.
- Similarly, we can apply DeMorgan's theorem for more than 2 Boolean variables also.

Simplification of Boolean Functions

- Till now, we discussed the postulates, basic laws and theorems of Boolean algebra.
- Now, let us simplify some Boolean functions.

Example 1

Let us simplify the Boolean function, f = p'qr + pq'r + pqr' + pqr

We can simplify this function in two methods.

Method 1

Given Boolean function, f = p'qr + pq'r + pqr' +pqr.

Step 1 − In first and second terms r is common and in third and fourth terms pq is common. So, take the common terms by using Distributive law.

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

Step 2 − The terms present in first parenthesis can be simplified to Ex-OR operation. The terms present in second parenthesis can be simplified to '1' using Boolean postulate

$$\Rightarrow f = p \oplus qr + pq1$$

Step 3 − The first term can't be simplified further. But, the second term can be simplified to pq using Boolean postulate.

$$\Rightarrow f = p \oplus qr + pq$$

Therefore, the simplified Boolean function is $f = p \oplus qr + pq$

Method 2

Given Boolean function, f = p'qr + pq'r + pqr' + pqr.

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

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

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

$$\Rightarrow f = qrp'+p + prq'+q + pqr'+r$$

Step 3 − Use Boolean postulate, x + x' = 1 for simplifying the terms present in each parenthesis.

$$\Rightarrow f = qr1 + pr1 + pq1$$

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

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

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

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

So, we got two different Boolean functions after simplifying the given Boolean function in each method. Functionally, those two Boolean functions are same. So, based on the requirement, we can choose one of those two Boolean functions.

Example 2

Let us find the complement of the Boolean function, f = p'q + pq'.

The complement of Boolean function is f' = p'q+pq''.

Step 1 − Use DeMorgan's theorem, x+y' = x'.y'.

$\Rightarrow f' = p''q'.pq''$

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

$\Rightarrow f' = \{p'' + q'\}.\{p' + q''\}$

Step3 − Use the Boolean postulate, x''=x.

$\Rightarrow f' = \{p + q'\}.\{p' + q\}$

$\Rightarrow f' = pp' + pq + p'q' + qq'$

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

$\Rightarrow f = 0 + pq + p'q' + 0$

$\Rightarrow f = pq + p'q'$

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

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 circuits 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

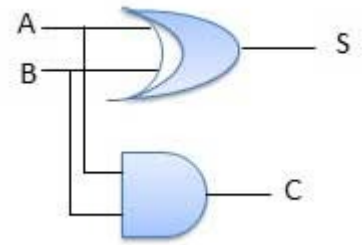
- Half adder is a combinational logic circuit with two inputs and two outputs.
- The half adder circuit is designed to add two single bit binary numbers A and B.
- It is the basic building block for addition of two single bit numbers.
- This circuit has two outputs carry and sum.

Block Diagram

Truth Table

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

Circuit Diagram

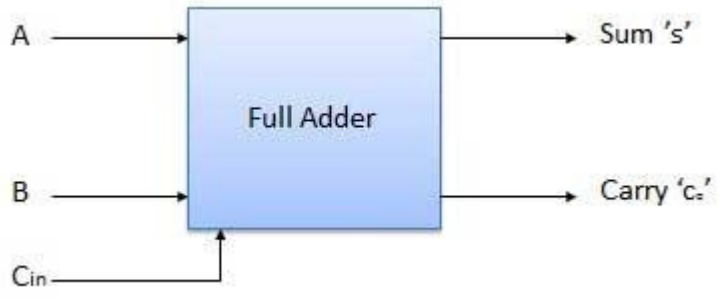


FULL ADDER

- Full adder is developed to overcome the drawback of Half Adder circuit.
- It can add two one-bit numbers A and B, and carry c.

- The full adder is a three input and two output combinational circuit.

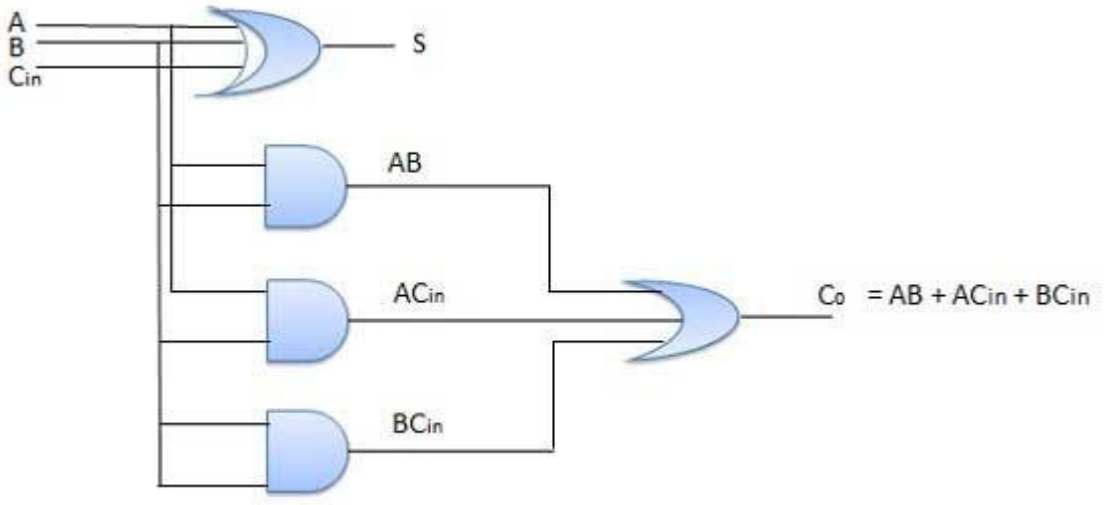
Block diagram



Truth Table

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

Circuit Diagram



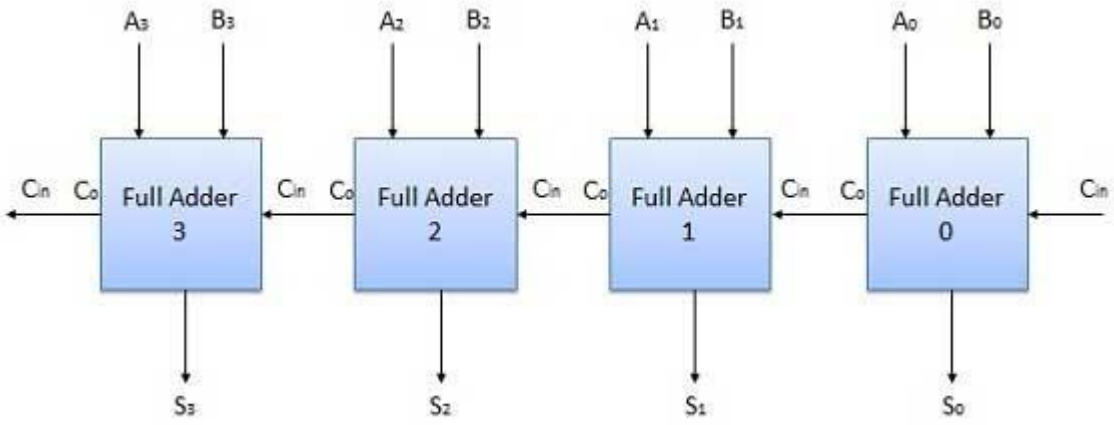
N-Bit Parallel Adder

- 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, A0 and B0 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 are shown in fig.
- The four bit parallel adder is a very common logic circuit.

Block diagram



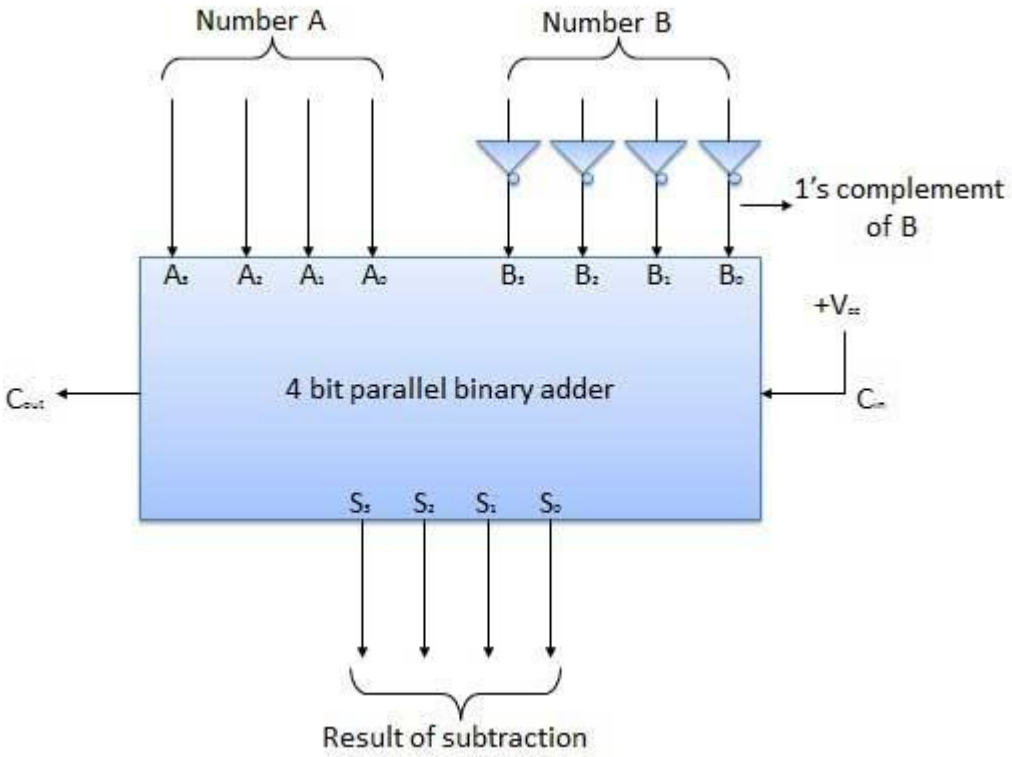
N-Bit Parallel Subtractor

- 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 C_{out} = 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



