MALAD KANDIVALI EDUCATION SOCIETY'S
NAGINDAS KHANDWALA COLLEGE OF COMMERCE,
ARTS & MANAGEMENT STUDIES & SHANTABEN NAGINDAS KHANDWALA COLLEGE OF SCIENCE
MALAD [W], MUMBAI – 64
(AUTONOMOUS)

(Reaccredited 'A' Grade by NAAC)
(AFFILIATED TO UNIVERSITY OF MUMBAI)
(ISO 9001:2015)

CERTIFICATE

Name:GUNJA SATRAJEET SINGH

Roll No: 88 **Programme**: BSc IT **Semester**: I

This is certified to be a bonafide record of practical works done by the above student in the college laboratory for the course **Fundamentals of Computers and Electronics** (Course Code: **1912UISPR**) for the partial fulfillment of First Semester of BSc IT during the academic year 2020-2021.

The journal work is the original study work that has been duly approved in the year 2020-2021 by the undersigned.

_____ _____

External Examiner **Ms. Vishakha Bagwe**
(Subject-In-Charge)

Date of Examination: **(College Stamp)**

Sr.no	Topic	Sign
1	Study and verify the truth table of various logic gates.	
2	Verify De-Morgan's Law	
3	Simplify given Boolean expression and realize it.	
4	Design and verify a half/full adder	
5	Design and verify half/full subtractor	
6	Design a 2 bit comparator using combinational circuits.	
7	Perform the following Operations related to memory locations: a. Store the data byte 32H into memory location 4000H. b. Exchange the contents of memory locations 2000H and 4000H	
8	Arithmetic operations: a. Subtract the contents of memory location 4001H from the memory location 2000H and place the result in memory location 4002H. b. Add the contents of memory location 4001H from the memory location 2000H and place the result in memory location 4002H.	

	c. Multiply the contents of memory location 4001H from the memory location 2000H	
9	Complement operations: a. Find the l's complement of the number stored at memory location 4400H and store the complemented number at memory location 4300H. b. Find the 2's complement of the number stored at memory location 4200H and store the complemented number at memory location 4300H.	
10	Logical operations: a. AND the contents of memory location 4001H from the memory location 2000H and place the result in memory location 4002H. b. OR the contents of memory location 4001H from the memory location 2000H and place the result in memory location 4002H c. X-OR the contents of memory location 4001H from the memory location 2000H and place the result in memory location 4002H.	

Practical No 1

Aim: Study and verify the truth table of various logic gates.

Basic Gates (AND, OR, NOT)

Theory:

- A logic gate is an elementary building block of a digital circuit. Most logic gates have two inputs and one output.
- At any given moment, every terminal is in one of the two binary conditions low (0) or high (1), represented by different voltage levels

.

- The logic state of a terminal can, and generally does, change often, as the circuit processes data.
- In most logic gates, the low state is approximately zero volts (0 V), while the high state is approximately five volts positive (+5 V).

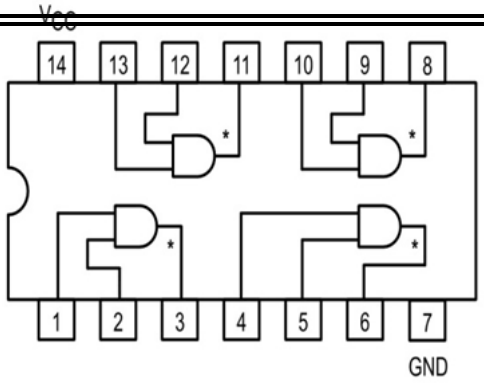
a) AND gate:

- The AND gate is an electronic circuit that gives a **high** output (1) only if **all** its inputs are high. A dot (.) is used to show the AND operation i.e. A.B
- Logic symbol

- Truth Table:

2 Input AND gate		
A	B	A.B
0	0	0
0	1	0
1	0	0
1	1	1

- Pin diagram:



- Output:

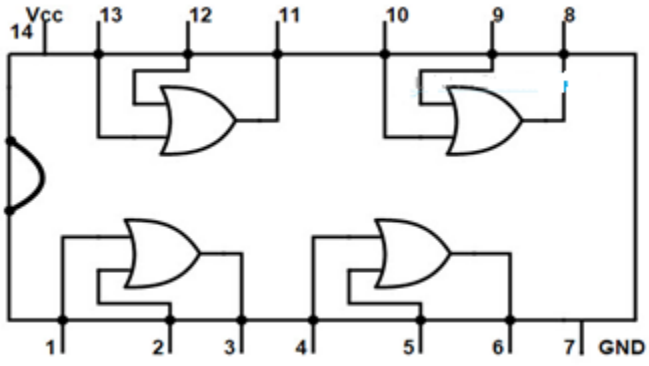
b) OR gate:

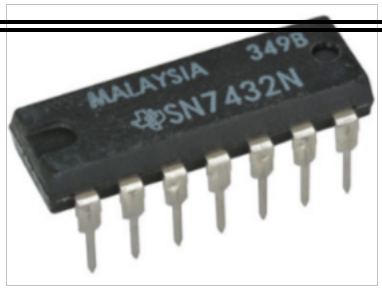
- The OR gate is an electronic circuit that gives a high output (1) if one or more of its inputs are high. A plus (+) is used to show the OR operation.
- Logic symbol:

