A

PRACTICAL FILE

ON

Computer Architecture And Organization(CS-404)

SUBMITTED PARTIAL FULFILMENT OF THE REQUIREMENT FOR THE AWARD OF

THE DEGREE OF

BACHELOR OF TECHNOLOGY

(COMPUTER SCIENCE & ENGINEERING)

SUBMITTED TO

RAJIV GANDHI PROUDYOGIKI VISHWAVIDYALAYA,BHOPAL

SUBMITTED TO:

Mr. Shushil

Dept. of CSE

SUBMITTED BY:

Aman Vishvkarma

Enroll no. 0928CS221026

II year/ IV semester

Department of Computer Science & Engineering

IPS COLLEGE OF TECHNOLOGY AND MANAGEMENT ,GWALIOR May 2024

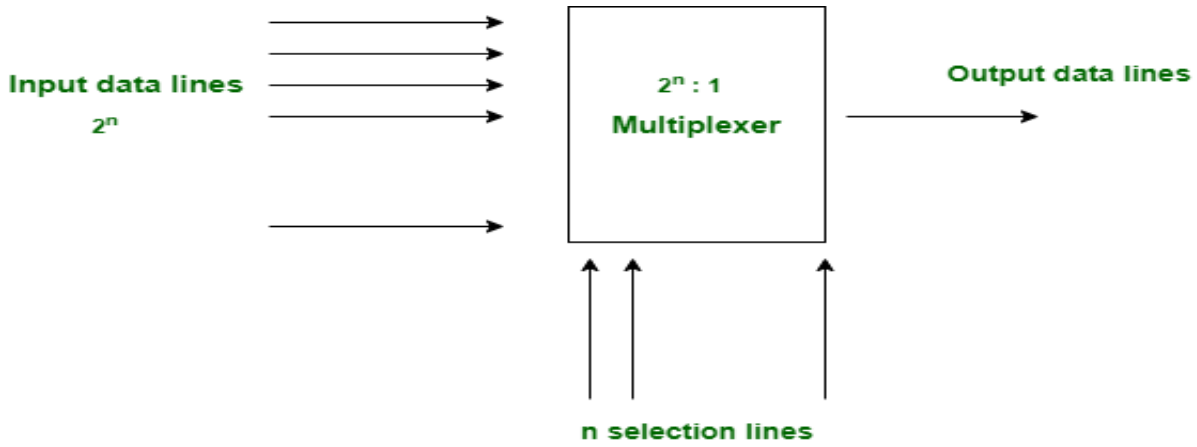
content

1.Study of multiplexer and demultiplexer.

Multiplexer :

A multiplexer is a data selector which takes several inputs and gives a single output. In multiplexer, we have 2n Input lines and 1 output lines where n is the number of selection lines.

The single-pole multi-position switch is a simple example of a non-electronic circuit of the multiplexer, and it is widely used in many electronic circuits. The multiplexer is used to perform high-speed switching and is constructed by electronic components.

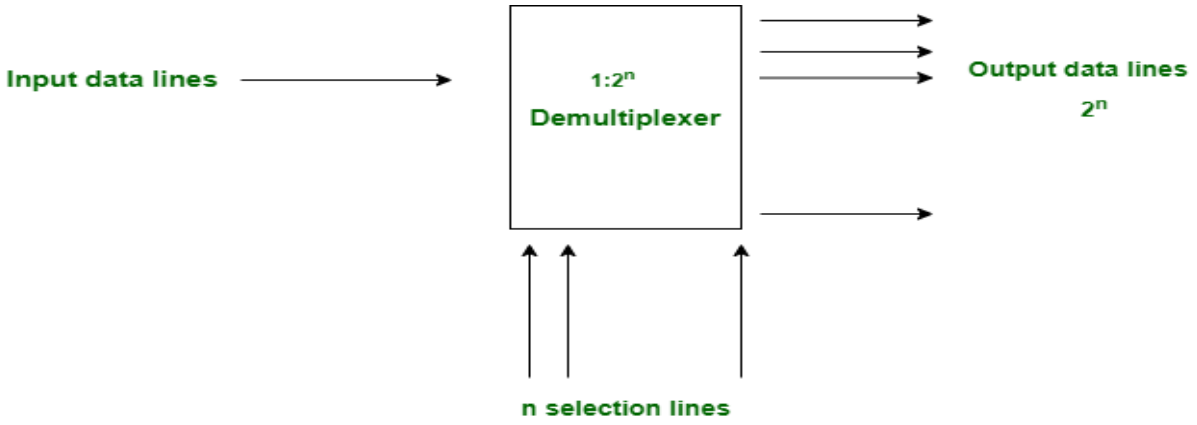


Demultiplexer :

Demultiplexer is a data distributor which takes a single input and gives several outputs. In demultiplexer we have 1 input and 2n output lines where n is the selection line.

