Number System

1) Introduction
2) Conversion
3) Arithmetic Operations
4) Complement (1's 2's 9^{th} Complement means n^{th} & $(n-1)$ Complement).
5) Various Codes (Gray code, BCD code etc.)
6) Floting points

Base/Radix	No. systems	no. of allowed symbols
2	Binary no. system	(0-1) 0, 1
8	Octal	(0-7) 0, 1, 2, 3, 4, 5, 6, 7
10	Decimal	(0-9) 0, 1, 2, 3, 4, 5, 6, 7, 8, 9
16	Hexadecimal	(0-15) 0 1 2 3 4 5 6 7 8 9 A B C D E F

$(0, n-1)_n$

Conversion

1] A number From decimal to others
- Binary
- Octal
- Hexadecimal

Example:- $(19.35)_{10} \rightarrow (?)_2 \rightarrow (10011.01011)_2$

$\div n \leftarrow \div 2$ (Divisor, Dividend) ; $\times 2 \rightarrow \times n$

Divisor	Dividend	Remainder
2	19	1
2	9	1
2	4	0
2	2	0
	1	

Successive Division

Remainder → LSB Least Significant Bit

1 (MSB) ← read from MSB to LSB ← 0 0 1 1

$(10011)_2$

MSB most Significant Bit

0.35×2		LSB
0.70	0	
0.70×2	1	read from LSB to MSB
1.40		
0.40×2	0	
0.80		
0.80×2	1	MSB
1.60		
0.60×2	1	$.01011$
1.20		

Decimal to octal :-

$(19.35)_{10} = (?)_8$

÷8 ×8

8	19	Remainder
	2	3
	↓	↓
	MSB	LSB

$(23)_8$

0.35×8
2.80 → 2
0.80×8
6.40 → 6 ↓ (26)

$(23.26)_8$

Decimal to Hexadecimal :-

$(19.35)_{10} \rightarrow (?)_{16} \rightarrow (13.59)_{16}$

16	19	Remainder.
	1	3

$(13)_{16}$

0.35×16
5.60 → 5
60×16
9.60 → 9 ↓ (59)16

Conversion

From others to Decimal :-

- Binary
- Octal
- Hexadecimal

$(1101.01)_2 \longrightarrow (?)_{10}$

(1) Binary to decimal

$(1101.01)_2 \longrightarrow (?)_{10}$

$1\times2^3 + 1\times2^2 + 0\times2^1 + 1\times2^0 \cdot 0\times2^{-1} + 1\times2^{-2}$

$8 + 4 + 0 + 1 \cdot 0 + \frac{1}{4}$

$\frac{1}{4} \hookrightarrow 0.25$

$(13.25)_{10}$

(2) Octal to decimal

$(17.4)_8 \longrightarrow (?)_{10}$

$1\times8^1 + 7\times8^0 \cdot 4\times8^{-1}$

$8 + 7 \cdot 4\times\frac{1}{8}_2$

15.50

(3) Hexa to decimal :-

$(A7.C)_{16} \longrightarrow (?)_{10}$

$A\times16^1 + 7\times16^0 \cdot C\times16^{-1}$ ($C \to 12$)

$160 + 7 \cdot \frac{12}{16} = 3/4$

$\Rightarrow (167.75)_{10}$

Conversion b/w Base (Power of 2)

(Base will be in Power of 2 → like (× Base = Y Base power of 2)

like Binary Octal Hexa

Base → 2 (we can write it 2^1) 8 (2^3) 16 (2^4)

B → O → make pair of 3 → 

From this method we can ~~Calculate~~ Convert

B ↔ O

O ↔ H

B ↔ H

① Ex. $(110\,11\,.\,110)_2 \rightarrow (33.6)_8$ (add 0 to make pair of 3)

$(011\,011\,.\,110)_2$ ($_3$ $_3$ $_6$)

421 011 . 110

in Octal → we represent a no. in 3-bit.

O → H

② $(237.61)_8 \rightarrow (\quad)_{16}$

open → 010 011 111 . 110 001 100

(9 F C 4)$_{16}$

4 bit to represent

8421

0010

open in 4 bit & make pair of 3 digit to convert it (Hexa to octal) → remove zero for the

Any 0s on the extreme left of the integer part & extreme right of fractional part of the equivalent binary no. should be omitted. (zeros)

A 11
B 12
C 13
D 14
E 15
F 15

③ B → H

0110 1010 . 0101 0000

6 A 6 8

$(6A.68)_{16}$

for reverse process open it in 4 bit no.

Ex:- if base is of 32, 64 then we have to just open no. in 5 digit 6 digit

Some important ques:

(1) $(123)_5 = (x8)_y$ [x must be less than y, $x < y$]

Find possible solutions.

→ Firstly we should Convert no. into decimal.

$1 \times 5^2 + 2 \times 5^1 + 3 \times 5^0 = xy^1 + 8 \times y^0$

$25 + 10 + 3 = xy + 8$

$38 = xy + 8$

$(xy)_{10} = 30$ → now possible values of x & y

x	y
15	2 X
2	15 ✓
6	5 X
5	6 X
30	1 X
1	30 ✓
10	3 X
3	10 ✓

$8 < y$ → Possible value.

Ans :- 3 possible ans.

(2) $(12)_3 + (34)_5 = (ab)_7$

firstly convert in decimal

$1 \times 3^1 + 2 \times 3^0 + 3 \times 5^1 + 4 \times 5^0 = (ab)_7$

$3 + 2 + 15 + 4$

$(5)_{10} \quad (19)_{10}$

$(24)_{10} = (ab)_7$

Decimal to other → Successive devision.

$7 \mid 24$

$\quad 3$ → 3 LSB

MSB

$\Rightarrow (33)_7$

Arithmetic Operation

Arithmetic operations in a Computer are done using binary numbers and not decimal numbers & these take place in its arithmetic unit. The electronic circuit of a binary adder with suitable shift register can perform all arithmetic operations.

① Binary Addition :-

Two binary no. added in same way as two decimal no. are added.