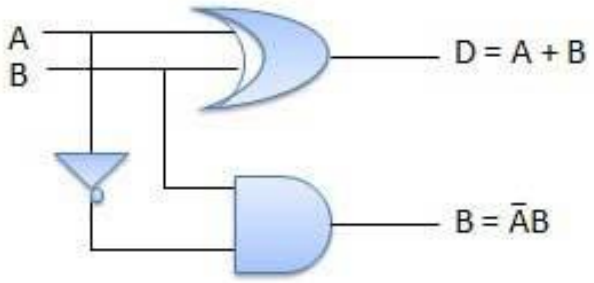
HALF SUBTRACTORS

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

Truth Table

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

Circuit Diagram



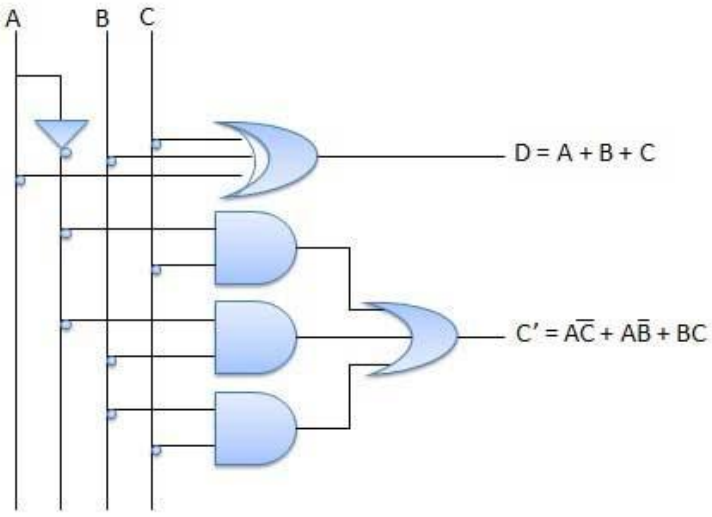
FULL SUBTRACTORS

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

Truth Table

Inputs			Output	
A	B	C	(A-B-C)	C'
0	0	0	0	0
0	0	1	1	1
0	1	0	1	1
0	1	1	0	1
1	0	0	1	0
1	0	1	0	0
1	1	0	0	0
1	1	1	1	1

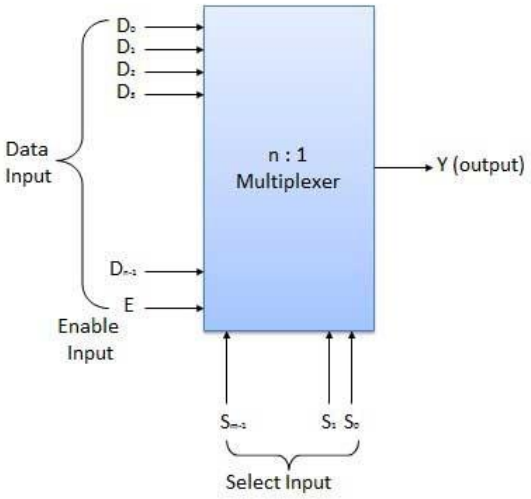
Circuit Diagram



MULTIPLEXERS

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

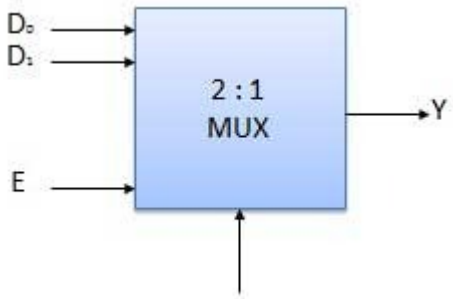
Block diagram



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

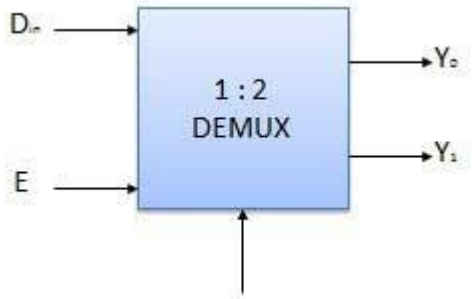
DEMULTIPLEXERS

- 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.

- Demultiplexer comes in multiple variations.

✓ 1 : 2 demultiplexer
✓ 1 : 4 demultiplexer
✓ 1 : 16 demultiplexer
✓ 1 : 32 demultiplexer

Block diagram



Truth Table

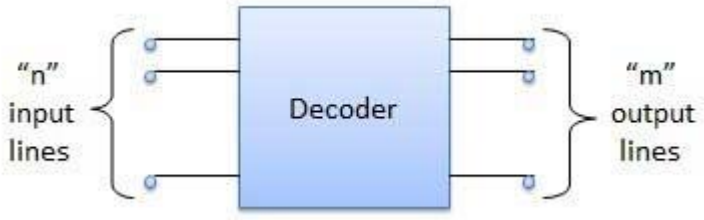
Enable	Select	Output
E	S	Y0 Y1
0	x	0 0
1	0	0 Din
1	1	Din 0

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



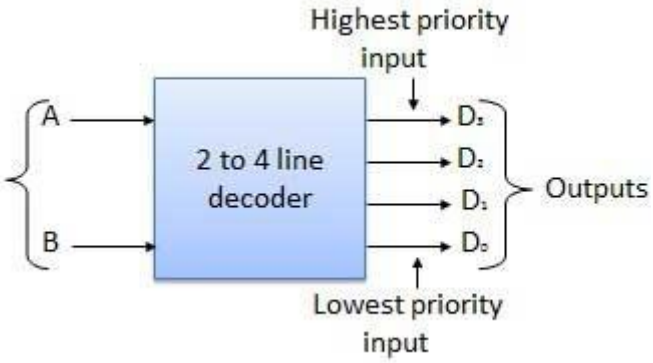
Examples of Decoders are following.

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

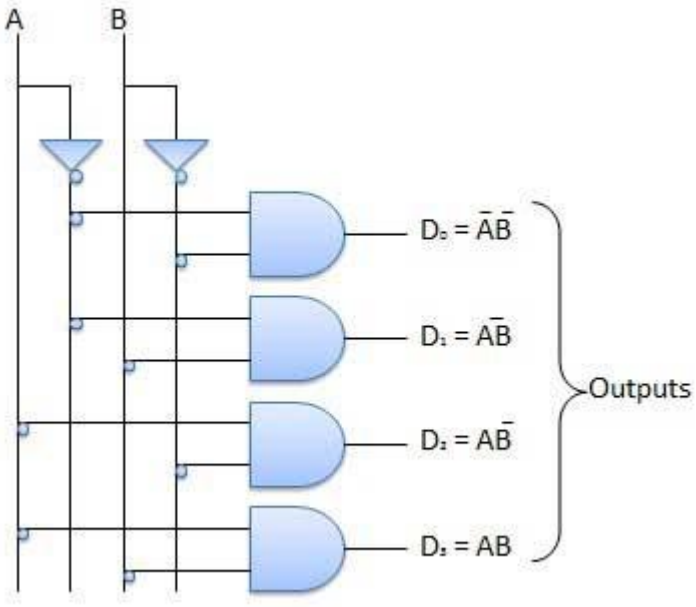
2 to 4 Line Decoder

- 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



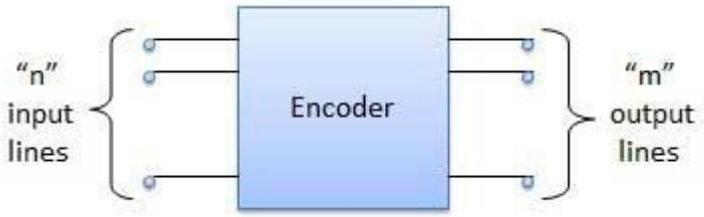
Logic circuit



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



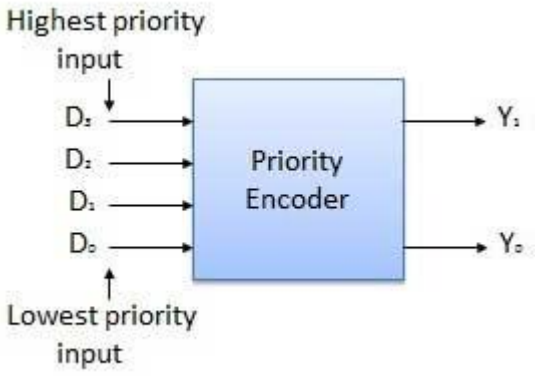
Examples of Encoders are following.

- 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 is 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.

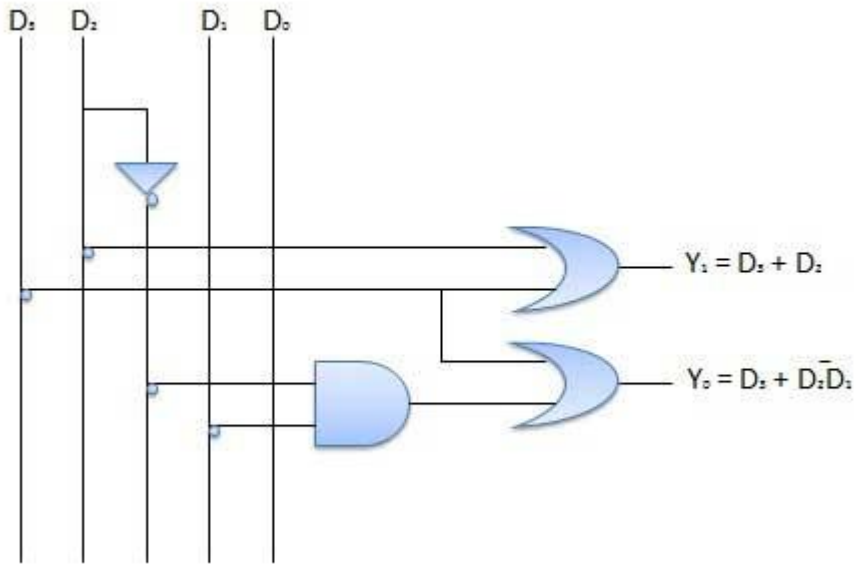
Block diagram



Truth Table

Highest	Inputs		Lowest	Outputs	
D_3	D_2	D_1	D_0	Y_0	Y_1
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



K-MAP (KARNAUGH MAP)

- The Boolean functions using Boolean postulates and theorems are a time consuming process and we have to re-write the simplified expressions after each step.
- To overcome this difficulty, Karnaugh introduced a method for simplification of Boolean functions in an easy way.
- This method is known as Karnaugh map method or K-map method.
- It is a graphical method, which consists of 2n cells for 'n' variables.
- The adjacent cells are differed only in single bit position.

