LOKMANYA TILAK COLLEGE OF ENGINEERING
Sector No. 4., Vikas Nagar, Koparkhairane, Navi Mumbai -400 709.

Computer Science and Engineering

(Artificial Intelligence & Machine Learning)

AY 2021-22 (Odd)

(SEM. III)

Practical file

for

Digital Logic &

Computer Architecture Lab (DLCA)

(CSL-302)

Name of Student: **PRATHAMESH CHIKANKAR**

Roll no. : **AIMLD08**

Faculty Incharge: Prof. Gauri Ashtankar

NAME - PRATHAMESH CHIKANKAR
ROLL NO. - AIMLD08
BRANCH & BATCH - CSE-(AI&ML) & A1
DATE OF SUBMISSION - 26/11/2021
SUBJECT - DLCOA LAB

INDEX VIEW

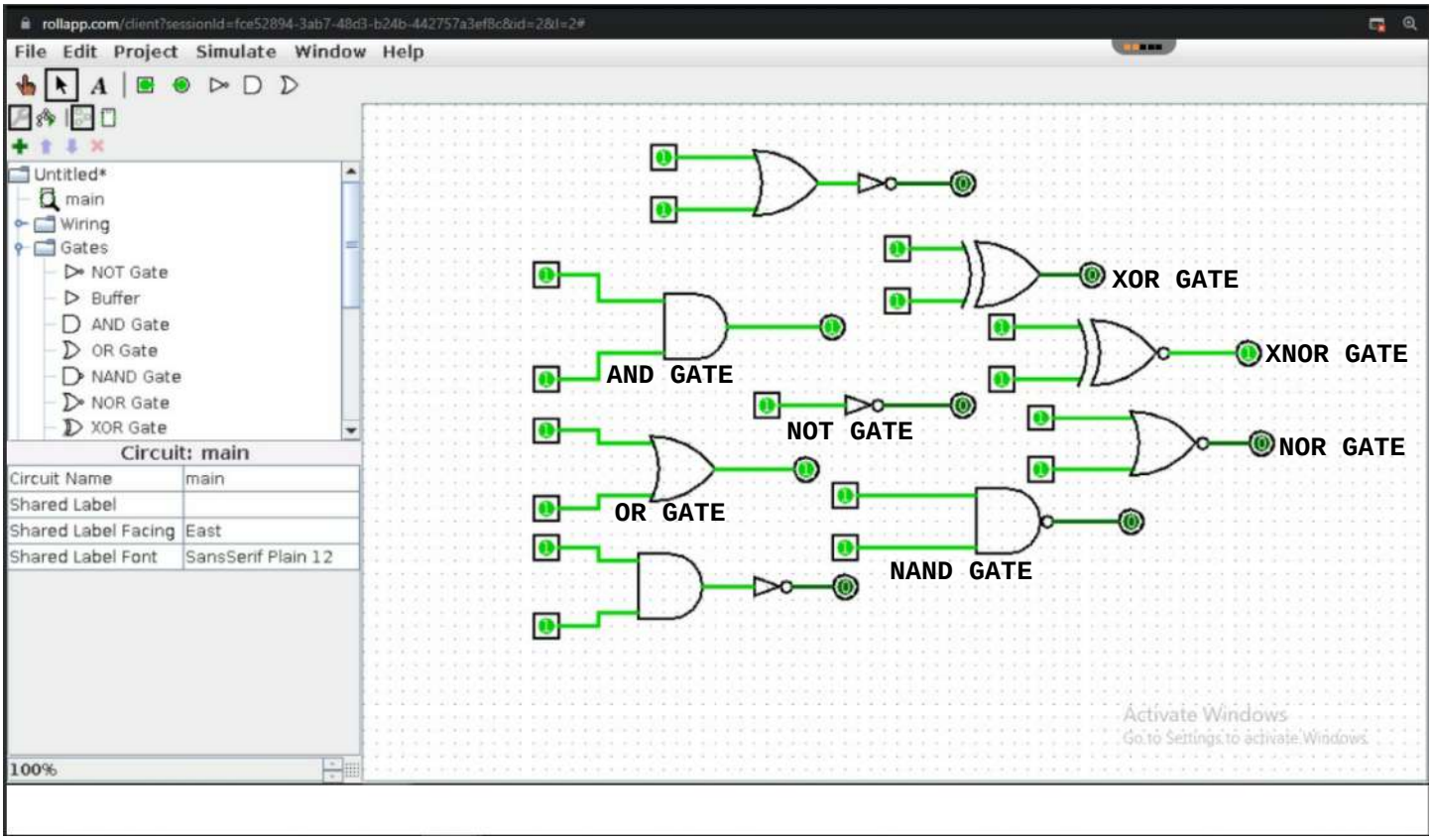
Experiment No. - 01

Roll No. - AIMLD08 **Date Of Performance - 08/09/2021**

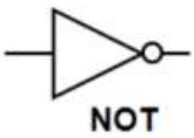
Aim - To verify the truth table of various logic gates.

Software Used - Logism

Screen Shots in PDF format -

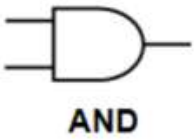


Result (in terms of truth table) -



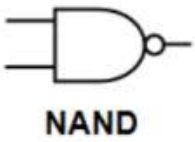
NOT

Input	Output
I	F
0	1
1	0



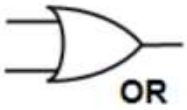
AND

Inputs		Output
A	B	F
0	0	0
1	0	0
0	1	0
1	1	1



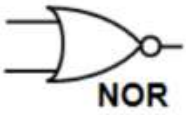
NAND

Inputs		Output
A	B	F
0	0	1
1	0	1
0	1	1
1	1	0



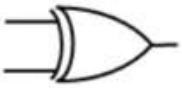
OR

Inputs		Output
A	B	F
0	0	0
1	0	1
0	1	1
1	1	1



NOR

Inputs		Output
A	B	F
0	0	1
1	0	0
0	1	0
1	1	0



EXCLUSIVE OR

Inputs		Output
A	B	F
0	0	0
0	1	1
1	0	1
1	1	0

EXCLUSIVE NOR

Inputs		Output
A	B	F
0	0	1
0	1	0
1	0	0
1	1	1

Conclusion - Logic gates are verified using truth tables.

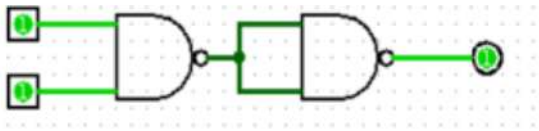
Experiment No. - 02

Roll No. - AIMLD08 **Date Of Performance - 15/09/2021**

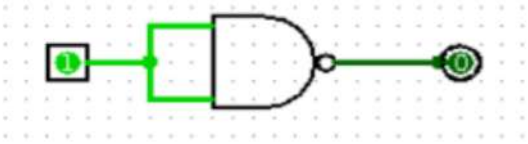
Aim - To verify NAND gate as Universal gate and design a NAND logic circuit that is equivalent to the AOI circuit shown below.

Software Used - Logism

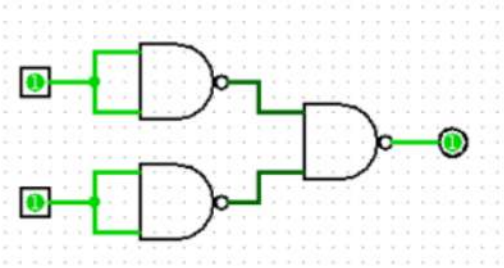
Screen Shots in PDF format -



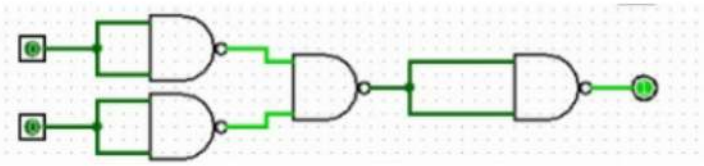
AND gate using NAND gates



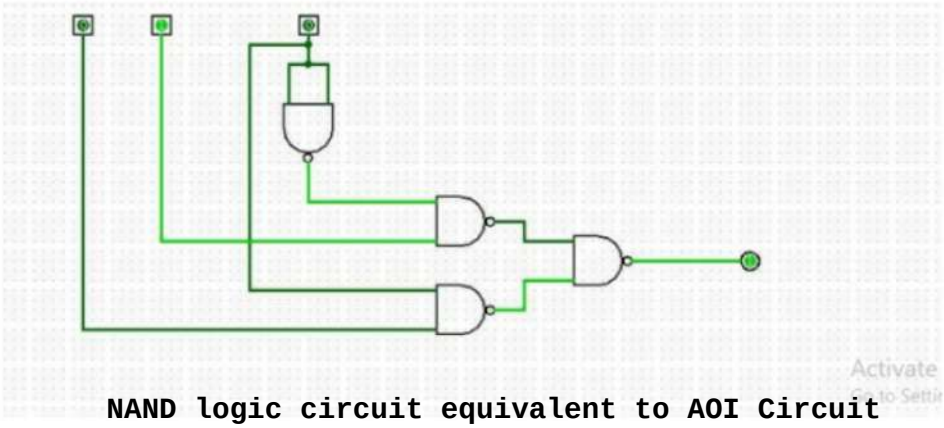
NOT gate using NAND gate



OR gate using NAND gate



NOR gate using NAND gate



NAND logic circuit equivalent to AOI Circuit

Result (in terms of truth table) -

AND using NAND: -

AND gate using NAND gate		
A	B	OUTPUT
0	0	0
0	1	0
1	0	0
1	1	1

OR using NAND: -

OR gate using NAND gate		
A	B	OUTPUT
0	0	0
0	1	1
1	0	1
1	1	1

NOR using NAND: -

NOR gate using NAND gate		
A	B	OUTPUT
0	0	1
0	1	0
1	0	0
1	1	0

NOT using NAND: -

NOT gate using NAND gate	
A	OUTPUT
0	1
1	0

AQI EQUIVALENT USING NAND GATE ONLY			
INPUT			OUTPUT
A	B	C	Z
1	1	1	0
1	1	0	1
1	0	1	1
1	0	0	1
0	1	1	1
0	1	0	1
0	0	1	1
0	0	0	1

Conclusion - All the basic gates using NAND GATE and the AQI equivalent using only NAND GATE.

