Fundamental of Computer

Number System

(1) Binary number system (0, 1)
(2) Octal number system (0, 1, 2, 3, 4, 5, 6, 7)
(3) Decimal number system (0, 1, 2, 3, 4, 5, 6, 7, 8, 9)
(4) Hexa - Decimal number system (0, 1, 2, 3, 4, 5, 6, 7, 8, 9)
(A B C D E F)

UTF-8- Unicode transmission format - 8

Base/Radix ⟶
Binary - 2
Octal - 8
Decimal - 10
Hexa - Decimal - 16

Conversion

(1) Decimal to Binary

$(15)_{10} = (?)_2$

	Remainder
2 | 15	1
2 | 7	1
2 | 3	1
1	

$(1111)_2$

$(6)_{10} = (?)_2$

	Remainder
2 | 6	0
2 | 3	1
1	

$(110)_2$

$(101010)_2$
MSB — most significant bit
LSB — least significant bit

Binary point
(110.10)
MSB LSB

$(97)_{10} = (?)_2$

		Reminder
2	97	1
2	48	0
2	24	0
2	12	0
2	6	0
2	3	1
	1	

$(1100001)_2$

Number System

* A number system can be considered as a mathematical notation of numbers using a set of digits or symbols.
* Every number system is identified with the help of its base or radix.
* Number system helps to count or measure objects and it helps in performing various mathematical calculations.

Base or Radix

* The base or radix of a number system can be referred to as the total number of different symbols which can be used in a particular number system.
* Radix means "root" in Latin.

* Base equals to 4 implies there are 4 different symbols in that number system.
* Similarly, base equals to "x" implies there are "x" different symbols in that number system.

Binary Number System

* The binary number system uses only two digits: 0 and 1. The numbers in this system have a base of 2.
* Digits 0 and 1 are called bits and 8 bits together make a byte. The data in computers is stored in terms of bits and bytes.
* Binary number system is very useful in electronic devices and computer systems.
* Example: $(1101.01)_2$

Octal Number System

* The octal number system uses eight digits 0, 1, 2, 3, 4, 5, 6 and 7 with the base of 8.
* Just like the binary, the octal number system is used in minicomputers but with digits from 0 to 7.
* Octal numbers are used for the representation of UTF8 numbers.
* Example: $(67.3)_8$

Decimal Number System

* The decimal number system uses ten digits: 0, 1, 2, 3, 4, 5, 6, 7, 8 and 9 with the base number as 10.
* The decimal number system is the system that we generally use to represent numbers in real life and mathematics etc.
* Example: $(98.7)_{10}$

Hexadecimal Number System

* The hexadecimal number system uses sixteen digits / alphabets: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9 and A, B, C, D, E, F with the base number as 16. Here, A-F of the hexadecimal system means the numbers 10-15 of the decimal number system respectively.
* Hexadecimal numbers are useful for handling memory address locations.

Decimal to Binary.

$(11.25)_{10} = (?)_2$

2 | 11	1	
2 | 5	1	$(11)_{10} = (1011)_2$
2 | 2	0	
1		

Solved

$(0.25)_{10} = (?)_2$

$0.25 \times 2 = 0.5$ 0 ↓

$0.5 \times 2 = 1.0$ 1 ↓

$(0.25)_{10} = (0.01)_2$

$(11.25)_{10} = (1011.01)_2$ Ans.

$(250.4375)_{10} = (?)_2$

2 | 250	0	
2 | 125	1	
2 | 62	0	$(250)_{10} = (11111010)_2$
2 | 31	1	
2 | 15	1	
2 | 7	1	
2 | 3	1	
1		

$(0.4375) \times 2 = 0.875$ 0 ↓

$0.875 \times 2 = 1.75$ 1 ↓

$0.75 \times 2 = 1.5$ 1

$0.5 \times 2 = 1.0$ 1

$(0.4375)_{10} = (0.0111)_2$

$\therefore (250.4375) = (11111010.0111)_2$ Ans.

$(7.43)_{10} = (?)_2$

2 | 7	1	
2 | 3	1	
1		

$(7)_{10} = (111)_2$

$(0.43)_{10} = (?)_2$

0.43×2	$= 0.86$	0
0.86×2	$= 1.72$	1
0.72×2	$= 1.44$	1
0.44×2	$= 0.88$	0
0.88×2	$= 1.76$	1
0.76×2	$= 1.52$	1
0.52×2	$= 1.04$	1
0.04×2	$= 0.08$	0
0.08×2	$= 0.16$	0
0.16×2	$= 0.32$	0
0.32×2	$= 0.64$	0
0.64×2	$= 1.28$	1
0.28×2	$= 0.56$	0
0.56×2	$= 1.12$	1
0.12×2	$= 0.24$	0
0.24×2	$= 0.48$	0
0.48×2	$= 0.96$	0
0.96×2	$= 1.92$	1
0.92×2	$= 1.84$	1
0.84×2	$= 1.68$	1
0.68×2	$= 1.36$	1
0.36×2	$= 0.72$	0

$(7.43)_{10} = (111.0110111000010100011110)_2$

Note:- To convert decimal to binary, divide integer part by 2 and multiply fractional part by 2.

#1 Decimal to octal Conversion:-

$(16)_{10} = (?)_8$

8 | 16 — 0
2

$(16)_{10} = (20)_8$

Q $(250.4375)_{10} \rightarrow (?)_8$

Sol:- $(250)_{10} = (?)_8$

8 | 250 — 2
8 | 31 — 6
3

$(250)_{10} = (362)_8$

$(0.4375)_{10} = (?)_8$

$0.4375 \times 8 = 3.5000$ ↓ 3

$0.5000 \times 8 = 4.0080$ ↓ 4

$0.0080 \times 8 = 0.0640$ ↓ 0

$(0.4375)_{10} = (0.064)_8$

$(362.34)_8$

$\therefore (250.4375)_{10} = (362.064)_8$ Ans

To convert decimal to hexa decimal ... integer part by 16 and multiply fra... part by 16.

Decimal to hexodecimal

Q. $(250.4375)_{10} = (?)_{16}$

Sol:- $(250)_{10} = (?)_{16}$

$(250)_{10}$

16 | 250 — 10
15

$(250)_{10} = (FA)_{16}$

$(0.4375)_{10} = (?)_{16}$

$0.4375 \times 16 = 7.0$ — 7

$(250.4375)_{10} = (FA.7)_{16}$

Q $(160.75)_{10} = (?)_2 = (?)_8 = (?)_{16}$.

Sol $(160)_{10} = (?)_2$

2	160	0
2	80	0
2	40	0
2	20	0
2	10	0
2	5	1
2	2	0
	1	

$(160)_{10} = (10100000)_2$

$(0.75)_{10} = (?)_2$

$0.75 \times 2 = 1.50 \quad 1$

$0.5 \times 2 = 1.0 \quad 1$

$(0.75)_{10} = (11)_2$

$\therefore (160.75)_{10} = (10100000.11)_2$. Ans

(i) $(160)_{10} = (?)_8$

$8 \mid 160 \quad 0$

$8 \mid 20 \quad 4$

2

$(160)_{10} = (240)_8$

and. $(0.75)_{10} = (?)_8$

$0.75 \times 8 = 6.00$

$\therefore (160.75)_{10} = (240.6)_8$. Ans

(ii) $(160)_{10} = (?)_{16}$

$16 \mid 160 \quad 0$

$10 \quad A$

$(160)_{10} = (A0)_{16}$.

$(0.75)_{10} = (?)_{16}$

$(0.75) \times 16 = 12.00 \quad 12$

$\therefore = C$

$\therefore (160.75)_{10} = (A0.C)_{16}$. Ans

Integer

LHS $(\quad)_{10}$ → $(?)_2$ / 2
→ $(?)_8$ / 8
→ $(?)_{16}$ / 16

LHS $(\cdot\ -)_{10}$ → $(?)_2$ × 2
→ $(?)_8$ × 8
→ $(?)_{16}$ × 16.