It is used to send a signal to one of the many devices. The main difference between a multiplexer and a de-multiplexer is that a multiplexer takes two or more signals and encodes them on a wire, whereas a de-multiplexer does reverse to what the multiplexer does.

Given below is the block diagram of the Demultiplexer, It will have one Input line and will give 2n output lines.



2. Study of half Adder and half subtractor.

Half Adder :

The Half-Adder is a basic building block of adding two numbers as two inputs and produce out two outputs. The adder is used to perform OR operation of two single bit binary numbers. The **augent** and **addent** bits are two input states, and **'carry**' and **'sum** 'are two output states of the half adder.

Block diagram

Truth Table

Inputs		Outputs	
A	B	Sum	Carry
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

In the above table,

1. 'A' and' B' are the input states, and 'sum' and 'carry' are the output states.
2. The carry output is 0 in case where both the inputs are not 1.
3. The least significant bit of the sum is defined by the 'sum' bit.

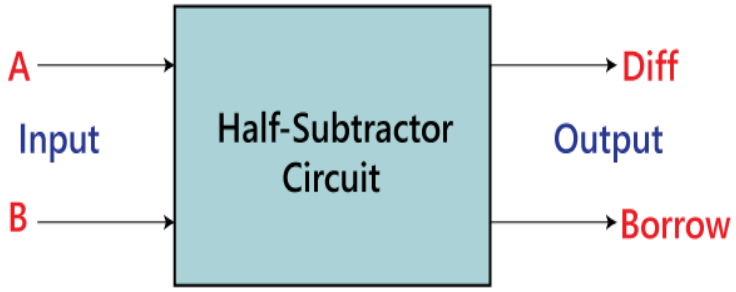
The SOP form of the sum and carry are as follows:

Sum=x'y+xy'
Carry = xy

Half Subtractor :

The half subtractor is also a building block for subtracting two binary numbers. It has two inputs and two outputs. This circuit is used to subtract two single bit binary numbers A and B. The **'diff**' and **'borrow'** are two output states of the half subtractor.

Block diagram



Truth Table

Inputs		Outputs	
A	B	Diff	Borrow
0	0	0	0
0	1	1	1
1	0	1	0
1	1	0	0

The SOP form of the **Diff** and **Borrow** is as follows:

Diff=A'B+AB'
Borrow = A'B

In the above table,

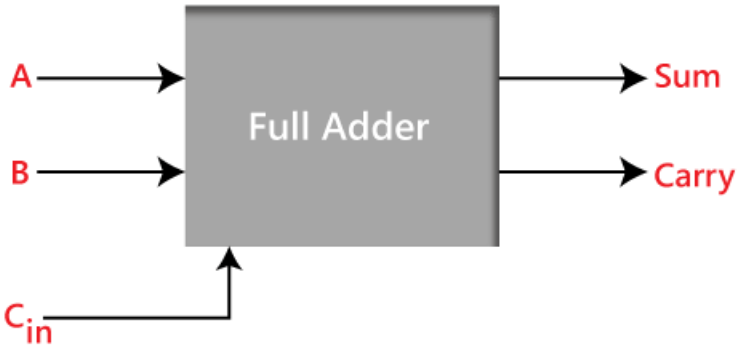
1. 'A' and 'B' are the input variables whose values are going to be subtracted.
2. The 'Diff' and 'Borrow' are the variables whose values define the subtraction result, i.e., difference and borrow.
3. The first two rows and the last row, the difference is 1, but the 'Borrow' variable is 0.
4. The third row is different from the remaining one. When we subtract the bit 1 from the bit 0, the borrow bit is produced.

3.Explain full Adder and Full Subtractor.

Full Adder:

The half adder is used to add only two numbers. To overcome this problem, the full adder was developed. The full adder is used to add three 1-bit binary numbers A, B, and carry C. The full adder has three input states and two output states i.e., sum and carry.

