Digital Electronics Syllabus

Unit	Details
I	**Number System:** Analog System, digital system, numbering system, binary number system, octal number system, hexadecimal number system, conversion from one number system to another, floating point numbers, weighted codes binary coded decimal, non-weighted codes Excess – 3 code, Gray code, Alphanumeric codes – ASCII Code, EBCDIC, ISCII Code, Hollerith Code, Morse Code, Teletypewriter (TTY), Error detection and correction, Universal Product Code, Code conversion. **Binary Arithmetic:** Binary addition, Binary subtraction, Negative number representation, Subtraction using 1's complement and 2's complement, Binary multiplication and division, Arithmetic in octal number system, Arithmetic in hexadecimal number system, BCD and Excess – 3 arithmetic.
II	**Boolean Algebra and Logic Gates:** Introduction, Logic (AND OR NOT), Boolean theorems, Boolean Laws, De Morgan's Theorem, Perfect Induction, Reduction of Logic expression using Boolean Algebra, Deriving Boolean expression from given circuit, exclusive OR and Exclusive NOR gates, Universal Logic gates, Implementation of other gates using universal gates, Input bubbled logic, Assertion level. **Minterm, Maxterm and Karnaugh Maps:** Introduction, minterms and sum of minterm form, maxterm and Product of maxterm form, Reduction technique using Karnaugh maps – 2/3/4/5/6 variable K-maps, Grouping of variables in K-maps, K-maps for product of sum form, minimize Boolean expression using K-map and obtain K-map from Boolean expression, Quine Mc Cluskey Method.
III	**Combinational Logic Circuits:** Introduction, Multi-input, multi-output Combinational circuits, Code converters design and implementations **Arithmetic Circuits:** Introduction, Adder, BCD Adder, Excess – 3 Adder, Binary Subtractors, BCD Subtractor, Multiplier, Comparator.
IV	**Multiplexer, Demultiplexer, ALU, Encoder and Decoder:** Introduction, Multiplexer, Demultiplexer, Decoder, ALU, Encoders. **Sequential Circuits: Flip-Flop:** Introduction, Terminologies used, S-R flip-flop, D flip-fop, JK flip-flop, Race-around condition, Master – slave JK flip-flop, T flip-flop, 12 14 | P a g e conversion from one type of flip-flop to another, Application of flip-flops.

V	**Counters:** Introduction, Asynchronous counter, Terms related to counters, IC7493 (4-bit binary counter), Synchronous counter, Bushing, Type TDesign, Type JK Design, Preset table counter, IC 7490, IC 7492,Synchronous counter ICs, Analysis of counter circuits. **Shift Register:** Introduction, parallel and shift registers, serial shifting, serial–in serial–out, serial–in parallel–out , parallel–in parallel–out, Ring counter, Johnson counter, Applications of shift registers, Pseudo-random binary sequence generator, IC7495, Seven Segment displays, analysis of shift counters.

Unit :1

Chapter 1:Number System

Analog System, digital system, numbering system, binary number system, octal number system, hexadecimal number system, conversion from one number system to another, floating point numbers, weighted codes binary coded decimal, non-weighted codes Excess – 3 code, Gray code, Alphanumeric codes – ASCII Code, EBCDIC, ISCII Code, Hollerith Code, Morse Code, Teletypewriter (TTY), Error detection and correction, Universal Product Code, Code conversion.

Analog System & Digital System:

Digital as well as Analog System, both are used to transmit signals from one place to another like audio/video. Digital system uses binary format as 0 and 1 whereas analog system uses electronic pulses with varing magnitude to send data.

Following are some of the important differences between Digital System and Analog System.

.

Sr. No.	Key	Digital System	Analog System
1	Signal Type	Digital System uses discrete signals as on/off representing binary format. Off is 0, On is 1.	Analog System uses continous signals with varying magnitude.
2	Wave Type	Digital System uses square waves.	Analog system uses sine waves.
3	Technology	Digital system first transform the analog waves to limited set of numbers and then record them as digital square waves.	Analog systems records the physical waveforms as they are originally generated.
4	Transmission	Digital transmission is easy and can be made noise proof with no loss at all.	Analog systems are affected badly by noise during transmission.

Sr. No.	Key	Digital System	Analog System
5	Flexibility	Digital system hardware can be easily modulated as per the requirements.	Analog system's hardwares are not flexible.
6	Bandwidth	Digital transmission needs more bandwidth to carry same information.	Analog tranmission requires less bandwidth.
7	Memory	Digital data is stored in form of bits.	Analog data is stored in form of waveform signals.
8	Power requirement	Digital system needs low power as compare to its analog counterpart.	Analog systems consume more power than digital systems.
9	Best suited for	Digital system are good for computing and digital electronics.	Analog systems are good for audio/video recordings.
10	Cost	Digital system are costly.	Analog systems are cheap.
11	Example	Digital system are: Computer, CD, DVD.	Analog systems are: Analog electronics, voice radio using AM frequency.

Numbering System:

In a digital system, the system can understand only the optional number system. In these systems, digits symbols are used to represent different values, depending on the index from which it settled in the number system.

In simple terms, for representing the information, we use the number system in the digital system.

The digit value in the number system is calculated using:

1. The digit
2. The index, where the digit is present in the number.

3. Finally, the base numbers, the total number of digits available in the number system.

Types of Number System

In the digital computer, there are various types of number systems used for representing information.

1. Binary Number System
2. Decimal Number System
3. Hexadecimal Number System
4. Octal Number System

Binary Number System

Generally, a binary number system is used in the digital computers. In this number system, it carries only two digits, either 0 or 1. There are two types of electronic pulses present in a binary number system. The first one is the absence of an electronic pulse representing '0'and second one is the presence of electronic pulse representing '1'. Each digit is known as a bit. A four-bit collection (1101) is known as a nibble, and a collection of eight bits (11001010) is known as a byte. The location of a digit in a binary number represents a specific power of the base (2) of the number system.

Characteristics:

1. It holds only two values, i.e., either 0 or 1.
2. It is also known as the base 2 number system.
3. The position of a digit represents the 0 power of the base(2). Example: 2^0
4. The position of the last digit represents the x power of the base(2). Example: 2^x, where x represents the last position, i.e., 1

Examples:$(10100)_2$, $(11011)_2$, $(11001)_2$, $(000101)_2$, $(011010)_2$.

Decimal Number System

The decimal numbers are used in our day to day life. The decimal number system contains ten digits from 0 to 9(base 10). Here, the successive place value or position, left to the decimal point holds units, tens, hundreds, thousands, and so on.

The position in the decimal number system specifies the power of the base (10). The 0 is the minimum value of the digit, and 9 is the maximum value of the digit. For example, the decimal number 2541 consist of the digit 1 in the unit position, 4 in the tens position, 5 in the hundreds position, and 2 in the thousand positions and the value will be written as:

$(2\times1000) + (5\times100) + (4\times10) + (1\times1)$

$(2\times10^3) + (5\times10^2) + (4\times10^1) + (1\times10^0)$

2000 + 500 + 40 + 1

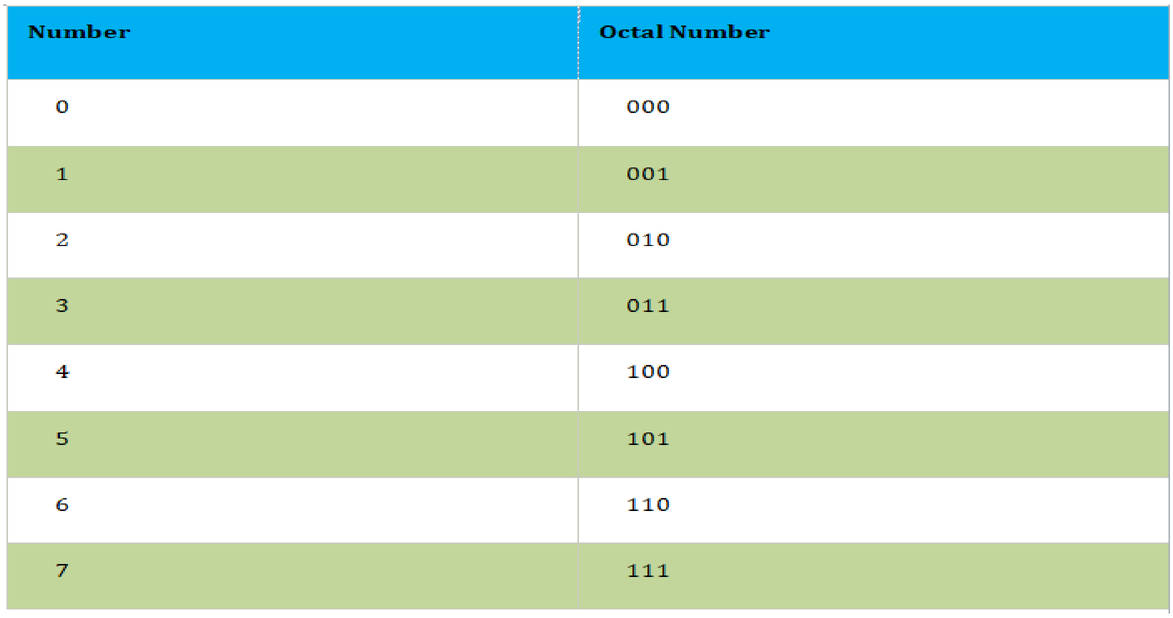
2541

Octal Number System:

The octal number system has base 8(means it has only eight digits from 0 to 7). There are only eight possible digit values to represent a number. With the help of only three bits, an octal number is represented. Each set of bits has a distinct value between 0 and 7.Below, we have described certain characteristics of the octal number system:

Characteristics:

1. An octal number system carries eight digits starting from 0, 1, 2, 3, 4, 5, 6, and 7.
2. It is also known as the base 8 number system.
3. The position of a digit represents the 0 power of the base(8). Example: 8^0
4. The position of the last digit represents the x power of the base(8). Example: 8^x, where x represents the last position, i.e., 1



Number	Octal Number
0	000
1	001
2	010
3	011
4	100
5	101
6	110
7	111

Examples:

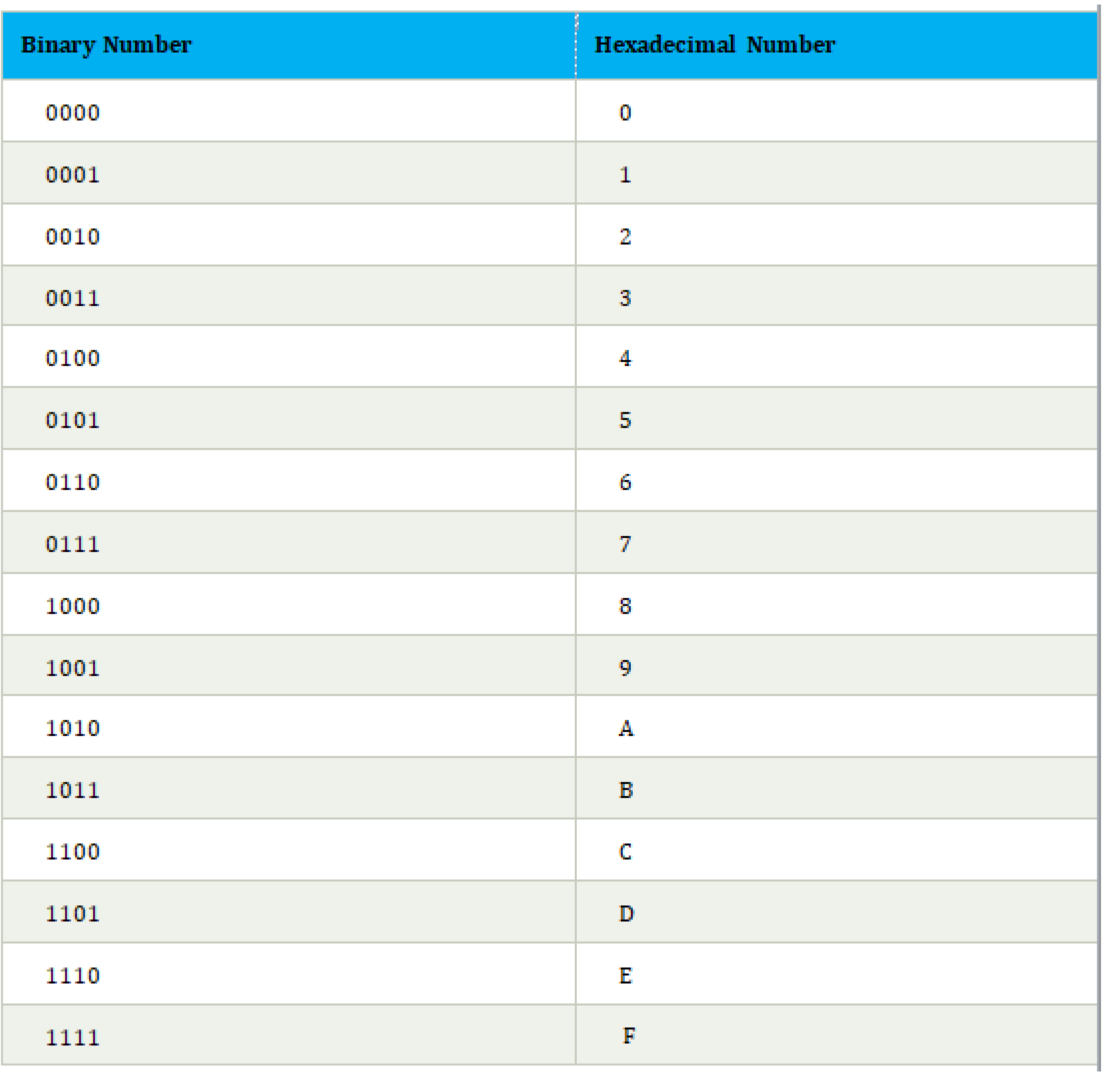
$(273)_8$, $(5644)_8$, $(0.5365)_8$, $(1123)_8$, $(1223)_8$.

Hexadecimal Number System

It is another technique to represent the number in the digital system called the **hexadecimal number system**. The number system has a base of 16 means there are total 16 symbols(0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F) used for representing a number. The single-bit representation of decimal values10, 11, 12, 13, 14, and 15 are represented by A, B, C, D, E, and F. Only 4 bits are required for representing a number in a hexadecimal number. Each set of bits has a distinct value between 0 and 15. There are the following characteristics of the octal number system:

Characteristics:

1. It has ten digits from 0 to 9 and 6 letters from A to F.
2. The letters from A to F defines numbers from 10 to 15.
3. It is also known as the base 16number system.
4. In hexadecimal number, the position of a digit represents the 0 power of the base(16). Example: 16^0
5. In hexadecimal number, the position of the last digit represents the x power of the base(16). Example: 16^x, where x represents the last position, i.e., 1



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

Examples:

$(FAC2)_{16}$, $(564)_{16}$, $(0ABD5)_{16}$, $(1123)_{16}$, $(11F3)_{16}$.

Conversion From One Number System To Another:

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

- Decimal to Other Base System
- Other Base System to Decimal
- Other Base System to Non-Decimal
- Binary to Octal
- Octal to Binary
- Binary to Hexadecimal
- Hexadecimal to Binary

Decimal to Other Base System

Steps

- **Step 1** – Divide the decimal number to be converted by the value of the new base.
- **Step 2** – Get the remainder from Step 1 as the rightmost digit (least significant digit) of new base number.
- **Step 3** – Divide the quotient of the previous divide by the new base.
- **Step 4** – Record the remainder from Step 3 as the next digit (to the left) of the new base number.

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

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

Example –

Decimal Number: 29_{10}

Calculating Binary Equivalent –

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

Step 4	3 / 2	1	1
Step 5	1 / 2	0	1

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

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

Other Base System to Decimal System

Steps

- **Step 1** – Determine the column (positional) value of each digit (this depends on the position of the digit and the base of the number system).
- **Step 2** – Multiply the obtained column values (in Step 1) by the digits in the corresponding columns.
- **Step 3** – Sum the products calculated in Step 2. The total is the equivalent value in decimal.

Example

Binary Number – 11101_2

Calculating Decimal Equivalent –

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

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

Other Base System to Non-Decimal System

Steps

- **Step 1** – Convert the original number to a decimal number (base 10).
- **Step 2** – Convert the decimal number so obtained to the new base number.

Example

Octal Number – 25_8

Calculating Binary Equivalent –

Step 1 – Convert to Decimal

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

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

Step 2 – Convert Decimal to Binary

Step	Operation	Result	Remainder
Step 1	21 / 2	10	1
Step 2	10 / 2	5	0
Step 3	5 / 2	2	1
Step 4	2 / 2	1	0
Step 5	1 / 2	0	1

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

Octal Number – 25_8 = Binary Number – 10101_2

Binary to Octal(Simple Method):

Steps

- **Step 1** – Divide the binary digits into groups of three (starting from the right).

- **Step 2** – Convert each group of three binary digits to one octal digit.

Example

Binary Number – 10101_2

Calculating Octal Equivalent –

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

Binary Number – 10101_2 = Octal Number – 25_8

Octal to Binary(Simple Method)

Steps

- **Step 1** – Convert each octal digit to a 3 digit binary number (the octal digits may be treated as decimal for this conversion).
- **Step 2** – Combine all the resulting binary groups (of 3 digits each) into a single binary number.

Example

Octal Number – 25_8

Calculating Binary Equivalent –

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

Octal Number – 25_8 = Binary Number – 10101_2

Binary to Hexadecimal(Simple Method):

Steps

- **Step 1** – Divide the binary digits into groups of four (starting from the right).
- **Step 2** – Convert each group of four binary digits to one hexadecimal symbol.

Example

Binary Number – 10101_2

Calculating hexadecimal Equivalent –

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

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

Hexadecimal to Binary(Simple Method)

Steps

- **Step 1** – Convert each hexadecimal digit to a 4 digit binary number (the hexadecimal digits may be treated as decimal for this conversion).
- **Step 2** – Combine all the resulting binary groups (of 4 digits each) into a single binary number.

Example

Hexadecimal Number – 15_{16}

Calculating Binary Equivalent –

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

Step 3	15_{16}	00010101_2

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

Conversion of Floating Point Numbers:

Binary Code:

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

Advantages of Binary Code

Following is the list of advantages that binary code offers.

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

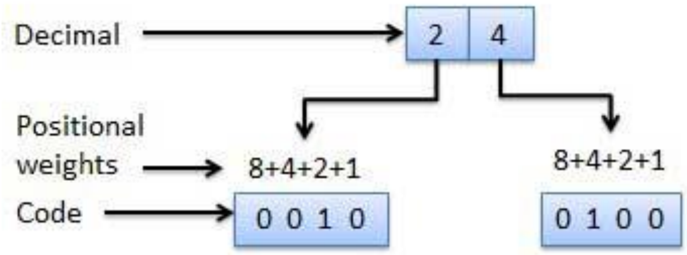
Classification of binary codes

The codes are broadly categorized into following four categories.

- Weighted Codes
- Non-Weighted Codes
- Binary Coded Decimal Code
- Alphanumeric Codes

Weighted Codes:

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

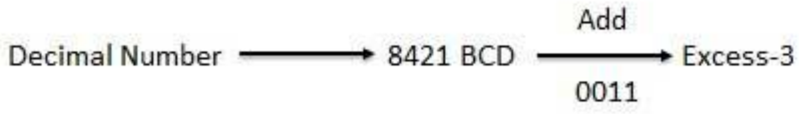


Non-Weighted Codes

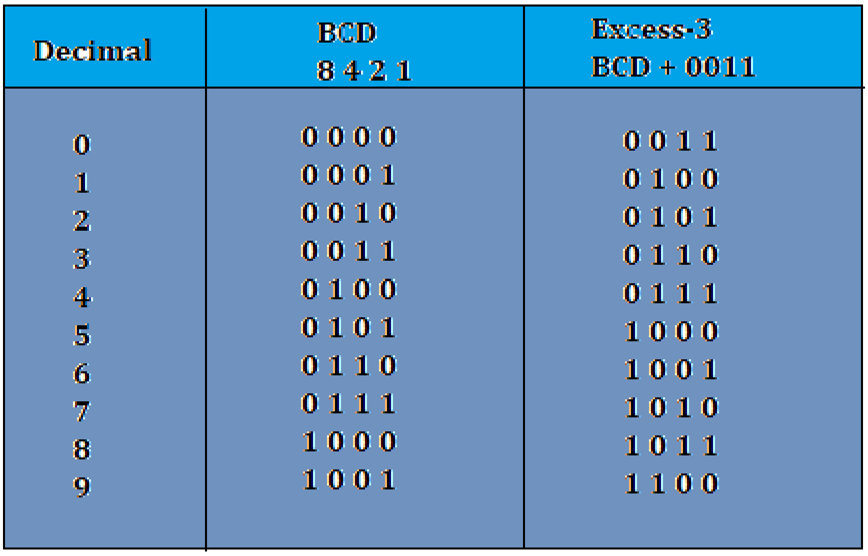
In this type of binary codes, the positional weights are not assigned. The examples of non-weighted codes are Excess-3 code and Gray code.

Excess-3 code

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



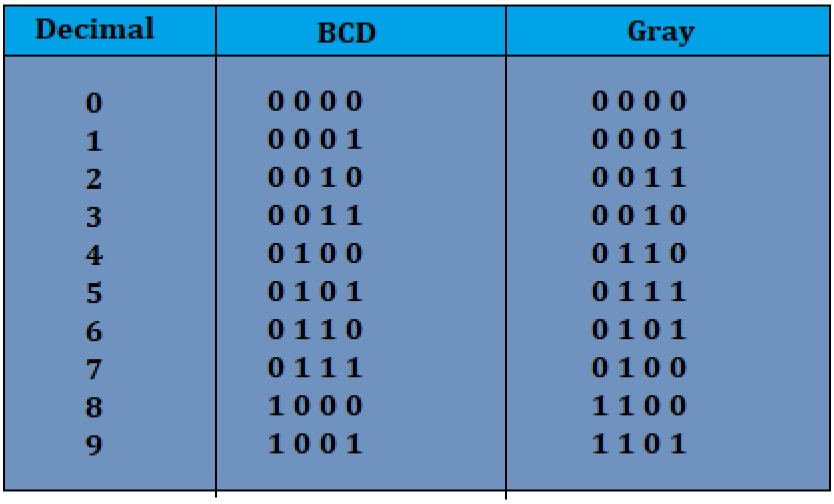
Example



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

Gray Code

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



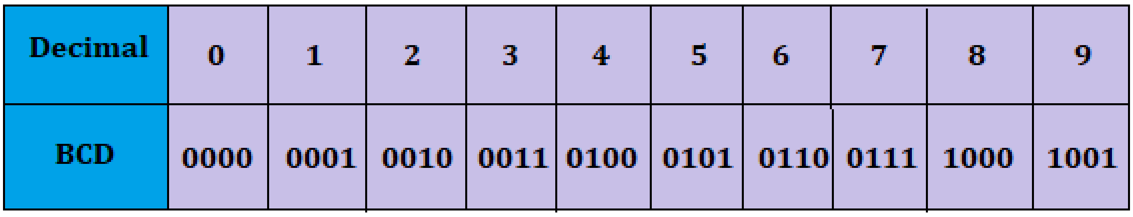
Decimal	BCD	Gray
0	0000	0000
1	0001	0001
2	0010	0011
3	0011	0010
4	0100	0110
5	0101	0111
6	0110	0101
7	0111	0100
8	1000	1100
9	1001	1101

Application of Gray code

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

Binary Coded Decimal (BCD) code

In this code each decimal digit is represented by a 4-bit binary number. BCD is a way to express each of the decimal digits with a binary code. In the BCD, with four bits we can represent sixteen numbers (0000 to 1111). But in BCD code only first ten of these are used (0000 to 1001). The remaining six code combinations i.e. 1010 to 1111 are invalid in BCD.



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

Advantages of BCD Codes

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

Disadvantages of BCD Codes

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

Alphanumeric codes

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

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

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

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

Error Detection and Correction Code

Error detection and correction code plays an important role in the transmission of data from one source to another. The noise also gets added into the data when it transmits from one system to another, which causes errors in the received binary data at other systems. The bits of the data may change(either 0 to 1 or 1 to 0) during transmission.

It is impossible to avoid the interference of noise, but it is possible to get back the original data. For this purpose, we first need to detect either an error **z** is present or not using error detection codes. If the error is present in the code, then we will correct it with the help of error correction codes.

Error detection code

The error detection codes are the code used for detecting the error in the received data **bitstream**. In these codes, some bits are included appended to the original bitstream.

Error detecting codes encode the message before sending it over the noisy channels. The encoding scheme is performed in such a way that the decoder at the receiving can find the errors easily in the receiving data with a higher chance of success.

Parity Code

In parity code, we add one parity bit either to the right of the LSB or left to the MSB to the original bitstream. On the basis of the type of parity being chosen, two types of parity codes are possible, i.e., even parity code and odd parity code.

Features of Error detection codes

These are the following features of error detection codes:

- These codes are used when we use message backward error correction techniques for reliable data transmission. A feedback message is sent by the receiver to inform the sender whether the message is received without any error or not at the receiver side. If the message contains errors, the sender retransmits the message.
- In error detection codes, in fixed-size blocks of bits, the message is contained. In this, the redundant bits are added for correcting and detecting errors.
- These codes involve checking of the error. No matter how many error bits are there and the type of error.
- Parity check, Checksum, and CRC are the error detection technique.

Error correction code

Error correction codes are generated by using the specific algorithm used for removing and detecting errors from the message transmitted over the noisy channels. The error-correcting codes find the correct number of corrupted bits and their positions in the message. There are two types of ECCs(Error Correction Codes), which are as follows.

Block codes

In block codes, in fixed-size blocks of bits, the message is contained. In this, the redundant bits are added for correcting and detecting errors.

Convolutional codes

The message consists of data streams of random length, and parity symbols are generated by the sliding application of the Boolean function to the data stream.

The hamming code technique is used for error correction.

Hamming Code

Hamming code is an example of a block code. The two simultaneous bit errors are detected, and single-bit errors are corrected by this code. In the hamming coding mechanism, the sender encodes the message by adding the unessential bits in the data. These bits are added to the specific position in the message because they are the extra bits for correction.

Unit :1

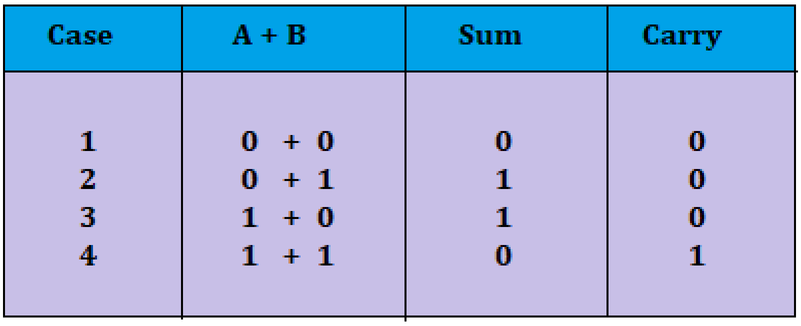
Chapter 2:Binary Arithmetic

Binary addition, Binary subtraction, Negative number representation, Subtraction using 1’s complement and 2’s complement, Binary multiplication and division, Arithmetic in octal number system, Arithmetic in hexadecimal number system, BCD and Excess – 3 arithmetic.

Binary Addition:

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

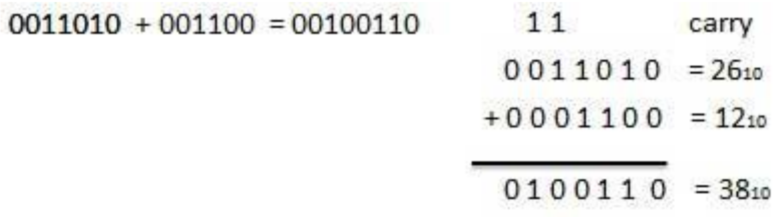
It is a key for binary subtraction, multiplication, division. There are four rules of binary addition.



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

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

Example – Addition



0011010 + 001100 = 00100110

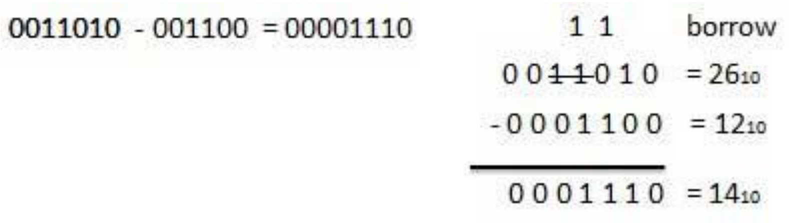
```
   1 1          carry
  0 0 1 1 0 1 0  = 26₁₀
 +0 0 0 1 1 0 0  = 12₁₀
 ______________
  0 1 0 0 1 1 0  = 38₁₀
```

Binary Subtraction:

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

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

Example – Subtraction



0011010 - 001100 = 00001110

1 1 borrow

0 0 ~~1 1~~ 0 1 0 $= 26_{10}$

-0 0 0 1 1 0 0 $= 12_{10}$

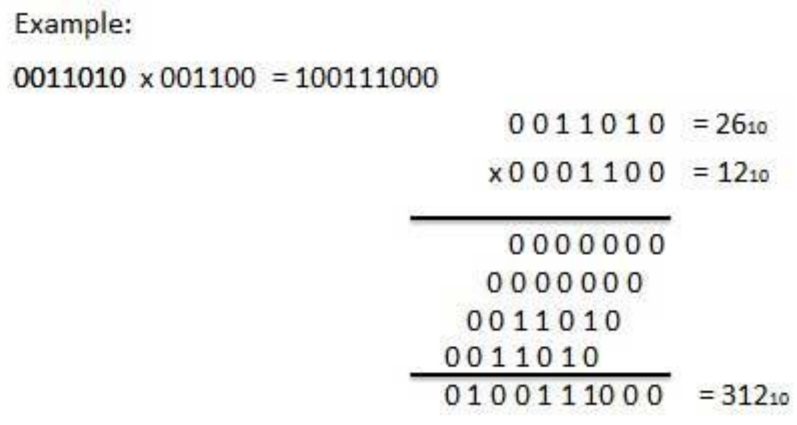
0 0 0 1 1 1 0 $= 14_{10}$

Binary Multiplication

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

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

Example – Multiplication



Example:

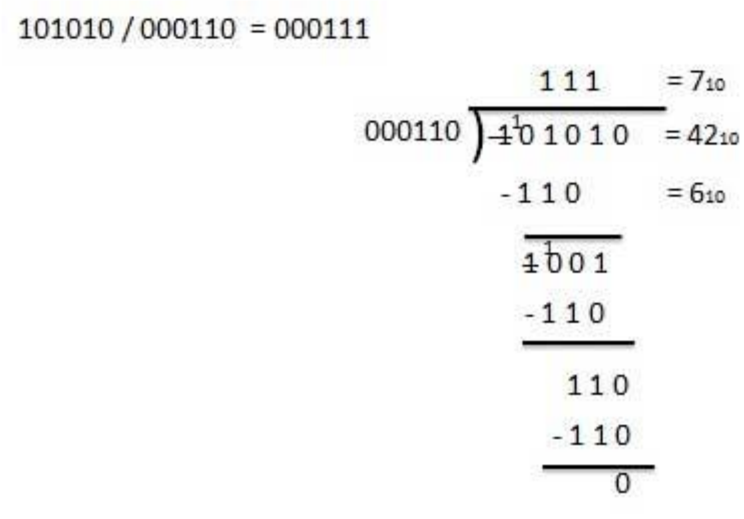
0011010 x 001100 = 100111000

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

Binary Division

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

Example – Division



```
101010 / 000110 = 000111

             111       = 7₁₀
000110 ) 101010        = 42₁₀
          -110         = 6₁₀
          ----
           1001
           -110
           ----
             110
            -110
            ----
               0
```

Complement:

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

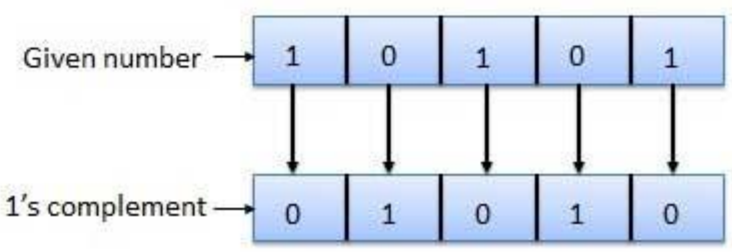
S.N.	Complement	Description
1	Radix Complement	The radix complement is referred to as the r's complement
2	Diminished Radix Complement	The diminished radix complement is referred to as the (r-1)'s complement

Binary System Complements

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

1's complement

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

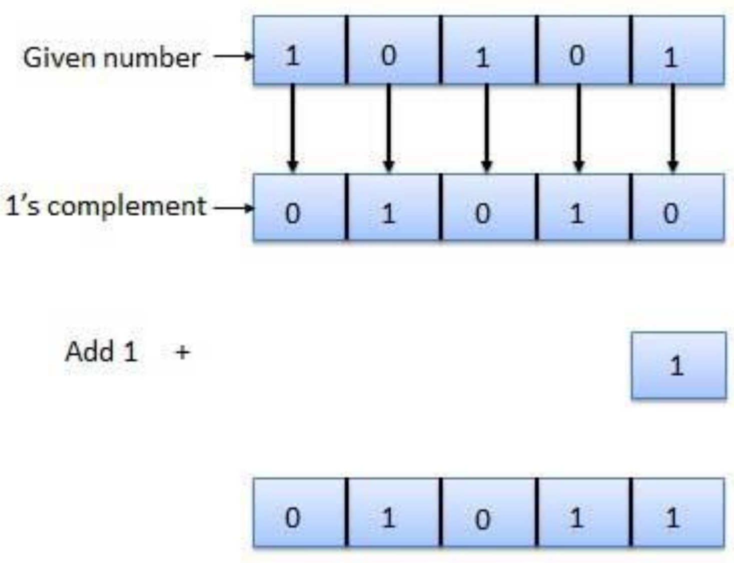


2's complement

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

2's complement = 1's complement + 1

Example of 2's Complement is as follows.



Subtraction using 1's complement and 2's complement:

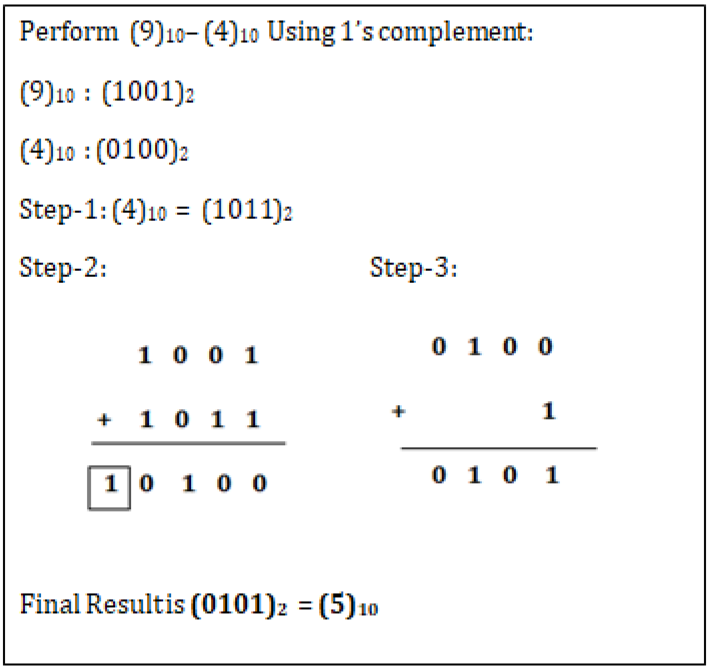
$(A)_2 - (B)_2$

Step-1: Convert the number to be subtracted that is $(B)_2$ to its 1's complement.

Step-2: Add 1's complement of $(B)_2$ and $(A)_2$ using Binary addition.

Step-3: If final carry is one ,then added $(A)_2$ to the result of addition obtain in step 2,to get the final result of $(A)_2 - (B)_2$.

Step-4: If the final carry is produce in step 2 is zero ,then the result obtain in step 2 is negative and in 1's complement form.So for the final result you have to complement all the bits.



Perform $(9)_{10}$– $(4)_{10}$ Using 1's complement:

$(9)_{10}$: $(1001)_2$

$(4)_{10}$: $(0100)_2$

Step-1: $(4)_{10}$ = $(1011)_2$

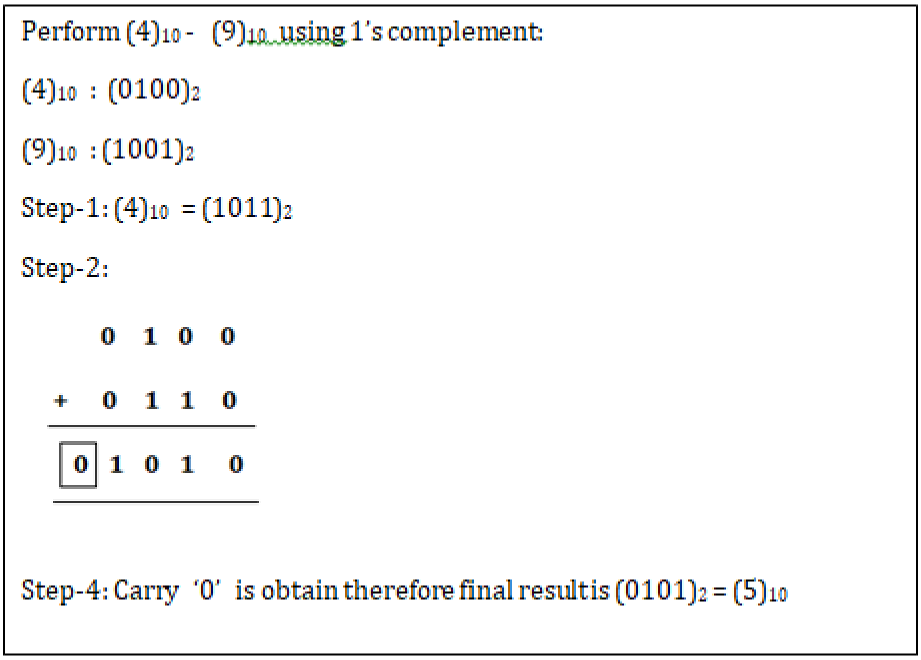
Step-2:

$$\begin{array}{r} 1\;0\;0\;1 \\ +\;1\;0\;1\;1 \\ \hline \boxed{1}\;0\;1\;0\;0 \end{array}$$

Step-3:

$$\begin{array}{r} 0\;1\;0\;0 \\ +\qquad\;\;1 \\ \hline 0\;1\;0\;1 \end{array}$$

Final Result is $\mathbf{(0101)_2}$ = $\mathbf{(5)_{10}}$



Perform $(4)_{10}$ - $(9)_{10}$ using 1's complement:

$(4)_{10}$: $(0100)_2$

$(9)_{10}$: $(1001)_2$

Step-1: $(4)_{10}$ = $(1011)_2$

Step-2:

$$\begin{array}{r} 0\;1\;0\;0 \\ +\;0\;1\;1\;0 \\ \hline \boxed{0}\;1\;0\;1\;0 \\ \hline \end{array}$$

Step-4: Carry '0' is obtain therefore final result is $(0101)_2$ = $(5)_{10}$

Subtraction using 2's complement:

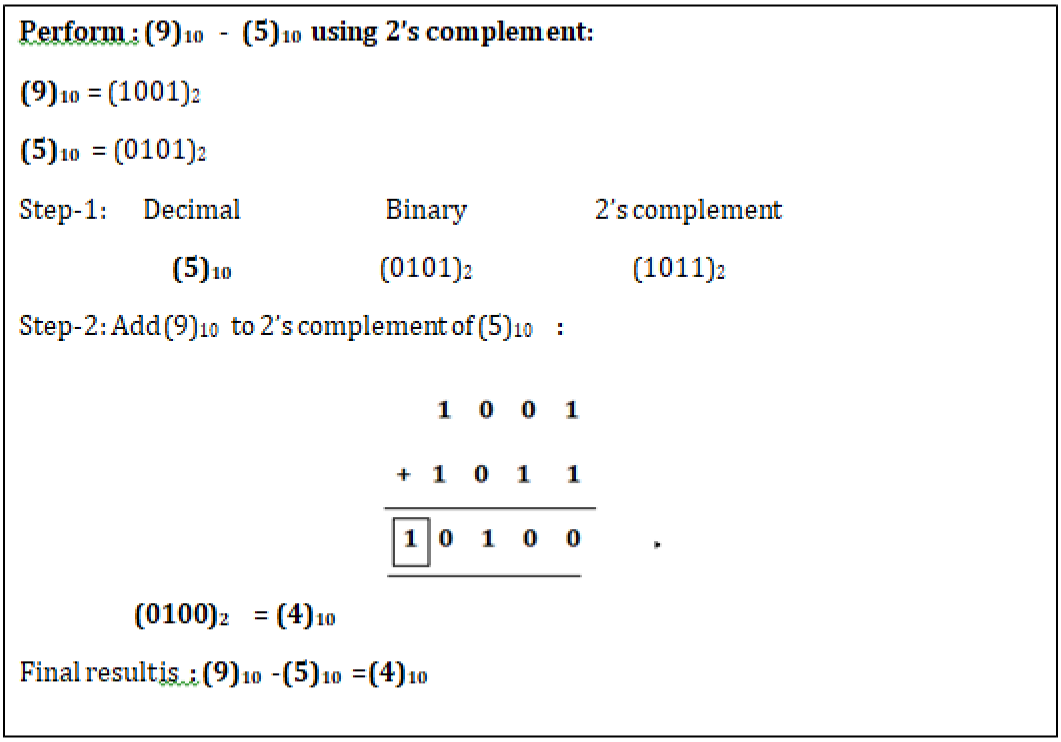
$(A)_2$ - $(B)_2$

Convert the number to be subtracted that is **$(B)_2$** to it's 2's complement.