Steps to solve expression using K-map-

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

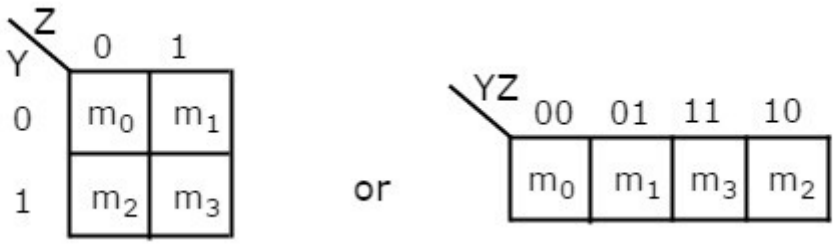
5. Make rectangular groups containing total terms in power of two like 2, 4, 8... (Except 1) and try to cover as many elements as you can in one group.
6. From the groups made in step 5 find the product terms and sum them up for SOP form.

K-Maps for 2 to 5 Variables

- K-Map method is most suitable for minimizing Boolean functions of 2 variables to 5 variables.
- Now, let us discuss about the K-Maps for 2 to 5 variables one by one.

2 Variable K-Map

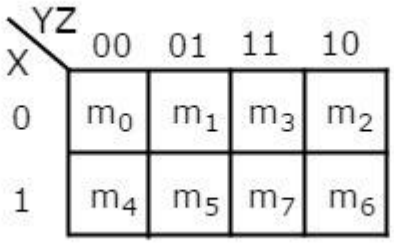
- The number of cells in 2 variables K-map is four, since the number of variables is two.
- The following figure shows 2 variables K-Map.



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

3 Variable K-Map

- The number of cells in 3 variables K-map is eight, since the number of variables is three.
- The following figure shows 3 variables K-Map.

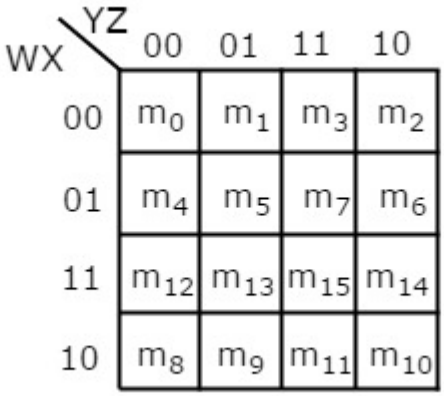


- There is only one possibility of grouping 8 adjacent min terms.
- The possible combinations of grouping 4 adjacent min terms are {(m_0, m_1, m_3, m_2), (m_4, m_5, m_7, m_6), (m_0, m_1, m_4, m_5), (m_1, m_3, m_5, m_7), (m_3, m_2, m_7, m_6) and (m_2, m_0, m_6, m_4)}.
- The possible combinations of grouping 2 adjacent min terms are {(m_0, m_1), (m_1, m_3), (m_3, m_2), (m_2, m_0), (m_4, m_5), (m_5, m_7), (m_7, m_6), (m_6, m_4), (m_0, m_4), (m_1, m_5), (m_3, m7) and (m_2, m_6)}.

- If x=0, then 3 variable K-map becomes 2 variable K-map.

4 Variable K-Map

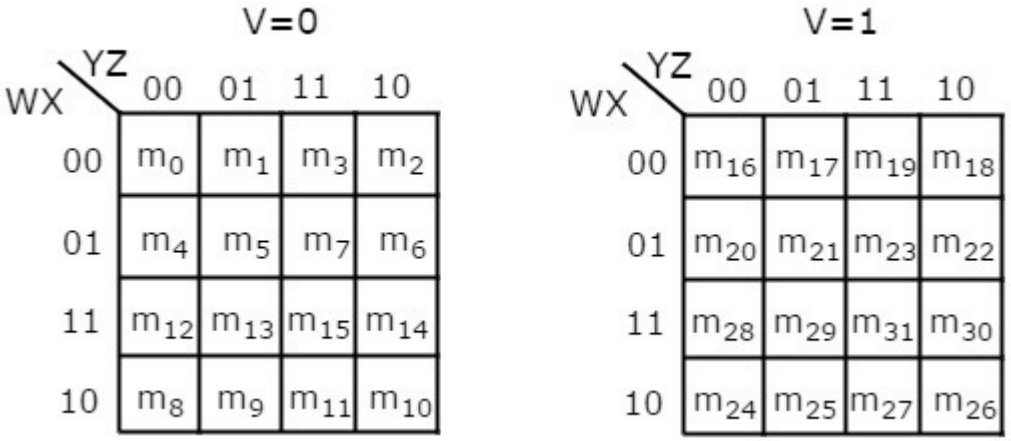
- The number of cells in 4 variables K-map is sixteen, since the number of variables is four. The following figure shows 4 variables K-Map.



- There is only one possibility of grouping 16 adjacent min terms.
- Let R_1, R_2, R_3 and R_4 represents the min terms of first row, second row, third row and fourth row respectively.
- Similarly, C_1, C_2, C_3 and C_4 represents the min terms of first column, second column, third column and fourth column respectively.
- The possible combinations of grouping 8 adjacent min terms are {(R_1, R_2), (R_2, R_3), (R_3, R_4), (R_4, R_1), (C_1, C_2), (C_2, C_3), (C_3, C_4), (C_4, C_1)}.
- If w=0, then 4 variable K-map becomes 3 variable K-map.

5 Variable K-Map

- The number of cells in 5 variables K-map is thirty-two, since the number of variables is 5. The following figure shows 5 variables K-Map.



- There is only one possibility of grouping 32 adjacent min terms.
- There are two possibilities of grouping 16 adjacent min terms. i.e., grouping of min terms from m_0 to m_{15} and m_{16} to m_{31}.
- If v=0, then 5 variable K-map becomes 4 variable K-map.

- In the above all K-maps, we used exclusively the min terms notation.

- Similarly, you can use exclusively the Max terms notation.

Minimization of Boolean Functions using K-Maps

- If we consider the combination of inputs for which the Boolean function is ‘1’, then we will get the Boolean function, which is in standard sum of products form after simplifying the K-map.
- Similarly, if we consider the combination of inputs for which the Boolean function is ‘0’, then we will get the Boolean function, which is in standard product of sums form after simplifying the K-map.
- Follow these rules for simplifying K-maps in order to get standard sum of products form.

✓ Select the respective K-map based on the number of variables present in the Boolean function.

✓ If the Boolean function is given as sum of min terms form, then place the ones at respective min term cells in the K-map. If the Boolean function is given as sum of products form, then place the ones in all possible cells of K-map for which the given product terms are valid.

✓ Check for the possibilities of grouping maximum number of adjacent ones. It should be powers of two. Start from highest power of two and up to least power of two. Highest power is equal to the number of variables considered in K-map and least power is zero.

✓ Each grouping will give either a literal or one product term. It is known as prime implicant. The prime implicant is said to be essential prime implicant, if at least single ‘1’ is not covered with any other groupings but only that grouping covers.

✓ Note down all the prime implicants and essential prime implicants. The simplified Boolean function contains all essential prime implicants and only the required prime implicants.

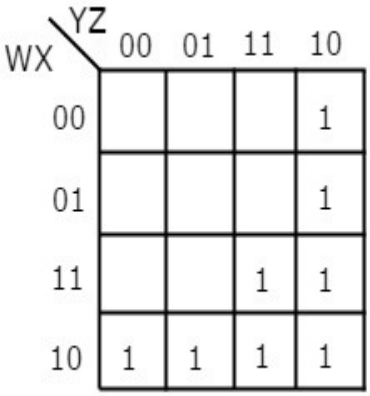
- **Note 1** − If outputs are not defined for some combination of inputs, then those output values will be represented with don’t care symbol ‘x’. That means, we can consider them as either ‘0’ or ‘1’.

- Note 2 − If don’t care terms also present, then place don’t cares ‘x’ in the respective cells of K-map. Consider only the don’t cares ‘x’ that are helpful for grouping maximum number of adjacent ones. In those cases, treat the don’t care value as ‘1’.

Example

- Let us simplify the following Boolean function, f W, X, Y, Z= WX’Y’ + WY + W’YZ’ using K-map.

- The given Boolean function is in sum of products form. It is having 4 variables W, X, Y & Z. So, we require 4 variable K-map. The 4 variable K-map with ones corresponding to the given product terms is shown in the following figure.



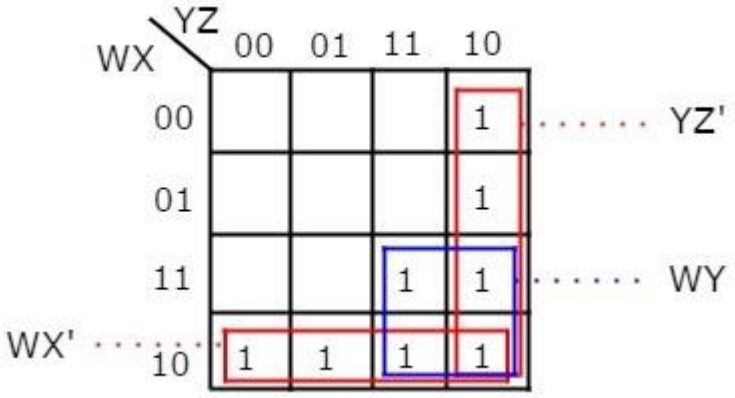
- Here, 1s are placed in the following cells of K-map.