Block diagram



Truth Table

Inputs			Outputs	
A	B	C_{in}	Sum	Carry
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

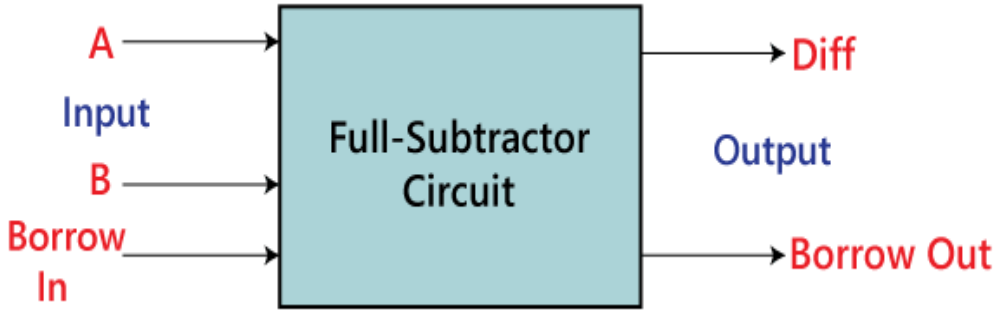
In the above table,

1. 'A' and' B' are the input variables. These variables represent the two significant bits which are going to be added
2. 'C_{in}' is the third input which represents the carry. From the previous lower significant position, the carry bit is fetched.
3. The 'Sum' and 'Carry' are the output variables that define the output values.
4. The eight rows under the input variable designate all possible combinations of 0 and 1 that can occur in these variables.

Full Subtractor :

The Half Subtractor is used to subtract only two numbers. To overcome this problem, a full subtractor was designed. The full subtractor is used to subtract three 1-bit numbers A, B, and C, which are minuend, subtrahend, and borrow, respectively. The full subtractor has three input states and two output states i.e., diff and borrow.

Block diagram



Truth Table

Inputs			Outputs	
A	B	$Borrow_{in}$	Diff	Borrow
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

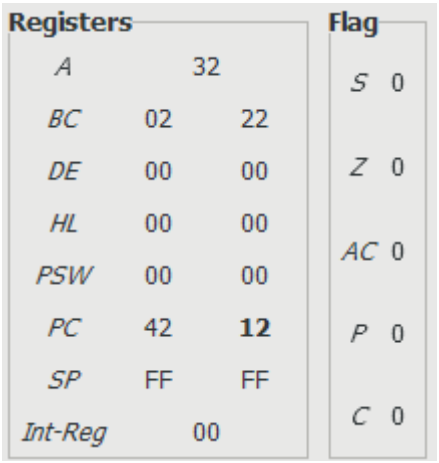
In the above table :

1. 'A' and' B' are the input variables. These variables represent the two significant bits that are going to be subtracted.
2. '$Borrow_{in}$' is the third input which represents borrow.
3. The 'Diff' and 'Borrow' are the output variables that define the output values.
4. The eight rows under the input variable designate all possible combinations of 0 and 1 that can occur in these variables.

4.WAP to add 8 bit number and store result in location 2000.

```
start: nop
mvi A,10H
sta 1998
;mvi B,12H
;sta 1999
mvi C,34
sta 1999
Add C
sta 2000
hlt
```

Output :



Registers		
A	32	
BC	02	22
DE	00	00
HL	00	00
PSW	00	00
PC	42	**12**
SP	FF	FF
Int-Reg	00	

Flag	
S	0
Z	0
AC	0
P	0
C	0

Address (Hex)	Address	Data
07CE	1998	16
07CF	1999	16
07D0	2000	50
07D1	2001	10
07D2	2002	0