Addition	Substration	Multiplication
(i) $0+0=0$	$0-0=0$	$0\times0=0$
(ii) $0+1=1$	$1-0=1$	$0\times1=0$
(iii) $1+0=1$	$1-1=0$	$1\times0=0$
(iv) $1+1=10$	$10-1=1$	$1\times1=1$

(iv) 2_{10}

eg:-

$$\begin{array}{r} 0\ 1\ 0\ 1 \\ +\ 1\ 0\ 0\ 0 \\ \hline 1\ 1\ 0\ 1 \end{array} \qquad \begin{array}{r} 5 \\ 8 \\ \hline 13 \end{array}$$

②

$$\begin{array}{r} {\scriptstyle 1\ \ 1} \\ 1\ 0\ 1\ 0 \\ +\ 0\ 0\ 1\ 1 \\ \hline 1\ 1\ 0\ 1 \end{array} \qquad \begin{array}{r} 10 \\ 3 \\ \hline 13 \end{array}$$

ii) Octal Addition (range 0-7)

Ex:

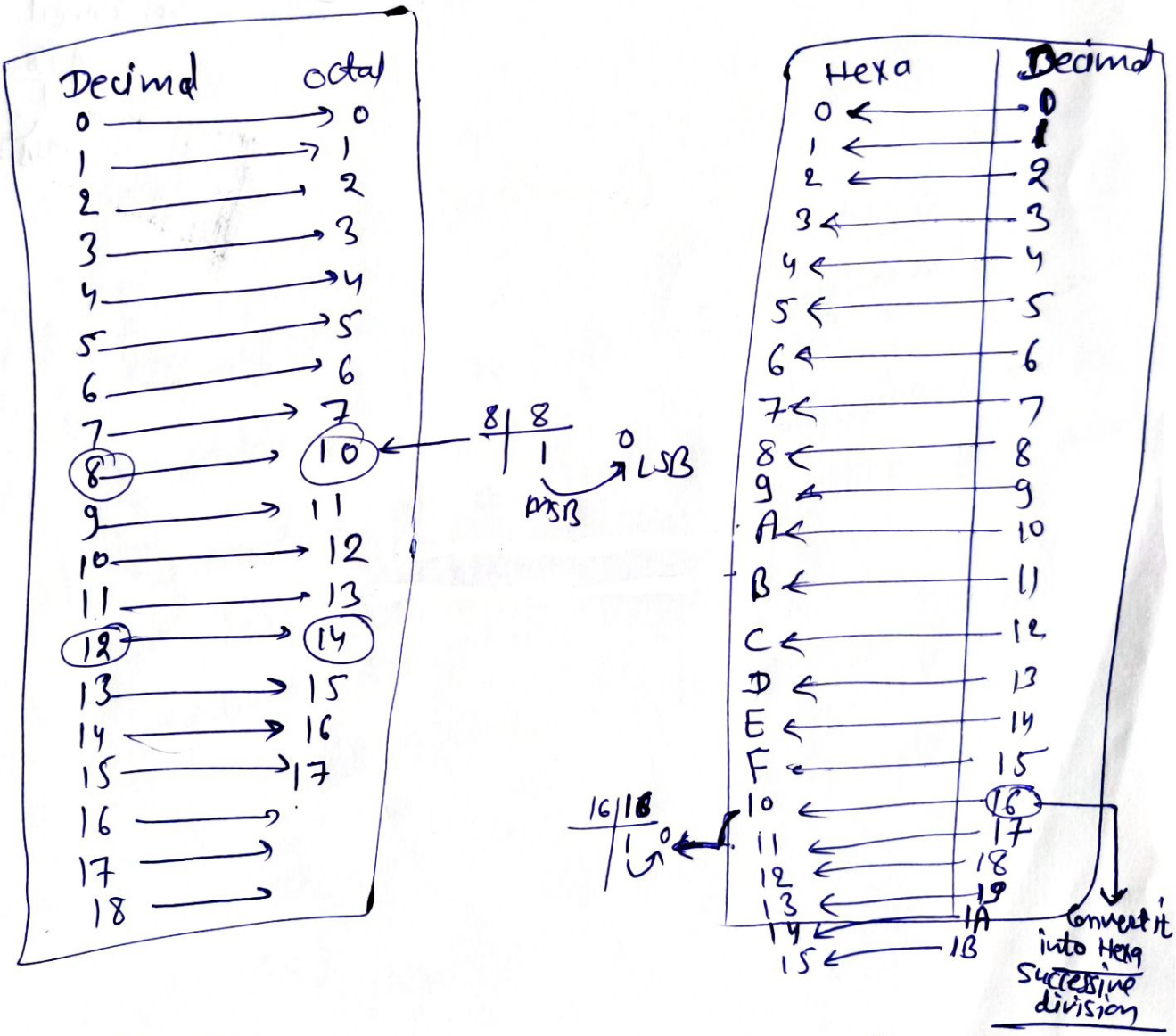
$$\begin{array}{r} 6\ 4\ 2 \\ +\ 3\ 2\ 1 \\ \hline 1\ 1\ 6\ 3 \end{array}$$

∴ decimal to octal

8 | 9
1 | 1 ↑ LSB
MSB

∴ 6
3
9 → It is not in octal range. So, we have to convert it in octal range.
(decimal)

#



Ex:

$$\begin{array}{r} \overset{1}{1}\ \overset{1}{4}\ 6 \\ +\ 1\ 3\ 6 \\ \hline 3\ 0\ 4 \end{array}$$

∵ 6+6 → 12, not in octal range
So convert this $(12)_{10}$ to $(\ )_8$ octal no.

8 | 12
1 | 4

Hexadecimal addition

Ex:

$$\begin{array}{cccc} & 8 & A & B \\ + & 9 & 3 & C \\ \hline 1 & 1 & E & 7 \end{array}$$

Ans

(1) → 10 → 11 + 12 = 23 (not in hexa range)

→ decimal no., for hexa decimal convert this no. in Hexa. by successive division of 23.

16 | 23	Remainder
1	7

msb ↗ LSB

→ 1 + A + 3 → 14 → it is in Hexa range.

Carry → E

→ → 8 + 9 → 17 → not in hexa range

So, convert it in hexa.

16 | 17	Remainder
1	1 LSB

msb ↗

② Binary Subtraction :-

Binary sub. is also carried out in same way as Decimal numbers are subtracted.

eg:

MSB			LSB	Decimal
1	1	0	1	13
− 1	0	0	1	− 9
0	1	0	0	4

0 − 0 → 0
1 − 0 → 1
Borrow 1 → 0
(1) 0 − 1 = 1

② 1 0 0 1 − 0 1 1 1 = 0 0 1 0 (Decimal 9 − 7 = 2)

③ 1 1 1 0 − 0 1 0 1 = 1 0 0 1 (Decimal 14 − 5 = 9)

④ 1 1 0 0 − 0 1 0 1 = 0 1 1 1 (Decimal 12 − 5 = 7)

⑤ 1 0 0 0 0 − 0 0 0 0 1 = 1 0 1 1 1 (Decimal 24 − 1 = 23)

⑥ 1 0 0 0 0 − 0 1 1 1 1 = 0 1 0 0 1

Ex:- Hexa decimal addition

	1	B	C
+	2	A	10
1	E	6	

Decimal 28 = Hexa 16 | 22 | 1 | 6

Binary Multiplication :- The procedure same as decimal multiplication.