✓ The cells, which are common to the intersection of Row 4 and columns 1 & 2 are corresponding to the product term, WX'Y'.

✓ The cells, which are common to the intersection of Rows 3 & 4 and columns 3 & 4 are corresponding to the product term, WY.

✓ The cells, which are common to the intersection of Rows 1 & 2 and column 4 are corresponding to the product term, W'YZ'.

- There are no possibilities of grouping either 16 adjacent ones or 8 adjacent ones.
- There are three possibilities of grouping 4 adjacent ones.
- After these three groupings, there is no single one left as ungrouped.
- So, we no need to check for grouping of 2 adjacent ones.
- The 4 variable K-map with these three groupings is shown in the following figure.



- Here, we got three prime implicants WX', WY & YZ'. All these prime implicants are essential because of following reasons.

✓ Two ones (m8 & m9) of fourth row grouping are not covered by any other groupings. Only fourth row grouping covers those two ones.

✓ Single one (m15) of square shape grouping is not covered by any other groupings. Only the square shape grouping covers that one.

- ✓ Two ones (m2 & m6) of fourth column grouping are not covered by any other groupings. Only fourth column grouping covers those two ones.

Therefore, the simplified Boolean function is

$$f = WX' + WY + YZ'$$

- Follow these rules for simplifying K-maps in order to get standard product of sums form.

- ✓ Select the respective K-map based on the number of variables present in the Boolean function.

- ✓ If the Boolean function is given as product of Max terms form, then place the zeroes at respective Max term cells in the K-map. If the Boolean function is given as product of sums form, then place the zeroes in all possible cells of K-map for which the given sum terms are valid.

- ✓ Check for the possibilities of grouping maximum number of adjacent zeroes. It should be powers of two. Start from highest power of two and up to least power of two. Highest power is equal to the number of variables considered in K-map and least power is zero.

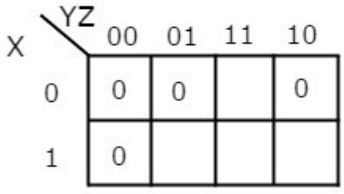
- ✓ Each grouping will give either a literal or one sum term. It is known as prime implicant. The prime implicant is said to be essential prime implicant, if at least single '0' is not covered with any other groupings but only that grouping covers.

- ✓ Note down all the prime implicants and essential prime implicants. The simplified Boolean function contains all essential prime implicants and only the required prime implicants.

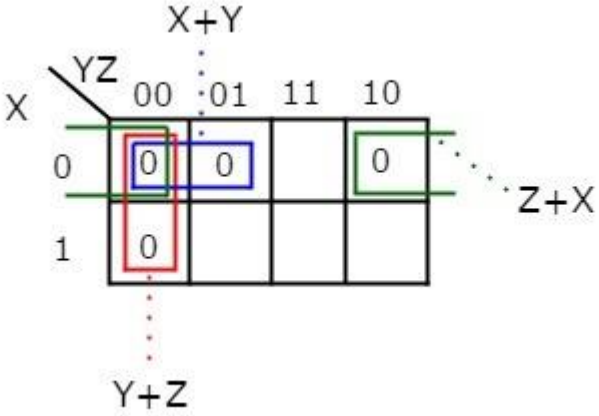
- **Note** − If don't care terms also present, then place don't cares 'x' in the respective cells of K-map. Consider only the don't cares 'x' that are helpful for grouping maximum number of adjacent zeroes. In those cases, treat the don't care value as '0'.

Example

- Let us simplify the following Boolean function, f(X, Y, Z) =∏M (0, 1, 2, 4) using K-map.
- The given Boolean function is in product of Max terms form. It is having 3 variables X, Y & Z. So, we require 3 variable K-map. The given Max terms are M_0, M_1, M_2 & M_4. The 3 variable K-map with zeroes corresponding to the given Max terms is shown in the following figure.



- There are no possibilities of grouping either 8 adjacent zeroes or 4 adjacent zeroes. There are three possibilities of grouping 2 adjacent zeroes.
- After these three groupings, there is no single zero left as ungrouped.
- The 3 variable K-map with these three groupings is shown in the following figure.



- Here, we got three prime implicants X + Y, Y + Z & Z + X.
- All these prime implicants are essential because one zero in each grouping is not covered by any other groupings except with their individual groupings.

Therefore, the simplified Boolean function is

f = X+Y.Y+Z.Z+X

- In this way, we can easily simplify the Boolean functions up to 5 variables using K-map method.
- For more than 5 variables, it is difficult to simplify the functions using K-Maps.
- Because, the number of cells in K-map gets doubled by including a new variable.
- Due to this checking and grouping of adjacent ones minterms or adjacent zeros Maxterms will be complicated.

SUM OF PRODUCT (SOP)

- A canonical sum of products is a Boolean expression that entirely consists of minterms.
- The Boolean function F is defined on two variables X and Y.
- The X and Y are the inputs of the Boolean function F whose output is true when any one of the inputs is set to true.
- The truth table for Boolean expression F is as follows:

Input		Output
X	Y	F
0	0	0
0	1	1
1	0	1
1	1	1

- In our previous section, we learned about how we can form the minterm from the variable's value.
- Now, a column will be added for the minterm in the above table.

- The complement of the variables is taken whose value is 0, and the variables whose value is 1 will remain the same.

Inputs	Output	Minterm
X	Y	F
0	0	0
0	1	1
1	0	1
1	1	1

- Now, we will add all the minterms for which the output is true to find the desired canonical SOP (Sum of Product) expression.

F=X' Y+XY'+XY

Converting Sum of Products (SOP) to shorthand notation

- The process of converting SOP form to shorthand notation is the same as the process of finding shorthand notation for minterms.
- There are the following steps to find the shorthand notation of the given SOP expression.

1. Write the given SOP expression.
2. Find the shorthand notation of all the minterms.
3. Replace the minterms with their shorthand notations in the given expression.

Example: F = X'Y+XY'+XY

1. Firstly, we write the SOP expression:

F = X'Y+XY'+XY

2. Now, we find the shorthand notations of the minterms X'Y, XY', and XY.

$X'Y = (01)_2 = m_1$
$XY' = (10)_2 = m_2$
$XY = (11)_2 = m_3$

3. In the end, we replace all the minterms with their shorthand notations:

F=m1+m2+m3

Converting shorthand notation to SOP expression

- The process of converting shorthand notation to SOP is the reverse process of converting SOP expression to shorthand notation.
- Let's see an example to understand this conversion.

Example:

- Let us assume that we have a Boolean function F, which defined on two variables X and Y.
- The minterms for the function F are expressed as shorthand notation is as follows:

F=∑(1,2,3)

- Now, from this expression, we will find the SOP expression.
- The Boolean function F has two input variables X and y and the output of F=1 for m1, m2, and m3, i.e., 1st, 2nd, and 3rd combinations. So,

F=∑ (1, 2, 3)
F= m1 + m2 + m3
F= 01 + 10 + 11

- Now, we replace zeros with either X' or Y' and ones with either X or Y. Simply, the complement variable is used when the variable value is 1 otherwise the non-complement variable is used.

F = ∑ (1, 2, 3)
F=01+10+11
F= A'B + AB' + AB

PRODUCT OF SUM (POS)

- A canonical product of sum is a Boolean expression that entirely consists of maxterms.
- The Boolean function F is defined on two variables X and Y.
- The X and Y are the inputs of the Boolean function F whose output is true when only one of the inputs is set to true.
- The truth table for Boolean expression F is as follows:

Inputs		**Output**
X	Y	F
0	0	0
0	1	1
1	0	1
1	1	0

- In our minterm and maxterm section, we learned about how we can form the maxterm from the variable's value.
- A column will be added for the maxterm in the above table.

- The complement of the variables is taken whose value is 0, and the variables whose value is 1 will remain the same.

Inputs	Output	Minterm
X	Y	F
0	0	0
0	1	1
1	0	1
1	1	1

- Now, we will multiply all the minterms for which the output is false to find the desired canonical POS (Product of sum) expression.

F=(X'+Y').(X+Y)

Converting Product of Sum (POS) to shorthand notation

- The process of converting POS form to shorthand notation is the same as the process of finding shorthand notation for maxterms.
- There are the following steps used to find the shorthand notation of the given POS expression.

1) Write the given POS expression.
2) Find the shorthand notation of all the maxterms.
3) Replace the minterms with their shorthand notations in the given expression.

Example: F = (X'+Y'). (X+Y)

1. Firstly, we will write the POS expression:

F = (X'+Y'). (X+Y)

2. Now, we will find the shorthand notations of the maxterms X'+Y' and X+Y.

$X'+Y' = (00)_2 = M_0$
$X+Y = (11)_2 = M_3$

3. In the end, we will replace all the minterms with their shorthand notations:

$F=M_0.M_3$

Converting shorthand notation to POS expression

- The process of converting shorthand notation to POS is the reverse process of converting POS expression to shorthand notation.
- Let's see an example to understand this conversion.

Example:

- Let us assume that we have a Boolean function F, defined on two variables X and Y.
- The maxterms for the function F are expressed as shorthand notation is as follows:

$F=\prod (1, 2, 3)$

- Now, from this expression, we find the POS expression.
- The Boolean function F has two input variables X and Y and the output of F=0 for M_1, M_2, and M_3, i.e., 1st, 2nd, and 3rd combinations. So,

$F=\prod (1, 2, 3)$
$F= M_1.M_2.M_3$
$F= 01.10.11$

- Next, we replace zeros with either X or Y and ones with either X' or Y'. Simply, if the value of the variable is 1, then we take the complement of that variable, and if the value of the variable is 0, then we take the variable "as is".

