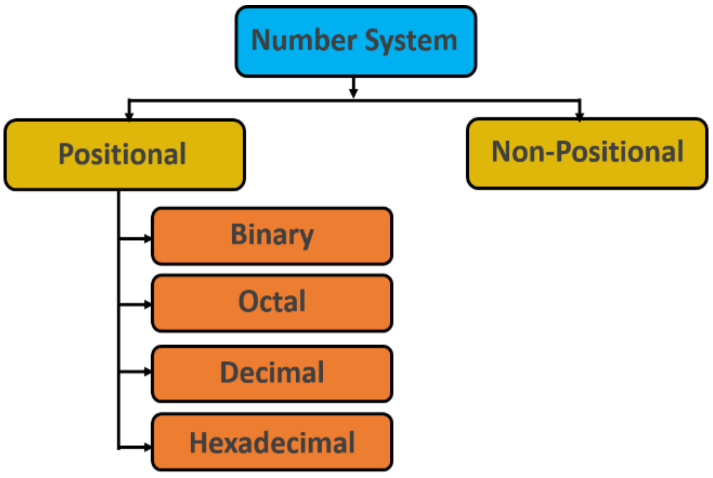
MODULE 2
Number Systems and Boolean Algebra

Number systems

The language we use to communicate with each other is comprised of words and characters. We understand numbers, characters and words. But this type of data is not suitable for computers. Computers only understand the numbers.



Non-Positional Number Systems: Non-Positional Number System does not use digits for the representation instead it use symbols for the representation. Each symbol represents same value regardless of its position in number. To find the value one has to count the symbols present in the number. For example, the roman number system is a good example of the non-positional number system. 1-2 II-2 III-3 IV-4 …..

Positional Number Systems: Positional Number System uses digits for the representation. Only few symbols called digits. These symbols represent different values. These values depending on the position. They occupy in number. To determine the value of each digit

Numbering System		
System	Base	Digits
Binary	2	0, 1
Octal	8	0,1,2,3,4,5,6,7
Decimal	10	0,1,2,3,4,5,6,7,8,9
Hexadecimal	16	0,1,2,3,4,5,6,7,8,9,A,B,C,D,E,F

1. The digit itself.
2. The position of the digit in the number.
3. The base of the number system.

Base of a number system: Total number of digits available in the number system.

Binary number system:

The base is 2. We can use only two symbols (0, 1) to represent any number. Each digit is called a bit. A group of four bits (1101) is called a nibble and group of eight bits (11001010) is called a byte. The position of each digit in a binary number represents a specific power of the base (2) of the number system.

$10101_2 = (1 \text{ x } 2^4) + (0 \text{ x } 2^3) + (1 \text{ x } 2^2) + (0 \text{ x } 2^1) + (1 \text{ x } 2^0) = 16 + 0 + 4 + 0 + 1 = 21_{10}$. Therefore

$10101_2=21_{10}$

Decimal number:

In our daily life we use decimal number system. It has base ten. Because there are 10 different Symbols (0,1,2,3,4,5,6,7,8,9)in this system to represent any number. Each position of a digit represents a specific power of the base (10). Ex: number 2586 represent as $(2x10^3) + (5X10^2) + (8x10^1) + (6x10^0) = (2586)_{10}$

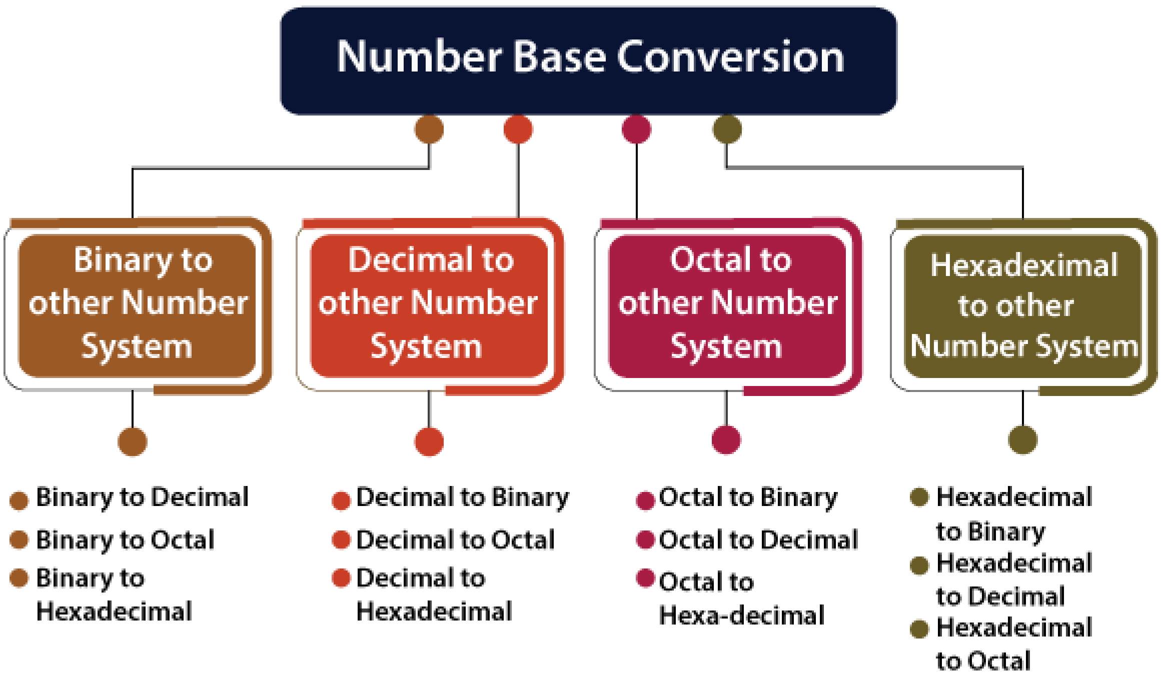
Octal number system:

There are only eight symbols in this number system .0, 1,2,3,4,5,6,7 .The largest single digit is 7. Each position of a digit represents a specific power of the base (8). Since there are only 8 digits, 3 bits
$(2^3 = 8)$ are sufficient to represent any octal number in binary.
Example: $2057_8 = (2 \times 8^3) + (0 \times 8^2) + (5 \times 8^1) + (7 \times 8^0) = 1024 + 0 + 40 + 7= 1071_{10}$

Hexadecimal number system:

The base is 16 in this number system. First 10 digits are same digits of decimal number system(0,1,2,3,4,5,6,7,8,9). The remaining six digit denoted by symbols A,B,C,D,E,F represent decimal value 10,11,12,13,14,15 respectively. Each position of a digit represents a specific power of the base (16). Since there are only 16 digits, 4 bits (24 = 16) are sufficient to represent any hexadecimal number in binary.
A=10, B=11, C=12, D=13, E=14, F=15

Example: $1AF_{16} = (1 \times 16^2) + (A \times 16^1) + (F \times 16^0) = 1 \times 256 + 10 \times 16 + 15 \times 1= 256 + 160 + 15= 431_{10}$



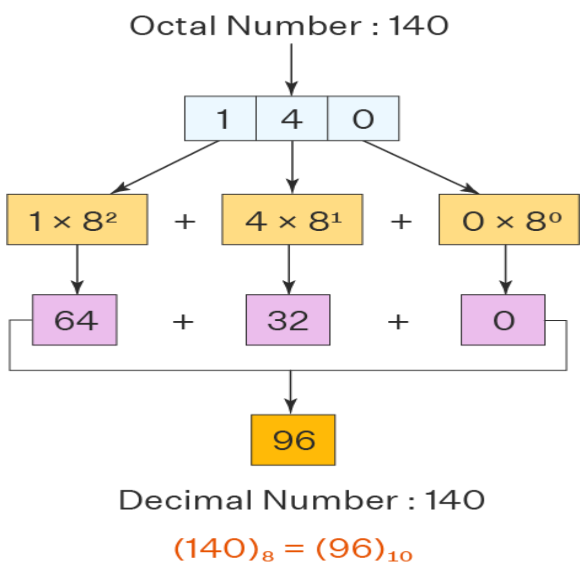
Converting a number of Another Base to a Decimal number:

Method

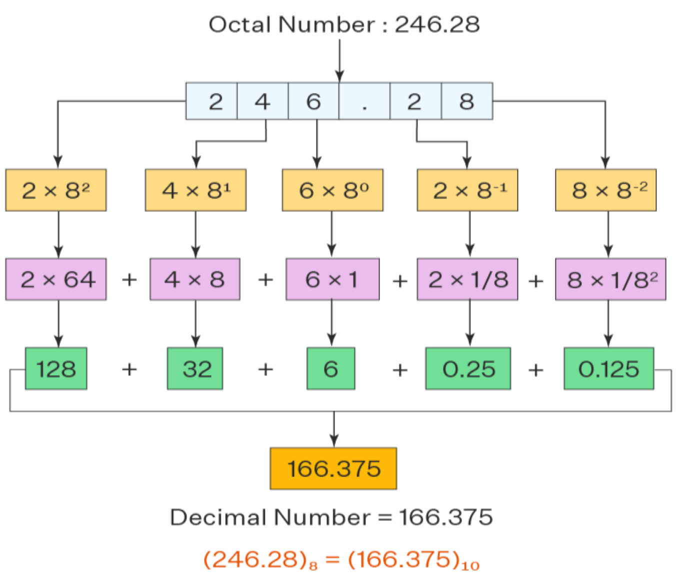
Step 1: Determine the column (positional) value of each digit
Step 2: Multiply the obtained column values by the digits in the corresponding columns
Step 3: Calculate the sum of these products
$4706_8 = ?_{10}$

$4706_8 = 4 \times 8^3 + 7 \times 8^2 + 0 \times 8^1 + 6 \times 8^0 = 4 \times 512 + 7 \times 64 + 0 + 6 \times 1 = 2048 + 448 + 0 + 6= 2502_{10}$

Ex : 1



Ex : 2



Converting a Decimal number to a number of Another Base

Division-Remainder Method:

Step 1: Divide the decimal number to be converted by the value of the new base

Step 2: Record the remainder from Step 1 as the rightmost digit (least significant digit) of the 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, recording remainders from right to left, until the quotient becomes zero in Step 3

Note that the last remainder thus obtained will be the most significant digit (MSD) of the new base number

Converting a Decimal number to a number of Another Base

Division-Remainder Method:

Step 1: Divide the decimal number to be converted by the value of the new base

Step 2: Record the remainder from Step 1 as the rightmost digit (least significant digit) of the 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, recording remainders from right to left, until the quotient becomes zero in Step 3

Note that the last remainder thus obtained will be the most significant digit (MSD) of the new base number

Converting a Decimal number to a number of Another Base

Division-Remainder Method:

Step 1: Divide the decimal number to be converted by the value of the new base

Step 2: Record the remainder from Step 1 as the rightmost digit (least significant digit) of the new base number

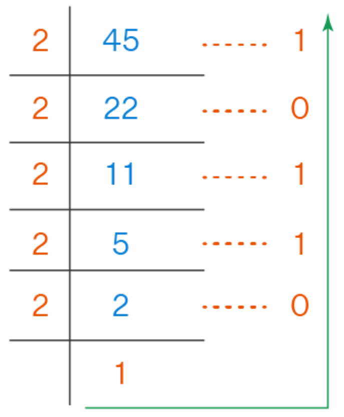
Step 3: Divide the quotient of the previous divide by the new base

Step 4: Record the remainder from Step 3 as the next digit (to the left) of the new base number

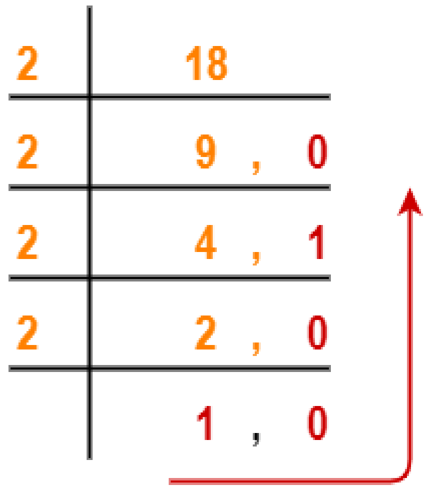
Repeat Steps 3 and 4, recording remainders from right to left, until the quotient becomes zero in Step 3

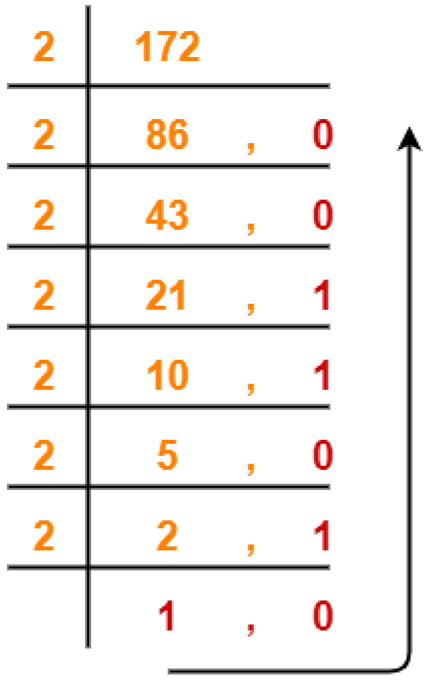
Note that the last remainder thus obtained will be the most significant digit (MSD) of the new base number

Ex 1: $45_{10} = (\ldots)_2$ Ex 2: $(18)_{10} = (\ )_2$ Ex 3: $(172)_{10} = (\ )_2$