- Truth table:

2 Input OR gate		
A	B	A+B
0	0	0
0	1	1
1	0	1
1	1	1

- Pin diagram:

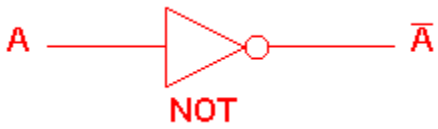




- Output:

c) NOT gate:

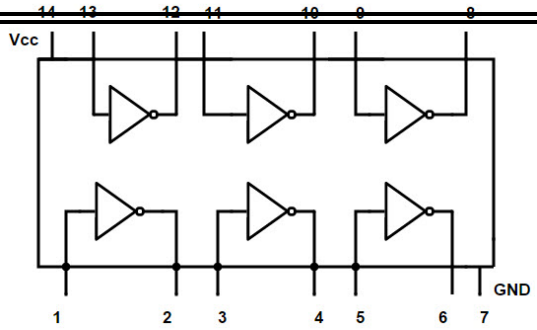
- The NOT gate is an electronic circuit that produces an inverted version of the input at its output.
- It is also known as an inverter. If the input variable is A, the inverted output is known as NOT A.
- This is also shown as A', or A with a bar over the top, as shown at the outputs.
- Logic symbol:



- Truth table:

NOT gate	
A	$\overline{A}$
0	1
1	0

- Pin diagram:



- Output:

Universal Gates (NAND, NOR)

Theory:

- A universal gate is a gate which can implement any Boolean function without need to use any other gate type.
- The NAND and NOR gates are universal gates.
- In practice, this is advantageous since NAND and NOR gates are economical andeasier to fabricate and are the basic gates used in all IC digital logic families.
- In fact, an AND gate is typically implemented as a NAND gate followed by an inverter
- Likewise, an OR gate is typically implemented as a NOR gate followed by an inverter

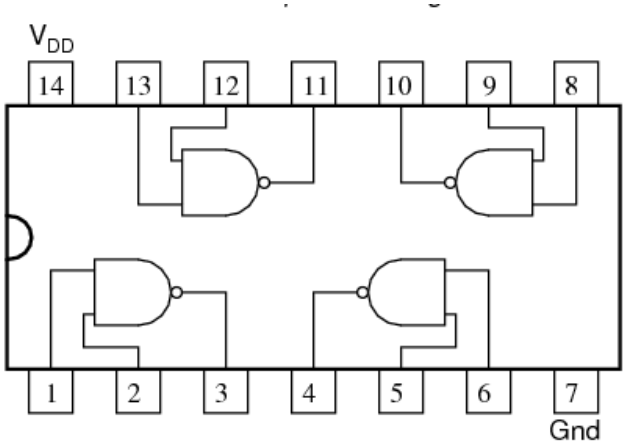
a) NAND gate:

- This is a NOT-AND gate which is equal to an AND gate followed by a NOT gate.
- The outputs of all NAND gates are high if any of the inputs are low.
- The symbol is an AND gate with a small circle on the output. The small circle represents inversion.
- Logic Symbol:

- Truth Table:

2 input NAND gate		
A	B	$\overline{A.B}$
0	0	1
0	1	1
1	0	1
1	1	0

- Pin diagram:



- Output:

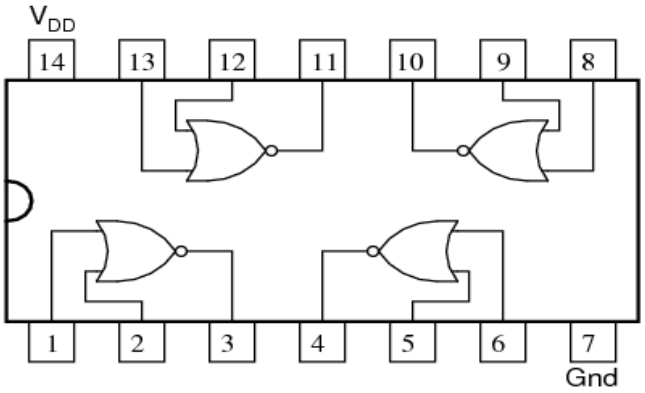
b) NOR gate:

- This is a NOT-OR gate which is equal to an OR gate followed by a NOT gate.
- The outputs of all NOR gates are low if any of the inputs are high.
- The symbol is an OR gate with a small circle on the output. The small circle represents inversion.
- Logic Symbol:

- Truth Table:

2 Input NOR gate		
A	B	$\overline{A+B}$
0	0	1
0	1	0
1	0	0
1	1	0

- Pin diagram:



- Output

Exclusive Gates (XOR, XNOR)

Theory:

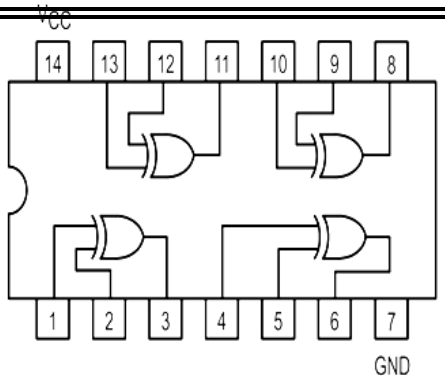
a) XOR gate:

- The '**Exclusive-OR**' gate is a circuit which will give a high output if **either, but not both**, of its two inputs are high. An encircled plus sign (⊕) is used to show the EOR operation.
- Logical symbol:

- Truth Table:

2 Input EXOR gate		
A	B	A⊕B
0	0	0
0	1	1
1	0	1
1	1	0

- Pin diagram:



- Output:

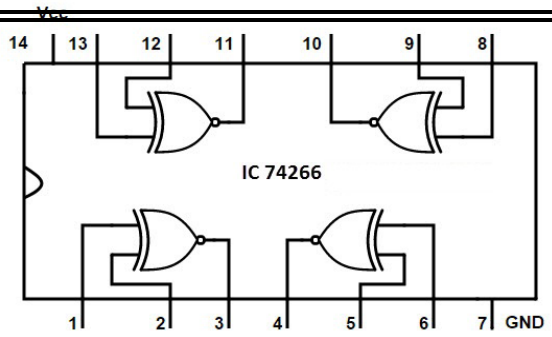
b) XNOR gate:

- The 'Exclusive-NOR' gate circuit does the opposite to the EOR gate.
- It will give a low output if either, but not both, of its two inputs are high.
- The symbol is an EXOR gate with a small circle on the output. The small circle represents inversion
- Logical symbol:

- Truth Table:

2 Input EXNOR gate		
A	B	$\overline{A \oplus B}$
0	0	1
0	1	0
1	0	0
1	1	1

- Pin diagram:



- Output:

Practical No 2

Aim: Verifying De Morgan's laws.

Theory:

De Morgan's Law:

- De Morgan has suggested two theorems which are extremely useful in Boolean algebra.

Theorem 1:

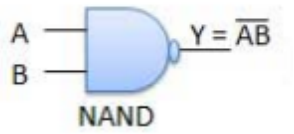
- Statement: It states that complement of a product is equal to the sum of the complements.

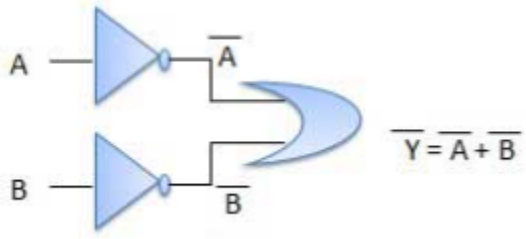
$\overline{A.B} = \overline{A} + \overline{B}$

NAND = Bubbled OR

- Description: The left hand side (LHS) of this theorem represents a NAND gate with inputs A and B, whereas the right hand side (RHS) of the theorem represents an OR gate with inverted inputs.This OR gate is called as Bubbled OR.

- Logic circuit:

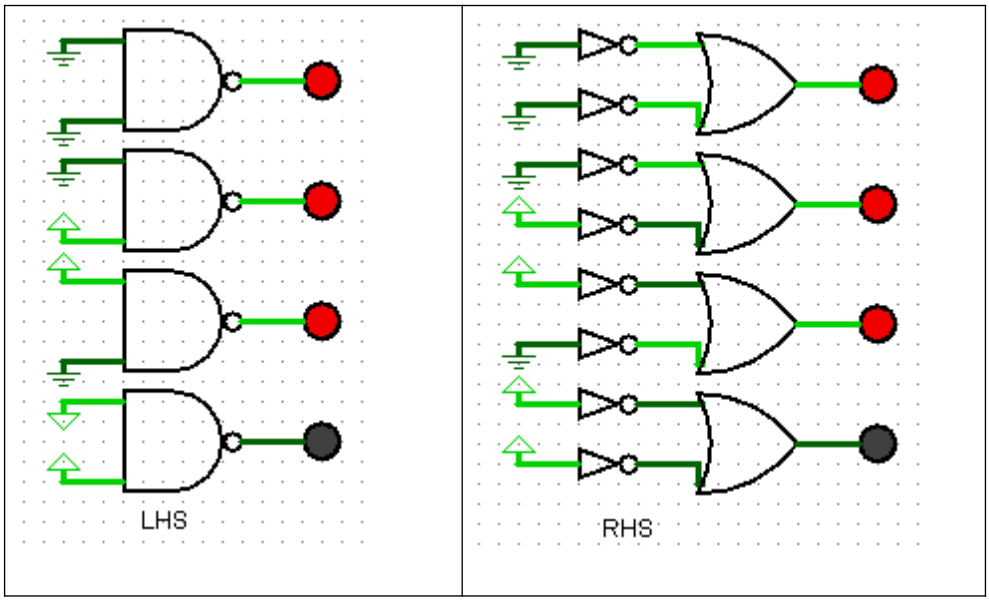




- Truth Table:

A	B	$\overline{AB}$	$\overline{A}$	$\overline{B}$	$\overline{A}+\overline{B}$
0	0	1	1	1	1
0	1	1	1	0	1
1	0	1	0	1	1
1	1	0	0	0	0

- Output:



Theorem 2:

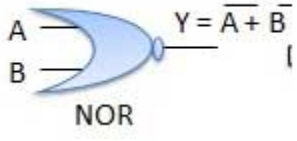
- Statement: It states that complement of the sum is equal to the product of the complement.

