Unit-I

Computer Codes, Gates and Boolean Algebra

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

Advantages of Binary Code

Following is the list of advantages that binary code offers.

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

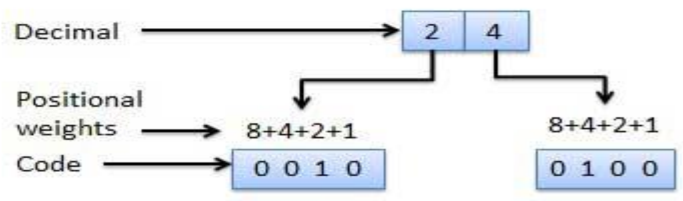
Classification of binary codes

The codes are broadly categorized into following four categories.

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

Weighted Codes

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



Non-Weighted Codes

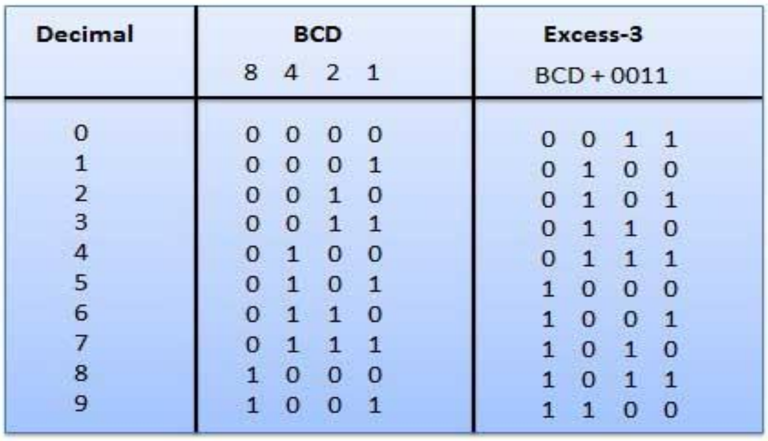
In this type of binary codes, the positional weights are not assigned. The examples of non-weighted codes are Excess-3 code and Gray code.

Excess-3 code

The Excess-3 code is also called as XS-3 code. It is non-weighted code used to express decimal numbers. The Excess-3 code words are derived from the 8421 BCD code words adding (0011)2 or (3)10 to each code word in 8421. The excess-3 codes are obtained as follows −

$$\text{Decimal Number} \longrightarrow \text{8421 BCD} \xrightarrow[\text{0011}]{\text{Add}} \text{Excess-3}$$

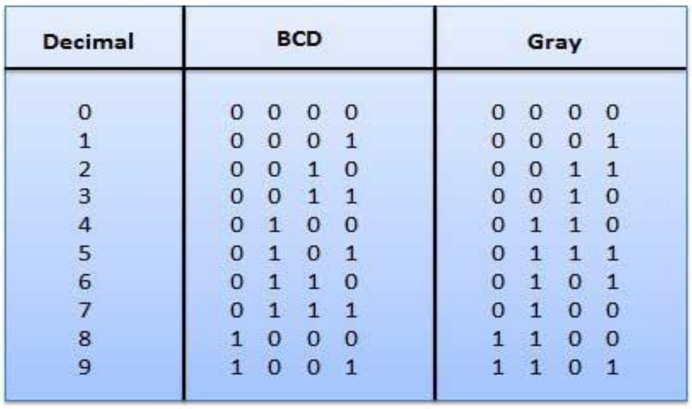
Example



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

Gray Code

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



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

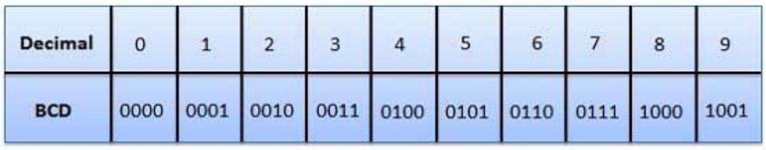
Application of Gray code

Gray code is popularly used in the shaft position encoders.

- A shaft position encoder produces a code word which represents the angular position of the shaft.

Binary Coded Decimal (BCD) code

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



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

Advantages of BCD Codes

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

Disadvantages of BCD Codes

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

Alphanumeric codes

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

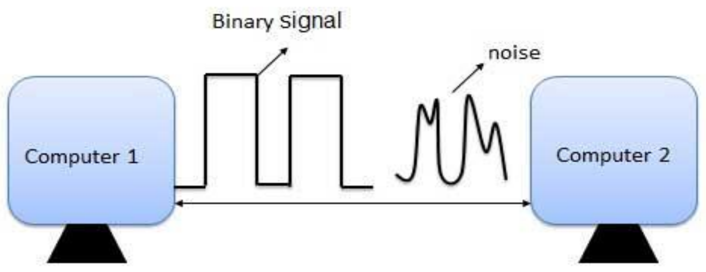
The alphanumeric codes are the codes that represent numbers and alphabetic characters. Mostly such codes also represent other characters such as symbol and various instructions necessary for conveying information. An alphanumeric code should at least represent 10 digits and 26 letters of alphabet i.e.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

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

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

What is Error?

Error is a condition when the output information does not match with the input information. During transmission, digital signals suffer from noise that can introduce errors in the binary bits travelling from one system to other. That means a 0 bit may change to 1 or a 1 bit may change to 0.



Error-Detecting codes

Whenever a message is transmitted, it may get scrambled by noise or data may get corrupted. To avoid this, we use error-detecting codes which are additional data added to a given digital message to help us detect if an error occurred during transmission of the message. A simple example of error-detecting code is **parity check**.

Error-Correcting codes

Along with error-detecting code, we can also pass some data to figure out the original message from the corrupt message that we received. This type of code is called an error-correcting code. Error-correcting codes also deploy the same strategy as error-detecting codes but additionally, such codes also detect the exact location of the corrupt bit.

In error-correcting codes, parity check has a simple way to detect errors along with a sophisticated mechanism to determine the corrupt bit location. Once the corrupt bit is located, its value is reverted (from 0 to 1 or 1 to 0) to get the original message.