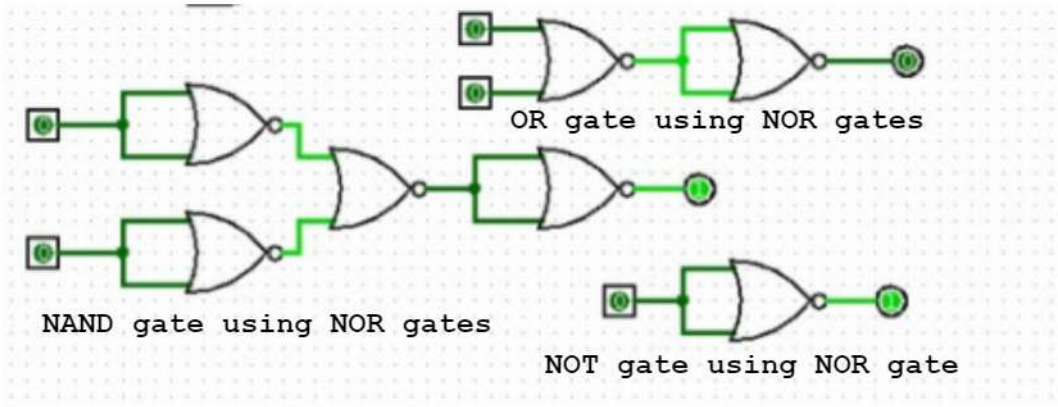
Experiment No. - 03

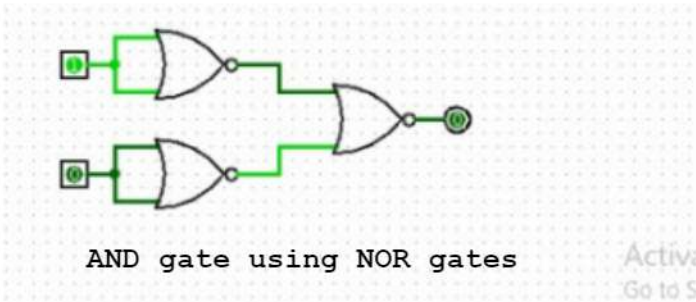
Roll No. - AIMLD08 **Date Of Performance - 22/09/2021**

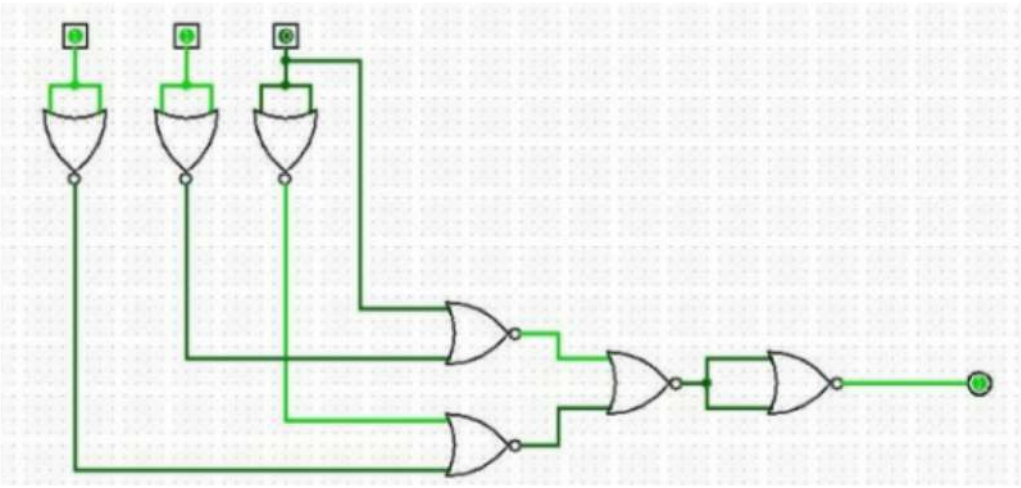
Aim - To verify NOR gate as Universal gate and design a NOR logic circuit that is equivalent to the AOI circuit shown.

Software Used - Logism

Screen Shots in PDF format -







NOR Logic circuit equivalent to AOI circuit.

Result (in terms of truth table) -

NAND gate using NOR gate		
A	B	OUTPUT
0	0	1
0	1	1
1	0	1
1	1	0

AND gate using NOR gate		
A	B	OUTPUT
0	0	0
0	1	0
1	0	0
1	1	1

OR gate using NOR gate		
A	B	OUTPUT
0	0	0
0	1	1
1	0	1
1	1	1

NOT gate using NOR gate	
A	OUTPUT
0	1
1	0

AQI EQUIVALENT USING NOR GATE ONLY			
INPUT			OUTPUT
A	B	C	Z
1	1	1	0
1	1	0	0
1	0	1	0
1	0	0	0
0	1	1	0
0	1	0	0
0	0	1	0
0	0	0	1

Conclusion - Learn to make all the basic gates using NOR GATE and the AQI equivalent using only NOR GATE.

Experiment No. - 04

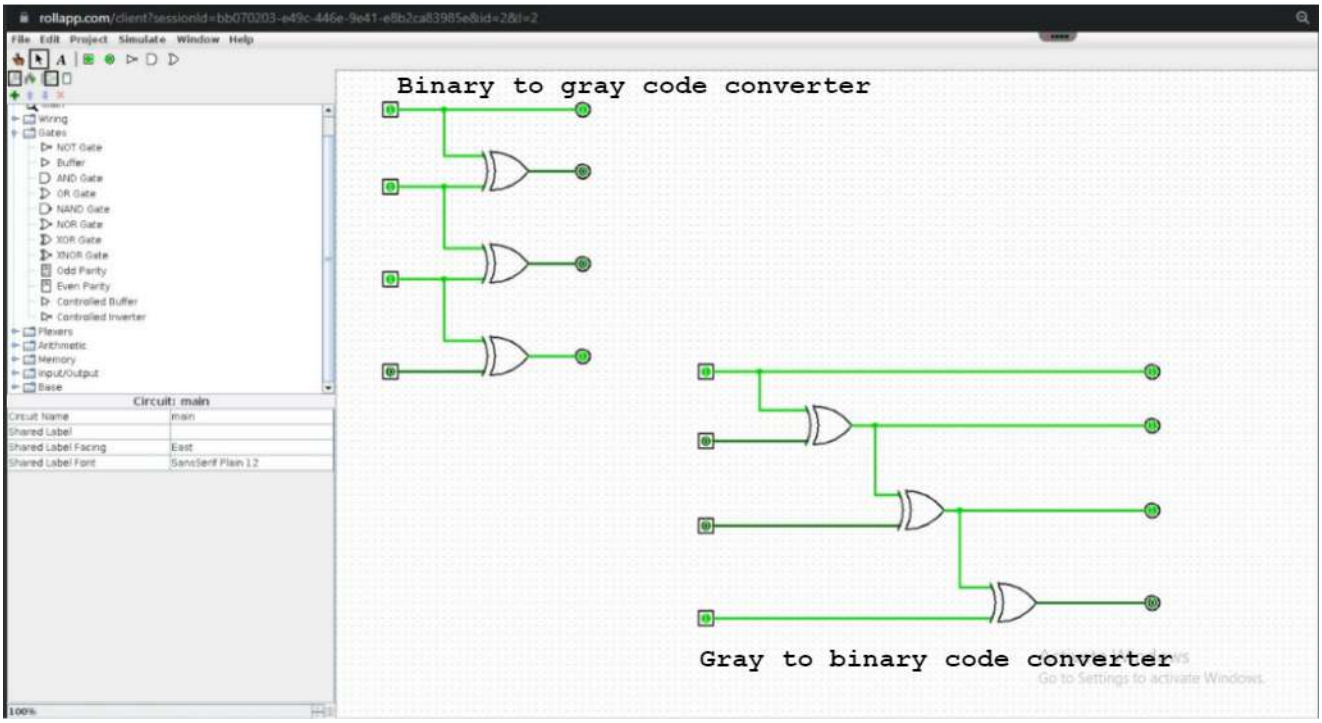
Roll No. - AIMLD08

Date Of Performance - 29/09/2021

Aim - To Study of Binary to Gray code converter and Gray to Binary.