Step-1: Add $(A)_2$ to the 2's complement to $(B)_2$

Step-2: If the carry is generated '1', then the result is positive and its true form.

Step-4: if the carry is not produced, then the result is negative and it's 2's complement form ,which has to be converted to the true form.



Perform : $(9)_{10}$ - $(5)_{10}$ using 2's complement:

$(9)_{10} = (1001)_2$

$(5)_{10} = (0101)_2$

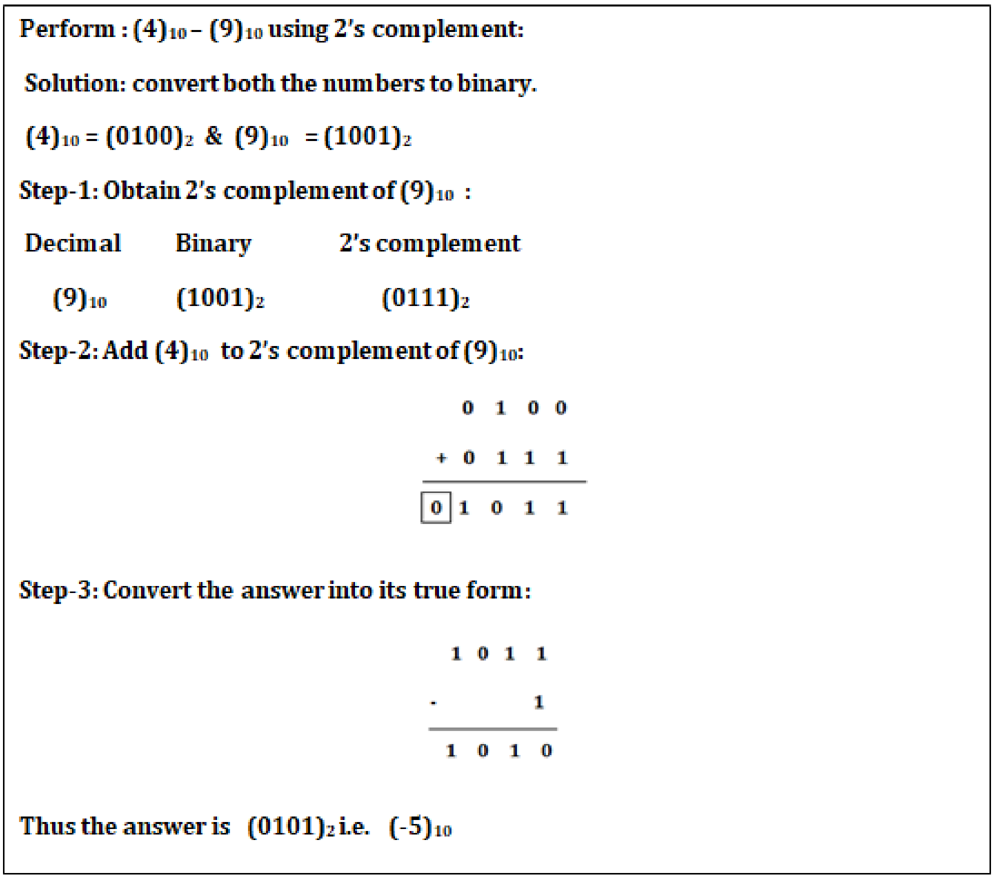
Step-1:	Decimal	Binary	2's complement
	$(5)_{10}$	$(0101)_2$	$(1011)_2$

Step-2: Add $(9)_{10}$ to 2's complement of $(5)_{10}$:

$$\begin{array}{r} 1\ \ 0\ \ 0\ \ 1 \\ +\ 1\ \ 0\ \ 1\ \ 1 \\ \hline \boxed{1}\ 0\ \ 1\ \ 0\ \ 0 \\ \hline \end{array} \quad .$$

$(0100)_2 = (4)_{10}$

Final resultis : $(9)_{10}$ -$(5)_{10}$ =$(4)_{10}$



Perform : $(4)_{10}$ – $(9)_{10}$ using 2's complement:

Solution: convert both the numbers to binary.

$(4)_{10} = (0100)_2$ & $(9)_{10} = (1001)_2$

Step-1: Obtain 2's complement of $(9)_{10}$:

Decimal	Binary	2's complement
$(9)_{10}$	$(1001)_2$	$(0111)_2$

Step-2: Add $(4)_{10}$ to 2's complement of $(9)_{10}$:

$$\begin{array}{r} 0\ \ 1\ \ 0\ \ 0 \\ +\ 0\ \ 1\ \ 1\ \ 1 \\ \hline \boxed{0}\ 1\ \ 0\ \ 1\ \ 1 \end{array}$$

Step-3: Convert the answer into its true form:

$$\begin{array}{r} 1\ \ 0\ \ 1\ \ 1 \\ -\ \ \ \ \ \ \ \ \ \ 1 \\ \hline 1\ \ 0\ \ 1\ \ 0 \end{array}$$

Thus the answer is $(0101)_2$ i.e. $(-5)_{10}$

Arithmetic of Octal Number System

Following are the characteristics of an octal number system.

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

Example

Octal Number – 12570_8

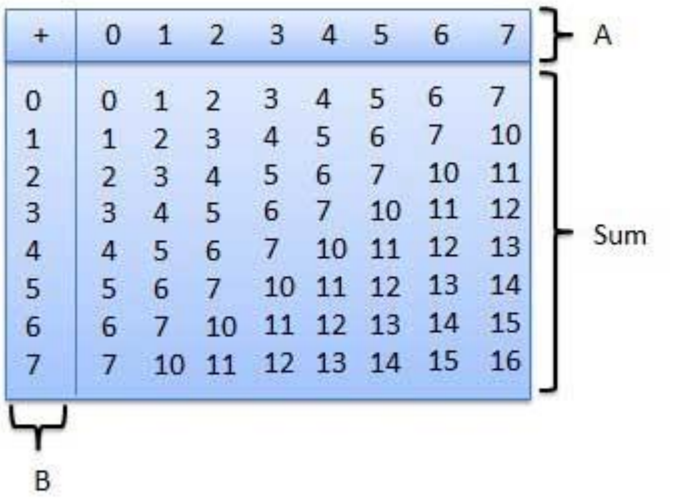
Calculating Decimal Equivalent –

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

Note – 12570_8 is normally written as 12570.

Octal Addition

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

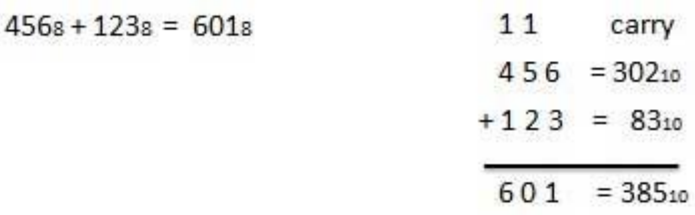


+	0	1	2	3	4	5	6	7
0	0	1	2	3	4	5	6	7
1	1	2	3	4	5	6	7	10
2	2	3	4	5	6	7	10	11
3	3	4	5	6	7	10	11	12
4	4	5	6	7	10	11	12	13
5	5	6	7	10	11	12	13	14
6	6	7	10	11	12	13	14	15
7	7	10	11	12	13	14	15	16

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

$6_8 + 5_8 = 13_8$.

Example – Addition



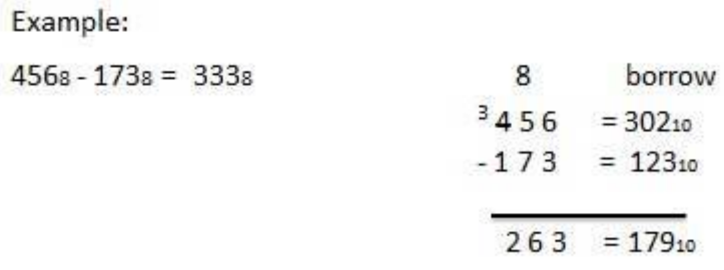
$456_8 + 123_8 = 601_8$

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

Octal Subtraction

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

Example – Subtraction



Example:

$456_8 - 173_8 = 333_8$

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

Arithmetic of Hexadecimal Number System:

Following are the characteristics of a hexadecimal number system.

- Uses 10 digits and 6 letters, 0,1,2,3,4,5,6,7,8,9,A,B,C,D,E,F.

- Letters represents numbers starting from 10. A = 10, B = 11, C = 12, D = 13, E = 14, F = 15.
- Also called base 16 number system.
- Each position in a hexadecimal number represents a 0 power of the base (16). Example – 16^0
- Last position in a hexadecimal number represents an x power of the base (16). Example – 16^x where x represents the last position - 1.

Example

Hexadecimal Number – $19FDE_{16}$

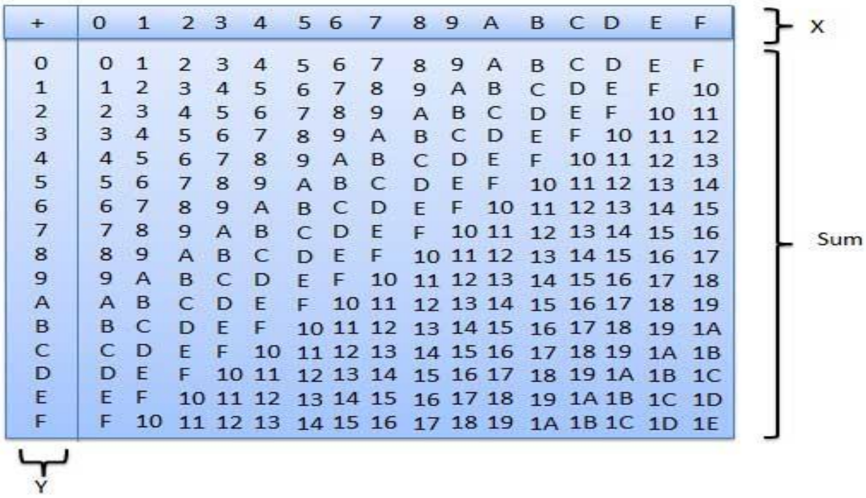
Calculating Decimal Equivalent –

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

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

Hexadecimal Addition

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

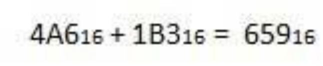


+	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
1	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	10
2	2	3	4	5	6	7	8	9	A	B	C	D	E	F	10	11
3	3	4	5	6	7	8	9	A	B	C	D	E	F	10	11	12
4	4	5	6	7	8	9	A	B	C	D	E	F	10	11	12	13
5	5	6	7	8	9	A	B	C	D	E	F	10	11	12	13	14
6	6	7	8	9	A	B	C	D	E	F	10	11	12	13	14	15
7	7	8	9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

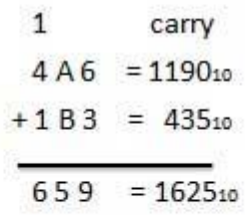
To use this table, simply follow the directions used in this example – Add A_{16} and 5_{16}. Locate A in the X column then locate the 5 in the Y column. The point in 'sum' area where these two columns intersect is the sum of two numbers.

$A_{16} + 5_{16} = F_{16}$.

Example – Addition



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

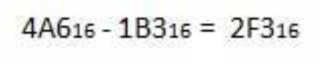


```
  1        carry
  4 A 6   = 1190₁₀
+ 1 B 3   =  435₁₀
---------
  6 5 9   = 1625₁₀
```

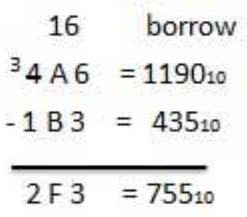
Hexadecimal Subtraction

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

Example - Subtraction



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



```
   16      borrow
 ³4 A 6   = 1190₁₀
- 1 B 3   =  435₁₀
---------
  2 F 3   =  755₁₀
```

UNIT-2

Chapter-1: Boolean Algebra and Logic Gates

Introduction, Logic (AND OR NOT), Boolean theorems, Boolean Laws, De Morgan's Theorem, Perfect Induction, Reduction of Logic expression using Boolean Algebra, Deriving Boolean expression from given circuit, exclusive OR and Exclusive NOR gates, Universal Logic gates, Implementation of other gates using universal gates, Input bubbled logic, Assertion level.

Introduction:

Binary logic:

A logic statement , is defined as a statement which is true if some condition is satisfied and false if that condition is not satisfied . For example ,a bulb turn ON ,if we close the switch, otherwise it is OFF.

Positive Logic:

- A "LOW" voltage level represents "logic 0" state and a comparatively "High" output voltage level represents "logic 1" state ,

-for example , 0 volts represent a logic 0 state and +5V represent logic 1. This is called as "positive logic".

Negative Logic:

- A "LOW" voltage level represent "logic 1" state and a "HIGH" output voltage level represents "logic 0" state,

- For example, 0 Volts represent a "logic 1" state and +5V represent "logic 0" .This is call as "Negative logic"

Logic Operations:

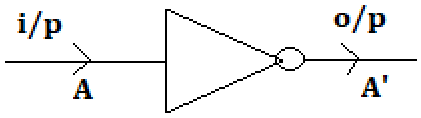
- Binary logic consist of binary variables and logical operations. The variables are denoted by letters such as A, B,C,X,Y,Z etc. Each variable has true only two possible value.

- There are three basic logic operations:

1)AND 2)OR 3)NOT

NOT Operation(Inversion):

- The NOT operation represents a process of logical inversion called as complementing.This operation change one logic level to the opposite logic level .

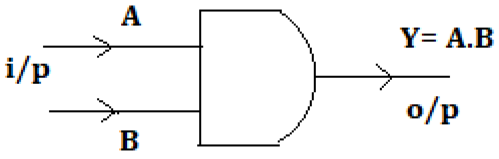


- The NOT operation is denoted by a bar(-) over the variable which is being inverted.

AND Operator:

-The AND operation produces a high (1) output only if all the input of a logic circuit as high(1).

-The "AND" operator represents logical multiplication. it is denoted by a dot between the two variables to be multiplied. i.e.

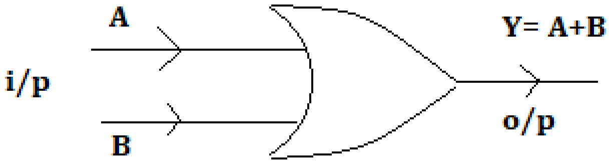


A.B ...Logical multiplication

-However sometimes this dot is not used and we denote the logical multiplication of A and B as AB

OR Operator:

-the OR-operation produces a HIGH(1) output when any one or all the inputs of a logic circuits are HIGH(1).



-The "OR" operator represents logical addition. it is denoted by a(+) sign between the two variables to be added.

-If A and B are to be added in a logic circuit then it is represented as.

A + B ... Logical addition

-And it is to be read as "A OR B"

Logic Gates:

1.Logic gates are electronics circuits that operate on one or more input signals and produce an output signal.

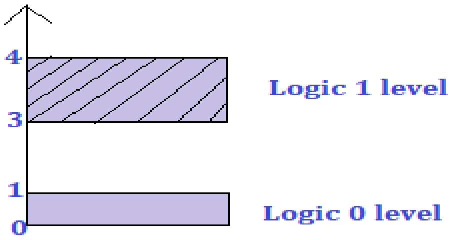
2.The voltages present in digital systems can have only one of the two possible values i.e. 0 or 1.

3.A specific voltage range corresponds to a logic 0 and another specific voltage range represent the logic 1 level .

4.For a particular system the voltage range for the 0 logic level may be 0 to 1V and the voltage range corresponding to logic 1 level may be 3 to 4V,But these ranges could be different for some other system.

4.The intermediate voltage range (1 to 3V) does not represent a 0 or 1.It does not represent any valid logical value.

5.This region is simply crossed while moving from a logic 0 to logic 1 or vice versa.



Voltage ranges corresponding to binary levels

Logic gates & its Types:

Digital electronic circuits operate with voltages of **two logic levels**namely Logic Low and Logic High. The range of voltages corresponding to Logic Low is represented with ‘0’. Similarly, the range of voltages corresponding to Logic High is represented with ‘1’.

The basic digital electronic circuit that has one or more inputs and single output is known as **Logic gate**. Hence, the Logic gates are the building blocks of any digital system. We can classify these Logic gates into the following three categories.

- Basic gates
- Universal gates

- Special gates

Now, let us discuss about the Logic gates come under each category one by one.

Basic Gates

In earlier chapters, we learnt that the Boolean functions can be represented either in sum of products form or in product of sums form based on the requirement. So, we can implement these Boolean functions by using basic gates. The basic gates are AND, OR & NOT gates.

AND gate

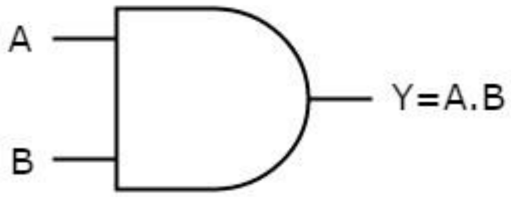
An AND gate is a digital circuit that has two or more inputs and produces an output, which is the **logical AND** of all those inputs. It is optional to represent the **Logical AND** with the symbol '.'.

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

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

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

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



This AND gate produces an output YY, which is the **logical AND** of two inputs A, B. Similarly, if there are 'n' inputs, then the AND gate produces an output, which is the logical AND of all those inputs. That means, the output of AND gate will be ‘1’, when all the inputs are ‘1’.

OR gate

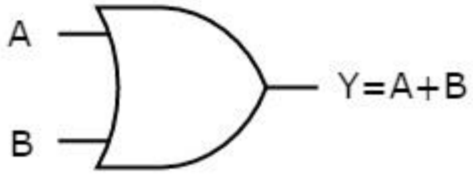
An OR gate is a digital circuit that has two or more inputs and produces an output, which is the logical OR of all those inputs. This **logical OR** is represented with the symbol ‘+’.

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

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

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

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



This OR gate produces an output YY, which is the **logical OR** of two inputs A, B. Similarly, if there are ‘n’ inputs, then the OR gate produces an output, which is the logical OR of all those inputs. That means, the output of an OR gate will be ‘1’, when at least one of those inputs is ‘1’.

NOT gate

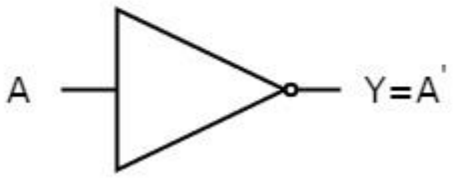
A NOT gate is a digital circuit that has single input and single output. The output of NOT gate is the **logical inversion** of input. Hence, the NOT gate is also called as inverter.

The following table shows the **truth table** of NOT gate.

A	**Y = A’**
0	1
1	0

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

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



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

Universal gates

NAND & NOR gates are called as **universal gates**. Because we can implement any Boolean function, which is in sum of products form by using NAND gates alone. Similarly, we can implement any Boolean function, which is in product of sums form by using NOR gates alone.

NAND gate

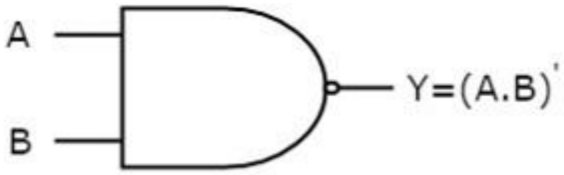
NAND gate is a digital circuit that has two or more inputs and produces an output, which is the **inversion of logical AND** of all those inputs.

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

A	**B**	**Y =** A.BA.B’
0	0	1
0	1	1
1	0	1
1	1	0

Here A, B are the inputs and Y is the output of two input NAND gate. When both inputs are ‘1’, the output, Y is ‘0’. If at least one of the input is zero, then the output, Y is ‘1’. This is just opposite to that of two input AND gate operation.

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



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

NOR gate

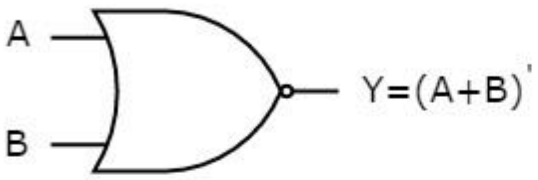
NOR gate is a digital circuit that has two or more inputs and produces an output, which is the **inversion of logical OR** of all those inputs.

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

A	**B**	**Y =** A+BA+B’
0	0	1
0	1	0
1	0	0
1	1	0

Here A, B are the inputs and Y is the output. If both inputs are ‘0’, then the output, Y is ‘1’. If at least one of the input is ‘1’, then the output, Y is ‘0’. This is just opposite to that of two input OR gate operation.

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



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

Special Gates

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

Ex-OR gate

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

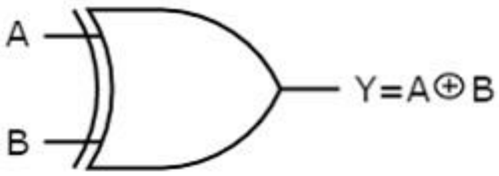
The following table shows the **truth table** of 2-input Ex-OR gate.

A	**B**	**Y = A⊕B**
0	0	0
0	1	1
1	0	1
1	1	0

Here A, B are the inputs and Y is the output of two input Ex-OR gate. The truth table of Ex-OR gate is same as that of OR gate for first three rows. The only modification is in the fourth row. That means, the output YY is zero instead of one, when both the inputs are one, since the inputs having even number of ones.

Therefore, the output of Ex-OR gate is ‘1’, when only one of the two inputs is ‘1’. And it is zero, when both inputs are same.

Below figure shows the **symbol** of Ex-OR gate, which is having two inputs A, B and one output, Y.



Ex-OR gate operation is similar to that of OR gate, except for few combinationss of inputs. That’s why the Ex-OR gate symbol is represented like that. The output of Ex-OR gate is ‘1’, when odd number of ones present at the inputs. Hence, the output of Ex-OR gate is also called as an **odd function**.

Ex-NOR gate

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

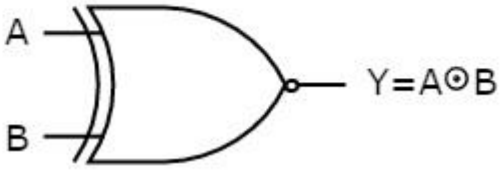
The following table shows the **truth table** of 2-input Ex-NOR gate.

A	B	Y = A⍰B
0	0	1
0	1	0
1	0	0
1	1	1

Here A, B are the inputs and Y is the output. The truth table of Ex-NOR gate is same as that of NOR gate for first three rows. The only modification is in the fourth row. That means, the output is one instead of zero, when both the inputs are one.

Therefore, the output of Ex-NOR gate is ‘1’, when both inputs are same. And it is zero, when both the inputs are different.

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



Ex-NOR gate operation is similar to that of NOR gate, except for few combinationss of inputs. That's why the Ex-NOR gate symbol is represented like that. The output of Ex-NOR gate is '1', when even number of ones present at the inputs. Hence, the output of Ex-NOR gate is also called as an **even function**.

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

Boolean Algebra:

The logical symbol 0 and 1 are used for representing the digital input or output. The symbols "1" and "0" can also be used for a permanently open and closed digital circuit. The digital circuit can be made up of several logic gates. To perform the logical operation with minimum logic gates, a set of rules were invented, known as the **Laws of Boolean Algebra**. These rules are used to reduce the number of logic gates for performing logic operations.

The Boolean algebra is mainly used for simplifying and analyzing the complex Boolean expression. It is also known as **Binary algebra** because we only use binary numbers in this. **George Boole** developed the binary algebra in **1854.**

Rules in Boolean algebra

1. Only two values(1 for high and 0 for low) are possible for the variable used in Boolean algebra.
2. The overbar(-) is used for representing the complement variable. So, the complement of variable C is represented as .
3. The plus(+) operator is used to represent the ORing of the variables.
4. The dot(.) operator is used to represent the ANDing of the variables.

Properties of Boolean algebra

These are the following properties of Boolean algebra:

Annulment Law

When the variable is AND with 0, it will give the result 0, and when the variable is OR with 1, it will give the result 1, i.e.,

B.0 = 0

B+1 = 1

Identity Law

When the variable is AND with 1 and OR with 0, the variable remains the same, i.e.,

B.1 = B

B+0 = B

Idempotent Law

When the variable is AND and OR with itself, the variable remains same or unchanged, i.e.,

B.B = B

B+B = B

Complement Law

When the variable is AND and OR with its complement, it will give the result 0 and 1 respectively.

B.B' = 0

B+B' = 1

Double Negation Law

This law states that, when the variable comes with two negations, the symbol gets removed and the original variable is obtained.

((A)')' = A

Commutative Law

This law states that no matter in which order we use the variables. It means that the order of variables doesn't matter in this law.

A.B = B.A

A+B = B+A

Associative Law

This law states that the operation can be performed in any order when the variables priority is of same as '*' and '/'.

(A.B).C = A.(B.C)

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

Distributive Law

This law allows us to open up of brackets. Simply, we can open the brackets in the Boolean expressions.

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

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

Absorption Law

This law allows us for absorbing the similar variables.

B+(B.A) = B

B.(B+A) = B

De Morgan Law:

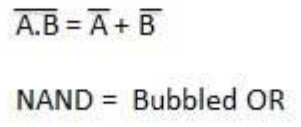
The operation of an OR and AND logic circuit will remain same if we invert all the inputs, change operators from AND to OR and OR to AND, and invert the output.

(A.B)' = A'+B'

(A+B)' = A'.B'

De Morgan has suggested two theorems which are extremely useful in Boolean Algebra. The two theorems are discussed below.

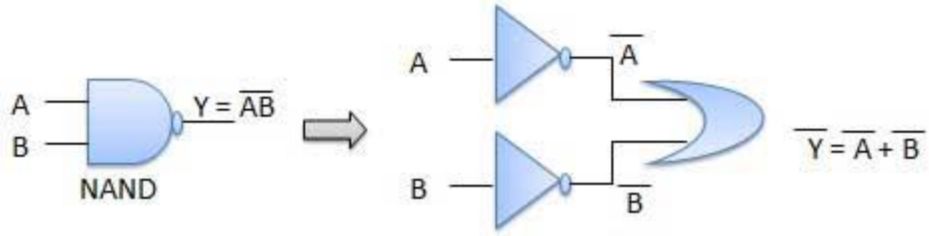
Theorem 1



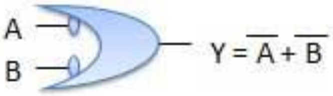
$\overline{A.B} = \overline{A} + \overline{B}$

NAND = Bubbled OR

- The left hand side (LHS) of this theorem represents a NAND gate with inputs A and B, whereas the right hand side (RHS) of the theorem represents an OR gate with inverted inputs.
- This OR gate is called as **Bubbled OR**.

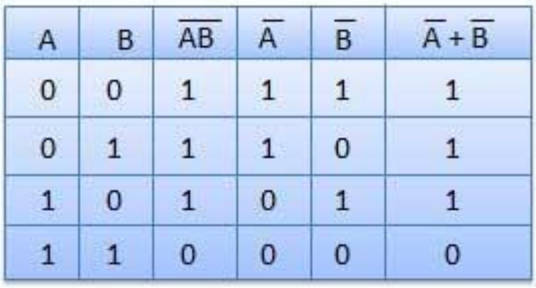


NAND $\equiv$ Bubbled OR



Bubbled OR

Table showing verification of the De Morgan's first theorem −



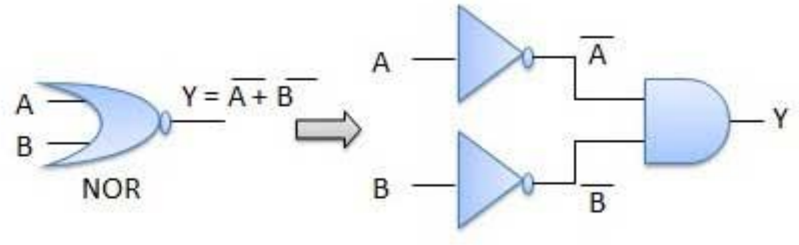
A	B	$\overline{AB}$	$\overline{A}$	$\overline{B}$	$\overline{A}+\overline{B}$
0	0	1	1	1	1
0	1	1	1	0	1
1	0	1	0	1	1
1	1	0	0	0	0

Theorem 2

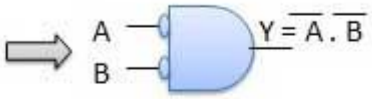
$$\overline{A + B} = \overline{A}.\overline{B}$$

NOR = Bubbled AND

- The LHS of this theorem represents a NOR gate with inputs A and B, whereas the RHS represents an AND gate with inverted inputs.
- This AND gate is called as **Bubbled AND**.

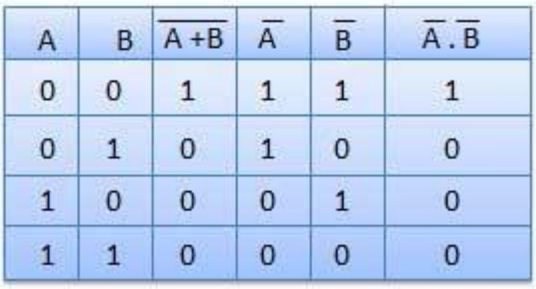


NOR ≡ Bubbled AND



Bubbled AND

Table showing verification of the De Morgan's second theorem −



A	B	$\overline{A+B}$	$\overline{A}$	$\overline{B}$	$\overline{A}\,.\,\overline{B}$
0	0	1	1	1	1
0	1	0	1	0	0
1	0	0	0	1	0
1	1	0	0	0	0

UNIT-2

Chapter-2: Minterm, Maxterm and Karnaugh Maps

Introduction, minterms and sum of minterm form, maxterm and Product of maxterm form, Reduction technique using Karnaugh maps – 2/3/4/5/6 variable K-maps, Grouping of variables in K-maps, K-maps for product of sum form, minimize Boolean expression using K-map and obtain K-map from Boolean expression, Quine Mc Cluskey Method.

Any logical expression can be expressed in two standard form as given below:

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

1.Sum of the product :

The sum and product in the SOP form are not the actual addition or multiplication. Rather, they are the OR and AND functions.

The SOP remembers where is 1. It doesn't cares about 0.

The SOP form is used to create boolean expression through the given truth table data.

It is called as SOP form as it contains the **sum** of **product terms**. For example, let A, B, and C are three literals or inputs of any combinational circuits and Y be the function, then the SOP expression may be written as given below:

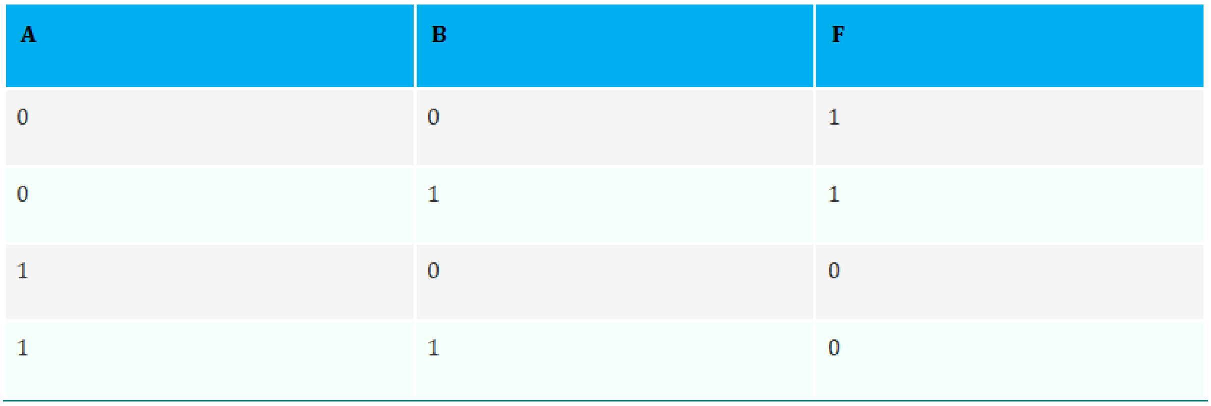
Y = ABC + AB'C + AC

Or

Y = A'BC + B'C + A'C'

etc. are some of the examples of SOP form boolean expression.

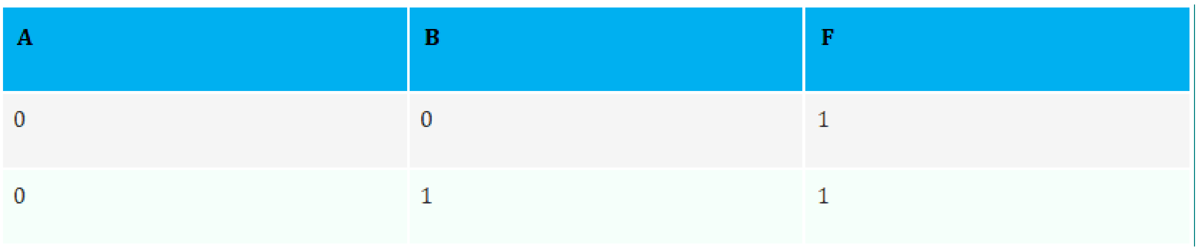
And if you want to find out the boolean expression in SOP form through the given truth table, then consider an example and understand about it. Let the given truth table is:



A	B	F
0	0	1
0	1	1
1	0	0
1	1	0

Here A and B are the two literals or variables and F is the function.

As we have already discussed that SOP only cares about 1. That is, SOP only look-up that where the function F holds 1 as its value. In the above truth table, if we remove all the rows where F holds 0, then the table will become:



A	B	F
0	0	1
0	1	1

Therefore, the expression in the SOP form will be:

F(A,B) = A'B' + A'B

Note - While writing the expression, we have to write the complement version of all those literals that contains 0 as its value to make it 1. That is, if A contains 0 as its value in first row and if we take the complement of A, then A will become A' and therefore 0 will become 1 as its value in that row.

As we can see from the above expression, that is **A'B'+A'B** which is the sum of **product of few literals**, therefore it is called as SOP or **Sum of Product** form.

In SOP form boolean expression, each product term consist of one or more than one literals. Here literals may be in its complemented or uncomplemented (natural or original) form.

Now let's take an example to understand it in a more clear way. Here we have designed a table in which there are 3 variables. Therefore, with 3 variables, we can create total of **2^3** or 8 combinations from **000** to **111** where first digit belongs to first variable, second digit belongs to second variable and so on as shown in the table given below:

Decimal	Binary	Minterm	Designation
0	000	A'B'C'	m_0
1	001	A'B'C	m_1
2	010	A'BC'	m_2
3	011	A'BC	m_3
4	100	AB'C'	m_4
5	101	AB'C	m_5
6	110	ABC'	m_6
7	111	ABC	m_7

As you can see from the above table, some literals that contains 0 is complemented to reverse and make it 1.

Row that contains the decimal number value as 0, then its binary representation will be 000 (for three-variable function), here A belongs to first digit which is 0 in this row, B belongs to second digit which is also 0, and the C belongs to third digit which is again 0 in this row. And here all the literals holds the value 0, therefore to make it 1 as result, we have to complement all the literal and we have already done it. Because to get output as 1 in case of AND gate, we must have all the input's value as 1, and here in first row ABC contains 000, therefore to make it 1, we have put all the literal's value as 1, it means A=1, B=1, and C=1. For this, we simply have complemented the literal, that is A', B', and C'.

In the same way as told above, we have done the required complement of all the literals in each and every rows where it contains 0.

Minterm:

Minterm is defined as, in any function there are some particular number of variables and if all the variables are in the product term in any ways either in complemented or uncomplemented form is called as minterm. For example, let's take an expression:

F(A,B,C) = ABC + BC + A'BC' + A'B

Here, we have a function of three variables that has the expression as given above. As we can see from the above boolean expression in SOP form, we have only two product term that holds all the three variables or literals which is **ABC** and **A'BC'**. Therefore, these two terms can be called as minterms.

Or in simple words, minterms are all those product term that contains all the variables that are in the function. Here literals or variables may be either in its complemented or uncomplemented form.

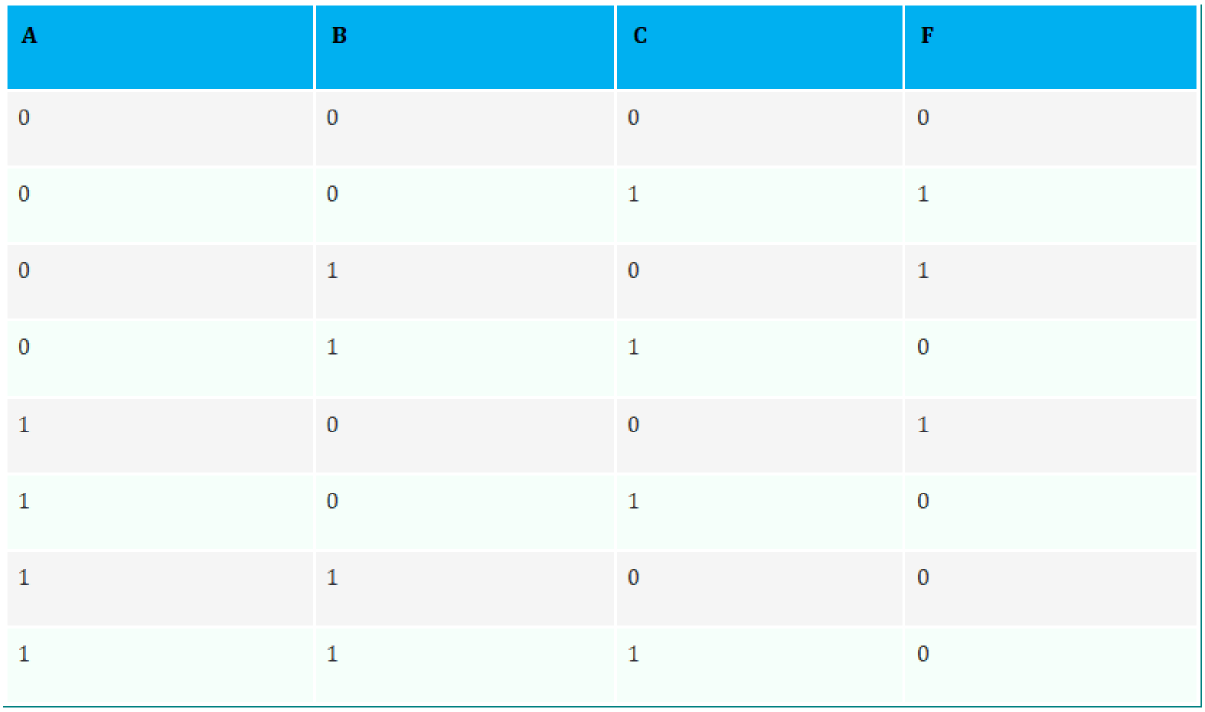
How to Calculate Total number of Minterm ?

