Foundation for Advancement of Science & Technology

CHAPTER # 02

- With 4 bits system we can ~~count~~ represent upto 15 decimal numbers

∴

Largest decimal number $= 2^n - 1$

where n = no. of bits of a system.

- Weight structure of a binary no. is

$2^{n-1} \dots 2^3, 2^2 2^1 2^0 . 2^{-1} 2^{-2} \dots 2^{n-1}$

↳ Binary points

Where n is the no. of bits from binary point.

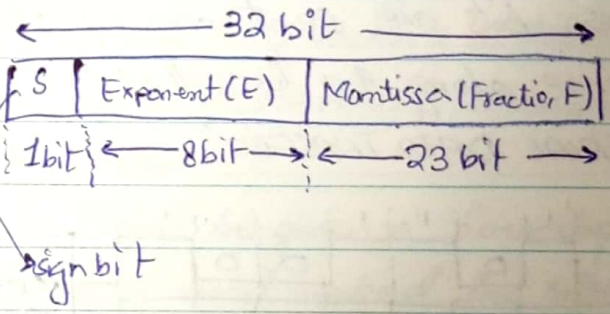
FLOATING POINT NUMBER

4.19×10^9 → Ko bits mein represent krne k do methods hai

- Single Precision 32-bit
- Double precision 64-bit Not in course!

Single Precision Floating Point Number!

it has a particular format, referred by IEEE 754 standard



→ in mantissa or fractional part, the binary point is understood to be the left of the 23 bit. effectively there are 24 bit in mantissa bcz in any binary number the left most bit is always a 1. Therefore this 1 is understood to be there although it doesnot occupy actual bit position.

Foundation for Advancement of Science & Technology

Example:

- 3.92×10^2 (isko $1.M \times 2^E$ mein convert krenge.)

$1.M \times 2^E$ → Exponent

2 → base

$E = E' + 127$

392

2	392 – 0
2	196 – 0
2	98 – 0
	49 – 1
	24 – 0
	12 – 0
	6 – 0
2	3 – 1
	1

$(1.10001000)_2$ ← we will place decimal after left most bit

1.10001000 ← mantissa ki pehli bit sign bit hogi

M → mantissa milgaya

For E:

$E = E' + 127$

$= 8 + 127$

$E = 135$ → isko ab binary mein krenge

135 – 1	
67 – 1	
33 – 1	
16 – 0	
8 – 0	
4 – 0	
2 – 0	
1	

$135 = 10000111$

$S = 1$

$E = 10000111$

$M = 10001000$

1	10000111	10001000000000000
S	E	M

→ 23 bit tak zeros likhenge

Example #2: 0.0110×2^6 → ye pehle se $1.M \times 2^E$ mein convert hoa hua hai.

1.10×2^{-2}

$M = 10$

$E' = -2$ → no. of bits after decimal

$E = E' + 127 = 125$

125 ka binary → 1111101

$E = 1111101$

0	1111101	10000---0
S	E	M

sign ki bit hum original question wale no. ko dekh kr lgayenge agr 3.07×10^{10} hai to 3 +ve to sign bit '0' agr -3.07×10^{10} to sign bit '1'.

- Floating point representation has 3 parts
  a. Mantisa
  b. Base
  c. Exponent. $\pm M \times B^E$

- Binary point floats to the right most sig 1 and an exponent is used

Number	Mantisa	Base	Expon
9×10^8	9	10	8
110×2^7	110	2	7
4364.784	4364784	10	-3

- Single Precision format: 32 bit

eg: $(1259.125)_{10}$ in Single precision format.

- Step#01 → First convert decimal to binary.

$(1259)_{10} \Rightarrow (10011100011)_2$
$(0.125)_{10} \Rightarrow (001)_2$

$(10011100011.001)_2$

Step#02 → Normalize the number

Formula → $(1.M)2^{E-127}$

(10011100011.001)

we shifted binary point to 10 places so

$1.0011100011001 \times 2^{10}$

↳ we have written in scientific notation

Step#03 →

$(1.M) \times 2^{(E-127)} = 1.0011100011001 \times 2^{10}$

M

Comparing exponents
$E - 127 = 10$
$E = 137$
$(137)_{10} \Rightarrow (10001001)_2$

0	10001001	00111001100000...0
1bit	8 bits	23 bit

↳ jo mantisa ki pheli bit hogi wohi sign bit hogi.