①
```
  0110
x 0011
------
  0110
 0110x
00000xx
0000xxx
--------
00100010
```

②
```
  1.01
x 10.1
-------
  101
 000x
10 1xx
-------
11.001
-------
```

Binary Division :- ⓐ 11001 ÷ 101 (Dividend ÷ Divisor)

```
          01101   → Quotient
      ------------
110 ) 1001 11 00
      110 ↓
      ----
      0011 1
        110 ↓
        ----
        0011 0
          110
          ---
          000   Remainder.
```

```
          101
      ------------
101 ) 11001
      101 ↓↓
      ------
        010 1
          101
          ---
          0000
```

*** 1's and 2's Complement :-**

Ex :- 0 1 1 0 1 0

1's Complement (invert it) → 1 0 0 1 0 1

+ 1

2's Complement → 1 0 0 1 1 0

(add +1 in 1's Complement of this no.)

1's Complement Subtraction :-

Ex: Subtract $(1010)_2$ from $(1000)_2$

Direct Subtraction

```
  1000   minuend
- 1010   subtrahend
-------
- 0010   difference
```

1's Complement →

1's comp. method

```
  1000   (+)
+ 0101   add these
-------
  1101
```

No carry is obtained. The answer is the 1's complement of 1101 and is opposite to sign, i.e. −0010.

2's Complement Subtraction

1) 2's complement of a binary no. can be obtained by adding 1 to its 1's complement.

2) Subtraction of a smaller (subtrahend) no. from a large (Minuend) one by the 2's Complement method steps :-

   i) Determine 2's Complement of smaller (subtrahend) no.

   ii) Add this to large no. (minuend).

   iii) Drop out the carry (there is always a carry in this case.)

Ex:

Direct method

```
  1 1 1 1
- 1 0 1 0
---------
  0 1 0 1
```

Subtrahend → 2's Comp.

2's Complement method (+)

```
  1 1 1 1
+ 0 1 1 0
---------
1 0 1 0 1
```

Carry → 1 (dropout / discarded this carry)

→ ans

→ then no. is $(0101)_2$.

☆ : The 2's complement method for subtraction of a larger no. (subtrahend) from a smaller no. (minuend).

Steps :-

(i) Determine the 2's complement of the larger no. (subtrahend).

(ii) Add the 2's Complement to smaller no..

(iii) There is no carry. The result is in 2's Complement form and is negative.

(iv) To get answer in true form, take 2's Complement and change the sign

Ex:

```
  1 0 0 0
- 1 0 1 0
---------
- 0 0 1 0
```

2's Complement method

```
  1 0 0 0
+ 0 1 1 0
---------
1 1 1 0
```

2's Complement → No carry

→ No carry obtained. The difference is -ve. & true ans. is the 2's Complement of $(1110)_2 \rightarrow (0010)$.

Complement No. System

- (r-1)'s Complement (Diminished Radix/Base) → $(r^K-1)-x$
- r's Complement (Radix/base complement) → (r^K-x)

r → base/Radix
x = Given No.
K = Total digits in 'x'

* r's Complement is (r-1)'s Comp. +1

Base 2 → 1's Comp → 2's Comp

$\bar{x} = (2^K-1)-x$

$\bar{x} = (2^K-x)$

Base 8 → 7's Comp $\bar{x} = (8^K-1)-x$
Base 8 → 8's Comp $\bar{x} = (8^K-x)$

Base 10 → 9's Comp. $\bar{x} = (10^K-1)-x$
Base 10 → 10's Comp $\bar{x} = (10^K-x)$

Base 16 → 15's Comp. $\bar{x} = (16^K-1)-x$
Base 16 → 16's Comp $\bar{x} = (16^K-x)$

Q $(101)_2$ → In base 2 → highest no. is (1)
This is 3 digit no., so write 1 → 3 times
then minus given no.

```
  1 1 1
- 1 0 1
-------
  0 1 0   1's Complement
    + 1
-------
  0 1 1   2's Complete
```

7's & 8's Complement:-

Q2 $(123)_8$

in Base 8, max no. is 7
write it 3 time because given no. has 3 digit.
then minus the given no. in this no.

Ans (777)
− 123
654 ← 7's Complement
+1
655 ← 8's Complement

$(543)_8$

777
−543
234 → 7's
+1
235 → 8's

Q3: $(123)_{16}$

In 16 Base, highest no. is (15) F
then write it 3 times. because given no. is in 3-digits.
& then minus the given no. from this no.

15-3 → (12) → C

FFF
− 123
EDC → 15's Complem
+1
EDD

15-2, 15-3, 15-5 → (10) A

FFF
235
DCA → 15's Cm
+1
DCB

* Signed Binary No. Representation :-

→ Binary no. —represented by→ seperate sign bit along with magnitude.

eg:→ 8 bit binary no. → first (MSB) show sign bit
→ remaining 7 bit show magnitude.

Ex:

	Sign	magnitude
+13	0	000 1101
0	0	000 0000
-46	1	010 1110

Here, no. 0 is assigned with the sign bit '0'.

→ range of no. is -128 to +127. This assume that the decimal sum is within -128 to +127 range. Otherwise, we get an overflow.

* Addition in the 2's Complement System :-

Addition has four cases (i) when both no. are +ve
(ii) when augent is positve & addend is -ve no.
(iii) when augend is a -ve & addend is +ve no.
(iv) when both the no. are negative.

Case 1 :- Two positive no.

eg:-

+29	→	0	001	1101	(augend)
+19	→	0	001	0011	(addend
48		0	011	0000	(result = 48)
sum		↑ signbit			

- The sign bits of both augend & addend are zero & the sign bit of the sum is 0, indicating that when the sum is positive they have the same no. of bits.

Case 2 :- Positive augend no. & negative addend no.

+39	→	0	010	0111	augend
−22	→	1	110	1010	addend
+17		1	0	001	0001 (result = 17)
sum		↑ Carry	↑ signbit		

- The carry is discarded & hence the result is 0001 0001.
- Sign bit of addend is 1. Sign bit also participate in the process of addition.

Case 3 :- Positive addend no. & −ve augend no.

−47	→	1	101	0001	augend
+29	→	0	001	1101	addend
		1	110	1110	result [−18]
		↑ signbit			

It is 2's complement form. Last 7 bit represent (2's) complement.

→ for true value take 2's Comp of this. i.e → 10010 → (+18)

Case 4 Two Negative No.