* Binary to Decimal

$(1011)_2 = (?)_{10}$

1 0 1 1

$1 \times 2^0 = 1 \times 1 = 1$
$1 \times 2^1 = 1 \times 2 = 2$
$0 \times 2^2 = 0 \times 4 = 0$
$1 \times 2^3 = 1 \times 8 = 8$

11

$(1011)_2 = (11)_{10}$ ✓

$(110001)_2 = (?)_{10}$

1 1 0 0 0 1

$1 \times 2^0 = 1 \times 1 = 1$
$0 \times 2^1 = 0 \times 2 = 0$
$0 \times 2^2 = 0 \times 4 = 0$
$0 \times 2^3 = 0 \times 8 = 0$
$1 \times 2^4 = 1 \times 16 = 16$
$1 \times 2^5 = 1 \times 32 = 32$

49

$(110001)_2 = (49)_{10}$ ✓

add
1 + 2 = 3.
Concate / append
(1) + (2) = 12

$(110.101)_2 = (?)_{10}$

110
$\hookrightarrow 0 \times 2^0 = 0 \times 1 = 0$
$1 \times 2^1 = 1 \times 2 = 2$
$1 \times 2^2 = 1 \times 4 = 4$
6

.101
$1 \times 2^{-1} = 1 \times 1/2 = 0.5$
$0 \times 2^{-2} = 0 \times 1/4 = 0$
$1 \times 2^{-3} = 1 \times 1/8 = 0.125$
0.625

$\therefore (110.101)_2 = (6.625)_{10}$ Ans.

Q. $(10001.1101)_2 = (?)_{10}$

10001
$1 \times 2^0 = 1 \times 1 = 1$
$0 \times 2^1 = 0 \times 2 = 0$
$0 \times 2^2 = 0 \times 4 = 0$
$0 \times 2^3 = 0 \times 8 = 0$
$1 \times 2^4 = 1 \times 16 = 16$
17

.1101
$1 \times 2^{-1} = 1 \times 1/2 = 0.5$
$1 \times 2^{-2} = 1 \times 1/4 = 0.25$
$0 \times 2^{-3} = 0 \times 1/8 = 0$
$1 \times 2^{-4} = 1 \times 1/16 = 0.0625$
0.8125

$\therefore (10001.1101)_2 = (17.8125)_{10}$ Ans.

* Octal to Decimal.

$(25.12)_8 = (?)_{10}$

$(25)_8 = (?)_{10}$

25

$5 \times 8^0 = 5 \times 1 = 5$

$2 \times 8^1 = 2 \times 8 = 16$

21

$(.12)_8 = (?)_{10}$

$1 \times 8^{-1} = 1 \times 1/8 = 0.125$

$2 \times 8^{-2} = 2 \times 1/64 = 0.3125$

0.15625

$(25.12)_8 = (21.15625)_{10}$

* Hexadecimal to Decimal.

$(1ACF.12)_{16} = (?)_{10}$

1 A C F

1 10 12 15

$15 \times 16^0 = 15 \times 1 = 15$

$12 \times 16^1 = 12 \times 16 = 192$

$10 \times 16^2 = 10 \times 256 = 2560$

$1 \times 16^3 = 1 \times 4096 = 4096$

6863

$(1ACF)_{16} = (6863)_{10}$

12

$1 \times 16^{-1} = 1 \times 1/16 = 0.0625$

$2 \times 16^{-2} = 2 \times 1/256 = 0.0078125$

0.0703125

$\therefore (1ACF.12)_{16} = (6863.0703125)_{10}$ Ans

(ii) $(80.01)_2 = (?)_{16}$

80

$2 \times 0 = 0 = 0$

$8 \times 2^1 = 2 \times 8 = 16$

16

Invalid

.01

$0 \times 2^{-1} = 0/2 = 0$

$1 \times 2^{-2} = 1 \times 1/4 = 0.25$

0.25

$\therefore (80.01)_2 = (16.25)_{16}$ Ans

Q. $(80.01)_{16} = (?)_{10}$

80

$0 \times 16^0 = 0 \times 1 = 0$

$8 \times 16^1 = 8 \times 16 = 128$

128

.01

$0 \times 16^{-1} = 0 \times 1/16 = 0$

$1 \times 16^{-2} = 1 \times 1/256 = 1/256 = 0.00390625$

$\therefore (80.01)_{16} = (128.00390625)_{10}$ Ans

Q. $(A01.12)_{16} = (?)_{10}$

Sol:-

A 0 1

10 0 1

$1 \times 16^0 = 1 \times 1 = 1$

$0 \times 16^1 = 0 \times 16 = 0$

$10 \times 16^2 = 10 \times 256 = 2560$

2561

.12

$1 \times 16^{-1} = 1 \times 1/16 = 0.0625$

$2 \times 16^{-2} = 2 \times 1/256 = 0.0078125$

0.0703125

$\therefore (A01.12)_{16} = (2561.0703125)_{10}$ Ans

Q. $(101.101)_2 = (?)_{10}$

Sol:-

101

$1 \times 2^0 = 1 \times 1 = 1$

$0 \times 2^1 = 0 \times 2 = 0$

$1 \times 2^2 = 1 \times 4 = 4$

5

.101

$1 \times 2^{-1} = 1 \times 1/2 = 0.5$

$0 \times 2^{-2} = 0 \times 1/4 = 0$

$1 \times 2^{-3} = 1 \times 1/8 = 0.125$

0.625

$\therefore (101.101)_2 = (5.625)_{10}$ Ans

* Octal to Binary

Method 1. It Convert octal number system to decimal number system. then decimal no. system Convert to binary number system. (Indirect)

2. Direct method.

Octal to Binary table :→

Octal N.S	Binary N.S
0	0 0 0
1	0 0 1
2	0 1 0
3	0 1 1
4	1 0 0
5	1 0 1
6	1 1 0
7	1 1 1

2^2 2^1 2^0
4 2 1

0 0 0 => 0
0 0 1 => 1
0 1 0 => 2
0 1 1 = 3
1 0 0 = 4
1 0 1 = 5
1 1 0 = 6
1 1 1 = 7

Hexadecimal to Binary table

Hexa N.S	Binary
0	0000
1	0001
2	0010
3	0011
4	0100
5	0101
6	0110

2^3 2^2 2^1 2^0
8 4 2 1

0 0 0 0 = 0
0 0 0 1 = 1
0 0 1 0 = 2
0 0 1 1 = 3
0 1 0 0 = 4
0 1 0 1 = 5
0 1 1 0 = 6
0 1 1 1 = 7
1 0 0 0 = 8
1 0 0 1 = 9.

7	0111
8	1000
9	1001
A	1010
B	1011
C	1101
D	1101
E	1110
F	1111

8 4 2 1

1010 = 10 A

1011 = 11 B

1101 = 12 C

1101 = 13 D

1110 = 14 E

1111 = 15 F

Octal to Binary 2^3 = bit

$(6)_8 = (?)_2$

6

110

$(110)_2$

$(25)_8 = (?)$

$\Rightarrow$ 2 5

010 101

$(010101)_2$ Ans

$(521)_8 = (?)_2$

5 2 1

101 010 001

$(101.010001)_2$ Ans

$(73.61)_8 = (?)_2$

7 3 . 6 1

111 011 110 001

$(111011.110001)_2$ Ans

$(56.16)_8 = (?)_2$

5 6 . 1 6

101 110 001 110

$(101110.001110)_2$

$\Rightarrow (101110.00111)_2$ Ans

⇒ Binary to Octal

$(110101)_2 = (?)_8$

$(110101)_2 \leftarrow$ R to Left

110	101
6	5

$(65)_8$ Ans

$(11001)_2 = (?)_8$

011	001
3	1

$(31)_8$ Ans

$(11010101.1001011)_2 = (?)_8$

Right to left Left to Right

011	010	101	.	100	101	100
3	2	5		4	5	4

$(325.454)_8$ Ans

Hexadecimal to Binary

$(A1)_{16} = (?)_2$

1010 0001

$(10100001)_2$ Ans

$(BCD.17)_{16} = (?)_2$

1011 1100 1101 . 0001 0111

$(101111001101.00010111)_2$ Ans

Binary to Hexadecimal.