$F = \sum (1, 2, 3)$
$F=01.10.11$
$F= (A+B'). (A'+B). (A'+B')$

Difference between SOP and POS in Digital Logic

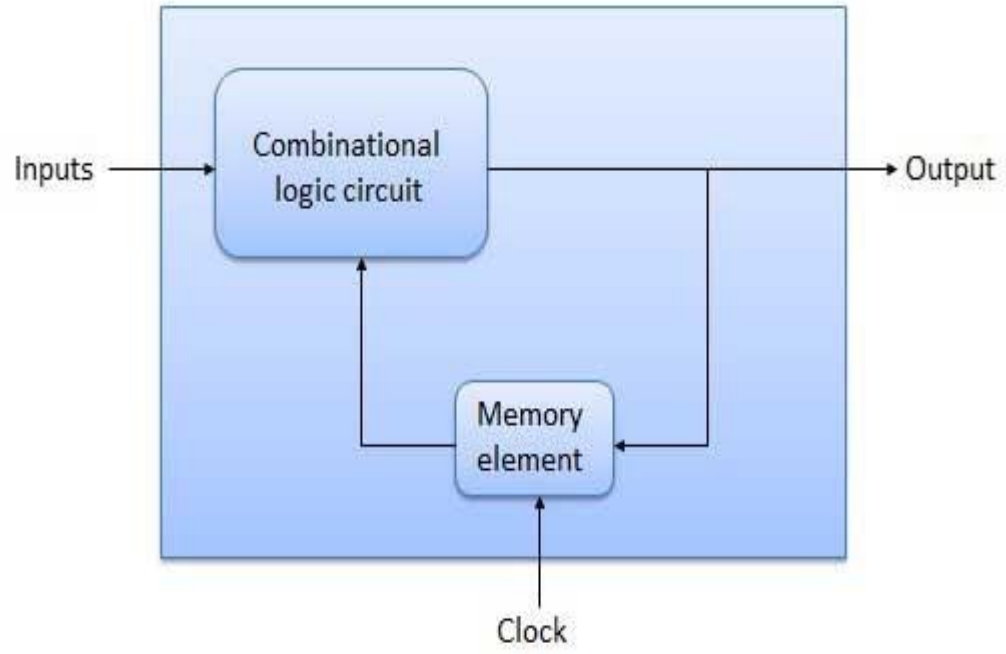
S.No.	SOP	POS
1	SOP stands for Sum of Products.	POS stands for Product of Sums.
2	It is a technique of defining Boolean terms as a sum of product terms.	It is a technique of defining the Boolean terms as the product of sum terms.
3	It prefers minterms.	t prefers maxterms.
4	In the case of SOP, the minterms are defined as 'm'.	In the case of POS, the Maxterms are defined as 'M'
5	It gives HIGH (1) output.	It gives LOW (0) output.
6	In SOP, we can get the final term by adding the product terms.	In POS, we can get the final term by multiplying the sum terms.

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



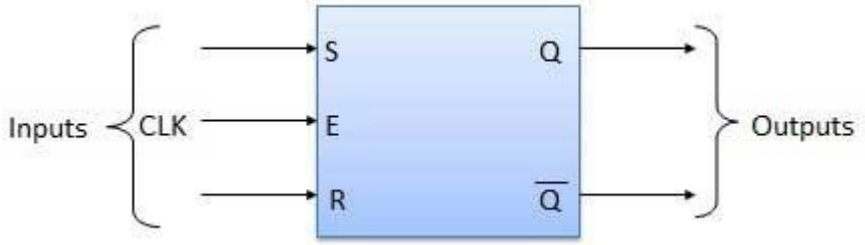
FLIP FLOP

- 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.

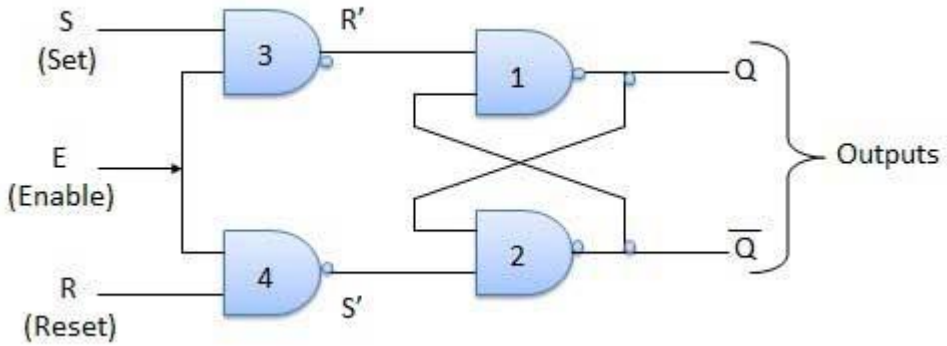
S-R Flip Flop

- 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

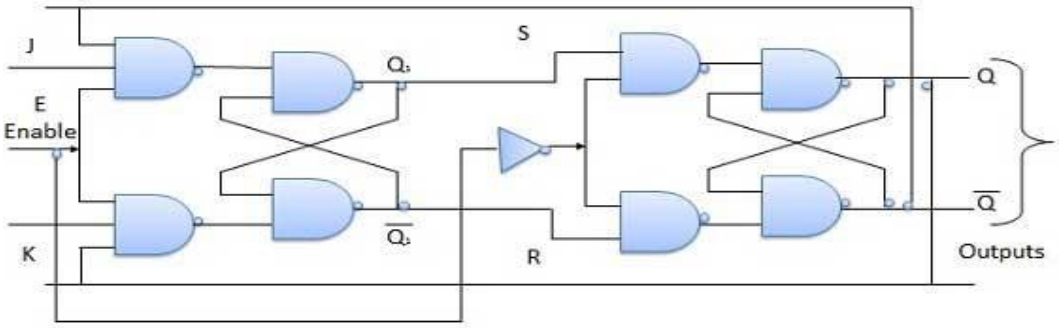
S.N.	Condition	Operation
1	S = R = 0 : No change	If S = R = 0 then output of NAND gates 3 and 4 are forced to become 1. Hence R' and S' both will be equal to 1. Since S' and R' are the input of the basic S-R latch using NAND gates, there will be no change in the state of outputs.

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 Qn+1 = 0 and Qn+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 Qn+1 = 1 and Qn+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 (positive level) the master is active and the slave is inactive.
- Whereas when clock = 0 (low level) 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 (No change)	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 Q1 = 0 and Q1 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

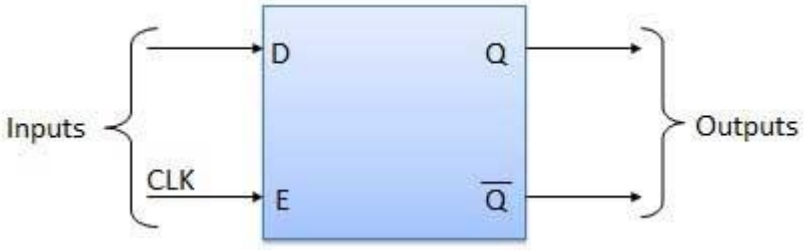
		we get a stable output from the Master slave.
3	J = 1 and K = 0 (Set)	Clock = 1 − Master active, slave inactive. Therefore outputs of the master become Q1 = 1 and Q1 bar = 0. That means S = 1 and R =0. Clock = 0 − Slave active, master inactive. Therefore outputs of the slave become Q = 1 and Q bar = 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

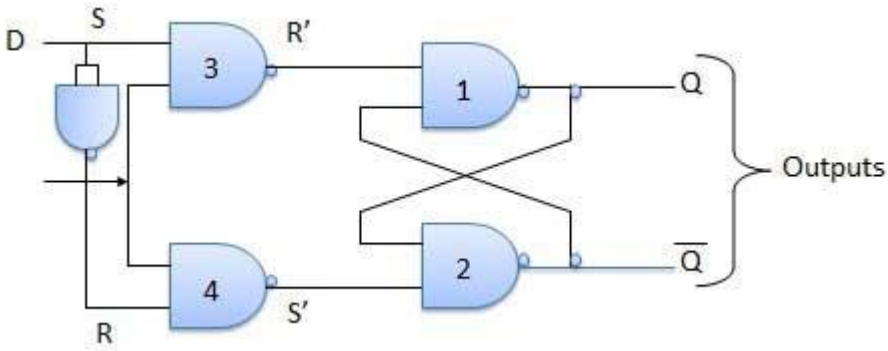
- Delay Flip Flop or D Flip Flop is the simple gated S-R latch with a NAND inverter connected between S and R inputs.
- It has only one input.
- The input data is appearing at the output after some time.
- Due to this data delay between i/p and o/p, it is called delay flip flop.
- S and R will be the 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

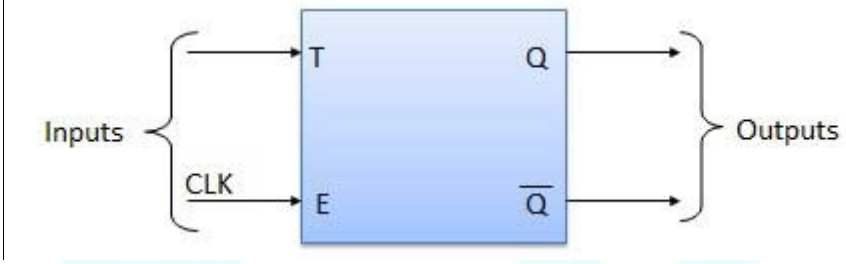
S.N.	Condition	Operation
1	E = 0	Latch is disabled. Hence no change in output.
2	E = 1 and D = 0	If E = 1 and D = 0 then S = 0 and R = 1. Hence irrespective of the present state, the next state is Qn+1

		= 0 and Qn+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 Qn+1 = 1 and Qn+1 bar = 0 irrespective of the present state.