Foundation for Advancement of Science & Technology

Subtraction By 2's Complement:

A + (-B) =

Step #01 → Make 2's complement of Subtrahend.

Step#02 → Perform addition

Step#03 → The final carry bit is neglected & act as a sign bit. If it is 1 → one +ve
0 → one -ve

0001100 → minuend
0001001 → subtrahend

2's complement
1110110
+1
1110111

0 0 0 1 1 0 0
1 1 1 0 1 1 1
10000011

Shortcut of 2's complement, phele 1 encounter ho to usko chor kr baaqi sab ko invert kadenge.

Multiplication

01001101 → multiplicand
x 00000100 → multiplier
→ 4

1st method → traditional (x)
2nd method → convert multiplier to deimal
And Add the multiplicand that much times
Add multiplicand 4 times.

Ex #02:
01010011
(x) 11000101 → multiplier

Step#01 → take 2's complement of multiplier → 00111011
Step#02 → Multiply Multiplicand and multiplier and avoid sign bit.

Foundation for Advancement of Science & Technology

1 0 1 0 0 1 1
× 0 1 1 1 0 1 1

1 0 1 0 0 1 1
× 0 1 1 1 0 1 1
1 0 1 0 0 1 1
1 0 1 0 0 1 1 x
0 0 0 0 0 0 0 0 x x
1 0 1 0 0 1 1 x x x
x x x x

"ARITHMETIC OPERATIONS"

ADDITION

These can be 4 cases:

1. Both +ve no.
2. +ve no. with magnitude large than -ve no.
3. -ve no. with mag larger than +ve no.
4. Both -ve no.

CASE #01:

0 0 0 0 0 1 1 1	7
0 0 0 0 0 1 0 0	+4
0 0 0 1 0 1 1	11

- The sum is +ve ∴ in True (uncomplemented) binary → Addition of 2 +ve no. yeilds a +ve no.

CASE #02:

0 0 0 0 1 1 1 1	15
1 1 1 1 1 0 1 0	+ -6
1 0 0 0 0 1 0 0 1	9

↳ discard the carry → isko delete karo +ve or -ve ke liye

- the final carry bit is discarded. The sum is +ve & therefore in True (uncomplemented) binary

CASE # 03:

```
  00010000    16
+ 11101000  + -24
  11111000    -8
```

2's comple 0 000 1000

- The sum is -ve & ∴ in 2's complement form
- Addition of a +ve no. and a larger -ve no. yeilds a -ve no. in 2's complement

- -ve numbers saare 2's complement form mein honge agr hum unka wapis 2's complement lein to original numbers mil jayega

CASE # 04:

```
  11111011    -5
+ 11110111  + -9
 (1)11110010  -14
```

↳ discarded

128 64 32 16 8 4 2 1

2's comple → 0 0 0 0 1 1 1 0 → 14

- Final carry bit is discarded. The sum is -ve & ∴ in 2's complement form

Overflow condition:

- If the sign bit of the result is different than the sign bit of the numbers that are added, overflow is indicated.

both are +ve result should also be +ve

```
 (0)1111101    125
+(0)0111010  + 58
 (1)0110111    183
```

sign incorrect

↳ magnitude incorrect

Yaha 183 ko represent krne k liye 8 bits required hai jab k hamare pass sirf 7 bits ha for magnitude ek sign bit hai. There is a carry into the sign bit which produces overflow condition

- Adding Series of Numbers:

01000100, 00011011, 00001110, 00010010.

```
  01000100    68
+ 00011011  + 27
  01011111    95
  00001110  + 14
  01101101    109
  00010010  + 18
  01111111    127   Final Sum
```

Foundation for Advancement of Science & Technology

SUBTRACTION

- It is a special case of addition.
- The subtraction operation changes the sign of the Subtrahend & adds it to the minuend.
- The sign of a +ve or -ve binary no. is changed by taking 2's complement.

eg: When we take 2's complement of +ve no. 00000100 (+4) we get

11111100 (-4)

$-128 + 64 + 32 + 16 + 8 + 4 = -4$

- To Subtract two signed numbers take 2's complement of the Subtrahend & add. Discard any final carry bit. Both no. must have same no. of bits.