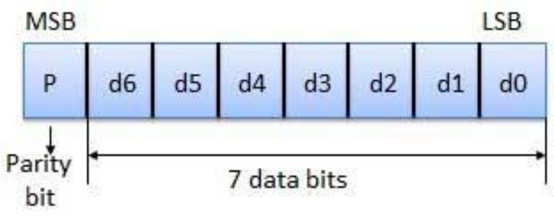
How to Detect and Correct Errors?

To detect and correct the errors, additional bits are added to the data bits at the time of transmission.

- The additional bits are called **parity bits**. They allow detection or correction of the errors.
- The data bits along with the parity bits form a **code word**.

Parity Checking of Error Detection

It is the simplest technique for detecting and correcting errors. The MSB of an 8-bits word is used as the parity bit and the remaining 7 bits are used as data or message bits. The parity of 8-bits transmitted word can be either even parity or odd parity.



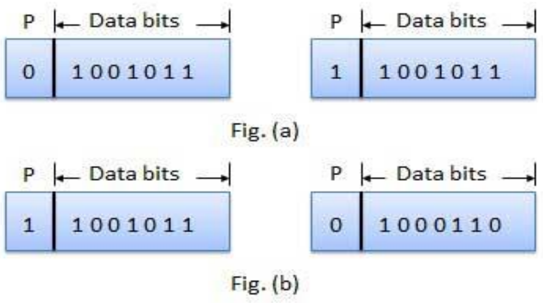
Even parity -- Even parity means the number of 1's in the given word including the parity bit should be even (2,4,6,....).

Odd parity -- Odd parity means the number of 1's in the given word including the parity bit should be odd (1,3,5,....).

Use of Parity Bit

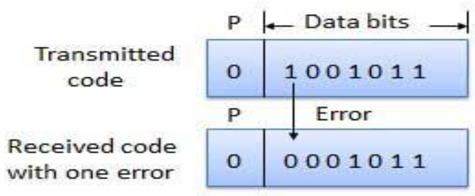
The parity bit can be set to 0 and 1 depending on the type of the parity required.

- For even parity, this bit is set to 1 or 0 such that the no. of "1 bits" in the entire word is even. Shown in fig. (a).
- For odd parity, this bit is set to 1 or 0 such that the no. of "1 bits" in the entire word is odd. Shown in fig. (b).



How Does Error Detection Take Place?

Parity checking at the receiver can detect the presence of an error if the parity of the receiver signal is different from the expected parity. That means, if it is known that the parity of the transmitted signal is always going to be "even" and if the received signal has an odd parity, then the receiver can conclude that the received signal is not correct. If an error is detected, then the receiver will ignore the received byte and request for retransmission of the same byte to the transmitter.



.

Cyclic Codes

The cyclic property of code words is that any cyclic-shift of a code word is also a code word. Cyclic codes follow this cyclic property.

For a linear code **C**, if every code word i.e., **C =** C_1, C2,......C_n C_1,C_2,......C_n from C has a cyclic right shift of components, it becomes a code word. This shift of right is equal to **n-1** cyclic left shifts. Hence, it is invariant under any shift. So, the linear code **C**, as it is invariant under any shift, can be called as a **cyclic code**.

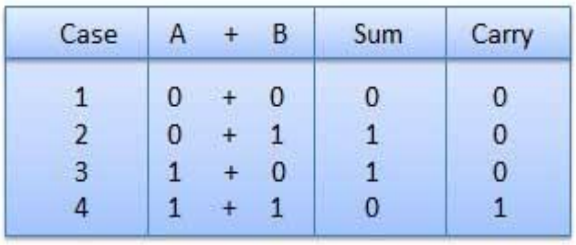
Cyclic codes are used for error correction. They are mainly used to correct double errors and burst errors.

Hence, these are a few error correcting codes, which are to be detected at the receiver. These codes prevent the errors from getting introduced and disturb the communication. They also prevent the signal from getting tapped by unwanted receivers.

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

Binary Addition

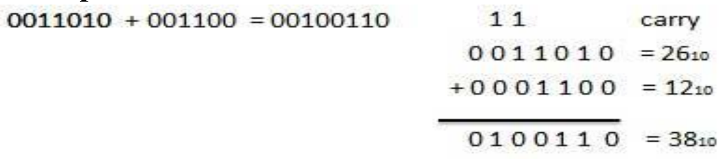
It is a key for binary subtraction, multiplication, division. There are four rules of binary addition.



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

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

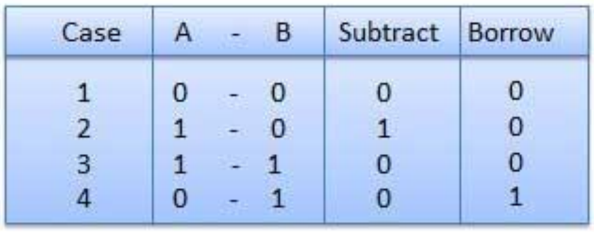
Example − Addition



```
0011010 + 001100 = 00100110          1 1           carry
                                    0 0 1 1 0 1 0   = 26₁₀
                                   +0 0 0 1 1 0 0   = 12₁₀
                                   ______________
                                    0 1 0 0 1 1 0   = 38₁₀
```