$-32 \rightarrow 1\ 110\ 0000$ Augend
$-44 \rightarrow 1\ 101\ 0100$ addend

$1\ 0\ 011\ 0100$ (result = −76)

carry ↑ ↑ sign bit

- Carry is discarded. & result is (1011 0100).
- The true magnitude of the sum is the 2's Complement of 011 0100.

1] Subtraction in the 2's Complement System:-

Case 1:- Both no. are +ve

```
+28 ⟶ 0001 1100
+19 ⟶ 0001 0011
```

To subtract +19 from +28, Computer will send the +19 to a 2's Complement to circuit to produce.

```
-19 ⟶ 1110 1101
```

the system will then add +28 & -19

```
+28 ⟶ 0001 1100
-19 ⟶ 1110 1101
Sum(9) ⟶ 1 0000 1001
```

2] positive no. & smaller -ve no.

```
+39 minuend    ⟶ 0010 0111
-21 Subtrahend ⟶ 1110 1011
```

The Computer sends -21 to a 2's Complement circuit to produce.

```
+21 ⟶ 0001 0101
```

It then adds +39 & +21

```
+39 ⟶ 0010 0111
+21 ⟶ 0001 0101
Sum(60) ⟶ 0011 1100
```

Case-3 :- positive small & -ve large no.

minuend → +19 ⟶ 0001 0011
Subtrahend → -43 ⟶ 1101 0101

The computer sends the 2's Complement of (-43)

+43 ⟶ 0010 1011

It then adds +19 & +43

+19 ⟶ 0001 0011
+43 ⟶ 0010 1011
Sum(62) 0011 1110

Case 4 :- Both are -ve no. → 2's Complement representation

-57 ⟶ 1100 0111
-33 ⟶ 1101 1111

Taking 2's Complemat of (-33).

+33 ⟶ 0010 0001

Then add +33 to -57,

-57 ⟶ 1100 0111 ✓
+33 ⟶ 0010 0001 ✓
(-24) 1110 1000

Arithmetic Overflow :-

- when the no. of bits in the sum exceeds the no. of bits in each of the numbers added, overflow results.
- This appears in the 9th significant place, is also called the excess-one. Overflow causes a sign change.
- Assume that both the i/p no. are in the range of −128 to +127.

Case 1 :- Problem arise when adds 2 positive no. & 2 negative no. In such a case, it is possible for the sum to be outside the range of −128 to +127.

① Two +ve No. :-

+ve no. + +ve no. = −ve no. overflow does exist.

+120	⟶	0111 1000
+ 65	⟶	+ 0100 0001
(185)		(1)0111 1001

Answer (185) is +ve but result showing −ve no., Answer is incorrect.

Case-2 :- Two −ve no. :-

−77	⟶	1011 0011 ⟶ 2's Complement
+(−122)	⟶	1000 0110 ⟶ 2's Complement
−199		(1)(0)011 1001

(1) Carry, (0) sign bit showing +ve no. but addition is (−199), i.e incorrect.

- ▸ overflow is a software problem not a hardware problem.
- ▸ In digital computers: an overflow occurs when a operation results in a quantity beyond the capacity of the storage register.
- ▸ Programmer must check overflow by using logic circuitry

* **9's Complement :-** The 9's complement of a decimal no. can be found by subtracting each digit in the number from 9.

Eg:- 19 → find 9's Complement of decimal no.

① 99 − 19 = 80 → 9's Comp. of 19

② 999 − 146 = 853 → 9's Complement of 146

③ 9999 − 4397 = 5602 → 9's complement of 4397

* **9's Complement Subtraction :-**

Subtraction of a smaller decimal no. from a larger one.

Steps :-
- take 9's complement of subtrahend & add to the minuend no.
- If carry produce, carry add to the result.
- Subtraction of larger no. from smaller no. does not produce a carry. and
- The result is a −ve in 9's Complement form.

Eg:- ① 18 − 06 by using 9's Complement.

18 − 06 = 12 (direct method)

99 − 06 = 93 (1's comp.)

18 + 93 → 9's Comp. of 06

Carry → 1 11 → +1 → 12

Add Carry to result

9's Complement System

(b) 39 − 23 → direct → 16

39 + 76 → 9's Compl of 23

1 15

+1 → add carry to result.

16

(3) direct method: 34 − 49 → −15

9's Comp system: 34 + 50 → 9's Complement 49

84 → There is no carry, so again 9's complement of output like (84)

99 − 84 → 15 → −15

(4) 49 − 84 → −35

9's Comp. method: 49 + 15 → 9's Comp. of 84

64

again 9's Complement of 64

99 − 64 → −35

10's Complement:- 10's complement of decimal no. is equal to its 9's Complement + 1.

Ex: (a) 9 find 10's complement

first we will take 9's complement of 9.

9 − 9 → 0 → 9's Complement of 9

+1 → 1 → 10's Complement of 9

(2) 739 ⟶ 9's Complement of 739

```
  999
- 739
-----
  260   → 9's Com. of 739
+  1
-----
  261   → 10's Comp. of 739
```

10's Complement Subtraction :-

- minuend is added to the 10's complement of the subtrahend & Carry is dropped.

Ex: (1) 9 - 4

∴ Regular Subtraction

```
  9
- 4
---
  5
```

10's Complement Sub.

```
  9
+ 6   → 10's Comp. of 4
---     Drop Carry
 15
(1)
```

Drop ← (1) → Answer

(2) 347 - 265

Direct

```
 347
-265
----
  82
```

9's

```
999
265
---
734   → 10's
 +1
---
735
```

10's Comp. Sub.

```
 347
 735
+
-----
1082
```

(1) → Drop Carry

Binary Coded Decimal (BCD) → Radix 10 → (0-9)

→ means every decimal is coded in binary.

- BCD is a combination of 4-binary digits that represent decimal no. (0 to 9).
- for ex: 8421 Code is a type of BCD., also indicate binary weights of the four bits.

To express BCD codes for decimal no. 10 to 15, each digit should be replaced by appropriate four bit code.

Ex: 0 to 9 ⟶ 0001 to 1001
10 to 15 ⟶ define each digit individually.

like:-
10 → 0001 0000
15 → 0001 0101

BCD addition:- it is numeric code.

steps / Rules for addition:-

(i) Add the two no. using the rules for binary addition.

(ii) If a 4-bit sum is equal to or less than 9, it is a valid BCD no.

(iii) If a 4-bit sum is greater than 9, or if a carry out of the group generated, it is invalid BCD result. Add 6(0110)$_2$ to the four bit sum in order to skip the six invalid states & return the code to BCD.

(iv) if carry occur when 6 is added, add the carry to the next four-bit group.