(a) 8 - 3: or 8 + (-3)

-3 ka 2's complement lenge

00001000 → (+8) minuend

\+ 11111101 → 2's complement of subtrahend (-3)

discard ← (1)00000101 → and +5

(b) 12 - (-9) = ?

-9 ka 2's complement + 9

000011100 → 12

\+ 000001001 → +9

000101.01 → 21

(c) -25 - 19

ismein hum +19 ka 2's lenge

+19

00010011

2's complement

11101101 → -19

2	19-1
2	9-1
2	4-0
2	2-0
	1

00011001 → 25

11100111 → -25

2	25-1
2	12-0
2	6-0
2	3-1
	1

11100111

\+ 11101101

128 64 32 16 8 4 2 1

(1)11010100 → -44

discard ← sign bit

-128 bhi krenge!

64 16 4 84

84 - 128 = -44

Foundation for Advancement of Science & Technology

(MULTIPLICATION)

- Two Methods:
1) Direct Addition Method
2) Partial Products

— DIRECT ADDITION —

→ you add the multiplicand a number of times to the multiplier.

→ Disadvantage → it becomes lengthy if multiplier is a larger number.

→ When two binary no. are (x) both must be in true (uncomplemented) form.

Q: 01001101 x 00000100
(multiplicand) (multiplier)

```
  01001101   1st time
  01001101   2nd time
  ----------
  10011010   Partial Sum
  01001101   3rd time
  ----------
  11100111   Partial Sum
  01001101   4th time
  ----------
  10011010 0 → product
```

∵ the sign of the multiplicand is 0, it has no effect on the outcome.

- All of the bits in the product are magnitude bits

— PARTIAL PRODUCT —

- ye wohi traditional method hai.

- The sign of product of a multiplication depends on the sign of the multiplicand and the multiplier, acc to the following two rules.

1) If the signs are the same, the product is +ve
2) If the signs are diff, the product is -ve.

→

Step#01: Determine the signs of multiplicand & multiplier. This tells the sign of product.

Step#02: Change any -ve no. to true (uncomplemented) form. it means take 2's complement of a -ve no.

Step#03: perform the traditional multiplication.

Step#04: If the sign bit that was determined in step 1 is -ve take the 2's complement of the product. If +ve leave the product in true form. Attach the sign bit to the product.

Q: 01010011 ⊗ 11000101

1 → sign bits are different so product will ~~also~~ be -ve.

2 → Taking 2's complement of multiplier

11000101 | 00111011

3 →

```
      10100111
      01111011
      ---------
      10100111   → 1st partial product
    + 10100111 x → 2nd partial product
      ---------
      11111001   → sum of 1st & 2nd
    0 0000000 x x → 3rd partial product
      ---------
    '0'1'1'1'1'1'0'0'1  Sum
    10100111 x x x
    ---------------
    1110010001
          :
    ---------------
    100111001000001  → Final Product
```

4→ Since sign of the product is as 1 as determined in Step#01, take the 2's Complement of the product

01100111011111

Now attach the sign bit.

1 01100111011111

↳ sign bit attached.

(DIVISION)

dividend = quotient
divisor

Step#00

- If the signs of dividend and divisor are,

⇒ same → quotient +ve

⇒ different → quotient -ve

- When two binary are divided they must be in true form.

Step#01: Same as of product. Initialize quotient to ZERO.