In a n-variable function, **Total number of Minterm = 2^n**.

For example, in a 3-variable function, the **total number of minterm = 2^3** or **2x2x2** or **8** minterms.

How to Create SOP Boolean Expression using Truth Table ?

Now let's take an example to understand about how to create a SOP form boolean expression with the help of given truth table.The given truth table is:



A	B	C	F
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	0

In above truth table, we have 1 at some places of function. We have total of three 1s present at 1st, 2nd and 4th position. As at 0th, 3rd, 5th, 6th and 7th position, there is 0.

Therefore, we have three minters, that is $\mathbf{m_1}$, $\mathbf{m_2}$, and $\mathbf{m_4}$.

Now, we have to write the literal's product term three times as given below in SOP form:

ABC + ABC + ABC

Now if we will look at the truth table, we have three minterm. And in the 1st row, the literals A holds 0, B holds 0, and C holds 1. As A and B holds 0, therefore we have to complement it to make it 1. Similarly follow the same procedure for 2nd and 4th row. Therefore after performing this procedure, then the above equation will become:

A'B'C + A'BC' + AB'C'

Here, A'B'C is m_1 or first minterm, A'BC' is m_2 or second minterm, and AB'C' is m_4 or fourth minterm.

Finally, we can write the complete boolean expression in SOP form as:

F(A,B,C) = A'B'C + A'BC' + AB'C'

How to Design Truth Table using SOP Boolean Expression ?

After learning all the steps and procedure as given above, you can also design the truth table itself with the help of given boolean expression in SOP form.

Let's take an example to understand it in a clear and practical way. Here is the SOP form boolean expression:

AB'C + AB'C' + A'BC'

As we can clearly clarify that the above expression is given in SOP form. And in SOP form, the uncomplemented literal or variable holds 1 and the complemented literal holds 0. Therefore, binary representation of the first product term in above expression that is **AB'C** is equal to **101**, **AB'C'** equals **100**, and **A'BC'** equals **010**. And the binary number **101**, **100**, and **010** represents **5**, **4**, and **2** respectively in decimal.

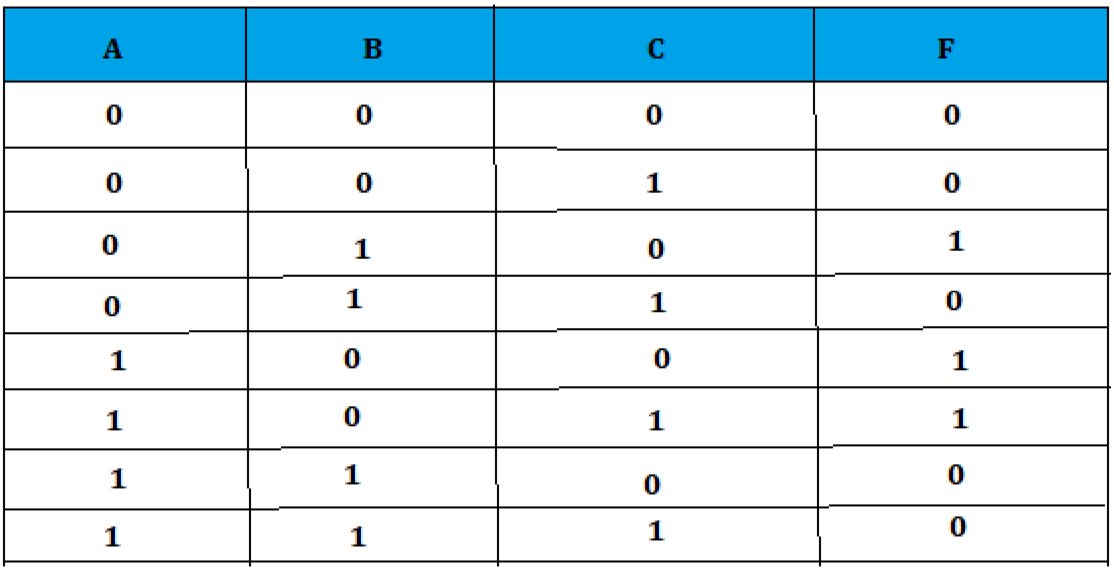
Therefore, we have three minters, that is $\mathbf{m_2}$, $\mathbf{m_4}$, and $\mathbf{m_5}$. And we can represent the function as:

F(A,B,C) = $\sum_m(2,4,5)$

And the truth table with minterm and designation of the above equation will be:

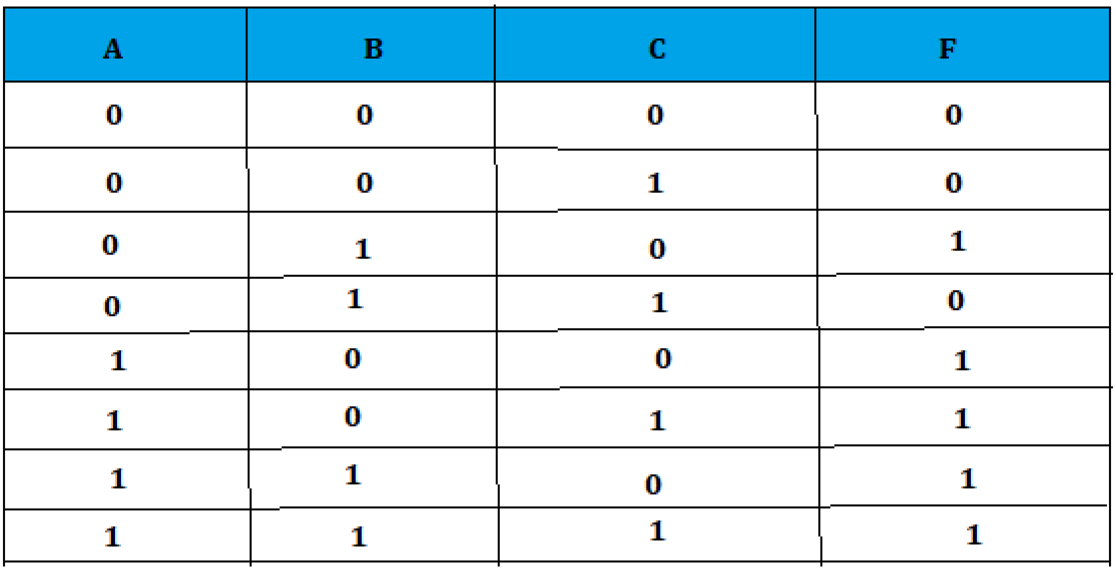
Decimal	Binary	Minterm	Designation
2	010	A'BC'	m_2
4	100	AB'C'	m_4
5	101	AB'C	m_5

And the complete truth table only with three variable and function of above equation will be:



A	B	C	F
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	1
1	1	0	0
1	1	1	0

SOP Form Example:Let's take an another example of SOP form. The truth table is given here:



A	B	C	F
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	1

In boolean algebra, the above function **F** can be expressed either in SOP or in POS form, but now we are dealing only about SOP form. Therefore let's get started to find out the boolean expression in SOP using given truth table.

As we all know that 1 indicates to high and 0 indicates to low. The SOP expression is written only when the function is high, and from the above truth table, we have total of 5 times where the function is high. It means we have total of 5 product terms that will be sum to make it in SOP form.

At first, the function is high when A is low, B is high, and C is low. Therefore, the product term in this case will be **A'BC'**. Hence where the variable is low, we have to complement it to make it high.

And as we all know that the AND operator that operates on three variables will give the output as 1 only when all the three input variables are 1.

Therefore, we must have to complement the variable (if it carries 0) to make it 1. And if the variable holds 1, then we have to leave it in its uncomplemented form or natural form.

Therefore, the above truth table gives the function as written below:

F(A,B,C) = A'BC' + AB'C' + AB'C + ABC' + ABC

The expression written above through the given truth table are in standard SOP or canonical SOP form.

Therefore the minterm will be $\mathbf{m_2}$, $\mathbf{m_4}$, $\mathbf{m_5}$, $\mathbf{m_6}$, and $\mathbf{m_7}$. And finally the function can be written as:

F(A,B,C) = $m_2 + m_4 + m_5 + m_6 + m_7$

Or,

F(A,B,C) = $\sum_m(2,4,5,6,7)$

POS(Product of the sum):

In **Product of Sum** (POS) form, literals are as sum terms and all the sum terms are ANDed to get the expression in POS form.

In other words, you can says that POS contains sum terms which are AND together.

Here each sum term may contain one or more than one literals or variables. And literals may be in its complemented or uncomplemented form. For example, the expression in POS form looks like:

(A'+B'+C).(A+B).(B'+C).(A+B+C')

Maxterm

All those sum terms that are present in the POS form boolean expression that contains all the variables are called as maxterms. For example, if a function is defined as:

F(A,B,C) = (A+B+C).(A+B').(A'+C').(A+B'+C)

As we can clearly see that there are two sum terms that contains all the three variables of the function that are **(A+B+C)** and **(A+B'+C)**. Therefore, these two variables can be called as maxterm.

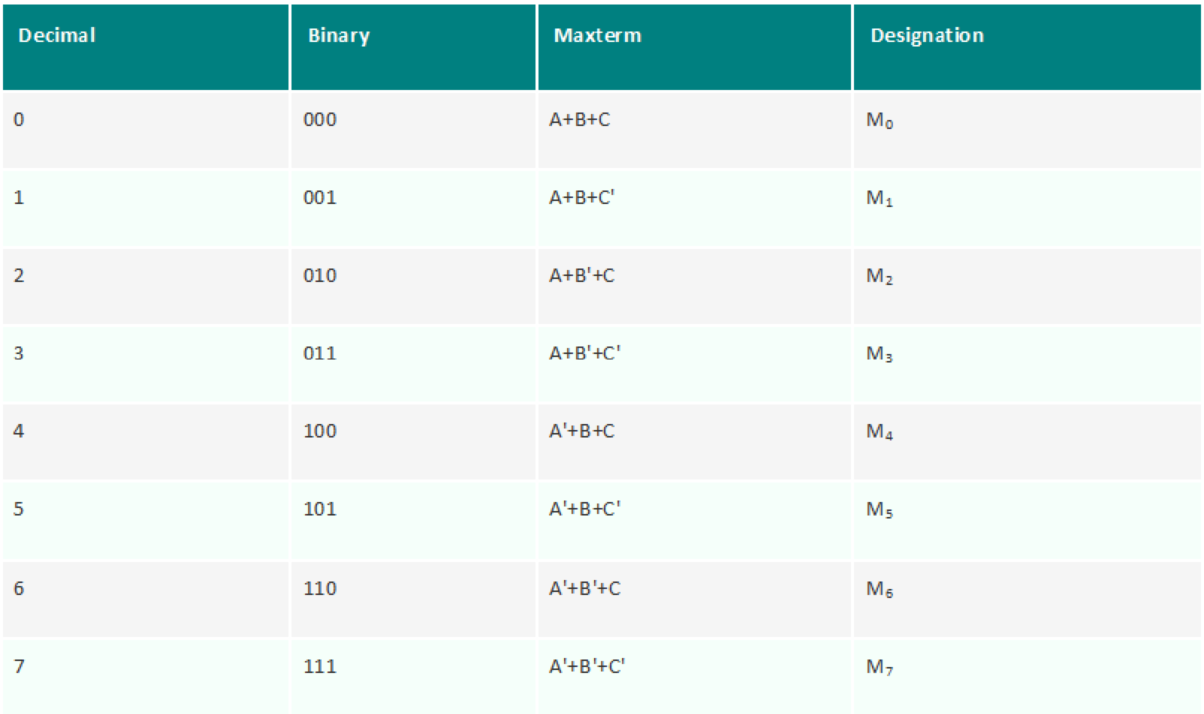
How to Calculate Total number of Maxterm ?

To calculate the total number of maxterms that can be present in a function of n variables, we uses the following formula:

total number of maxterm = 2^n

where n is the number of variable present in the function. Therefore, in a 3-variable function, we will have total number of maxterm $\mathbf{2^3}$ which is equal to 8. Therefore we can have 8 maxterms in a 3-variable function.

Let's consider the following table:



Decimal	Binary	Maxterm	Designation
0	000	A+B+C	M_0
1	001	A+B+C'	M_1
2	010	A+B'+C	M_2
3	011	A+B'+C'	M_3
4	100	A'+B+C	M_4
5	101	A'+B+C'	M_5
6	110	A'+B'+C	M_6
7	111	A'+B'+C'	M_7

In above truth table, we have three variable function, then the total maxterm can be calculated as:

Total Maxterm = 2^3 = 8

POS form only cares about 0. Therefore, to get the output as 0 from one maxterm or one sum term, all the three variables must holds 0 as its value. That is, **A+B+C** will be 0 only when A contains 0, B contains 0, and C also contains 0. As the **+** operator is called as OR operator, and in case of this operator, if any of the all input is 1, then the output will be 1, therefore we have to make all the input 0 to get 0 at output. Therefore, if any of the variable holds 1, we have complement that variable to reverse its value and make it 0.

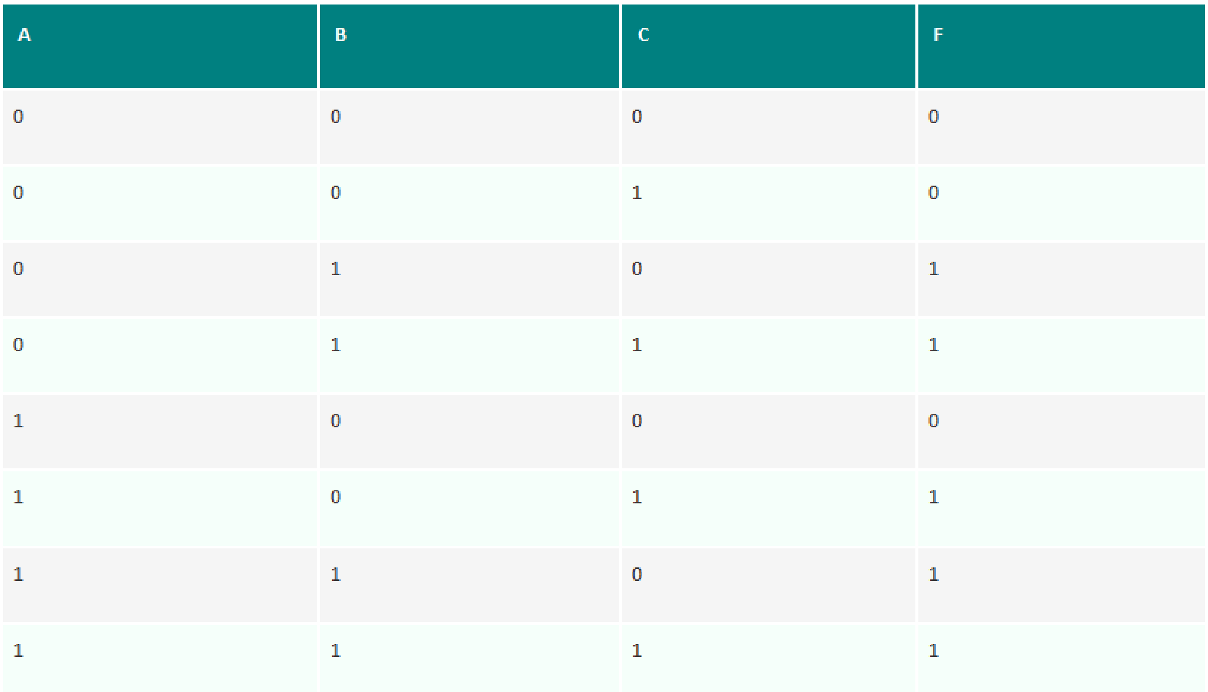
In above truth table, in first row, value of all the three variables are already 0. But in second row, we have binary representation as **001**. Here first digit belongs to A, second digit belongs to B, and the third digit belongs to C. It means A contains 0, B contains 0, and C contains 1 in second row where the decimal value is 1. Therefore, to make **(A+B+C)** 0. We have to complement C to make it 0, and then the expression or sum term **A+B+C** also gives 0. Therefore in that row, we have written **(A+B+C')** as maxterm to get 0. In this way, all the maxterms are written as shown in the above table.

Note - We have to complement all the variable where it contains 1 as its value.

How to Create POS Boolean Expression using Truth Table ?

To understand POS form in a more clear way we must have to understand about how anyone can create boolean expression in POS form using the given truth table.

Let's suppose we have the following truth table as given:



A	B	C	F
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

As we have already discussed that in POS form, we only cares about 0. Therefore, we only have to look-up where the function holds 0 or in which row, function has 0.

From the above truth table, we have 0 present in function at **0th**, **1st**, **4th**, and **7th** place. It means total of 4 zero that the function holds. Therefore we have total 4 maxterm. Then write the sum term with all the three variable and AND all the 4 maxterm as given below:

(A+B+C).(A+B+C).(A+B+C).(A+B+C)

Now as you can see from the above table, at **0th** row, all the three variable holds 0. And at **1st** row, A and B contains 0 but C contains 1, therefore we have to complement the variable C here. Or in second sum term, we have to write C in its complemented form. In the same way we have to write the complemented version of the variable where it holds 1 in the above truth table. After performing this for all the four sum term or maxterm, here is the final and correct POS form boolean expression you will get:

(A+B+C).(A+B+C').(A'+B+C).(A'+B'+C')

And below is the way to define the function with the help of above POS form expression:

$F(A,B,C) = \Pi_M(0,1,4,7)$

Or,

$F(A,B,C) = (M_0,M_1,M_4,M_7)$

After understanding the above topic, you can also define the function in POS form using any given truth table.

How to Design Truth Table using POS Boolean Expression ?

Under this topic, we will learn how to create truth table using the given boolean expression in POS form.

Let's take an example. Here is the boolean expression given in POS form:

(A+B'+C').(A'+B'+C).(A+B'+C)

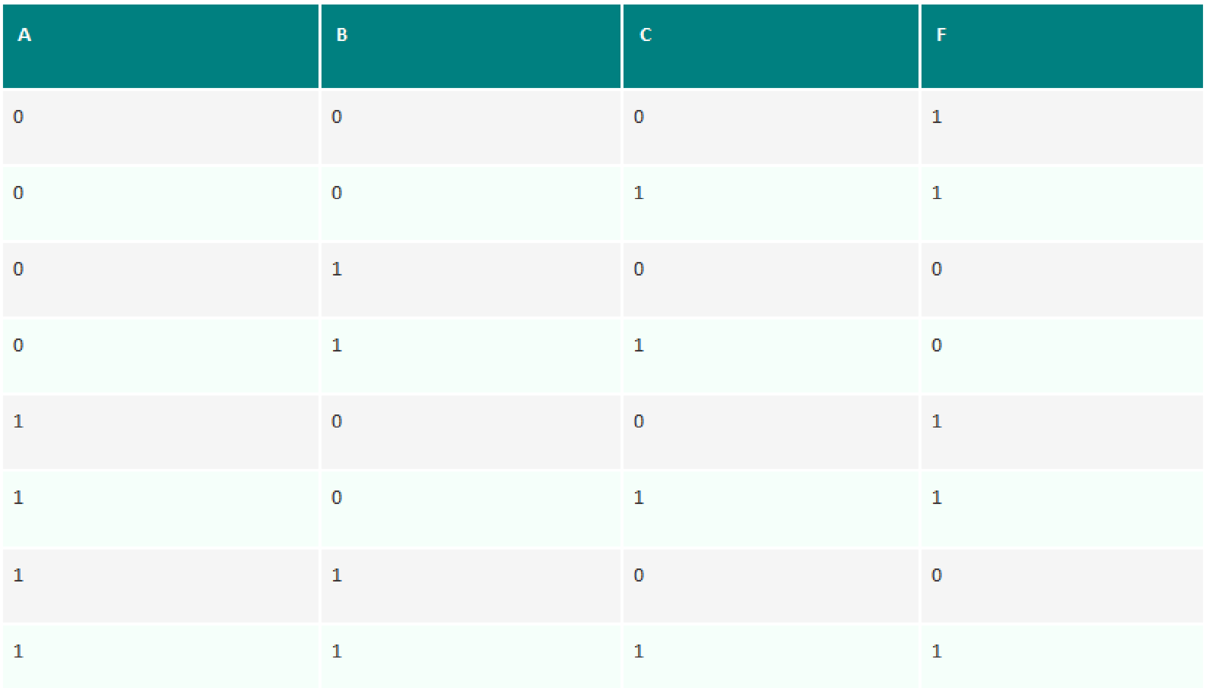
We have to design the truth table using the above expression in POS form.

We have following three sum terms:

1. **A+B'+C'**
2. **A'+B'+C**
3. **A+B'+C**

The first sum term **(A+B'+C')** is **011** in binary. As in POS form boolean expression, the uncomplemented form holds 0 and the complemented form holds 1. The second sum term **(A'+B'+C)** is **110**. And the third sum term **(A+B'+C)** holds **010**.

We have designed the truth table as given below. Here we have to put maxterm as 0 only in three rows. That is, where binary representation is **011**, **110**, and **010** in this case.

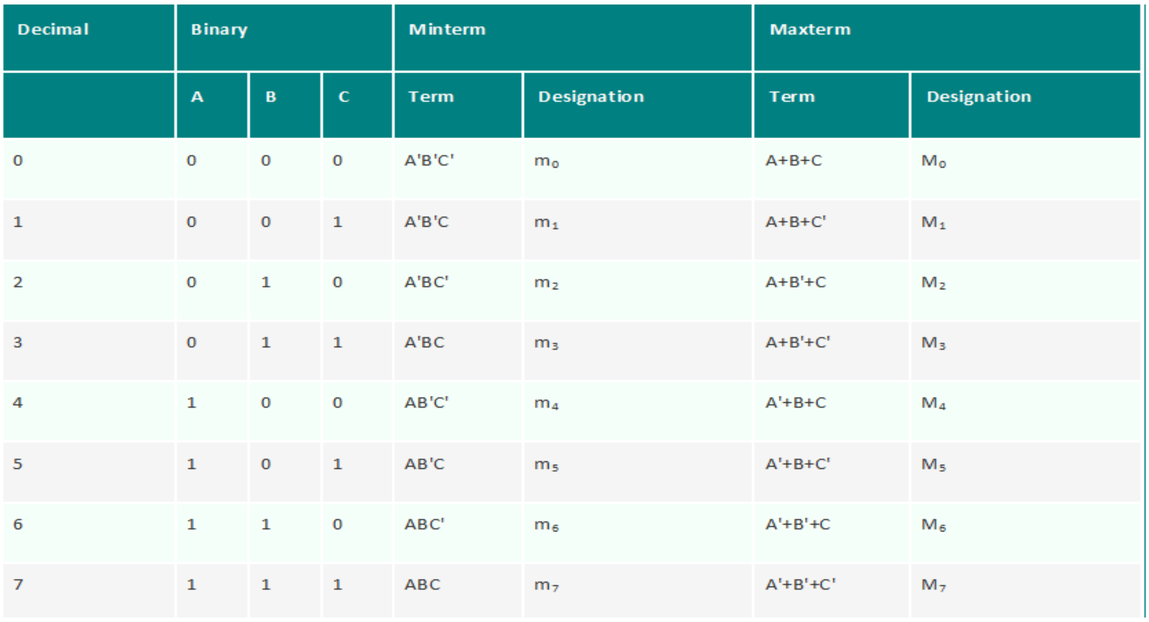


A	B	C	F
0	0	0	1
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	1
1	1	0	0
1	1	1	1

As we can see from the above truth table, the function contains three 0. The first 0 is in that row where the binary value is **010** and the second 0 is in that row where the binary value is **011** and finally the third 0 is in that row where the binary value is **110**. And at all the other places, we have 1.

Truth Table for Minterm and Maxterm of Three Variable

The truth table given below shows the minterm and maxterm both along with its designation for 3-variable.



Decimal	Binary			Minterm		Maxterm	
	A	B	C	Term	Designation	Term	Designation
0	0	0	0	A'B'C'	m_0	A+B+C	M_0
1	0	0	1	A'B'C	m_1	A+B+C'	M_1
2	0	1	0	A'BC'	m_2	A+B'+C	M_2
3	0	1	1	A'BC	m_3	A+B'+C'	M_3
4	1	0	0	AB'C'	m_4	A'+B+C	M_4
5	1	0	1	AB'C	m_5	A'+B+C'	M_5
6	1	1	0	ABC'	m_6	A'+B'+C	M_6
7	1	1	1	ABC	m_7	A'+B'+C'	M_7

Basic Theorems:

1)AND laws:

These laws related to the AND operation therefore they are called as “AND ” laws .The laws are as follow:

(1). A.0 = 0

That means if one input of an AND operation is permanently kept at the LOW(0) level, then the output is zero irrespective of the other variable.

(2). A.1 = A

That means if one input of an AND operation is HIGH(1) permanently , then the output is always equal to the other variable.

-So if A= 0 ,then Y=0.1 =0 i.e. Y=A and if A=1, then Y= 1.1=1 so Y=A.

(3). A.A = A

That means if both the inputs in an AND operation have the same value either ‘0’ or ‘1’ then the output will also have the same value as that of the input.

(4). A.A’ = 0

This law states that the result of an ‘AND ’ operation on a variable (A) and its complement(A’) is always LOW(0)

If A=0 then A’ =1 and Y=0.1 =0 whereas if A=1 then A’ =0 and Y=1.0=0

2)OR Laws:

These laws use the OR operation. Therefore they are called as OR laws . The OR laws are as follows:

(1). A+0 =A

That means if one variable of an ‘OR’ operation is LOW(0) permanently ,then the output is always equal to the other variable.

B =0 permanently therefore for A= 0 ,Y=0+0=0 i.e. Y=A and for

A=1 , Y=1+0 =1 i.e Y=A.

(2) . A+1 =1

That means if one variable of an ‘OR’ operation is HIGH(1) permanently , then the output is HIGH(1) permanently irrespective of the value of the other variable.

Here B=1 permanently ,so if A=0 then Y=0+1=1 and if A=1 then Y=1+1 =1.

Thus output remains HIGH(1) always , irrespective of the value of A.

(3). A+A =A

This law states that if both the variables of an OR operation have the same value either ‘0’ or ‘1’ then the output also will be equal to the input i.e. 0 or 1 respectively.

For A=0 , Y=0+0=0 i.e. Y=A and for A = 1 , Y= 1+1 =1 i.e. Y=A

(4). A+A’= 1

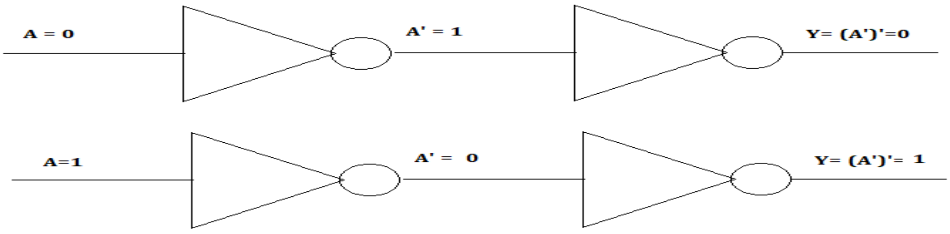
This laws states that the results of an ‘OR’ operation on a variable and its complement it always 1 (HIGH).

If A=0 then A’ =1 and Y=0+1 =1 whereas if A=1 then A’ =0 and Y=1+0=1

3). Inversion Law:

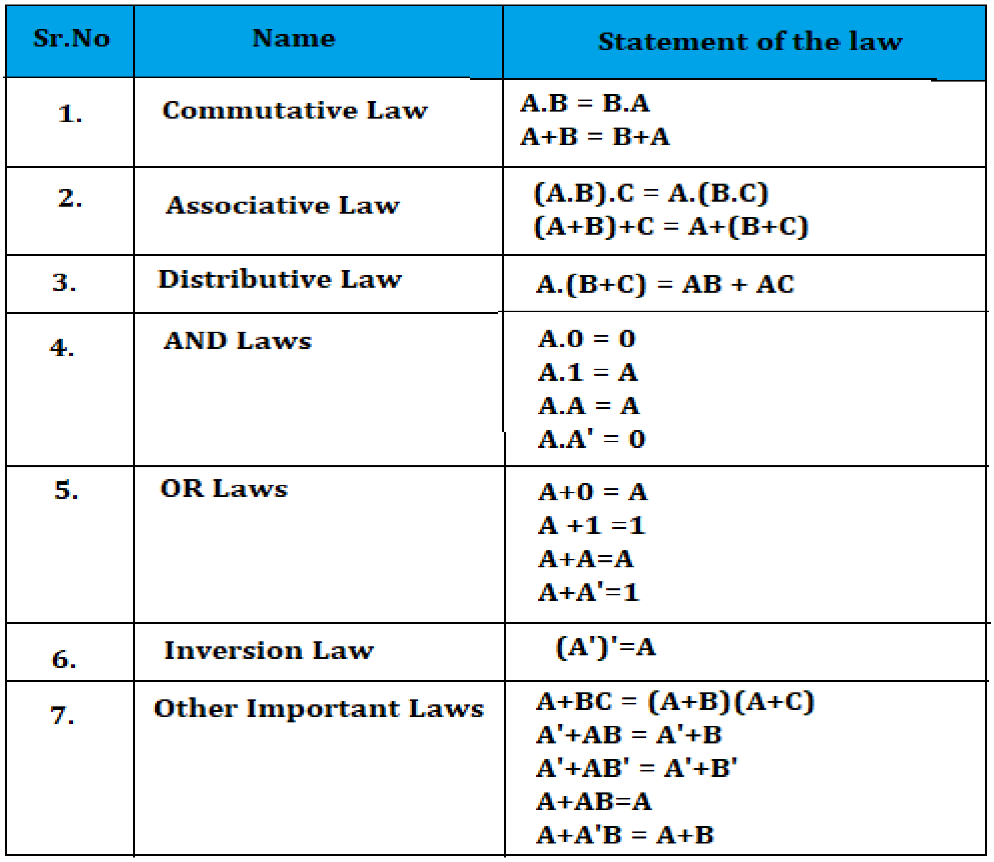
This law uses the ‘NOT’ operation ..the inversion law states that if a variable is subjected to a double inversion then it will result in the original variable itself i.e.

(A’)’=A



Inversion Law: (A')'= A

Inversion law ,which shows that if A=0 then A’=1 and (A’)’=0 therefore Y= A whereas if A =1 then A’=0 and (A’)’=1 therefore Y=A.



Sr.No	Name	Statement of the law
1.	Commutative Law	A.B = B.A A+B = B+A
2.	Associative Law	(A.B).C = A.(B.C) (A+B)+C = A+(B+C)
3.	Distributive Law	A.(B+C) = AB + AC

4.	AND Laws	A.0 = 0
		A.1 = A
		A.A = A
		A.A' = 0

5.	OR Laws	A+0 = A A +1 =1 A+A=A A+A'=1
6.	Inversion Law	(A')'=A
7.	Other Important Laws	A+BC = (A+B)(A+C) A'+AB = A'+B A'+AB' = A'+B' A+AB=A A+A'B = A+B

Karnaugh Map(K-Map) method:-

The **K-map** is a systematic way of simplifying Boolean expressions. With the help of the K-map method, we can find the simplest POS and SOP expression, which is known as the minimum expression. The K-map provides a cookbook for simplification.

Just like the truth table, a K-map contains all the possible values of input variables and their corresponding output values. However, in K-map, the values are stored in cells of the array. In each cell, a binary value of each input variable is stored.

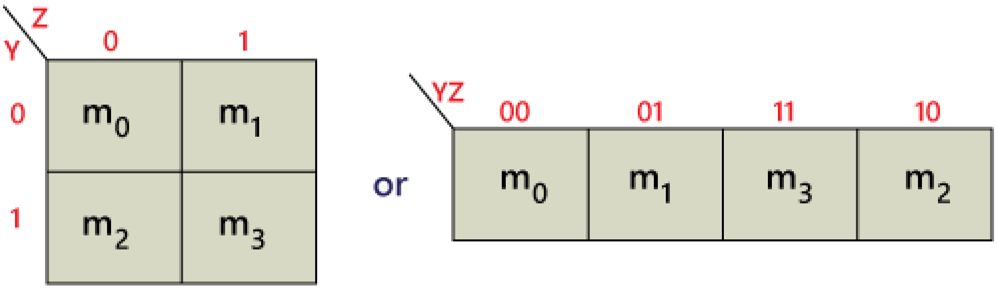
The K-map method is used for expressions containing 2, 3, 4, and 5 variables. For a higher number of variables, there is another method used for simplification called the Quine-McClusky method. In K-map, the number of cells is similar to the total number of variable input combinations. For example, if the number of variables is three, the number of cells is 2^3=8, and if the number of variables is four, the number of cells is 2^4. The K-map takes the SOP and POS forms. The K-map grid is filled using 0's and 1's. The K-map is solved by making groups. There are the following steps used to solve the expressions using K-map:

1. First, we find the K-map as per the number of variables.
2. Find the maxterm and minterm in the given expression.
3. Fill cells of K-map for SOP with 1 respective to the minterms.

4. Fill cells of the block for POS with 0 respective to the maxterm.
5. Next, we create rectangular groups that contain total terms in the power of two like 2, 4, 8, ... and try to cover as many elements as we can in one group.
6. With the help of these groups, we find the product terms and sum them up for the SOP form.

2 Variable K-map

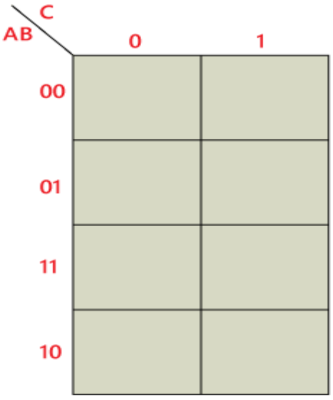
There is a total of 4 variables in a 2-variable K-map. There are two variables in the 2-variable K-map. The following figure shows the structure of the 2-variable K-map:

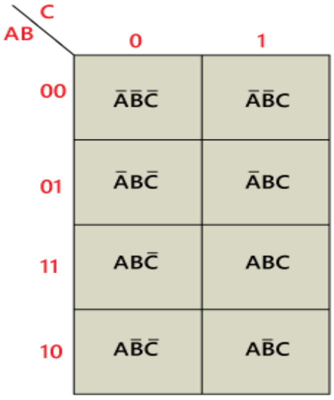


- In the above figure, there is only one possibility of grouping four adjacent minterms.
- The possible combinations of grouping 2 adjacent minterms are {(m_0, m_1), (m_2, m_3), (m_0, m_2) and (m_1, m_3)}.

3-variable K-map

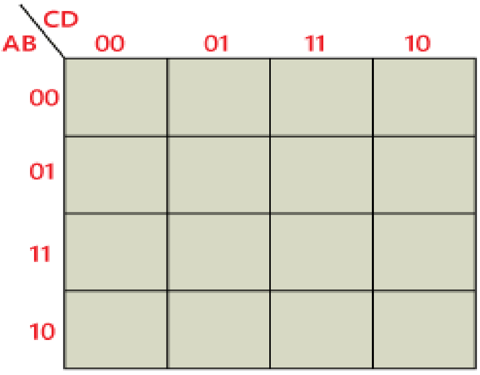
The 3-variable K-map is represented as an array of eight cells. In this case, we used A, B, and C for the variable. We can use any letter for the names of the variables. The binary values of variables A and B are along the left side, and the values of C are across the top. The value of the given cell is the binary values of A and B at left side in the same row combined with the value of C at the top in the same column. For example, the cell in the upper left corner has a binary value of 000, and the cell in the lower right corner has a binary value of 101.



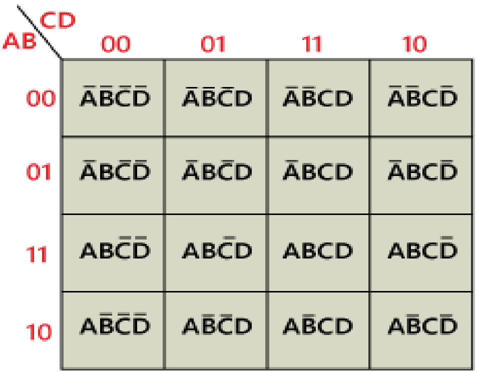


The 4-Variable Karnaugh Map

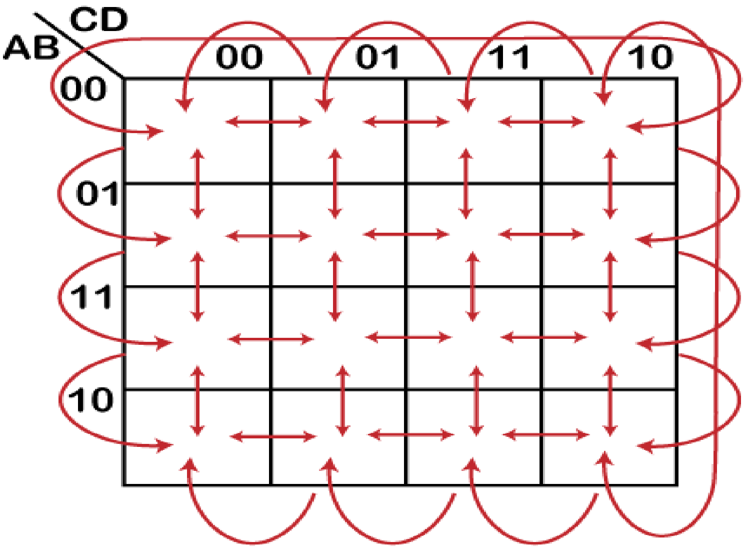
The 4-variable K-map is represented as an array of 16 cells. Binary values of A and B are along the left side, and the values of C and D are across the top. The value of the given cell is the binary values of A and B at left side in the same row combined with the binary values of C and D at the top in the same column. For example, the cell in the upper right corner has a binary value of 0010, and the cell in the lower right corner has a binary value of 1010.