Ex: (a) Add the BCD no.-

$$\begin{array}{r} 1001 \\ +\,0100 \\ \hline 1101 \end{array} \longrightarrow \text{invalid BCD no.}$$

$$\begin{array}{r} +\,0110 \\ \hline 0001\ 0011 \end{array} \longrightarrow \text{Add 6}$$

1 3 ⟶ valid BCD no.

$$\begin{array}{r} 9 \\ +\,4 \\ \hline (13)_{10} \end{array}$$

(b)

```
  0001   1001
+ 0001   0100
-----------------
  0010   1101   → Right group invalid.
+        0110   → Add 6
-----------------
  0011   0011   → valid BCD no.
    3      3
```

```
  19
+ 14
----
 (33)10
```

Binary Codes

- Alphanumeric codes
  - ASCII (7 Bit)
    - 26 lower Charac english
    - 26 Capital Charac english
    - 25 special Charac.
    - 128 value encoded.
  - EBCDIC (8 Bit)
    - (256 value encoded)
  - UNICODE
    - Now a day extensively use
      - UTF
        - 8 bit.
        - 16 bit
        - 32 bit.
- Numeric codes
  - Binary Coded Decimal (Decimal no encoded in binary)
    - Weighted Code
      - 8 4 -2 -1
      - 6 3 1 -1
      - 2 4 2 1
      - 3 3 2 1
      - 8 4 2 1
    - Non-weighted
      - Gray Code (Unit distance code)
      - Excess-3 Code

ASCII → American standard code for information interchange.

EBCDIC → Extended Binary Coded decimal interchange code.

Non-weighted are codes that are not positionally weighted. This means that each position within binary no. is not assigned a fixed value.

Codes :

1] Excess-3 code

Ex: $[643]_{10}$ Convert into its excess-3 code.

Decimal no.	6	4	3
Add 3 to each bit	+3	+3	+3
Sum →	9	7	6
BCD →	1001	0111	0110

Convert it into BCD

Gray code :- This is called minimum-Change code (Min-distance code).
(non weighted code)

- In which only one bit in the code group changes when moving from one step to the next.

Decimal no.	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

Binary to Gray

Ex:1 Convert $[0110]_2$ to Gray code.

Ex-OR

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

1 ⊕ 0 ⊕ 1 ⊕ 1 ⊕ 0

1 1 1 0 1

(MSB same)

Binary → Gray

Ex.2 $[1\oplus0\oplus1\oplus0\oplus1\oplus1\oplus0\oplus1]_2$ → Binary

1 1 1 1 1 0 1 1 → Gray code

Gray to Binary :-

Ex: 1 1 0 1 0 1 → Gray

1 0 0 1 1 0

(MSB same)

Ex:: 1 0 1 1

1 1 0 1

Boolean Algebra :-

B.A. can be used to simplify the design of logic circuits.

Boolean Laws

(1) $A+0=A$	(5) $A\cdot 1=A$
(2) $A+1=1$, $1+\bar{A}=1$	(6) $A\cdot 0=0$
(3) $A+A=A$	(7) $A\cdot A=A$
(4) $A+\bar{A}=1$	(8) $A\cdot\bar{A}=0$
	(9) $\bar{\bar{A}}=A$

[1] Basic Laws of Boolean Algebra :-

Logical operations can be expressed & minimized mathematically using the rules, laws, & theorems of Boolean algebra.

(i) **Boolean Addition** :- Same as logical OR operation.

$0+0=0$
$0+1=1$
$1+0=1$
$1+1=1$

(ii) **Boolean Multiplication** :- Same as logical AND operation.

$0\cdot 0=0$
$0\cdot 1=0$
$1\cdot 0=0$
$1\cdot 1=1$

* Properties of Boolean Algebra :-

Boolean Algebra is a mathematical system that consist two binary operators denoted by symbols (+) & (·). & one unary operator denoted by symbol either (−) or prime(').

(i) Commutative Property :-

$Y = A + B = B + A$

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

The order of variables which are operated by the binary operators does not effect the result, when variables A & B are replaced by each other, the o/p (Y) will not change.

(ii) Associative Property :-

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

$A \cdot (B \cdot C) = (A \cdot B) \cdot C$

When binary operators (+) & (·) operates on more than two variables, result does not depend on grouping of the variables.

(iii) Distributive Property :- (a) Boolean Addition is distributive over Boolean multiplication.

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

Proof :- $A + BC$

$= A \cdot 1 + BC$ $\quad \because (A \cdot 1 = A)$

$= A(1 + B) + BC$ $\quad \because (1 + B = 1)$

$= A \cdot 1 + AB + BC$ $\quad \because [A(B + C) = AB + BC]$

$= A \cdot (1 + C) + AB + BC$ $\quad \because (1 + C = 1)$

$= A \cdot A + AC + AB + BC$ $\longrightarrow (\because A \cdot A = A)$

$= A(A + C) + B(A + C) \Rightarrow \underline{(A + C)(B + A)}$ H.P.

(6) Boolean multiplication is also distributive over boolean addition. This law states that (·) operator is b/w one variable & two variables which are operated by (+) operator will have same result if one variable is operated by (·) operator individually with another variables & then operated by (+) operator.

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

Absorption Laws:-

(i) $A + AB = A$

Proof:- take L.H.S.

$A + AB \Rightarrow A \cdot 1 + AB$

$\Rightarrow A(1+B)$

$\Rightarrow A \cdot 1 \Rightarrow A$

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

Take L.H.S

$A \cdot (A+B) \Rightarrow A \cdot A + A \cdot B$

$\Rightarrow A + AB$

$\Rightarrow A(1+B) \Rightarrow A \cdot 1 \Rightarrow A$

(iii) $A + \bar{A}B = A + B$ $\quad \because A + BC = (A+B)(A+C)$

Take L.H.S

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

$\Rightarrow 1 \cdot (A+B)$ $\quad (\because A + \bar{A} = 1)$

$\Rightarrow A + B$

(iv) $A \cdot (\bar{A}+B) = AB$

take L.H.S

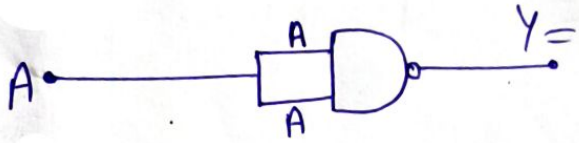
$A \cdot (\bar{A}+B) \Rightarrow A \cdot \bar{A} + AB$ $\quad (\because A\bar{A} = 0)$

$\Rightarrow 0 + AB \Rightarrow AB$

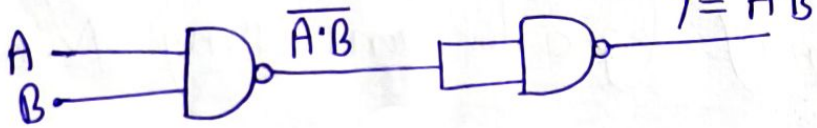
Realisation of logic function using NAND Gates :-

- NAND & NOR gates are called universal gates or universal building blocks because both can be used to implement any gate like AND, OR, NOT or any combination of these basic gates.