Step#02: Subtract the divisor from the dividend using 2's complement addition to get the first partial remainder and add 1 to quotient. If this partial remainder is +ve, go to step#03. If the partial remainder is 0 or -ve, the division is complete.

Step#03: Subtract the divisor from the partial remainder and add 1 to the quotient. If the result is +ve, repeat for the next partial remainder If the result is ZERO or -ve, the division is complete.

- Continue to subtract the divisor from the dividend & the partial remainders until there is a zero or -ve result.
- Count the no. of times that the divisor is subtracted and you have the quotient.

Q# 01100100 ÷ 00011001

→ both are positive quotient will also be +ve. Initially quotient is zero: 00000000

2→ 01100100 → Dividend

+ 11100111 → 2's complement of divisor

01001011 → +ve 1st partial remainder.

Add 1 to quotient → 00000001

Step 3 → Subtract the divisor from 1st partial remainder using 2's.

01001011 → 1st partial remainder
11100111 → 2's of divisor
00110010 → +ve 2nd partial remainder

Add 1 to quotient → 00000001 + 00000001 = 00000010

'0'0'1'1'0'0'1 0 → 2nd partial
11100111 → 2's divisor
discard ← (1) 00011001 → +ve 3rd

'''''''
00011001
+ 11100111
discard ← (1) 00000000 → ZERO remainder

Final quotient → 00000100 (as we have performed the subtraction 4 times)

convert 4 into binary.

HEXADECIMAL

— (Addition) —

Step #01: In any given column of an addition problem, think of the two hexadecimal digits in terms of their decimal values. For eg: $5_{16} = 5_{10}$ & $C_{16} = 12_{10}$

Step #02: If the sum of these two digits is 15 or less, bring down the corresponding decimal digit.

Step #03: If the sum of these two digits is greater than 15_{10} bring down the ammount of the sum that exceeds 16_{10} and carry a 1 to the next column.

Foundation for Advancement of Science & Technology

Q: $23_{16} + 16_{16}$

$$\begin{array}{r} 2\ 3 \\ 1\ 6 \\ \hline (3\ 9)_{16} \end{array}$$

$$\begin{array}{r} 5\ 8_{16} \\ 2\ 2_{16} \\ \hline (7\ A)_{16} \end{array}$$

$8 + 2 = 10_{16} = A$

$$\begin{array}{r} 2\ B \rightarrow 11 \\ +\ 8\ 4 \quad {\scriptstyle +4 \atop 14 \rightarrow F} \\ \hline (A\ F)_{16} \end{array}$$

$$\begin{array}{r} DF \rightarrow F_{16} + C_{16} = 15_{10} + 12_{10} = 27_{10} \\ AC \quad 27 - 16 = 11_{10} = B_{16} \text{ with a 1 carry} \\ \hline (18B)_{16} \end{array}$$

↑ right column

left col:

$D_{16} + A_{16} + 1_{16} = 13_{10} + 10_{10} + 1_{10} = 24_{10}$

$24_{10} - 16_{10} = 8_{10} = 8_{16}$ with a 1 carry

(SUBTRACTION)

(BCD CODE)

- Binary coded decimal (BCD) is a way to express each of the decimal digits with a binary code.
- In this code each decimal digit is represented by a 4-bit binary number

→ positional weights are 8-4-2-1

→ Sometimes BCD code is called 8421 code.

Decimal	BCD code (8 4 2 1)	
0	0 0 0 0	Valid BCD code
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	

to 1100
1101
0001 1001
1 9

$(C)_{16}$ = 12
$+(D)_{16}$ = 13
$(19)_{16}$

(25)$_{10}$

0001

Invalid Codes: We are only representing 10 out of 16 combinations in 8421 code. The rest of the combinations 10 (1010), 11 (1011), 12 (1100), 13 (1101), 14 (1110), 15 (1111) are called invalid Code.

- The numbers in decimal beyond 9 when converted into BCD are called Packed BCD.