Software Used - Logism

Screen Shots in PDF format -



Result (in terms of truth table) -

GRAY CODE				BINARY CODE			
A/G1	B/G2	C/G3	D/G4	B1/A	B2/B	B3/C	B4/D
0	0	0	0	0	0	0	0
0	0	0	1	0	0	0	1
0	0	1	0	0	0	1	1
0	0	1	1	0	0	1	0
0	1	0	0	0	1	1	1
0	1	0	1	0	1	1	0
0	1	1	0	0	1	0	0
0	1	1	1	0	1	0	1
1	0	0	0	1	1	1	1
1	0	0	1	1	1	1	0
1	0	1	0	1	1	0	0
1	0	1	1	1	1	0	1
1	1	0	0	1	0	0	0
1	1	0	1	1	0	0	1
1	1	1	0	1	0	1	1
1	1	1	1	1	0	1	0

Conclusion - We have verified the Binary code to Gray code and Gray code to Binary code conversion using truth table.

Experiment No. - 05

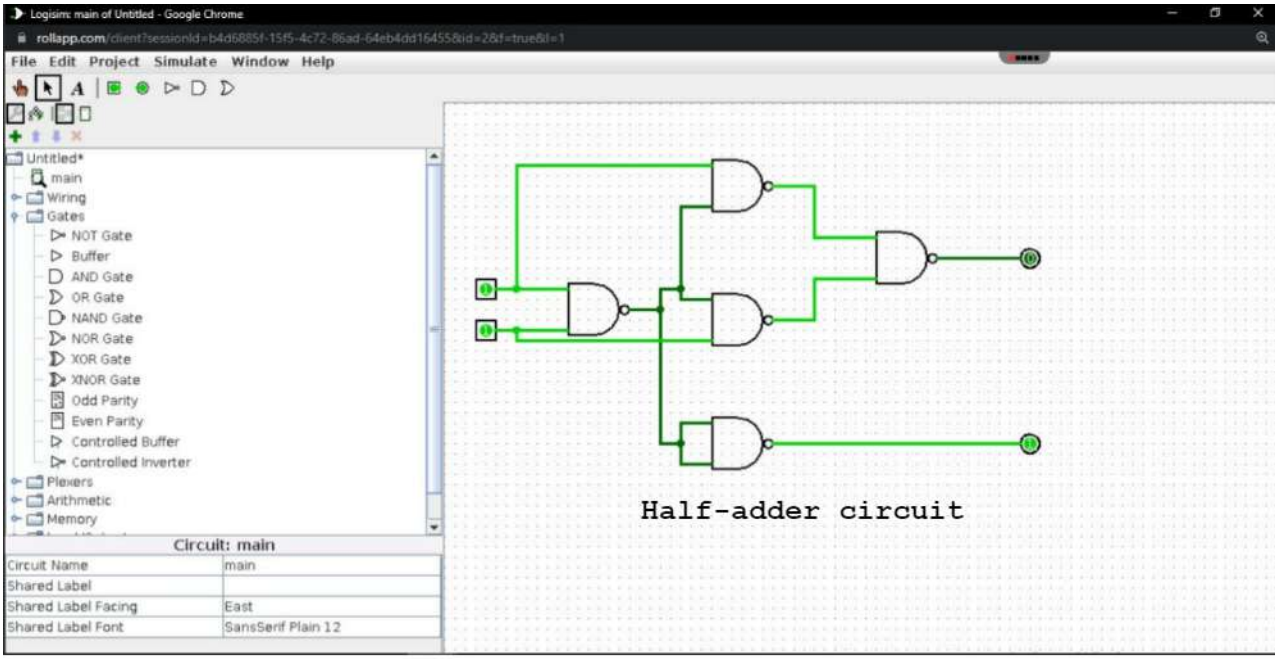
Roll No. - AIMLD08

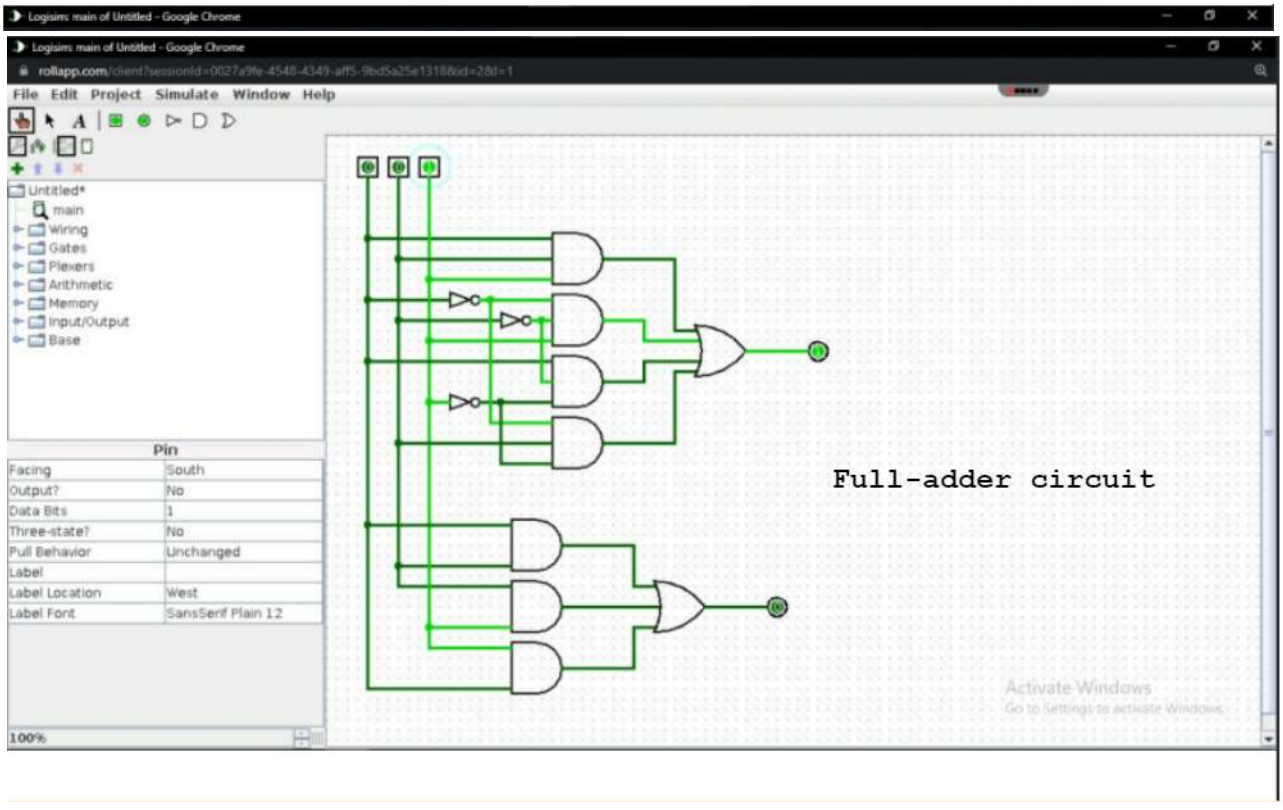
Date Of Performance - 12/10/2021 & 25/10/2021

Aim - To Study of Half and Full adder.

Software Used - Logism

Screen Shots in PDF format -





Result (in terms of truth table) -

HALF-ADDER			
INPUT		OUTPUT	
A	B	SUM	CARRY
1	1	0	1
1	0	1	0
0	1	1	0
0	0	0	0

FULL-ADDER				
INPUT			OUTPUT	
A	B	Cin	SUM	CARRY
1	1	1	1	1
1	1	0	0	1
1	0	1	0	1
1	0	0	1	0
0	1	1	0	1
0	1	0	1	0
0	0	1	1	0
0	0	0	0	0

Conclusion - Verified the Binary code to Gray code and Gray code to Binary code conversion using truth table