$\therefore 45_{10} = 101101_2$





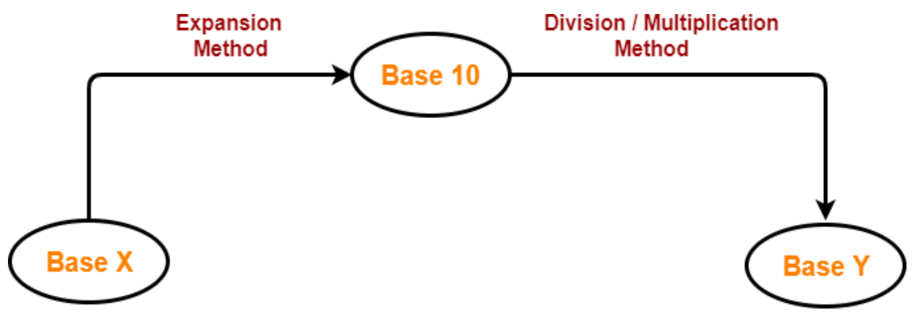
From here, $(18)_{10} = (10010)_2$ From here, $(172)_{10} = (10101100)_2$

<u>Converting a number of Some Base to a number of Another Base</u>

Method

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



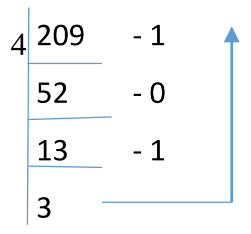
Conversion of Bases

Example: $545_6 = ?_4$

Solution: **Step 1:** Convert from base 6 to base 10

$$545_6 = 5 \times 6^2 + 4 \times 6^1 + 5 \times 6^0 = 5 \times 36 + 4 \times 6 + 5 \times 1 = 180 + 24 + 5 = 209_{10}$$

Step 2: Convert 209_{10} to base 4



Hence, $209_{10} = 31014_4$. So, $545_6 = 209_{10} = 31014_4$. Thus, $545_6 = 31014_4$

Shortcut Method for Converting a Binary Number to its Equivalent Octal Number

Step 1: Divide the digits into groups of three starting from the right

Step 2: Convert each group of three binary digits to one octal digit using the method of binary to decimal conversion

Ex: $1101010_2 = ?_8$

Step 1: Divide the binary digits into groups of 3 starting from right

$\underline{001}$ $\underline{101}$ $\underline{010}$

Step 2: Convert each group into one octal digit

$001_2 = 0 \times 2^2 + 0 \times 2^1 + 1 \times 2^0 = 1$

$101_2 = 1 \times 2^2 + 0 \times 2^1 + 1 \times 2^0 = 5$

$010_2 = 0 \times 2^2 + 1 \times 2^1 + 0 \times 2^0 = 2$

Hence, $1101010_2 = 152_8$

Shortcut Method for Converting an Octal Number to Its Equivalent Binary Number

Step 1: Convert each octal digit to a 3 digit binary number

Step 2: Combine all the resulting binary groups (of 3 digits each) into a single binary number

Ex: $562_8 = ?_2$

Step 1: Convert each octal digit to 3 binary digits

$5_8 = 101_2$ $6_8 = 110_2$, $2_8 = 010_2$

Step 2: Combine the binary groups

$562_8 = \underset{5}{\underline{101}}\ \underset{6}{\underline{110}}\ \underset{2}{\underline{010}}$

Hence, $562_8 = 101110010_2$

Shortcut Method for Converting a Binary Number to its Equivalent Hexadecimal Number

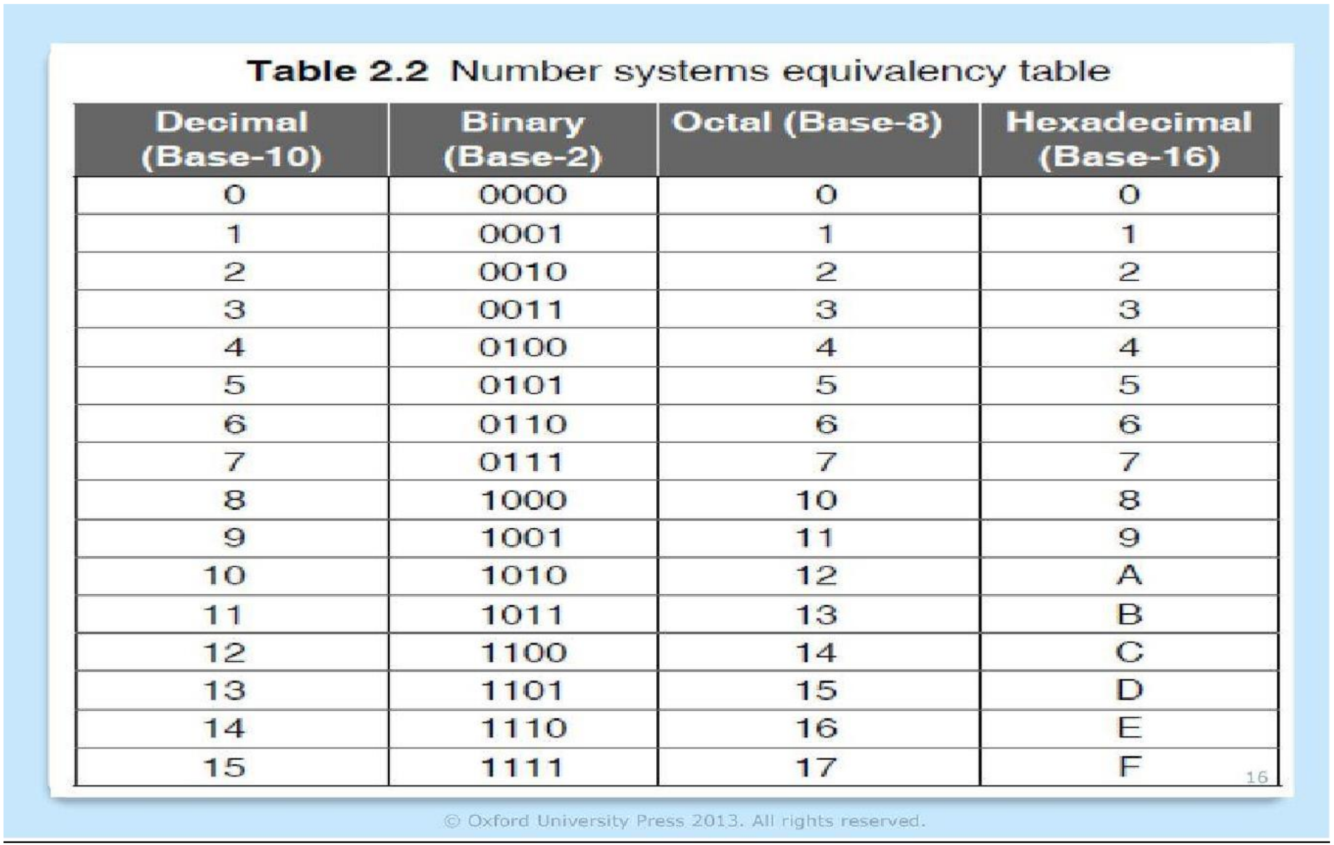


Table 2.2 Number systems equivalency table

Decimal (Base-10)	Binary (Base-2)	Octal (Base-8)	Hexadecimal (Base-16)
0	0000	0	0
1	0001	1	1
2	0010	2	2
3	0011	3	3
4	0100	4	4
5	0101	5	5
6	0110	6	6
7	0111	7	7
8	1000	10	8
9	1001	11	9
10	1010	12	A
11	1011	13	B
12	1100	14	C
13	1101	15	D
14	1110	16	E
15	1111	17	F

16

Method

Step 1: Divide the binary digits into groups of four starting from the right

Step 2: Combine each group of four binary digits to one hexadecimal digit

Ex: $111101_2 = ?_{16}$

Step 1: Divide the binary digits into groups of four starting from the right

$\underline{0011}$ $\underline{1101}$

Step 2: Convert each group into a hexadecimal digit

$0011_2 = 0 \times 2^3 + 0 \times 2^2 + 1 \times 2^1 + 1 \times 2^0 = 3_{10} = 3_{16}$

$1101_2 = 1 \times 2^3 + 1 \times 2^2 + 0 \times 2^1 + 1 \times 2^0 = 13_{10} = D_{16}$

Hence, $111101_2 = 3D_{16}$

Shortcut Method for Converting a Hexadecimal Number to its Equivalent Binary Number

Method

Step 1: Convert the decimal equivalent of each hexadecimal digit to a 4 digit binary number

Step 2: Combine all the resulting binary groups (of 4 digits each) in a single binary number

Ex: $2AB_{16} = ?_2$

Step 1: Convert each hexadecimal digit to a 4 digit binary number

$2_{16} = 2_{10} = 0010_2$ $A_{16} = 10_{10} = 1010_2$

$B_{16} = 11_{10} = 1011_2$

Step 2: Combine the binary groups

$2AB_{16} = \underset{2}{\underline{0010}} \quad \underset{A}{\underline{1010}} \quad \underset{B}{\underline{1011}}$

Hence, $2AB_{16} = 001010101011_2$

Fractional Numbers

Fractional numbers are formed same way as decimal number system.
In general, a number in a number system with base b would be written as: $a_n\, a_{n-1} \ldots a_0 \,.\, a_{-1}\, a_{-2} \ldots a_{-m}$
And would be interpreted to mean: $a_n \times b^n + a_{n-1} \times b^{n-1} + \ldots + a_0 \times b^0 + a_{-1} \times b^{-1} + a_{-2} \times b^{-2} + \ldots + a_{-m} \times b^{-m}$

Position	: 4	3	2	1	0	.	-1	-2	-3	-4
Position Value:	2^4	2^3	2^2	2^1	2^0	.	2^{-1}	2^{-2}	2^{-3}	2^{-4}
Quantity	: 16	8	4	2	1	.	1/2	1/4	1/8	1/16

Ex 1: $110.101_2 = 1 \times 2^2 + 1 \times 2^1 + 0 \times 2^0 + 1 \times 2^{-1} + 0 \times 2^{-2} + 1 \times 2^{-3}$

$= 4 + 2 + 0 + 0.5 + 0 + 0.125 = 6.625_{10}$

Ex 2: $127.548 = 1 \times 8^2 + 2 \times 8^1 + 7 \times 8^0 + 5 \times 8^{-1} + 4 \times 8\text{-}^2$

$= 64 + 16 + 7 + 5/8 + 4/64 = 87 + 0.625 + 0.0625 = 87.687510$

Arithmetic involving Number Systems

Data is manipulated by using the arithmetic instructions in digital computers. Data is manipulated to produce results necessary to give solution for the computation problems. The *Addition, subtraction, multiplication and division are the four basic arithmetic operations.* If we want then we can derive other operations by using these four operations.

To execute arithmetic operations there is a separate section called arithmetic processing unit in central processing unit. The arithmetic instructions are performed generally on binary or decimal data. Negative numbers may be in a signed magnitude or signed complement representation. There are three ways of representing negative fixed point - binary numbers signed magnitude, signed 1's complement or signed 2's complement.

Importance of Binary

Information is handled in a computer by electronic/electrical components. Electronic components operate in binary mode (can only indicate two states – on (1) or off (0)). Binary number system has only two digits (0 and 1), and is suitable for expressing two possible states.
In binary system, computer circuits only have to handle two binary digits rather than ten decimal digits causing:

- Simpler internal circuit design
- Less expensive
- More reliable circuits

Basic arithmetic operations using binary numbers: *Addition (+), Subtraction (-), Multiplication (*), Division (/)*
Binary arithmetic is simple to learn as binary number system has only two digits – 0 and 1.

Binary Addition

Rule for binary addition is as follows:

0 + 0 = 0
0 + 1 = 1
1 + 0 = 1
1 + 1 = 0 plus a carry of 1 to next higher column

Example 1: Add binary numbers 10011 and 1001 in both decimal and binary form

Solution:	Binary	Decimal	
	Carry 11	Carry 1	
	10011 +	19 +	
	1001	9	
	11100	28	in this example, carry are generated for first and second columns

Example 2: Add binary numbers 100111 and 11011 in both decimal and binary form

Solution :	Binary	Decimal
	Carry 11111	carry 1
	100111 +	39 +
	11011	27
	1000010	66

Binary Subtraction

Rule:	0 - 0 = 0	
	0 - 1 = 1	Borrow 1 from next high order digit
	1 - 0 = 1	
	1 - 1 = 0	

Binary subtraction is also similar to that of decimal subtraction with the difference that when 1 is subtracted from 0,it is necessary to borrow 1 from the next higher order bit and that bit is reduced by 1 and the remainder is 1.

For fractional numbers, the rules of subtraction are the same with the binary point properly placed.

Ex1:
```
 1 0 1 0 -
   1 0 1
 ---------
 0 1 0 1
```

Ex 2: 0011010 – 001100
```
 0 0 1 1 0 1 0 -
   0 0 1 1 0 0
 ---------------
  0 0 0 1 1 1 0
 ---------------
```

Decimal Equivalent: 0 0 1 1 0 1 0 = 26_{10}, 0 0 1 1 0 0 = 12_{10}, Therefore, 26 – 12 = 14_{10} is binary 0 0 0 1 1 1 0_2

Ex 3: 0100010 – 0001010
```
 0 1 0 0 0 1 0
 0 0 0 1 0 1 0
 ---------------
 0 0 1 1 0 0 0 = 24₁₀
 ---------------
```

Decimal Equivalent: $0\ 1\ 0\ 0\ 0_{1\ 0} = 34_{10}$, $0\ 0\ 0\ 1\ 0_{1\ 0} = 10_{10}$, Therefore, 34 – 10 = 24_{10} is binary $0\ 0\ 1\ 1\ 0\ 0\ 0_{2}$

<u>Binary Multiplication & Division :</u> Whenever at least one input is 0, then multiplication is always 0.