$(1111001.101100)_2 = (?)_{16}$

0111	1001	.	1011	0000
7	9		B	0

$\therefore (79.B0)_{16}$ Ans

Hexadecimal to octal

step 1 → Hexa → binary
step 2 → Binary → Octal

$(AB.23)_{16}$

$(\boxed{010}\boxed{101}\boxed{011}.\boxed{001}\boxed{000}\boxed{110})_2$

$(2\ 53.106)_8$

Octal to Hexa.

$(76.12)_8 = (?)_{16}$

$(111110.001010)_2$

$\boxed{0011}\ \boxed{1110}\ .\ \boxed{0010}\ \boxed{1000}$

3 14 2 8

$(3E.28)_{16}$ A

Q.1 $(10110.11111)_2 = (?)_{16} = (?)_8$

Q.2 $(AB.CD)_{16} = (?)_2 = (?)_8$

Q1. Solution :- I Binary to hexadecimal.

$(10110.11111)_2 = (?)_{16}$.

$\boxed{0\,0\,0\,1\,0}\boxed{1\,1\,0}\ .\ \boxed{1\,1\,1\,1}\boxed{1\,0\,0\,0}$

↑ ↑ ↑ ↑

? 6 ? ?

$(?6.F?)_{16}$ A

Date
Page

Q1 $(10110.11111)_2 = (?)_8$

$\Rightarrow$ [01 0] [110] . [111] [110]

2 6 7 6

$\therefore (10110.11111)_2 = (26.76)_8$ Ans

Q2. $(AB.CD)_{16} = (?)_2$

$= A\ B\ .\ C\ D$

$10\ 11\ 12\ 13$

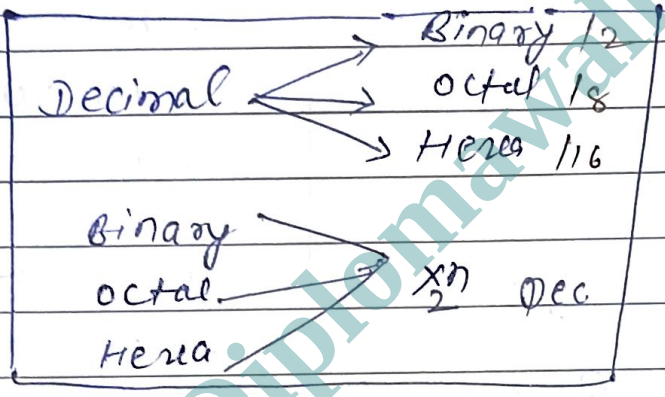
$(1010\ 1011.11001101)_2$ Ans

$(AB.CD)_{16} = (?)_8$

$A\ B\ .\ C\ D$

[01 0] [10 11] . [110 0 11] [01 0]

$= (253.632)_8$ Ans



Complement

1's complement

0 ⟶ 1

1 ⟶ 0

$(5)_{10} = (101)_2$

$(-5)_{10} = (010)_2$

2^5	2^4	2^3	2^2	2^1	2^0	
32	16	8	4	2	1	
			1	0	1	→ 5
			1	1	0	→ 6
	1	1	0	0	0	→ 24
		0	1	1	1	
					1	

2's complement:

Step-1 : Convert into 1's complement

Step-2 then add +1 a LSB.

Sum	Carry
0 + 0 = 0	0
0 + 1 = 1	0
1 + 0 = 1	0
1 + 1 = 2	1
1 + 1 + 1 = 1	1

Q $(5)_{10} = (101)_2$

step 1 $= (010)_2$

step 2 $=$ 010 + 1 = 011

$(-5)_{10} = (011)_2$

Q. $(11)_{10}$

step 1 – $(1011)_2$

$(-11)_{10} = (0100)_2$

step 2:- 0100 + 1 = 0101 ✓

0	+
1	–

$(5)_{10} = (101)_2 = (0101)_2$

1's complement

$(0101)_2$

1010

2 sign.

1's 2's

Q. $(6)_{10}$ = $(6)_{10} = (0110)_2$

$= 1001$

+1 → LSB

0000

+1

$(-6)_{10} = (1010)_2$ Ans.

$(7)_{10} = (0111)_2$

(1000)

1000

+1

1001

$(-7)_{10} = (1001)_2$ Ans.

Complement :-

The complement is used for representing the negative decimal number in binary form. Different types of complement are possible for the binary number, but 1's and 2's complements are mostly used for binary numbers. We can find the 1's complement of the binary number by simply inverting the given number. For example, 1's complement of binary number 1011001 is 0100110. We can find the 2's complement of the binary number by changing each bit (0 to 1 and 1 to 0) and adding to the least significant bit. For example, 2's complement of binary number 1011001 is (0100110) + 1 = 0100111.

Use :-

* The main use of complement is to represent a signed binary number.
* It is also used to perform various arithmetic operations such as addition and subtraction.

How to Find One's complement of a Number?

Step - 1 :- Convert the number of any number system to a binary number system.

i.e. If the number is in octal, decimal, hexadecimal, or any other number system; so convert it into a binary number system.

step 2 :- After having the number in binary form. Now, invert or exchange all the 0's to 1 and all the 1's to 0.

step 3 :- The resulting binary number is the 1's complement of the given number

⇒. Subtractions by 1's complement :-

step 1 :- Take 1's complement of the subtrahend

step 2 :- Add with minuend

step 3 :- If the result of above addition has carry bit 1, then add it to the least significant bit (LSB) of given result.

How to Find 2's complement of a number?

step 1 :- Convert the number of any number system to a binary number system. i.e if the number is in octal, decimal, hexadecimal or any other number system; so convert it into a binary

number system.

Step 2 :- After having the number in binary form. Now invert or exchange all the 0's to 1 and all the 1's to 0

Step 3 :- 1's complement of given number and add 1 to the least significant bit (LSB).

Step 4 :- → The resulting binary number is the 1's complement of the given number.

Subtractions by 2's complement :-

Step 1 :- Take 2's complement of the subtrahend

Step 2 :- Add with minuend.

Step 3 :- → If the result of the above addition has carry bit 1. then it is discarded.

8 4 2 1

BCD Code

The BCD stands for Binary coded Decimal Number. In BCD code, each digit of the decimal number is represented as its equivalent binary number.

Binary code A B C D	Decimal Number	BCD Code B_5 B_4 B_3 B_2 B_1
0 0 0 0	0	0 0 0 0 0
0 0 0 1	1	0 0 0 0 1
0 0 1 0	2	0 0 0 1 0
0 0 1 1	3	0 0 0 1 1
0 1 0 0	4	0 0 1 0 0
0 1 0 1	5	0 0 1 0 1
0 1 1 0	6	0 0 1 1 0
0 1 1 1	7	0 0 1 1 1
1 0 0 0	8	0 1 0 0 0
1 0 0 1	9	0 1 0 0 1
1 0 1 0	10	1 0 0 0 0
1 0 1 1	11	1 0 0 0 1
1 1 0 0	12	1 0 0 1 0
1 1 0 1	13	1 0 0 1 1
1 1 1 0	14	1 0 1 0 0
1 1 1 1	15	1 0 1 0 1

Step to convert the binary number to BCD:-

Step 1:- We will convert the binary number into decimal.

$(11101)_{BCD}$ to $(?)_2$

11101 double add

13 + 14 (27)

2 9 $(00101001)_{BCD}$

0010 1001

Step 2 :- We will convert the decimal number into BCD.

Steps to convert the BCD to Binary Conversion

Step 1 :- We will convert the BCD number into a decimal by making the four-bit groups and finding the equivalent decimal number for each group.

Step 2 :- We will convert a decimal number into Binary using the process of converting decimal to binary number.

Excess - 3 code

The Excess - 3 code can also be represented as the xs - 3 code. In excess - 3, each digit of the decimal number is represented by adding 3 in each decimal digit.

Decimal	BCD	Excess - 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

Steps to convert the binary number into Excess - 3 code :

Method 1

step1 :- Convert the binary number into decimal.

step2 :- Add 3 in each digit of the decimal no.

step 3 :- Find the binary code of each digit of the newly generated decimal number.

Method 2

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

steps to convert the Excess-3 code into binary number :-

step1 :- We will make the group of 4 bits and write the equivalent decimal number from the Excess - 3 table.

step 2 :- We find the binary number of the decimal number using a decimal to binary conversion.