5. WAP to multiply two 8 bit numbers stored at memory location 2000 and 2001 and stores the result at memory location 2000 and 2001

start: nop

mvi A,5H

sta 2000

mvi B,2H

sta 2001

mov C,B

label: ADD A

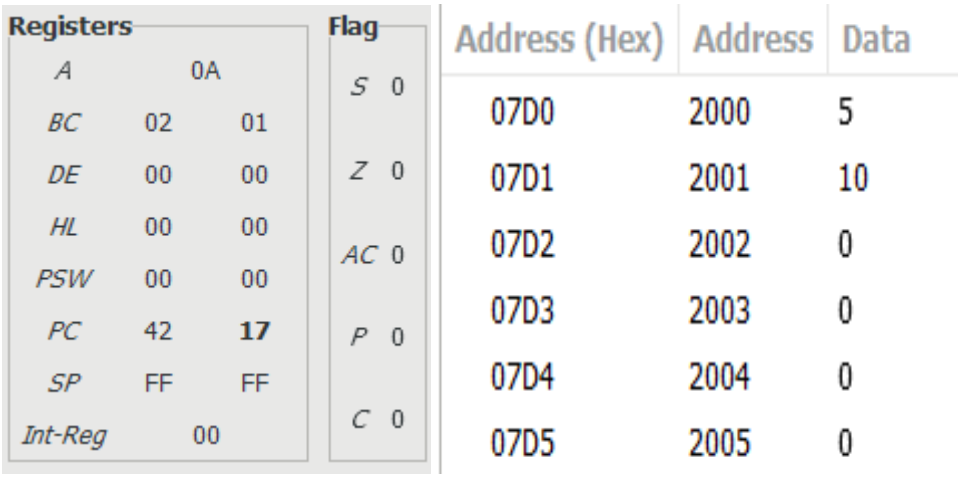
dcr C

jz label

sta 2001

hlt

Output :



Registers			Flag	
A	0A		S	0
BC	02	01	Z	0
DE	00	00	AC	0
HL	00	00	P	0
PSW	00	00	C	0
PC	42	**17**		
SP	FF	FF		
Int-Reg	00			

Address (Hex)	Address	Data
07D0	2000	5
07D1	2001	10
07D2	2002	0
07D3	2003	0
07D4	2004	0
07D5	2005	0

6. Implement a c++ program to conversion of number system.

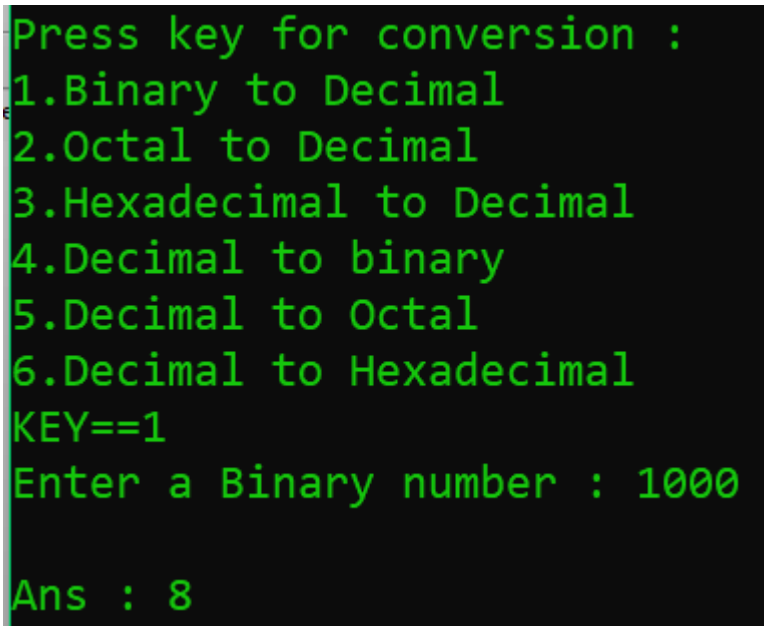
```
#include<iostream>
#include<math.h>
using namespace std;
class conversion{
        public:
        int num,i=0,j,b=0,c,sum=0;

        void bin_to_dec()
        {
                cout<<"Enter a Binary number : ";
                cin>>num;
                        while(num>0)
                {
                        int s=num%10;
                        if(s==1)
                        {
                    c=s*pow(2,i);
                   }
                   else if(s>1)
                     {
                        cout<<"Not a binary number";
                        break;
                          }
                   b=b+c;
                   i=i+1;
                   num=num/10;
                }
                cout<<endl<<b;
        }
        void oct_to_dec()
        {
                cout<<"Enter a Octal number : ";
```

```
                cin>>num;
                        while(num>0)
                {
                        int s=num%10;
                    c=s*pow(8,i);
                   b=b+c;
                   i=i+1;
                   num=num/10;
                }
                cout<<endl<<b;
        }

 void hex_to_dec()
{

 char hexval[10];
  cout<<"Enter a hexadecimal number : ";
  cin>>hexval;
  int  p=0;

 int dec_val = 0;

for ( i = 1; hexval[i]!='\0'; i++) {
        p=p+1;
}
 for ( i = 0;hexval[i]!='\0'; i++) {

    if (hexval[i] >= '0' && hexval[i] <= '9') {
       dec_val += (int(hexval[i]) - 48) * pow(16,p);
        p=p-1;
    } else{ if (hexval[i] >= 'A' && hexval[i] <= 'F') {
       dec_val += (int(hexval[i]) - 55) * pow(16,p);
       p=p-1;
    }
```

```
        }
    }
    cout<<dec_val;
}

void dec_to_bin(){
        cout<<"Enter a decimal number : ";
        cin>>num;
        while(num>0)
        {
                c=num%2;
                sum=pow(10,i)*c+sum;
                num=num/2;
                i++;
        }
        cout<<sum;
}
void dec_to_oct(){
cout<<"Enter a decimal number : ";
        cin>>num;
        while(num>0)
        {
                c=num%8;
                sum=pow(10,i)*c+sum;
                num=num/8;
                i++;
        }
        cout<<sum;
}
void dec_to_hexa()
{
        cout<<"Enter a decimal number : ";
        cin>>num;
        while(num>0)
```

```
        {
                c=num%16;
                sum=pow(10,i)*c+sum;
                num=num/16;
                i++;
        }
        cout<<sum;
};
int main(){
        conversion ob;
        int key;
        cout<<"Press key for conversion :\n";
        cout<<"1.Binary to Decimal \n2.Octal to Decimal \n3.Hexadecimal to Decimal\n";
cout<<"4.Decimal to binary\n5.Decimal to Octal\n6.Decimal to Hexadecimal\nKEY==";
        cin>>key;
        if(key==1)
            ob.bin_to_dec();
   else if(key==2)
           ob.oct_to_dec();
   else if(key==3)
        ob.hex_to_dec();
        else if(key==4)
            ob.dec_to_bin();
        else if(key==5)
            ob.dec_to_oct();
        else if(key==6)
            ob.dec_to_hexa();
        else
        cout<<"wrong key";
}
```