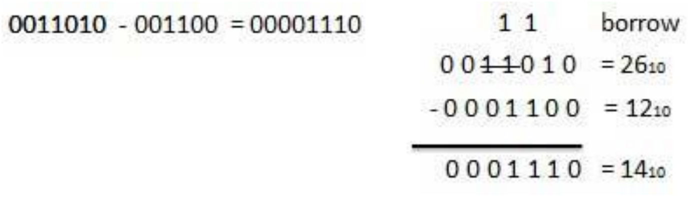
Binary Subtraction

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



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

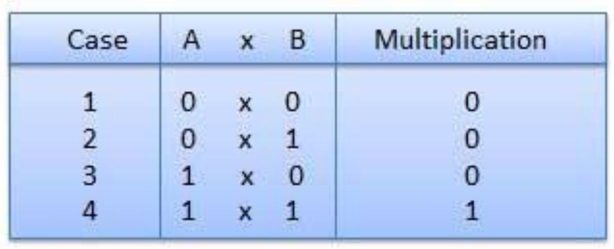
Example − Subtraction



```
0011010 - 001100 = 00001110

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

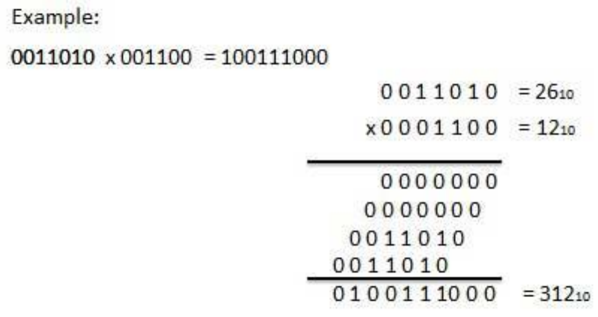
Binary Multiplication

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

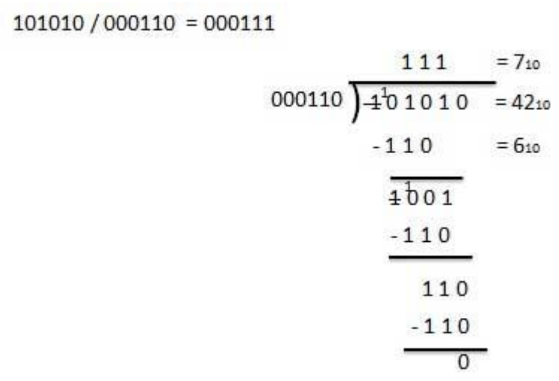


```
Example:

0011010 x 001100 = 100111000

           0 0 1 1 0 1 0   = 26₁₀
         x 0 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

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

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

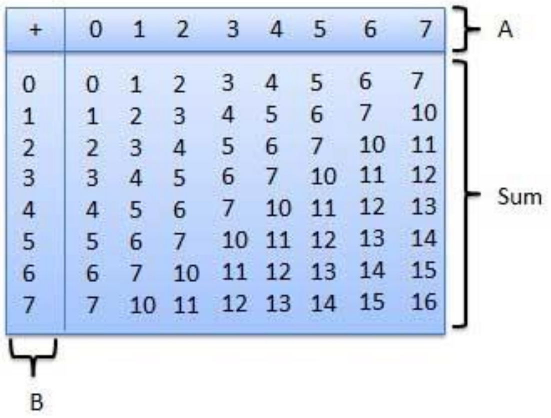
Calculating Decimal Equivalent −

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.

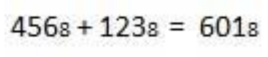


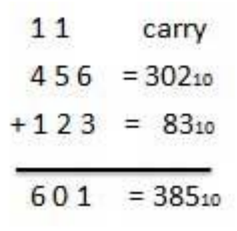
+	0	1	2	3	4	5	6	7
0	0	1	2	3	4	5	6	7
1	1	2	3	4	5	6	7	10
2	2	3	4	5	6	7	10	11
3	3	4	5	6	7	10	11	12
4	4	5	6	7	10	11	12	13
5	5	6	7	10	11	12	13	14
6	6	7	10	11	12	13	14	15
7	7	10	11	12	13	14	15	16

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

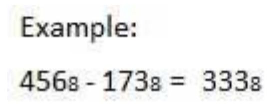

$456_8 + 123_8 = 601_8$


```
  1 1       carry
  4 5 6   = 302₁₀
+ 1 2 3   =  83₁₀
─────────
  6 0 1   = 385₁₀
```

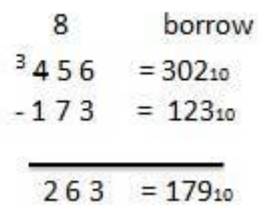
Octal Subtraction

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:

$456_8 - 173_8 = 333_8$


```
     8        borrow
  ³4 5 6   = 302₁₀
  - 1 7 3   = 123₁₀
─────────
    2 6 3   = 179₁₀
```

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 −

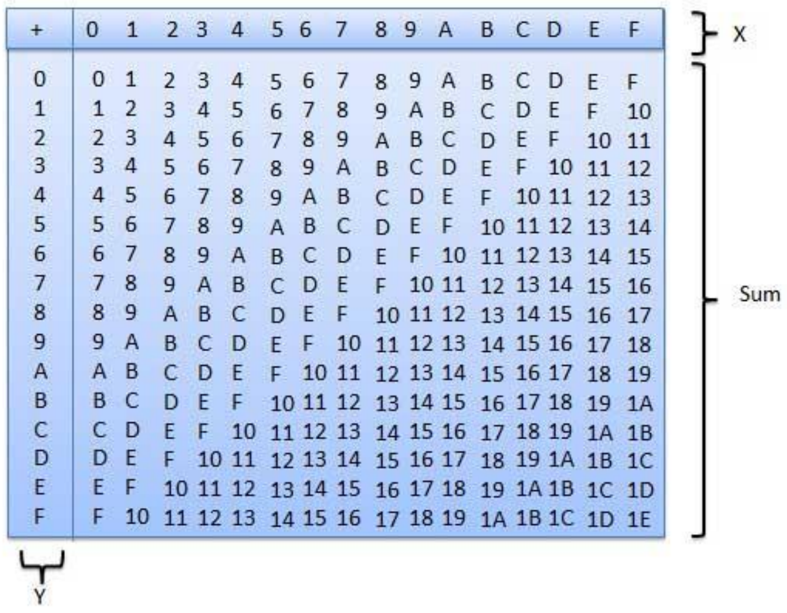
Step	Hexadecimal Number	Decimal Number
Step 1	$19FDE_{16}$	$((1 \times 16^4) + (9 \times 16^3) + (F \times 16^2) + (D \times 16^1) + (E \times 16^0))_{10}$

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

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

$$\begin{array}{rcl} 1 & & \text{carry} \\ 4\,A\,6 & = & 1190_{10} \\ +1\,B\,3 & = & 435_{10} \\ \hline 6\,5\,9 & = & 1625_{10} \end{array}$$