Binary - coded decimal (BCD) is used in many applications, including:-

* Digital systems
BCD is used in calculators, cash registers and accounting systems where precise decimal arithmetic is required.

* Digital displays
BCD is used in digital displays where it can be difficult to display large numbers.

* Real - time clocks:
BCD is used in real-time clock circuits and measurement systems to keep track of wall - clock time.

* Database management systems
BCD is used in database management systems to offer a relatively easy way to get around size limitations on integer arithmetic.

Excess - 3 is used in many applications, including:-

* Arithmetic operations
Excess - 3 code is a non - weighted binary code that's important for arithmetic operations. It's used to overcome issues

that arise when adding two decimal digits that sum to more than 9 using the 8421 BCD code.

* Cash registers
Excess - 3 code was used in cash registers in the 1970s.

* Hand - held Calculators.
Excess - 3 code was used in hand - held portable electronic calculators in the 1970's

Gray code
The Gray code is a sequence of binary number systems which is also known as reflected binary code. In the typical sequence of binary numbers produced by the hardware that could provide an error or ambiguity during the change from one number to the next, gray codes are highly helpful.

Gray code applications
- Gray code can be used to simplify Boolean functions in karnaugh maps.
- Its main applications are in industry and robotics.
- In communication systems, it is used to detect unexpected changes in data.

Decimal	Binary code	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

Binary to Gray code conversion

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

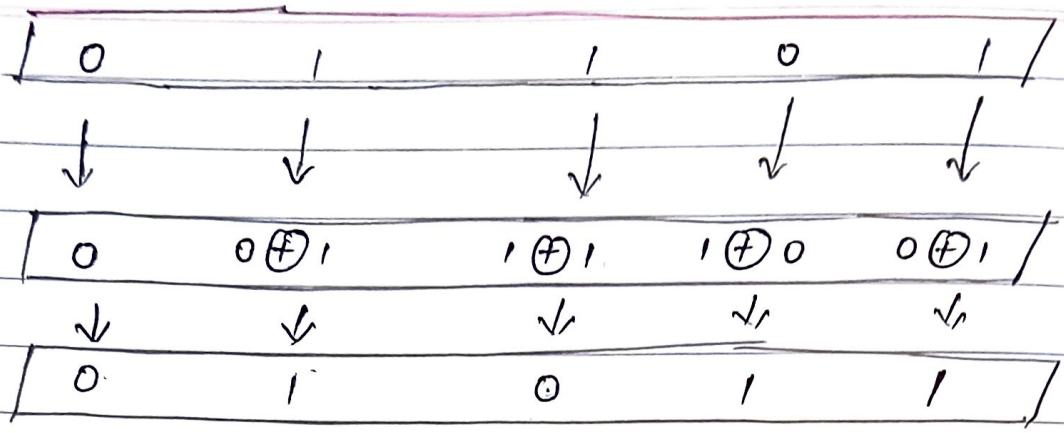
Logic circuit for Binary to Gray code converter

How to convert Binary to Gray code

Step 1 :- In the Gray code the MSB will always be the same as the 1'st bit of the given binary number.

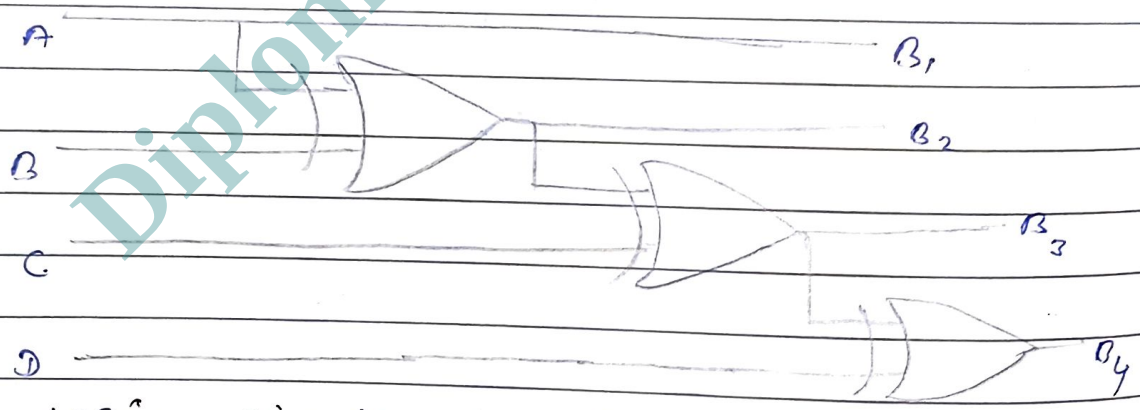
Step 2 :- In order to perform the 2nd bit of the gray code. We perform the exclusive or (XOR) of the 1'st and 2nd bit of the binary number. It means that if both the bits are different, the result will be one else the result will be 0.

Step 3 :- In order to get the 3rd bit of the gray code, We need to perform the exclusive-or (XOR) of the 2'nd and 3rd bit of the binary number. The process remains the same for the 4th bit of the Gray code.



Gray to Binary code conversion:

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



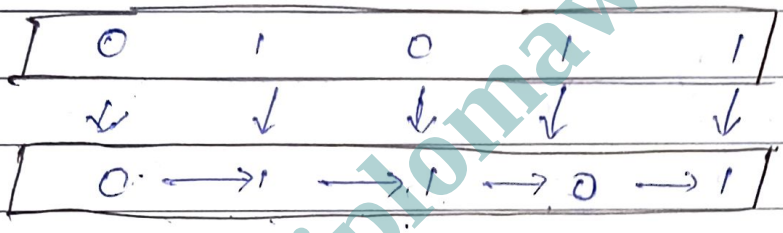
Logic circuit for Gray to Binary code converter

HOW to convert Gray code to Binary code

Step 1:- The 1st bit of the binary number is similar to MSB of the Gray code.

Step 2 :- The 2nd bit of the binary number is the same as the 1st bit of the binary number when the 2nd bit of the Gray code is 0; otherwise, the 2nd bit is an altered bit of the 1st bit of the binary number. It means if the 1st bit of the binary is 1, then the 2nd bit is 0, and if it is 0, then the 2nd bit is 1.

Step 3 :- The 2nd step continues for all the bits of the binary number.



What is a Logic Gate?

- A logic gate is an electronic circuit designed by using electronic components like diodes, transistors, resistors, and more. A logic gate is designed to perform logical operations in digital systems like computers, communication systems etc.
- It performs logical operation based on the inputs provided to it and produces a logical output that can be either "true" or "false".
- These gates can have one input or more than

one input, but most of the gates have two inputs.

Types of logic Gates

The logic gates can be classified into the following major types:-

1. Basic logic Gates.
   There are three basic logic Gates
   1. AND Gate
   2. OR Gate
   3. NOT Gate

2. Universal logic Gates
   There are two basic logic gates:
   1. NOR Gate
   2. NAND Gate

3. Derived logic Gates
   There are two basic logic gates:
   1. XOR Gate
   2. XNOR Gate.

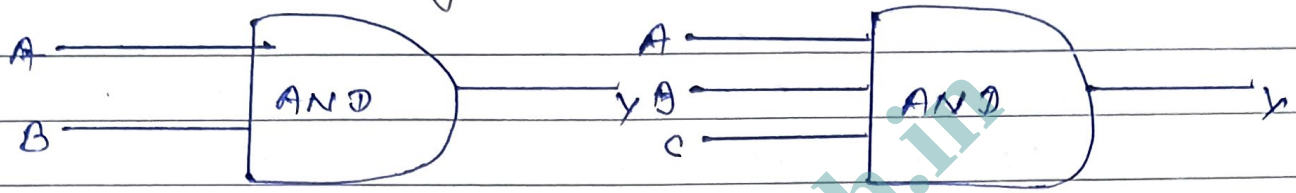
Basic logic Gates

1. AND Gate

An AND gate is a type of basic logic gate used in various digital circuits and systems. It produces a high

or logic 1 or True output, only when all its inputs are high or logic 1 or true. for all other combinations of inputs, it produces a low or logic 0 or false output.