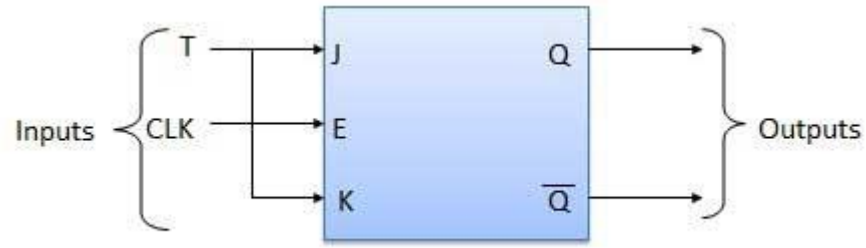
Toggle Flip Flop / T Flip Flop

- 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

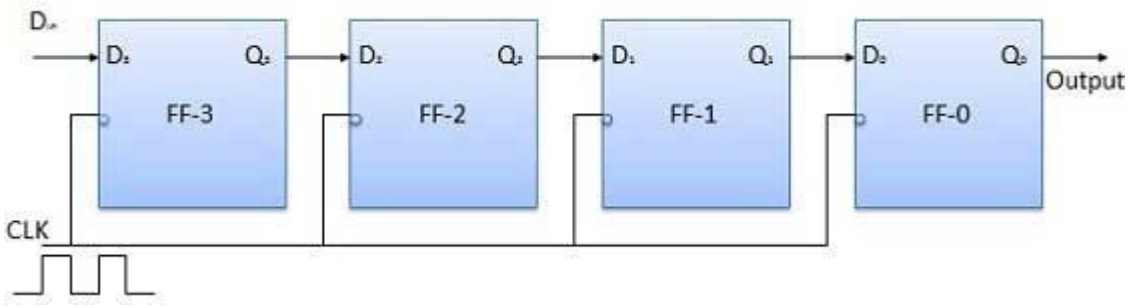
- Flip-flop is a 1 bit memory cell which can be used for storing the digital data.
- To increase the storage capacity in terms of number of bits, we have to use a group of flip-flop. Such a group of flip-flop is known as a Register.
- The n-bit register will consist of n number of flip-flop and it is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

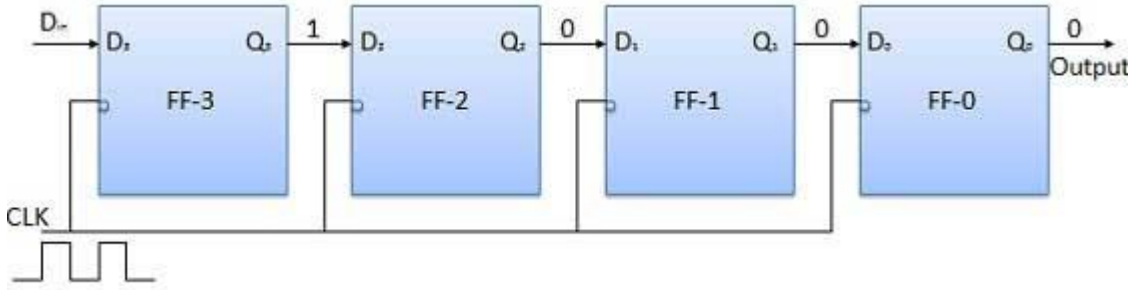
Block Diagram



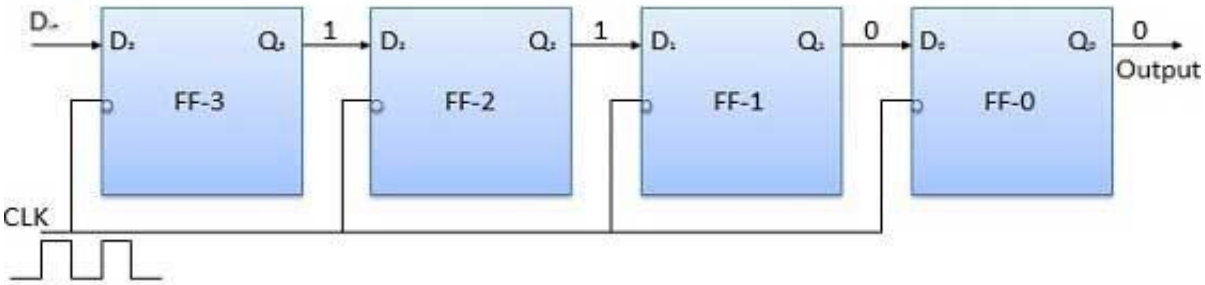
Operation

- 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}.

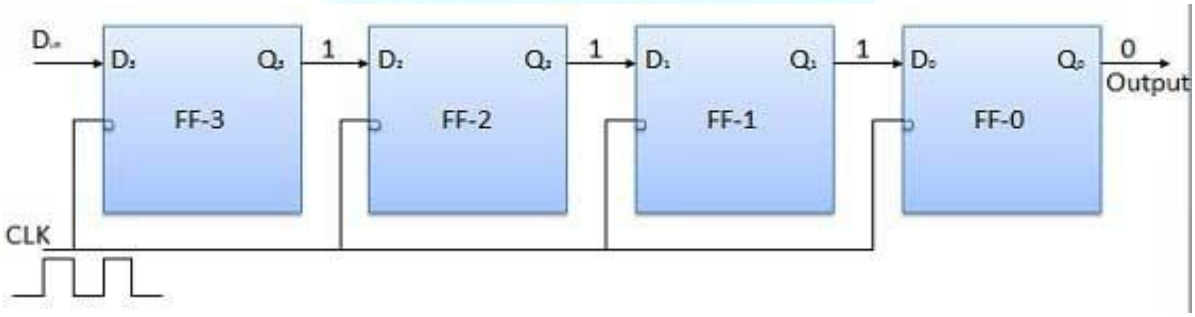
- So $D_{in} = D_3 = 1$. Apply the clock. On the first falling edge of clock, the FF-3 is set, and stored word in the register is $Q_3\ Q_2\ Q_1\ Q_0 = 1000$.



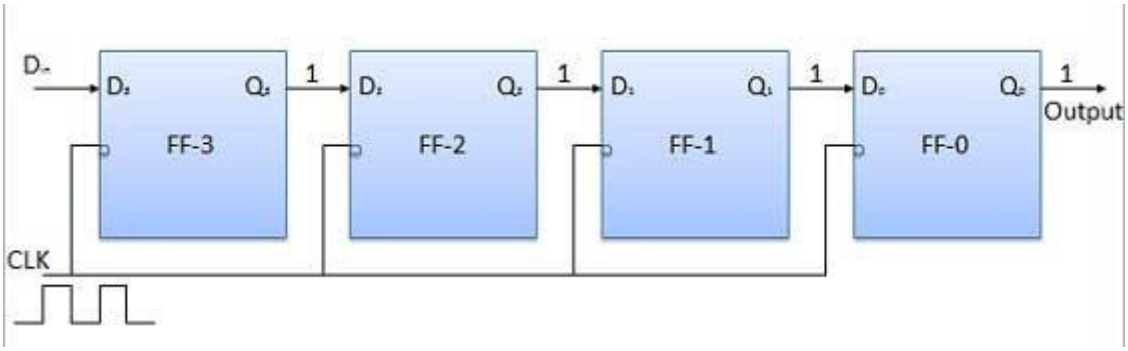
- Apply the next bit to D_{in}. So $D_{in} = 1$. As soon as the next negative edge of the clock hits, FF-2 will set and the stored word change to $Q_3\ Q_2\ Q_1\ Q_0 = 1100$.



- 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 Din = 1 and with the fourth negative clock edge arriving, the stored word in the register is Q3 Q2 Q1 Q0 = 1111.

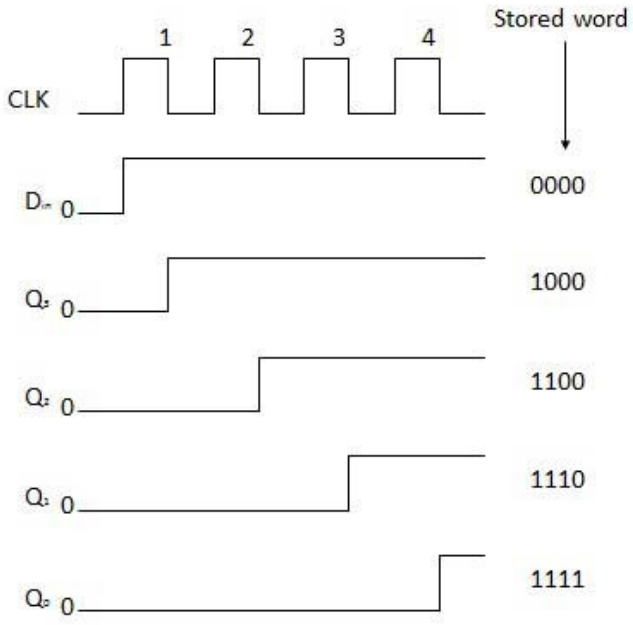


Truth Table

	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

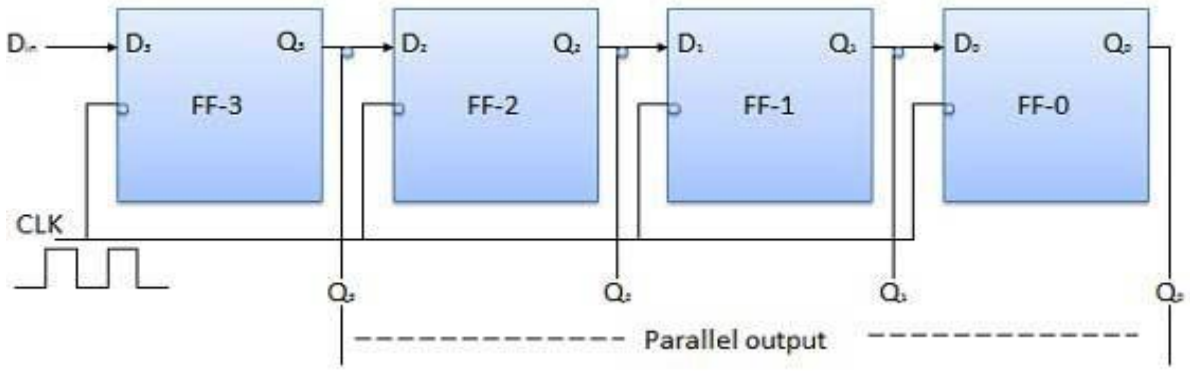
Waveforms



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**.