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

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

$$\begin{array}{rl} 16 & \text{borrow} \\ {}^{3}4\,A\,6 & = 1190_{10} \\ -1\,B\,3 & = 435_{10} \\ \hline 2\,F\,3 & = 755_{10} \end{array}$$

Unsigned and Signed Binary Numbers

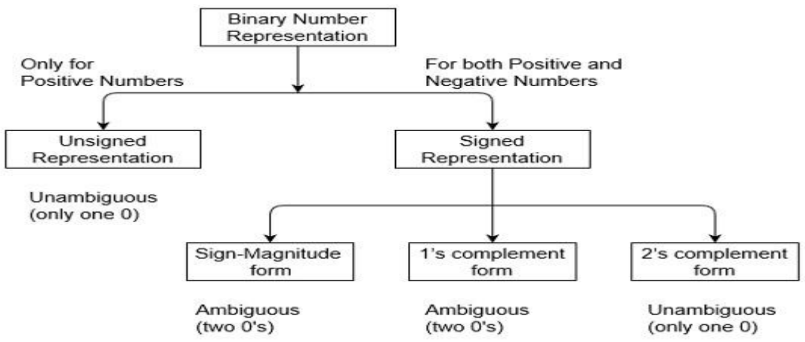
Variables such as integers can be represent in two ways, i.e., signed and unsigned. Signed numbers use sign flag or can be distinguish between negative values and positive values. Whereas unsigned numbers stored only positive numbers but not negative numbers.

Number representation techniques like: Binary, Octal, Decimal and Hexadecimal number representation techniques can represent numbers in both signed and unsigned ways. Binary Number System is one the type of Number Representation techniques. It is most popular and used in digital systems. Binary system is used for representing binary quantities which can be represented by any device that has only two operating states or possible conditions. For example, a switch has only two states: open or close.

In the Binary System, there are only two symbols or possible digit values, i.e., 0 and 1. Represented by any device that only 2 operating states or possible conditions. Binary numbers are indicated by the addition of either a *0b* prefix or a *2* suffix.

Representation of Binary Numbers:

Binary numbers can be represented in signed and unsigned way. Unsigned binary numbers do not have sign bit, whereas signed binary numbers uses signed bit as well or these can be distinguishable between positive and negative numbers. A signed binary is a specific data type of a signed variable.



1. Unsigned Numbers:

Unsigned numbers don’t have any sign, these can contain only magnitude of the number. So, representation of unsigned binary numbers are all positive numbers only. For example, representation of positive decimal numbers are positive by default. We always assume that there is a positive sign symbol in front of every number.

Representation of Unsigned Binary Numbers:

Since there is no sign bit in this unsigned binary number, so N bit binary number represent its magnitude only. Zero (0) is also unsigned number. This representation has only one zero (0), which is always positive. Every number in unsigned number representation has only one unique binary equivalent form, so this is unambiguous representation technique. The range of unsigned binary number is from 0 to (2^n-1).

Example-1: Represent decimal number 92 in unsigned binary number.

Simply convert it into Binary number, it contains only magnitude of the given number. = $(92)_{10}$

= $(1x2^6+0x2^5+1x2^4+1x2^3+1x2^2+0x2^1+0x2^0)_{10}$

= $(1011100)_2$

It’s 7 bit binary magnitude of the decimal number 92.

Example-2: Find range of 5 bit unsigned binary numbers. Also, find minimum and maximum value in this range.

Since, range of unsigned binary number is from 0 to (2^n-1). Therefore, range of 5 bit unsigned binary number is *from* 0 to (2^5-1) which is equal from minimum value 0 (i.e., 00000) to maximum value 31 (i.e., 11111).

2. Signed Numbers:

Signed numbers contain sign flag, this representation distinguish positive and negative numbers. This technique contains both sign bit and magnitude of a number. For example, in representation of negative decimal numbers, we need to put negative symbol in front of given decimal number.

Representation of Signed Binary Numbers:

There are three types of representations for signed binary numbers. Because of extra signed bit, binary number zero has two representation, either positive (0) or negative (1), so ambiguous representation. But 2’s complementation representation is unambiguous representation because of there is no double representation of number 0. These are: Sign-Magnitude form, 1’s complement form, and 2’s complement form which are explained as following below.

(a) Sign-Magnitude form:

For n bit binary number, 1 bit is reserved for sign symbol. If the value of sign bit is 0, then the given number will be positive, else if the value of sign bit is 1, then the given number will be negative. Remaining (n-1) bits represent magnitude of the number. Since magnitude of number zero (0) is always 0, so there can be two representation of number zero (0), positive (+0) and negative (-0), which depends on value of sign bit. Hence these representations are ambiguous generally because of two representation of number zero (0). Generally sign bit is a most significant bit (MSB) of representation. The range of Sign-Magnitude form is from $(2^{(n-1)}-1)$ to $(2^{(n-1)}-1)$.

For example, range of 6 bit Sign-Magnitude form binary number is from (2^5-1) to (2^5-1) which is equal from minimum value -31 (i.e., 1 11111) to maximum value +31 (i.e., 0 11111). And zero (0) has two representation, -0 (i.e., 1 00000) and +0 (i.e., 0 00000).

(b) 1’s complement form:

Since, 1’s complement of a number is obtained by inverting each bit of given number. So, we represent positive numbers in binary form and negative numbers in 1’s complement form. There is extra bit for sign representation. If value of sign bit is 0, then number is positive and you can directly represent it in simple binary form, but if value of sign bit 1, then number is negative and you have to take 1’s complement of given binary number. You can get negative number by 1’s complement of a positive number and positive number by using 1’s complement of a negative number. Therefore, in this representation, zero (0) can have two representation, that’s why 1’s complement form is also ambiguous form. The range of 1’s complement form is *from* $(2^{(n-1)}-1)$ to $(2^{(n-1)}-1)$.

For example, range of 6 bit 1’s complement form binary number is from (2^5-1) to (2^5-1) which is equal from minimum value -31 (i.e., 1 00000) to maximum value +31 (i.e., 0 11111). And zero (0) has two representation, -0 (i.e., 1 11111) and +0 (i.e., 0 00000).