The logic symbols for the two and three input AND gates:-



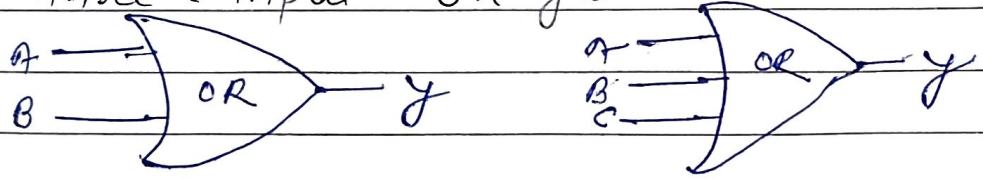
Boolean Expression of AND Gate

$Y = A \cdot B$

$Y = A \cdot B \cdot C$

2. OR Gate

An OR gate is a type of logic gate used to perform logical addition. It can have two or more inputs and one output. The output of the OR gate is low or logic 0 only when all its inputs are low or logic 0. For rest input combinations, the output of the OR gate is high or logic 1. The logic symbols for a two-input and a three-input OR gate.



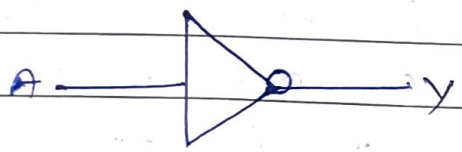
2. Boolean Expression of OR Gate

$$Y = A + B$$
$$Y = A + B + C$$

3. NOT Gate

The NOT gate is a type of basic logic gate used in digital electronics to implement the inversion function. Since, it performs the inversion operation, it is also known as inverter. It has only one input line and one output line. The output of the NOT gate is high or logic 1 when its input is low or logic 0. The output of the NOT gate is low or logic 0 when its input is high or logic 1.

The logic symbol of the NOT gate



Boolean Expression of NOT Gate:

$$Y = A'$$

universal logic Gates

Universal Logic Gates

1. **NOR Gate:-**

NOR Gate is a type of universal logic gate, because this logic gate can be used for implementation of any other types of logic gate.

NOR means "NOT + OR". That means the OR output is NOTed or inverted. Therefore, the NOR gate is a combination of OR gate and a NOT gate.

NOR = OR + NOT

A NOR gate is a type of logic gate whose output is HIGH (Logic 1), only when all its inputs are LOW (Logic 0), and it gives an output LOW (Logic 0), even if any of its inputs become HIGH (Logic 1).

The logic Symbol of a two input NOR gate.

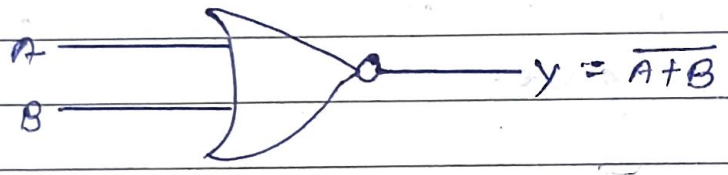


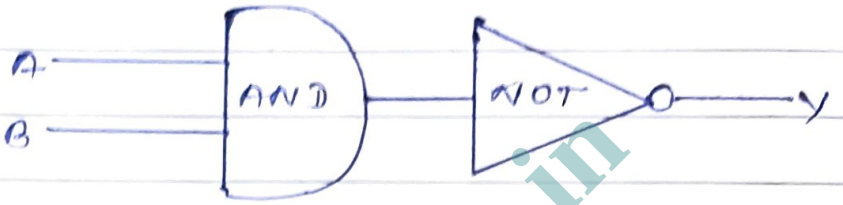
Figure 1 - NOR Gate.

Boolean Expression of NOR Gates.

$Y = (A+B)'$

2. NAND Gate

The NAND gate is a universal gate that basically a combination of two basic logic gates, namely, AND gate and NOT gate. It is designed by connecting a NOT gate to the output line of the AND gate.

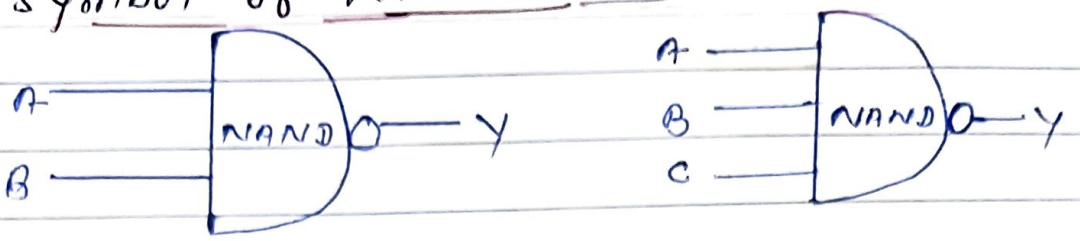


The NAND gate can have two or more input lines and output line. The output of the NAND gate is low or logic 0 only when all its inputs are high or logic 1. Otherwise, the output of the NAND gate is high or logic 1.

- The NAND gate is basically a logic gate that performs the inverse operation of an AND gate.

Being a universal gate, the NAND gate can implement any possible Boolean function or operation of any other type of logic gate.

Logic symbol of NAND Gate



Boolean Expression of NAND Gate

$Y = (AB)'$

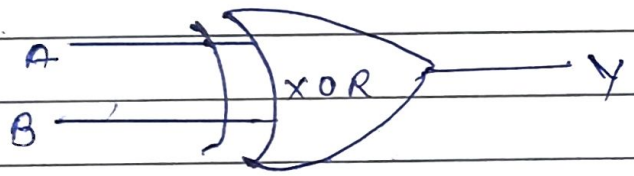
$Y = (ABC)'$

Derived Logic Gates

1. XOR Gate

The XOR gate is a type of logic gate in digital electronics which has two inputs and one output. The output of the XOR gates is high or logic 1, only when both the inputs are different. For the same inputs, the output of the XOR gate is low or logic 0. XOR gate is also called Exclusive OR gate or Ex-OR gate.

Logic symbol of XOR Gate



Boolean Expression of XOR Gate

$Y = A \oplus B$

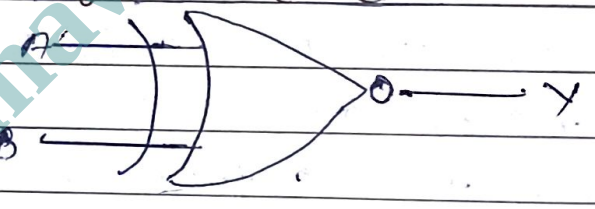
$Y = AB' + A'B$

2. XNOR Gate

The XNOR gate is a logic gate that has two inputs and one output. The output of the XNOR gate is high, only when both of its inputs same i.e, either both inputs are high or both inputs are low. If the inputs are dissimilar, i.e, one is high and the other low, the output is low or logic 0.

XNOR Gate = XOR Gate + NOT Gate.

Logic Symbol of XNOR Gate:



Boolean Expression of XNOR Gate

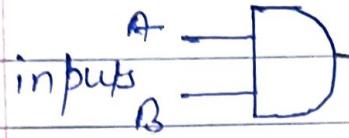
$Y = A \odot B$ or $Y = A \overline{\oplus} B$.

Basic Logic Gate.

(i) AND Gate.

0 - off - Low

1 - ON - High.



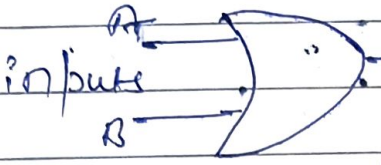
$O = A \cdot B$

Symbol

Input		Output
A	B	O
0	0	0
0	1	0
1	0	0
1	1	1

Truth table

(ii) OR Gate



Symbol

$O = A + B$

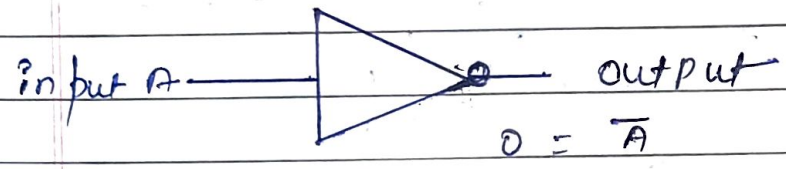
Inputs		output
A	B	O
0	0	0
0	1	1
1	0	1
1	1	1

Truth table

★ If any input is high (1) then output will be high.

★ If any input is low (0), then the output will be low (0).