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

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

There are four part in any division: Dividend, Divisor, quotient, and remainder. These are given as following rules for binary division, whenever divisor is 0, then result is always not defined.

<u>Complement of a Binary Number</u>

Complement of a binary number can be obtained by transforming all its 0’s to 1’s and all its 1’s to 0’s

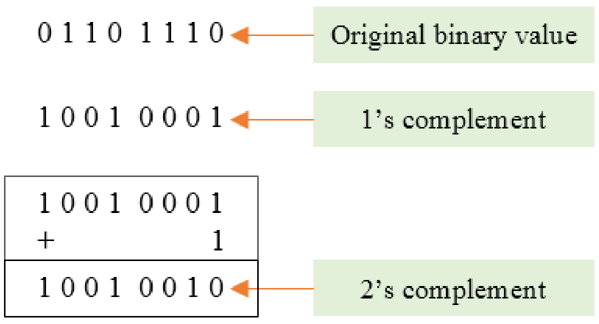
Example 1: Complement of 1 0 1 1 0 1 0 is 0 1 0 0 1 0 1

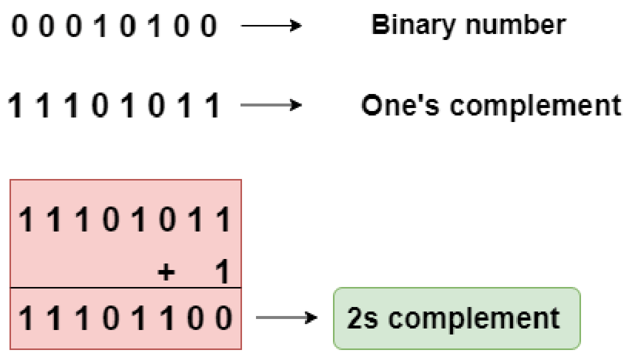
```
1 0 1 1 0 1 0
↓ ↓ ↓ ↓ ↓ ↓ ↓
0 1 0 0 1 0 1
```

Example 2: Complement of 0 0 0 1 0 0 1 1 is 1 1 1 0 1 1 0 0

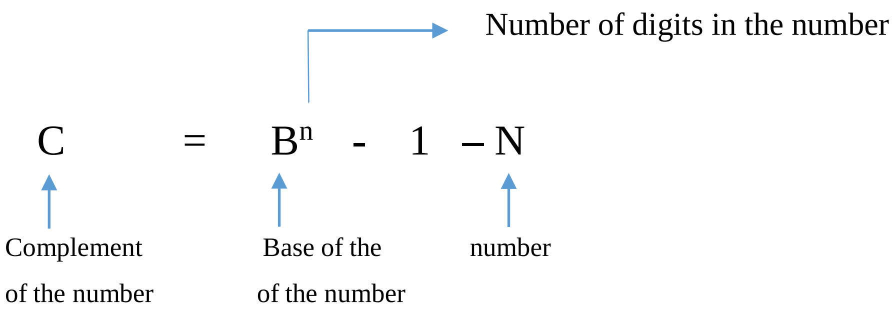
1's and 2's complement of a Binary Number

1's complement of a binary number is the number that can be obtained by changing all ones to zeros and all zeros to ones of a given binary number. Whereas, 2's complement is a binary number that can be obtained by adding 1 to one's complement of a given binary number.





Complement of a Number



Example 1: Find the complement of 37_{10}

Solution : Since the number has 2 digits and the value of base is 10,

$(\text{Base})^n - 1 = 10^2 - 1 = 99$

Now 99 - 37 = 62

Hence, complement of $37_{10} = 62_{10}$

Example 2: Find the complement of 6_8

Solution : Since the number has 1 digit and the value of base is 8,

$(\text{Base})^n - 1 = 8^1 - 1 = 7_{10} = 7_8$

Now $7_8 - 6_8 = 1_8$

Hence, complement of $6_8 = 1_8$

Complementary Method of Subtraction

Involves following 3 steps:

Step 1: Find the complement of the number you are subtracting (subtrahend)

Step 2: Add this to the number from which you are taking away (minuend)

Step 3: If there is a carry of 1, add it to obtain the result; if there is no carry, recomplement the sum and attach a negative sign.

Example 1: Subtract 56_{10} from 92_{10} using complementary method.

Solution : Step 1: Complement of $56_{10} = 10^2 - 1 - 56 = 99 – 56 = 43_{10}$

Step 2: 92 + 43 (complement of 56) = 135 (note 1 as carry)

Step 3: 35 + 1 (add 1 carry to sum) Result = 36

The result may be verified using the method of normal subtraction: 92 - 56 = 36

Example 2: Subtract 35_{10} from 18_{10} using complementary method.

Solution : Step 1: Complement of $35_{10} = 10^2 - 1 - 35 = 99 - 35 = 64_{10}$

Step 2: 18 +

64 (complement of 35)

82

Step 3: Since there is no carry, re-complement the sum and attach a negative sign to obtain the result.

Result = - (99 - 82) = -17

The result may be verified using normal subtraction: 18 - 35 = -17

Binary Subtraction Using Complementary Method

Example 1: Subtract 0111000_2 (56_{10}) from 1011100_2 (92_{10}) using complementary method.

Solution : 1011100 +

1000111 (complement of 0111000)

1 0100011

$\xrightarrow{\quad}$ 1 (add the carry of 1)

0100100

Result = $0100100_2 = 36_{10}$

Example 2: Subtract 100011_2 (35_{10}) from 010010_2 (18_{10}) using complementary method.

Solution : 010010 +

011100 (complement of 100011)

101110 Since there is no carry, we have to complement the sum and attach a negative sign to it.

Hence, Result = -010001_2 (complement of 101110_2) = -17_{10}

Digital Codes (BCD Code,ASCII Code ,Unicode, Gray Code, Excess-3 Code)

BCD or Binary Coded Decimal

Binary Coded Decimal, or BCD, is another process for converting decimal numbers into their binary equivalents.
It is a form of binary encoding where each digit in a decimal number is represented in the form of bits.
This encoding can be done in either 4-bit.It is a fast and efficient system that converts the decimal numbers into binary numbers as compared to the existing binary system.

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

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

Ex 1. **Convert $(123)_{10}$ in BCD**

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

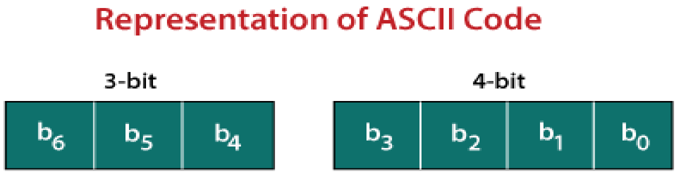
Ex 2. **Convert $(324)_{10}$ in BCD** $(324)_{10}$ -> 0011 0010 0100 (BCD)

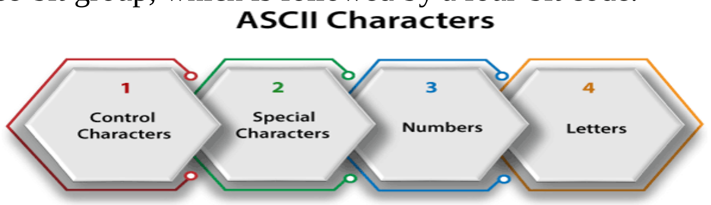
Again from the truth table above, 3 -> 0011   2 -> 0010   4 -> 0100 thus, BCD becomes -> 0011 0010 0100

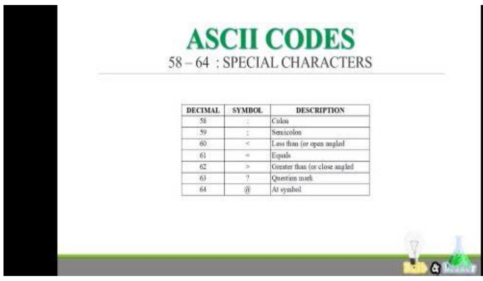
Decimal	Binay (BCD)
	8 4 2 1
0	0 0 0 0
1	0 0 0 1
2	0 0 1 0
3	0 0 1 1
4	0 1 0 0
5	0 1 0 1
6	0 1 1 0
7	0 1 1 1
8	1 0 0 0
9	1 0 0 1

ASCII Code

The ASCII stands for American Standard Code for Information Interchange. The ASCII code is an alphanumeric code used for data communication in digital computers. The ASCII is a 7-bit code capable of representing 2^7 or 128 number of different characters. The ASCII code is made up of a three-bit group, which is followed by a four-bit code.



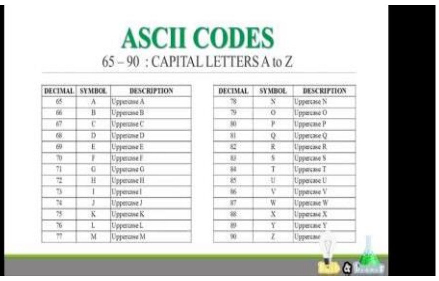




ASCII CODES

58 – 64 : SPECIAL CHARACTERS

DECIMAL	SYMBOL	DESCRIPTION
58	:	Colon
59	;	Semicolon
60	<	Less than (or open angled
61	=	Equals
62	>	Greater than (or close angled
63	?	Question mark
64	@	At symbol

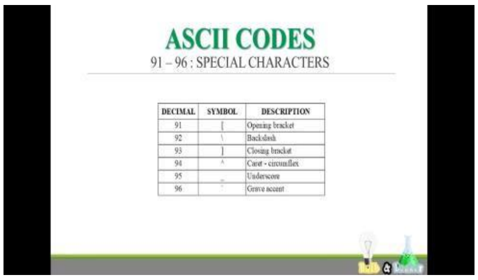


ASCII CODES

65 – 90 : CAPITAL LETTERS A to Z

DECIMAL	SYMBOL	DESCRIPTION
65	A	Uppercase A
66	B	Uppercase B
67	C	Uppercase C
68	D	Uppercase D
69	E	Uppercase E
70	F	Uppercase F
71	G	Uppercase G
72	H	Uppercase H
73	I	Uppercase I
74	J	Uppercase J
75	K	Uppercase K
76	L	Uppercase L
77	M	Uppercase M

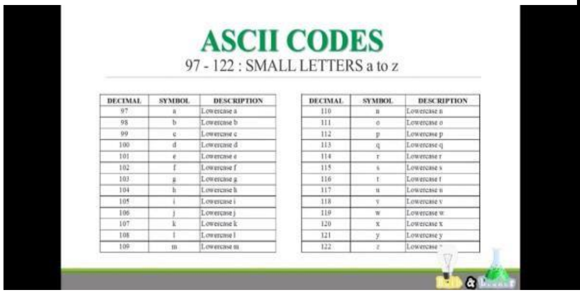
DECIMAL	SYMBOL	DESCRIPTION
78	N	Uppercase N
79	O	Uppercase O
80	P	Uppercase P
81	Q	Uppercase Q
82	R	Uppercase R
83	S	Uppercase S
84	T	Uppercase T
85	U	Uppercase U
86	V	Uppercase V
87	W	Uppercase W
88	X	Uppercase X
89	Y	Uppercase Y
90	Z	Uppercase



ASCII CODES

91 – 96 : SPECIAL CHARACTERS

DECIMAL	SYMBOL	DESCRIPTION
91	[	Opening bracket
92	\	Backslash
93	]	Closing bracket
94	^	Caret - circumflex
95	_	Underscore
96	`	Grave accent

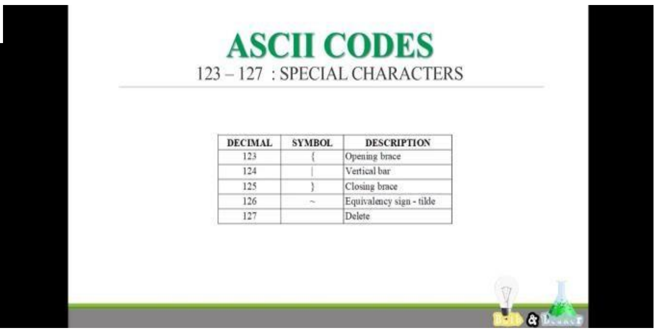


ASCII CODES

97 - 122 : SMALL LETTERS a to z

DECIMAL	SYMBOL	DESCRIPTION
97	a	Lowercase a
98	b	Lowercase b
99	c	Lowercase c
100	d	Lowercase d
101	e	Lowercase e
102	f	Lowercase f
103	g	Lowercase g
104	h	Lowercase h
105	i	Lowercase i
106	j	Lowercase j
107	k	Lowercase k
108	l	Lowercase l
109	m	Lowercase m

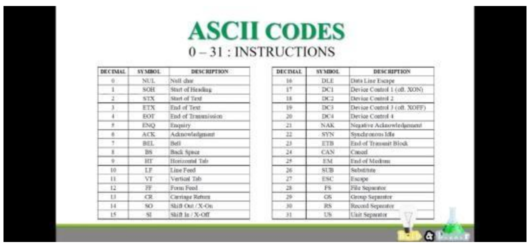
DECIMAL	SYMBOL	DESCRIPTION
110	n	Lowercase n
111	o	Lowercase o
112	p	Lowercase p
113	q	Lowercase q
114	r	Lowercase r
115	s	Lowercase s
116	t	Lowercase t
117	u	Lowercase u
118	v	Lowercase v
119	w	Lowercase w
120	x	Lowercase x
121	y	Lowercase y
122	z	Lowercase



ASCII CODES

123 – 127 : SPECIAL CHARACTERS

DECIMAL	SYMBOL	DESCRIPTION
123	{	Opening brace
124	|	Vertical bar
125	}	Closing brace
126	~	Equivalency sign - tilde
127		Delete