(c) 2’s complement form:

Since, 2’s complement of a number is obtained by inverting each bit of given number plus 1 to least significant bit (LSB). So, we represent positive numbers in binary form and negative numbers in 2’s complement form. There is extra bit for sign representation. If value of sign bit is 0, then number is positive and you can directly represent it in simple binary form, but if value of sign bit 1, then number is negative and you have to take 2’s complement of given binary number. You can get negative number by 2’s complement of a positive number and positive number by directly using simple binary representation. If value of most significant bit (MSB) is 1, then take 2’s complement from, else not. Therefore, in this representation, zero (0) has only one (unique) representation which is always positive. The range of 2’s complement form is *from* $(2^{(n-1)})$ to $(2^{(n-1)}-1)$.

For example, range of 6 bit 2’s complement form binary number is from (2^5) to (2^5-1) which is equal from minimum value -32 (i.e., 1 00000) to maximum value +31 (i.e., 0 11111). And zero (0) has two representation, -0 (i.e., 1 11111) and +0 (i.e., 0 00000).

Logic Gates

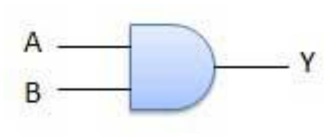
Logic gates are the basic building blocks of any digital system. It is an electronic circuit having one or more than one input and only one output. The relationship between the input and the output is based on a **certain logic**. Based on this, logic gates are named as AND gate, OR gate, NOT gate etc.

AND Gate

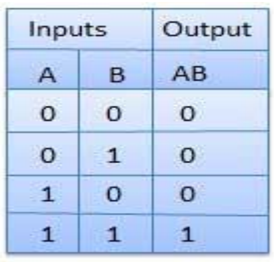
A circuit which performs an AND operation is shown in figure. It has n input (n >= 2) and one output.

Y	=	A AND B AND C .... N
Y	=	A.B.C .... N
Y	=	ABC .... N

Logic diagram



Truth Table



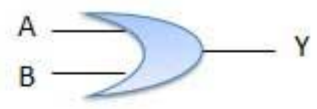
Inputs		Output
A	B	AB
0	0	0
0	1	0
1	0	0
1	1	1

OR Gate

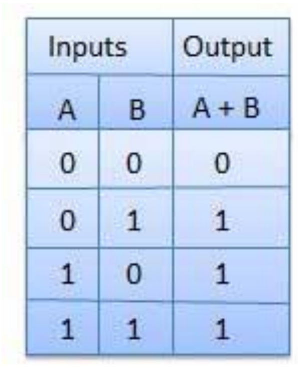
A circuit which performs an OR operation is shown in figure. It has n input (n >= 2) and one output.

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

Logic diagram



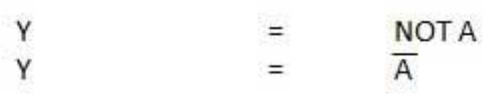
Truth Table



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

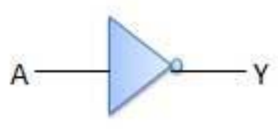
NOT Gate

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

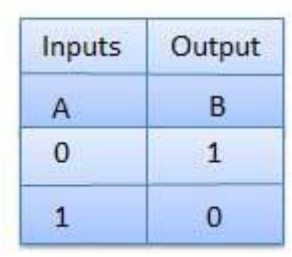


Y	=	NOT A
Y	=	$\overline{A}$

Logic diagram



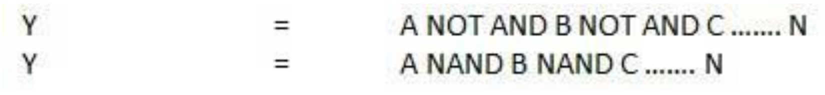
Truth Table



Inputs	Output
A	B
0	1
1	0

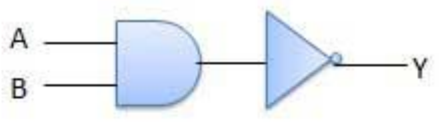
NAND Gate

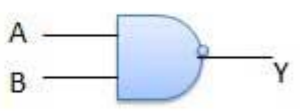
A NOT-AND operation is known as NAND operation. It has n input (n >= 2) and one output.



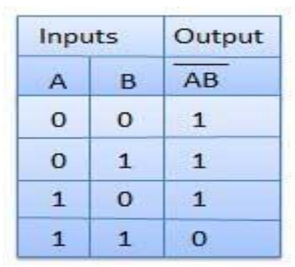
Y	=	A NOT AND B NOT AND C .... N
Y	=	A NAND B NAND C .... N

Logic diagram





Truth Table



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

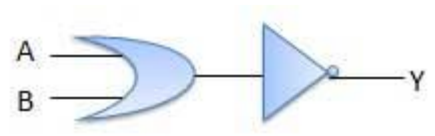
NOR Gate

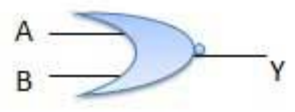
A NOT-OR operation is known as NOR operation. It has n input (n >= 2) and one output.

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

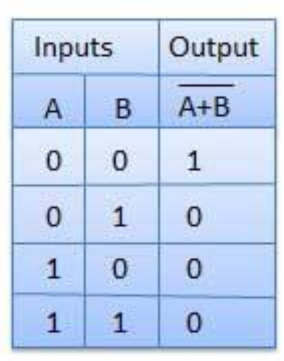
Y = A NOR B NOR C .... N

Logic diagram





Truth Table



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

XOR Gate

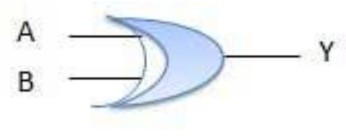
XOR or Ex-OR gate is a special type of gate. It can be used in the half adder, full adder and subtractor. The exclusive-OR gate is abbreviated as EX-OR gate or sometime as X-OR gate. It has n input (n >= 2) and one output.