(iii) NOT Gate (Inverter)



$O = \overline{A}$

Symbol

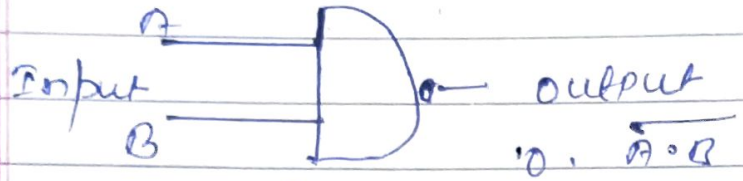
inputs	output
A	O
0	1
1	0

Truth table.

Same = 0
diff = 1

Universal logic Gate

NAND Gate (NOT + AND)



$O = \overline{A \cdot B}$

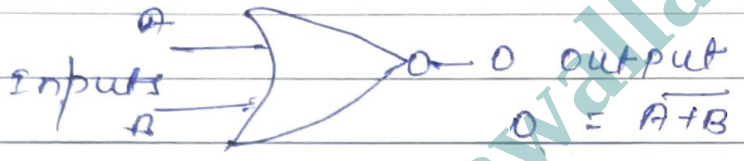
Symbol

Input		output
A	B	O
0	0	1
0	1	1
1	0	1
1	1	0

Truth table

NOR Gate (NOT + OR)

Sam = 1
diff 0



$O = \overline{A+B}$

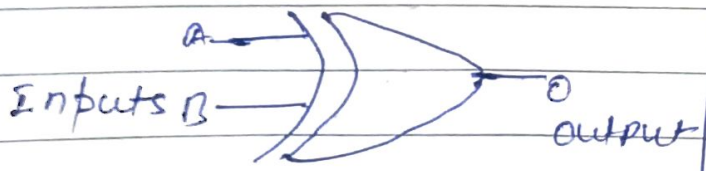
Symbol

Input		output
A	B	O
0	0	1
0	1	0
1	0	0
1	1	0

Truth table.

Derived logical Gates

Exclusive - OR Gate



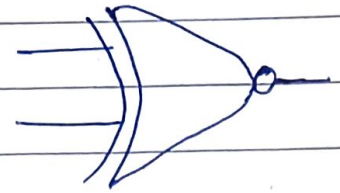
$O = A \oplus B$

Symbol

input		output
A	B	O
0	0	0
0	1	1
1	0	1
1	1	0

Truth table

Exclusive - NOR Gate



Symbol

Inputs		output
A	B	O
0	0	1
0	1	0
1	0	0
1	1	1

Boolean algebra

Boolean algebra is a branch of dealing with logical operations on variables. There can be only two possible values of variables in boolean algebra, i.e either 1 or 0. The Boolean algebra is mainly used for simplifying and analyzing the complex Boolean expression. It is also known as Binary algebra because we only we binary numbers in this. George Boole developed the binary algebra in 1854.

Rules in Boolean algebra :-

- only two values (1 for high and 0 for low) are possible for the variable used in Boolean algebra.

- The overbar (-) is used for representing the complement variable. So, the complement of variable C is represented as.
- The plus (+) operator is used to represent the ORing of the variables.
- The dot (·) operator is used to represent the ANDing of the variables.

Rules or Laws of Boolean Algebra

A set of rules or laws of Boolean Algebra expressions have been invented to help reduce the number of logic gates needed to perform a particular logic operation resulting in a list of functions or theorems known commonly as the laws of Boolean Algebra.

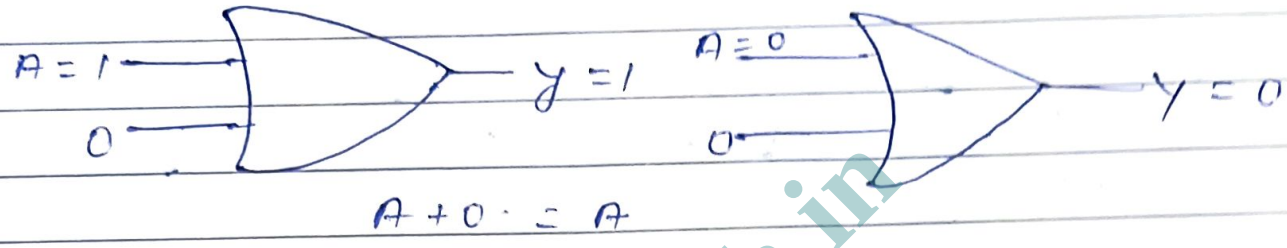
Rules of Boolean algebra.

There are the following rules of Boolean algebra, which are mostly used in manipulating and simplifying Boolean expressions.

1	$A + 0 = A$	7	$A \cdot A = A$
2	$A + 1 = 1$	8	$A \cdot A' = 0$
3	$A \cdot 0 = 0$	9	$A'' = A$
4	$A \cdot 1 = A$	10	$A + AB = A$
5	$A + A = A$	11	$A + A'B = A + B$
6	$A + A' = 1$	12	$(A+B)(A+C) = A + BC$

Rule 1:- $A + 0 = A$

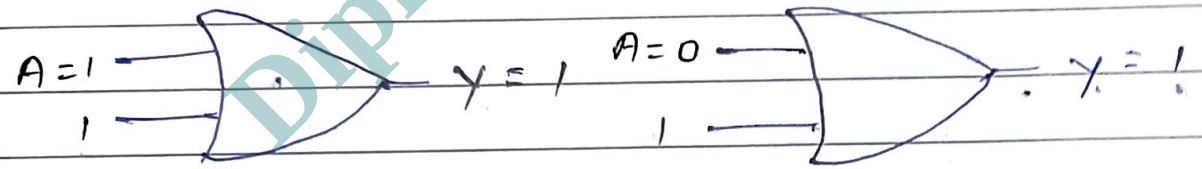
We have an input variable A whose value is either 0 or 1. When we perform OR operation with 0, the result will be the same as the input variable.



$A + 0 = A$

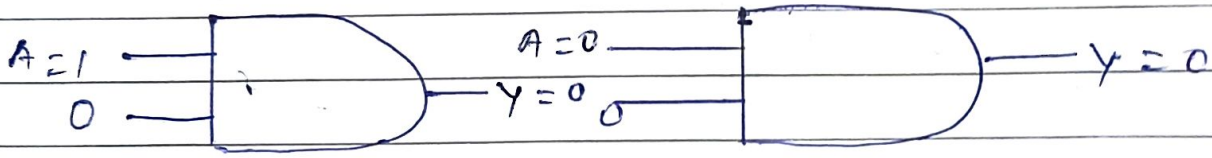
Rule 2:- $(A + 1) = 1$

We have an input variable A whose value is either 0 or 1. When we perform OR operation with 1, the result will always be '1'.



Rule 3:- $(A \cdot 0) = 0$

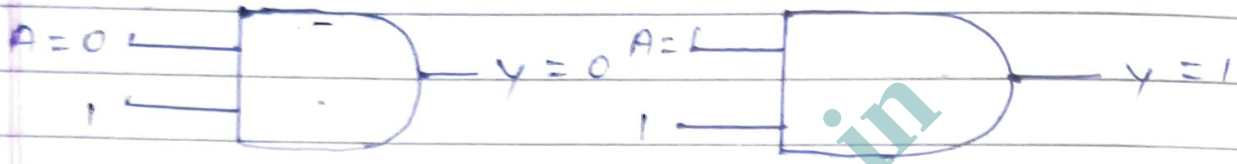
We have an input variable A whose value is either 0 or 1. When we perform the AND operation with 0, the result will always be 0.



Rule 4 :- $(A \cdot 1) = A$

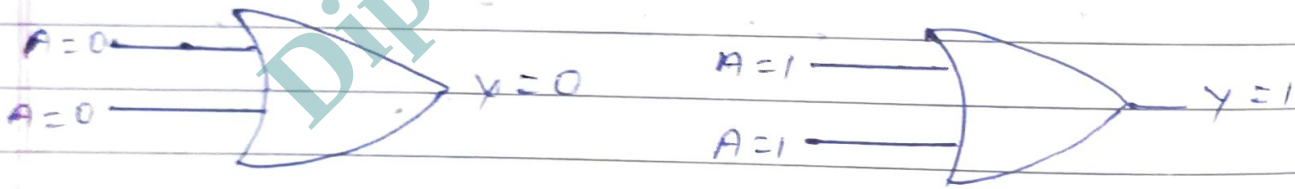
We have an input variable A whose value is either 0 or 1. When we perform the AND operation with 1, the result will always be equal to the input variable.