AB \ CD	00	01	11	10
00				
01				
11				
10				



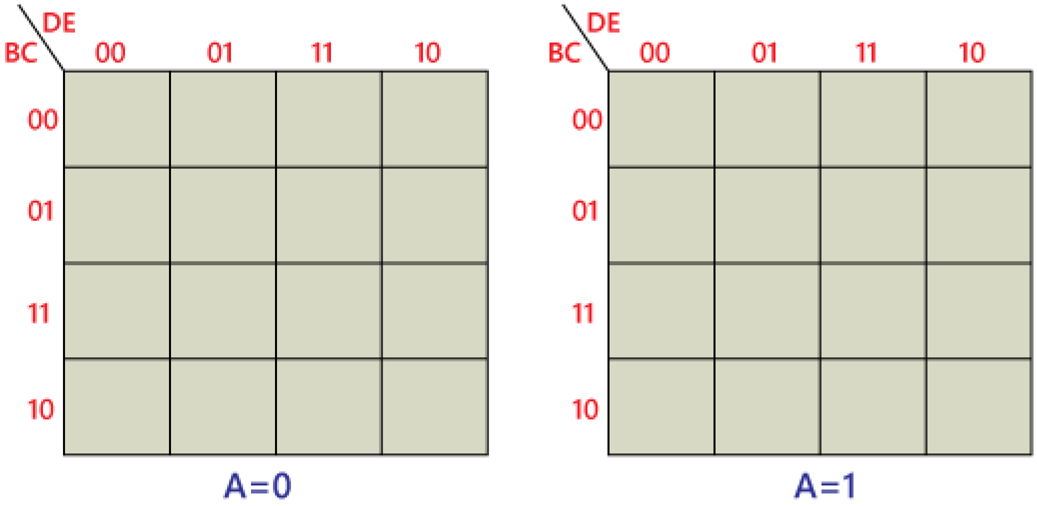
AB \ CD	00	01	11	10
00	$\bar{A}\bar{B}\bar{C}\bar{D}$	$\bar{A}\bar{B}\bar{C}D$	$\bar{A}\bar{B}CD$	$\bar{A}\bar{B}C\bar{D}$
01	$\bar{A}B\bar{C}\bar{D}$	$\bar{A}B\bar{C}D$	$\bar{A}BCD$	$\bar{A}BC\bar{D}$
11	$AB\bar{C}\bar{D}$	$AB\bar{C}D$	$ABCD$	$ABC\bar{D}$
10	$A\bar{B}\bar{C}\bar{D}$	$A\bar{B}\bar{C}D$	$A\bar{B}CD$	$A\bar{B}C\bar{D}$



5-variable K-map

With the help of the 32- cell K-map, the boolean expression with 5 variables can be simplified. For constructing a 5-variable K-map, we use two 4-variable K-maps. The cell adjacencies within each of the 4- variable maps for the 5-variable map are similar to the 4- variable map.

A K-map for five variables (PQRST) can be constructed using two 4-variable maps. Each map contains 16 cells with all combinations of variables Q, R, S, and T. One map is for P = 0, and the other is for P = 1).



Simplification of boolean expressions using Karnaugh Map

As we know that K-map takes both SOP and POS forms. So, there are two possible solutions for K-map, i.e., minterm and maxterm solution. Let's start and learn about how we can find the minterm and maxterm solution of K-map.

Minterm Solution of K Map

There are the following steps to find the minterm solution or K-map:

Step 1:

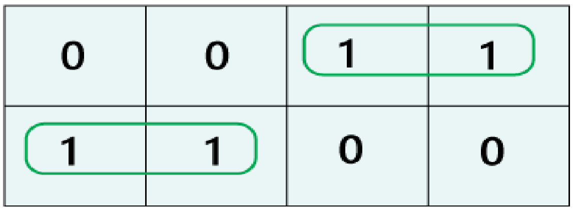
Firstly, we define the given expression in its canonical form.

Step 2:

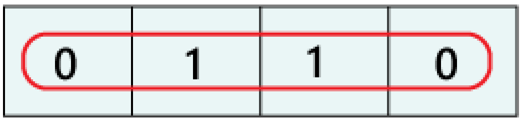
Next, we create the K-map by entering 1 to each product-term into the K-map cell and fill the remaining cells with zeros.

Step 3:

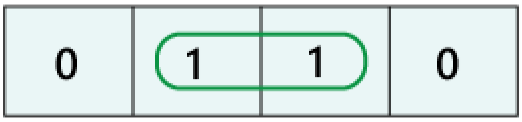
Next, we form the groups by considering each one in the K-map.



Notice that each group should have the largest number of 'ones'. A group cannot contain an empty cell or cell that contains 0.



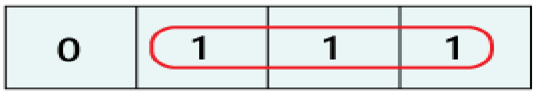
Incorrect



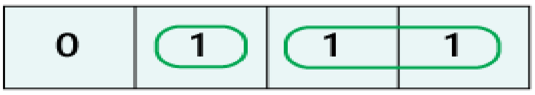
Correct

In a group, there is a total of 2^n number of ones. Here, n=0, 1, 2, ...n.

Example: $2^0=1$, $2^1=2$, $2^2=4$, $2^3=8$, or $2^4=16$.

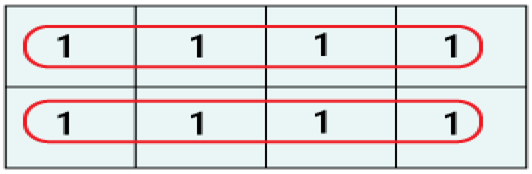


Incorrect

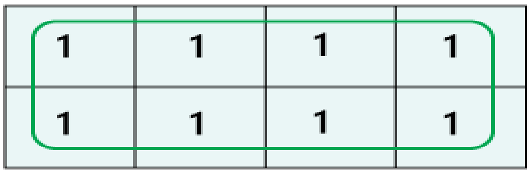


Correct

We group the number of ones in the decreasing order. First, we have to try to make the group of eight, then for four, after that two and lastly for 1.

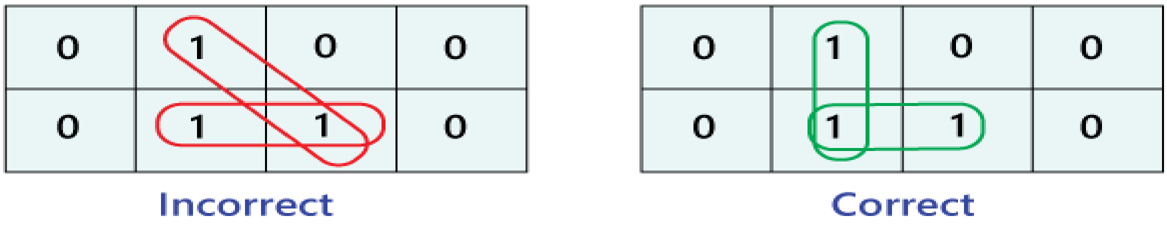


Incorrect



Correct

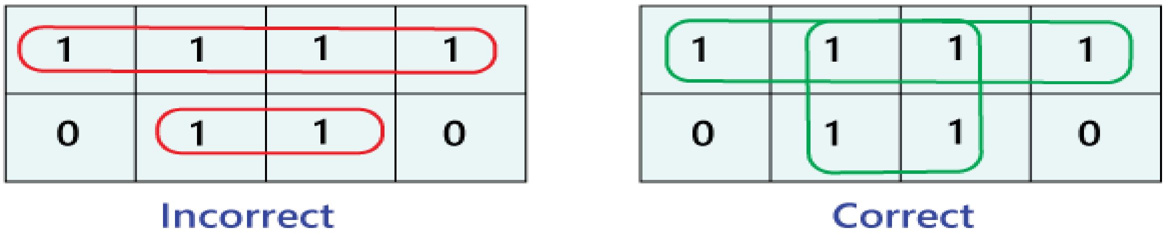
In horizontally or vertically manner, the groups of ones are formed in shape of rectangle and square. We cannot perform the diagonal grouping in K-map.



0	1	0	0
0	1	1	0

0	1	0	0
0	1	1	0

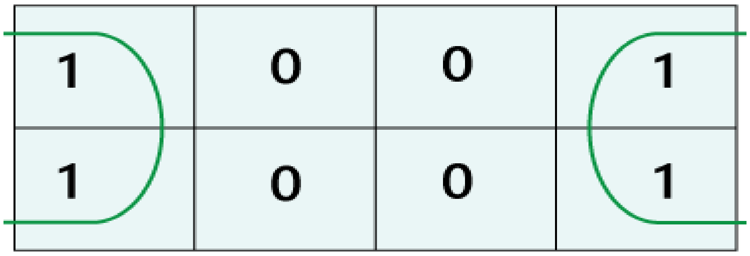
The elements in one group can also be used in different groups only when the size of the group is increased.



1	1	1	1
0	1	1	0

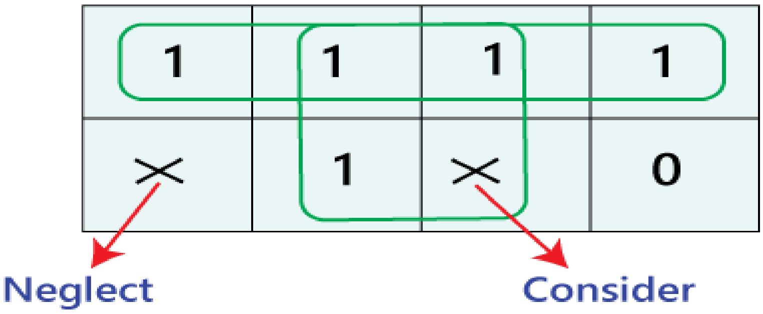
1	1	1	1
0	1	1	0

The elements located at the edges of the table are considered to be adjacent. So, we can group these elements.



1	0	0	1
1	0	0	1

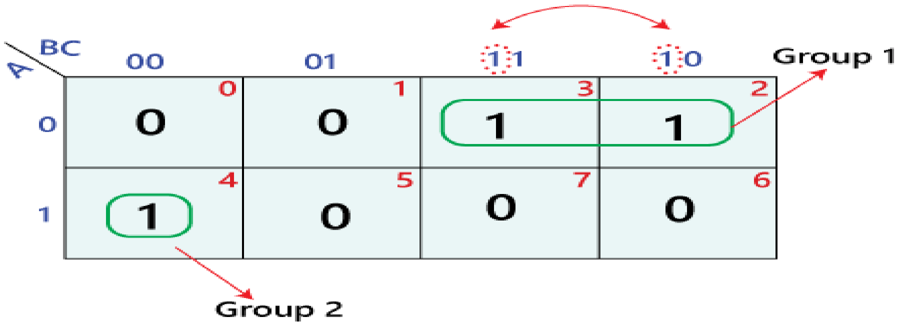
We can consider the 'don't care condition' only when they aid in increasing the group-size. Otherwise, 'don't care' elements are discarded.



1	1	1	1
×	1	×	0

Step 4:

In the next step, we find the boolean expression for each group. By looking at the common variables in cell-labeling, we define the groups in terms of input variables. In the below example, there is a total of two groups, i.e., group 1 and group 2, with two and one number of 'ones'. In the first group, the ones are present in the row for which the value of A is 0. Thus, they contain the complement of variable A. Remaining two 'ones' are present in adjacent columns. In these columns, only B term in common is the product term corresponding to the group as A'B. Just like group 1, in group 2, the one's are present in a row for which the value of A is 1. So, the corresponding variables of this column are B'C'. The overall product term of this group is AB'C'.



	00	01	11	10
0	0	0	1	1
1	1	0	0	0

Step 5:

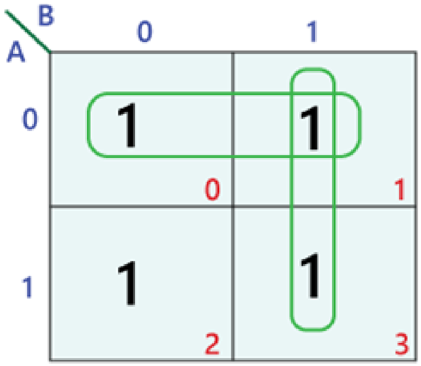
Lastly, we find the boolean expression for the Output. To find the simplified boolean expression in the SOP form, we combine the product-terms of all individual groups. So the simplified

the product-terms of all individual groups. So the simplified expression of the above k-map is as follows:

```
A'+AB'C'
```

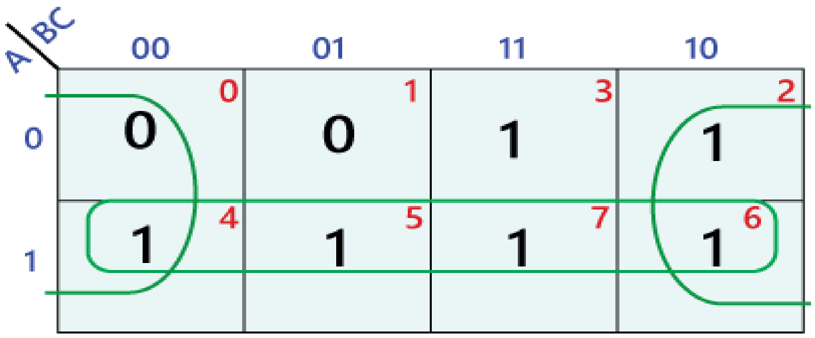
Let's take some examples of 2-variable, 3-variable, 4-variable, and 5-variable K-map examples.

Example 1: Y=A'B' + A'B+AB



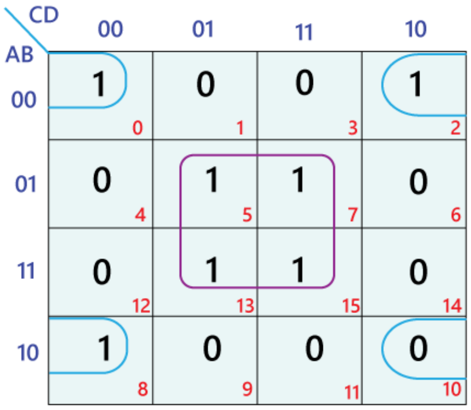
Simplified expression: Y=A'+B

Example 2: Y=A'B'C'+A' BC'+AB' C'+AB' C+ABC'+ABC



Simplified expression: Y=A+C'

Example 3: Y=A'B'C' D'+A' B' CD'+A' BCD'+A' BCD+AB' C' D'+ABCD'+ABCD



Simplified expression: Y=BD+B'D'

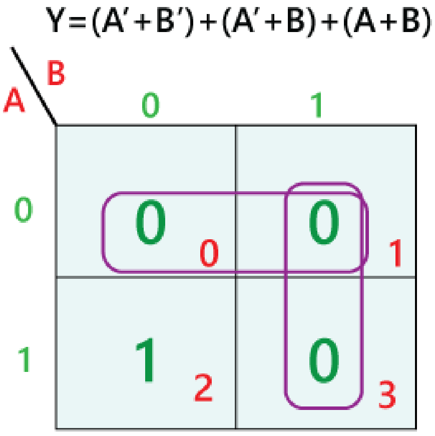
Maxterm Solution of K-Map

To find the simplified maxterm solution using K-map is the same as to find for the minterm solution. There are some minor changes in the maxterm solution, which are as follows:

1. We will populate the K-map by entering the value of 0 to each sum-term into the K-map cell and fill the remaining cells with one's.
2. We will make the groups of 'zeros' not for 'ones'.
3. Now, we will define the boolean expressions for each group as sum-terms.
4. At last, to find the simplified boolean expression in the POS form, we will combine the sum-terms of all individual groups.

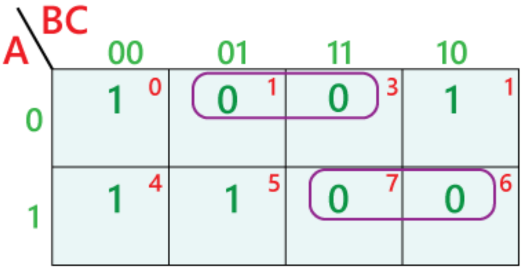
Let's take some example of 2-variable, 3-variable, 4-variable and 5-variable K-map examples

Example 1: Y=(A'+B')+(A'+B)+(A+B)



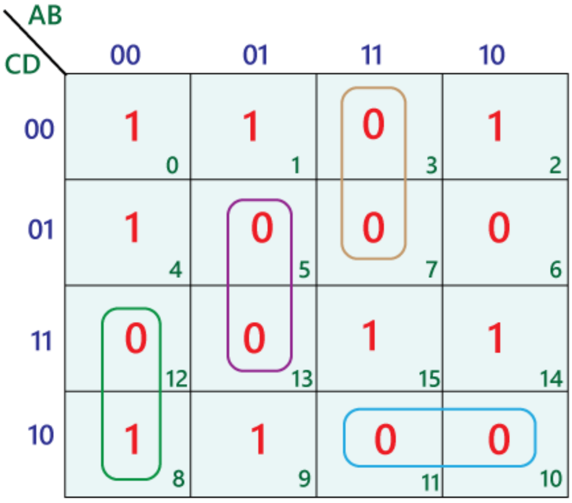
Simplified expression: A'B

Example 2: Y=(A + B + C') + (A + B' + C') + (A' + B' + C) + (A' + B' + C')



Simplified expression: Y=(A + C') .(A' + B')

Example 3: F(A,B,C,D)=π(3,5,7,8,10,11,12,13)



AB CD	00	01	11	10
00	1 0	1 1	0 3	1 2
01	1 4	0 5	0 7	0 6
11	0 12	0 13	1 15	1 14
10	1 8	1 9	0 11	0 10

Simplified expression: Y=(A + C') .(A' + B')

Unit:3

Combinational Logic Circuits: Introduction, Multi-input, multi-output Combinational circuits, Code converters design and implementations

Arithmetic Circuits: Introduction, Adder, BCD Adder, Excess – 3 Adder, Binary Subtractors, BCD Subtractor, Multiplier, Comparator

Introduction:

Types of digital circuit:

-The digital system in general are classified into two categories namely:

1) Combinational logic circuits.

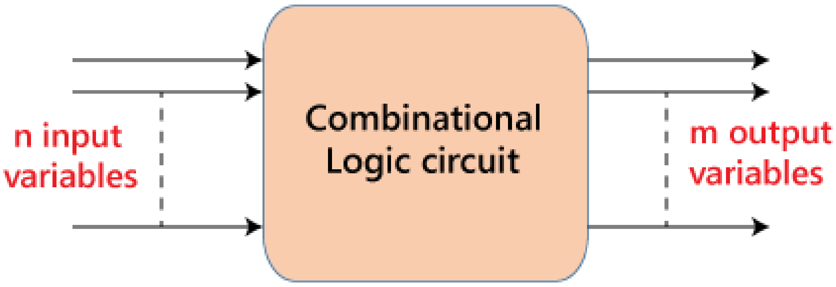
2) Sequential logic circuits.

Combinational Logic circuits:

- The combinational logic circuits are the circuits that contain different types of logic gates. Simply, a circuit in which different types of logic gates are combined is known as a **combinational logic circuit**.
- The output of the combinational circuit is determined from the present combination of inputs, regardless of the previous input. The input variables, logic gates, and output variables are the basic components of the combinational logic circuit.
- There are different types of combinational logic circuits, such as Adder, Subtractor, Decoder, Encoder, Multiplexer, and De-multiplexer.

There are the following characteristics of the combinational logic circuit:

- At any instant of time, the output of the combinational circuits depends only on the present input terminals.
- The combinational circuit doesn't have any backup or previous memory. The present state of the circuit is not affected by the previous state of the input.
- The n number of inputs and m number of outputs are possible in combinational logic circuits.



Block diagram of a combinational circuit

The 'n' input variable comes from the external source while the 'm' output variable goes to the external destination. In many applications, the source or destinations are storage registers.

Half Adder

The half adder is a basic building block having two inputs and two outputs. The adder is used to perform OR operation of two single bit binary numbers. The **carry** and **sum** are two output states of the half adder.

Full Adder

The half adder is used to add only two numbers. To overcome this problem, the full adder was developed. The full adder is used to add three 1-bit binary numbers A, B, and carry C. The full adder has three input states and two output states i.e., sum and carry.

Half Subtractors

The half subtractor is also a building block of subtracting two binary numbers. It has two inputs and two outputs. This circuit is used to subtract two single bit binary numbers A and B. The **'diff'** and **'borrow'** are the two output state of the half adder.

Full Subtractors

The Half Subtractor is used to subtract only two numbers. To overcome this problem, full subtractor was designed. The full subtractor is used to subtract three 1-bit numbers A, B, and C, which are **minuend**, **subtrahend**, and **borrow,** respectively. The full subtractor has three input states and two output states i.e., diff and borrow.

Multiplexers

The multiplexer is a combinational circuit that has n-data inputs and a single output. It is also known as the **data selector** which selects one input from the inputs and routes it to the output. With the help of the selected inputs, one input line from the n-input lines is selected. The enable input is denoted by E, which is used in cascade.

De-multiplexers

A De-multiplexer performs the reverse operation of a multiplexer. The de-multiplexer has only one input, which is distributed over several outputs. One output line is selected at a time by selecting lines. The input is transmitted to the selected output line.

Decoder

A decoder is a combinational circuit having n inputs and to a maximum of m = 2n outputs. The decoder is the same as the de-multiplexer. The only difference between de-multiplexer and decoder is that in the decoder, there is no data input. The decoder performs an operation that is completely opposite of an encoder.

Encoder:-

The encoder is used to perform the reverse operation of the decoder. An encoder having n number of inputs and m number of outputs is used to produce m-bit binary code which is related to the digital input number. The encoder takes the digital word and converts it into another digital word.

Code Conversion:

-In this section we are going to convert one type of code into another type.

- Some of the code converters given following:

1)Binary to BCD converter.

2)BCD to Binary Converter.

3) BCD to Excess 3

4) Excess 3 to BCD.

5)Binary to Gray code converter.

6)Gray code to Binary converter.

7)BCD to Gray code converter.

1)Binary to BCD converter:

We will convert a 4-bit binary words into 5-bit BCD words.

BCD is binary coded decimal number, where each digit of a decimal number is respected by its equivalent binary number. That means, LSB of a decimal number is represented by its equivalent binary number and similarly other higher significant bits of decimal number are also represented by their equivalent binary numbers.

For example, BCD Code of 14 is-

The procedure to be given below:

Step to be follow:

Step-1: Write the truth table showing the relation between binary as input and BCD as output.

Step-2: For each BCD output (B_0,B_1,...) write a K-map.

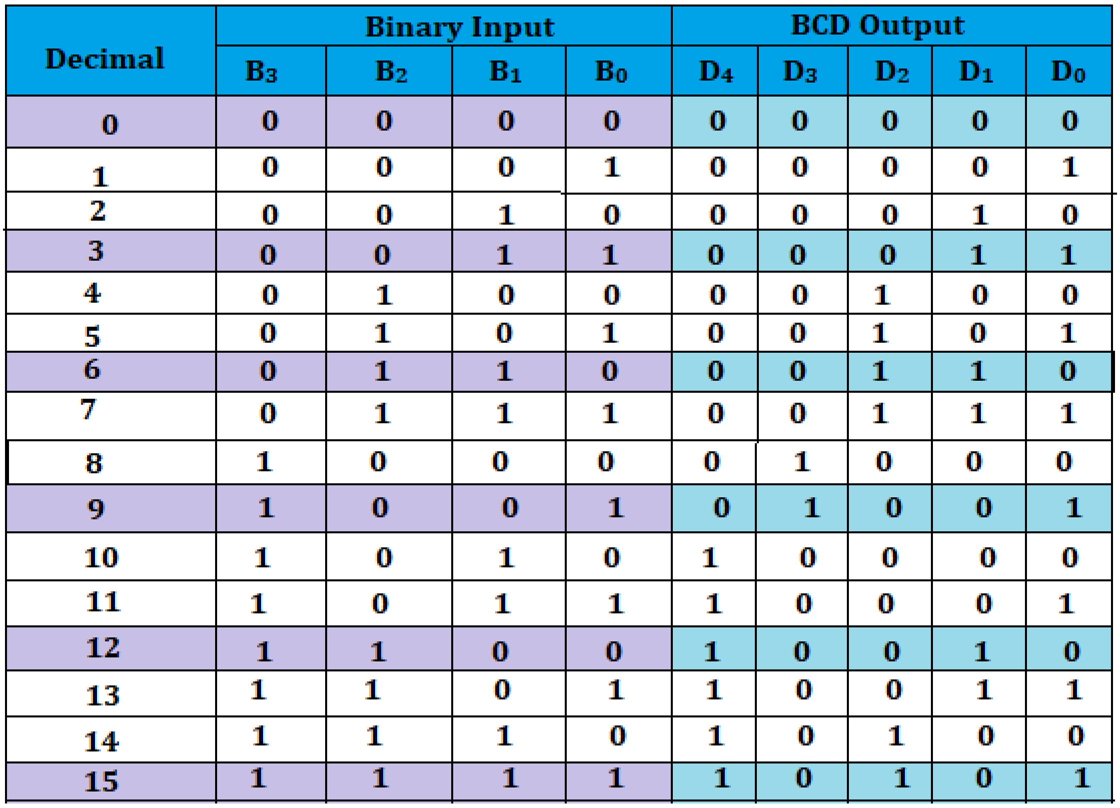
Step-3: From the K-map obtain a simplified expression for each BCD output in terms of binary inputs.

Step-4: Realize the code converter using the basic gates.

Step-1: Truth table Relating Binary and BCD:

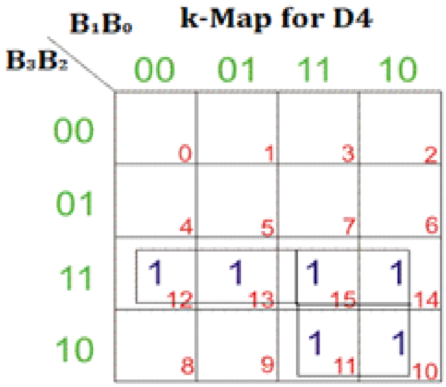
-Below table shows the truth table relating binary to BCD . B_3 B_2 B_1B_0 is the four bit binary word with B_3 acting as MSB.

-$D_4D_3D_2D_1D_0$ is the five bit BCD output.



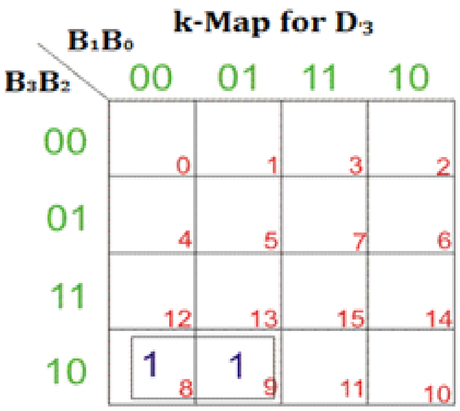
Decimal	Binary Input				BCD Output				
	B_3	B_2	B_1	B_0	D_4	D_3	D_2	D_1	D_0
0	0	0	0	0	0	0	0	0	0
1	0	0	0	1	0	0	0	0	1
2	0	0	1	0	0	0	0	1	0
3	0	0	1	1	0	0	0	1	1
4	0	1	0	0	0	0	1	0	0
5	0	1	0	1	0	0	1	0	1
6	0	1	1	0	0	0	1	1	0
7	0	1	1	1	0	0	1	1	1
8	1	0	0	0	0	1	0	0	0
9	1	0	0	1	0	1	0	0	1
10	1	0	1	0	1	0	0	0	0
11	1	0	1	1	1	0	0	0	1
12	1	1	0	0	1	0	0	1	0
13	1	1	0	1	1	0	0	1	1
14	1	1	1	0	1	0	1	0	0
15	1	1	1	1	1	0	1	0	1

Step-2: For each BCD output write a K-Map.


k-Map for D4

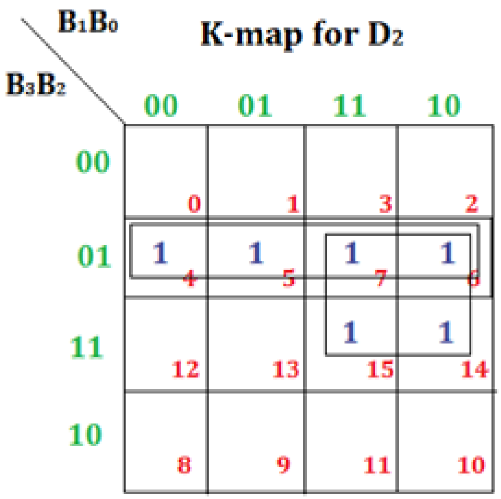
B₃B₂ \ B₁B₀	00	01	11	10
00	0	1	3	2
01	4	5	7	6
11	1 12	1 13	1 15	1 14
10	8	9	1 11	1 10

$D_4 = B_3B_2 + B_3B_1$


k-Map for D₃

B₃B₂ \ B₁B₀	00	01	11	10
00	0	1	3	2
01	4	5	7	6
11	12	13	15	14
10	1 8	1 9	11	10

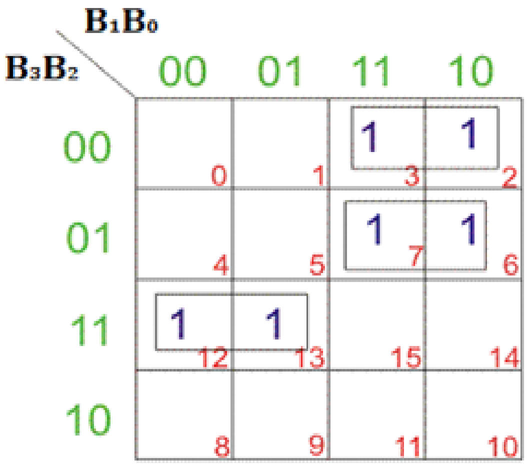
$D_3 = B_3B_2'B_1'$


K-map for D₂

B₃B₂ \ B₁B₀	00	01	11	10
00	0	1	3	2
01	1 4	1 5	1 7	1 6
11	12	13	1 15	1 14
10	8	9	11	10

$D_2 = B_3'B_2+B_2B_1$

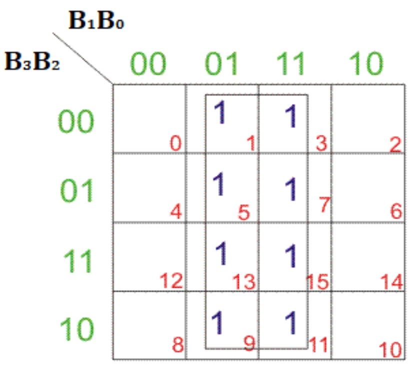
K - map for D₁



B₃B₂ \ B₁B₀	00	01	11	10
00	0	1	1 3	1 2
01	4	5	1 7	1 6
11	1 12	1 13	15	14
10	8	9	11	10

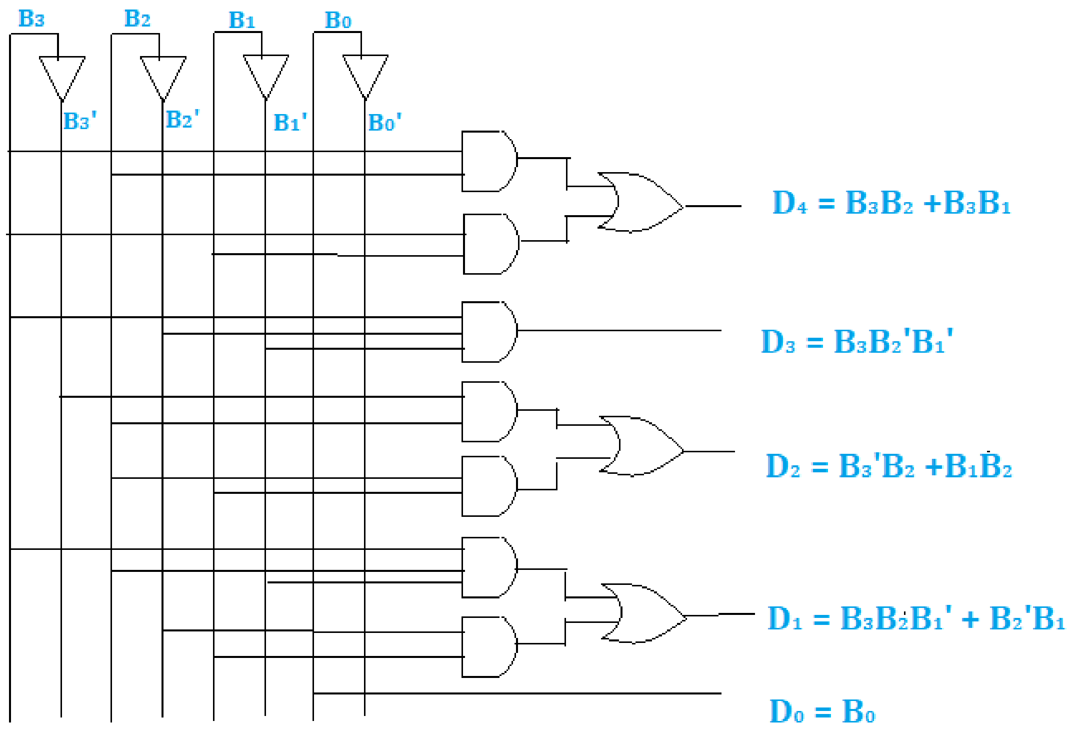
$D_1 = B_3B_2B_1' + B_2'B_1$

K - map for D₀



B₃B₂ \ B₁B₀	00	01	11	10
00	0	1 1	1 3	2
01	4	1 5	1 7	6
11	12	1 13	1 15	14
10	8	1 9	1 11	10

$D_0 = B_0$



BCD to Excess-3 conversion

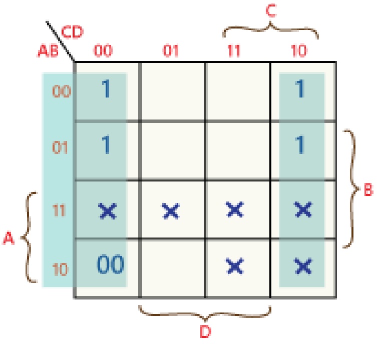
To understand the process of converting BCD to Excess-3, it is required to have knowledge of Number System and Number Base Conversion.

The Excess-3 binary code is an example of a self-complementary BCD code. A self-complementary binary code is a code which is always complimented in itself. By replacing the bit 0 to 1 and 1 to 0 of a number, we find the 1's complement of the number. The sum of the 1'st complement and the binary number of a decimal is equal to the binary number of decimal 9.

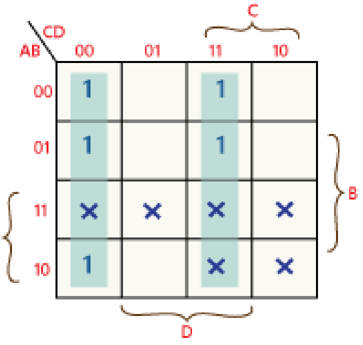
The process of converting BCD to Excess-3 is quite simple from other conversions. The Excess-3 code can be calculated by adding 3, i.e., 0011 to each four-digit BCD code. Below is the truth table for the conversion of BCD to Excess-3 code. In the below table, the variables A, B, C, and D represent the bits of the binary numbers. The variable 'D' represents the LSB, and the variable 'A' represents the MSB. In the same way, the variables w, x, y, and z represent the bits of the Excess-3 code. The variable 'z' represents the LSB, and the variable 'w' represents the MSB. The 'don't care conditions' is expressed by the variable 'X'.

Decimal number	BCD Code				Excess-3 Code			
	A	B	C	D	W	x	y	z
0	0	0	0	0	0	0	1	1
1	0	0	0	1	0	1	0	0
2	0	0	1	0	0	1	0	1
3	0	0	1	1	0	1	1	0
4	0	1	0	0	0	1	1	1
5	0	1	0	1	1	0	0	0
6	0	1	1	0	1	0	0	1
7	0	1	1	1	1	0	1	0
8	1	0	0	0	1	0	1	1
9	1	0	0	1	1	1	0	0

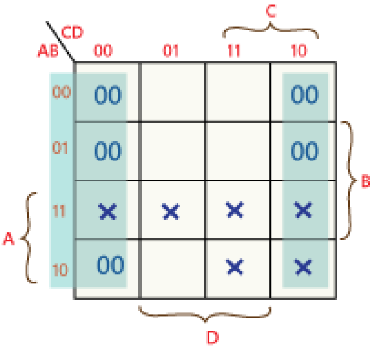
Now, we will use the K-map method to design the logical circuit for the conversion of BCD to Excess-3 code as:



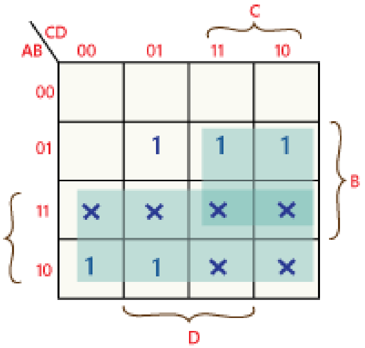
z=D′



y=CD+C′D′



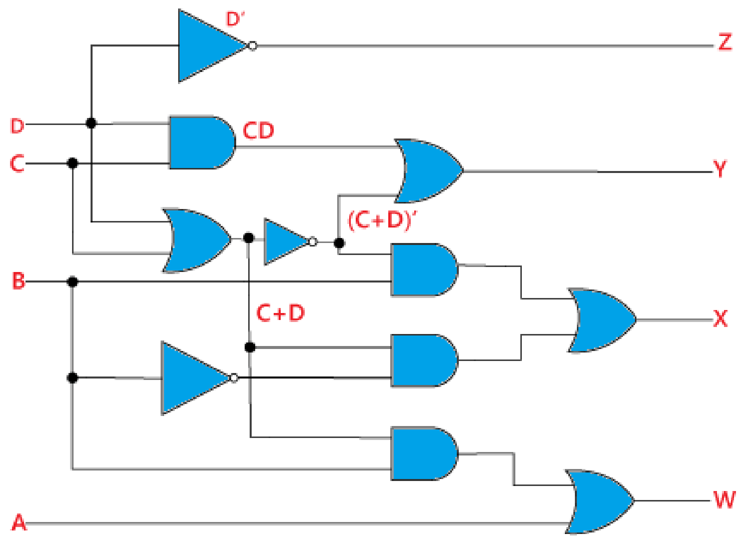
x=B′C+B′D+BC′D′



w=A+BC+BD

So,

w=A+BC+BD
x=B' C+B' D+BC' D'
y=CD+C'D'
z=D'

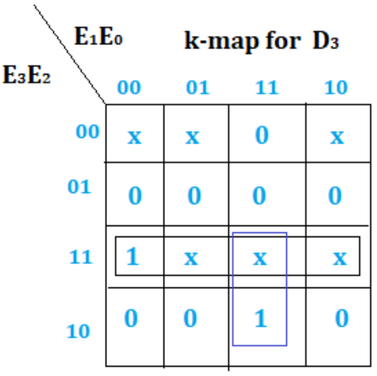


Excess-3 to BCD conversion

The process of converting Excess-3 to BCD is opposite to the process of converting BCD to Excess-3. The BCD code can be calculated by subtracting 3, i.e., 0011 from each four-digit Excess-3 code. Below is the truth table for the conversion of Excess-3 code to BCD. In the below table, the variables w, x, y, and z represent the bits of the Excess-3 code. The variable 'z' represents the LSB, and the variable 'w' represents the MSB. In the same way, the variables A, B, C, and D represent the bits of the binary numbers. The variable 'D' represents the LSB, and the variable 'A' represents the MSB. The 'don't care conditions' is defined by the variable 'X'.