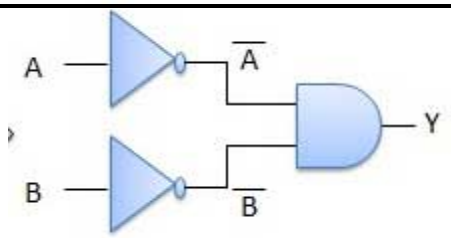
$$\overline{A+B} = \overline{A}.\overline{B}$$

NOR = Bubbled AND

- Description:The LHS of this theorem represents a NOR gate with inputs A and B, whereas the RHS represents an AND gate with inverted inputs.This AND gate is called as Bubbled AND.

- Logic circuit:





- Truth Table:

A	B	$\overline{A+B}$	$\overline{A}$	$\overline{B}$	$\overline{A}.\overline{B}$
0	0	1	1	1	1
0	1	0	1	0	0
1	0	0	0	1	0
1	1	0	0	0	0

- Output:

Practical No 3

Aim: Simplify given Boolean expression and realize it.

Theory:

a. Laws:

Boolean Laws

1. **Annulment Law :** A term AND'ed with a "0" equals 0 or OR'ed with a "1" will equal 1.
   - $A \cdot 0 = 0$ A variable AND'ed with 0 is always equal to 0.
   - $A + 1 = 1$ A variable OR'ed with 1 is always equal to 1.
2. **Identity Law :** A term OR'ed with a "0" or AND'ed with a "1" will always equal that term.
   - $A + 0 = A$ A variable OR'ed with 0 is always equal to the variable.
   - $A \cdot 1 = A$ A variable AND'ed with 1 is always equal to the variable.
3. **Idempotent Law :** An input that is AND'ed or OR'ed with itself is equal to that input.
   - $A + A = A$ A variable OR'ed with itself is always equal to the variable.
   - $A \cdot A = A$ A variable AND'ed with itself is always equal to the variable.
4. **Complement Law :** A term AND'ed with its complement equals "0" and a term OR'ed with its complement equals "1".
   - $A \cdot \bar{A} = 0$ A variable AND'ed with its complement is always equal to 0.
   - $A + \bar{A} = 1$ A variable OR'ed with its complement is always equal to 1.
5. **Commutative Law :** The order of application of two separate terms is not important.
   - $A \cdot B = B \cdot A$ The order in which two variables are AND'ed makes no difference.
   - $A + B = B + A$ The order in which two variables are OR'ed makes no difference.
6. **Double Negation Law :** A term that is inverted twice is equal to the original term.
   - $\bar{\bar{A}} = A$ A double complement of a variable is always equal to the variable.
7. **Distributive Law :** This law permits the multiplying or factoring out of an expression.
   - $A(B + C) = A.B + A.C$ (OR Distributive Law)
   - $A + (B.C) = (A + B).(A + C)$ (AND Distributive Law)
8. **Absorptive Law :** This law enables a reduction in a complicated expression to a simpler one by absorbing like terms.
   - $A + (A.B) = A$ (OR Absorption Law)
   - $A(A + B) = A$ (AND Absorption Law)
9. **Associative Law :** This law allows the removal of brackets from an expression and regrouping of the variables.
   - $A + (B + C) = (A + B) + C = A + B + C$ (OR Associate Law)
   - $A(B.C) = (A.B)C = A \cdot B \cdot C$ (AND Associate Law)

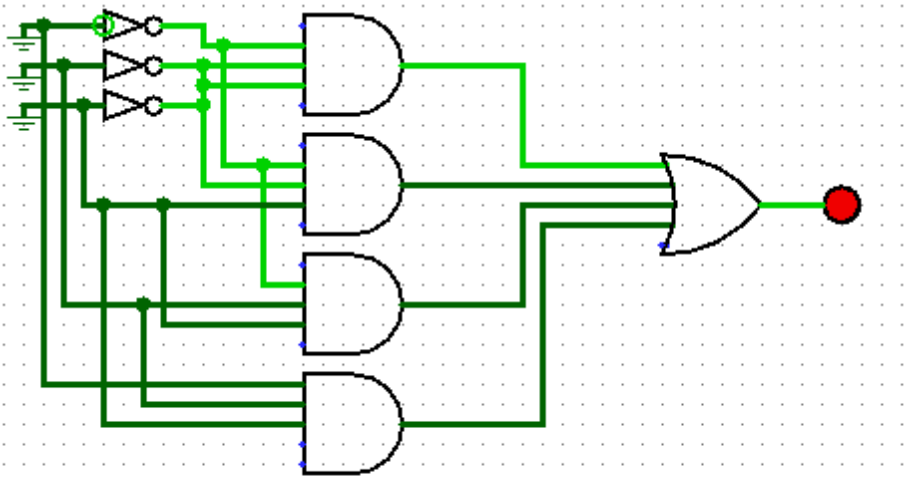
b. Simplification:

$$
\begin{aligned}
f_1 &= \bar{x}_1\bar{x}_2\bar{x}_3 + \bar{x}_1\bar{x}_2x_3 + \bar{x}_1x_2x_3 + x_1x_2x_3 \\
&= \bar{x}_1\bar{x}_2(\bar{x}_3 + x_3) + (\bar{x}_1 + x_1)x_2x_3 \\
&= \bar{x}_1\bar{x}_2 \cdot 1 + 1 \cdot x_2x_3 \\
&= \bar{x}_1\bar{x}_2 + x_2x_3
\end{aligned}
$$