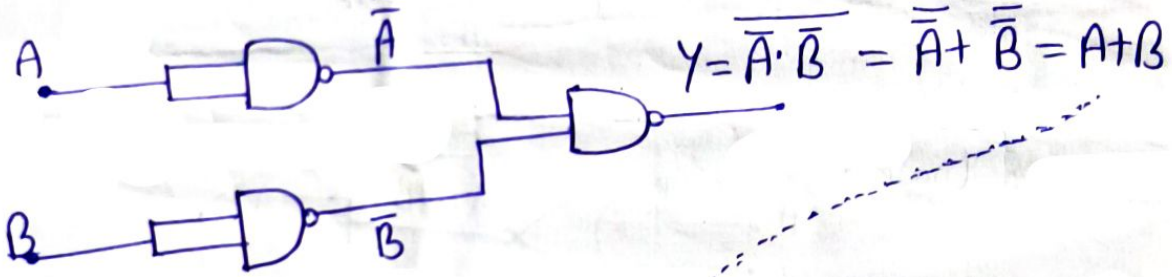
1] Not gate (inverter) :-



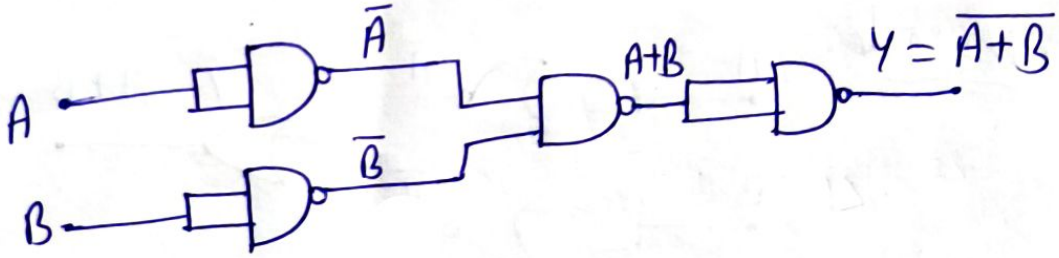
2] AND gate :-



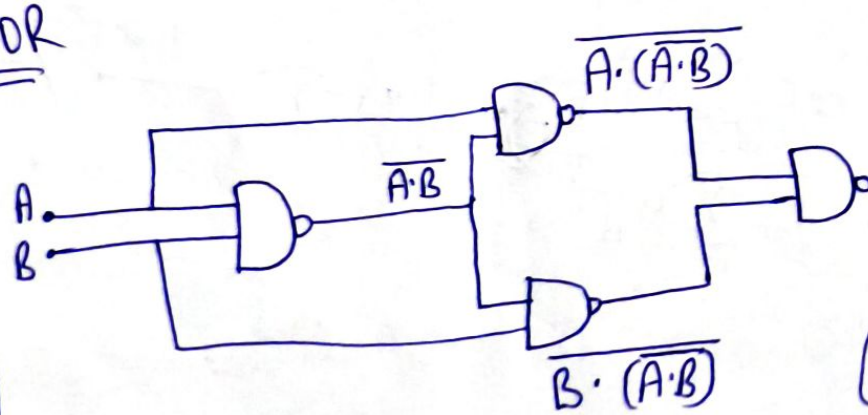
3] OR Gate :-



4) NOR Gate :-



5] Ex-OR



$$Y = \overline{\overline{A \cdot (A \cdot B)} \cdot \overline{B \cdot (A \cdot B)}}$$

$$Y = \overline{A \cdot (A \cdot B)} + \overline{B \cdot (A \cdot B)}$$

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

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

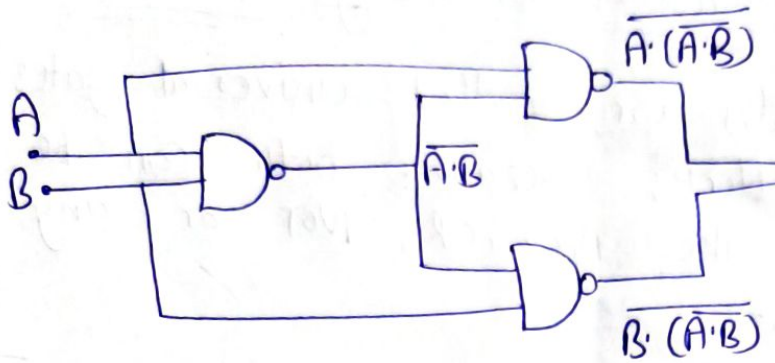
$$Y = A\bar{A} + A\bar{B} + B\bar{A} + B\bar{B}$$