Y = A XOR B XOR C .... N

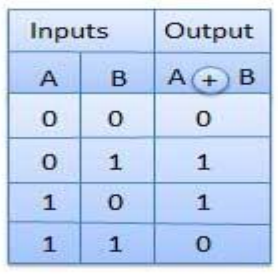
Y = $A \oplus B \oplus C$ .... N

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

Logic diagram



Truth Table



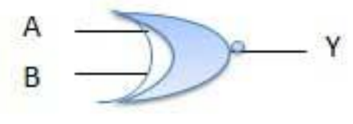
Inputs		Output
A	B	$A \oplus B$
0	0	0
0	1	1
1	0	1
1	1	0

XNOR Gate

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

$$Y = A \text{ XOR } B \text{ XOR } C \ldots\ldots\ldots N$$
$$Y = A \odot B \odot C \ldots\ldots\ldots N$$
$$Y = \overline{A}\,\overline{B} + AB$$

Logic diagram



Boolean Algebra

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

Rule in Boolean Algebra

Following are the important rules used in Boolean algebra.

- Variable used can have only two values. Binary 1 for HIGH and Binary 0 for LOW.
- Complement of a variable is represented by an overbar (-). Thus, complement of variable B is represented $\overline{B}$ as . Thus if B = 0 then $\overline{B}$ = 1 and B = 1 then $\overline{B}$ = 0.
- ORing of the variables is represented by a plus (+) sign between them. For example

ORing of A, B, C is represented as A + B + C.
- Logical ANDing of the two or more variable is represented by writing a dot between them such as A.B.C. Sometime the dot may be omitted like ABC.

Boolean Laws

There are six types of Boolean Laws.

Commutative law

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

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

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

Associative law

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

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

Distributive law

Distributive law states the following condition.

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

AND law

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

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

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

OR law

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

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

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

INVERSION law

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

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

Simplification Using Algebraic Functions

In this approach, one Boolean expression is minimized into an equivalent expression by applying Boolean identities.

Problem 1

Minimize the following Boolean expression using Boolean identities −

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

Solution

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

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

[By idempotent law, BC’ = BC’ + BC’]

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

Or, $F(A,B,C)=A'B+B(C'+C)+C'(B+AB')$

[By distributive laws]

Or, $F(A,B,C)=A'B+B.1+C'(B+A)$

[(C' + C) = 1 and absorption law (B + AB')= (B + A)]

Or, $F(A,B,C)=A'B+B+C'(B+A)$

[B.1 = B]

Or, $F(A,B,C)=B(A'+1)+C'(B+A)$

Or, $F(A,B,C)=B.1+C'(B+A)$

[(A' + 1) = 1]

Or, $F(A,B,C)=B+C'(B+A)$

[As, B.1 = B]

Or, $F(A,B,C)=B+BC'+AC'$

Or,$F(A,B,C)=B(1+C')+AC'$

Or,$F(A,B,C)=B.1+AC'$

[As, (1 + C') = 1]

Or,$F(A,B,C)=B+AC'$

[As, B.1 = B]

So,$F(A,B,C)=B+AC'$

is the minimized form.

Problem 2

Minimize the following Boolean expression using Boolean identities −

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

Solution

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

Or, $F(A,B,C)=A.A+A.C+B.A+B.C$

[Applying distributive Rule]

Or, $F(A,B,C)=A+A.C+B.A+B.C$

[Applying Idempotent Law]

Or, $F(A,B,C)=A(1+C)+B.A+B.C$

[Applying distributive Law]

Or, $F(A,B,C)=A+B.A+B.C$

[Applying dominance Law]

Or, $F(A,B,C)=(\mathrm{B}+1).A+B.C$

[Applying distributive Law]

Or, $F(A,B,C)=1.A+B.C$

[Applying dominance Law]

Or, $F(A,B,C)=A+B.C$

[Applying dominance Law]

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

is the minimized form.

Karnaugh Maps

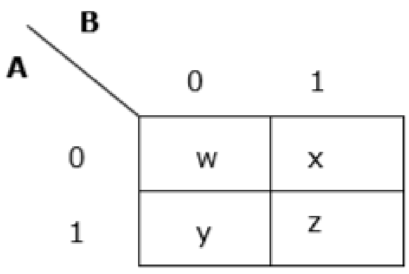
The Karnaugh map (K–map), introduced by Maurice Karnaughin in 1953, is a grid-like representation of a truth table which is used to simplify boolean algebra expressions. A Karnaugh map has zero and one entries at different positions. It provides grouping together Boolean expressions with common factors and eliminates unwanted variables from the expression. In a K-map, crossing a vertical or horizontal cell boundary is always a change of only one variable.

Example 1

An arbitrary truth table is taken below −

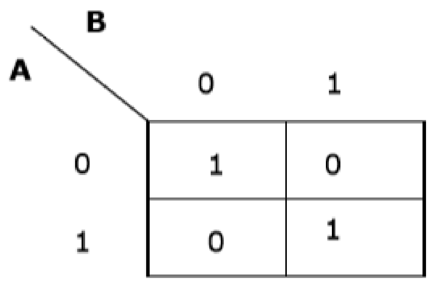
A	B	A operation B
0	0	w
0	1	x
1	0	y
1	1	z

Now we will make a k-map for the above truth table −



Example 2

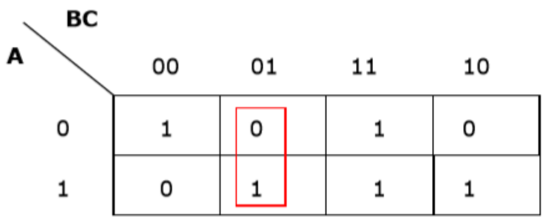
Now we will make a K-map for the expression − AB+ A’B’



Simplification Using K-map

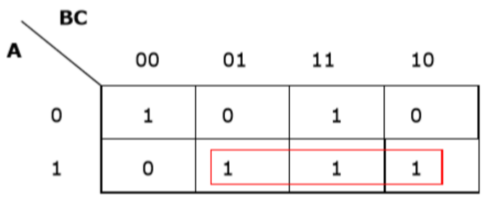
K-map uses some rules for the simplification of Boolean expressions by combining together adjacent cells into single term. The rules are described below −

Rule 1 − Any cell containing a zero cannot be grouped.