Output :


Press key for conversion :
1.Binary to Decimal
2.Octal to Decimal
3.Hexadecimal to Decimal
4.Decimal to binary
5.Decimal to Octal
6.Decimal to Hexadecimal
KEY==1
Enter a Binary number : 1000
Ans : 8

8.Assume that 3 bytes of data are stored at consecutive memory addresses of the data memory starting at 2000. Write a program which loads register C with (2000), i.e. with data contained at memory address2000, D with (2001), E with (2002) and A with (2001)

```
start: nop
lda 2000
mov C,A
lda 2001
mov D,A
lda 2002
mov E,A
lda 2003

hlt
```

Output -

Registers			Flag	
A	0D		S	0
BC	00	0F	Z	0
DE	0B	0E	AC	0
HL	00	00	P	0
PSW	00	00	C	0
PC	42	**14**		
SP	FF	FF		
Int-Reg	00			

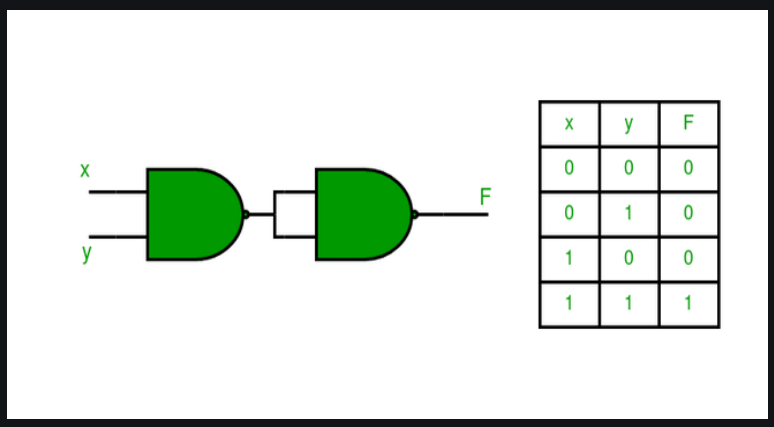
Address (Hex)	Address	Data
07D0	2000	15
07D1	2001	11
07D2	2002	14
07D3	2003	13