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

$\because A \cdot \bar{A} = 0$

$B \cdot \bar{B} = 0$

6] Ex-NOR :-



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

$Y = (\overline{A\bar{B}})\cdot(\overline{B\bar{A}})$

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

$Y = A\cdot\bar{A} + \bar{A}\bar{B} + AB + B\bar{B}$

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

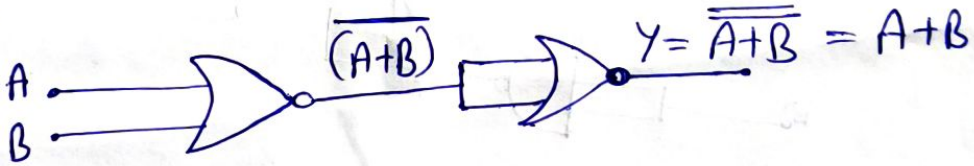
$Y = A \odot B$

Realisation of logic functions using NOR Gates :-

1) NOT gate

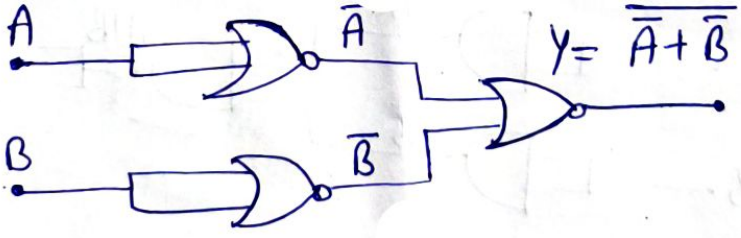
$Y = \overline{A+A} = \bar{A}$

2) OR Gate :-



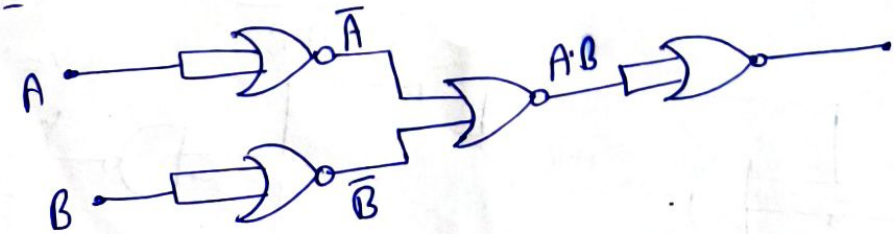
$Y = \overline{\overline{A+B}} = A+B$

3) AND gate :-



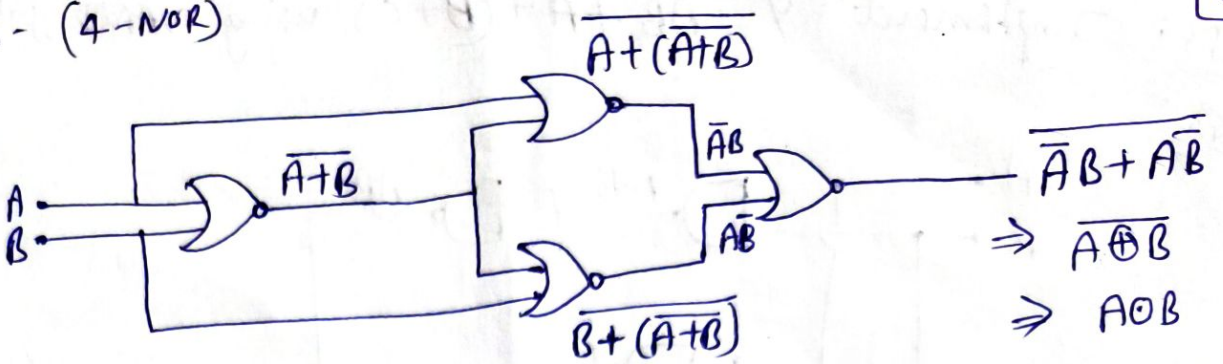
$Y = \overline{\bar{A}+\bar{B}} \Rightarrow \bar{\bar{A}}\cdot\bar{\bar{B}} \Rightarrow AB$

4) NAND Gate :-

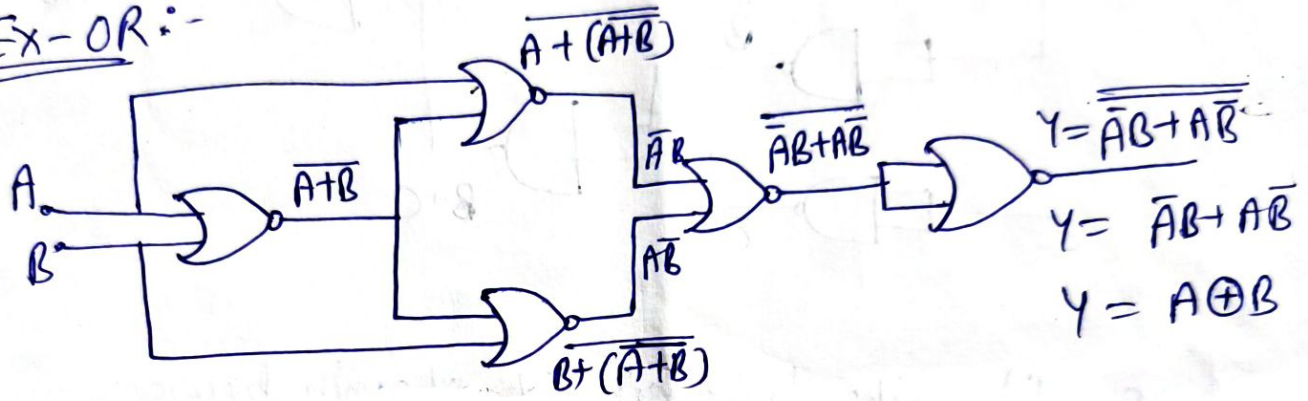


$Y = \overline{A\cdot B}$

5) Ex-NOR :- (4-NOR)

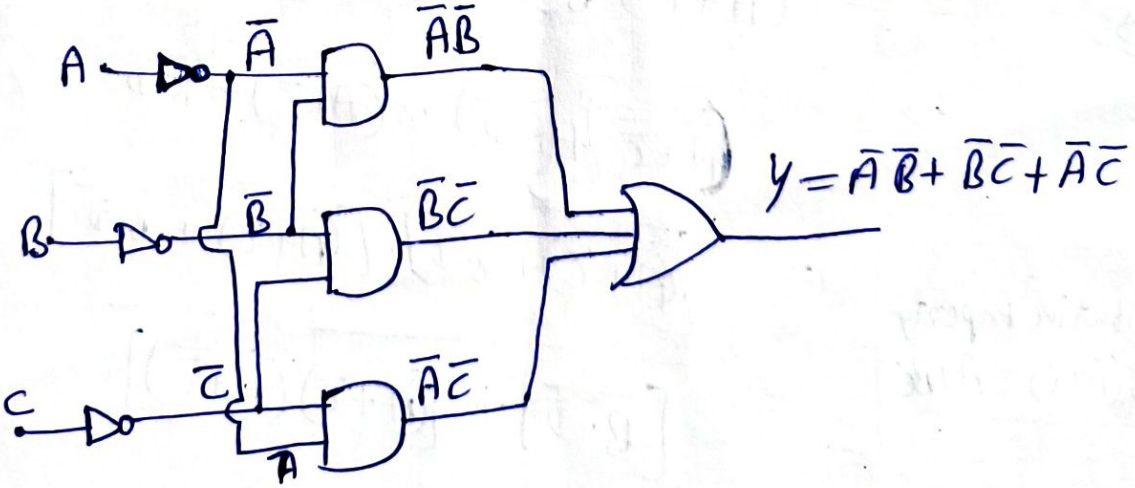


6) Ex-OR :-



Q1. Realise the logic expression $Y=\bar{B}\bar{C}+\bar{A}\bar{C}+\bar{A}\bar{B}$ using basic logic gates.

Ans
- 3 product terms obtained by 3-AND gate.
- Complemented form obtained by 3 Not gates.
- one OR gate.