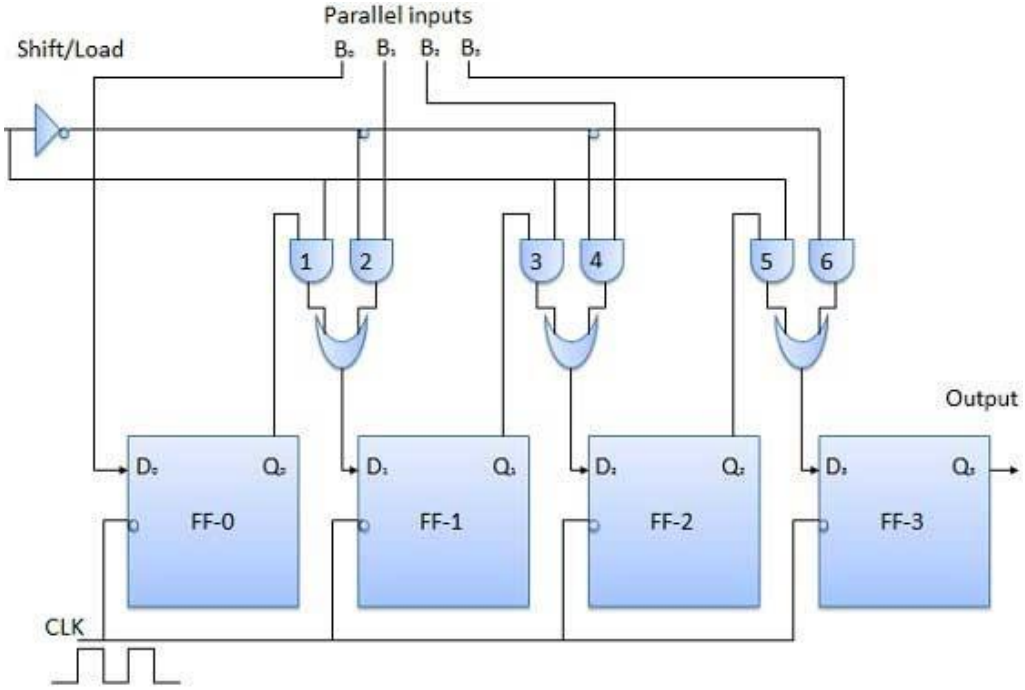
Load mode

- 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.

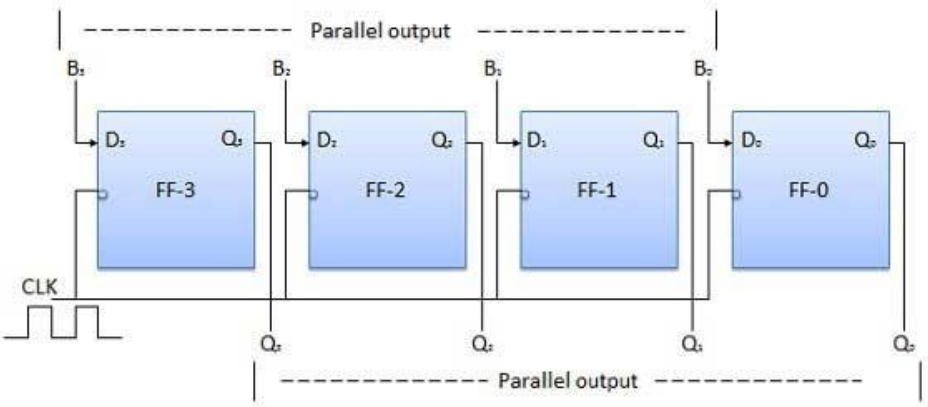
Block Diagram



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.

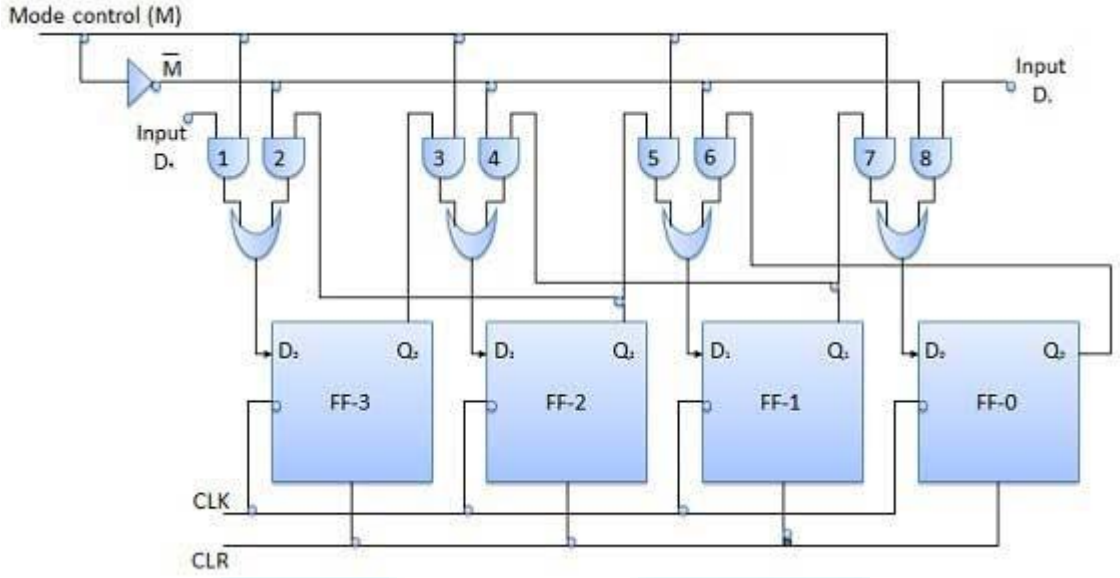
Block Diagram



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

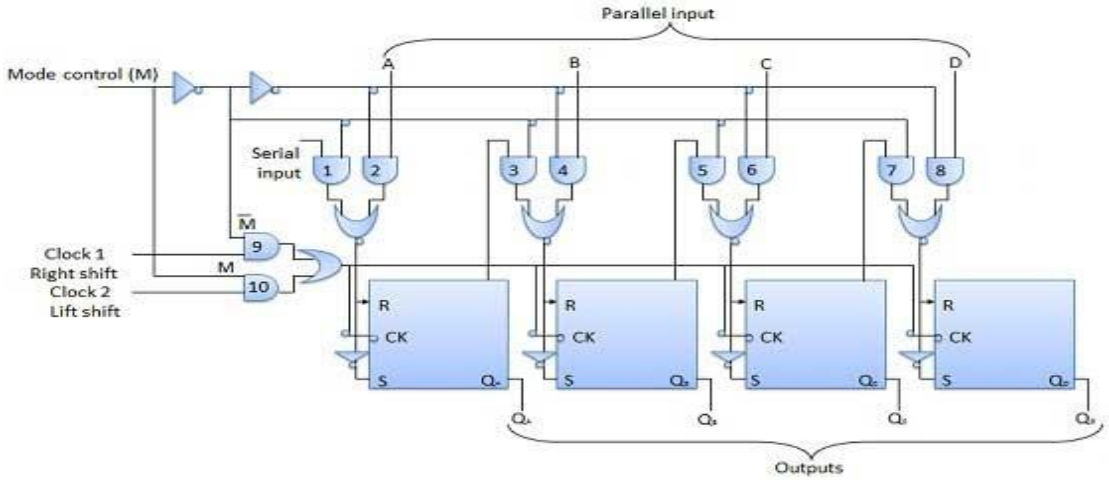
S.N.	Condition	Operation
1	With M = 1 − Shift right operation	If M = 1, then the AND gates 1, 3, 5 and 7 are enabled whereas the remaining AND gates 2, 4, 6 and 8 will be disabled. The data at DR is shifted to right bit by bit from FF-3 to FF-0 on the application of clock pulses. Thus with M = 1 we get 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 DL 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
    - ✓ Left 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

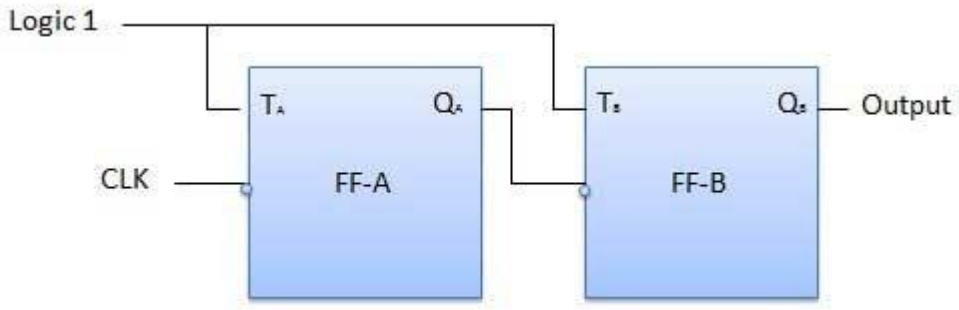
- Counter is a sequential circuit.
- A digital circuit which is used for counting pulses is known counter.
- Counter is the widest application of flip-flops.
- It is a group of flip-flops with a clock signal applied.
- Counters are of two types.

✓ **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 is 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 QB 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

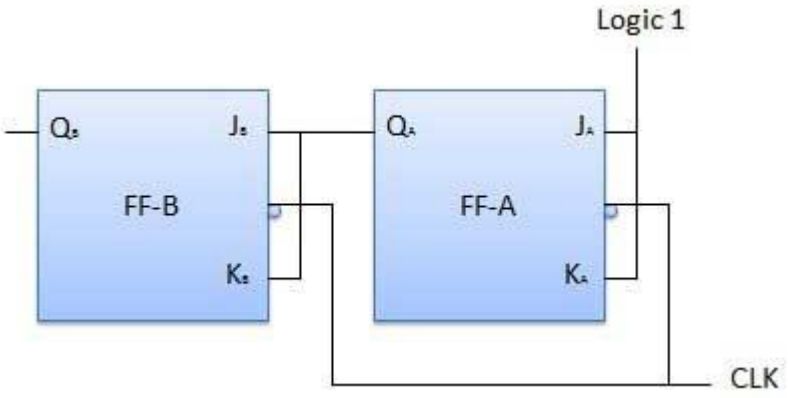
Synchronous counters

- 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 = Q bar) output of the previous FF.