Excess-3 inputs				BCD outputs			
E3	E2	E2	E1	D3	D2	D1	D0
0	0	1	1	0	0	0	0
0	1	0	0	0	0	0	1
0	1	0	1	0	0	1	0
0	1	1	0	0	0	1	1
0	1	1	1	0	1	0	0
1	0	0	0	0	1	0	1
1	0	0	1	0	1	1	0
1	0	1	0	0	1	1	1
1	0	1	1	1	0	0	0
1	1	0	0	1	0	0	1

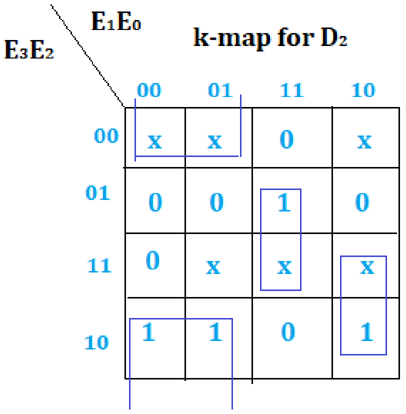
Now, we will use the K-map method to design the logical circuit for the conversion of Excess-3 code to BCD as:



k-map for D_3

E_3E_2 \ E_1E_0	00	01	11	10
00	x	x	0	x
01	0	0	0	0
11	1	x	x	x
10	0	0	1	0

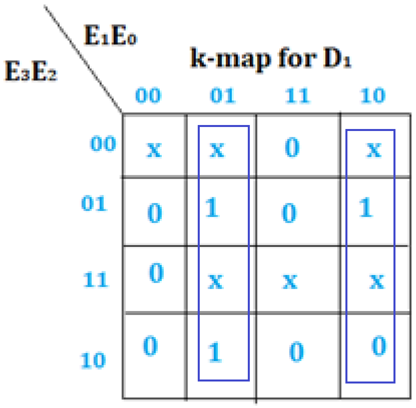
$D_3 = E_3E_2 + E_3E_1E_0 = E_3(E_2+E_1E_0)$



k-map for D_2

E_3E_2 \ E_1E_0	00	01	11	10
00	x	x	0	x
01	0	0	1	0
11	0	x	x	x
10	1	1	0	1

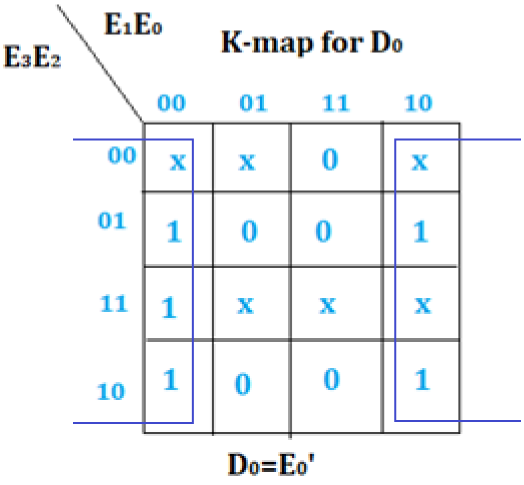
$D_2=E_2'E_1' +E_2E_1E_0+E_3E_1E_0'$



k-map for D_1

E_3E_2 \ E_1E_0	00	01	11	10
00	x	x	0	x
01	0	1	0	1
11	0	x	x	x
10	0	1	0	0

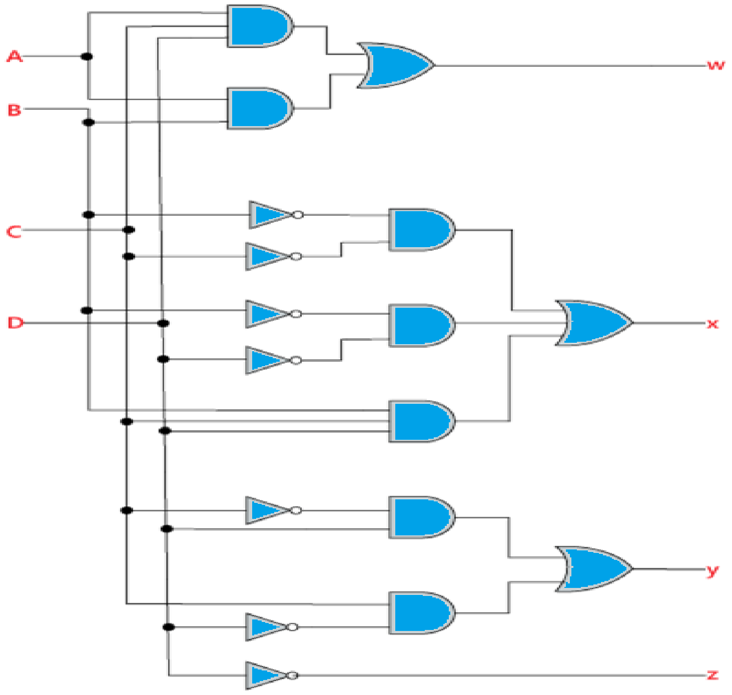
$D_1 = E_1'E_0+E_1E_0'$



K-map for D_0

E_3E_2 \ E_1E_0	00	01	11	10
00	x	x	0	x
01	1	0	0	1
11	1	x	x	x
10	1	0	0	1

$D_0=E_0'$



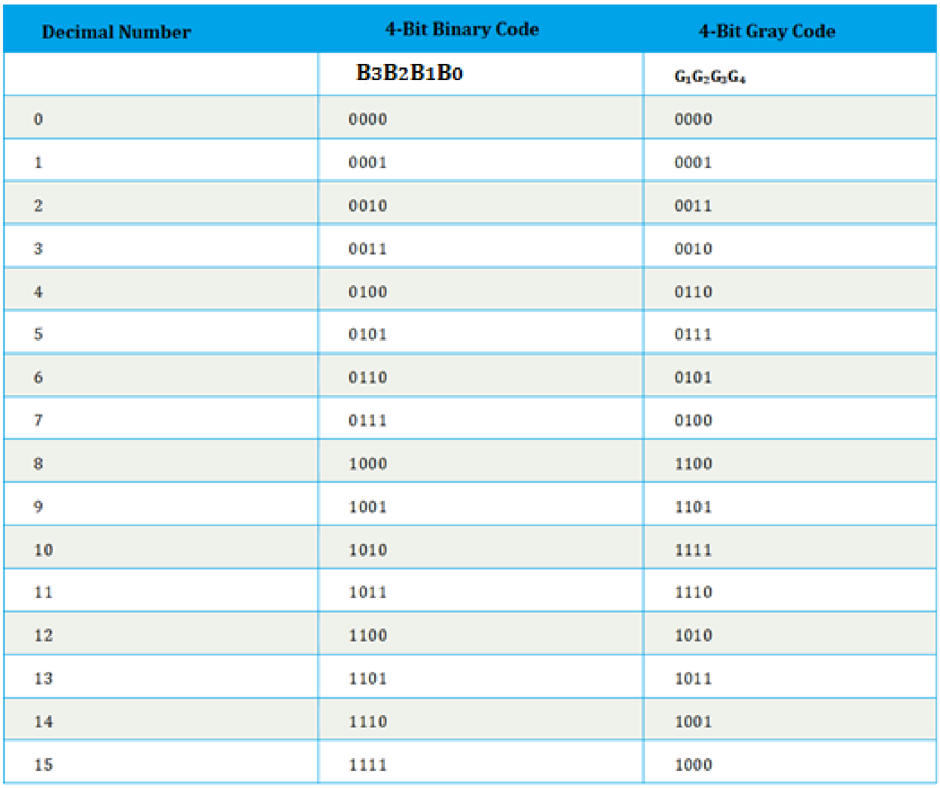
So,

w=AB+ACD
B=x' y'+x' z'+xyz
C=y' z+yz'
D=z'

Binary to Gray code converter.

The Binary to Gray code converter is a logical circuit that is used to convert the binary code into its equivalent Gray code. By putting the MSB of 1 below the axis and the MSB of 1 above the axis and reflecting the (n-1) bit code about an axis after 2^{n-1} rows, we can obtain the n-bit gray code.

The 4-bit binary to gray code conversion table is as follows:



Decimal Number	4-Bit Binary Code	4-Bit Gray Code
	B3B2B1B0	$G_1G_2G_3G_4$
0	0000	0000
1	0001	0001
2	0010	0011
3	0011	0010
4	0100	0110
5	0101	0111
6	0110	0101
7	0111	0100
8	1000	1100
9	1001	1101
10	1010	1111
11	1011	1110
12	1100	1010
13	1101	1011
14	1110	1001
15	1111	1000

How to Convert Binary to Gray Code:

- In the Gray code, the MSB will always be the same as the 1'st bit of the given binary number.
- In order to perform the 2nd bit of the gray code, we perform the exclusive-or (XOR) of the 1'st and 2nd bit of the binary number. It means that if both the bits are different, the result will be one else the result will be 0.
- In order to get the 3rd bit of the gray code, we need to perform the exclusive-or (XOR) of the 2nd and 3rd bit of the binary number. The process remains the same for the 4th bit of the Gray code. Let's take an example to understand these steps.

K-map for G_3

B_3B_2 \ B_1B_0	00	01	11	10
00	0	0	0	0
01	0	0	0	0
11	1	1	1	1
10	1	1	1	1

$$G_3 = B_3$$

K-map for G_2

B_3B_2 \ B_1B_0	00	01	11	10
00	0	0	0	0
01	1	1	1	1
11	0	0	0	0
10	1	1	1	1

$$G_2 = B_3'B_2+B_3B_2'$$

K-map for G_1

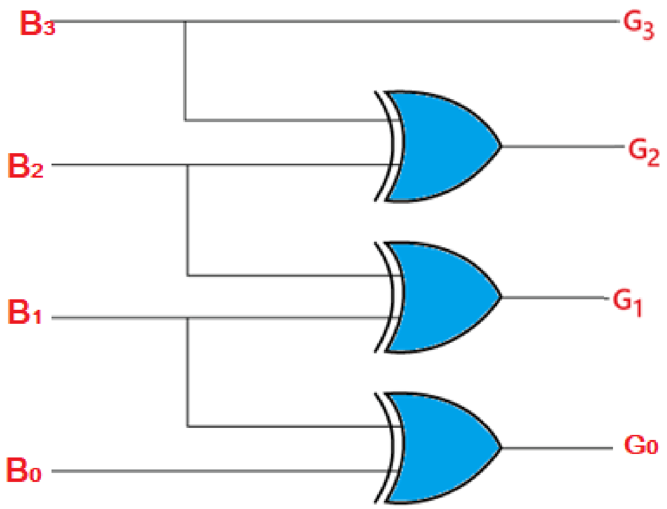
B_3B_2 \ B_1B_0	00	01	11	10
00	0	0	1	1
01	1	1	0	0
11	1	1	0	0
10	0	0	1	1

$$G_1 = B_2B_1'+B_2'B_1$$

K-map for G_0

B_3B_2 \ B_1B_0	00	01	11	10
00	0	1	0	1
01	0	1	0	1
11	0	1	0	1
10	0	1	0	1

$$G_0 = B_1'B_0+B_1B_0'$$



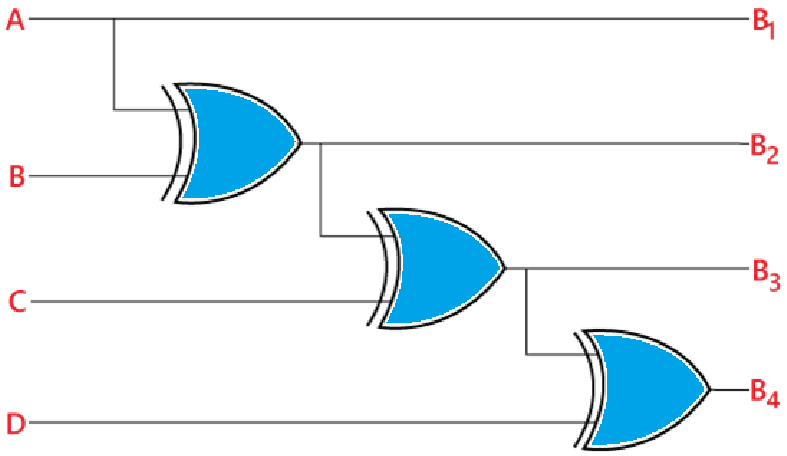
Logic Circuit for Binary to Gray Code Converter

Gray to Binary Code Conversion:

The Gray to Binary code converter is a logical circuit that is used to convert the gray code into its equivalent binary code. There is the following circuit used to convert the Gray code to binary number.

There are the following steps used to convert the Gray code into binary.

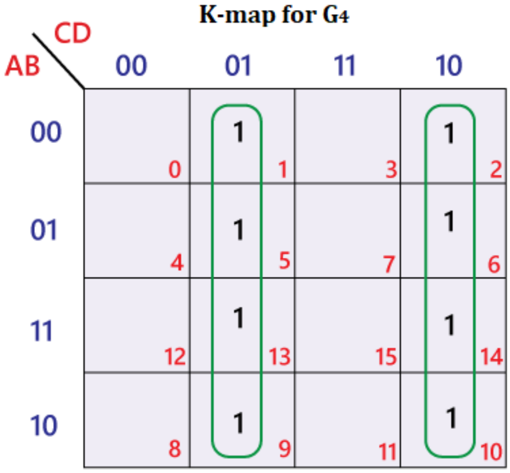
- Just like binary to gray, in gray to binary, the 1st bit of the binary number is similar to the MSB of the Gray code.
- The 2nd bit of the binary number is the same as the 1st bit of the binary number when the 2nd bit of the Gray code is 0; otherwise, the 2nd bit is altered bit of the 1st bit of binary number. It means if the 1st bit of the binary is 1, then the 2nd bit is 0, and if it is 0, then the 2nd bit be 1.
- The 2nd step continues for all the bits of the binary number.



Logic Circuit for Gray to Binary Code Converter

The Karnaugh maps (K-maps) for G_4, G_3, G_2, and G_1 are as follows:

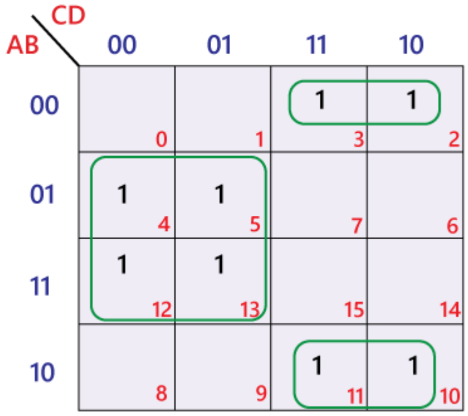
K-map for G_4



AB \ CD	00	01	11	10
00	0	1 1	3	1 2
01	4	1 5	7	1 6
11	12	1 13	15	1 14
10	8	1 9	11	1 10

$$G_4 = \bar{C}D + C\bar{D} = C \oplus D$$

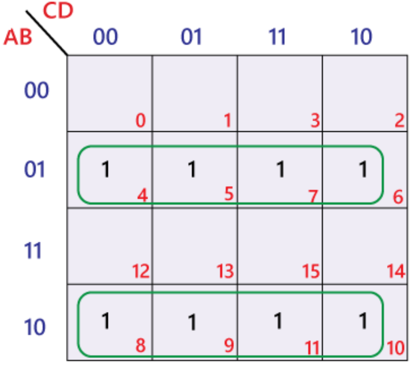
K-map for G_3



AB \ CD	00	01	11	10
00	0	1	1 3	1 2
01	1 4	1 5	7	6
11	1 12	1 13	15	14
10	8	9	1 11	1 10

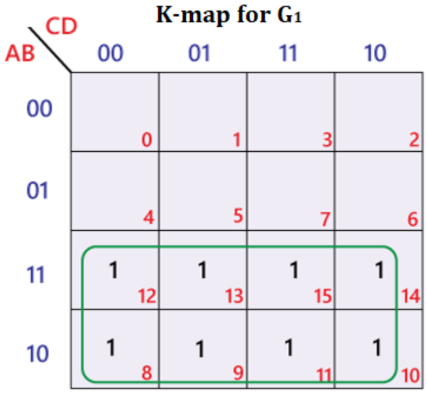
$$G_3 = \bar{B}C + B\bar{C} = B \oplus C$$

K-map for G2



$G_2 = \bar{A}B + A\bar{B} = A \oplus B$

K-map for G1



$G_1 = A$

Binary adder and subtractors:

Addition of two binary digits is most basic operation performed by the digital computer.

Types of binary adder:

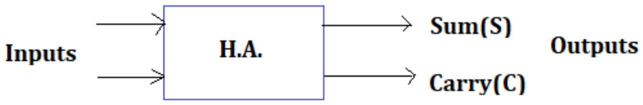
In this section we are going to learn digital circuit with which are used to 'add' two 'binary' numbers. The binary adder are of two types:

1)half adder & 2) Full Adder.

Half Adder:

Half adder is a combinational logic circuit with two input and two output. It is the basic building blocks for addition od two 'single' bit numbers. This circuit has two output namely 'carry' and 'sum' .The block diagram of half adder is as shows as(a)

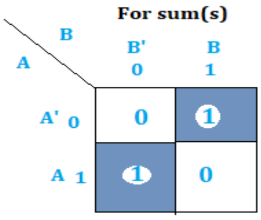
The half adder circuit is supposed to add two single bit binary numbers A and B . Therefore the truth table of a half adder is shown below (b)



(a) Block diagram

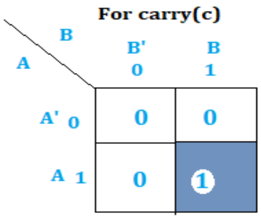
inputs		outputs	
A	B	Sum	Carry
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

(b) Truth Table



S= A'B +AB'

(a)K-map for sum output



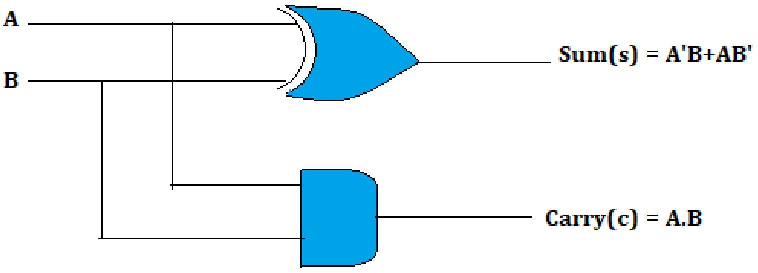
C= AB

(b) K-map for carry output

Boolean expression for sum(S) and carry(C) output are obtained from K-maps.

The disadvantage of half adder is that addition of three bits are not possible to perform.

Hence half adder circuit is as shown in figure below:



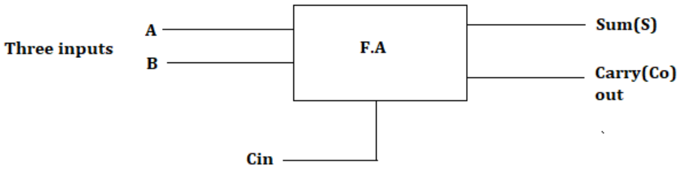
Half adder circuit

FULL ADDER:

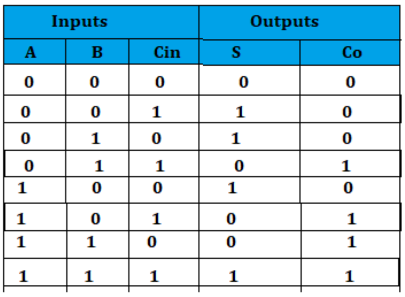
To overcome the drawback of half adder circuit ,a 3 single bit adder circuit called full adder is developed.

It can add two one-bit numbers A and B ,and carry C_0 , The full adder is a three input and two output combinational circuit.

The block diagram of full adder is as shown in below with truth table.



(a) Block diagram

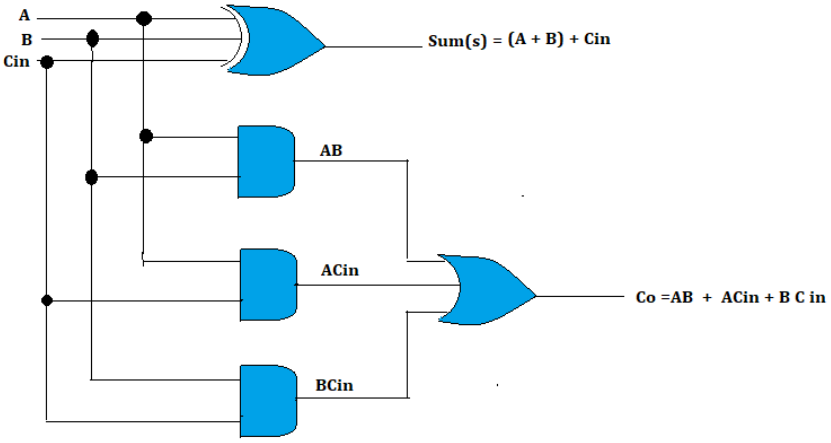


Inputs			Outputs	
A	B	Cin	S	Co
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

(b) Truth Table

The K-map:

-The k-map for the sum(s) and Carry(C_0) outputs and the corresponding Boolean expression are as shown in fig.



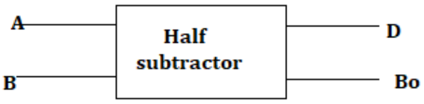
Half Subtractor:

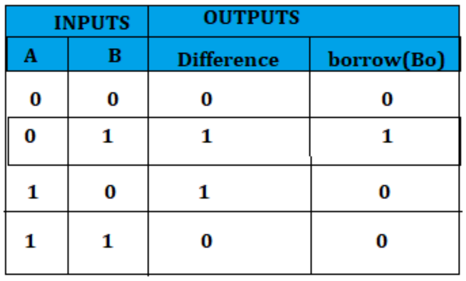
-half subtractor is a combinational circuit with two inputs and two outputs(difference and borrow.

-It produces the difference between the two binary bits at the input and also produced an output(borrow) to indicate if a 1 has been borrow.

-In the subtraction 9A-B) , A is called as minuend bit and B is called as subtrahend bit.

-Truth table chowing the output of a half subtractor for all the possible combinations of input.





INPUTS		OUTPUTS	
A	B	Difference	borrow(Bo)
0	0	0	0
0	1	1	1
1	0	1	0
1	1	0	0

Truth table for half subtractor

the k-maps for the output of half subtractor are as follow:

A \ B	B'	B
A'	0	1
A	1	0

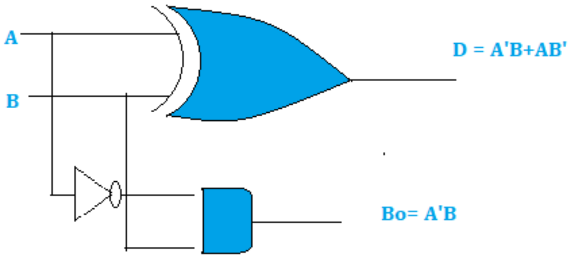
D = AB'+A'B

A \ B	B'	B
A'	0	1
A	0	0

Bo = A'B

Disadvantage of half subtractor.

Half subtractor can only perform the subtraction of two binary bits.But while performing the subtraction ,it does not take into account the borrow of the lower significant stage.

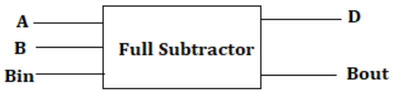


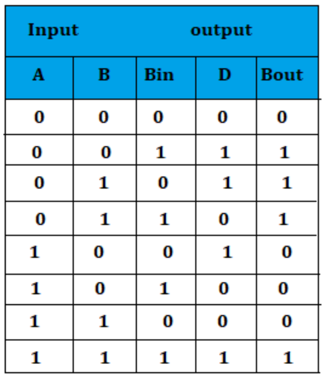
Full Subtractor:

-The disadvantage of a half subtractor is overcome if we use the full subtractor.

-The full subtractor is a combinational circuit with three inputs A,B, and Bin and two outputs D and Bo.

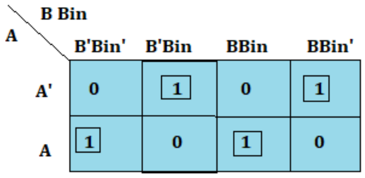
-A is menuend ,B is subtracted ,Bin is the borrow produced by the previous stage ,D is the difference output and Bo is the borrow output.





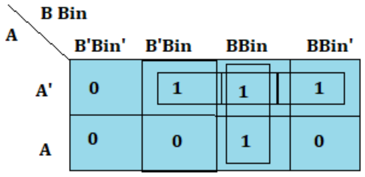
Input			output	
A	B	Bin	D	Bout
0	0	0	0	0
0	0	1	1	1
0	1	0	1	1
0	1	1	0	1
1	0	0	1	0
1	0	1	0	0
1	1	0	0	0
1	1	1	1	1

Truth Table



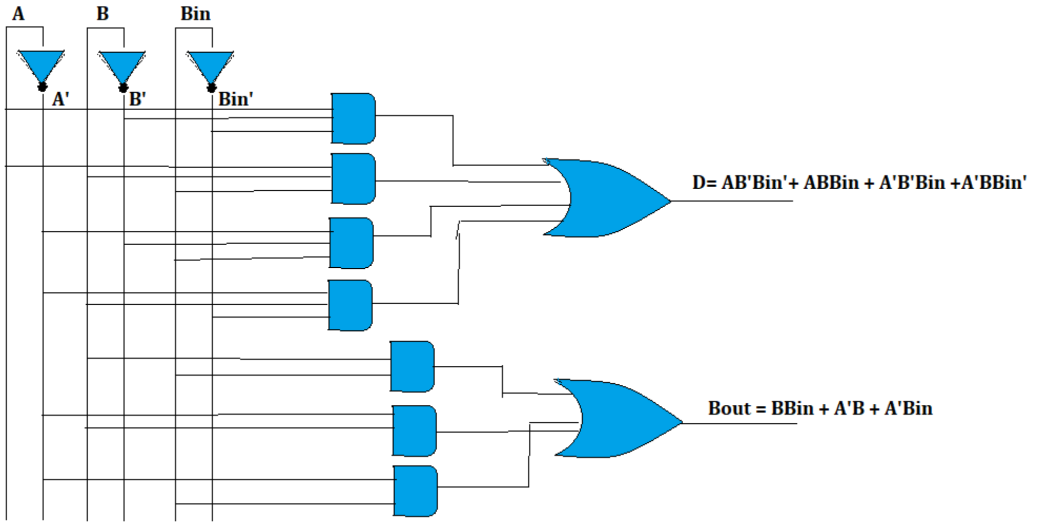
A \ B Bin	B'Bin'	B'Bin	BBin	BBin'
A'	0	1	0	1
A	1	0	1	0

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



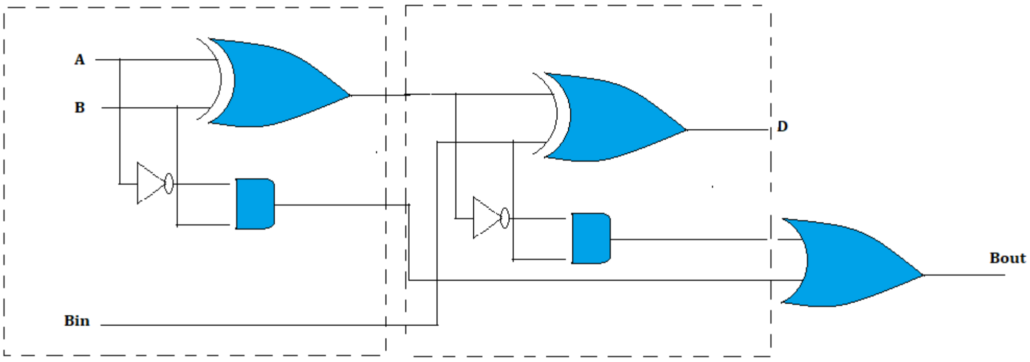
A \ B Bin	B'Bin'	B'Bin	BBin	BBin'
A'	0	1	1	1
A	0	0	1	0

Bout = BBin + A'B+A'Bin


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

Full subtractor using half subtractor:

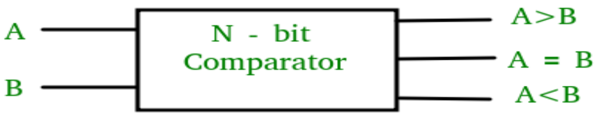
Fig. shows the implementation of a full subtractor using two half subtractors and an OR gate.



Full Subtractor Using Two Half Subtractor

Magnitude Comparators:

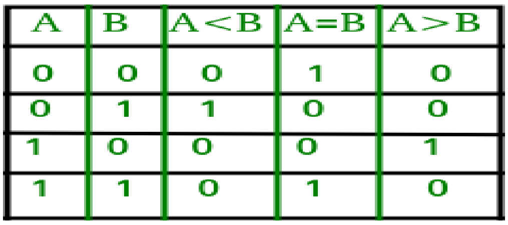
A magnitude digital Comparator is a combinational circuit that **compares two digital or binary numbers** in order to find out whether one binary number is equal, less than or greater than the other binary number. We logically design a circuit for which we will have two inputs one for A and other for B and have three output terminals, one for A > B condition, one for A = B condition and one for A < B condition.



1)One bit binary Comparator

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

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



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

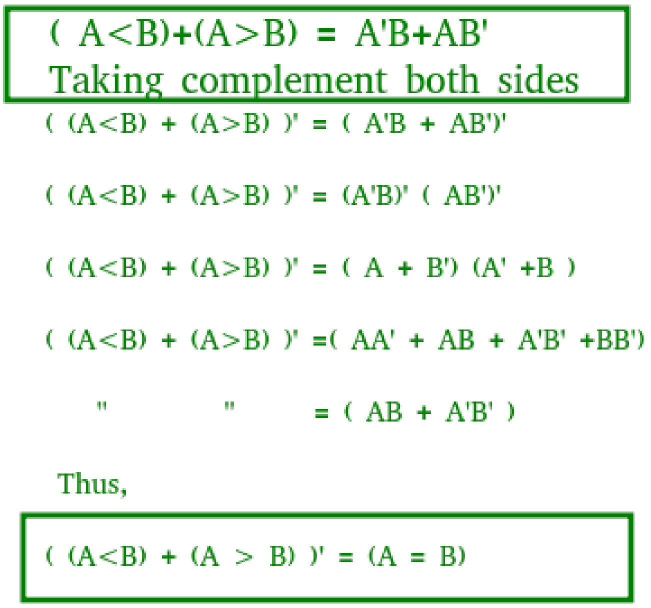
From the above truth table logical expressions for each output can be expressed as follows:

A>B: AB'

A<B: A'B

A=B: A'B' + AB

From the above expressions we can derive the following formula:



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

Taking complement both sides

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

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

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

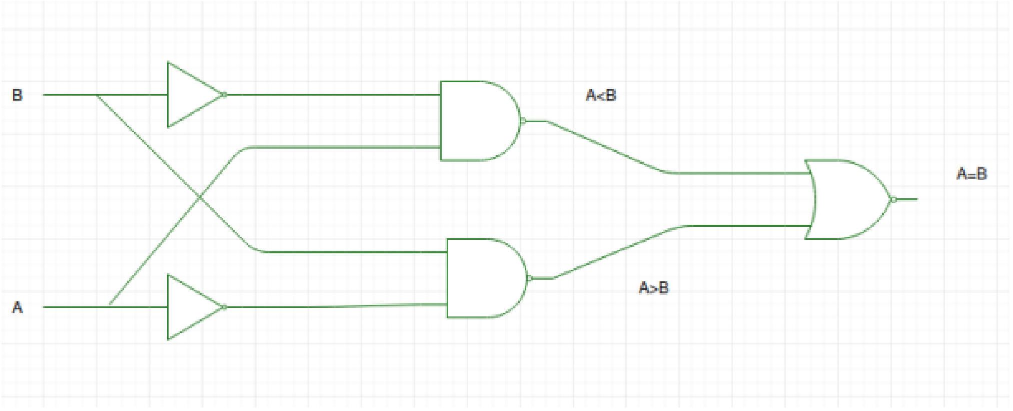
$((A<B) + (A>B))' =(AA' + AB + A'B' +BB')$

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

Thus,

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

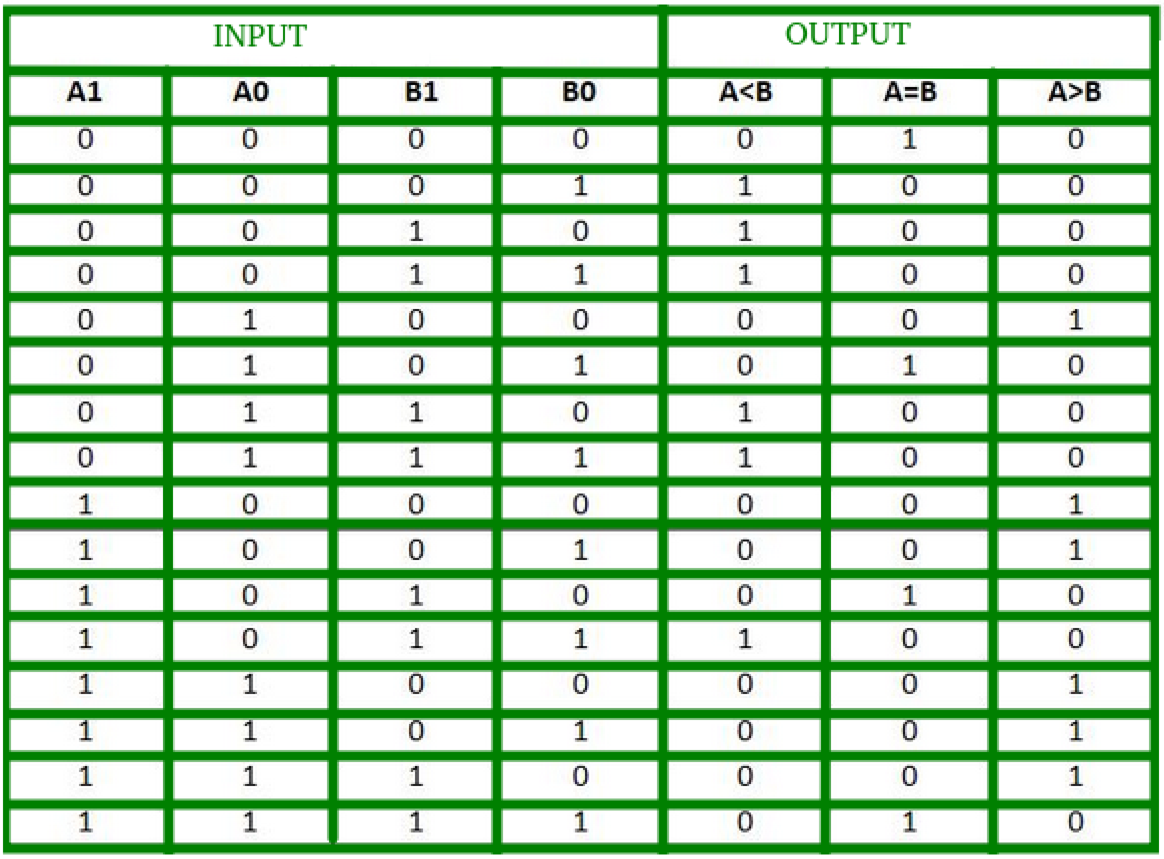
By using these Boolean expressions, we can implement a logic circuit for this comparator as given below:



2)Two bit binary Comparator:

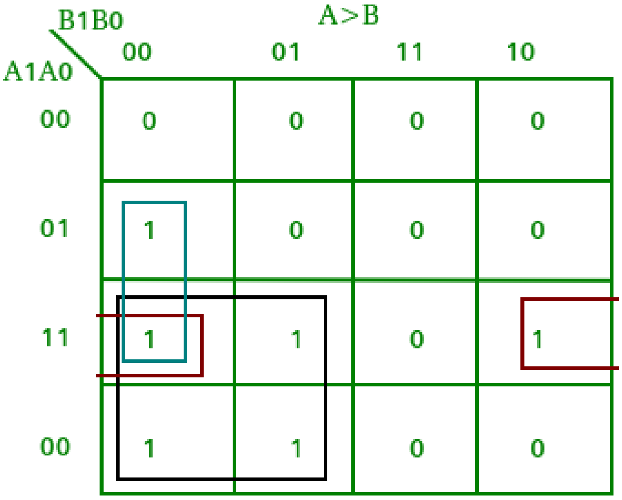
A comparator used to compare two binary numbers each of two bits is called a 2-bit Magnitude comparator. It consists of four inputs and three outputs to generate less than, equal to and greater than between two binary numbers.

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



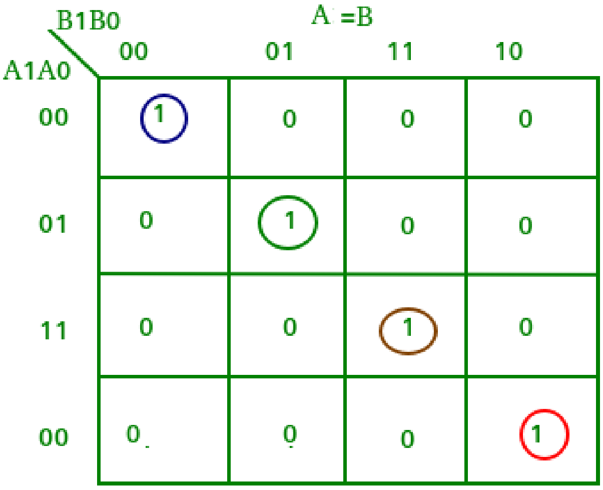
INPUT				OUTPUT		
A1	**A0**	**B1**	**B0**	**A<B**	**A=B**	**A>B**
0	0	0	0	0	1	0
0	0	0	1	1	0	0
0	0	1	0	1	0	0
0	0	1	1	1	0	0
0	1	0	0	0	0	1
0	1	0	1	0	1	0
0	1	1	0	1	0	0
0	1	1	1	1	0	0
1	0	0	0	0	0	1
1	0	0	1	0	0	1
1	0	1	0	0	1	0
1	0	1	1	1	0	0
1	1	0	0	0	0	1
1	1	0	1	0	0	1
1	1	1	0	0	0	1
1	1	1	1	0	1	0

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



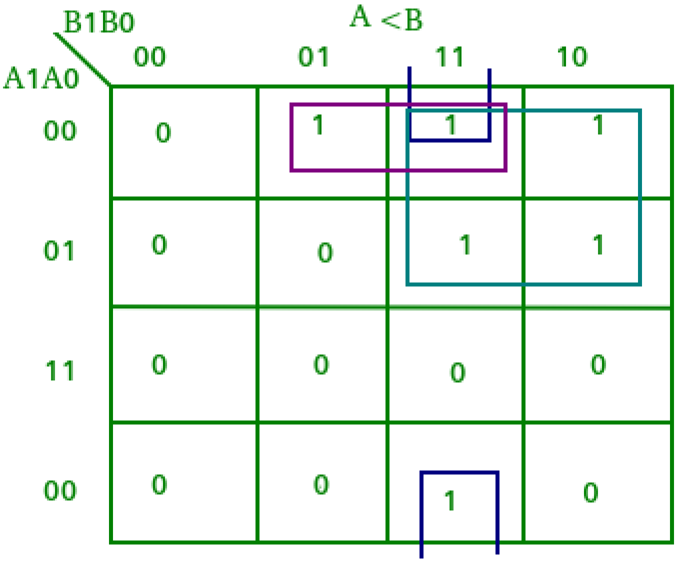
A>B

B1B0 A1A0	00	01	11	10
00	0	0	0	0
01	1	0	0	0
11	1	1	0	1
00	1	1	0	0



A =B

B1B0 A1A0	00	01	11	10
00	1	0	0	0
01	0	1	0	0
11	0	0	1	0
00	0	0	0	1



A <B

B1B0 A1A0	00	01	11	10
00	0	1	1	1
01	0	0	1	1
11	0	0	0	0
00	0	0	1	0

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

A>B:A1B1’ + A0B1’B0’ + A1A0B0’

A=B: A1’A0’B1’B0’ + A1’A0B1’B0 + A1A0B1B0 + A1A0’B1B0’

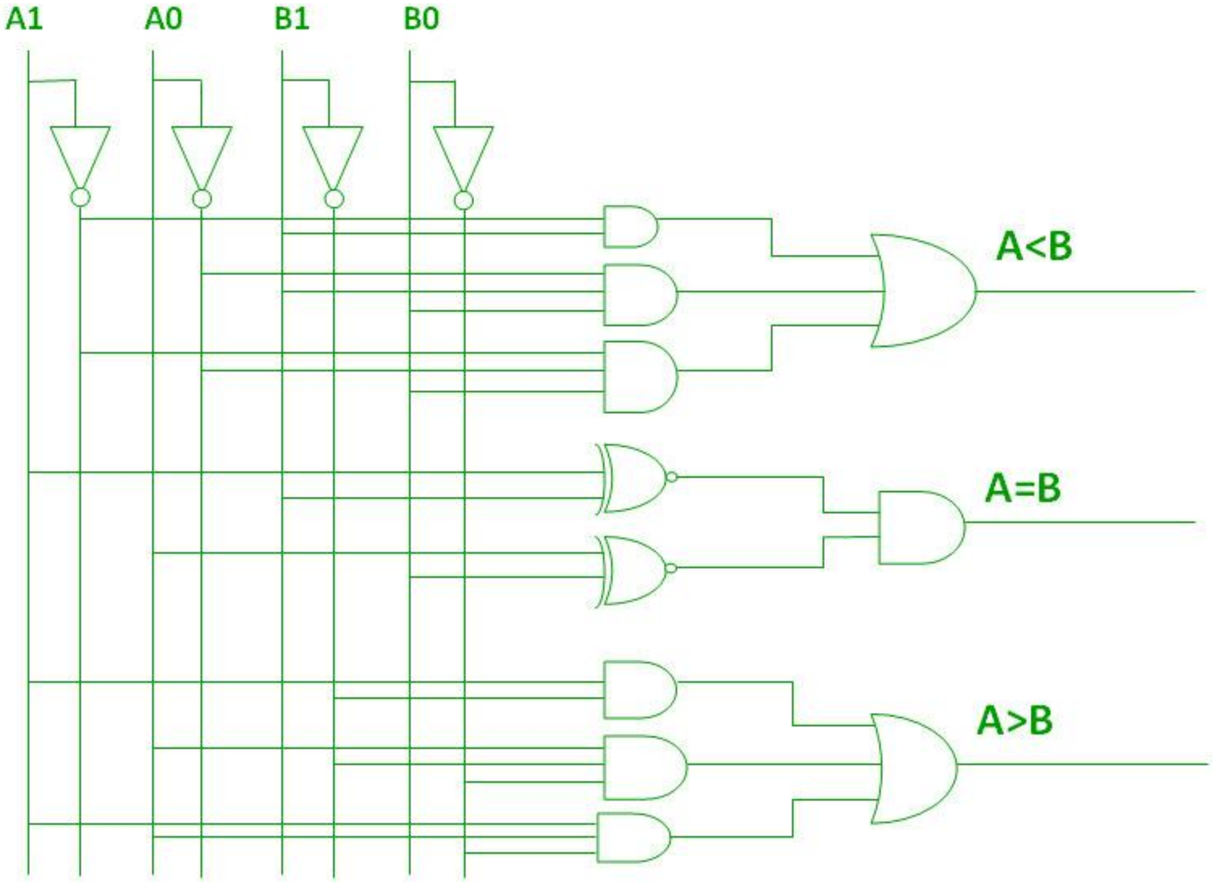
: A1’B1’ (A0’B0’ + A0B0) + A1B1 (A0B0 + A0’B0’)

: (A0B0 + A0’B0’) (A1B1 + A1’B1’)

: (A0 Ex-Nor B0) (A1 Ex-Nor B1)

A<B:A1’B1 + A0’B1B0 + A1’A0’B0

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



4 -Bit Parallel Adder(IC7483):

-The block diagram of 4 bit parallel adder using full adders is shown below. Let the two four bit words that are to be added be A and B and be denoted as follows:

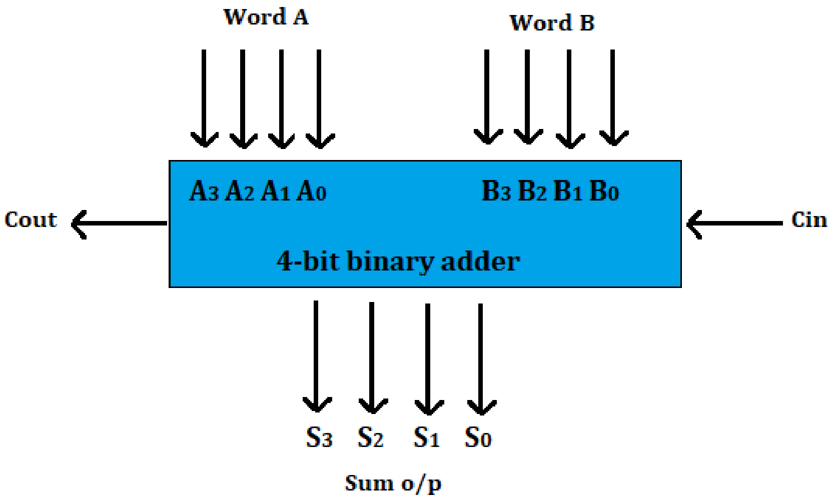
A =A3A2A1A0, B=B3B2B1B0

-A0 and B0 represent the LSBs of the four bit words A and B. Hence Full adder -0 is the lowest stage . Hence its Cin has been connected to 0 permanently.