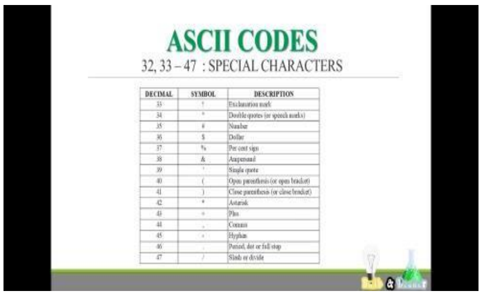


ASCII CODES

0 – 31 : INSTRUCTIONS

DECIMAL	SYMBOL	DESCRIPTION
0	NUL	Null char
1	SOH	Start of Heading
2	STX	Start of Text
3	ETX	End of Text
4	EOT	End of Transmission
5	ENQ	Enquiry
6	ACK	Acknowledgment
7	BEL	Bell
8	BS	Back Space
9	HT	Horizontal Tab
10	LF	Line Feed
11	VT	Vertical Tab
12	FF	Form Feed
13	CR	Carriage Return
14	SO	Shift Out / X-On
15	SI	Shift In / X-Off

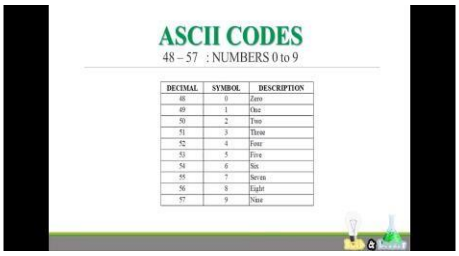
DECIMAL	SYMBOL	DESCRIPTION
16	DLE	Data Line Escape
17	DC1	Device Control 1 (oft. XON)
18	DC2	Device Control 2
19	DC3	Device Control 3 (oft. XOFF)
20	DC4	Device Control 4
21	NAK	Negative Acknowledgement
22	SYN	Synchronous Idle
23	ETB	End of Transmit Block
24	CAN	Cancel
25	EM	End of Medium
26	SUB	Substitute
27	ESC	Escape
28	FS	File Separator
29	GS	Group Separator
30	RS	Record Separator
31	US	Unit Separator



ASCII CODES

32, 33 – 47 : SPECIAL CHARACTERS

DECIMAL	SYMBOL	DESCRIPTION
33	!	Exclamation mark
34	"	Double quotes (or speech marks)
35	#	Number
36	$	Dollar
37	%	Per cent sign
38	&	Ampersand
39	'	Single quote
40	(	Open parenthesis (or open bracket)
41	)	Close parenthesis (or close bracket)
42	*	Asterisk
43	+	Plus
44	,	Comma
45	-	Hyphen
46	.	Period, dot or full stop
47	/	Slash or divide



ASCII CODES

48 – 57 : NUMBERS 0 to 9

DECIMAL	SYMBOL	DESCRIPTION
48	0	Zero
49	1	One
50	2	Two
51	3	Three
52	4	Four
53	5	Five
54	6	Six
55	7	Seven
56	8	Eight
57	9	Nine

Unicode

Unicode is a universal character encoding standard. This standard includes roughly 100000 characters to represent characters of different languages. While ASCII uses only 1 byte the Unicode uses 4 bytes to represent characters. Hence, it provides a very wide variety of encoding. It has three types namely UTF-8, UTF-16, UTF-32. Among them, UTF-8 is used mostly it is also the default encoding for many programming languages.

UTF-8-It is the most commonly used form of encoding. Furthermore, it has the capacity to use up to 4 bytes for representing the characters. It uses:

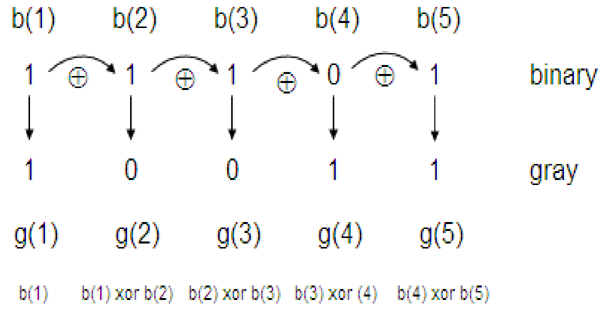
- 1 byte to represent English letters and symbols.
- 2 bytes to represent additional Latin and Middle Eastern letters and symbols.
- 3 bytes to represent Asian letters and symbols.
- 4 bytes for other additional characters.

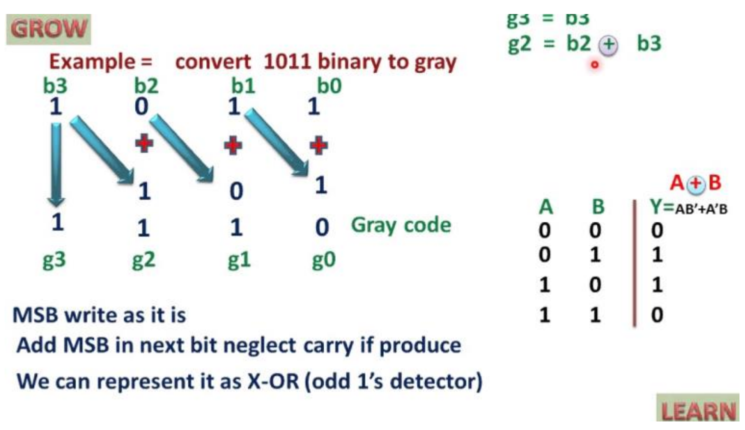
Difference between Unicode and ASCII

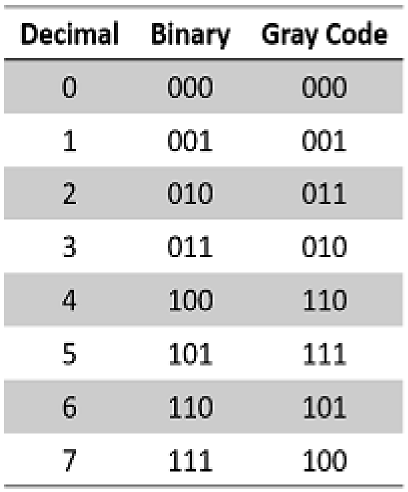
Unicode Coding Scheme	**ASCII Coding Scheme**
• It uses variable bit encoding according to the requirement. For example, UTF-8, UTF-16, UTF-32	• It uses 7-bit encoding. As of now, the extended form uses 8-bit encoding.
• It is a standard form.	• It is not a standard all over the world.
• People use this scheme all over the world.	• It has only limited characters hence, it cannot be used all over the world.
• The Unicode characters themselves involve all the characters of the ASCII encoding. Therefore we can say that it is a superset for it.	• It has its equivalent coding characters in the Unicode.
• It has more than 128,000 characters.	• In contrast, it has only 256 characters.

Gray code

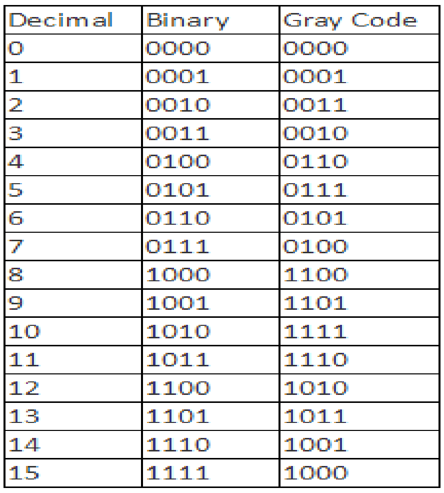
The reflected binary code or Gray code is an ordering of the binary numeral system such that two successive values differ in only one bit (binary digit). Gray codes are very useful in the normal sequence of binary numbers generated by the hardware that may cause an error or ambiguity during the transition from one number to the next. So, the Gray code can eliminate this problem easily since only one bit changes its value during any transition between two numbers.







Decimal	Binary	Gray Code
0	000	000
1	001	001
2	010	011
3	011	010
4	100	110
5	101	111
6	110	101
7	111	100



Decimal	Binary	Gray Code
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

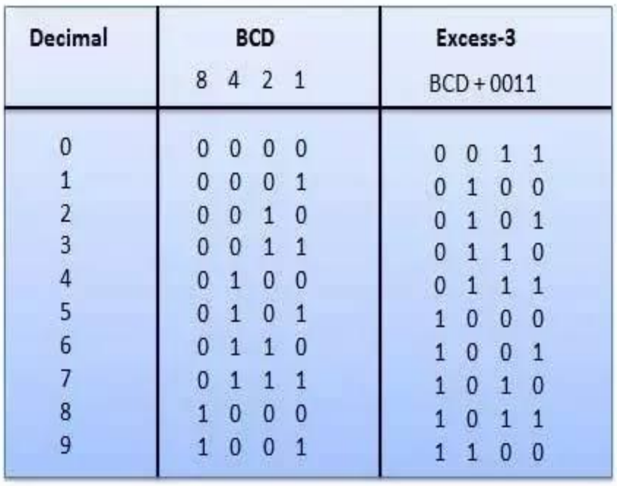
Excess-3 code

The excess-3 code is also treated as **XS-3 code**. The excess-3 code is a non-weighted and self-complementary BCD code used to represent the decimal numbers. This code has a biased representation. This code plays an important role in arithmetic operations because it resolves deficiencies encountered when we use the 8421 BCD code for adding two decimal digits whose sum is greater than 9. The Excess-3 code uses a special type of algorithm, which differs from the binary positional number system or normal non-biased BCD.

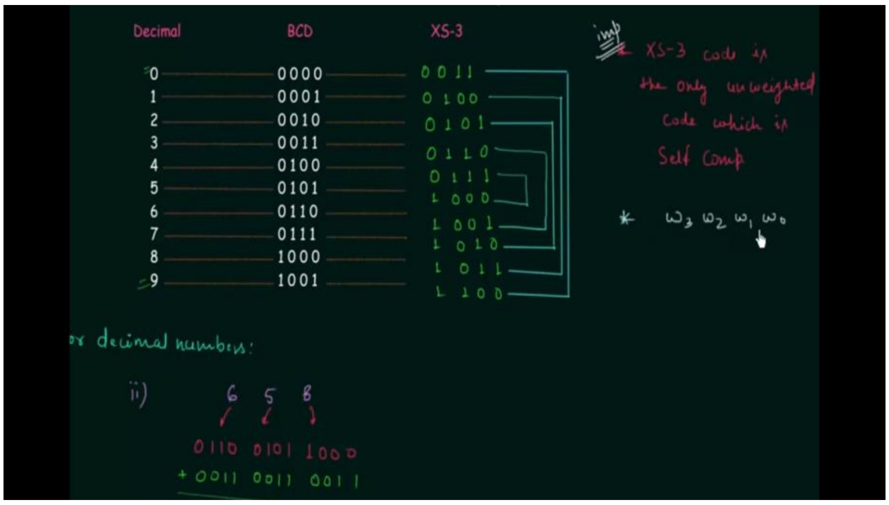
We can easily get an excess-3 code of a decimal number by simply adding 3 to each decimal digit. And then we write the 4-bit binary number for each digit of the decimal number. We can find the excess-3 code of the given binary number by using the following steps:

1. We find the decimal number of the given binary number.
2. Then we add 3 in each digit of the decimal number.
3. Now, we find the binary code of each digit of the newly generated decimal number.

We can also add 0011 in each 4-bit BCD code of the decimal number for getting excess-3 code.



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



Decimal	BCD	XS-3
0	0000	0011
1	0001	0100
2	0010	0101
3	0011	0110
4	0100	0111
5	0101	1000
6	0110	1001
7	0111	1010
8	1000	1011
9	1001	1100

imp XS-3 code is the only unweighted code which is Self Comp

* $\omega_3\ \omega_2\ \omega_1\ \omega_0$

or decimal numbers:

ii) 6 5 8

0110 0101 1000

+ 0011 0011 0011

Fundamental concepts of Boolean Algebra

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

Boolean equations can have either of two possiblevalues, 0 and 1

- Logical Addition- Symbol '**+**', also known as '**OR**' operator, used for logical addition. Thisfollows law of binary addition.
- Logical Multiplication-Symbol '**.**', also known as '**AND**' operator, used forlogical multiplication. This follows law of binarymultiplication.
- Complementation- Symbol '**-**', also known as '**NOT**' operator, used forcomplementation. This follows law of binary compliment.

NOT	AND	OR
0' = 1 1' = 0	$0 \cdot 0 = 0$ $0 \cdot 1 = 0$ $1 \cdot 0 = 0$ $1 \cdot 1 = 1$	$0 + 0 = 0$ $0 + 1 = 1$ $1 + 0 = 1$ $1 + 1 = 1$

Boolean Algebra Terminologies :

Boolean Variables: A boolean variable is defined as a variable or a symbol defined as a variable or a symbol, generally an alphabet that represents the logical quantities such as 0 or 1.

Boolean Function: A boolean function consists of binary variables, logical operators, constants such as 0 and 1, equal to the operator, and the parenthesis symbols.

Literal: A literal may be a variable or a complement of a variable.

Complement: The complement is defined as the inverse of a variable, which is represented by a bar over the variable.

Boolean Expression and Variables :

A Boolean expression is an expression that produces a Boolean value when evaluated, true or false, the only way to express a Boolean value. Whereas boolean variables are variables that store Boolean numbers. P + Q = R is a Boolean phrase in which P, Q, R are Boolean variables that can only store two values: 0 and 1. To effectively comprehend Boolean logic, we must first comprehend the rules of Boolean logic, as well as the truth table and logic gates.

Truth Tables:

A truth table represents all the variety of combinations of input values and outputs in a tabular manner. All the possibilities of the input and output are shown in it. In logic problems such as Boolean algebra and electronic circuits, truth tables are commonly used. T or 1 denotes 'True' & F or 0 denotes 'False' in the truth table.