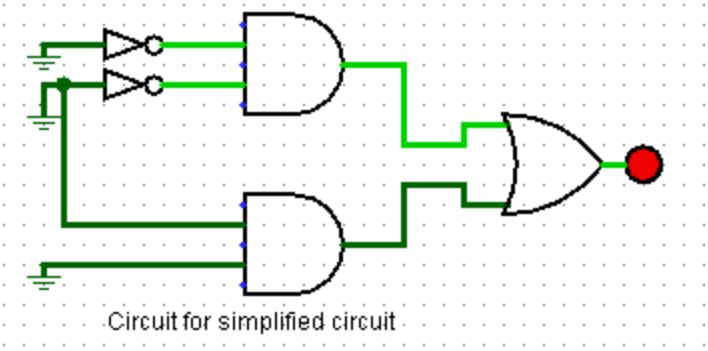
c.Truth Table:

X_1	X_2	X_3	O/P of Original Circuit	O/P for Simplified Circuit	
0	0	0	1	1	
0	0	1	1	1	
0	1	0	0	0	
0	1	1	1	1	
1	0	0	0	0	
1	0	1	0	0	
1	1	0	0	0	
1	1	1	1	1	

d. Output:



Circuit for given Expression



Circuit for simplified circuit

Practical No 4

Aim: Design and verify a half/full adder

Theory:

ADDER

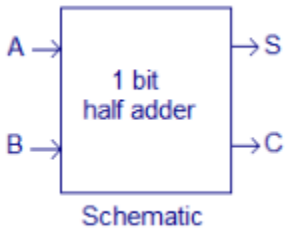
- Adder circuit is a combinational digital circuit that is used for adding two numbers. A typical adder circuit produces a sum bit (denoted by S) and a carry bit (denoted by C) as the output.

- Typically adders are realized for adding binary numbers but they can be also realized for adding other formats like BCD (binary coded decimal, XS-3 etc.
- Adder circuits are of two types: Half adder ad Full adder

1. Half Adder:
    - Half adder is a combinational arithmetic circuit that adds two numbers and produces a sum bit (S) and carry bit (C) as the output.
    - IfA and B are the input bits, then sum bit (S) is the X-OR of A and B and the carry bit (C) will be the AND of A and B.
    - From this it is clear that a half adder circuit can be easily constructed using one X-OR gate and one AND gate.

    - Half adder is the simplest of all adder circuit, but it has a major disadvantage

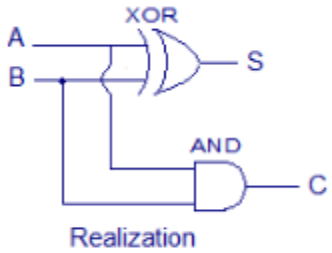
    - The half adder can add only two input bits (A and B) and has nothing to do with the carry if there is any in the input.
    - So if the input to a half adder have a carry, then it will be neglected it and adds only the A and B bits.
    - That means the binary addition process is not complete and that's why it is called a half adder.
    - The truth table, schematic representation and XOR//AND realization of a half adder are as follows:



Schematic

Inputs		Outputs	
A	B	S	C
0	0	0	0
1	0	1	0
0	1	1	0
1	1	0	1

Truth table



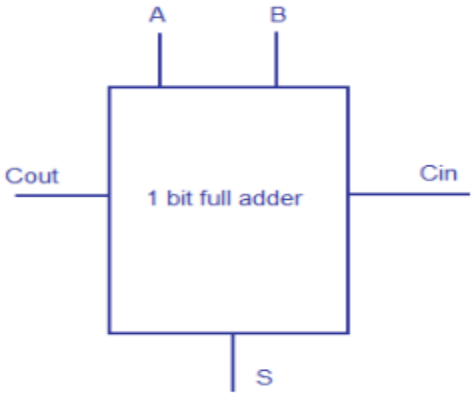
Realization

2. Full Adder:

- Full adder is a logic circuit that adds two input operand bits plus a Carry in bit and outputs a Carry out bit and a sum bit.

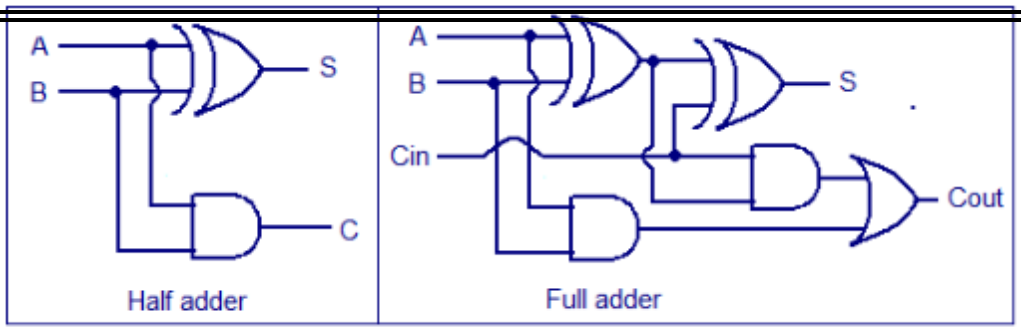
- The Sum Out (Sout) of a full adder is the XOR of input operand bits A, B and the Carry in (Cin) bit.