-The rest of connections are exactly same as those done for the n- bit parallel adder.

-The 4 -bit parallel adder is a very common logic circuit.It is normally shown by a block diagram

It has two 4 bit input A3.....A0 and B3......B0 a carry input and carry output and $-bit sum output S3 S2 S1 S0.



4-Bit Adder IC 7463/Fast Adder:

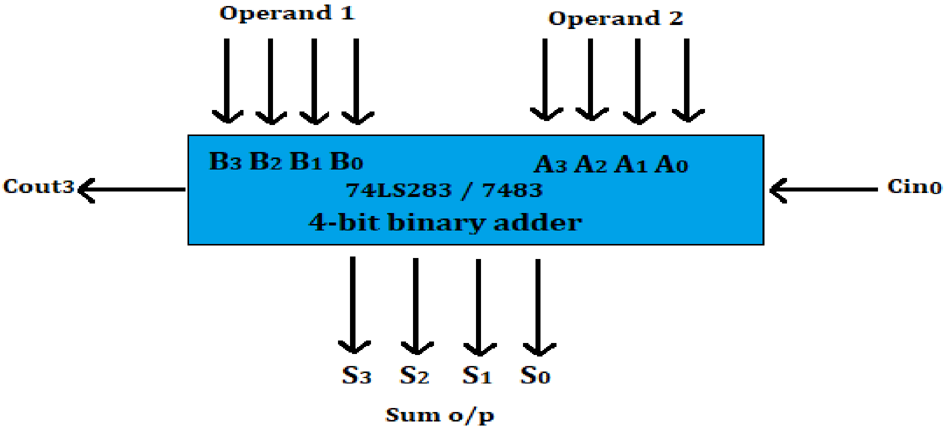
-The most common binary parallen adder in the integrated circuit form is IC 74 LS 83/74 LS 283.

-It is a 4-bit parallel adder, which consists of four interconnected full adder alongwith the look-ahead carry circuit.

-The IC 7483 and 74283 are TTL MSI(medium scale integration) for 4 bit parallel adders and both of them have the same pin configuration.

-In fig shows the functional symbol of IC 74 LS 283 . A3 A2 A1 A0 is four bit word A and B3 B2 B1 B0 is another word B. Both these words are applied at the inputs os the IC. 7483/74 LS 283.

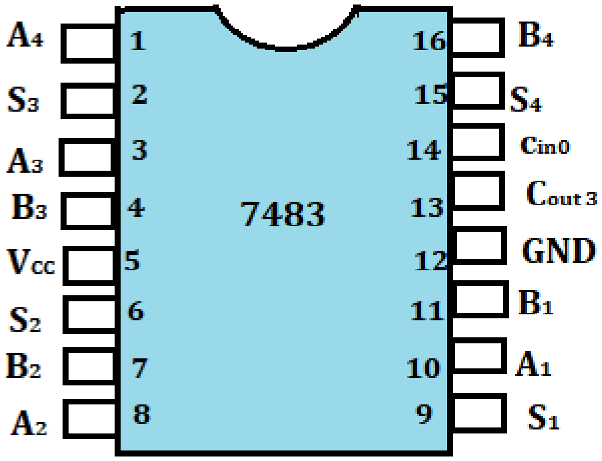
- Cin 0 is the input carry and Cout 3 represents the output carry. S3 S2 S1 S0 represent the sum output with S3 MSB.



Pin Diagram:

-The pin diagram of IC 7483 is shown in fig below,

- This IC adds the two Four bit word A and B and the bit at Cin 0 and produces a four bit sum output along with carry output at Cout 3.



4-Bit Subtractor using 4 bit adder(Using 2's complement):

4-bit binary Adder-Subtractor:

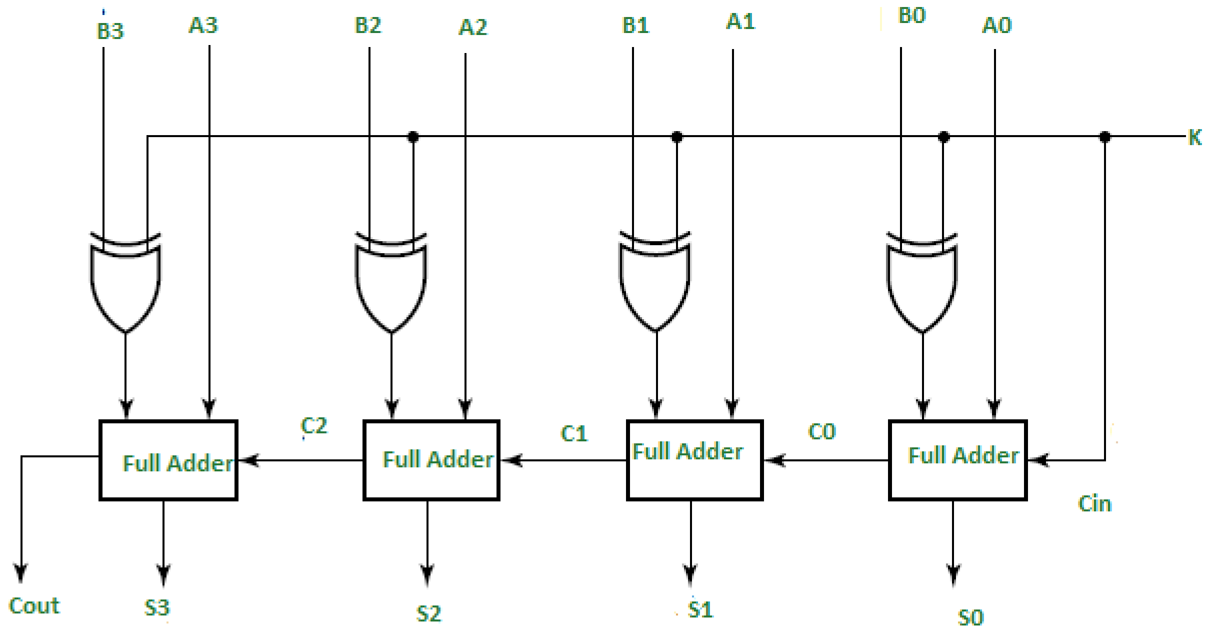
In Digital Circuits, A **Binary Adder-Subtractor** is one which is capable of both addition and subtraction of binary numbers in one circuit itself. The operation being performed depends upon the binary value the control signal holds. It is one of the components of the ALU (Arithmetic Logic Unit).
This Circuit Requires prerequisite knowledge of Exor Gate, Binary Addition and Subtraction, Full Adder.

Lets consider two 4-bit binary numbers A and B as inputs to the Digital Circuit for the operation with digits

```
A0 A1 A2 A3 for A

B0 B1 B2 B3 for B
```

The circuit consists of 4 full adders since we are performing operation on 4-bit numbers. There is a control line K that holds a binary value of either 0 or 1 which determines that the operation being carried out is addition or subtraction.



As shown in the figure, the first full adder has control line directly as its input(input carry C0), The input A0 (The least significant bit of A) is directly input in the full adder. The third input is the exor of B0 and K (S in fig But do not confuse it with Sum-S). The two outputs produced are Sum/Difference (S0) and Carry (C1).

If the value of K (Control line) is 1, th output of B0(exor)K=B0′(Complement B0). Thus the operation would be A+(B0′). Now 2's complement subtraction for two numbers A and B is given by A+B'. This suggests that when K=1, the operation being performed on the four bit numbers is subtraction.

Similarly If the Value of K=0, B0 (exor) K=B0. The operation is A+B which is simple binary addition. This suggests that When K=0, the operation being performed on the four bit numbers is addition.

Then C0 is serially passed to the second full adder as one of it's outputs.The sum/difference S0 is recorded as the least significant bit of the sum/difference. A1, A2, A3 are direct inputs to the second, third and fourth full adders.Then the third input is the B1, B2, B3 EXORed with K to the second, third and fourth full adder respectively. The carry C1, C2 are serially passed to the successive full adder as one of the inputs. C3 becomes the total carry to the sum/difference. S1, S2, S3 are recorded to form the result with S0.

For an n-bit binary adder-subtractor, we use n number of full adders.

Binary Multiplier(4x4 bit binary Multiplicatiion):

A binary multiplier is a combinational logic circuit used in digital systems to perform the multiplication of two binary numbers. These are most commonly used in various applications especially in the field of digital signal processing to perform the various algorithms.

Commercial applications like computers, mobiles, high speed calculators and some general purpose processors require binary multipliers.

Compared with addition and subtraction, multiplication is a complex process. In multiplication process, the number which is to be multiplied by the other number is called as multiplicand and the number multiplied is called as multiplier.

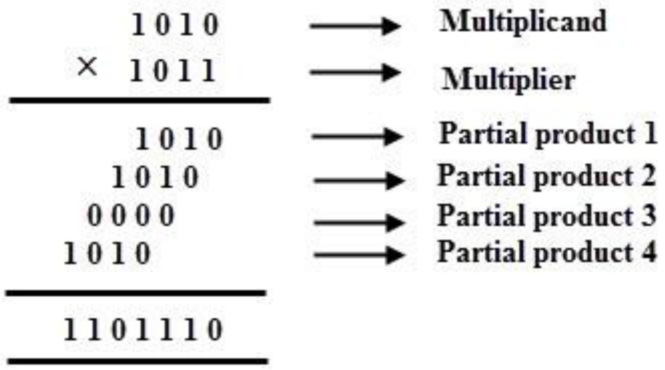
Binary Multiplication

Similar to the multiplication of decimal numbers, binary multiplication follows the same process for producing a product result of the two binary numbers. The binary multiplication is much easier as it contains only 0s and 1s. The four fundamental rules for binary multiplication are

$$0 \times 0 = 0$$
$$0 \times 1 = 0$$
$$1 \times 0 = 0$$
$$1 \times 1 = 1$$

The multiplication of two binary numbers can be performed by using two common methods, namely partial product addition and shifting, and using parallel multipliers.

Before discussing about the types, let us look at the unsigned binary numbers multiplication process. Consider a two 4 bit binary numbers as 1010 and 1011, and its multiplication of these two is given as



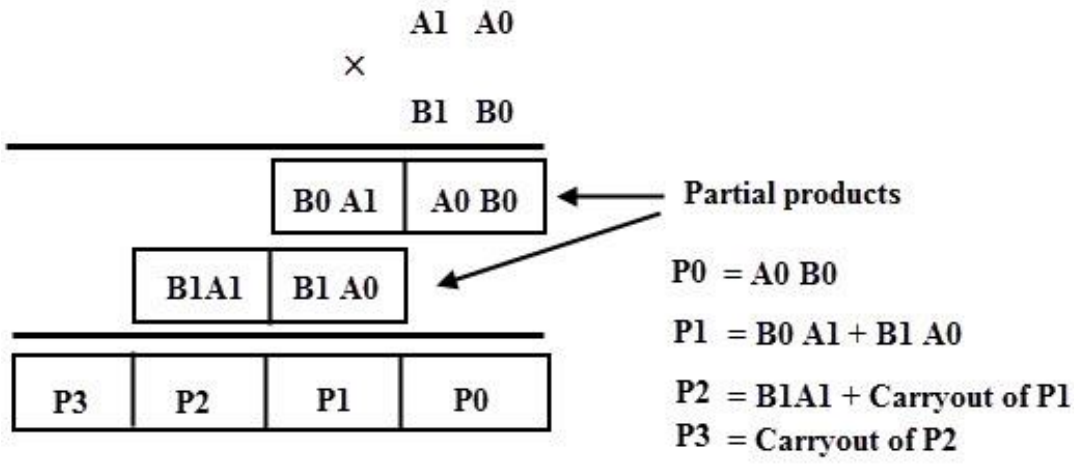
From the above multiplication, partial products are generated for each digit in the multiplier. Then all these partial products are added to produce the final product value. In the partial product multiplication, when the multiplier bit zero, the partial product is zero, and when the multiplier bit is 1, the resulted partial product is the multiplicand.

As similar to the decimal numbers, each successive partial product is shifted one position left relative to the preceding partial product before summing all partial products.

Therefore, this multiplication uses n-shifts and adds to multiply n-bit binary number. The combinational circuit implemented to perform such multiplication is called as an array multiplier or combinational multiplier.

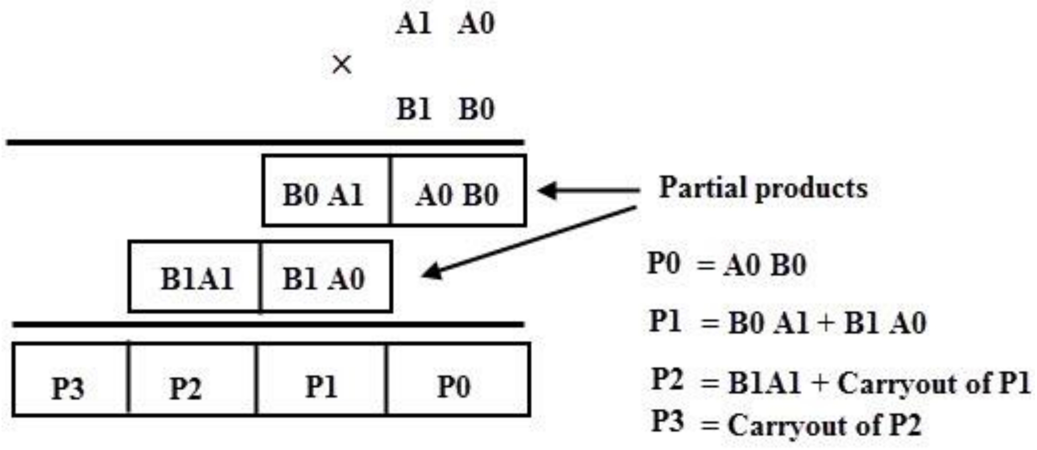
Parallel Binary Multiplier Circuit

Let us consider two unsigned 2 bit binary numbers A and B to generalize the multiplication process. The multiplicand A is equal to A1A0 and the multiplier B is equal to B1B0. The figure below shows the multiplication process of two 2 bit binary numbers.



This process involves the multiplication of two digits and the addition of digits with or without carry. After the multiplication of the each bit to the multiplicand, partial products are generated, and then these products are added to produce the total sum which represents the binary multiplication value.

This multiplication is implemented by combinational circuit such that the multiplication is performed with AND gates whereas the addition is carried out by using half adders as shown in figure.

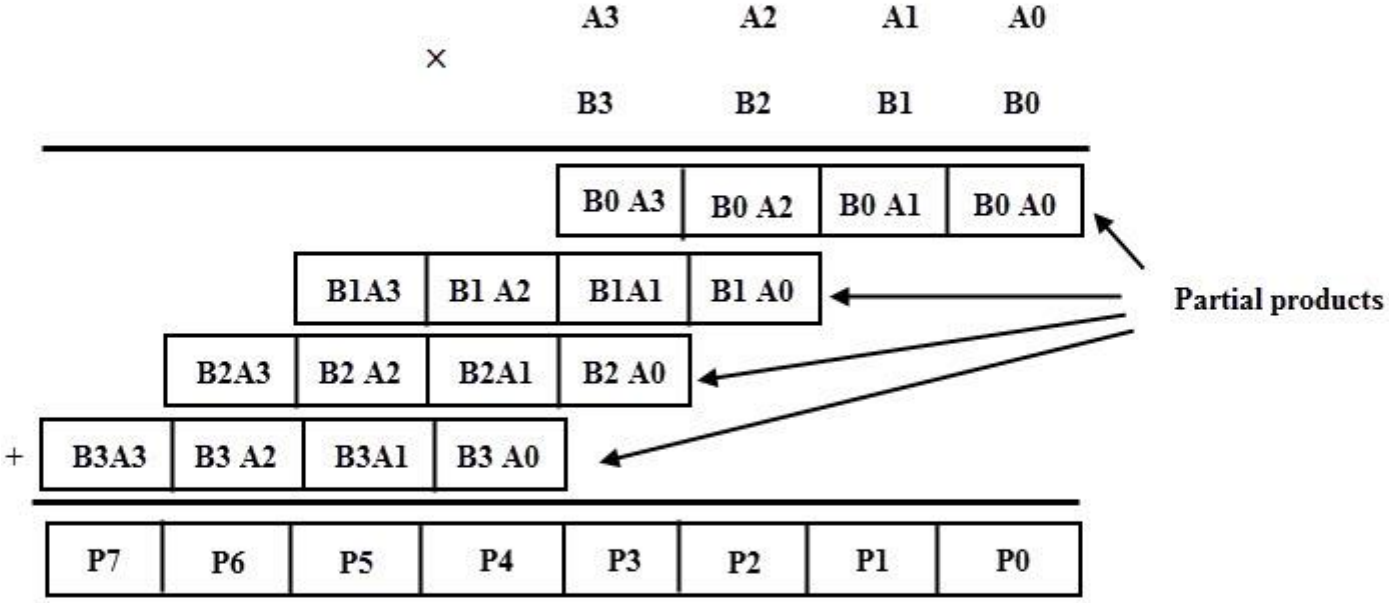


The first partial product is obtained by the AND gate which is nothing but a least significant bit of the multiplication result. Since the second partial product is shifted to the left position, the first partial second term and second partial product first term is added by half adder and produce the sum output along with the carry out.

This carry out is added at the next half adder as an input as shown in figure. Likewise, it produces the multiplication result of two binary numbers by using the simple circuit configuration. The multiplication of the two 2 bit number results a 4-bit binary number.

Let us consider two unsigned 4 bit numbers multiplication in which the multiplicand, A is equal to A3A2 A1A0 and the multiplier B is equal to B3B2B1B0. The partial products are produced depending on each multiplier bit multiplied by the multiplicand.

Each partial product consists of four product terms and these are shifted to the left relative to the previous partial product as shown in figure. All these partial products are added to produce the 8 bit product.

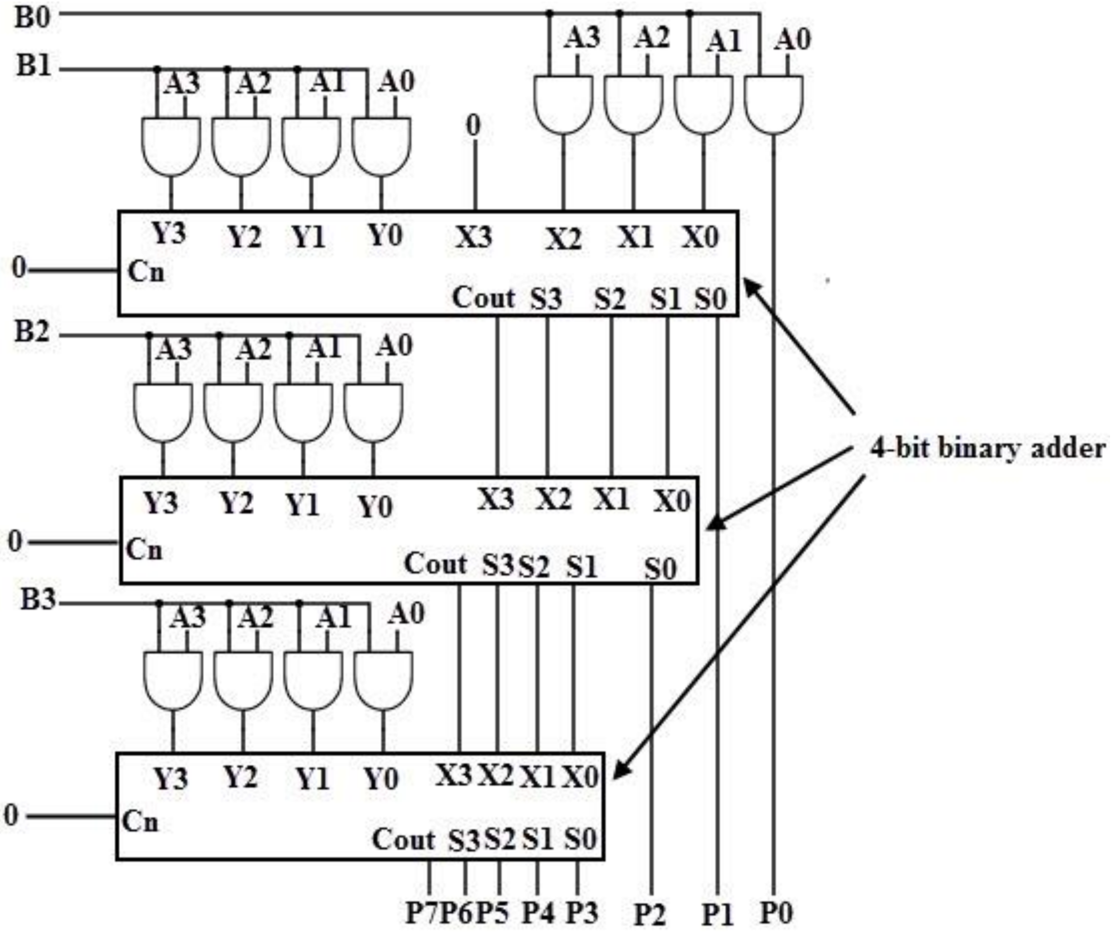


The logic circuit for the 4× 4 binary multiplication can be implemented by using three binary full adders along with AND gates.

In the above operation the first partial product is obtained by multiplying B0 with A3A2 A1A0, the second partial product is formed by multiplying B1 with A3A2 A1A0, likewise for 3rd and 4th partial products. So these partial products can be implemented with AND gates as shown in figure.

These partial products are then added by using 4 bit parallel adder. The three most significant bits of first partial product with carry (considered as zero) are added with second partial term in the first full adder.

Then the result is added to the next partial product with carry out and it goes on till the final partial product, finally it produces 8 bit sum which indicates the multiplication value of the two binary numbers.


B0
B1
B2
B3
A3
A2
A1
A0
0
Y3
Y2
Y1
Y0
X3
X2
X1
X0
Cn
Cout
S3
S2
S1
S0
4-bit binary adder
P7
P6
P5
P4
P3
P2
P1
P0

Unit-4

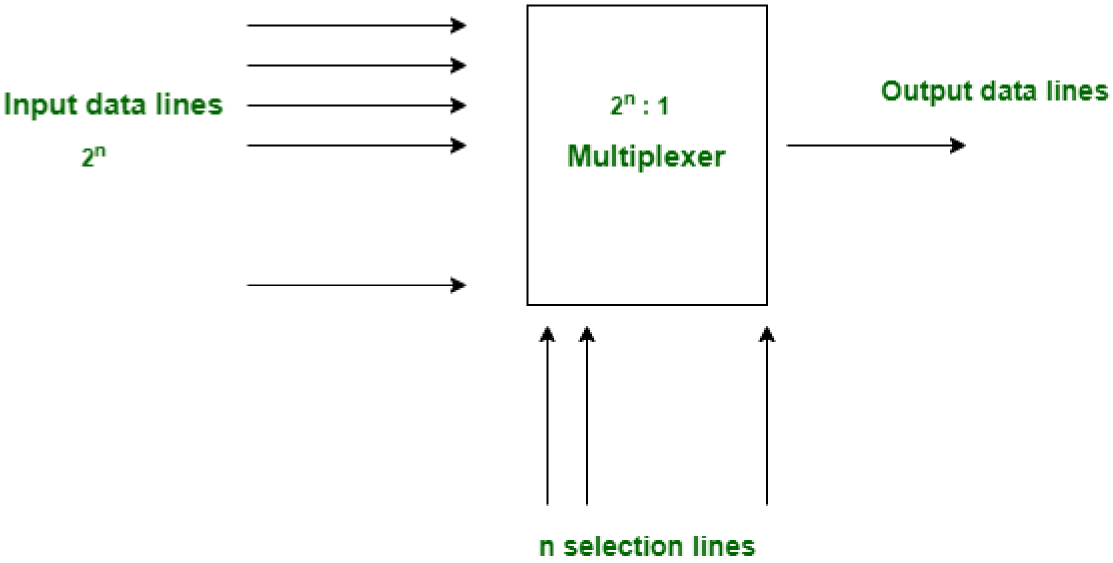
Multiplexer, Demultiplexer, ALU, Encoder and Decoder: Introduction, Multiplexer, Demultiplexer, Decoder, ALU, Encoders.

Sequential Circuits: Flip-Flop: Introduction, Terminologies used, S-R flip-flop, D flip-fop, JK flip-flop, Race-around condition, Master – slave JK flip-flop, T flip-flop, 12 14 | P a g e conversion from one type of flip-flop to another, Application of flip-flops.

Multiplexer:

Multiplexer is a combinational circuit that has maximum of 2^n data inputs, 'n' selection lines and single output line. One of these data inputs will be connected to the output based on the values of selection lines.

Since there are 'n' selection lines, there will be 2^n possible combinations of zeros and ones. So, each combination will select only one data input. Multiplexer is also called as **Mux.**



Types of multiplexer:

1)2:1 multiplexer

2)4:1 multiplexer

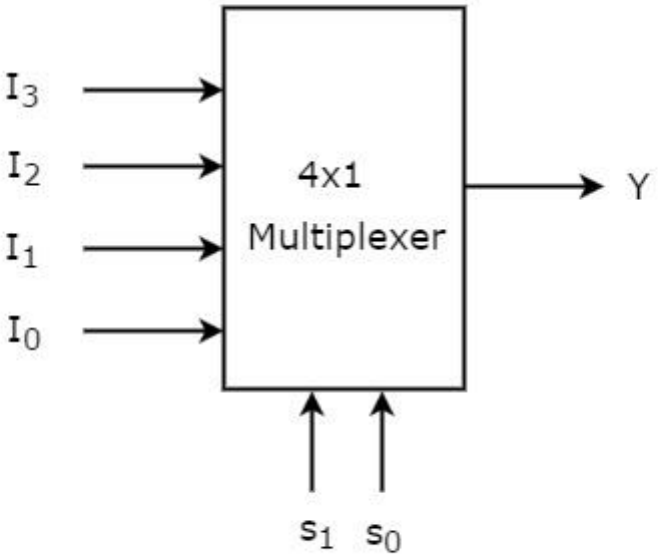
3)8:1 multiplexer

4)16:1 multiplexer

5)32:1 multiplexer

4x1 Multiplexer

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



One of these 4 inputs will be connected to the output based on the combination of inputs present at these two selection lines. **Truth table** of 4x1 Multiplexer is shown below.

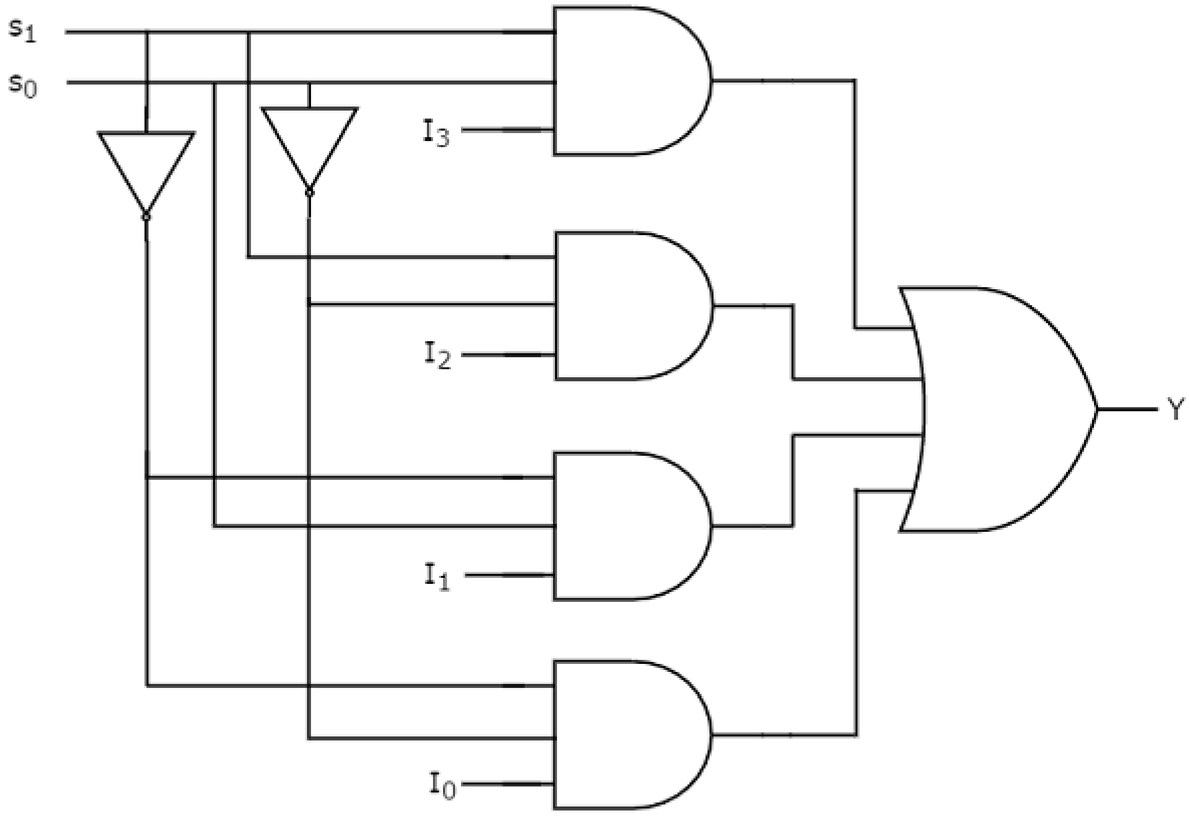
Selection Lines		Output
S_1	**S_0**	**Y**
0	0	I_0
0	1	I_1
1	0	I_2

1	1	I_3

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

Y=S1′S0′I0+S1′S0I1+S1S0′I2+S1S0I3Y=S1′S0′I0+S1′S0I1+S1S0′I2+S1S0I3

We can implement this Boolean function using Inverters, AND gates & OR gate. The **circuit diagram** of 4x1 multiplexer is shown in the following figure.



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

Implementation of Higher-order Multiplexers.

Now, let us implement the following two higher-order Multiplexers using lower-order Multiplexers.

- 8x1 Multiplexer
- 16x1 Multiplexer

8x1 Multiplexer:

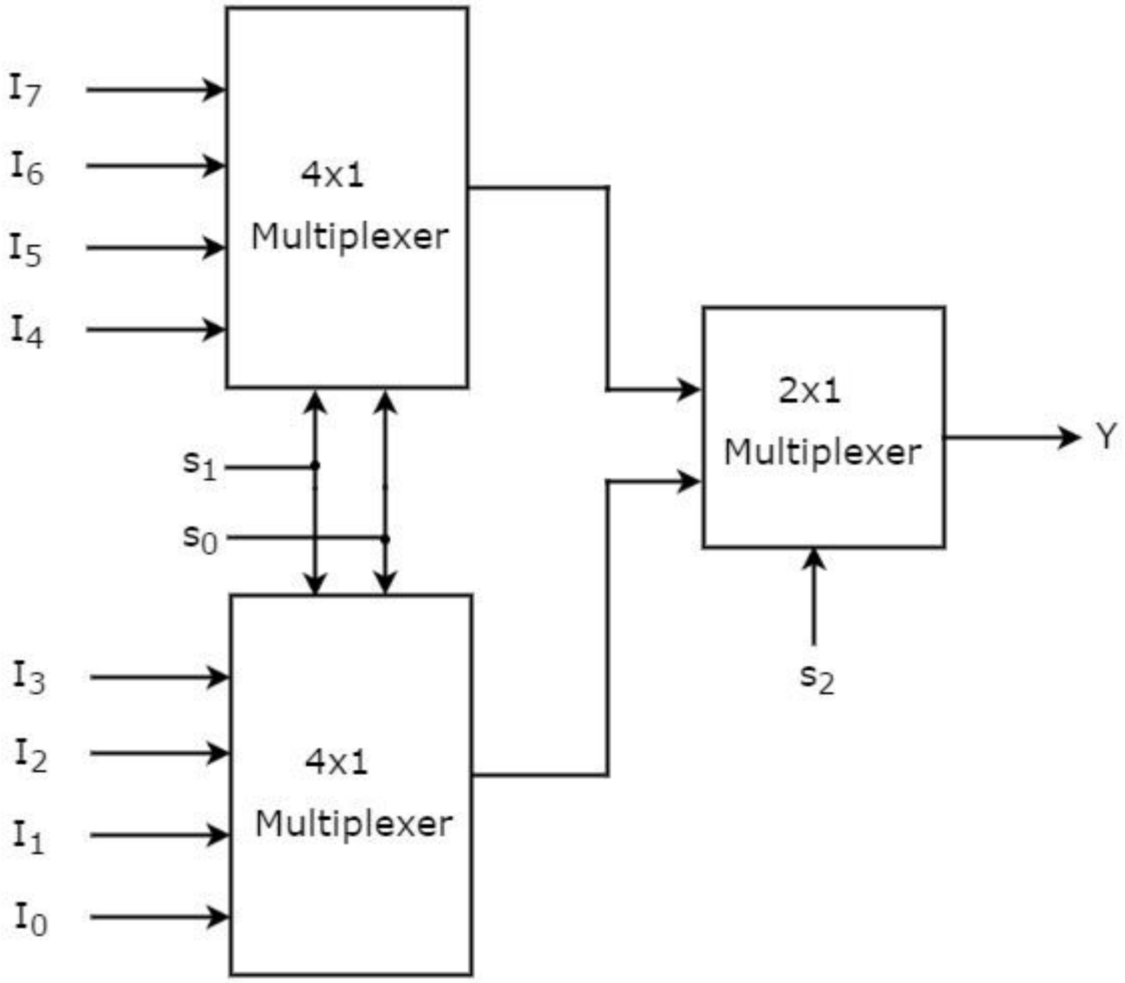
In this section, let us implement 8x1 Multiplexer using 4x1 Multiplexers and 2x1 Multiplexer. We know that 4x1 Multiplexer has 4 data inputs, 2 selection lines and one output. Whereas, 8x1 Multiplexer has 8 data inputs, 3 selection lines and one output.

So, we require two **4x1 Multiplexers** in first stage in order to get the 8 data inputs. Since, each 4x1 Multiplexer produces one output, we require a **2x1 Multiplexer** in second stage by considering the outputs of first stage as inputs and to produce the final output.

Let the 8x1 Multiplexer has eight data inputs I_7 to I_0, three selection lines s_2, s_1 & s0 and one output Y. The **Truth table** of 8x1 Multiplexer is shown below.

Selection Inputs			**Output**
S_2	**S_1**	**S_0**	**Y**
0	0	0	I_0
0	0	1	I_1
0	1	0	I_2
0	1	1	I_3
1	0	0	I_4
1	0	1	I_5
1	1	0	I_6
1	1	1	I_7

We can implement 8x1 Multiplexer using lower order Multiplexers easily by considering the above Truth table. The **block diagram** of 8x1 Multiplexer is shown in the following figure.



The same **selection lines, s_1 & s_0** are applied to both 4x1 Multiplexers. The data inputs of upper 4x1 Multiplexer are I_7 to I_4 and the data inputs of lower 4x1 Multiplexer are I_3 to I_0. Therefore, each 4x1 Multiplexer produces an output based on the values of selection lines, s_1 & s_0.

The outputs of first stage 4x1 Multiplexers are applied as inputs of 2x1 Multiplexer that is present in second stage. The other **selection line, s_2** is applied to 2x1 Multiplexer.

- If s_2 is zero, then the output of 2x1 Multiplexer will be one of the 4 inputs I_3 to I_0 based on the values of selection lines s_1 & s_0.
- If s_2 is one, then the output of 2x1 Multiplexer will be one of the 4 inputs I_7 to I_4 based on the values of selection lines s_1 & s_0.

Therefore, the overall combination of two 4x1 Multiplexers and one 2x1 Multiplexer performs as one 8x1 Multiplexer.