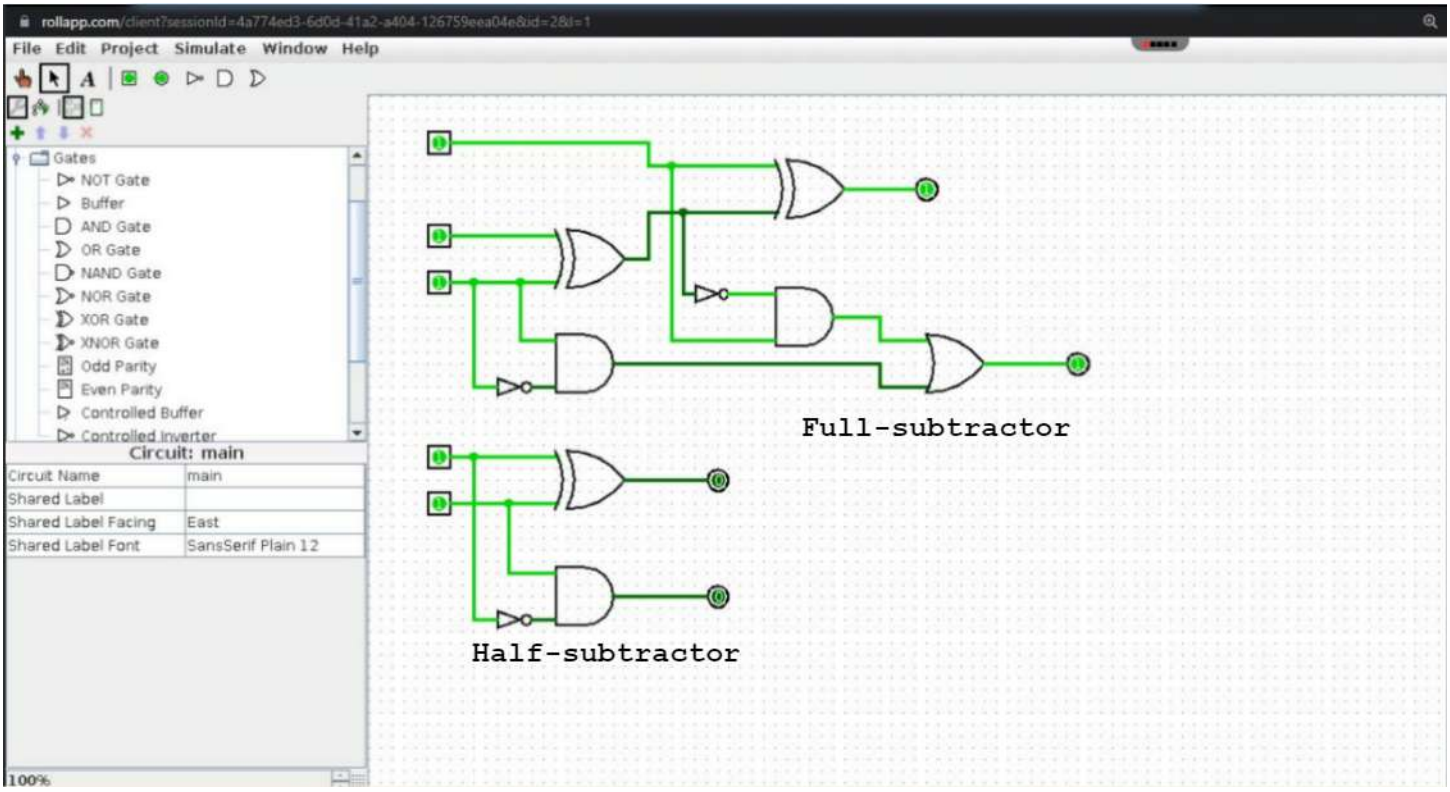
Experiment No. - 06

Roll No. - AIMLD08 **Date Of Performance - 15/11/2021**

Aim - To Study of Half and Full subtractor.

Software Used - Logism

Screen Shots in PDF format -



Result (in terms of truth table) -

HALF SUBTRACTOR			
INPUTS		OUTPUTS	
MINUEND(A)	SUBTRAHEND(B)	DIFFERENCE(D)	BORROW (Bo)
0	0	0	0
0	1	1	1
1	0	1	0
1	1	0	0

FULL SUBTRACTOR				
INPUTS			OUTPUTS	
MIINUEND	SUBTRAHEND	BORROW	DIFFERENCE	BORROW
A	B	B in	D	Bo
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
B in = borrow input				

Conclusion - Successfully designed the Half Subtractor and Full Subtractor by verifying the truth tables for both the circuits.

Experiment No. - 07

Roll No. - AIMLD08 **Date Of Performance - 22/11/2021**

Aim - To Implement Booth Algorithm for Binary multiplication.

Software Used - VS-CODE

Program -

Booth Algorithm - Notepad

File Edit Format View Help

```
#include<stdio.h>
#include<math.h>
#define BIT 4
void d_b(int dec_s, int bin[BIT])
{
    //function to convert decimal(int type) to binary(array type)
    int carry,dec,i;
    carry = 0;
     dec = dec_s>0?dec_s:-dec_s; //checking if decimal is +ve or -ve
     for(i=0; i<BIT;i++)
     {
        bin[i] = (dec%2); //store the remainder
        dec/=2; //reduce the number by a factor of 2
    }
    if(dec_s>=0)
    return;
    else
    {
        for(int j=0; j<BIT;j++)
        {
            bin[j] = bin[j]==0?1:0; //complementing the bits if decimal is found to be less than zero
        }
        if(bin[0]+1<=1)
        {
            bin[0]+=1;
            return;
        }
         else
         {
            carry =1; //if bit additon exceeds 1 then carry is generated   for (int i=0;i<BIT;i++)
        }
        if(bin[i]+carry<=1)
        {
            bin[i]+=carry;
            carry =0;
            return;
        }
        else
        {
            bin[i] = (bin[i]+carry)%2;
        }
    }
}
int b_d(int a[BIT],int q[BIT])
{
    //binary to decimal conversion
    int i=0;
    int ans=0;
    for(i;i<BIT;i++)
    {
        ans+=q[i]*pow(2,i);
    }
    for(int k=0;k<BIT;k++,i++)
    {
        ans+=a[k]*pow(2,i);
    }
    return ans;
}
void twos (int m[BIT], int m2[BIT])
{
    //calculating 2's complement
    for(int i=0; i<BIT;i++)
    {
        m2[i]=m[i]==0?1:0; //calculating 1's complement
    }
    int carry =1; //adding 1 to the above result
    for (int i=0;i<BIT;i++)
    {
        if(m2[i]+carry<=1)
        {
            m2[i]+=carry;
            carry =0;
            return;
        }
        else
        {
            m2[i] = (m2[i]+carry)%2; //if result exceeds 1 then take the remainder
        }
    }
}
```

```
int add (int a[BIT], int m[BIT])
{
    int carry =0;
    for (int i=0;i<BIT;i++)
    {
        carry += m[i];
        if(a[i]+carry<=1)
        {
            a[i]+=carry;
            carry =0;
        }
        else
        {
            a[i] = (a[i]+carry)%2;
            carry = 1;
        }
    }
    return carry;
}
int ashr(int a[BIT],int q[BIT],int q_1)
{
    //performing ASR b/w A Q and Q-1
    q_1 = q[0];
    for(int i=1; i<BIT;i++ )
    {
        q[i-1]=q[i];
    }
    q[BIT-1]=a[0];
    for(int i=1; i<BIT; i++)
    {
        a[i-1]=a[i];
    }
    return q_1;
}
int main() // main function to set the flow of the program
{
    int multiplicand, multiplier, q_1, count;
    printf(" Enter the multiplicand: ");
    scanf("%d", &multiplicand);
    printf(" Enter the multiplier: ");
    scanf("%d", &multiplier);
    q_1 =0;
    int a[BIT] = {0};
    int m[BIT]={0};
    int m2[BIT]={0};
    int q[BIT]={0};
    count = BIT;
    d_b(multiplicand,m); //converting user input into the binary equivalent
    d_b(multiplier,q);
    twos(m,m2);
    printf("\n Entered Multiplicand: ");
    for(int k=BIT-1; k>=0;k--)
    {
        printf(" %d",m[k]);
    }
    printf("\n Entered Multiplier: ");
    for(int k=BIT-1; k>=0;k--)
    {
        printf(" %d",q[k]);
    }
    printf("\n\n\t\tA\t Q\t\tQ-1\tOPERATION");
    int a1=1;
    for(int i=count;i!=0;i--)
    {
        printf("\n LOOP NO:%d ",a1);
        a1++;
        for(int k=BIT-1; k>=0;k--)
        {
            printf(" %d",a[k]);
        }
        printf("\t");
        for(int k=BIT-1; k>=0;k--)
        {
            printf(" %d",q[k]);
        }
        printf("\t %d", q_1);
        if(q[0]==1&&q_1==0)
```

```
        {
            add(a,m2);
            printf("\n ");
            for(int k=BIT-1; k>=0;k--)
            {
                printf(" %d",a[k]);
            }
            printf("\t");
            for(int k=BIT-1; k>=0;k--)
            {
                printf(" %d",q[k]);
            }
            printf("\t %d\t ADD M", q_1);
        }
        else if(q[0]==0&&q_1==1)
        {
            add(a,m);
            printf("\n ");
            for(int k=BIT-1; k>=0;k--)
            {
                printf(" %d",a[k]);
            }
            printf("\t");
            for(int k=BIT-1; k>=0;k--)
            {
                printf(" %d",q[k]);
            }
            printf("\t %d\t SUB M", q_1);
        }
        q_1 = ashr(a,q,q_1);
        printf("\n ");
        for(int k=BIT-1; k>=0;k--)
        {
            printf(" %d",a[k]);
        }
        printf("\t");
        for(int k=BIT-1; k>=0;k--)
        {
            printf(" %d",q[k]);
        }
        printf("\t %d\t ASHR", q_1);
    }
    printf("\n\nFINAL RESULT: \n");
    printf("\n The answer in binary form is:");
    for(int k=BIT-1; k>=0;k--)
    {
        printf(" %d",a[k]);
    }
    for(int k=BIT-1; k>=0;k--)
    {
        printf(" %d",q[k]);
    }
    int sign = a[BIT-1];
    if(sign)
    {
        for(int k=BIT-1; k>=0;k--)
        {
            a[k] = a[k]==1?0:1;
            q[k] = q[k]==1?0:1;
        }
        int one[BIT] = {0};
        one[0]=1;
        int carry=0;
        carry = add(q,one);
        if(carry)
        carry = add(a,one);
    }
    int ans = b_d(a,q);
    printf("\n The answer in decimal form is: ");
    if(sign)
    printf("-");
    printf("%d \n",ans);
}
```

Result - Output

```
PS C:\Users\Admin\AppData\Local\Temp> cd "C:\Users\Admin\AppData\Local\Temp\" ; if ($?) { gcc tempCodeRunnerFile.c -o tempCodeRunnerFile } ;
if ($?) { .\tempCodeRunnerFile }
 Enter the multiplicand: 2
 Enter the multiplier: 4

 Entered Multiplicand:  0 0 1 0
 Entered Multiplier:  0 1 0 0

              A       Q           Q-1     OPERATION
 LOOP NO:1  0 0 0 0     0 1 0 0       0
  0 0 0 0       0 0 1 0       0      ASHR
 LOOP NO:2  0 0 0 0     0 0 1 0       0
  0 0 0 0       0 0 0 1       0      ASHR
 LOOP NO:3  0 0 0 0     0 0 0 1       0
  1 1 1 0       0 0 0 1       0      ADD M
  1 1 1 1       0 0 0 0       1      ASHR
 LOOP NO:4  1 1 1 1     0 0 0 0       1
  0 0 0 1       0 0 0 0       1      SUB M
  0 0 0 0       1 0 0 0       0      ASHR

FINAL RESULT:

 The answer in binary form is: 0 0 0 0 1 0 0 0
 The answer in decimal form is: 8
PS C:\Users\Admin\AppData\Local\Temp>
```

Conclusion - Implemented Booth Algorithm for Binary multiplication Successfully.