Operator Precedence:

Each operator has a precedence (Priority) level. Higher the operator's precedence level, earlier it is evaluated. Expression is scanned from left to right

- First, expressions enclosed within parentheses are evaluated
- Then, all complement (NOT) operations are performed
- Then, all '×' (AND) operations are performed
- Finally, all '+' (OR) operations are performed

Postulates of Boolean Algebra,

Postulate 1:

(a) A = 0, if and only if, A is not equal to 1
(b) A = 1, if and only if, A is not equal to 0

Postulate 2:

(a) $x + 0 = x$
(b) $x \times 1 = x$

Postulate 3: Commutative Law

(a) $x + y = y + x$
(b) $x \times y = y \times x$

Postulate 4: Associative Law

(a) $x + (y + z) = (x + y) + z$
(b) $x \times (y \times z) = (x \times y) \times z$

Postulate 5: Distributive Law

(a) $x \times (y + z) = (x \times y) + (x \times z)$
(b) $x + (y \times z) = (x + y) \times (x + z)$

Postulate 6:

(a) $x + x' = 1$
(b) $x \times x' = 0$

Laws of Boolean Algebra:

Every law in Boolean algebra has two forms that are obtained by exchanging all the ANDs to ORs and 1s to 0s and vice versa. This is known as the Boolean algebra duality principle. The order of operations for Boolean algebra, from highest to lowest priority is NOT, then AND, then OR. Expressions inside brackets are always evaluated first.

1. **Commutative Law**	(a) $A + B = B + A$	(b) $A \cdot B = B \cdot A$
2. **Associative Law**	(a) $(A + B) + C = A + (B + C)$	(b) $(A \cdot B) \cdot C = A \cdot (B \cdot C)$
3. **Distributive Law**	(a) $A \cdot (B + C) = A \cdot B + A \cdot C$	(b) $A + (B \cdot C) = (A + B) \cdot (A + C)$
4. **Null Law**	(a) $1 + A = 1$	(b) $0 \cdot A = 0$
5. **Identity law**	(a) $0 + A = A$	(b) $1 \cdot A = A$
6. **Idempotent Law**	(a) $A + A = A$	(b) $A \cdot A = A$
7. **Complementarity Law**	(a) $A' + A = 1$	(b) $A' \cdot A = 0$
8. **Uniting Theorem**	(a) $A \cdot B + A \cdot B' = A$	(b) $(A + B) \cdot (A + B') = A$
9. **Absorption Theorem**	(a) $A + A \cdot B = A$	(b) $A \cdot (A + B) = A$
10. **Adsorption Theorem**	(a) $A + A' \cdot B = A + B$	(b) $A \cdot (A' + B) = A \cdot B$
11. **De Morgan's Theorem**	(a) $(A + B)' = A' \cdot B'$	(b) $(A \cdot B)' = A' + B'$

<u>Principle of duality</u>

This principle states that any algebraic equality derived from these axioms will still be valid whenever the OR and AND operators, and identity elements 0 and 1, have been interchanged. i.e. changing every OR into AND and vice versa, and every 0 into 1 and vice versa.

Boolean Expressions and Their Corresponding Duals

Given Expression	Dual	Given Expression	Dual
0 = 1	1 = 0	A. (A+B) = A	A + A.B = A
0.1 = 0	1 + 0 = 1	AB = A + B	A+B = A.B
A.0 = 0	A + 1 = 1	(A+C) (A +B) = AB + AC	AC + AB = (A+B). (A+C)
A.B = B. A	A + B = B + A	A+B = AB + AB +AB	AB = (A+B).(A+B).(A+B)
A.A = 0	A + A = 1	AB + A + AB = 0	((A+B)).A.(A+B) = 1

<u>Theorems of Boolean Algebra</u>

Boolean algebraic theorems are the theorems that are used to change the form of a boolean expression. Sometimes these theorems are used to minimize the terms of the expression, and sometimes they are used just to transfer the expression from one form to another.

1. **De Morgan’s Theorem :**

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

Thus, the complement of the product of variables is equal to the sum of their individual complements.

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

Thus, the complement of the sum of variables is equal to the product of their individual complements.

The above two laws can be extended for n variables as

(A1 . A2 . A3 ... An)' = A1' + A2' + ... + An'   And   (A1 + A2 + ... + An)' = A1' . A2' . A3' ... An'

2. Transposition Theorem : It states that: AB + A'C = (A + C) (A' + B)

Proof: RHS = (A + C) (A' + B)

=AA' + A'C + AB + CB= 0 + A'C + AB + BC

=A'C + AB + BC(A + A') =AB + ABC + A'C + A'BC

= AB + A'C

= LHS

3. Redundancy Theorem :

This theorem is used to eliminate the redundant terms. A variable is associated with some variable and its complement is associated with some other variable and the next term is formed by the left over variables, then the term becomes redundant. Example: AB + BC' + AC = AC + BC'

Proof: LHS = AB + BC' + AC

= AB(C + C') + BC'(A + A') + AC(B + B') = ABC + ABC' + ABC' + A'BC' + ABC + AB'C

= ABC + ABC' + A'BC' + AB'C

= AC(B + B') + BC'(A + A')

= AC + BC'

= RHS

4. Duality Theorem : Dual expression is equivalent to write a negative logic of the given boolean relation. For this, Change each OR sign by and AND sign and vice-versa.,Complement any 0 or 1 appearing in the expression. Keep literals as it is. Example: Dual of A(B+C) = A+(B.C) = (A+B)(A+C)

5. Complementary Theorem : For obtaining complement expression, Change each OR sign by AND sign and vice-versa. Complement any 0 or 1 appearing in the expression. Complement the individual literals.

Example: Complement of A(B+C) = A'+(B'.C') = (A'+B')(A'+C')

Boolean functions

A Boolean function is a function that has n variables or entries, so it has 2^n possible combinations of the variables. These functions will assume only 0 or 1 in its output. An example of a Boolean function is this, f(a,b,c) = a . b + c

Truth table of the function

a	b	c	f(a,b,c)
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

A Boolean function is an expression formed with Binary variables, Operators (OR, AND, and NOT), Parentheses, and equal sign. The value of a Boolean function can be either 0 or 1.A Boolean function may be represented as: An algebraic expression, or a truth table.

Ex: **W=X + Y'.Z** Variable W is a function of X, Y, and Z, can also be written as W = f (X, Y, Z)

The RHS of the equation is called an ***expression.*** The symbols X, Y, Z are the ***literals*** of the function. For a given Boolean function, there may be more than one algebraic expressions.

X	Y	Z	W
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	1

Minimization of Boolean Functions

The process of simplifying the algebraic expression of a boolean function is called **minimization**. Minimization is important since it reduces the cost and complexity of the associated circuit. Minimization of Boolean functions deals with reduction in number of literals, reduction in number of terms.

F1= x' . y' . z + x' . y . z + x. y' F1 has 3 literals (x, y, z) and 3 terms

F2= x . y' + x' . z F2 has 3 literals (x, y, z) and 2 terms

F2 can be realized with fewer electronic components, resulting in a cheaper circuit.

X	Y	Z	F1	F2
0	0	0	0	0
0	0	1	1	1
0	1	0	0	0
0	1	1	1	1
1	0	0	1	1
1	0	1	1	1
1	1	0	0	0
1	1	1	0	0

Both F1 and F2 produce the same result

Ex : Minimize the following Boolean expression using Boolean identities −**F(A,B,C)=A′B+BC′+BC+AB′C′**

Solution :F(A,B,C)=A′B+BC′+BC+AB′C′

F(A,B,C)=A′B+(BC′+BC′)+BC+AB′C′ [By idempotent law, BC’ = BC’ + BC’]
F(A,B,C)=A′B+(BC′+BC)+(BC′+AB′C′)
F(A,B,C)=A′B+B(C′+C)+C′(B+AB′) [By distributive laws]
F(A,B,C)=A′B+B.1+C′(B+A) [(C' + C) = 1 and absorption law (B + AB')= (B + A)]
F(A,B,C)=A′B+B+C′(B+A) [B.1 = B]
F(A,B,C)=B(A′+1)+C′(B+A)
F(A,B,C)=B.1+C′(B+A) [(A' + 1) = 1]
F(A,B,C)=B+C′(B+A) [As, B.1 = B]
F(A,B,C)=B+BC′+AC′
F(A,B,C)=B(1+C′)+AC′
F(A,B,C)=B.1+AC′ [As, (1 + C') = 1]
F(A,B,C)=B+AC′ [As, B.1 = B]
So **F(A,B,C)=B+AC′** is the minimized form.

Ex: Minimize the following Boolean expression −F(A,B,C)=(A+B)(A+C)

Solution
Given, F(A,B,C)=(A+B)(A+C)
F(A,B,C)=A.A+A.C+B.A+B.C [Applying distributive Rule]
F(A,B,C)=A+A.C+B.A+B.C [Applying Idempotent Law]
F(A,B,C)=A(1+C)+B.A+B.C [Applying distributive Law]
F(A,B,C)=A+B.A+B.C [Applying dominance Law]
F(A,B,C)=A(1+B)+B.C [Applying distributive Law]
F(A,B,C)=A.1+B.C [Applying dominance Law]
F(A,B,C)=A+B.C
So F(A,B,C)=A+BC is the minimized form.

Complement of a Boolean Function

The complement of a Boolean function is obtained by interchanging: Operators OR with AND and Complementing each literal.This is based on ***De Morgan's theorems***, whose general form is:

$(A + A + A + ... + A_n)' = A'. A'. A'. ... A'_n$

$(A . A. A ... A_n)' = A' + A' + A' + ... + A'_n$

Ex : F1 = X’. Y. Z’ + X’. Y’. Z
F1’=(X’+Y+Z’) . (X’+Y’+Z)
F1’=(X+Y’+Z). (X+Y+Z’)

Canonical Form

Any Boolean function F( ) can be expressed as a *unique* **sum** of **min**terms and a unique **product** of **max**terms under a fixed variable ordering.In other words, every function F() has two canonical forms:

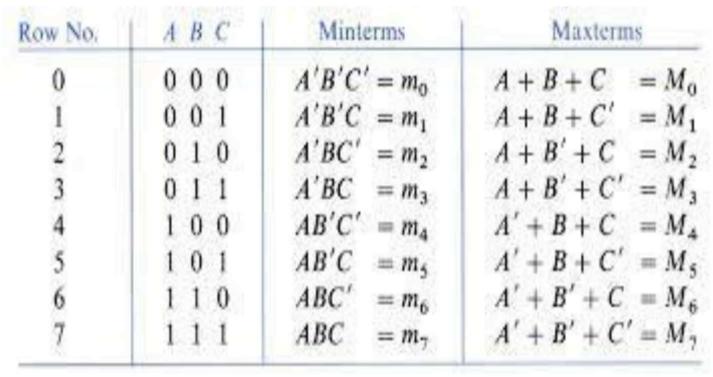
- Canonical Sum-Of-Products(sum of minterms)
- Canonical Product-Of-Sums(product of maxterms)

Canonical Sum-Of-Products:The minterms included,those m*j* such that F( ) = 1 in row *j* of the truth table for F( ).
Canonical Product-Of-Sums:The maxterms included,those M*j* such that F( ) =0 in row *j* of the truth table for F( ).

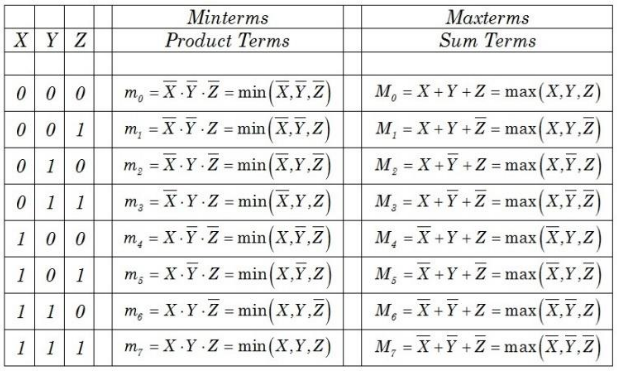
f1(a,b,c) = Σ m(1,2,4,6), where Σ indicates that this is a sum-of-products form, and m(1,2,4,6) indicates that the minterms to be included are m1, m2, m4, and m6.

f1(a,b,c) = Π M(0,3,5,7), where Π indicates that this is a product-of-sums form, and M(0,3,5,7) indicates that the maxterms to be included are M0, M3, M5, and M7.

Since mj = Mj' for any *j*, Σ m(1,2,4,6) = Π M(0,3,5,7) = f1(a,b,c)



Row No.	A B C	Minterms	Maxterms
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	$ABC = m_7$	$A'+B'+C' = M_7$



				Minterms	Maxterms
X	Y	Z		Product Terms	Sum Terms
0	0	0		$m_0 = \overline{X}\cdot\overline{Y}\cdot\overline{Z} = \min(\overline{X},\overline{Y},\overline{Z})$	$M_0 = X+Y+Z = \max(X,Y,Z)$
0	0	1		$m_1 = \overline{X}\cdot\overline{Y}\cdot Z = \min(\overline{X},\overline{Y},Z)$	$M_1 = X+Y+\overline{Z} = \max(X,Y,\overline{Z})$
0	1	0		$m_2 = \overline{X}\cdot Y\cdot\overline{Z} = \min(\overline{X},Y,\overline{Z})$	$M_2 = X+\overline{Y}+Z = \max(X,\overline{Y},Z)$
0	1	1		$m_3 = \overline{X}\cdot Y\cdot Z = \min(\overline{X},Y,Z)$	$M_3 = X+\overline{Y}+\overline{Z} = \max(X,\overline{Y},\overline{Z})$
1	0	0		$m_4 = X\cdot\overline{Y}\cdot\overline{Z} = \min(X,\overline{Y},\overline{Z})$	$M_4 = \overline{X}+Y+Z = \max(\overline{X},Y,Z)$
1	0	1		$m_5 = X\cdot\overline{Y}\cdot Z = \min(X,\overline{Y},Z)$	$M_5 = \overline{X}+Y+\overline{Z} = \max(\overline{X},Y,\overline{Z})$
1	1	0		$m_6 = X\cdot Y\cdot\overline{Z} = \min(X,Y,\overline{Z})$	$M_6 = \overline{X}+\overline{Y}+Z = \max(\overline{X},\overline{Y},Z)$
1	1	1		$m_7 = X\cdot Y\cdot Z = \min(X,Y,Z)$	$M_7 = \overline{X}+\overline{Y}+\overline{Z} = \max(\overline{X},\overline{Y},\overline{Z})$

Standard Forms

Standard forms are "like" canonical forms, except that not all variables need appear in the individual product (SOP) or sum (POS) terms.