16x1 Multiplexer:

In this section, let us implement 16x1 Multiplexer using 8x1 Multiplexers and 2x1 Multiplexer. We know that 8x1 Multiplexer has 8 data inputs, 3 selection lines and one output. Whereas, 16x1 Multiplexer has 16 data inputs, 4 selection lines and one output.

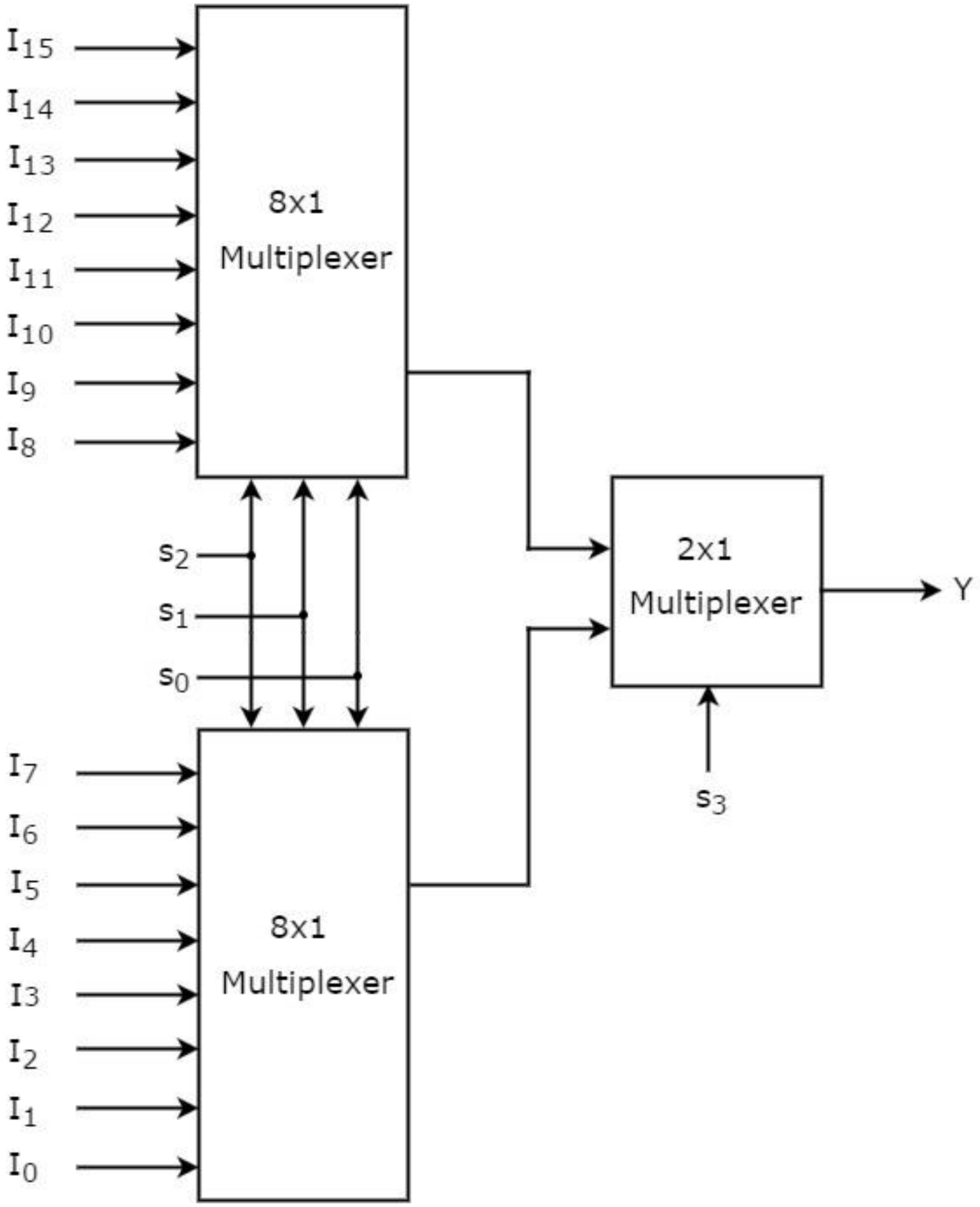
So, we require two **8x1 Multiplexers** in first stage in order to get the 16 data inputs. Since, each 8x1 Multiplexer produces one output, we require a 2x1 Multiplexer in second stage by considering the outputs of first stage as inputs and to produce the final output.

Let the 16x1 Multiplexer has sixteen data inputs I_{15} to I_0, four selection lines s_3 to s_0 and one output Y. The **Truth table** of 16x1 Multiplexer is shown below.

Selection Inputs				**Output**
S_3	**S_2**	**S_1**	**S_0**	**Y**
0	0	0	0	I_0
0	0	0	1	I_1
0	0	1	0	I_2
0	0	1	1	I_3
0	1	0	0	I_4
0	1	0	1	I_5
0	1	1	0	I_6
0	1	1	1	I_7
1	0	0	0	I_8
1	0	0	1	I_9
1	0	1	0	I_{10}
1	0	1	1	I_{11}
1	1	0	0	I_{12}
1	1	0	1	I_{13}

1	1	1	0	I_{14}
1	1	1	1	I_{15}

We can implement 16x1 Multiplexer using lower order Multiplexers easily by considering the above Truth table. The **block diagram** of 16x1 Multiplexer is shown in the following figure.



The **same selection lines, s_2, s_1 & s_0** are applied to both 8x1 Multiplexers. The data inputs of upper 8x1 Multiplexer are I_{15} to I_8 and the data inputs of lower 8x1 Multiplexer are I_7 to I_0. Therefore, each 8x1 Multiplexer produces an output based on the values of selection lines, s_2, s_1 & s_0.

The outputs of first stage 8x1 Multiplexers are applied as inputs of 2x1 Multiplexer that is present in second stage. The other **selection line, s_3** is applied to 2x1 Multiplexer.

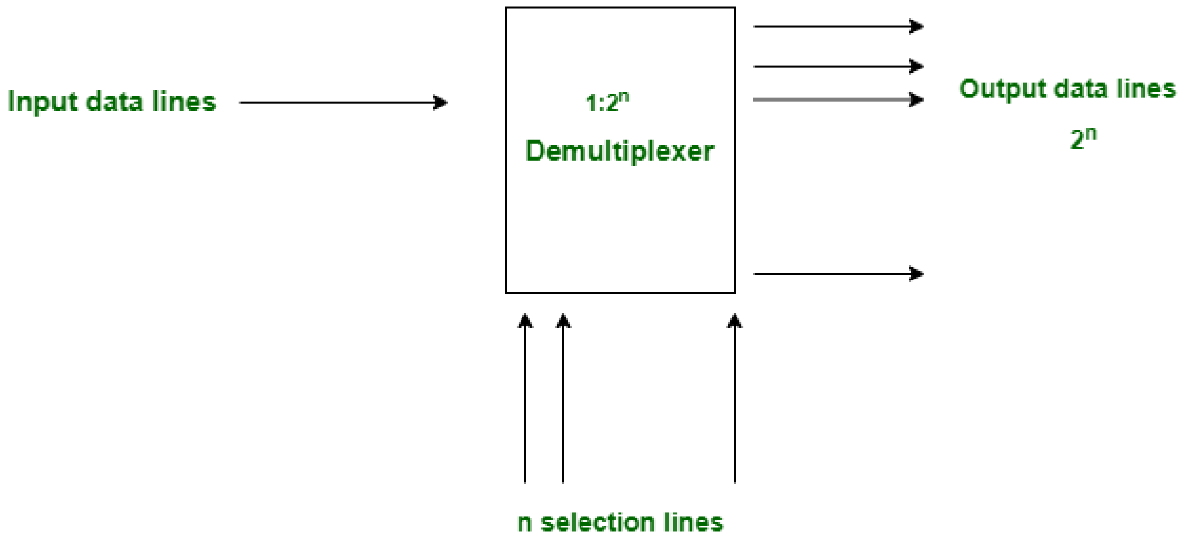
- If s_3 is zero, then the output of 2x1 Multiplexer will be one of the 8 inputs Is_7 to I_0 based on the values of selection lines s_2, s_1 & s_0.
- If s_3 is one, then the output of 2x1 Multiplexer will be one of the 8 inputs I_{15} to I_8 based on the values of selection lines s_2, s_1 & s_0.

Therefore, the overall combination of two 8x1 Multiplexers and one 2x1 Multiplexer performs as one 16x1 Multiplexer.

De- Multiplexer:

De-Multiplexer is a combinational circuit that performs the reverse operation of Multiplexer. It has single input, 'n' selection lines and maximum of 2^n outputs. The input will be connected to one of these outputs based on the values of selection lines.

Since there are 'n' selection lines, there will be 2^n possible combinations of zeros and ones. So, each combination can select only one output. De-Multiplexer is also called as **De-Mux**.



Types Of De-Multiplexer:

1)1:2 De-multiplexer

1)1:4 De-multiplexer

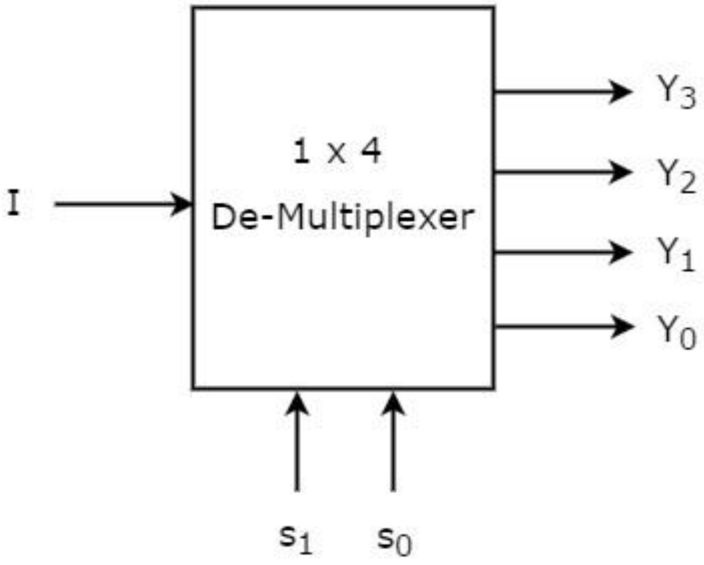
1)1:8 De-multiplexer

1)1:16 De-multiplexer

1)1:32 De-multiplexer

1x4 De-Multiplexer

1x4 De-Multiplexer has one input I, two selection lines, s_1 & s_0 and four outputs Y_3, Y_2, Y_1 &Y_0. The **block diagram** of 1x4 De-Multiplexer is shown in the following figure.



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

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

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

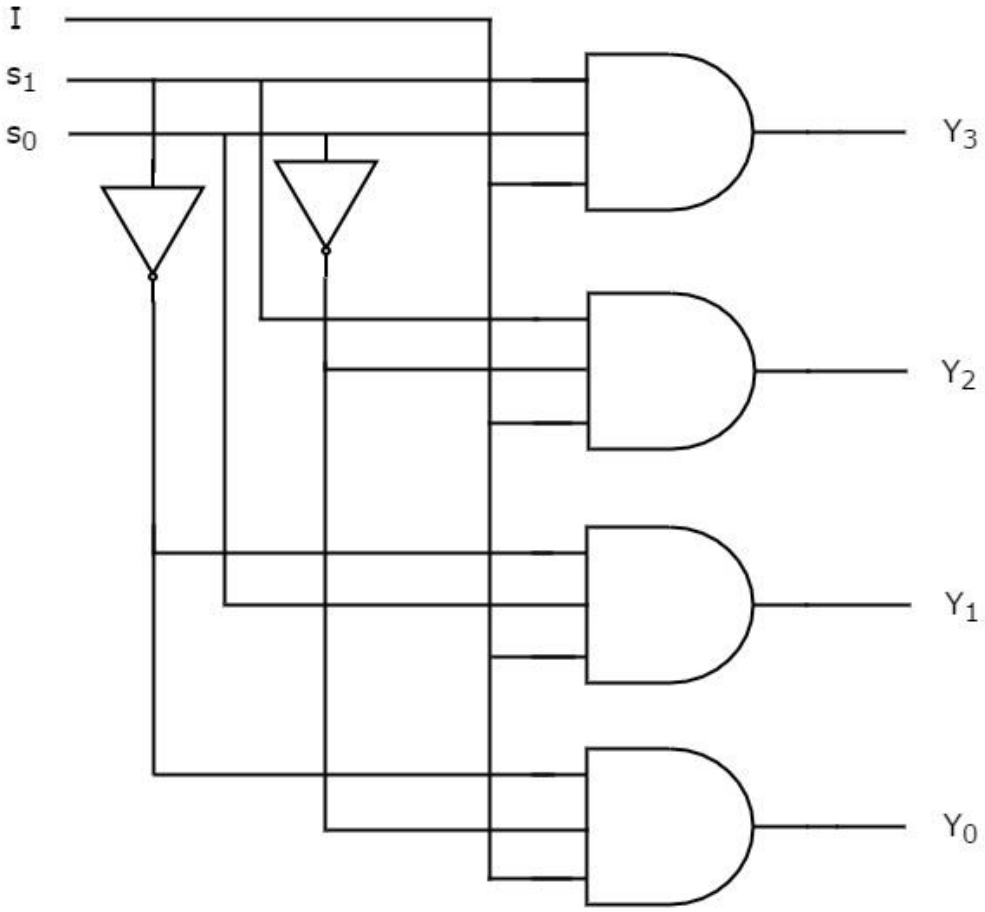
$$Y3=s1s0IY3=s1s0I$$

$$Y2=s1s0'IY2=s1s0'I$$

$$Y1=s1's0IY1=s1's0I$$

$$Y0=s1's0'IY0=s1's0'I$$

We can implement these Boolean functions using Inverters & 3-input AND gates. The **circuit diagram** of 1x4 De-Multiplexer is shown in the following figure.



We can easily understand the operation of the above circuit. Similarly, you can implement 1x8 De-Multiplexer and 1x16 De-Multiplexer by following the same procedure.

Implementation of Higher-order de-multiplexer:

Now, let us implement the following two higher-order De-Multiplexers using lower-order De-Multiplexers.

- 1x8 De-Multiplexer
- 1x16 De-Multiplexer

1x8 De-Multiplexer

In this section, let us implement 1x8 De-Multiplexer using 1x4 De-Multiplexers and 1x2 De-Multiplexer. We know that 1x4 De-Multiplexer has single input, two selection lines and four outputs. Whereas, 1x8 De-Multiplexer has single input, three selection lines and eight outputs.

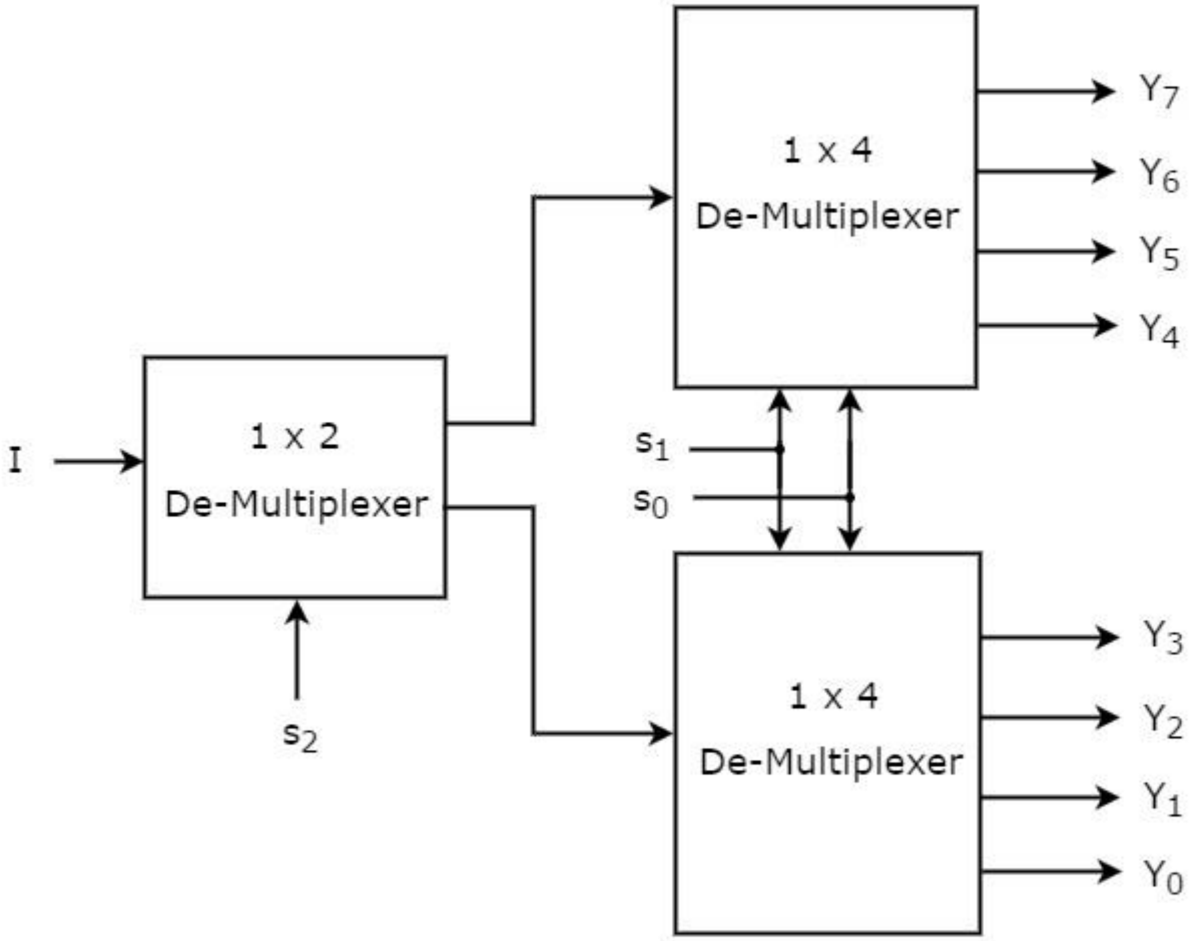
So, we require two **1x4 De-Multiplexers** in second stage in order to get the final eight outputs. Since, the number of inputs in second stage is two, we require **1x2 DeMultiplexer** in first stage so that the outputs of first stage will be the inputs of second stage. Input of this 1x2 De-Multiplexer will be the overall input of 1x8 De-Multiplexer.

Let the 1x8 De-Multiplexer has one input I, three selection lines s_2, s_1 & s_0 and outputs Y_7 to Y_0. The **Truth table** of 1x8 De-Multiplexer is shown below.

Selection Inputs			**Outputs**							
s_2	**s_1**	**s_0**	**Y_7**	**Y_6**	**Y_5**	**Y_4**	**Y_3**	**Y_2**	**Y_1**	**Y_0**
0	0	0	0	0	0	0	0	0	0	**I**
0	0	1	0	0	0	0	0	0	**I**	0
0	1	0	0	0	0	0	0	**I**	0	0
0	1	1	0	0	0	0	**I**	0	0	0
1	0	0	0	0	0	**I**	0	0	0	0
1	0	1	0	0	**I**	0	0	0	0	0

1	1	0	0	**I**	0	0	0	0	0	0
1	1	1	**I**	0	0	0	0	0	0	0

We can implement 1x8 De-Multiplexer using lower order Multiplexers easily by considering the above Truth table. The **block diagram** of 1x8 De-Multiplexer is shown in the following figure.



The common **selection lines, s_1 & s_0** are applied to both 1x4 De-Multiplexers. The outputs of upper 1x4 De-Multiplexer are Y_7 to Y_4 and the outputs of lower 1x4 De-Multiplexer are Y_3 to Y_0.

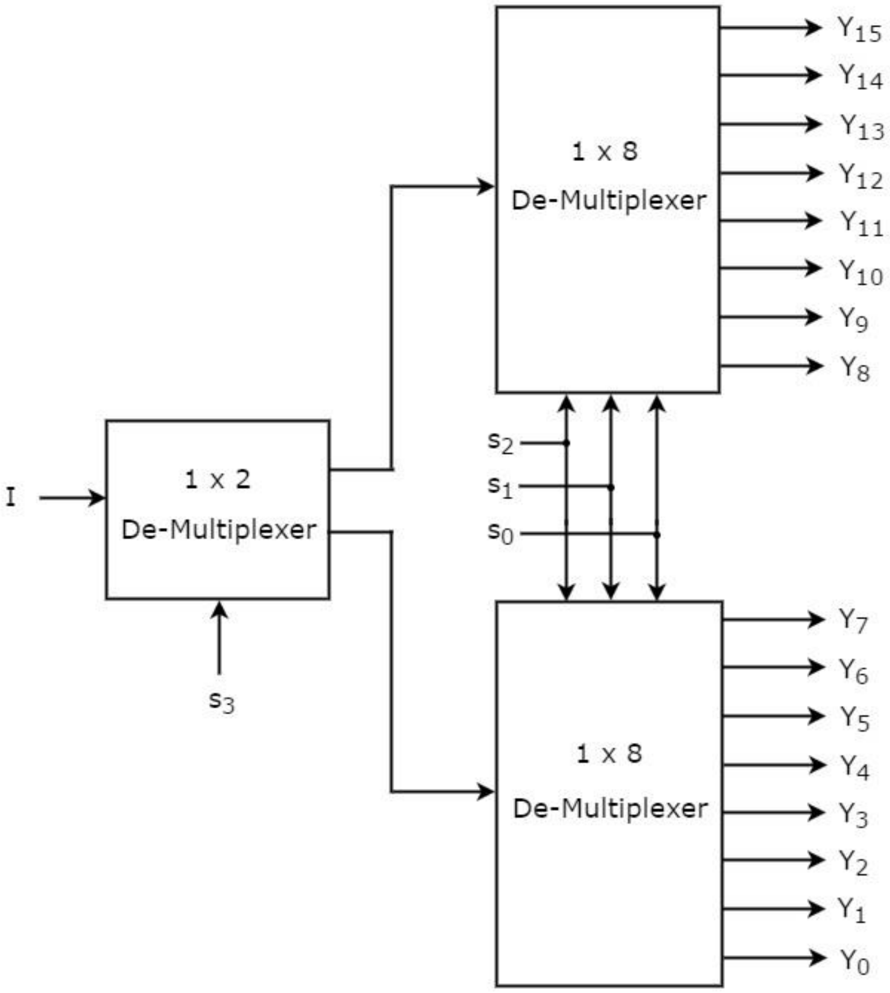
The other **selection line, s_2** is applied to 1x2 De-Multiplexer. If s_2 is zero, then one of the four outputs of lower 1x4 De-Multiplexer will be equal to input, I based on the values of selection lines s_1 & s_0. Similarly, if s_2 is one, then one of the four outputs of upper 1x4 DeMultiplexer will be equal to input, I based on the values of selection lines s_1 & s_0.

1x16 De-Multiplexer

In this section, let us implement 1x16 De-Multiplexer using 1x8 De-Multiplexers and 1x2 De-Multiplexer. We know that 1x8 De-Multiplexer has single input, three selection lines and eight outputs. Whereas, 1x16 De-Multiplexer has single input, four selection lines and sixteen outputs.

So, we require two **1x8 De-Multiplexers** in second stage in order to get the final sixteen outputs. Since, the number of inputs in second stage is two, we require **1x2 DeMultiplexer** in first stage so that the outputs of first stage will be the inputs of second stage. Input of this 1x2 De-Multiplexer will be the overall input of 1x16 De-Multiplexer.

Let the 1x16 De-Multiplexer has one input I, four selection lines s_3, s_2, s_1 & s_0 and outputs Y_{15} to Y_0. The **block diagram** of 1x16 De-Multiplexer using lower order Multiplexers is shown in the following figure.



The common **selection lines s_2, s_1 & s_0** are applied to both 1x8 De-Multiplexers. The outputs of upper 1x8 De-Multiplexer are Y_{15} to Y_8 and the outputs of lower 1x8 DeMultiplexer are Y_7 to Y_0.

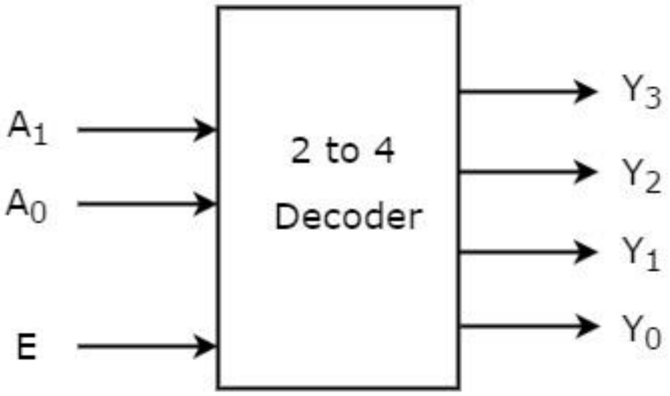
The other **selection line, s_3** is applied to 1x2 De-Multiplexer. If s_3 is zero, then one of the eight outputs of lower 1x8 De-Multiplexer will be equal to input, I based on the values of selection lines s_2, s_1 & s_0. Similarly, if s3 is one, then one of the 8 outputs of upper 1x8 De-Multiplexer will be equal to input, I based on the values of selection lines s_2, s_1 & s_0

Decoder:

Decoder is a combinational circuit that has 'n' input lines and maximum of 2^n output lines. One of these outputs will be active High based on the combination of inputs present, when the decoder is enabled. That means decoder detects a particular code. The outputs of the decoder are nothing but the **min terms** of 'n' input variables lineslines, when it is enabled.

2 to 4 Decoder

Let 2 to 4 Decoder has two inputs A_1 & A_0 and four outputs Y_3, Y_2, Y_1 & Y_0. The **block diagram** of 2 to 4 decoder is shown in the following figure.



One of these four outputs will be '1' for each combination of inputs when enable, E is '1'. The **Truth table** of 2 to 4 decoder is shown below.

Enable	**Inputs**		**Outputs**			
E	**A_1**	**A_0**	**Y_3**	**Y_2**	**Y_1**	**Y_0**
0	x	x	0	0	0	0
1	0	0	0	0	0	1
1	0	1	0	0	1	0

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

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

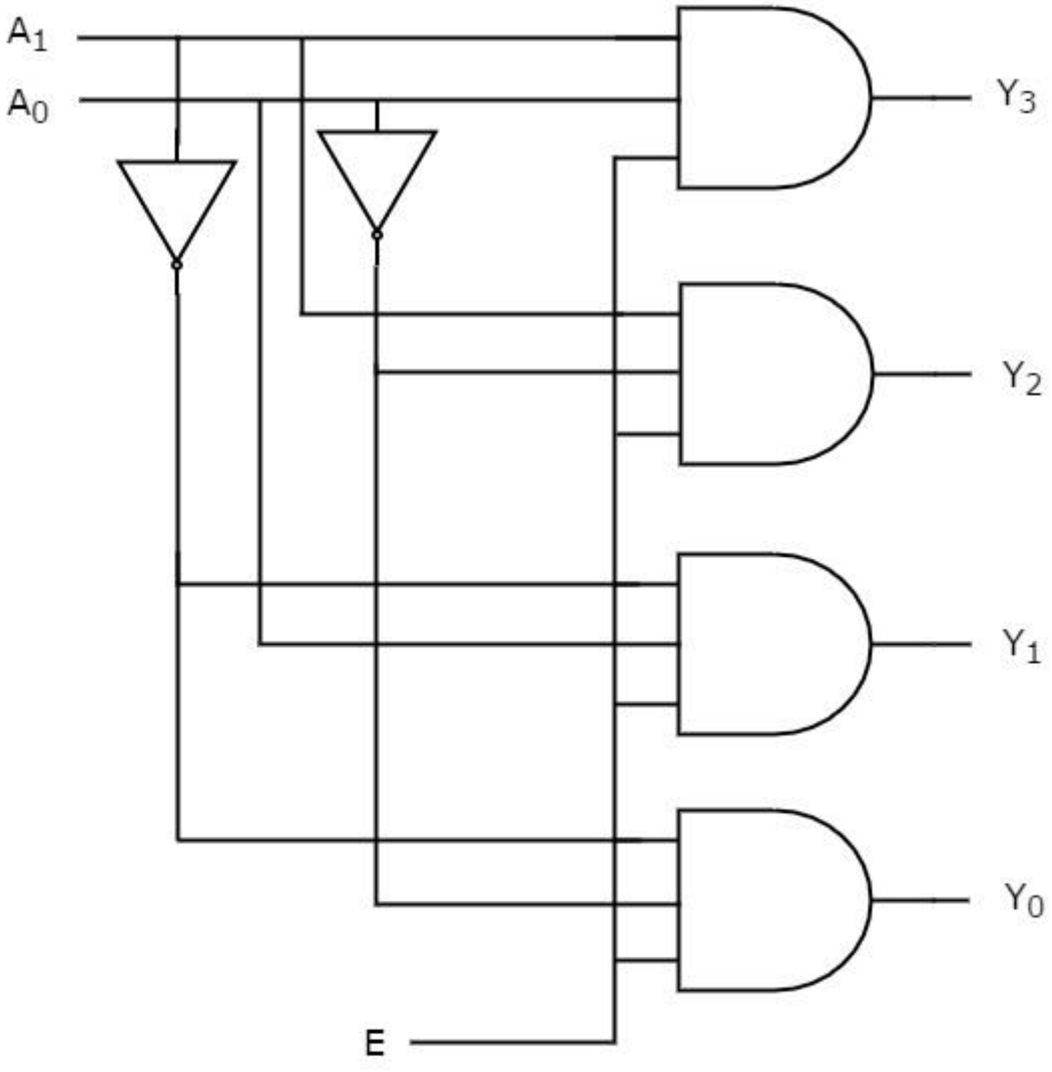
Y3=E.A1.A0Y3=E.A1.A0

Y2=E.A1.A0′Y2=E.A1.A0′

Y1=E.A1′.A0Y1=E.A1′.A0

Y0=E.A1′.A0′Y0=E.A1′.A0′

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



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

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

Implementation of Higher-order Decoders

Now, let us implement the following two higher-order decoders using lower-order decoders.

- 3 to 8 decoder
- 4 to 16 decoder

3 to 8 Decoder

In this section, let us implement **3 to 8 decoder using 2 to 4 decoders**. We know that 2 to 4 Decoder has two inputs, A_1 & A_0 and four outputs, Y_3 to Y_0. Whereas, 3 to 8 Decoder has three inputs A_2, A_1 & A_0 and eight outputs, Y_7 to Y_0.

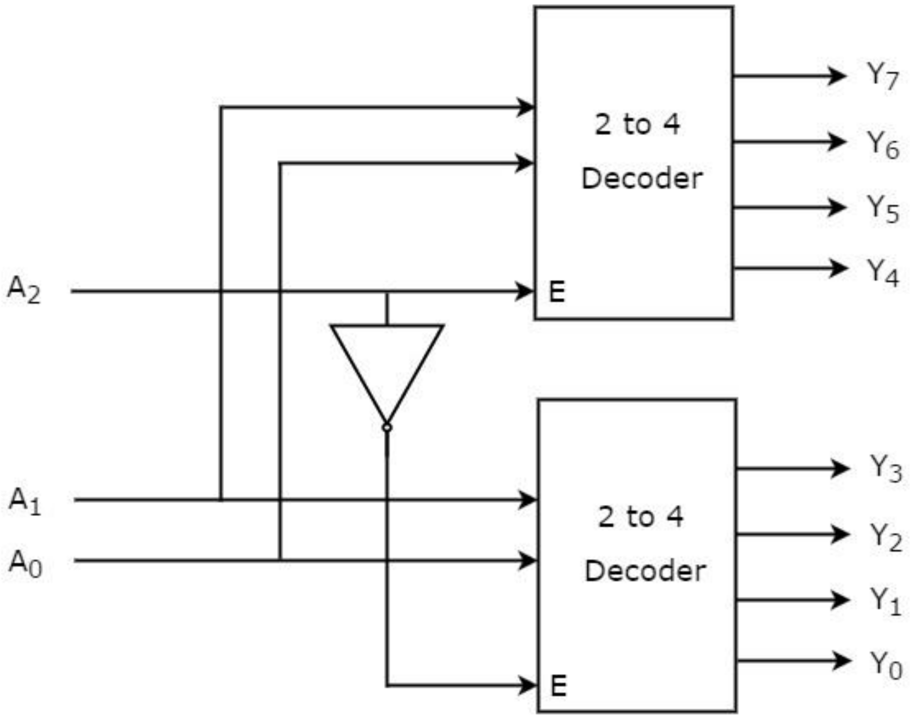
We can find the number of lower order decoders required for implementing higher order decoder using the following formula.

Requirednumberoflowerorderdecoders=m2m1Requirednumberoflowerorderdecoders =m2m1

Where,

m1m1 is the number of outputs of lower order decoder.
m2m2 is the number of outputs of higher order decoder.
Here, m1m1 = 4 and m2m2 = 8. Substitute, these two values in the above formula.
Requirednumberof2to4decoders=84=2Requirednumberof2to4decoders=84=2

Therefore, we require two 2 to 4 decoders for implementing one 3 to 8 decoder. The **block diagram** of 3 to 8 decoder using 2 to 4 decoders is shown in the following figure.



The parallel inputs A_1 & A_0 are applied to each 2 to 4 decoder. The complement of input A_2 is connected to Enable, E of lower 2 to 4 decoder in order to get the outputs, Y_3 to Y_0. These are the **lower four min terms**. The input, A_2 is directly connected to Enable, E of upper 2 to 4 decoder in order to get the outputs, Y_7 to Y_4. These are the **higher four min terms**.

4 to 16 Decoder

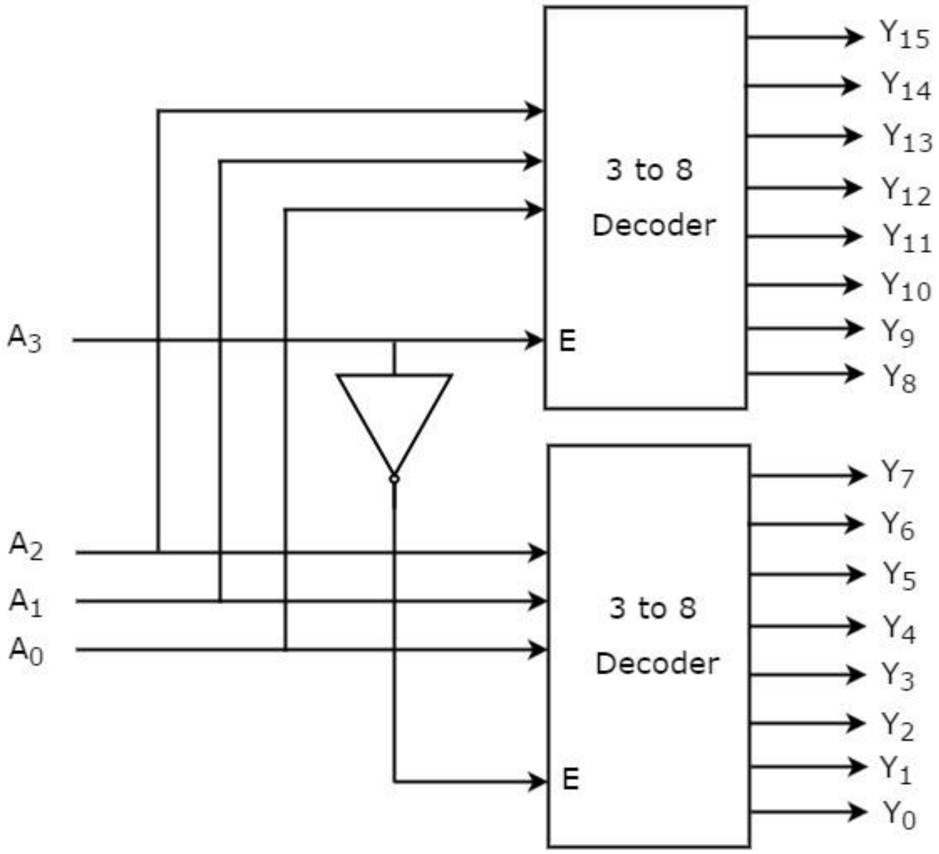
In this section, let us implement **4 to 16 decoder using 3 to 8 decoders**. We know that 3 to 8 Decoder has three inputs A_2, A_1 & A_0 and eight outputs, Y_7 to Y_0. Whereas, 4 to 16 Decoder has four inputs A_3, A_2, A_1 & A_0 and sixteen outputs, Y_{15} to Y_0

We know the following formula for finding the number of lower order decoders required.

Requirednumberoflowerorderdecoders=m2m1Requirednumberoflowerorderdecoders =m2m1

Substitute, m1m1 = 8 and m2m2 = 16 in the above formula.
Requirednumberof3to8decoders=168=2Requirednumberof3to8decoders=168=2

Therefore, we require two 3 to 8 decoders for implementing one 4 to 16 decoder. The **block diagram** of 4 to 16 decoder using 3 to 8 decoders is shown in the following figure.



The parallel inputs A_2, A_1 & A_0 are applied to each 3 to 8 decoder. The complement of input, A3 is connected to Enable, E of lower 3 to 8 decoder in order to get the outputs, Y_7 to Y_0. These are the **lower eight min terms**. The input, A_3 is directly connected to Enable, E of upper 3 to 8 decoder in order to get the outputs, Y_{15} to Y_8. These are the **higher eight min terms**.

latches and flip flop:

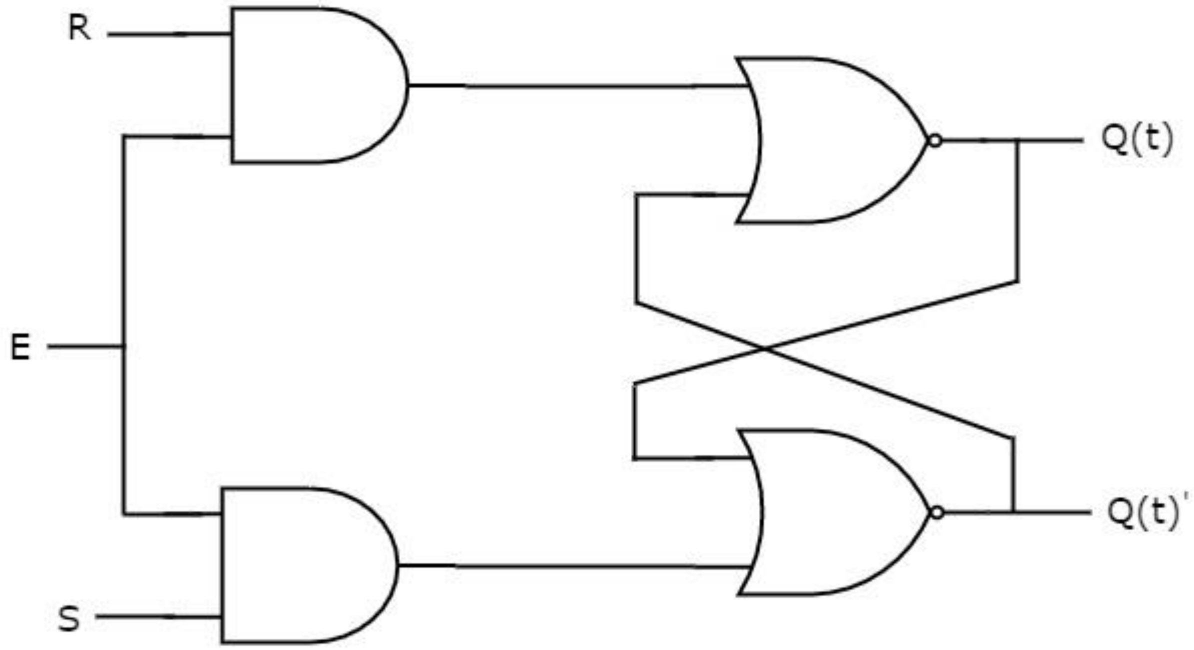
There are two types of memory elements based on the type of triggering that is suitable to operate it.