Inputs			Outputs	
A	B	Cin	Cout	S
0	0	0	0	0
1	0	0	0	1
0	1	0	0	1
1	1	0	1	0
0	0	1	0	1
1	0	1	1	0
0	1	1	1	0
1	1	1	1	1



1 bit full adder truth table & schematic

- A Full adder can be made by combining two half adder circuits together (a half adder is a circuit that adds two input bits and outputs a sum bit and a carry bit).



Full adder & half adder circuit

Practical No 5

SUBTRACTOR

Aim: Design and verify half/full subtractor

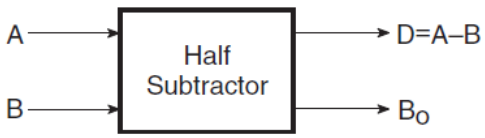
Theory:

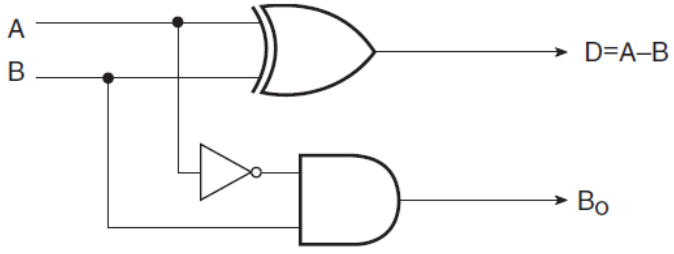
- Subtractor is the one which used to subtract two binary number(digit) and provides Difference and Borrow as a output.
- In digital electronics we have two types of subtractor.
- They are of two types: Half and Full Subtractor

1. Half Subtractor:
    - Half Subtractor is used for subtracting one single bit binary digit from

      another single bit binary digit.
    - The truth table of Half Subtractor is shown below:

A	B	D	B_O
0	0	0	0
0	1	1	1
1	0	1	0
1	1	0	0



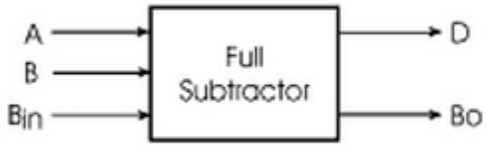


A	B	A'	A'.B	A xor

			(Borrow)	B(Difference)
0	0	1	0	0
0	1	1	1	1
1	0	0	0	1
1	1	0	0	0

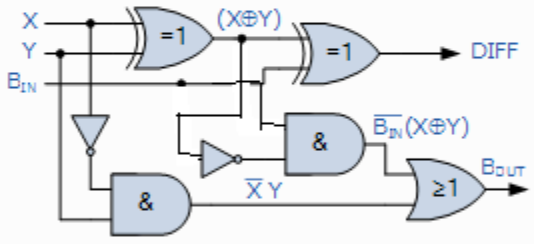
2. Full Subtractor:

- A logic Circuit Which is used for Subtracting Three Single bit Binary digit is known as Full Subtractor.



- The Truth Table of Full Subtractor is Shown Below:

Symbol	Truth Table				
	B-in	Y	X	Diff.	B-out
	0	0	0	0	0
	0	0	1	1	1
	0	1	0	1	1
	0	1	1	0	1
	1	0	0	1	0
	1	0	1	0	0
	1	1	0	0	0
	1	1	1	1	1



Practical No 6

Aim: Design a 1 bit and 2 bit comparator using combinational circuits.

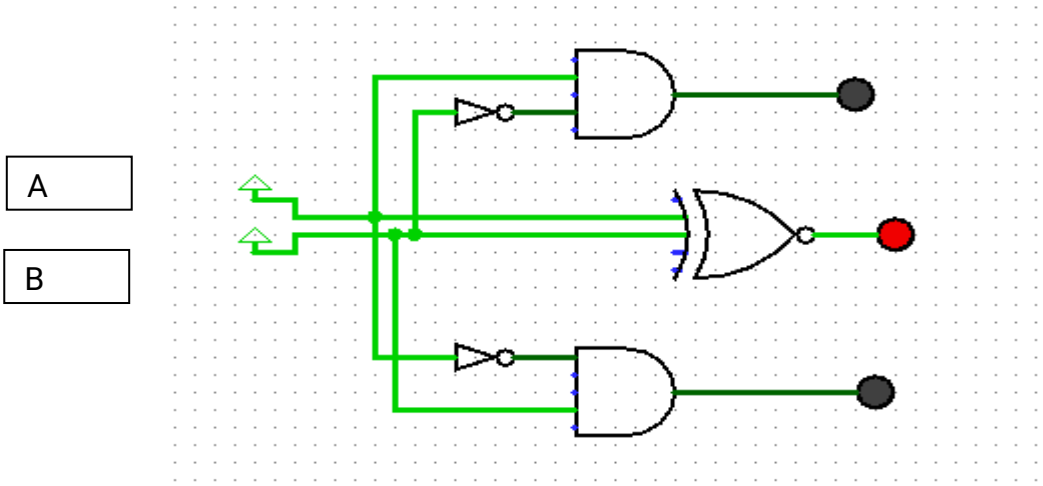
Theory:-

Data comparison is needed in digital systems while performing arithmetic or logical operations. This comparison determines whether one number is greater than, equal, or less than the other number. A digital comparator is widely used in combinational system and is specially designed to compare the relative magnitudes of binary numbers.

1-bit Comparator:-

A comparator used to compare two bits is called a single bit comparator. It consists of two inputs each for two single bit numbers and three outputs to generate less than, equal to and greater than between two binary numbers.