★ Conversion of Decimal to BCD

i) $(17)_{10}$ → 0001 0111

0001 0111

digits
dono ~~numbers~~ ka alag alag BCD likhenge.

ii) $(156)_{10}$ → 0001 0101 0110

0001 0101 0110

★ Comparision of Binary & BCD :

	Binary	BCD
	it uses 4 bits	it uses 8 bits
$(10)_{10}$ →	1010	0001 0000
$(12)_{10}$ →	1100	0001 0010

→ BCD is less efficent than Binary as it consumes more no. of bit

→ Arithemetic operation is more complex in BCD

★ Conversion of BCD to Decimal

pehle se '0' lgadenge

i) 0001 0100 → 14

1 4

4 ka grp bnayenge.

ii) 0100 1001 → 49

(4 9)$_{10}$

Foundation for Advancement of Science & Technology

(BCD Addition)

CASE #01:

Sum ≤ 9, Final Carry = 0, Valid

CASE #02:

Sum ≤ 9, Final carry = 1, Invalid

Add 6 (0110) ↵

CASE #03:

Sum > 9, Final Carry = 0, Invalid

add 6 (0110) ↵

Q: $(2)_{10} + (6)_{10} = ?$

```
  0 0 1 0   → 2
+ 0 1 1 0   → 6      Sum<9
-----------
  1 0 0 0   → (8)
```

carry = 0 Ans Valid

$(3)_{10} + (7)_{10}$

```
  0 0 1 1   → 3.
+ 0 1 1 1   → 7      Sum>9
-----------
  1 0 1 0   → (10)
```

Carry = 0 Valid

Now adding 6 (0110)

```
  1 0 1 0
+ 0 1 1 0
-----------
0 0 0 1 | 0 0 0 0   → Ans.
  1       0
```

$(10)_{10}$

Q: $(8)_{10} + (9)_{10}$

```
  1 0 0 0
+ 1 0 0 1
-----------
(1) 0 0 0 1   → Sum < 9
 ↓
F.C = 1
```

Now add 6

```
  1 0 0 0 1
+   0 1 1 0
-----------
0 0 0 1 | 0 1 1 1
  1       7   → (17)10
```

Q: $(57)_{10} + (26)_{10}$

57

26

0101 0111 (57)

+ 0010 0110 (26)

0111 1101

ye valide hai q k less than 9

ye 13 hai or greater than 9 yaha 6 add krenge.

0111 1101

+ 0110

10000 0011

8 3 Ans

$(83)_{10}$

GRAY CODE

→ unweighted

Binary → Gray Code

Step#01: The MSB (left-most) in the Gray code is the same as corresponding MSB in binary no.

Step#02: Going from left to right, add each adjacent pair of binary code bits to get the next Gray code bit.
- Discard carries!

eg: $(10110)_2$ → Gray code

1 0 1 1 0 Binary

1 1 1 0 1 Gray

carry discarded

Foundation for Advancement of Science & Technology

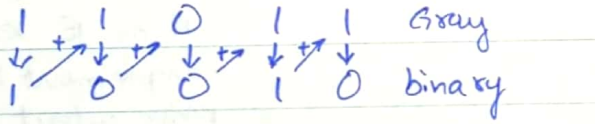
CHP#03

LOGIC GATES

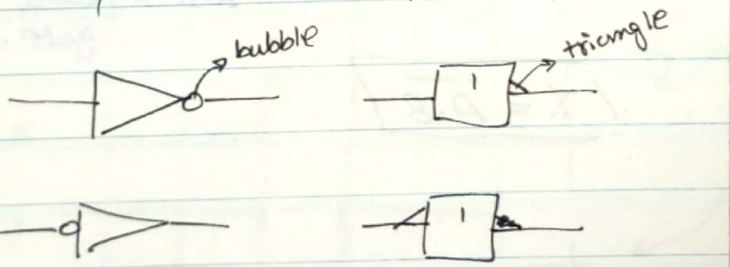
Gray Code → Binary

Step#01: MSB in the binary code is the same as the corresponding bit in the Gray Code

Step#02: Add each binary code bit generated to the Gray Code bit in the next adjacent position. Discard carries.