9. Implement logic gates using NAND and NOR gate.

1. Implementation of AND Gate using Universal gates.

a) Using NAND Gates

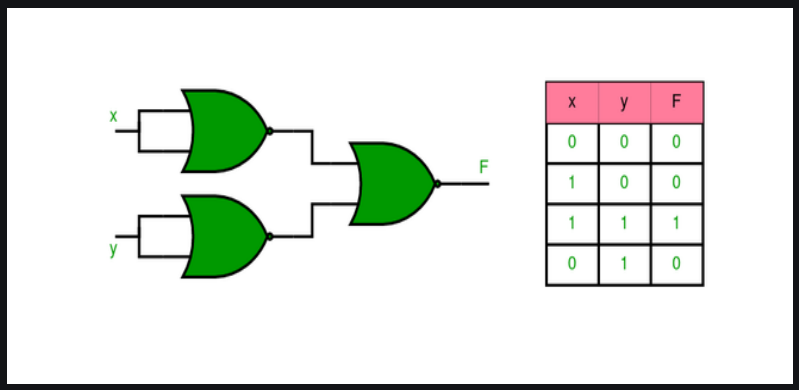
The AND gate can be implemented by using two NAND gates in the below fashion:



x	y	F
0	0	0
0	1	0
1	0	0
1	1	1

b) Using NOR Gates

Implementation of AND gate using only NOR gates as shown below:

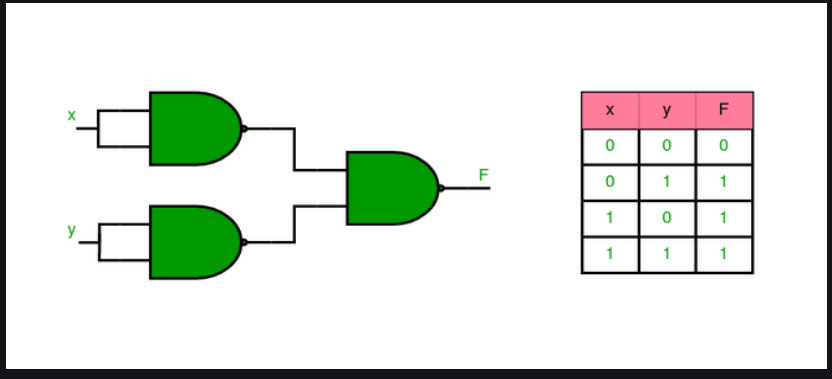


x	y	F
0	0	0
1	0	0
1	1	1
0	1	0

2. Implementation of OR Gate using Universal gates.

a) Using NAND Gates

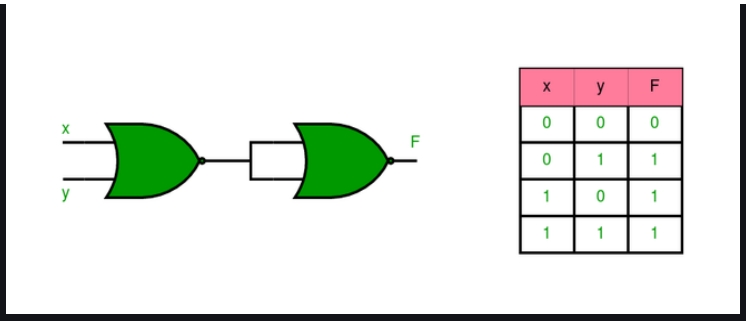
The OR gate can be implemented using the NAND gate as below:



x	y	F
0	0	0
0	1	1
1	0	1
1	1	1

b) Using NOR Gates

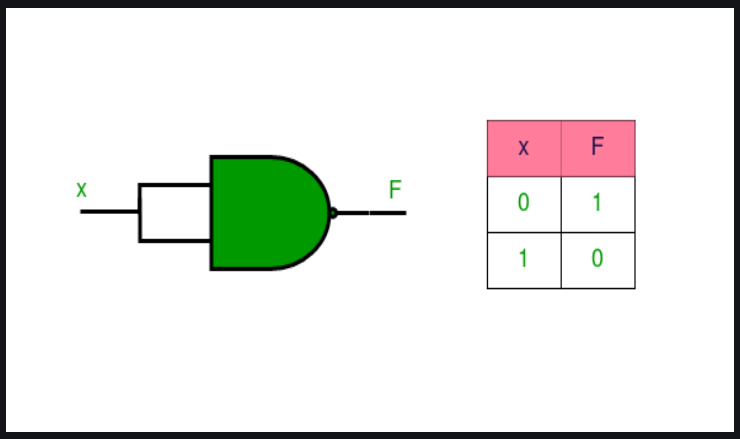
Implementation of OR gate using two NOR gates as shown in the picture below:



x	y	F
0	0	0
0	1	1
1	0	1
1	1	1

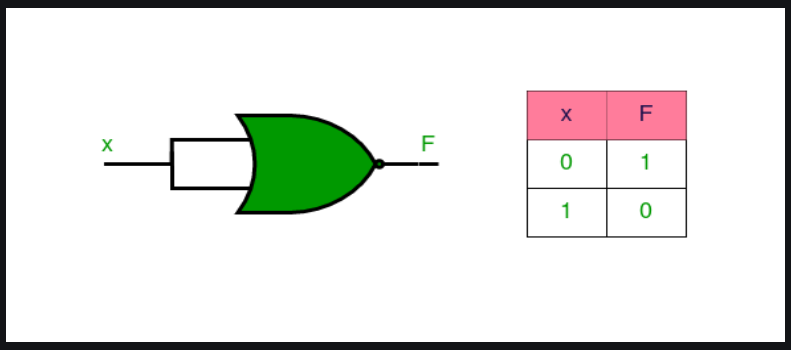
3. Implementation of NOT Gate using Universal gates.

a) Using NAND Gates



x	F
0	1
1	0

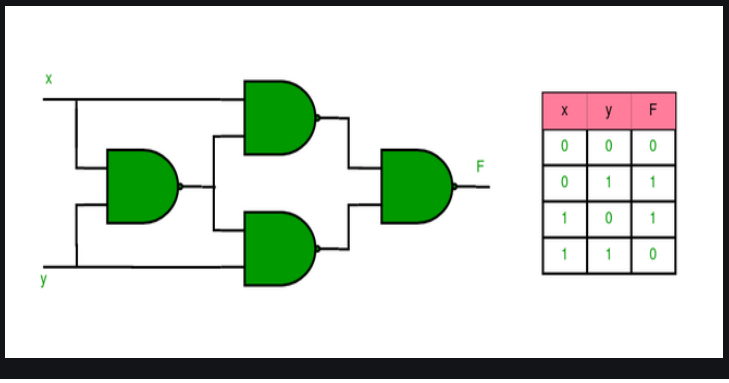
b) Using NOR Gates



x	F
0	1
1	0

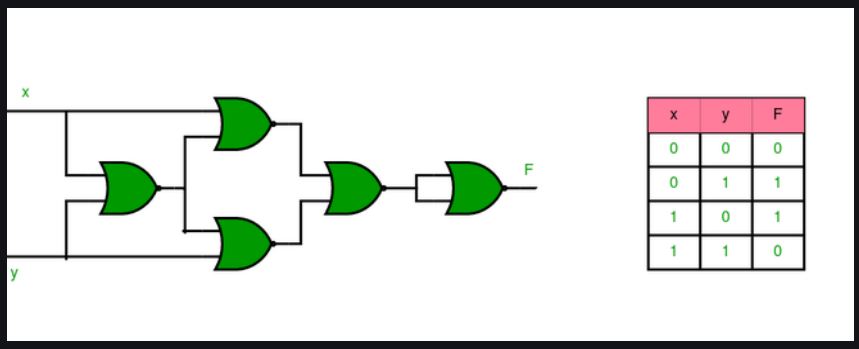
4. Implementation of XOR Gate using Universal gates.