Experiment No. - 08

Roll No. - AIMLD08 **Date Of Performance - 22/11/2021**

Aim - To Implement Restoring division Algorithm for Binary numbers.

Software Used - VS-CODE

Program -

Restoring division Algorithm - Notepad

File Edit Format View Help

```
#include <stdio.h>
#include <conio.h>
#include <math.h>
int a = 0, b = 0, c = 0, com[5] = {1, 0, 0, 0, 0}, s = 0;
int anum[5] = {0}, anumcp[5] = {0}, bnum[5] = {0};
int acomp[5] = {0}, bcomp[5] = {0}, rem[5] = {0}, quo[5] = {0}, res[5] = {0};
void binary()
{
        a = fabs(a);
        b = fabs(b);
        int r, r2, i, temp;
        for (i = 0; i < 5; i++)
        {
                r = a % 2;
                a = a / 2;
                r2 = b % 2;
                b = b / 2;
                anum[i] = r;
                anumcp[i] = r;
                bnum[i] = r2;
                if (r2 == 0)
                {
                        bcomp[i] = 1;
                }
                if (r == 0)
                {
                        acomp[i] = 1;
                }
        }
        //part for two's complementing
        c = 0;
        for (i = 0; i < 5; i++)
        {
                res[i] = com[i] + bcomp[i] + c;
                if (res[i] >= 2)
                {
                        c = 1;
                }
                else
                c = 0;
                res[i] = res[i] % 2;
        }
        for (i = 4; i >= 0; i--)
        {
                bcomp[i] = res[i];
        }
}
void add(int num[])
{
        int i;
        c = 0;
        for (i = 0; i < 5; i++)
        {
                res[i] = rem[i] + num[i] + c;
                if (res[i] >= 2)
                {
                        c = 1;
                }
                else
                c = 0;
                res[i] = res[i] % 2;
        }
        for (i = 4; i >= 0; i--)
        {
                rem[i] = res[i];
                printf("%d", rem[i]);
        }
        printf(":");
        for (i = 4; i >= 0; i--)
        {
                printf("%d", anumcp[i]);
        }
}
```

```
void shl()
{
        //for shift left
        int i;
        for (i = 4; i > 0; i--)
        {
                //shift the remainder
        rem[i] = rem[i - 1];
        }
        rem[0] = anumcp[4];
        for (i = 4; i > 0; i--)
        {
                //shift the remtient
                anumcp[i] = anumcp[i - 1];
        }
        anumcp[0] = 0;
        printf("\nSHIFT LEFT: ");
        //display together
        for (i = 4; i >= 0; i--)
        {
                printf("%d", rem[i]);
        }
        printf(":");
        for (i = 4; i >= 0; i--)
        {
                printf("%d", anumcp[i]);
        }
}
void main()
{
        int i;
        printf("\t\tRESTORING DIVISION ALGORITHM");
        printf("\nEnter two numbers to multiply: ");
        printf("\nBoth must be less than 16");
        //simulating for two numbers each below 16
        do
        {
                printf("\nEnter A: ");
                scanf("%d", &a);
                printf("Enter B: ");
                scanf("%d", &b);
        }
        while (a >= 16 || b >= 16);
        printf("\nExpected Quotient = %d", a / b);
        printf("\nExpected Remainder = %d", a % b);
        if (a * b < 0)
        {
                s = 1;
        }
        binary();
        printf("\n\nUnsigned Binary Equivalents are: ");
        printf("\nA = ");
        for (i = 4; i >= 0; i--)
        {
                printf("%d", anum[i]);
        }
        printf("\nB = ");
        for (i = 4; i >= 0; i--)
        {
                printf("%d", bnum[i]);
        }
        printf("\nB'+ 1 = ");
        for (i = 4; i >= 0; i--)
        {
                printf("%d", bcomp[i]);
        }
        printf("\n\n-->");
        //division part
        shl();
        for (i = 0; i < 5; i++)
        {
                printf("\n-->"); //start with subtraction
                printf("\nSUB B: ");
                add(bcomp);
                if (rem[4] == 1)
                {
                        //simply add for restoring
                        printf("\n-->RESTORE");
                        printf("\nADD B: ");
                        anumcp[0] = 0;
                        add(bnum);
                }
```

```
            else
            {
                    anumcp[0] = 1;
            }
            if (i < 4)
            shl();
        }
        printf("\n----------------------------");
        printf("\nSign of the result = %d", s);
        printf("\nRemainder is = ");
        for (i = 4; i >= 0; i--)
        {
            printf("%d", rem[i]);
        }
        printf("\nQuotient is = ");
        for (i = 4; i >= 0; i--)
        {
            printf("%d", anumcp[i]);
        }
        getch();
}
```

Result - Output

```
PS C:\Users\Admin\AppData\Local\Temp> cd "C:\Users\Admin\AppData\Local\Temp\" ; if ($?) { gcc tempCodeRunnerFile.c -o tempCodeRunnerFile } ;
if ($?) { .\tempCodeRunnerFile }
                RESTORING DIVISION ALGORITHM
Enter two numbers to multiply:
Both must be less than 16
Enter A: 2
Enter B: 3

Expected Quotient = 0
Expected Remainder = 2

Unsigned Binary Equivalents are:
A = 00010
B = 00011
B'+ 1 = 11101

-->
SHIFT LEFT: 00000:00100
-->
SUB B: 11101:00100
-->RESTORE
ADD B: 00000:00100
SHIFT LEFT: 00000:01000
-->
SUB B: 11101:01000
-->RESTORE
ADD B: 00000:01000
SHIFT LEFT: 00000:10000
-->
SUB B: 11101:10000
-->RESTORE
ADD B: 00000:10000
SHIFT LEFT: 00001:00000
-->
SUB B: 11110:00000
-->RESTORE
ADD B: 00001:00000
SHIFT LEFT: 00010:00000
-->
SUB B: 11111:00000
-->RESTORE
ADD B: 00010:00000
----------------------------
Sign of the result = 0
Remainder is = 00010
Quotient is = 00000
```

Conclusion - Implemented Restoring division Algorithm for Binary numbers successfully.

Experiment No. - 09

Roll No. - AIMLD08 **Date Of Performance - 22/11/2021**

Aim - To Implement Non Restoring division Algorithm for Binary numbers.