$\therefore (A \cdot 1) = A$



Rule 5 :- $(A + A) = A$

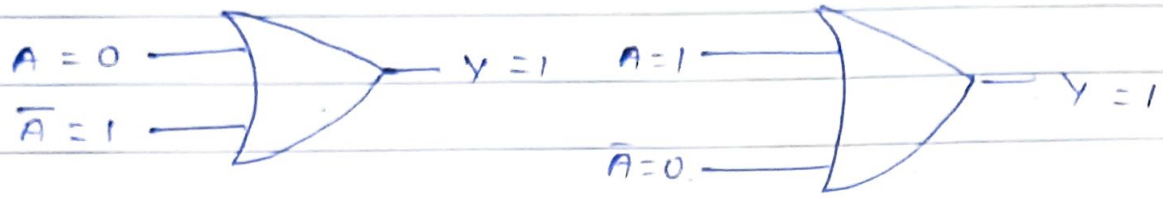
We have an input variable A whose value is either 0 or 1. When we perform the OR operation with the same variable, the result will always be equal to the input variable.



Rule 6 :- $(A + A') = 1$

We have an input variable A whose value is either 0 or 1. When we perform the OR operation with the complement of that variable, the result will always be equal to 1.

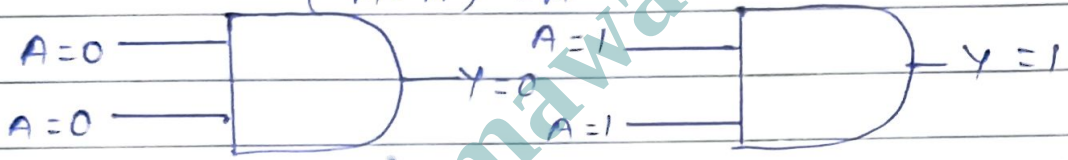
$(A + A') = 1$



Rule 7 :- $(A \cdot A) = A$

We have on input variable A whose value is either 0 or 1. When we perform the AND . operation with the same variable, the result will always be equal to that variable only.

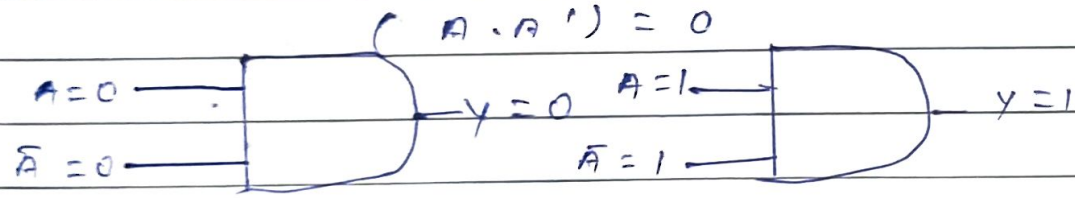
$(A \cdot A) = A$



Rule 8 :- $(A \cdot A') = 0$

We have on input variable A whose value is either 0 or 1. When we perform the AND operation with the complement of that variable, the result will always be equal to 0

$(A \cdot A') = 0$

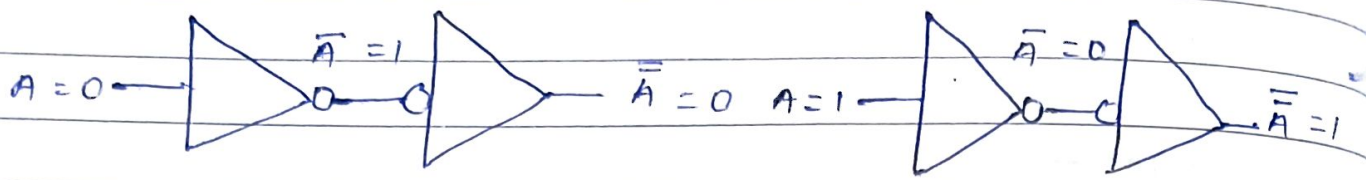


Rule 9 : $A = (A')'$

If we perform the double complement of

the variable, the result will be the same as the original variable.

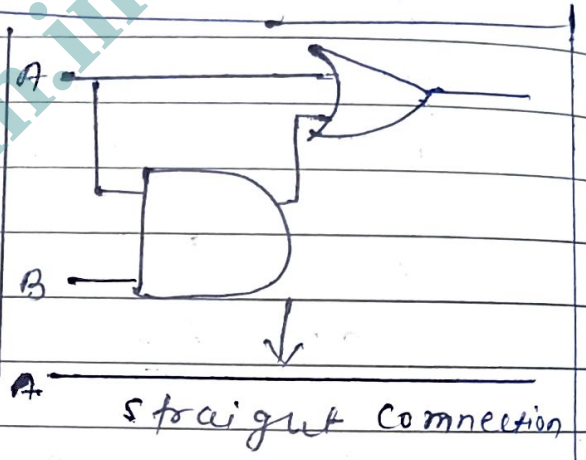
$$A = (A')'$$



Rule 10 :- $(A + AB) = A$

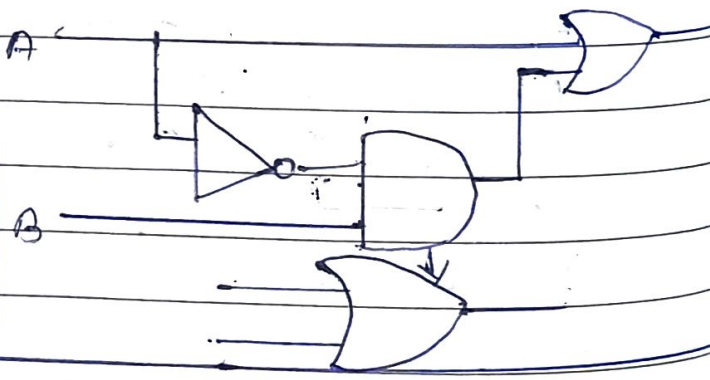
A	B	AB	A + AB
0	0	0	0
0	1	0	0
1	0	0	1
1	1	1	1

Equal



Rule 11 :- $A + \bar{A}B = A + B$

A	B	$\bar{A}B$	$A + \bar{A}B$	$A + B$



Rule 12 :- $(A+B)(A+C) = A+BC$

A	B	C	A+B	A+C	(A+B)(A+C)	BC	A+BC
0	0	0	0	0			
0	0	1	0	1			
0	1	0	1	0			
0	1	1	1	1			
1	0	0	1	1			
1	0	1	1	1			
1	1	0	1	1			
1	1	1	1	1			

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

$A \cdot A + AC + AB + BC$

$A + AC + AB + BC$

$A(1 + C + B) + BC$

$A \cdot 1 + BC$

$A + BC$

$A + AB = A$

$= A(1+B)$

$= A \cdot 1$

$= A$

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

$A + AB + \bar{A}B + 0$ [$\because A(1+A) = 1$]

$A(A+B) + \bar{A}B + A \cdot \bar{A}$

$A(A+B) + \bar{A}(B+A)$

$(A+B)(A+\bar{A})$

$(A+B) \cdot 1$

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

Laws of Boolean algebra

These are the following laws of Boolean algebra:-

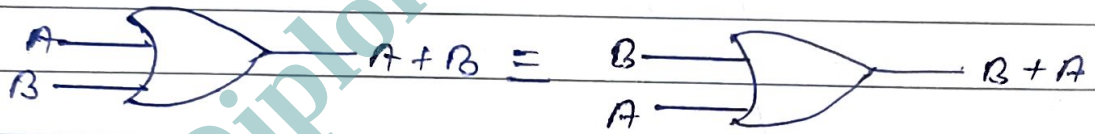
Commutative law:

This law states that no matter in which order we use the variable. It means that the order of variables doesn't matter. In Boolean algebra, the OR and the addition operations are similar.