- Circuit diagram



Truth table

A	B	A>B	A=B	A<B
0	0	0	1	0
0	1	0	0	1
1	0	1	0	0
1	1	0	1	0

- K-Map for output

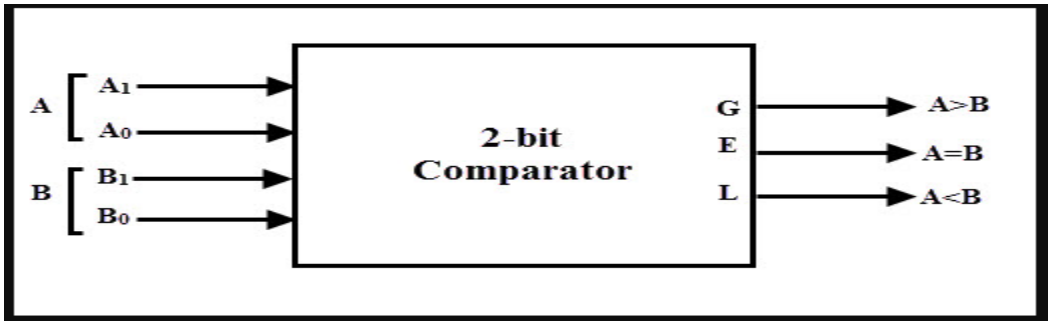
  (students will draw on their own)

2-bit comparator:-

- A 2-bit comparator compares two binary numbers, each of two bits and produces their relation such as one number is equal or greater than or less than the other. The figure below shows the block diagram of a two-bit

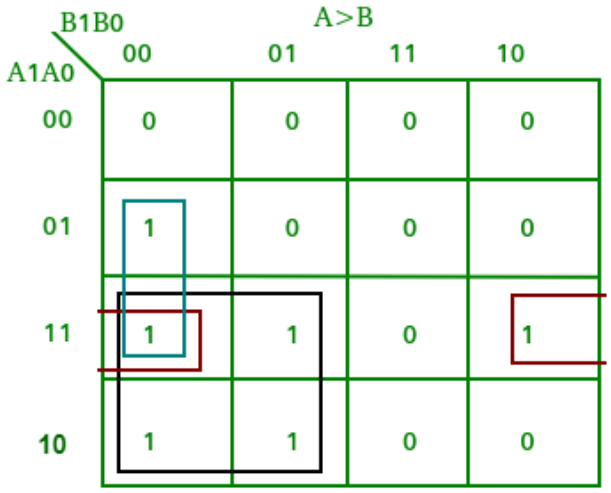
comparator which has four inputs and three outputs.

- The first number A is designated as A = A1A0 and the second number is designated as B = B1B0. This comparator produces three outputs as G (G = 1 if A>B), E (E = 1, if A = B) and L (L = 1 if A<B).



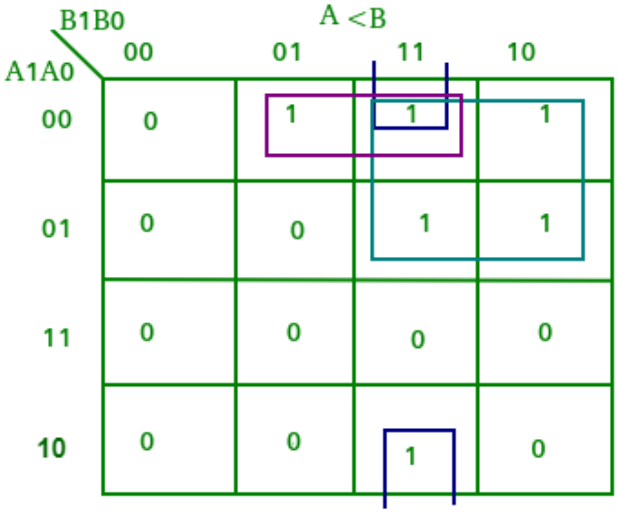
Truth table:-

INPUT				OUTPUT		
A1	A0	B1	B0	A<B	A=B	A>B
0	0	0	0	0	1	0
0	0	0	1	1	0	0
0	0	1	0	1	0	0
0	0	1	1	1	0	0
0	1	0	0	0	0	1
0	1	0	1	0	1	0
0	1	1	0	1	0	0
0	1	1	1	1	0	0
1	0	0	0	0	0	1
1	0	0	1	0	0	1
1	0	1	0	0	1	0
1	0	1	1	1	0	0
1	1	0	0	0	0	1
1	1	0	1	0	0	1
1	1	1	0	0	0	1
1	1	1	1	0	1	0



Kmap:
(A0~ B1 ~ B0) +(A1~B1)+ (A1A0~B0)

(~A1~A0~B1~B0)+ (~A1A0~B1B0) + (A1A0B1B0) + (A1~A0B1~B0)