(NOT GATE)

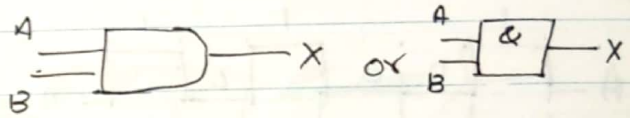


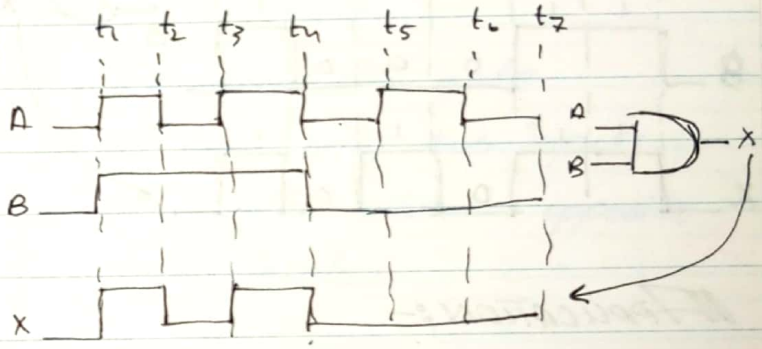
distinctive shape symbol with negation indicator

Rectangular outline Symbols with polarity indicator

- bubble indicates conversion or complementation when appears on input or output.

(AND GATE)



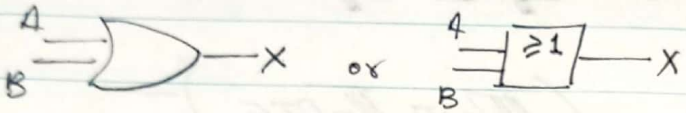


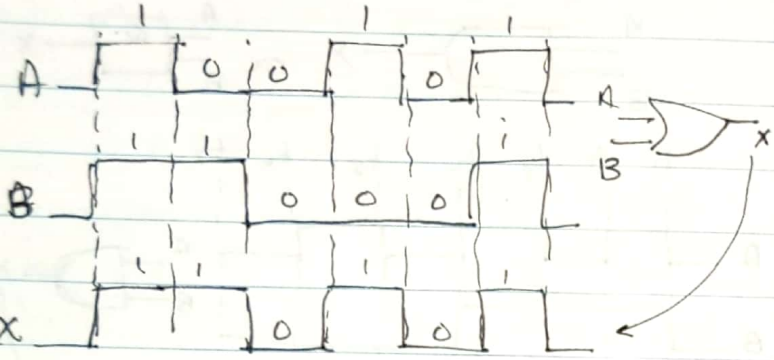
The output X is high only when both A & B are high.

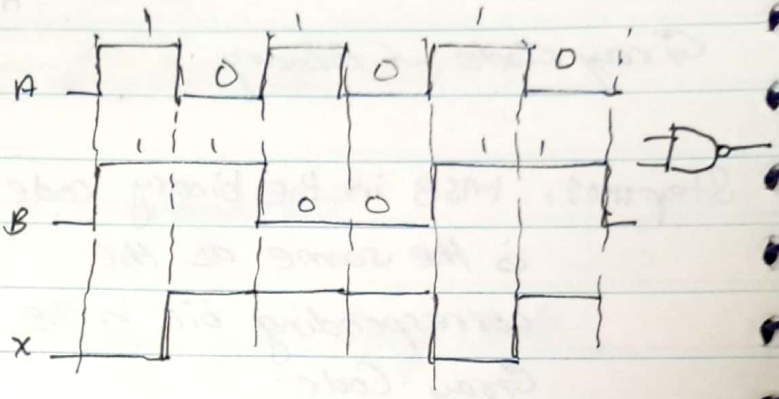
APPLICATIONS:

1) The AND gate as an enabler/ Inhibit Device
2) A seat belt Alarm system diagrams from book.