Software Used - VS-CODE

Program -

Non Restoring division Algorithm - Notepad

File Edit Format View Help

```
#include<stdio.h>
void main()
{
        int a[10],m[10],q[10],i,j,c=0,b[10],mc[10],z=0,s[10],x[10];
        printf("\nEnter Divisor [M] (BINARY FORM): ");
        for(i=4;i>=1;i--)
        {
                scanf("%d",&m[i]);
                mc[i]=!m[i];
                a[i]=0, x[i]=0;
        }
        x[1]=1, x[5]=0, m[5]=0, mc[5]=1;
        for(i=1;i<=5;i++)
        {
                s[i]=x[i]^mc[i]^z;
                c=(x[i]&&mc[i])||(x[i]&&z)||(mc[i]&&z);
                z=c;
                mc[i]=s[i];
        }
        printf("\nEnter Divident [Q] (BINARY FORM): ");
        for(i=4;i>=1;i--)
        {
                scanf("%d",&q[i]);
        }
        printf("\n Step \t\t Action Performed \t\t A \t\tQ\n");
        printf("\n\n 0\t\t Initialization\t\t 0 0 0 0 0 \t ");
        for(i=4;i>=1;i--)
        {
                printf(" %d",q[i]);
        }
        for(j=1;j<=4;j++)
        {
                printf("\n\n %d",j);
                printf("\t\t Left Shift \t\t ");
                for(i=5;i>=2;i--)
                {
                        a[i]=a[i-1];
                }
                a[1]=q[4];
                for(i=4;i>=2;i--)
                {
                        q[i]=q[i-1];
                }
                for(i=5;i>=1;i--)
                {
                        printf(" %d",a[i]);
                }
                printf("\t ");
                for(i=4;i>=2;i--)
                {
                        printf(" %d",q[i]);
                }
                if(a[5]==0)
                {
                        z=0;
                        for(i=1;i<=5;i++)
                        {
                                s[i]=a[i]^mc[i]^z;
                                c=(a[i]&&mc[i])||(a[i]&&z)||(mc[i]&&z); z=c;
                                a[i]=s[i];
                        }
                        if(a[5]==1)
                        {
                                q[1]=0;
                        }
```

```
                else
                {
                        q[1]=1;
                }
                printf("\n\n\t\t a = a-m\t\t ");
                for(i=5;i>=1;i--)
                {
                        printf(" %d",a[i]);
                }
                printf("\t ");
                for(i=4;i>=1;i--)
                {
                        printf(" %d",q[i]);
                }
        }
        else
        {
                z=0;
                for(i=1;i<=5;i++)
                {
                        s[i]=a[i]^m[i]^z;
                        c=(a[i]&&m[i])||(a[i]&&z)||(m[i]&&z);
                        z=c;
                        a[i]=s[i];
                }
                if(a[5]==1)
                {
                        q[1]=0;
                }
                else
                {
                        q[1]=1;
                }
                printf("\n\n\t\t a = a+m\t\t ");
                for(i=5;i>=1;i--)
                {
                        printf(" %d",a[i]);
                }
                printf("\t ");
                for(i=4;i>=1;i--)
                {
                        printf(" %d",q[i]);
                }
        }
    }
    if(a[5]==1)
    {
            printf("\n\n\n 5");
            for(i=1;i<=5;i++)
            {
                    s[i]=a[i]^m[i]^z;
                    c=(a[i]&&m[i])||(a[i]&&z)||(m[i]&&z);
                    z=c;
                    a[i]=s[i];
            }
            printf("\t\t a = a+m\t\t ");
            for(i=5;i>=1;i--)
            {
                    printf(" %d",a[i]);
            }
            printf("\t ");
            for(i=4;i>=1;i--)
            {
                    printf(" %d",q[i]);
            }
    }
    printf("\nQuotient [Q] :");
    for(i=4;i>=1;i--)
    {
            printf(" %d",q[i]);
    }
    printf("\nRemainder [A] :");
    for(i=4;i>=1;i--)
    {
            printf(" %d",a[i]);
    }
}
```

Result - Output

```
PS C:\Users\Admin\AppData\Local\Temp> cd "C:\Users\Admin\AppData\Local\Temp\" ; if ($?) { gcc tempCodeRunnerFile.c -o tempCodeRunnerFile } ;
if ($?) { .\tempCodeRunnerFile }

Enter Divisor [M] (BINARY FORM): 0
1
0
0

Enter Divident [Q] (BINARY FORM): 0
0
1
0

 Step          Action Performed              A            Q


 0             Initialization         0 0 0 0 0      0 0 1 0

 1             Left Shift              0 0 0 0 0      0 1 0

               a = a-m                 1 1 1 0 0      0 1 0 0

 2             Left Shift              1 1 0 0 0      1 0 0

               a = a+m                 1 1 1 0 0      1 0 0 0

 3             Left Shift              1 1 0 0 1      0 0 0

               a = a+m                 1 1 1 0 1      0 0 0 0

 4             Left Shift              1 1 0 1 0      0 0 0

               a = a+m                 1 1 1 1 0      0 0 0 0


 5             a = a+m                 0 0 0 1 0      0 0 0 0
Quotient [Q] : 0 0 0 0
Remainder [A] : 0 0 1 0
PS C:\Users\Admin\AppData\Local\Temp>
```

Conclusion - Implemented Non-Restoring division Algorithm for Binary numbers successfully.

Experiment No. - 10

Roll No. - AIMLD08

Date Of Performance
06/12/2021

Aim - C Program to find the floating point IEEE 754 representation.

Software Used - VS-CODE

Program -

Floating Point Representation.txt - Notepad

File Edit Format View Help

```
#include<stdio.h>
int binary(int n,int i)
{
    int k;
    for(i--;i>=0;i--)
   {
      k=n>>i;
      if(k&1)
          printf("1");
      else
         printf("0");
    }
}
typedef union
{
      float f;
      struct
      {
            unsigned int mantissa : 23;
            unsigned int exponent : 8;
            unsigned int sign : 1;
       }
       field;
}
myfloat;
int main()
{
           myfloat var;
           printf("Enter any float number: ");
           scanf("%f",&var.f);
           printf("%d ",var.field.sign);
           binary(var.field.exponent, 8);
           printf(" ");
           binary(var.field.mantissa, 23);
           printf("\n");
           return 0;
}
```

Result-Output -

```
Enter any float number: 105.625
0 10000101 10100110100000000000000
```

Conclusion - We are able to find the float number using floating point IEEE 754 representation successfully.

PAGE: a)
DATE: / /

Name : Prathamesh Chikankar

Roll No. and year : AIML008 and SE

Branch and Division : CSE-(AI&ML) and D

Subject : DLCOA

ASSIGNMENT NO. 1

1) Convert decimal number 123.45 into binary, Octal, hexadecimal and Base-N 5 system.

Conversion:

decimal Number 123.45

- into binary :-

integer $(123.45)_{10}$ fractional

integer

$123 \div 2 = 61$	remainder	1
$61 \div 2 = 30$	remainder	1
$30 \div 2 = 15$	remainder	0
$15 \div 2 = 7$	remainder	1
$7 \div 2 = 3$	remainder	1
$3 \div 2 = 1$	remainder	1
$1 \div 2 = 0$	remainder	1

fractional $(0.45)_{10}$

$0.45 \times 2 = 0.9$	integer	0
$0.9 \times 2 = 1.8$	integer	1
$0.8 \times 2 = 1.6$	integer	1
$0.6 \times 2 = 1.2$	integer	1
$0.2 \times 2 = 0.4$	integer	0
$0.4 \times 2 = 0.8$	integer	0
$0.8 \times 2 = 1.6$	integer	1

$\therefore (123.45)_{10} = (1101111.0111001)_2$

- into Octal :-

integer $(123)_{10}$

$123 \div 8 = 15$	remainder	3
$15 \div 8 = 1$	remainder	7
$1 \div 8 = 0$	remainder	1

$\therefore (123)_{10} = (173)_8$

for fractional $(0.45)_{10}$

$0.45 \times 8 = 3.6$ intger 3

$0.6 \times 8 = 4.8$ intger 4

$0.8 \times 8 = 6.4$ integer 6

$0.4 \times 8 = 3.2$ integer 3

$0.2 \times 8 = 1.6$ intger 1

$\therefore (0.45)_{10} = (34631)$

$\therefore (123.45)_{10} = (173.34631)_8$

- into Hexadecimal $(123.45)_{10}$

integer $(123)_{10}$

$123 \div 16 = 7$ simander B

$7 \div 16 = 0$ simandir 7

$(123)_{10} = (7B)_{16}$

fractional $(0.45)_{10}$

$0.45 \times 16 = 7.2$ intger 7

$0.2 \times 16 = 3.2$ integer 3

$0.2 \times 16 = 3.2$ integes 3

$0.2 \times 16 = 3.2$ integes 3

$\therefore (123.45)_{10} = (7B.7333)_{16}$

- into Base-5 system

integer $(123)_{10}$

$123 \div 5 = 24$ simaindes 3

$24 \div 5 = 4$ simoondes 4

$4 \div 5 = 0$ simaindes 4

$\therefore (123)_{10} = (443)_5$

decimal part $(0.45)_{10}$

$0.45 \times 5 = 2.25$ integer 2

$0.25 \times 5 = 1.25$ integer 1

$0.25 \times 5 = 1.25$ integer 1

$0.25 \times 5 = 1.25$ integer 1

$\therefore (0.45)_{10} = (0.2111)_5$

$\therefore (123.45)_{10} = (443.2111)_5$

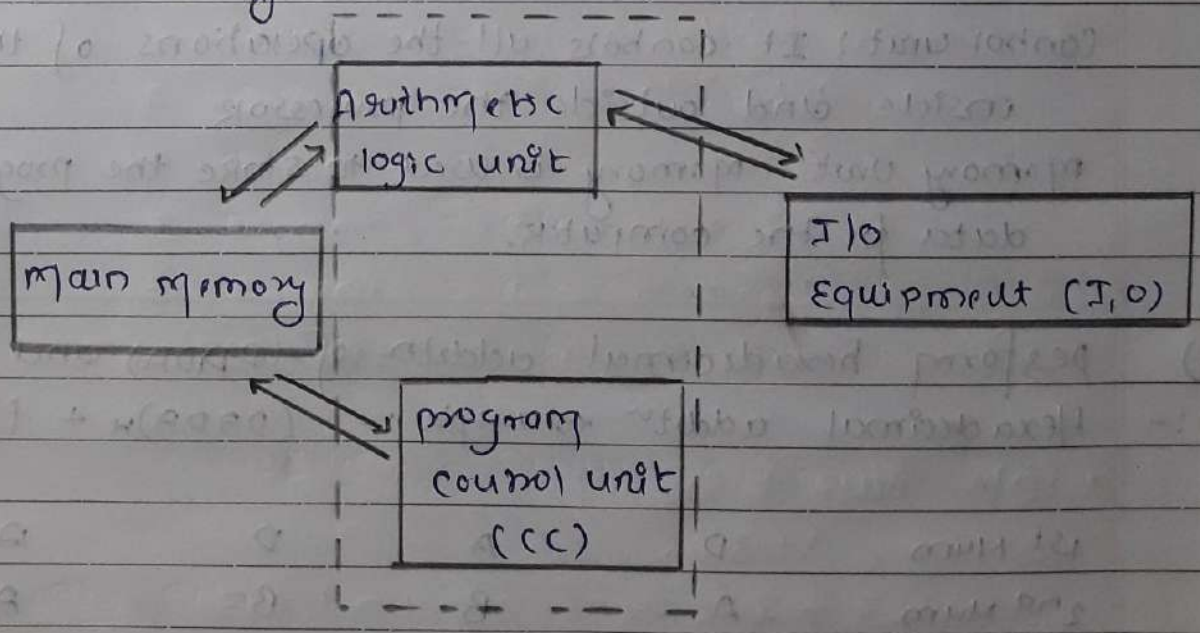
---

2) Explain the Von-Neumann architecture in detail.