For two variables, the commutative law of addition is written as:

$$A + B = B + A$$

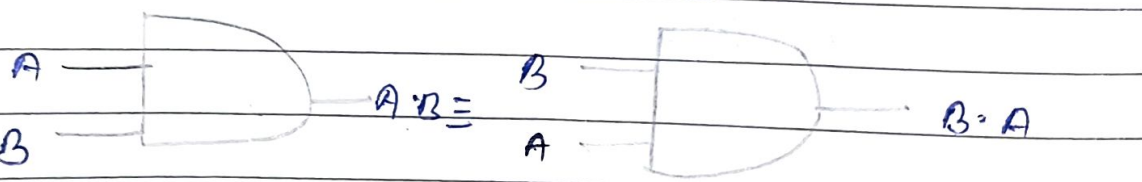
Commutative laws



For two variables, the commutative law of multiplication is written as:-

$$A \cdot B = B \cdot A$$

Commutative laws



Associative law

This law states that the operation can be performed in any order when the variable priority is same. In the below diagram,

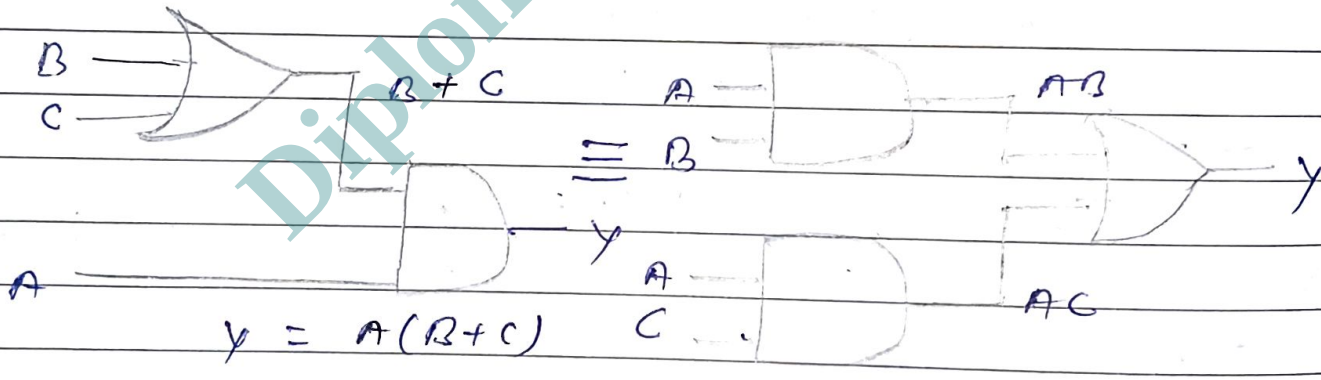
Distributive Law :-

According to this law, if we perform the OR operation of two or more variables and then perform the AND operation of the result with a single variables. then the result will be similar to performing the AND operation of that single variable with each two or more variable and then perform the OR operation of that product.

For three variables, the distributive law is written as :-

$$A(B+C) = AB + AC$$

Distributive law.



Demorgan's Theorem

Demorgan's Theorem is a powerful theorm in Booleon algebra which has a set of two rules or laws. These two laws were developed to show the relationship between two variable AND, OR and NOT operations.

(Dereloped by Augustus Demorgan)

Demorgan's first theorm (Law 1)

Demorgan's first law states that the complement of a sum (ORing) of variables is equal to the product (ANDing) of their individual complements. In other words, the complement of two or more ORed variables is equivalent to the AND of the complements of each of the individual variables, i.e—

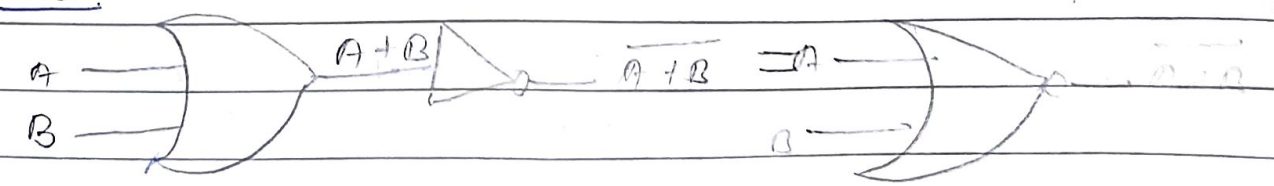
$$\overline{A+B} = \overline{A} \cdot \overline{B}$$

or, it may also be represented as,

$$(A+B)' = A' \cdot B'$$

The logic implementation of left side and right side of this law is.

LHS



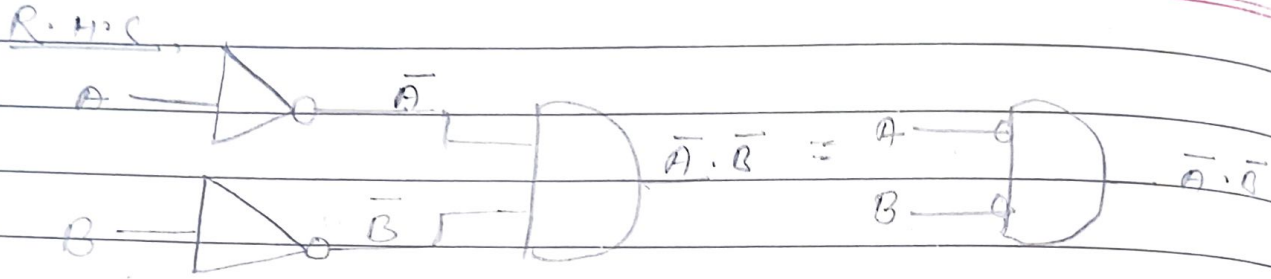


Figure 1 - DeMorgan's first Law

Thus, Demorgan's first law proves that the NOR gate is equivalent to a bubbled AND gate.

The following truth table shows the proof of this law

Left side			Right side		
Input		Output	Input		Output
A	B	$(A+B)'$	A'	B'	$A' \cdot B'$
0	0	1	1	1	1
0	1	0	1	0	0
1	0	0	0	1	0
1	1	0	0	0	0

The truth table proves that the Boolean expression on the left is equivalent to that on the right side of the expression of Demorgan's first law.

Demorgan's second Theorem (law 2)

Demorgan's second law states that the complement of the product (ANDing) of variables is equivalent to the sum (ORing) of their

individual complements.
In other words, the complement of two or more ANDed variables is equal to the sum of the complement of each of the individual variables, i.e -

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

It may also be represented as,

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

The logic implementation of left and right sides of this expression is

L.H.S

R.H.S

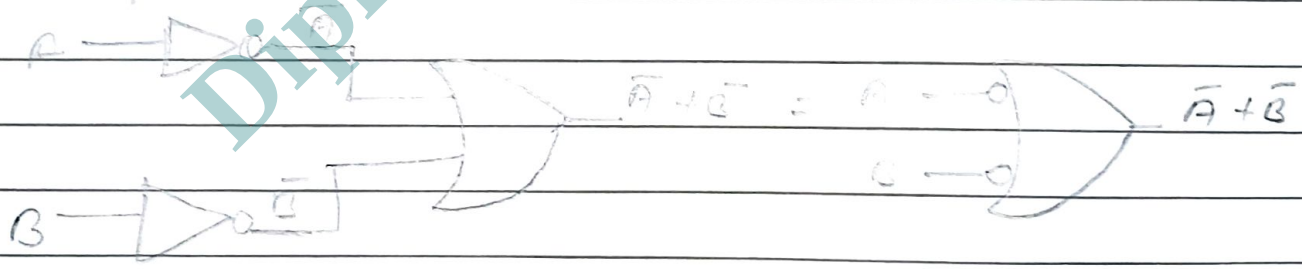


Figure 2:- Demorgan's second law

Hence, Demorgan's second law proves that the NAND gate is equivalent to a bubbled OR gate.

The following truth table shows ~~that~~ proof of this law:-

Left side			Right side		
Input		output	Input		output
A	B	$\overline{AB}$	A'	B'	A'+B'
0	0	1	1	1	1
0	1	1	1	0	1
1	0	1	0	1	1
1	1	0	0	0	0

The truth table proves that the Boolean expression on the left side is equivalent to that on the right side of the expression of Demorgan's second law.