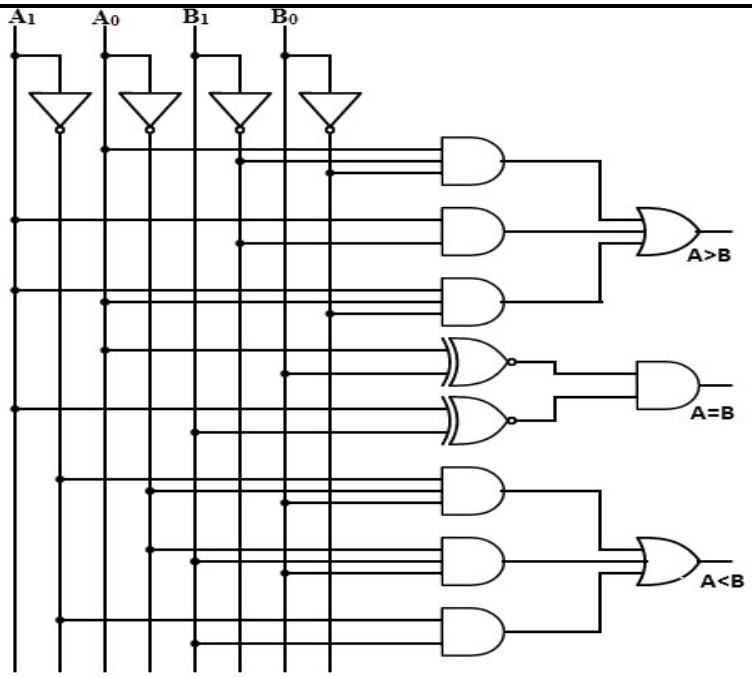


(~A1~A0B0) + (~A1B1) + (~A0B1B0)

From the above K-maps logical expressions for each output can be expressed as follows:

```
A>B:A1B1' + A0B1'B0' + A1A0B0'
A=B: A1'A0'B1'B0' + A1'A0B1'B0 + A1A0B1B0 + A1A0'B1B0'
   : A1'B1' (A0'B0' + A0B0) + A1B1 (A0B0 + A0'B0')
   : (A0B0 + A0'B0') (A1B1 + A1'B1')
   : (A0 Ex-Nor B0) (A1 Ex-Nor B1)
A<B:A1'B1 + A0'B1B0 + A1'A0'B0
```

Circuit diagram:-



PRACTICAL -7

STATEMENT:-
Store the data byte 32H into memory location 400H.

4000H

32

```
   MVI   A, 32H
     STA   4000H

   HLT
```

mVI- store data in accu/reg.

STA- reg to mem

LDA- mem to reg

STATEMENT:
Exchange the contents of memory location 2000H and 4000H

To exchange the data from the location we first need to store data at 2000H and

Data: 35H 65H

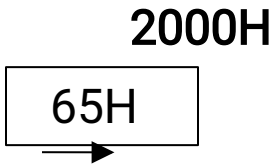
```
mVI   A, 32 H
STA   2000H
mVI   A, 65 H
STA   4000H
```

Now to exchange the data

First we need to load the data from memory

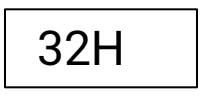
```
LDA  2000H          -  mem to acc
mov B,A             - A ▸  B

LDA 4000H           -  B-32 H   A-65 H
```




```
STA 2000H

mov A,B             B     A
STA 4000H                 4000H
HLT
```



PRACTICAL -8

a. Subtract the contents of memory location 4001H from the memory location 2000H and place the result in memory location 4002H.

2000H	4000H
16	11

```
LDA  4001 H
Mov  B,A
LDA   2000H
 SUB  B
STA  4002H
HLT
```

4002H
05

b. Add the contents of memory location 4001H from the memory location 2000H and place the result in memory location 4002H.

2000H	4000H
05	05

```
LDA  4001 H
MOV  B,A
LDA   2000
```

~~ADD B~~

```
STA  4002H
HLT
```

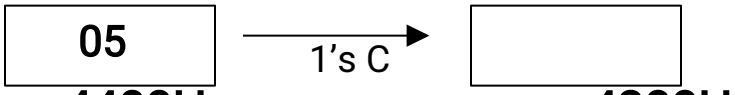
4002H

10

PRACTICAL-9

STATEMENT:-

Find the 1's complement of the number stored at memory location 4400H and state the complemented number at memory location 4300H.



4400H 4300H

(4400H)= 55H= 0101 0101

Result = (4300H) = 1010 1010 =AAH

```
mVI  A, 55H
                    prerequiste
STA  4400H
```

To find the complement first take the number from 4400H(load).

```
LDA   4400H
CMA
STA   4300H
HLT
```

Find the 2's complement of the number stored at memory location 4200H and store the complemented number at memory location 4300H.

4800H

(4200H)=55H = 0101 0101

1's 1010 1010
+ 1

1010 1011 (ABH)

4300H

Prerequistes
mVI A,55H
STA 4200H

To find 2's C
LDA 4200H - A 55
CMA - complement A
ADI , 01 H - add 1 to comple(A).
STA 4300 H - store at 4300H.
HLT.

PRACTICAL-10

a) AND the contents memory location 4001H from the memory location 2000H and place the result in memory location 4002H.

40001H 2000H

04 02

```
LDA   4000H
MOV   B,A

LDA
ANA   B
4002H
STA   4002H
HLT
```

4001H 2000H ans

04 02 → 00

b) OR the contents of memory location 4001H from the memory location 2000H and place the result in memory location 4002H.

40001H 2000H

04 02

```
LDA   4000H
MOV   B, A
LDA
ORA   B
4002H
STA   4002H
HLT
```

4001H 2000H ans

04 02 → 11