a) Using NAND Gates



x	y	F
0	0	0
0	1	1
1	0	1
1	1	0

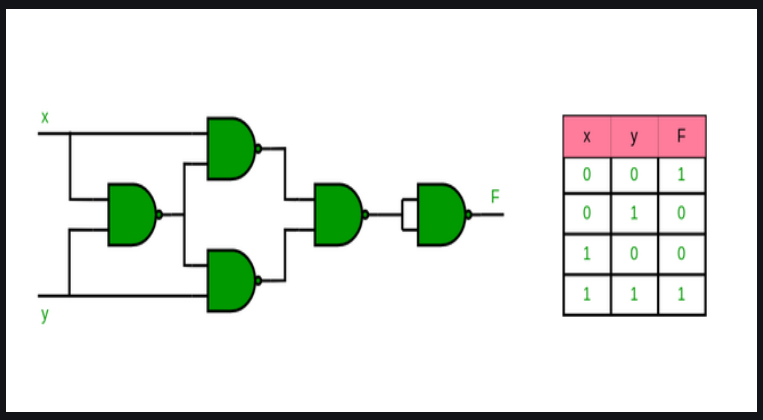
b) Using NOR Gates



x	y	F
0	0	0
0	1	1
1	0	1
1	1	0

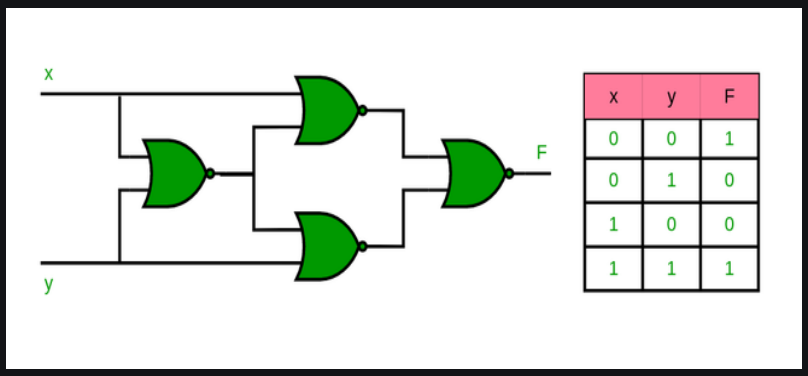
5. Implementation of XNOR Gate using Universal gates.

a) Using NAND Gate



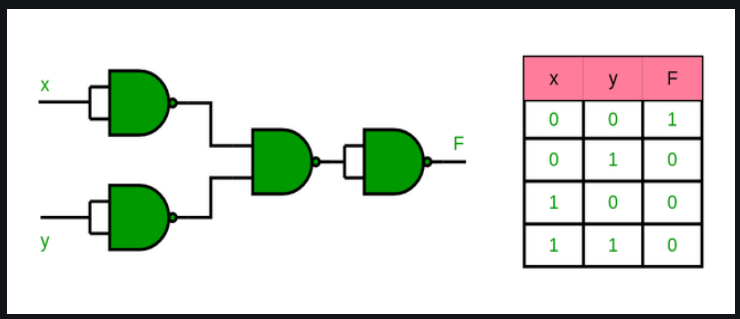
x	y	F
0	0	1
0	1	0
1	0	0
1	1	1

b) Using NOR Gate



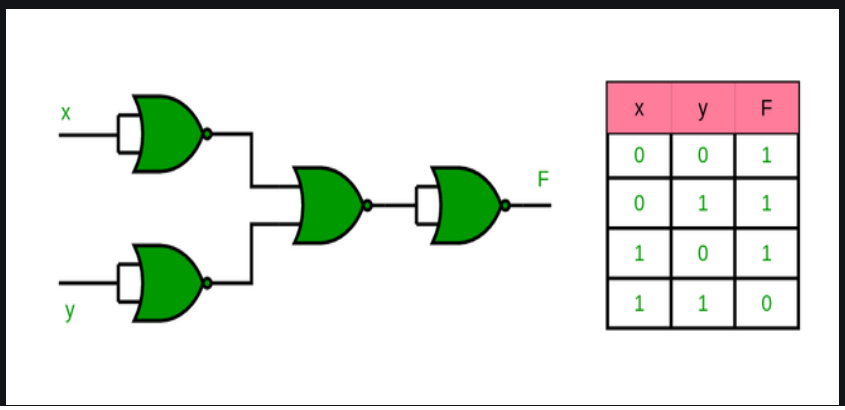
x	y	F
0	0	1
0	1	0
1	0	0
1	1	1

6. Implementation of NOR Gate using NAND Gates



x	y	F
0	0	1
0	1	0
1	0	0
1	1	0

7. Implementation of NAND Gate using NOR Gates



x	y	F
0	0	1
0	1	1
1	0	1
1	1	0

10.Design a full adder using Logic gates.