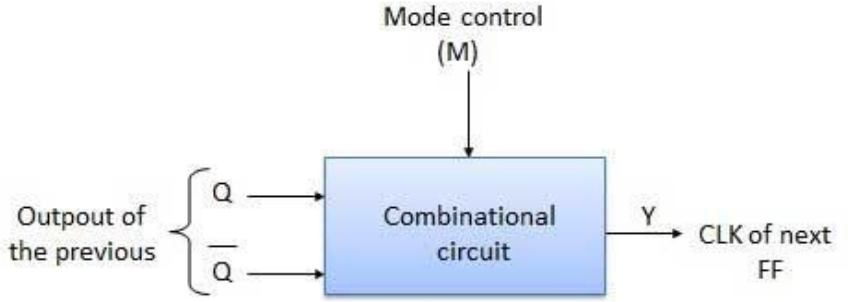
UP counting mode (M=0) − The Q output of the preceding FF is connected to the clock of the next stage if up counting is to be achieved. For this mode, the mode select input M is at logic 0 (M=0).

DOWN counting mode (M=1) − If M = 1, then the Q bar output of the preceding FF is connected to the next FF. This will operate the counter in the counting mode.

Example
3-bit binary up/down ripple counter.

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

Block Diagram



Truth Table

Inputs			Outputs	
M	Q	$\overline{Q}$	Y	
0	0	0	0	Y = Q for up counter
0	0	1	0	
0	1	0	1	
0	1	1	1	
1	0	0	0	$Y = \overline{Q}$ for up counter
1	0	1	1	
1	1	0	0	
1	1	1	1	

Operation

S.N.	Condition	Operation
1	Case 1 − With M = 0 (Up counting mode)	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 Hence Q_A gets connected to the clock input of FF-B and QB 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.4 will be disabled.
2	Case 2: With M = 1 (Down counting mode)	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-N Counter)

- 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 = 2n.

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.

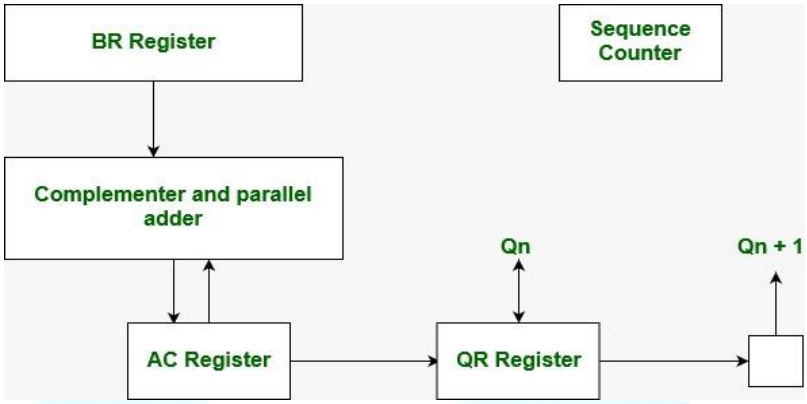
BOOTH'S ALGORITHM

- Booth algorithm gives a procedure for **multiplying binary integers** in signed 2's complement representation **in efficient way**, i.e., less number of additions/subtractions required.
- It operates on the fact that strings of 0's in the multiplier require no addition but just shifting and a string of 1's in the multiplier from bit weight 2^k to weight 2^m can be treated as 2^(k+1) to 2^m.
- As in all multiplication schemes, booth algorithm requires examination **of the multiplier bits** and shifting of the partial product.
- Prior to the shifting, the multiplicand may be added to the partial product, subtracted from the partial product, or left unchanged according to following rules:

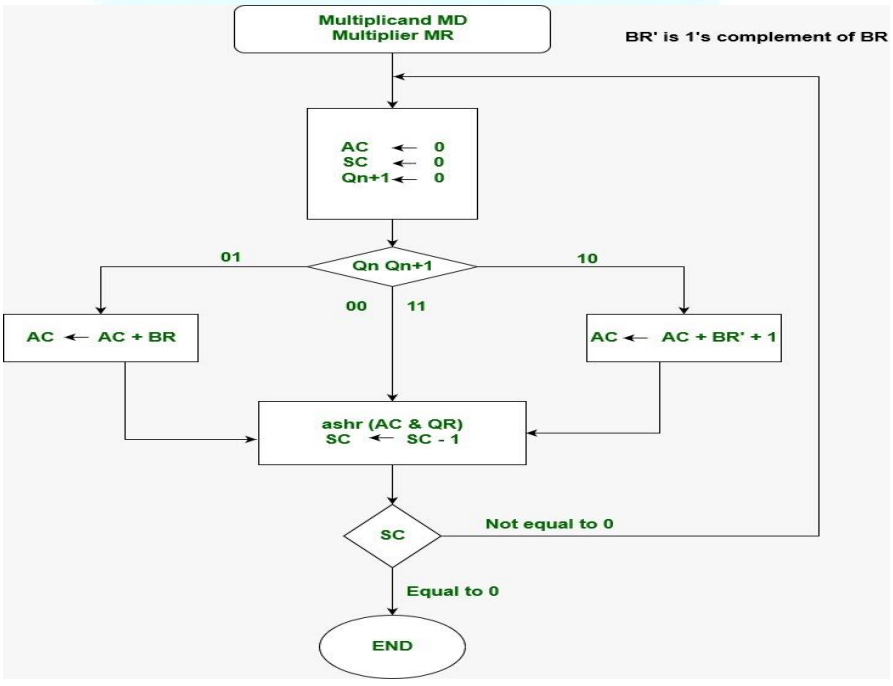
1. The multiplicand is subtracted from the partial product upon encountering the first least significant 1 in a string of 1's in the multiplier
2. The multiplier is added to the partial product upon encountering the first 0 (provided that there was a previous '1') in a string of 0's in the multiplier.
3. The partial product does not change when the multiplier bit is identical to the previous multiplier bit.

- **Hardware Implementation of Booths Algorithm –** The hardware implementation of the booth algorithm requires the register configuration shown in the figure below.

BOOTH'S ALGORITHM FLOWCHART



- We name the register as A, B and Q, AC, BR and QR respectively.
- Qn designates the least significant bit of multiplier in the register QR.
- An extra flip-flop Qn+1is appended to QR to facilitate a double inspection of the multiplier.
- Qn+1 is appended to QR to facilitate a double inspection of the multiplier.
- The flowchart for the booth algorithm is shown below.



- AC and the appended bit Qn+1 are initially cleared to 0 and the sequence SC is set to a number n equal to the number of bits in the multiplier.
- The two bits of the multiplier in Qn and Qn+1are inspected.
- If the two bits are equal to 10, it means that the first 1 in a string has been encountered.
- This requires subtraction of the multiplicand from the partial product in AC. If the 2 bits are equal to 01, it means that the first 0 in a string of 0's has n = been encountered.
- This requires the addition of the multiplicand to the partial product in AC.
- When the two bits are equal, the partial product does not change.
- An overflow cannot occur because the addition and subtraction of the multiplicand follow each other.
- As a consequence, the 2 numbers that are added always have a opposite signs, a condition that excludes an overflow.
- The next step is to shift right the partial product and the multiplier (including Qn+1).
- This is an arithmetic shift right (ashr) operation which AC and QR ti the right and leaves the sign bit in AC unchanged.
- The sequence counter is decremented and the computational loop is repeated n times.

Example – A numerical example of booth's algorithm is shown below for n = 4. It shows the step by step multiplication of -5 and -7.

MD = -5 = 1011, MD = 1011, MD'+1 = 0101
MR = -7 = 1001
The explanation of first step is as follows: Qn+1
AC = 0000, MR = 1001, Qn+1 = 0, SC = 4
Qn Qn+1 = 10
So, we do AC + (MD)'+1, which gives AC = 0101
On right shifting AC and MR, we get
AC = 0010, MR = 1100 and Qn+1 = 1

OPERATION	AC	MR	Qn+1	SC
0000	1001	0	4	
AC + MD' + 1	0101	1001	0	
ASHR	0010	1100	1	3
AC + MR	1101	1100	1	
ASHR	1110	1110	0	2
ASHR	1111	0111	0	1
AC + MD' + 1	0010	0011	1	0

Product is calculated as follows:

Product = AC MR
Product = 0010 0011 = 35

FLOATING POINT REPRESENTATION

- The floating-point representation can implement operations for high range values.
- The numerical evaluations are carried out using floating-point values.
- It can create calculations easy, scientific numbers are described as follows −

  The number 5,600,000 can be described as 0.56 * 107.

  Therefore, 0.56 is the mantissa and 7 is the value of the exponent.

- Binary numbers can also be described in exponential form.
- The description of binary numbers in the exponential form is called floating-point representation.
- The floating-point representation breaks the number into two parts, the left-hand side is a **signed**, fixed-point number known as a **mantissa** and the right-hand side of the number is known as the **exponent**.
- The floating-point values are also authorized with a sign; 0 denoting the positive value and 1 denoting the negative value.

  The general structure of floating-point representation of a binary number −

  $$x=(x_0*2^0+x_1*2^1+x_2*2^2\pm\text{−−}\mp b_{-(n-1)}*2^{-(n-1)})$$

  $$\text{Mantissa}*2^{\text{Exponent}}$$

- In the following syntax, the decimal point is transferred left for negative exponents of two and right for positive exponents of two.
- Both the mantissa and the exponent are signed values enabling negative numbers and negative exponents commonly.

Example − Convert 111101.1000110 into floating-point value.

111101.1000110 = 1.111011000110 * 2^5 converted to floating-point value

→ Denotes negative sign value

- In this example, the integer value is converted to a floating-point value by changing the radix point next to the signed integer and scaling up the number to the exponential form by multiplying the value with the base 2.
- The value remains unaltered and this phase is known as the normalized method.