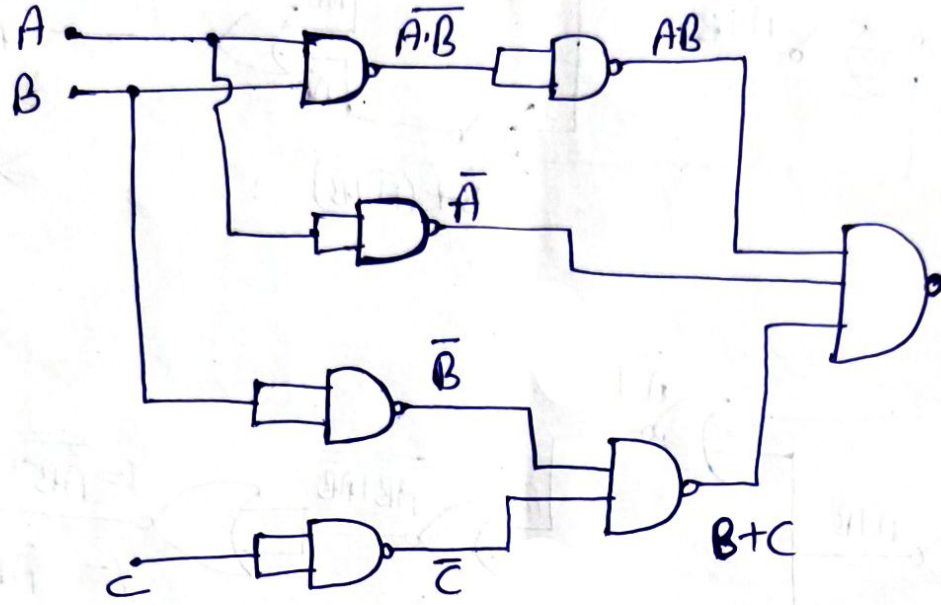


Q2 :- $Y=(A+B)(\bar{A}+C)(B+D)$ using basic gates.

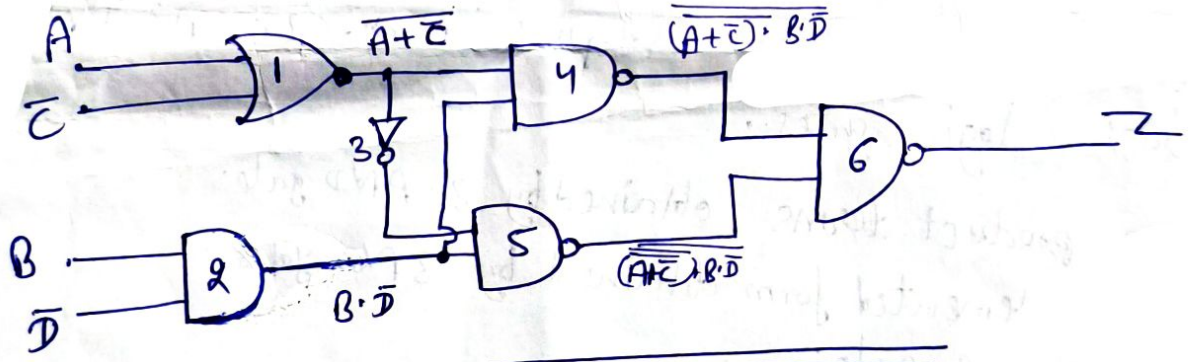
Ans 1 Not gate, 3 OR gate, 1 AND gate.

Q3: Implement $Y = \overline{A \cdot B} + A + \overline{(B+C)}$ using NAND gates only.

Ans:-



Q4:- Simplify the logic circuit shown in figure.



Ans

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

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

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

Distributive Property
$\therefore (A+B)(A+C) = A+BC$

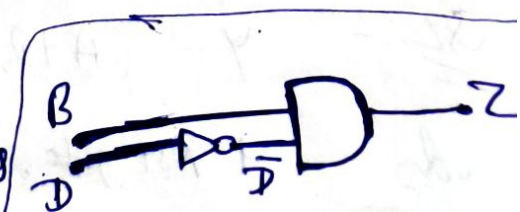
$$= \overline{[B \cdot \bar{D}] + [(A+\bar{C})(A+\bar{C})]}$$

$A \cdot \bar{A} = 0$

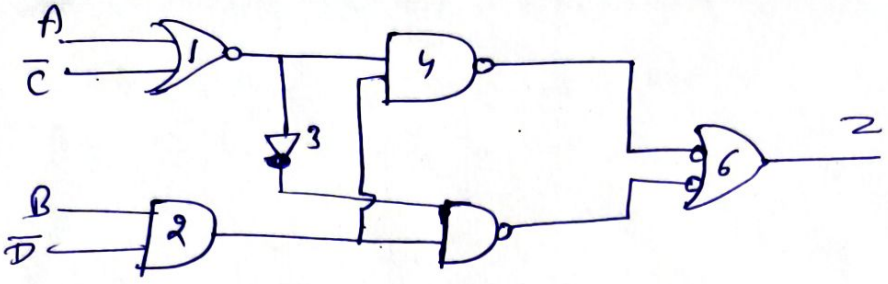
$$= \overline{B \cdot \bar{D}}$$

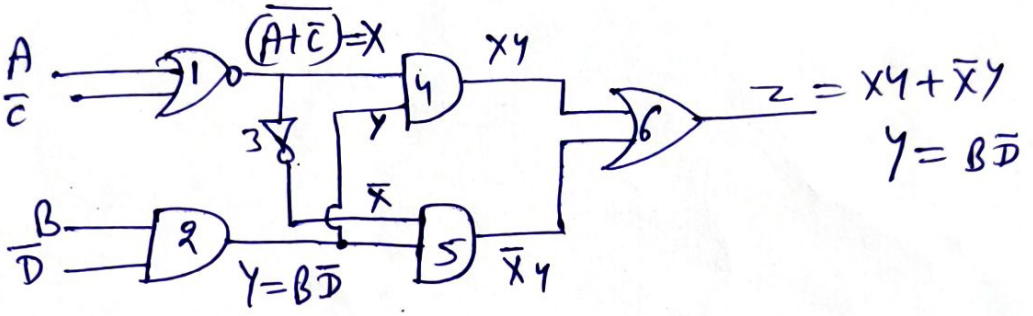
$$\Rightarrow B \cdot \bar{D}$$

Now → Simplified logic ckt is



or we can replace 6 No. NAND gate by bubble OR gate





#

4
4

Logic Gates	No. of NAND	No. of NOR
NOT	1	1
AND	2	3
OR	3	2
EX-OR	4	5
EX-NOR	5	4
NAND	1	4
NOR	4	1