- Latches
- Flip-flops

Latches operate with enable signal, which is **level sensitive**. Whereas, flip-flops are edge sensitive. We will discuss about flip-flops in next chapter. Now, let us discuss about SR Latch & D Latch one by one.

SR Latch

SR Latch is also called as **Set Reset Latch**. This latch affects the outputs as long as the enable, E is maintained at ‘1’. The **circuit diagram** of SR Latch is shown in the following figure.



This circuit has two inputs S & R and two outputs Qtt & Qtt'. The **upper NOR gate** has two inputs R & complement of present state, Qtt' and produces next state, Qt+1t+1 when enable, E is '1'.
Similarly, the **lower NOR gate** has two inputs S & present state, Qtt and produces complement of next state, Qt+1t+1' when enable, E is '1'.

We know that a **2-input NOR gate** produces an output, which is the complement of another input when one of the input is '0'. Similarly, it produces '0' output, when one of the input is '1'.

- If S = 1, then next state Qt+1t+1 will be equal to '1' irrespective of present state, Qtt values.
- If R = 1, then next state Qt+1t+1 will be equal to '0' irrespective of present state, Qtt values.

At any time, only of those two inputs should be '1'. If both inputs are '1', then the next state Qt+1t+1 value is undefined.

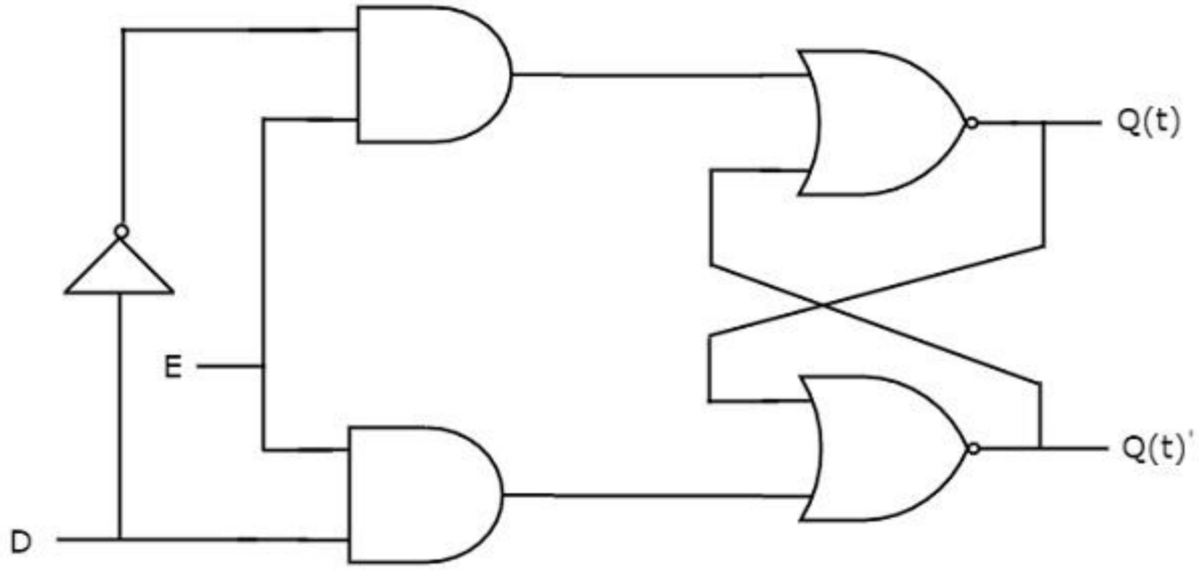
The following table shows the **state table** of SR latch.

S	**R**	**Q**t+1t+1
0	0	Qtt
0	1	0
1	0	1
1	1	-

Therefore, SR Latch performs three types of functions such as Hold, Set & Reset based on the input conditions.

D Latch

There is one drawback of SR Latch. That is the next state value can't be predicted when both the inputs S & R are one. So, we can overcome this difficulty by D Latch. It is also called as Data Latch. The **circuit diagram** of D Latch is shown in the following figure.



This circuit has single input D and two outputs Qtt & Qtt'. D Latch is obtained from SR Latch by placing an inverter between S amp;& R inputs and connect D input to S. That means we eliminated the combinations of S & R are of same value.

- If D = 0 → S = 0 & R = 1, then next state Qt+1t+1 will be equal to '0' irrespective of present state, Qtt values. This is corresponding to the second row of SR Latch state table.
- If D = 1 → S = 1 & R = 0, then next state Qt+1t+1 will be equal to '1' irrespective of present state, Qtt values. This is corresponding to the third row of SR Latch state table.

The following table shows the **state table** of D latch.

D	Qt+1t+1
0	0
1	1

Therefore, D Latch Hold the information that is available on data input, D. That means the output of D Latch is sensitive to the changes in the input, D as long as the enable is High.

In this chapter, we implemented various Latches by providing the cross coupling between NOR gates. Similarly, you can implement these Latches using NAND gates.

Flip-flop:

We can implement flip-flops in two methods.

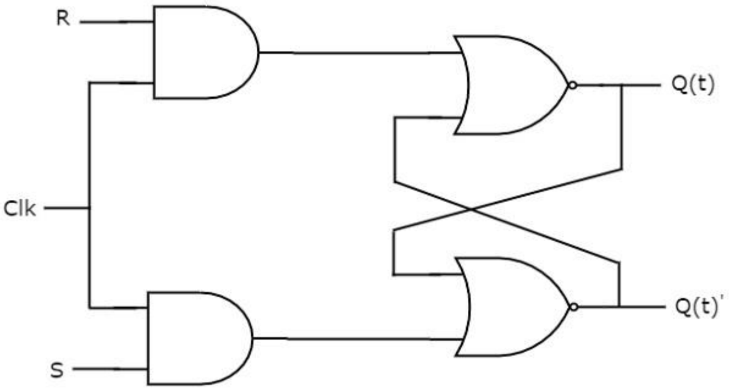
In first method, **cascade two latches** in such a way that the first latch is enabled for every positive clock pulse and second latch is enabled for every negative clock pulse. So that the combination of these two latches become a flip-flop.

In second method, we can directly implement the flip-flop, which is edge sensitive. In this chapter, let us discuss the following **flip-flops** using second method.

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

SR Flip-Flop

SR flip-flop operates with only positive clock transitions or negative clock transitions. Whereas, SR latch operates with enable signal. The **circuit diagram** of SR flip-flop is shown in the following figure.



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

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

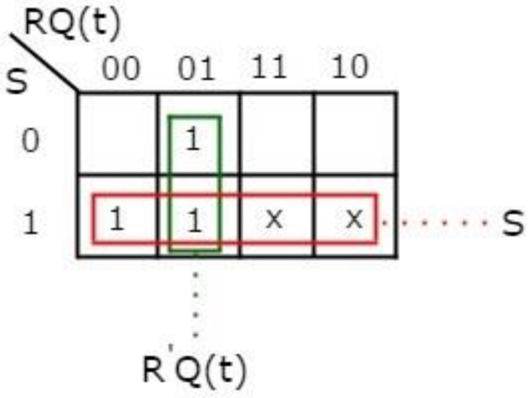
S	**R**	**Q**t+1t+1
0	0	Qtt
0	1	0
1	0	1

1	1	-

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

Present Inputs		**Present State**	**Next State**
S	**R**	**Q**tt	**Q**t+1t+1
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	1
1	1	0	x
1	1	1	x

By using three variable K-Map, we can get the simplified expression for next state, Qt+1t+1. The **three varia K-Map** for next state, Qt+1t+1 is shown in the following figure.

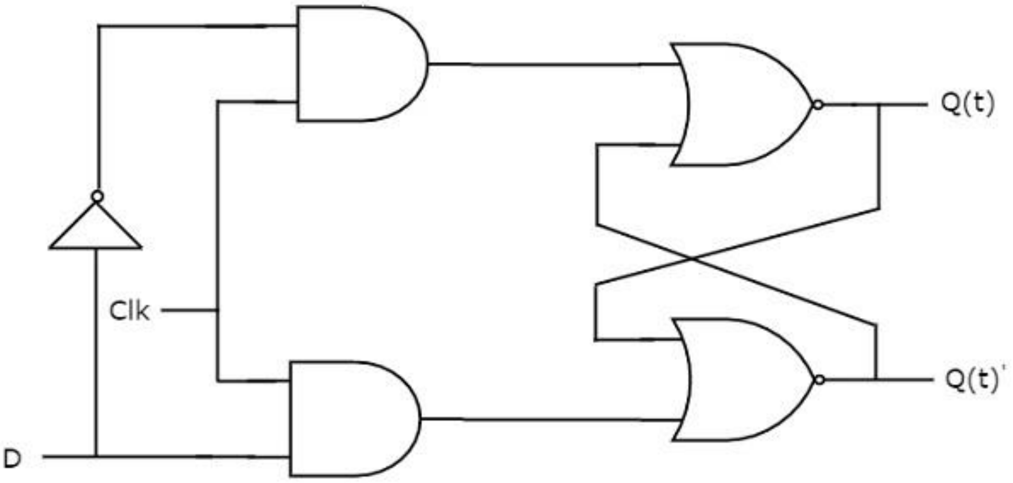


The maximum possible groupings of adjacent ones are already shown in the figure. Therefore, the **simplified expression** for next state Qt+1t+1 is

$$Q(t+1)=S+R'Q(t)Q(t+1)=S+R'Q(t)$$

D Flip-Flop

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



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

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

D	Qt + 1t + 1
0	0
1	1

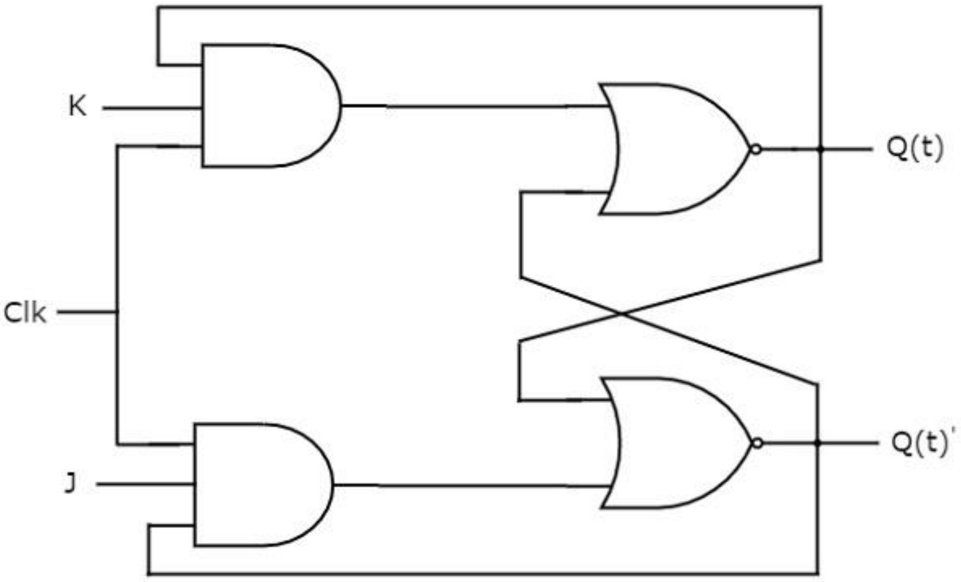
Therefore, D flip-flop always Hold the information, which is available on data input, D of earlier positive transition of clock signal. From the above state table, we can directly write the next state equation as

$$Qt+1t+1 = D$$

Next state of D flip-flop is always equal to data input, D for every positive transition of the clock signal. Hence, D flip-flops can be used in registers, **shift registers** and some of the counters.

JK Flip-Flop

JK flip-flop is the modified version of SR flip-flop. It operates with only positive clock transitions or negative clock transitions. The **circuit diagram** of JK flip-flop is shown in the following figure.



This circuit has two inputs J & K and two outputs Qtt & Qtt'. The operation of JK flip-flop is similar to SR flip-flop. Here, we considered the inputs of SR flip-flop as **S = J Q**tt' and **R = KQ**tt in order to utilize the modified SR flip-flop for 4 combinations of inputs.

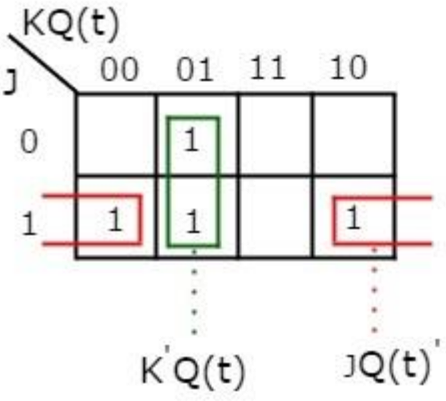
The following table shows the **state table** of JK flip-flop.

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

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

Present Inputs		Present State	Next State
J	**K**	**Q**tt	**Q**t+1t+1
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	0

By using three variable K-Map, we can get the simplified expression for next state, Qt+1t+1. **Three variable K-Map** for next state, Qt+1t+1 is shown in the following figure.

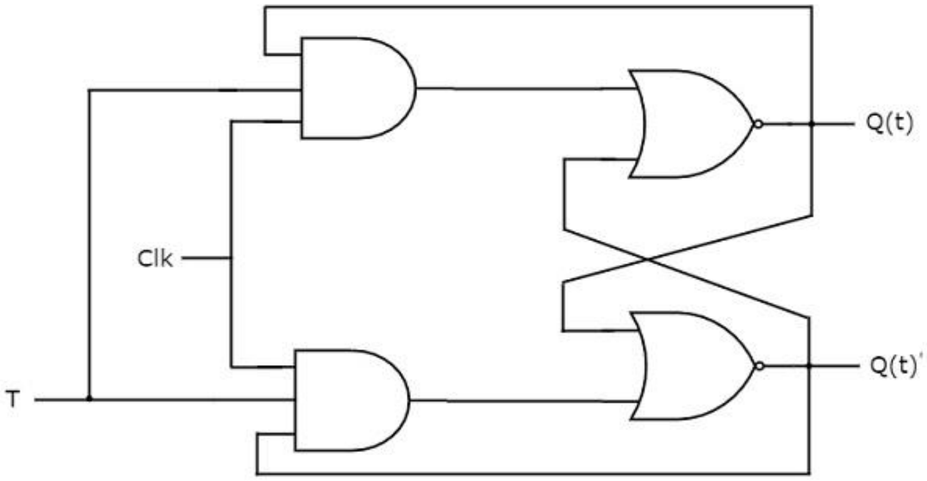


The maximum possible groupings of adjacent ones are already shown in the figure. Therefore, the **simplified expression** for next state Qt+1t+1 is

$$Q(t+1)=JQ(t)'+K'Q(t)Q(t+1)=JQ(t)'+K'Q(t)$$

T Flip-Flop

T flip-flop is the simplified version of JK flip-flop. It is obtained by connecting the same input 'T' to both inputs of JK flip-flop. It operates with only positive clock transitions or negative clock transitions. The **circuit diagram** of T flip-flop is shown in the following figure.



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

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

D	**Q**t+1t+1
0	Qtt
1	Qtt'

Here, Qtt & Qt+1t+1 are present state & next state respectively. So, T flip-flop can be used for one of these two functions such as Hold, & Complement of present state based on the input conditions, when positive transition of clock signal is applied. The following table shows the **characteristic table** of T flip-flop.

Inputs	**Present State**	**Next State**
T	**Q**tt	**Q**t+1t+1
0	0	0

0	1	1
1	0	1
1	1	0

From the above characteristic table, we can directly write the **next state equation** as

$$Q(t+1)=T'Q(t)+TQ(t)'Q(t+1)=T'Q(t)+TQ(t)'$$

$$\Rightarrow Q(t+1)=T\oplus Q(t)\Rightarrow Q(t+1)=T\oplus Q(t)$$

The output of T flip-flop always toggles for every positive transition of the clock signal, when input T remains at logic High 11. Hence, T flip-flop can be used in **counters.**

In this chapter, we implemented various flip-flops by providing the cross coupling between NOR gates. Similarly, you can implement these flip-flops by using NAND gates.

UNIT-5

Counters: Introduction, Asynchronous counter, Terms related to counters, IC7493 (4-bit binary counter), Synchronous counter, Bushing, Type TDesign, Type JK Design, Preset table counter, IC 7490, IC 7492,Synchronous counter ICs, Analysis of counter circuits.

Shift Register: Introduction, parallel and shift registers, serial shifting, serial–in serial–out, serial–in parallel–out , parallel–in parallel–out, Ring counter, Johnson counter, Applications of shift registers, Pseudo-random binary sequence generator, IC7495, Seven Segment displays, analysis of shift counters.

Counter:

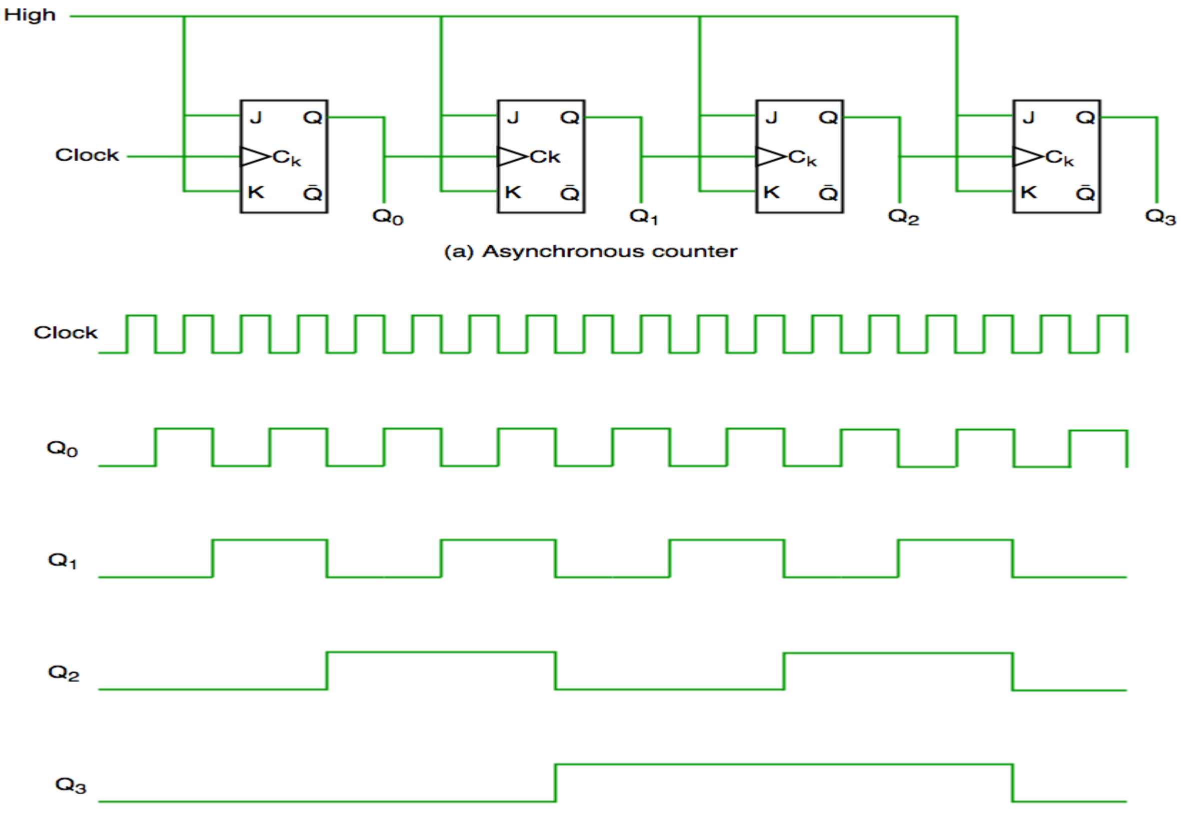
A **Counter** is a device which stores (and sometimes displays) the number of times a particular event or process has occurred, often in relationship to a clock signal. Counters are used in digital electronics for counting purpose, they can count specific event happening in the circuit. For example, in UP counter a counter increases count for every rising edge of clock. Not only counting, a counter can follow the certain sequence based on our design like any random sequence 0,1,3,2… .They can also be designed with the help of flip flops.

Counters are broadly divided into two categories

1. Asynchronous counter
2. Synchronous counter

1. **Asynchronous Counter**
In asynchronous counter we don't use universal clock, only first flip flop is driven by main clock and the clock input of rest of the following flip flop is driven by output of

previous flip flops. We can understand it by following diagram-



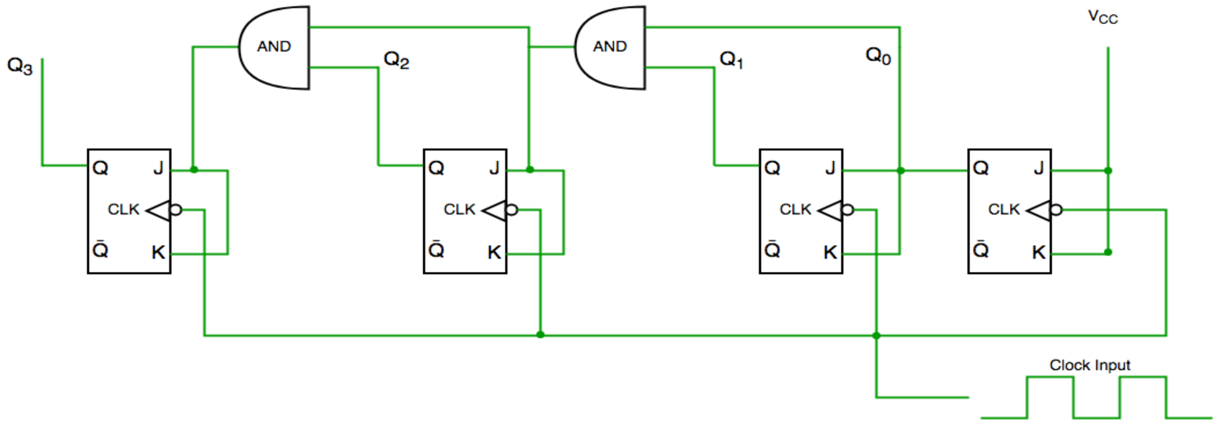
(a) Asynchronous counter

(b) Timing Diagram

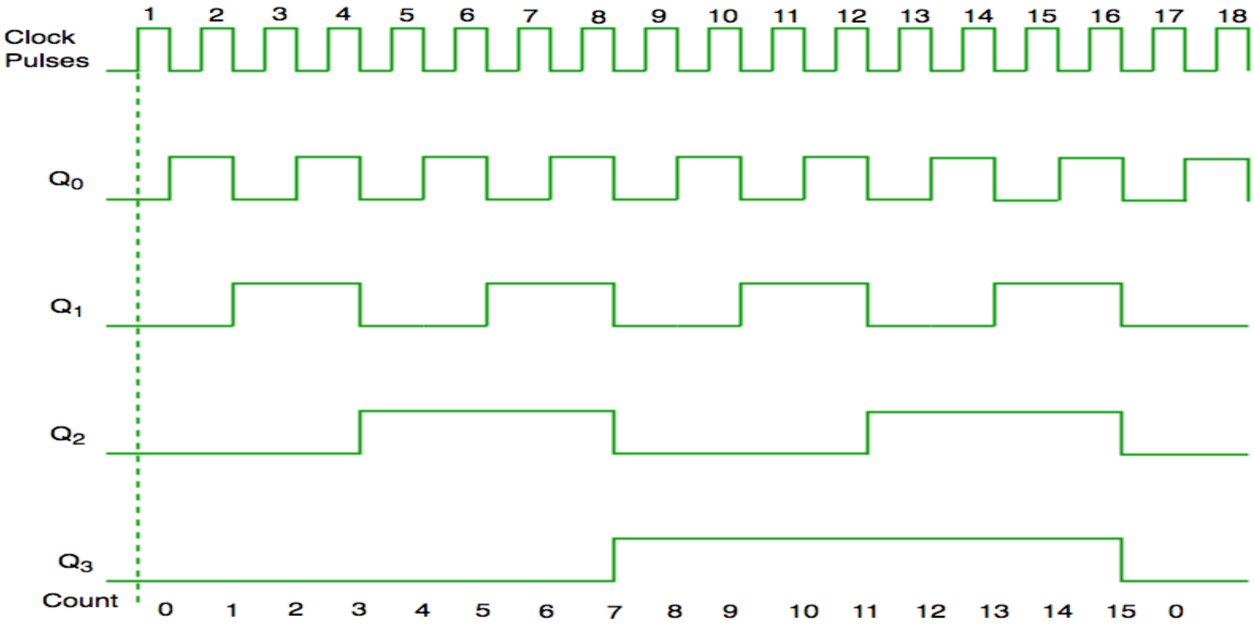
It is evident from timing diagram that Q0 is changing as soon as the rising edge of clock pulse is encountered, Q1 is changing when rising edge of Q0 is encountered(because Q0 is like clock pulse for second flip flop) and so on. In this way ripples are generated through Q0,Q1,Q2,Q3 hence it is also called **RIPPLE counter.**

2. **Synchronous Counter**

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



Synchronous counter circuit



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 = 2^n.

Type of modulus

- 2-bit up or down (MOD-4)
- 3-bit up or down (MOD-8)
- 4-bit up or down (MOD-16)

Application of counters

- Frequency counters
- Digital clock
- Time measurement
- A to D converter
- Frequency divider circuits
- Digital triangular wave generator.

Shift register:

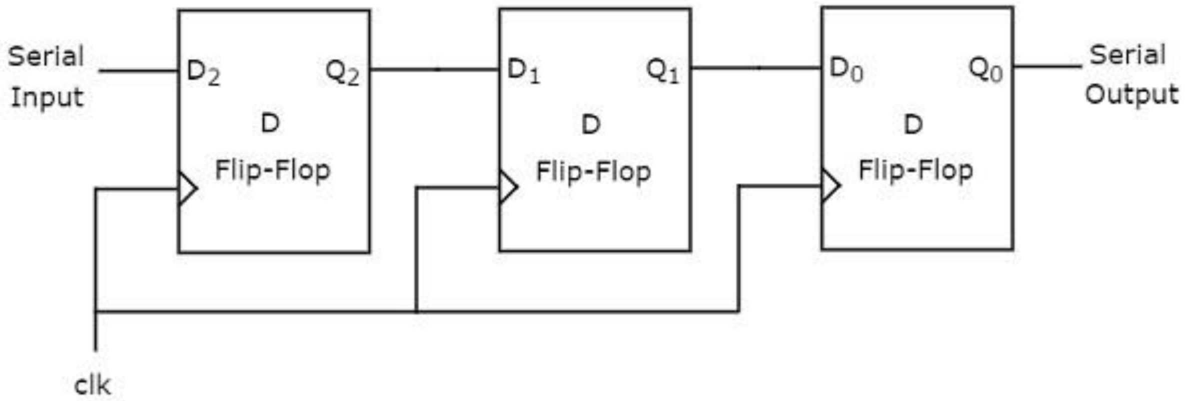
We know that one flip-flop can store one-bit of information. In order to store multiple bits of information, we require multiple flip-flops. The group of flip-flops, which are used to hold store the binary data is known as **register**.

If the register is capable of shifting bits either towards right hand side or towards left hand side is known as **shift register**. An 'N' bit shift register contains 'N' flip-flops. Following are the four types of shift registers based on applying inputs and accessing of outputs.

- Serial In − Serial Out shift register
- Serial In − Parallel Out shift register
- Parallel In − Serial Out shift register
- Parallel In − Parallel Out shift register

Serial In − Serial Out SISO Shift Register

The shift register, which allows serial input and produces serial output is known as Serial In – Serial Out SISOSISO shift register. The **block diagram** of 3-bit SISO shift register is shown in the following figure.



This block diagram consists of three D flip-flops, which are **cascaded**. That means, output of one D flip-flop is connected as the input of next D flip-flop. All these flip-flops are synchronous with each other since, the same clock signal is applied to each one.

In this shift register, we can send the bits serially from the input of left most D flip-flop. Hence, this input is also called as **serial input**. For every positive edge triggering of clock signal, the data shifts from one stage to the next. So, we can receive the bits serially from the output of right most D flip-flop. Hence, this output is also called as **serial output**.

Example

Let us see the working of 3-bit SISO shift register by sending the binary information **"011"** from LSB to MSB serially at the input.

Assume, initial status of the D flip-flops from leftmost to rightmost is Q2Q1Q0=000Q2Q1Q0=000. We can understand the **working of 3-bit SISO shift register** from the following table.

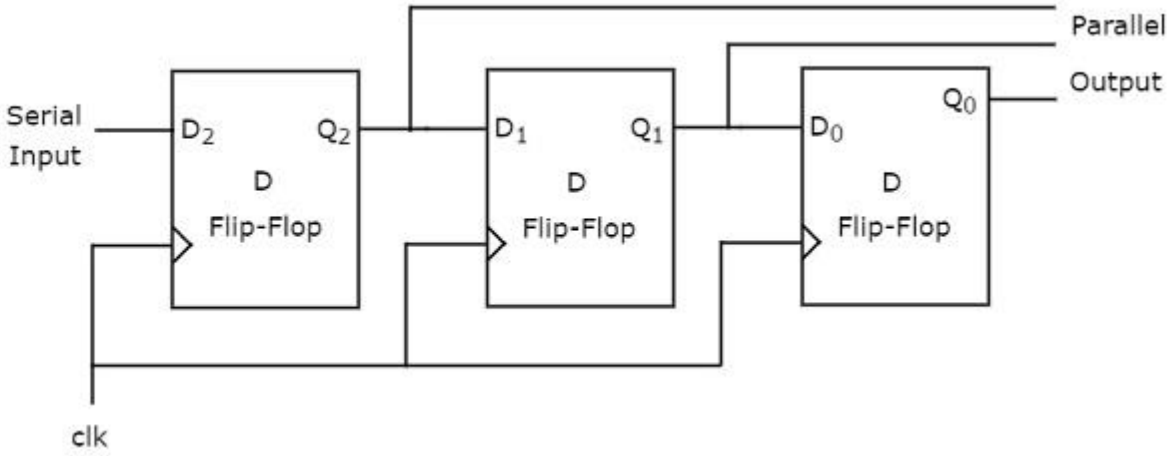
No of positive edge of Clock	Serial Input	Q_2	Q_1	Q_0
0	-	0	0	0
1	1LSBLSB	1	0	0
2	1	1	1	0
3	0MSBMSB	0	1	1LSBLSB
4	-	-	0	1
5	-	-	-	0MSBMSB

The initial status of the D flip-flops in the absence of clock signal is Q2Q1Q0=000Q2Q1Q0=000. Here, the serial output is coming from Q0Q0. So, the LSB 11 is received at 3rd positive edge of clock and the MSB 00 is received at 5th positive edge of clock.

Therefore, the 3-bit SISO shift register requires five clock pulses in order to produce the valid output. Similarly, the **N-bit SISO shift register** requires **2N-1** clock pulses in order to shift 'N' bit information.

Serial In - Parallel Out SIPO Shift Register

The shift register, which allows serial input and produces parallel output is known as Serial In – Parallel Out SIPOSIPO shift register. The **block diagram** of 3-bit SIPO shift register is shown in the following figure.



This circuit consists of three D flip-flops, which are cascaded. That means, output of one D flip-flop is connected as the input of next D flip-flop. All these flip-flops are synchronous with each other since, the same clock signal is applied to each one.

In this shift register, we can send the bits serially from the input of left most D flip-flop. Hence, this input is also called as **serial input**. For every positive edge triggering of clock signal, the data shifts from one stage to the next. In this case, we can access the outputs of each D flip-flop in parallel. So, we will get **parallel outputs** from this shift register.

Example

Let us see the working of 3-bit SIPO shift register by sending the binary information **"011"** from LSB to MSB serially at the input.

Assume, initial status of the D flip-flops from leftmost to rightmost is Q2Q1Q0=000Q2Q1Q0=000. Here, Q2Q2 & Q0Q0 are MSB & LSB respectively. We can understand the **working of 3-bit SIPO shift register** from the following table.

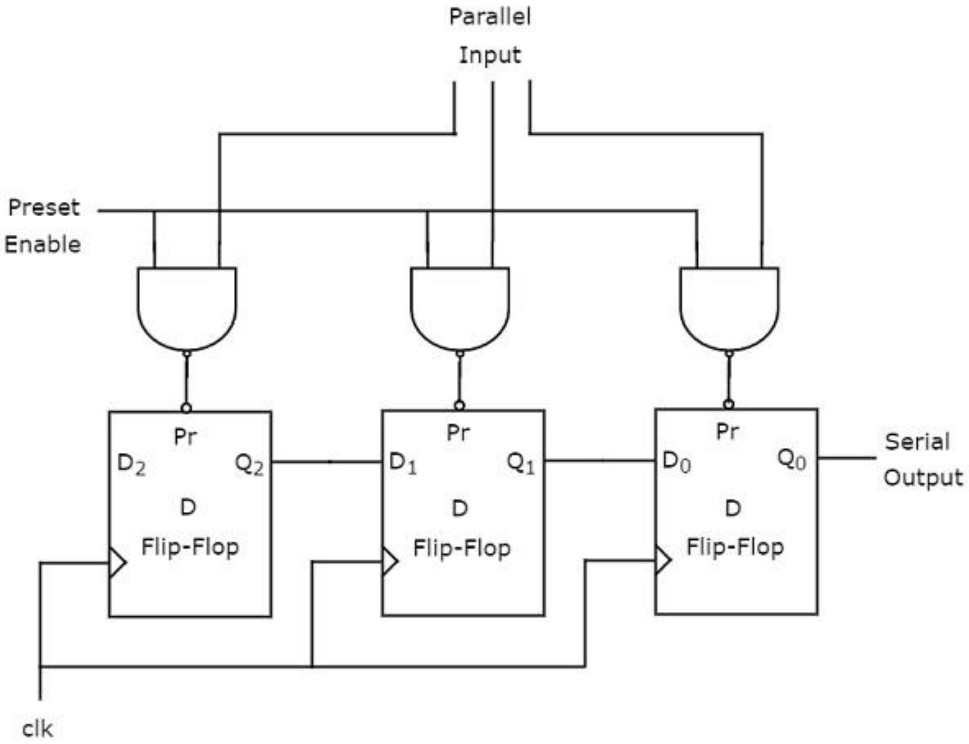
No of positive edge of Clock	**Serial Input**	**Q_2**MSBMSB	**Q_1**	**Q_0**LSBLSB
0	-	0	0	0
1	1LSBLSB	1	0	0
2	1	1	1	0
3	0MSBMSB	0	1	1

The initial status of the D flip-flops in the absence of clock signal is Q2Q1Q0=000Q2Q1Q0=000. The binary information **“011”** is obtained in parallel at the outputs of D flip-flops for third positive edge of clock.

So, the 3-bit SIPO shift register requires three clock pulses in order to produce the valid output. Similarly, the **N-bit SIPO shift register** requires **N** clock pulses in order to shift ‘N’ bit information.

Parallel In − Serial Out PISO Shift Register

The shift register, which allows parallel input and produces serial output is known as Parallel In − Serial Out PISOPISO shift register. The **block diagram** of 3-bit PISO shift register is shown in the following figure.



This circuit consists of three D flip-flops, which are cascaded. That means, output of one D flip-flop is connected as the input of next D flip-flop. All these flip-flops are synchronous with each other since, the same clock signal is applied to each one.

In this shift register, we can apply the **parallel inputs** to each D flip-flop by making Preset Enable to 1. For every positive edge triggering of clock signal, the data shifts from one stage to the next. So, we will get the **serial output** from the right most D flip-flop.

Example

Let us see the working of 3-bit PISO shift register by applying the binary information **“011”** in parallel through preset inputs.

Since the preset inputs are applied before positive edge of Clock, the initial status of the D flip-flops from leftmost to rightmost will be Q2Q1Q0=011Q2Q1Q0=011. We can understand the **working of 3-bit PISO shift register** from the following table.

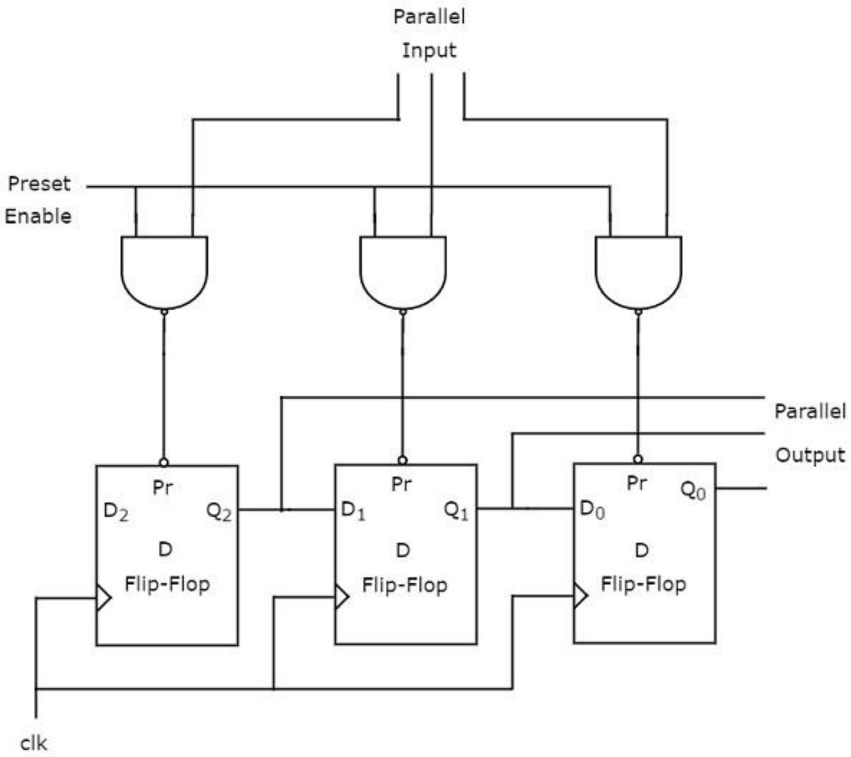
No of positive edge of Clock	Q_2	Q_1	Q_0
0	0	1	1LSBLSB
1	-	0	1
2	-	-	0LSBLSB

Here, the serial output is coming from Q0Q0. So, the LSB 11 is received before applying positive edge of clock and the MSB 00 is received at 2nd positive edge of clock.

Therefore, the 3-bit PISO shift register requires two clock pulses in order to produce the valid output. Similarly, the **N-bit PISO shift register** requires **N-1** clock pulses in order to shift 'N' bit information.

Parallel In - Parallel Out PIPOPIPO Shift Register

The shift register, which allows parallel input and produces parallel output is known as Parallel In – Parallel Out PIPOPIPO shift register. The **block diagram** of 3-bit PIPO shift register is shown in the following figure.



This circuit consists of three D flip-flops, which are cascaded. That means, output of one D flip-flop is connected as the input of next D flip-flop. All these flip-flops are synchronous with each other since, the same clock signal is applied to each one.

In this shift register, we can apply the **parallel inputs** to each D flip-flop by making Preset Enable to 1. We can apply the parallel inputs through preset or clear. These two are asynchronous inputs. That means, the flip-flops produce the corresponding outputs, based on the values of asynchronous inputs. In this case, the effect of outputs is independent of clock transition. So, we will get the **parallel outputs** from each D flip-flop.

Example

Let us see the working of 3-bit PIPO shift register by applying the binary information **"011"** in parallel through preset inputs.

Since the preset inputs are applied before positive edge of Clock, the initial status of the D flip-flops from leftmost to rightmost will be Q2Q1Q0=011Q2Q1Q0=011. So, the binary information **"011"** is obtained in parallel at the outputs of D flip-flops before applying positive edge of clock.

Therefore, the 3-bit PIPO shift register requires zero clock pulses in order to produce the valid output. Similarly, the **N-bit PIPO shift register** doesn't require any clock pulse in order to shift 'N' bit information.