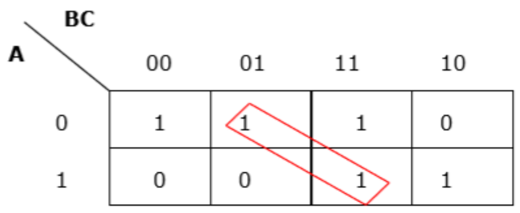
Wrong grouping

Rule 2 − Groups must contain 2n cells (n starting from 1).

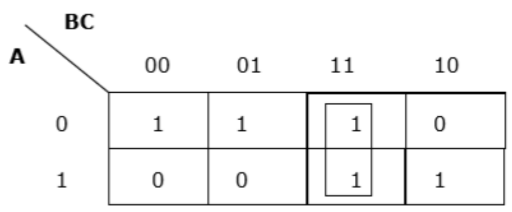


Wrong grouping

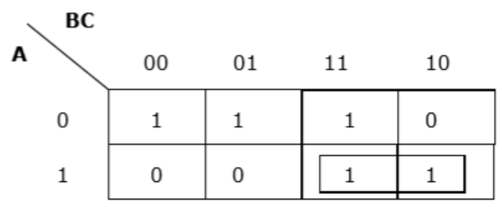
Rule 3 − Grouping must be horizontal or vertical, but must not be diagonal.



Wrong diagonal grouping

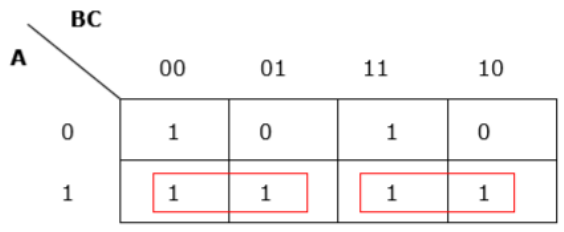


Proper vertical grouping

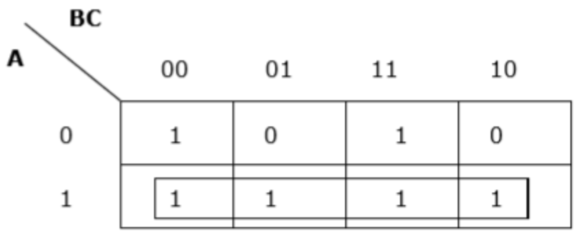


Proper horizontal grouping

Rule 4 − Groups must be covered as largely as possible.

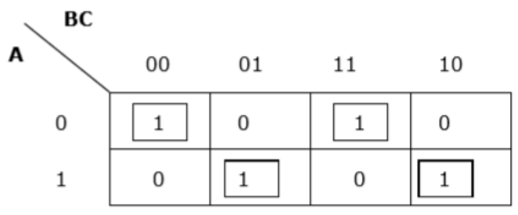


Insufficient grouping



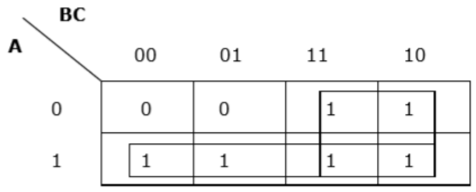
Proper grouping

Rule 5 − If 1 of any cell cannot be grouped with any other cell, it will act as a group itself.



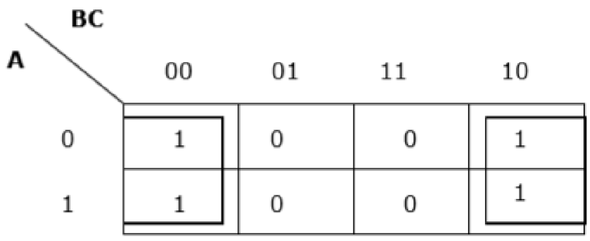
Proper grouping

Rule 6 − Groups may overlap but there should be as few groups as possible.



Proper grouping

Rule 7 − The leftmost cell/cells can be grouped with the rightmost cell/cells and the topmost cell/cells can be grouped with the bottommost cell/cells.



Proper grouping

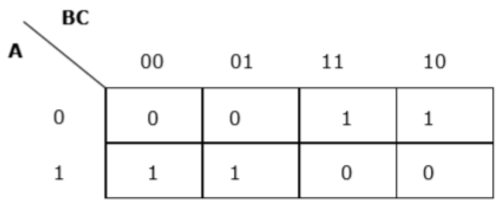
Problem

Minimize the following Boolean expression using K-map −

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

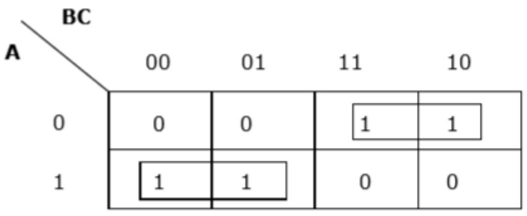
Solution

Each term is put into k-map and we get the following −



K-map for F (A, B, C)

Now we will group the cells of 1 according to the rules stated above −



K-map for F (A, B, C)

We have got two groups which are termed as $A'B$

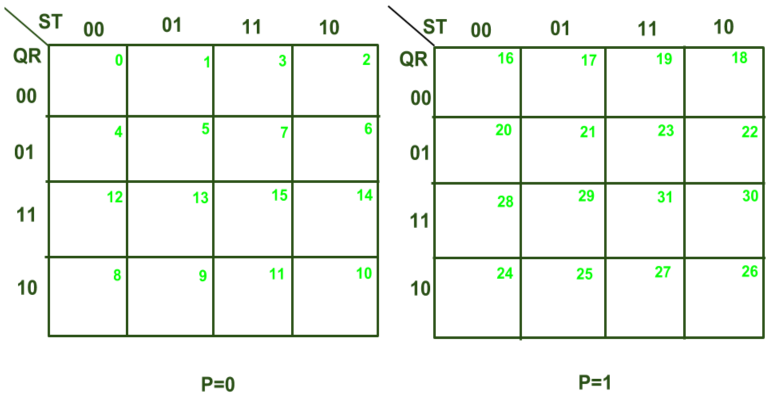
and AB'. Hence, $F(A,B,C)=A'B+AB'=A\oplus B$. It is the minimized form.