Example: f1(a,b,c) = a'b'c + bc' + ac' is a standard sum-of-products form
f1(a,b,c) = (a+b+c)•(b'+c')•(a'+c') is a standard product-of-sums form.

Conversion of SOP from standard to canonical form :
Expand *non-canonical* terms by inserting equivalent of 1 in each missing variable x: (x + x') = 1.
Remove duplicate minterms
f1(a,b,c) = a'b'c + bc' + ac'
= a'b'c + (a+a')bc' + a(b+b')c'
= a'b'c + abc' + a'bc' + abc' + ab'c'
= a'b'c + abc' + a'bc + ab'c'

Conversion of POS from standard to canonical form :
Expand noncanonical terms by adding 0 in terms of missing variables (e.g., xx' = 0) and using the distributive law.Remove duplicate maxterms
f1(a,b,c) = (a+b+c)•(b'+c')•(a'+c')
= (a+b+c)•(aa'+b'+c')•(a'+bb'+c')
= (a+b+c)•(a+b'+c')•(a'+b'+c')•(a'+b+c')•(a'+b'+c')
= (a+b+c)•(a+b'+c')•(a'+b'+c')•(a'+b+c')

Example – Express the Boolean function **F = A + B'C** as standard sum of minterms.

A = A(B + B') = AB + AB' This function is still missing one variable, so
A = AB(C + C') + AB'(C + C') = ABC + ABC'+ AB'C + AB'C'
B'C = B'C(A + A') = AB'C + A'B'C
F = A + B'C = ABC + ABC' + AB'C + AB'C' + AB'C + A'B'C
F = A'B'C + AB'C' + AB'C + ABC' + ABC
F = m1 + m4 + m5 + m6 + m7 SOP is represented as Σ (1, 4, 5, 6, 7)

Conversion Between Canonical Forms

For converting the canonical expressions, we have to change the symbols ∏, ∑. These symbols are changed when we list out the index numbers of the equations. From the original form of the equation, these indices numbers are excluded. The SOP and POS forms of the boolean function are duals to each other.
There are the following steps using which we can easily convert the canonical forms of the equations:

1. Change the operational symbols used in the equation, such as ∑, ∏.
2. Use the Duality's De-Morgan's principal to write the indexes of the terms that are not presented in the given form of an equation or the index numbers of the Boolean function.

Conversion of POS to SOP form :
There are the following steps to convert the POS function F = Π x, y, z (2, 3, 5) = x y' z' + x y' z + x y z' into SOP form:

1. In the first step, we change the operational sign to Σ.
2. Next, we find the missing indexes of the terms, 000, 110, 001, 100, and 111.
3. Finally, we write the product form of the noted terms.

000 = x' * y' * z'   001 = x' * y' * z   100 = x * y' * z'   110 = x * y* z'   111 = x * y * z

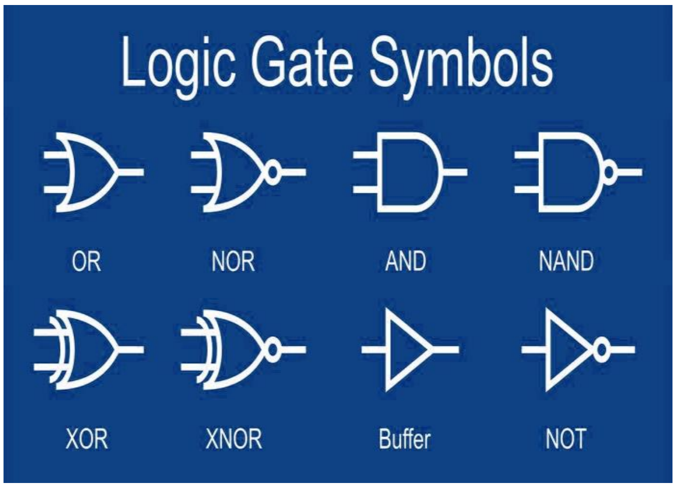
POS function F = Π x, y, z (2, 3, 5) = x y' z' + x y' z + x y z'
So the SOP form is: F = Σ x, y, z (0, 1, 4, 6, 7) = (x' * y' * z') + (x' * y' * z) + (x * y' * z') + (x * y* z') + (x * y * z)
Conversion of SOP form to POS form :
There are the following steps used to convert SOP function F = ∑ x, y, z (0, 2, 3, 5, 7) = x' y' z' + z y' z' + x y' z + xyz' + xyz into POS:

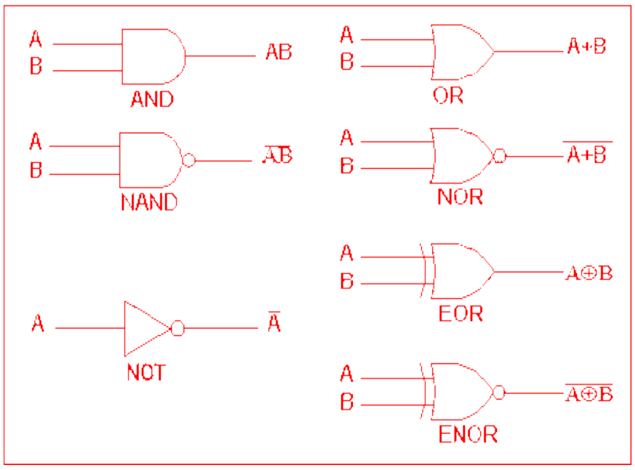
- In the first step, we change the operational sign to ∏.
- We find the missing indexes of the terms, 001, 110, and 100.
- We write the sum form of the noted terms.

001 = (x + y + z)   100 = (x + y' + z')   110 = (x + y' + z')

SOP function F = ∑ x, y, z (0, 2, 3, 5, 7) = x' y' z' + z y' z' + x y' z + xyz' + xyz
So, the POS form is: F = Π x, y, z (1, 4, 6) = (x + y + z) * (x + y' + z') * (x + y' + z')

Logic Gates





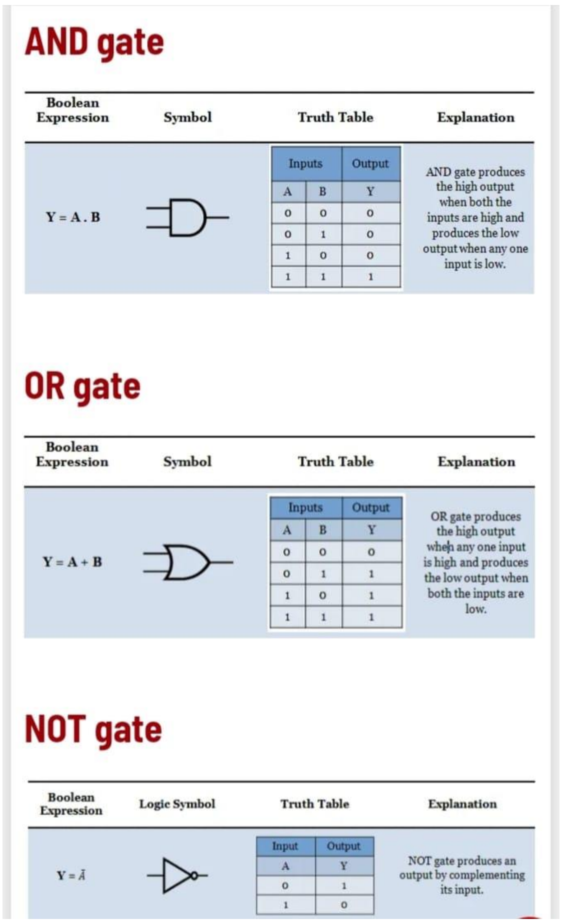
Logic gates are the basic building blocks of any digital system. Generally, a digital circuit will consists of number of logic gates, which receive the binary inputs and perform a logical operations like AND, OR and NOT. There are two states in digital logic circuit design – Logic ‘0’ and Logic ‘1’ or FALSE and TRUE or LOW and HIGH value.

These states can be explained with a simple lamp switch. The lamp has only two states in its operation – either be turned ON or turned OFF. The ON state represents the HIGH voltage and the OFF state represents the LOW voltage value.

In the same way, the state of the digital system is defined. Logic 1 represents HIGH or TRUE output and Logic 0 represents LOW or FALSE output.

Different types of Logic Gates

The available gates are AND, OR, NOT, NOR, NAND, Ex-OR and Ex-NOR gate. Among which AND, OR, NOT gate are the basic gates, whereas NAND and NOR gate are the universal gates. Let us take a look at all the gates in detail below.



AND gate

Boolean Expression: $Y = A . B$ | Symbol | Truth Table | Explanation

Inputs		Output
A	B	Y
0	0	0
0	1	0
1	0	0
1	1	1

AND gate produces the high output when both the inputs are high and produces the low output when any one input is low.

OR gate

Boolean Expression: $Y = A + B$ | Symbol | Truth Table | Explanation

Inputs		Output
A	B	Y
0	0	0
0	1	1
1	0	1
1	1	1

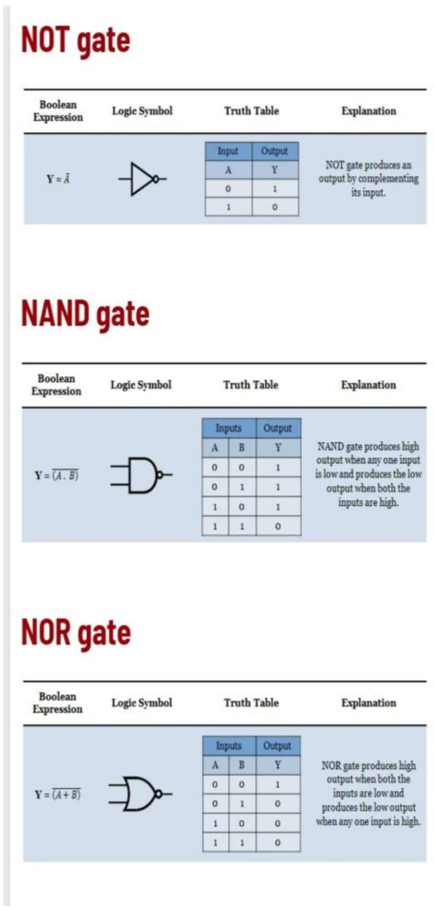
OR gate produces the high output when any one input is high and produces the low output when both the inputs are low.

NOT gate

Boolean Expression: $Y = \bar{A}$ | Logic Symbol | Truth Table | Explanation

Input	Output
A	Y
0	1
1	0

NOT gate produces an output by complementing its input.



NOT gate

Boolean Expression: $Y = \bar{A}$ | Logic Symbol | Truth Table | Explanation

Input	Output
A	Y
0	1
1	0

NOT gate produces an output by complementing its input.

NAND gate

Boolean Expression: $Y = \overline{(A . B)}$ | Logic Symbol | Truth Table | Explanation

Inputs		Output
A	B	Y
0	0	1
0	1	1
1	0	1
1	1	0

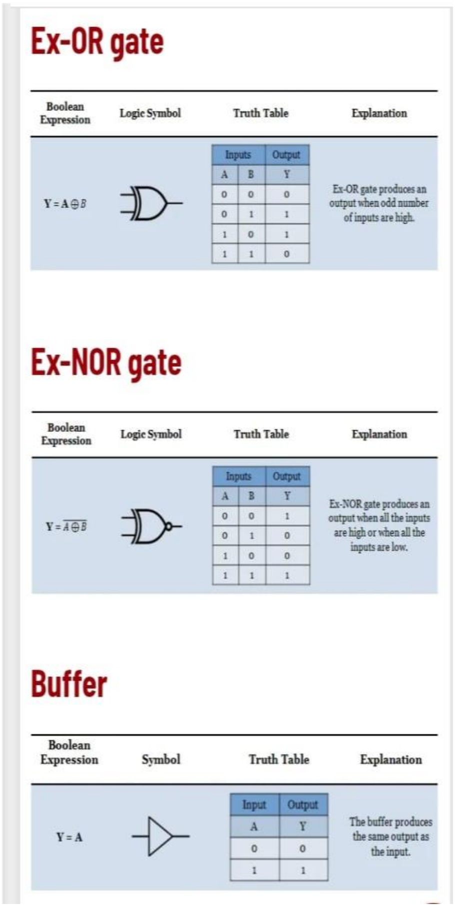
NAND gate produces high output when any one input is low and produces the low output when both the inputs are high.

NOR gate

Boolean Expression: $Y = \overline{(A + B)}$ | Logic Symbol | Truth Table | Explanation

Inputs		Output
A	B	Y
0	0	1
0	1	0
1	0	0
1	1	0

NOR gate produces high output when both the inputs are low and produces the low output when any one input is high.