(OR GATE)

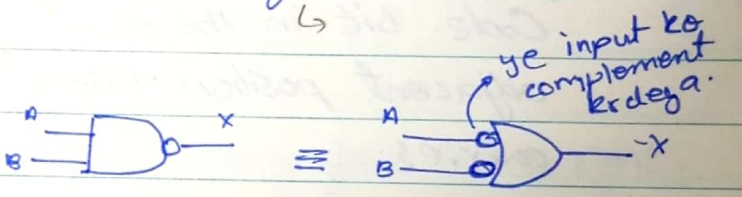






The NAND gate can be used as a Negative-OR.

↳



↳ is mein phele A or B ka complement krega phir output milega

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

Applying DeMorgan

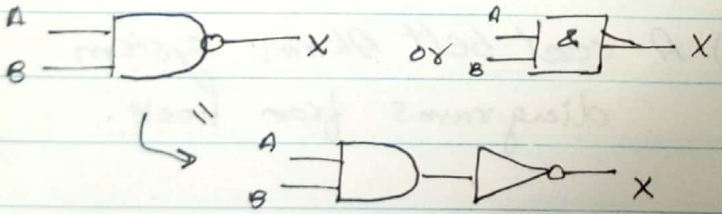
$X = \overline{A \cdot B}$ = Nand gate.

APPLICATION:-

1) A simplified intrusion detection system using an OR gate

(NAND GATE)

- Contraction of NOT-AND gate



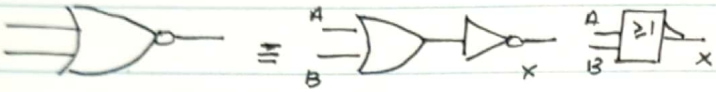
$\boxed{X = \overline{A \cdot B}}$

- Produces LOW when all are HIGH.

A	B	X
0	0	1
0	1	1
1	0	1
1	1	0

(NOR GATE)

- Contraction of NOT-OR gate.

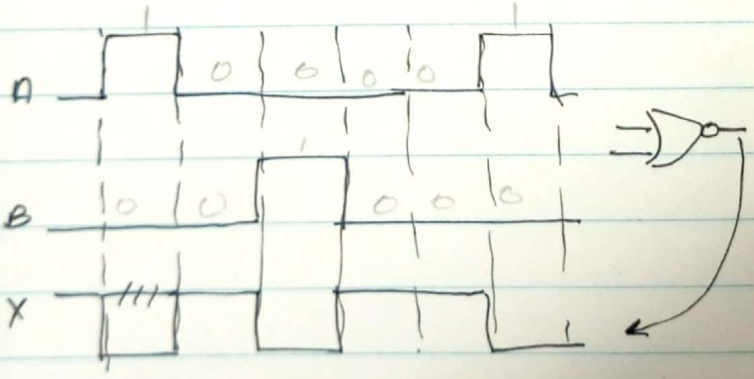


- It produces HIGH when all inputs are low.

A	B	X
0	0	1
0	1	0
1	0	0
1	1	0

$X = \overline{A+B}$

- This gives output opposite to OR gate.

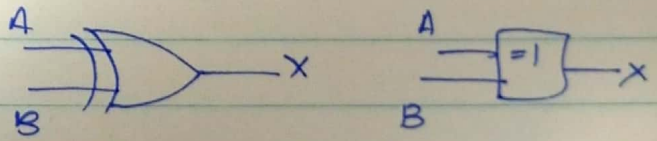


- Negative AND = NOR gate.

(XOR-GATE)

- Exclusive OR gate
- It has only two inputs.
- Only high when opposite logic levels.

A	B	X
0	0	0
1	0	1
0	1	1
1	1	0



- An XOR gate can be used as two bit adder

Input A	Input B	Output A+B	
0	0	0	
0	1	1	
1	0	1	without carry bit
1	1	0	→ 1

Foundation for Advancement of Science & Technology

(XNOR - GATE)

- Exclusive NOR
- Only 2 inputs
- Output opposite to XOR.
- When same input it produces high

A	B	X
0	0	1
0	1	0
1	0	0
1	1	1