5 variable K-Map in Digital Logic

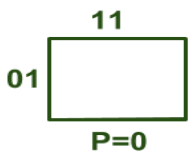
Prerequisite Implicant in K-Map
Karnaugh Map or K-Map is an alternative way to write truth table and is used for the simplification of Boolean Expressions. So far we are familiar with 3 variable K-Map & 4 variable K-Map. Now, let us discuss the 5-variable K-Map in detail.

Any Boolean Expression or Function comprising of 5 variables can be solved using the 5 variable K-Map. Such a 5 variable K-Map must contain **= 32 cells** . Let the 5-variable Boolean function be represented as :
f (P Q R S T) where P, Q, R, S, T are the variables and P is the most significant bit variable and T is the least significant bit variable.

The structure of such a K-Map for SOP expression is given below :



The cell no. written corresponding to each cell can be understood from the example described here:



Here for variable **P=0, we have Q = 0, R = 1, S = 1, T = 1 i.e. (PQRST)=(00111)** . In decimal form, this is equivalent to **7**. So, for the cell shown above the corresponding cell no. = 7. In a similar manner, we can write cell numbers corresponding to every cell as shown in the above figure.
Now let us discuss how to use a 5 variable K-Map to minimize a Boolean Function.

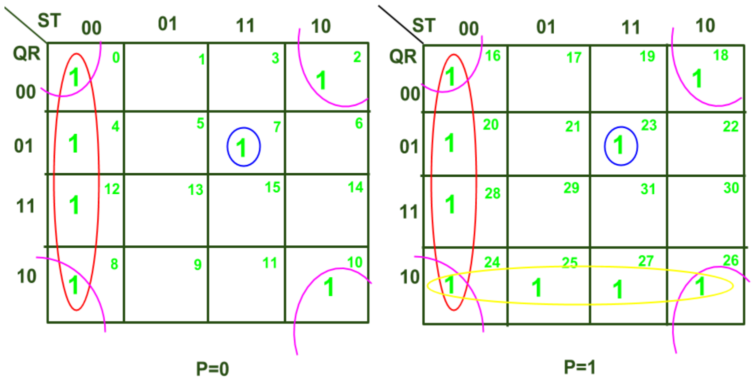
<u>Rules to be followed :</u>

1. If a function is given in compact canonical SOP(Sum of Products) form then we write **"1"** corresponding to each minterm (provided in the question) in the corresponding cell numbers. For eg: For we will write "1" corresponding to cell numbers (0, 1, 5, 7, 30 and 31).
2. If a function is given in compact canonical POS(Product of Sums) form then we write **"0"** corresponding to each maxterm (provided in the question) in the corresponding cell numbers. For eg: For we will write "0" corresponding to cell numbers (0, 1, 5, 7, 30 and 31).

<u>Steps to be followed :</u>

1. Make the largest possible size subcube covering all the marked 1's in case of SOP or all marked 0's in case of POS in the K-Map. It is important to note that each subcube can only contain terms in powers of 2 . Also a subcube of cells is possible if and only if in that subcube for every cell we satisfy that "m" number of cells are *adjacent cells* .
2. All *Essential Prime Implicants (EPIs)* must be present in the minimal expressions.

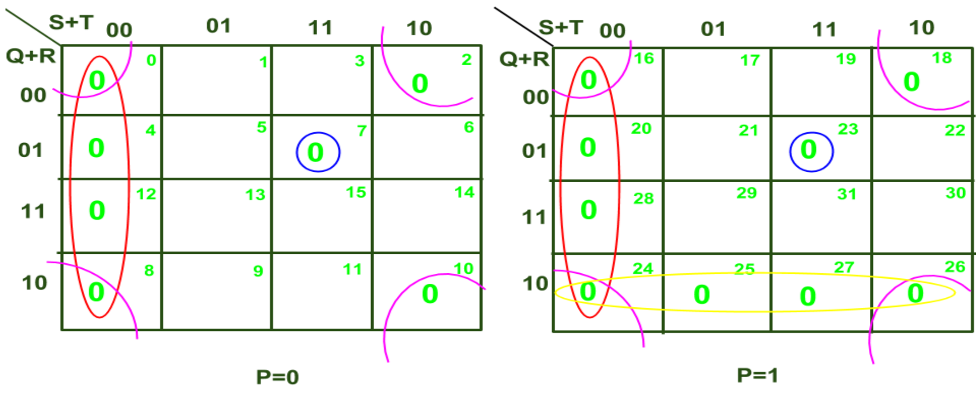
I. Solving SOP function –
For clear understanding, let us solve the example of SOP function minimization of 5 Variable K-Map using the following expression:



In the above K-Map we have 4 subcubes:

- **<u>Subcube 1:</u>** The one marked in red comprises of cells (0, 4, 8, 12, 16, 20, 24, 28)
- **<u>Subcube 2:</u>** The one marked in blue comprises of cells (7, 23)
- **<u>Subcube 3:</u>** The one marked in pink comprises of cells (0, 2, 8, 10, 16, 18, 24, 26)
- **<u>Subcube 4:</u>** The one marked in yellow comprises of cells (24, 25, 26, 27)

II. Solving POS function –
Now, let us solve the example of POS function minimization of 5 Variable K-Map using the following expression:



In the above K-Map we have 4 subcubes:

- **Subcube 1:** The one marked in red comprises of cells (0, 4, 8, 12, 16, 20, 24, 28)
- **Subcube 2:** The one marked in blue comprises of cells (7, 23)
- **Subcube 3:** The one marked in pink comprises of cells (0, 2, 8, 10, 16, 18, 24, 26)
- **Subcube 4:** The one marked in yellow comprises of cells (24, 25, 26, 27)