Ex-OR gate

Boolean Expression: $Y = A \oplus B$ | Logic Symbol | Truth Table | Explanation

Inputs		Output
A	B	Y
0	0	0
0	1	1
1	0	1
1	1	0

Ex-OR gate produces an output when odd number of inputs are high.

Ex-NOR gate

Boolean Expression: $Y = \overline{A \oplus B}$ | Logic Symbol | Truth Table | Explanation

Inputs		Output
A	B	Y
0	0	1
0	1	0
1	0	0
1	1	1

Ex-NOR gate produces an output when all the inputs are high or when all the inputs are low.

Buffer

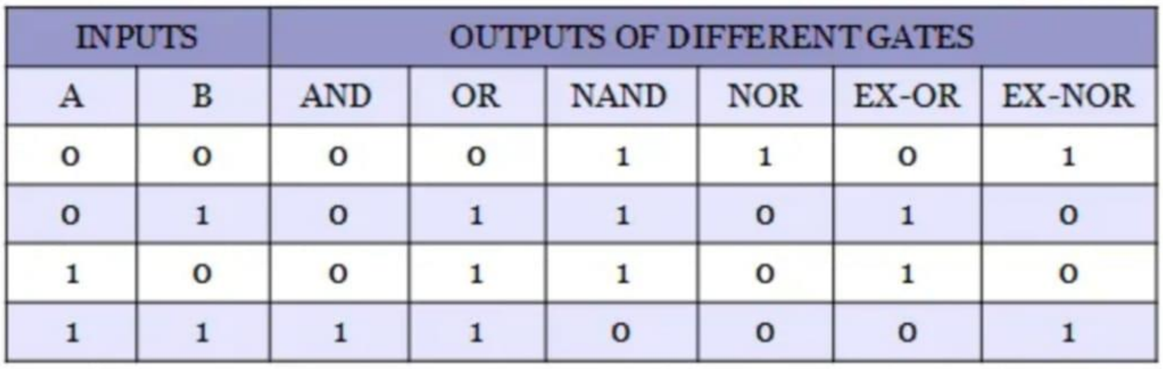
Boolean Expression: $Y = A$ | Symbol | Truth Table | Explanation

Input	Output
A	Y
0	0
1	1

The buffer produces the same output as the input.

Comparison of Truth table of different logic gates

For your clear understanding, the below table is given, which shows the comparison between different gates.



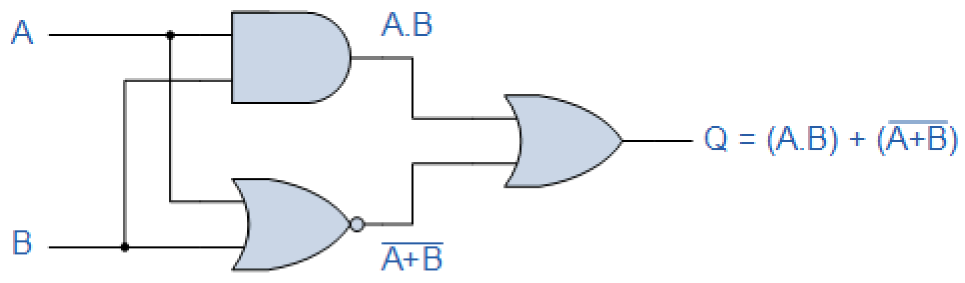
INPUTS		OUTPUTS OF DIFFERENT GATES					
A	B	AND	OR	NAND	NOR	EX-OR	EX-NOR
0	0	0	0	1	1	0	1
0	1	0	1	1	0	1	0
1	0	0	1	1	0	1	0
1	1	1	1	0	0	0	1

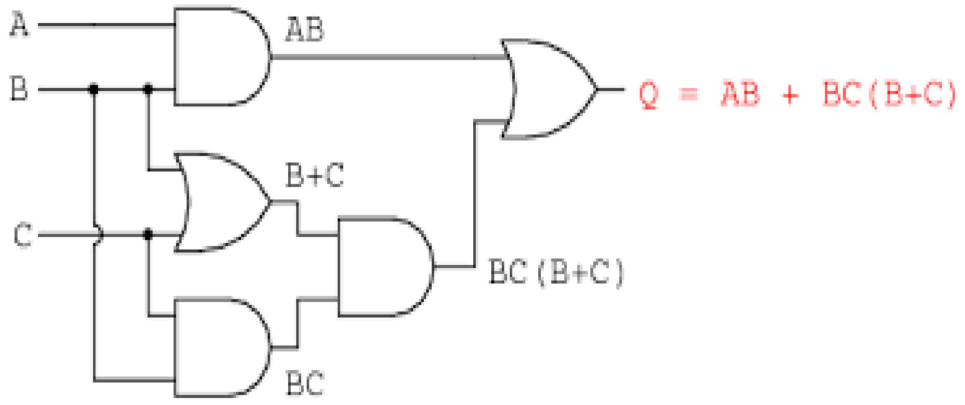
Truth table for AND, OR, NAND, NOR, Ex-OR and Ex-NOR gates

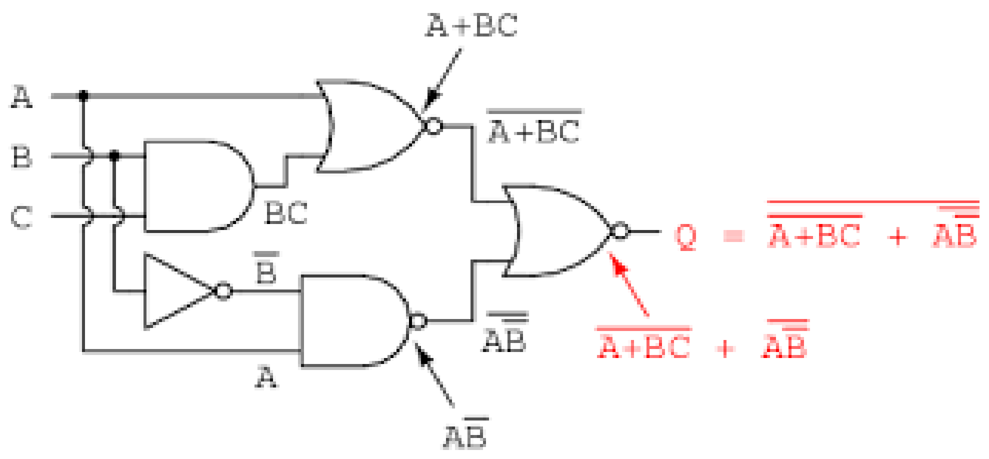
Logic Circuits

A digital circuit is typically constructed from small electronic circuits called logic gates that can be used to create combinational logic. Each logic gate is designed to perform a function of boolean logic when acting on logic signals.

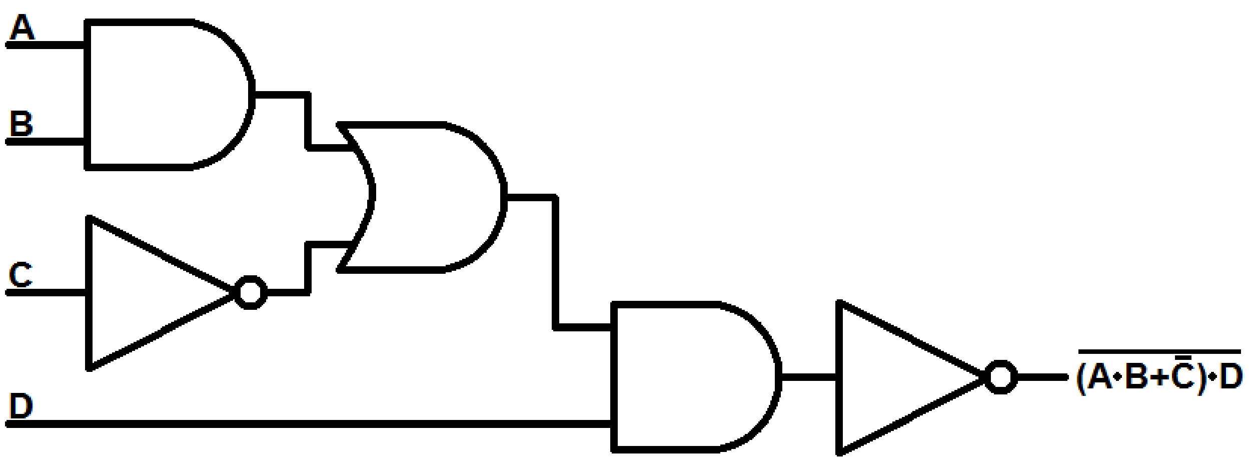
- When logic gates are interconnected to form a gating / logic network, it is known as a *combinational logic circuit*
- The Boolean algebra expression for a given logic circuit can be derived by systematically progressing from input to output on the gates
- The three logic gates (AND, OR, and NOT) are logically complete because any Boolean expression can be realized as a logic circuit using only these three gates



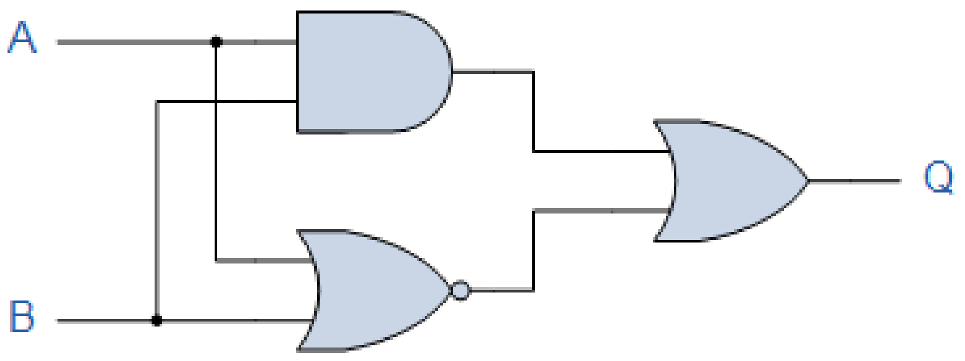




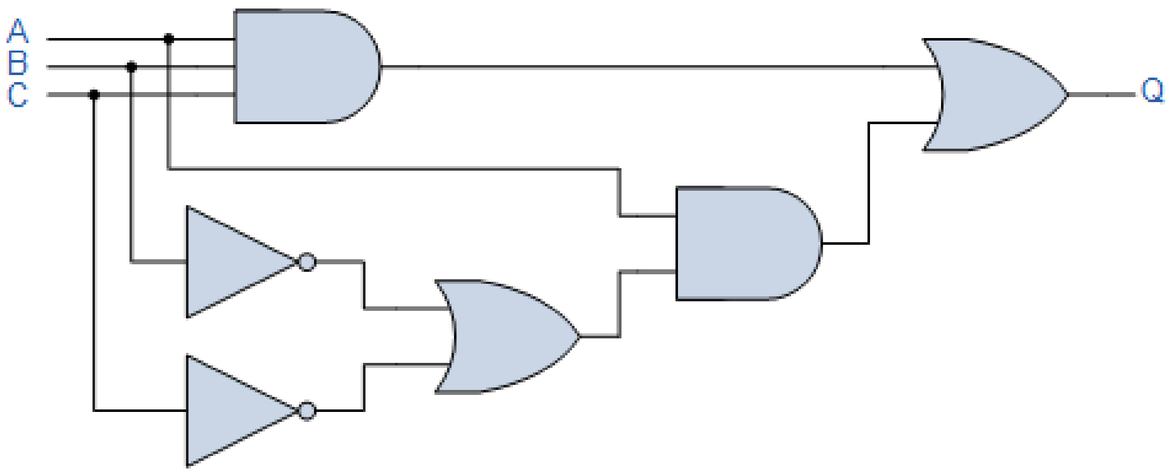
A
B
C
A+BC
$\overline{A+BC}$
BC
$\overline{B}$
A
$\overline{A\overline{B}}$
$A\overline{B}$
Q = $\overline{\overline{A+BC} + \overline{A\overline{B}}}$
$\overline{A+BC} + \overline{A\overline{B}}$



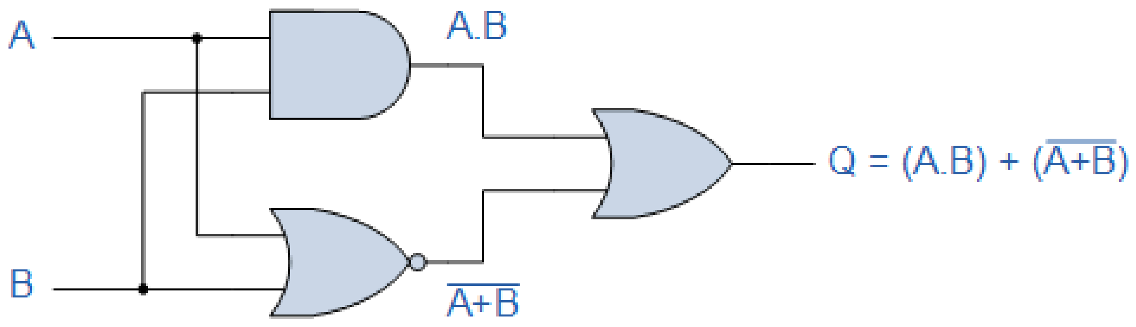
A
B
C
D
$\overline{(A \cdot B + \bar{C}) \cdot D}$