Ans:-

processor — program memory and data memory

the name is derived from the mathematical and early computer scientists John Von-Neumann.



- The computer has a common memory for data as well as code to be executed.
- The processor needs two clock cycle to complete an instructions first to get an instruction and

Second to get the data.
- the system has 3-unit CPU, memory and other devices key features of a von Neumann machine.
- the von Neumann machine was stored program concept.
- the program and data are stored in the same memory unit.
- Each location of the memory has a unique address, i.e. no two locatⁿ have same memory location.

Input unit: A compiles accepts input from the uses through input devices.
Output unit: the result is given back by the compiles to the wax through an output device.
ALU: arithmetic or logical opeatⁿ like multiplicatⁿ, addⁿ, division, AND, OR, XOR, etc.. ase pesformed.
Conbol unit: It conbols all the opeations of the system inside and outside the processor
Memory Unit: Memory is used to stoRe the program and data for the computeR.

---

3) pesform hexadecimal addⁿ of (DADA) and (ABBA).

Soln:- Hexadecimal addⁿ example :- $(DADA)_H + (ABBA)_H$

1st Num	D	A	D	A
2nd Num	A	B	B	A
	$(D+A+1)_H$	$(A+B+1)_H$	$(D+B)+1)_H$	$(A+)_H$
	↓	↓	↓	↓
	$(13+10+1)_{10}$	$(10+11+1)_{10}$	$(13+11-1+1)$	$(10+10)_{10}$

1 1 1 1

$= (24)_{10} - (16)_{10}$ $= (22)_{10} - 16)_{10}$ $= (13+11+1)_{10}$ $= (10+10)_{10}$

$= (8)_{10}$ $= (6)_{10}$ $= (9)_{10}$ $= (4)_{10}$

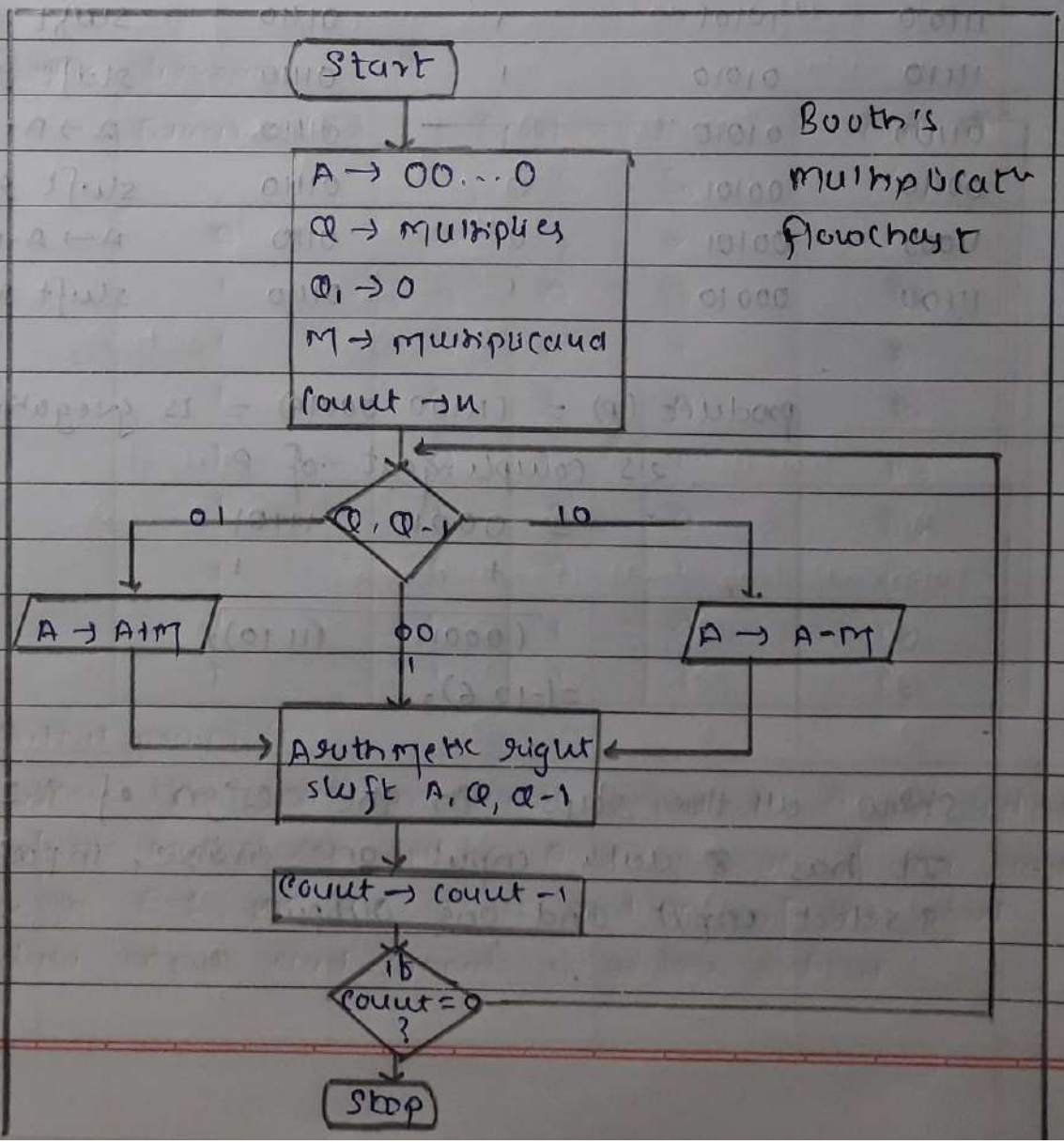
1 = 8 = 6 = 9 = 4

Hence, $(DADA)_H + (ABBH)_H$

$= (8694)_H$

4) Draw the flowchart for the Booth's multiplication and show all the iteration steps for the multiplication of (14) and (-9).

Ans:-



$(14) \times (-9)$

(M) → Multiplicand = $(14)_{10}$ = $(01110)_2$

(Q) → Multiplier = $(-9)_{10}$ = $(10111)_2$

2's complement $Q = M^{\sim} + 1$

$= 01000 + 1 \Rightarrow 01001$

A	Q	Q-1	M	Operation
00000	10111	0	01110	Initial value
10010	10111	0	01110	A → A-M
11001	01011	1	01110	shift right
11100	10101	1	01110	shift right
11110	01010	1	01110	shift right
01100	01010	1	01110	A → A-M
00110	00101	0	01110	shift right
11000	00101	0	01110	A → A-M
11100	00010	1	01110	shift right

product (P) = (11100 00010) = is negative

2's complement of P

= 00011 11101

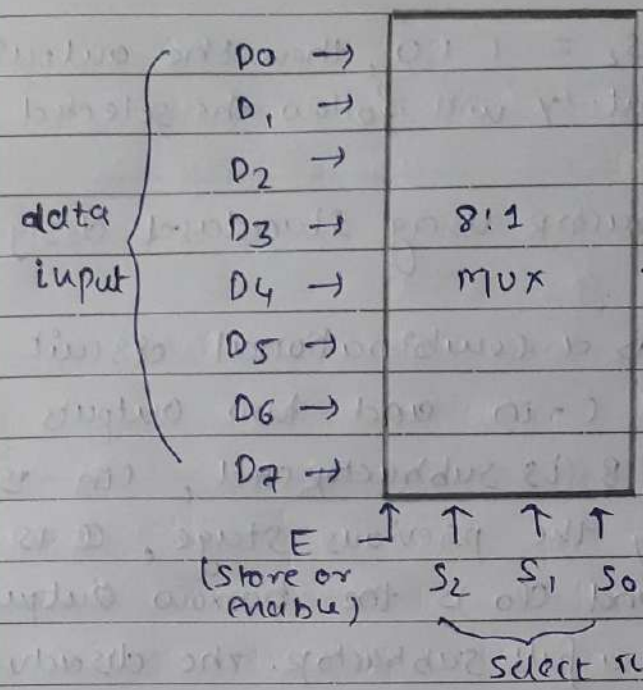
\+ 1

(00011 11110)$_2$

$= (-126)_2$

5) Show all the steps in the design of 8:1 mux.

Soln:- It has 8 data input, one enable, input, 3 select input and one output.



Block diagram

Enable (E)	Select input			Output
	S_2	S_1	S_0	'Y'
0	x	x	x	0
1	0	0	0	D_0
1	0	0	1	D_1
1	0	1	0	D_2
1	0	1	1	D_3
1	1	0	0	D_4
1	1	0	1	D_5
1	1	1	0	D_6
1	1	1	1	D_7

Output principle

- When the store or enable input is 0 the output of the multiplexer will be 0 irrespective of any input
- With E=1, we can select and one of the eight data input and connect it to the output.