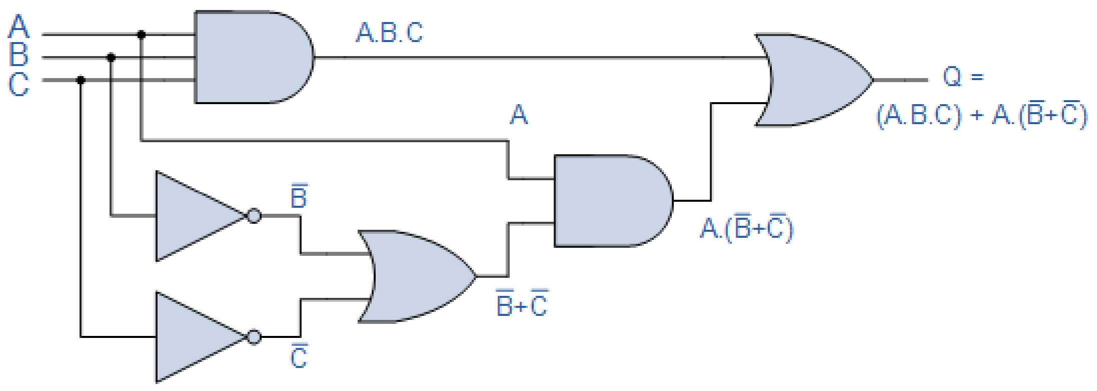


A
B
Q

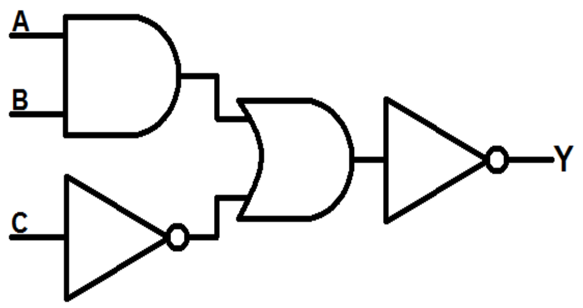


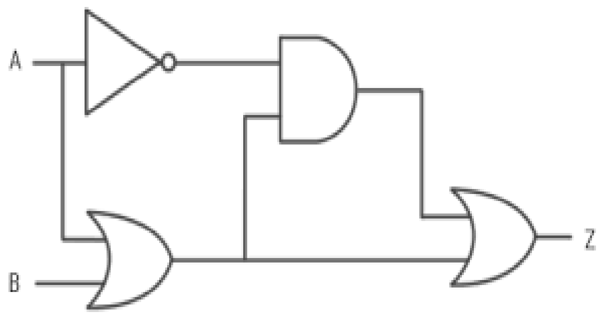
A
B
C
Q

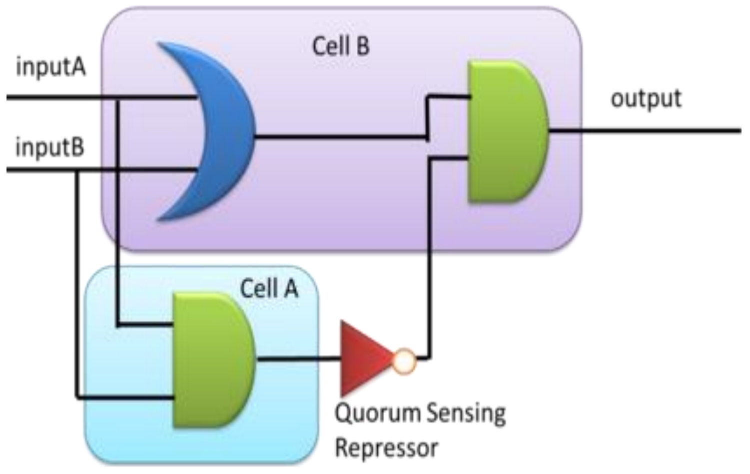

A
B
A.B
$\overline{A+B}$
$Q = (A.B) + (\overline{A+B})$

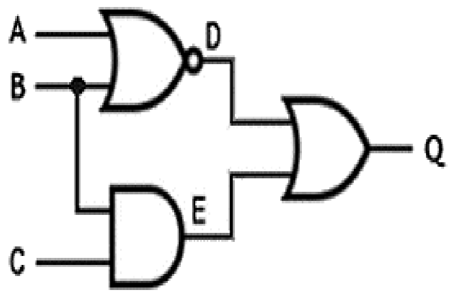

A
B
C
A.B.C
A
$\overline{B}$
$\overline{C}$
$\overline{B}+\overline{C}$
$A.(\overline{B}+\overline{C})$
Q =
$(A.B.C) + A.(\overline{B}+\overline{C})$

Home Work :


A
B
C
Y


A
B
Z


inputA
inputB
Cell B
output
Cell A
Quorum Sensing
Repressor


A
B
C
D
E
Q

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.

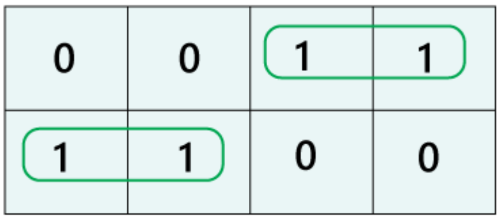
Minterm Solution of K Map

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

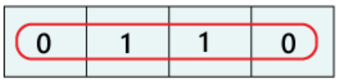
Step 1: Firstly, we define the given expression in its canonical form.

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.

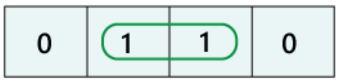
Step 3: Next, we form the groups by considering each one in the K-map.



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



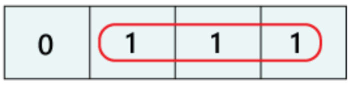
Incorrect



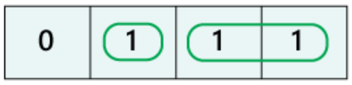
Correct

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 $2^4=16$.

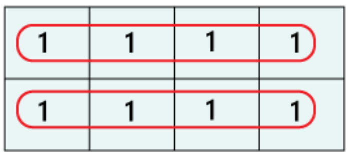


Incorrect

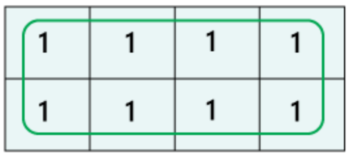


Correct

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.

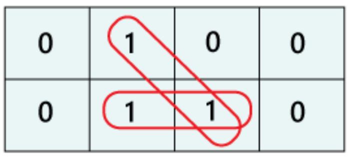


Incorrect

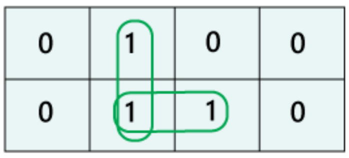


Correct

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.

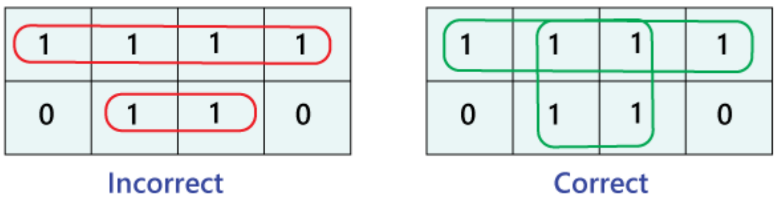


Incorrect

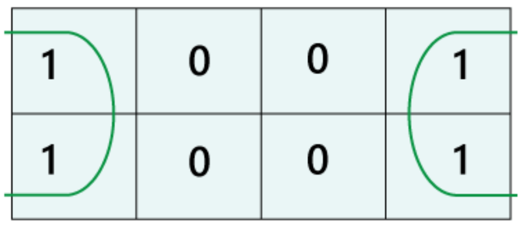


Correct

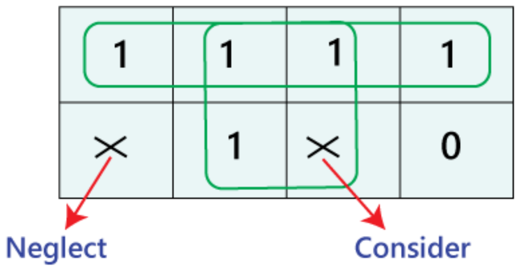
The elements in one group can also be used in different groups only when the size of the group is increased.



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



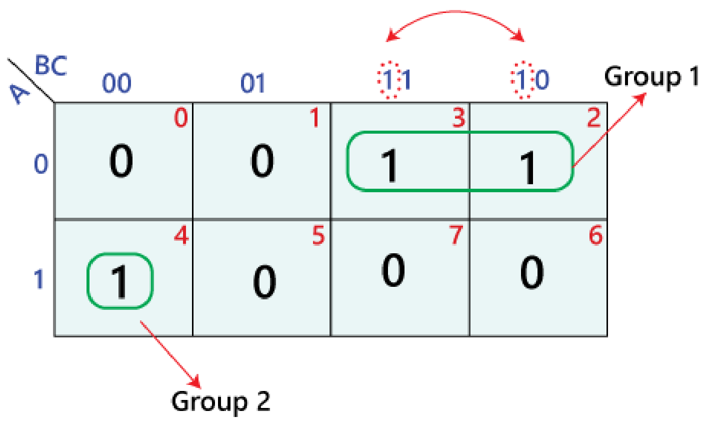
We can consider the 'don't care condition' only when they aid in increasing the group-size. Otherwise, 'don't care' elements are discarded.



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'.

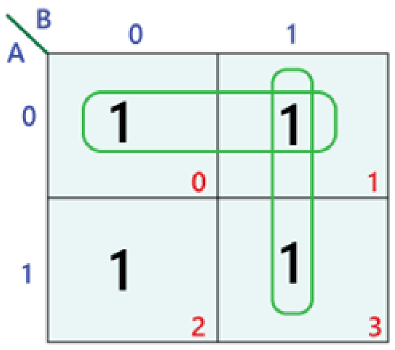


Lastly, we find the boolean expression for the Output. To find the simplified boolean expression in the SOP form, we combine the product-terms of all individual groups. So the simplified expression of the above k-map is as follows:

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

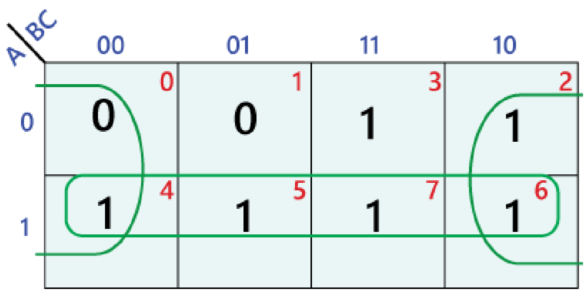
Let's take some examples of 2-variable, 3-variable, 4-variable, and 5-variable K-map examples.

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



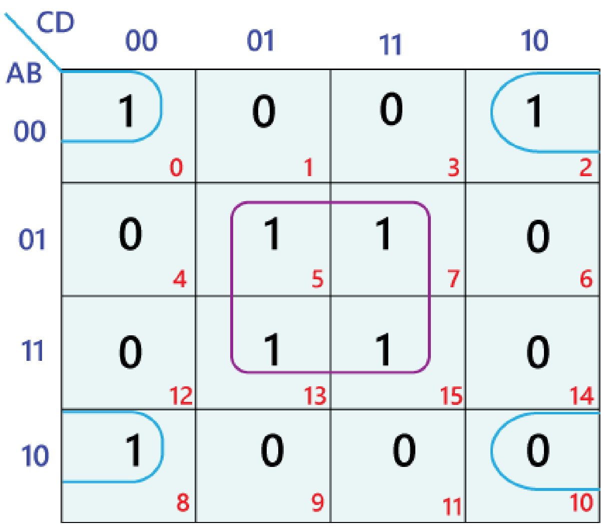
Simplified expression: Y=A'+B

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

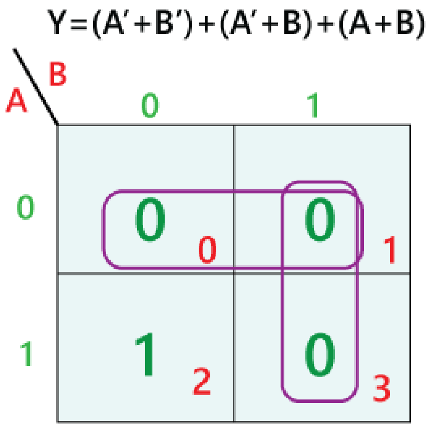
To find the simplified maxterm solution using K-map is the same as to find for the minterm solution. There are some minor changes in the maxterm solution, which are as follows:

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.

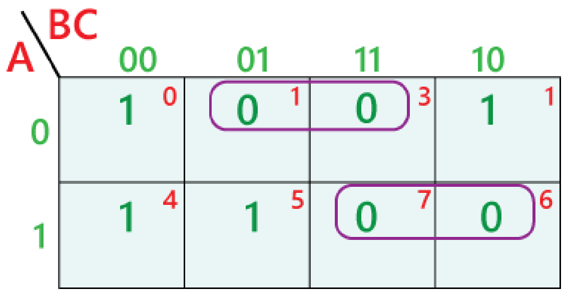
Let's take some example of 2-variable, 3-variable, 4-variable and 5-variable K-map examples

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

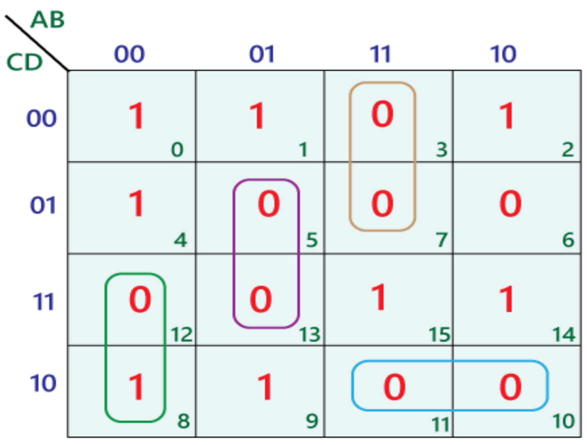


Simplified expression: A'B

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)



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

****