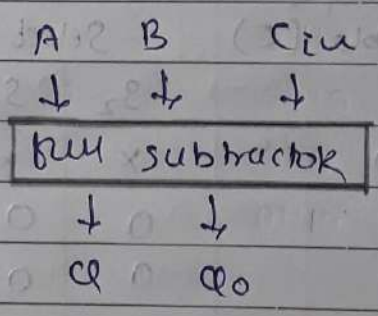
- for eg. if $S_0, S_1, S_2 = 1\ 1\ 0$, then the output D_3 is selected and output Y will follow the selected ruput D_5.

---

6) Design a full subtractor using standard design procedure processor.

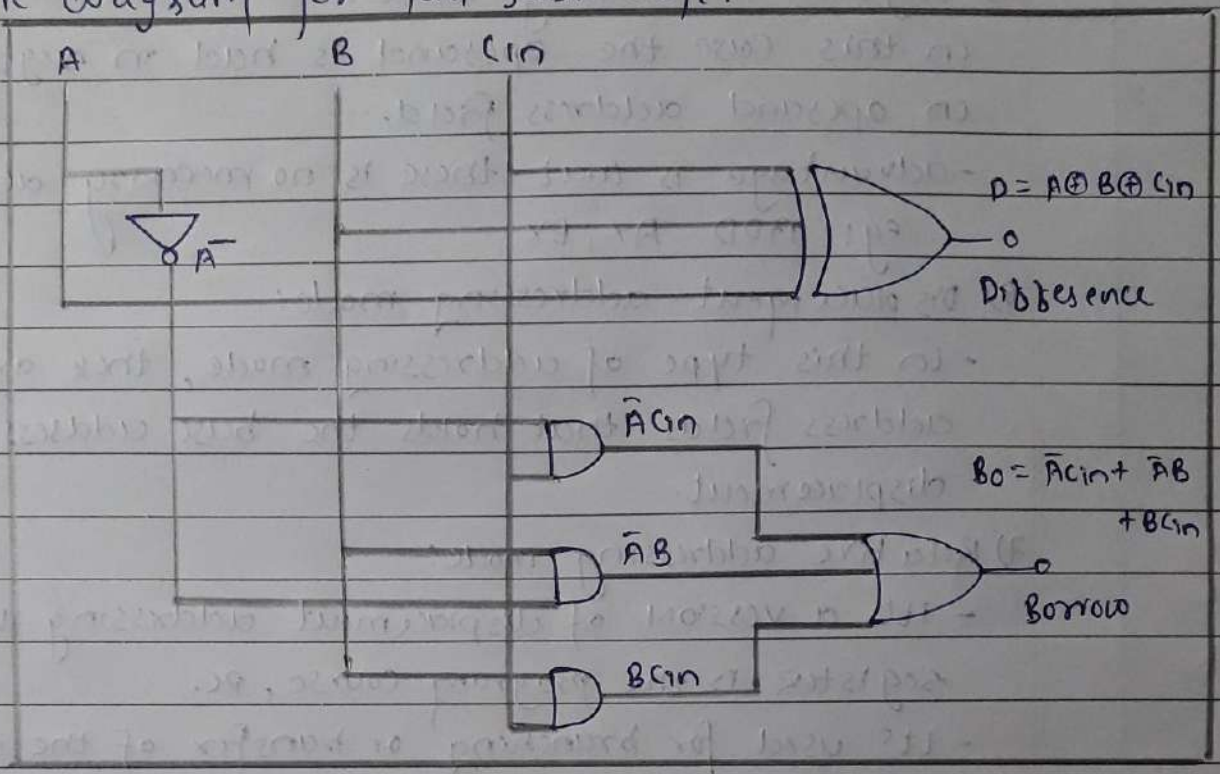
Ans:- A full subtractor is a combinational circuit with 3 inputs A, B and C-in and two outputs Q and Q_0

- A is a minvend, B is subtractor ctd, Cin is the Borrow produced by the previous stage, Q is the difference output and Q_0 is the borrow output.
- Show the symbol of a full subtractor. The disadvantage of half subtractor is overcome if we use the full subtractor.



Inputs			Outputs	
A (minvend)	B (subtrahend)	Cin (previous borrow)	(A-B-Cin)	Q_0
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

Logic diagram for full subtractor:-



7) List and explain various addressing modes of the generic processor.

Ans:-

1) Immediate addressing mode: The operand is part of instruction. the operand is in immediate next location of the op-code. for eg:- ADD AX, 000SH

2) Direct addressing mode: The address field contains memory address of operand. eg:- Intel 8085 instruction: LDA 1000 loads 8 bit data from address (1000)H to accumulator.

3) Indirect addressing mode: In this case memory location has the address of the operand in another memory location. eg:- ADD AX, [[1000]], disadvantage - slower.

4) Register addressing mode (indirect): In this the operand memory address is pointed to be content of register R. It requires one less memory access than indirect addressing mode.

5) Register addressing mode:
In this case the opesand is held in register named in opesand address field.
- advantage is that these is no memory access.
eg: ADD AX, BX

6) Displacement addressing mode:
- In this type of addressing mode, these are two address fields that holds the base addsess and the displacement.

7) Relative addressing mode:
- It is a vession of displacement addressing where the register is the program course, PC.
- It's used for branching or transfer of the instruct.

8) Stack addressing mode:
- used to access the data from the top of the stack
- used to push & pop instruct to access the stack.

PAGE: 01
DATE: / /

Name: Prathamesh Chikankar

Roll No. & year: AIML108 & SE

Branch & Division: ISE-(AI&ML) and D

Subject: DLCOA

Assignment No. 2

AIM1D08
PAGE : 01
DATE : / /

1) Enlist the properties of memory systems and explain each in brief.

Ans:- Locatn of memory

- CPU : this includes CPU registeR and on-chip cache memory.
- Internal: This includes memory processoR on directly accessed.
- External: This is normally removable or virtual memory and access is slowER.
- capacity: measured in terms of word size and number of words. word size is size of each locatn. Number of words is numbeR of locatn. The capacity of a memory is maximum no. of words it can store.
- unit of transfr: size of data that is transferred in one clock cycle.
- Access method: method that how memory locatn is accessed
  i) Sequential access
  ii) direct access
  iii) random access
  iv) associative access
- Performance: The performance of the memory depends on its speed of operatn or the data transfr rate.
- physical type: the physical type material using which the memory is made & can be different like
  i) semiconductoR
  ii) magnetic DISK
  iii) optical DisR.
- Organisatn: the memory can be sequentially organised or interleaved manner.

- physical characteristics: includes volatility power consumption and erasable, non-erasable properties.
- performance: Access time, Memory cycle time and memory Bandwidth.
- word size: No. of bit accessed from memory in one access cycle.

---

2) Explain cache mapping methods perform and show the cache mapping using all methods with following specifications: 32 KBytes cache memory, 32 MBytes of main memory and size of cache line 16 Bytes.

Ans:- there are three types of mapping methods.

i) Direct mapping
ii) 2 way set associative mapping
iii) fully associative mapping

- Direct mapping: in this case each block of main memory can map to only one cache line. Given block maps to line (i mod j) where i is line number and j is total memory number of line in cache memory.

memory size: (mm)

memory size (mm) = 32 bytes = 2^{25} bytes
cache line size = 16 Kbytes = 2^4 bytes
cache memory size = 32 KB = 2^{15} bytes
Size of mm block = $\frac{2^{25}}{2^4} = 2^{21}$

size of mm address = 25 bits
No. of cache line = $\frac{2^{15}}{2^4} = 2^{11}$.

Table:

Cache line	mm block		
0	0	2^{12}	2^{13}
1	1	$1^{12}+1$	$2^{13}+1$
2	2	$2^{12}+1$	$2^{13}+2$
⋮	⋮	⋮	⋮
$2^{11}-1$	$2^{11}-1$	$2^{13}-1$	$2^{14}-1$

Block offset
(size) = 4, Block number = 21

tag	line	word
10 bits	11 bits	4 bits

- N way set associate mapping: In this case cache is divided into N sets each set containing a number of lines. A given block can be in one of the N line of i set.

main memory size 32 Mbytes = 2^{25} bytes
Cache line size = 16 bytes = 2^4 bytes
Size of cache memory = 2^{15} bytes
Size of each tag = 2^{14} bytes
No. of line in each block = $\frac{2^{14}}{2^4} = 2^{10}$

No. of mm blocks = 2^{21} bytes

tag	set	word
11 bits	10 bits	4 bits.

Table:

Cache line		mm block	
0	0	2^{10}	2^{11}
1	1	1	1
2	2	2	2
⋮	⋮	⋮	⋮
$2^{10}-1$	$2^{10}-1$	$2^{11}-1$	$2^{12}-1$

- fully associative mapping : in this type of mapping, memory block on load into any line of cache. there are 2 fields tag and word

MM size $= 2^{25}$ bytes
Cache size $= 2^{15}$ bytes
cache line size $= 2^4$
No. of cache lines $= 2^{11}$ bytes

tag	word
21 bits	4 bits

3) Explain in details the concept of Hardwired control and unit design.

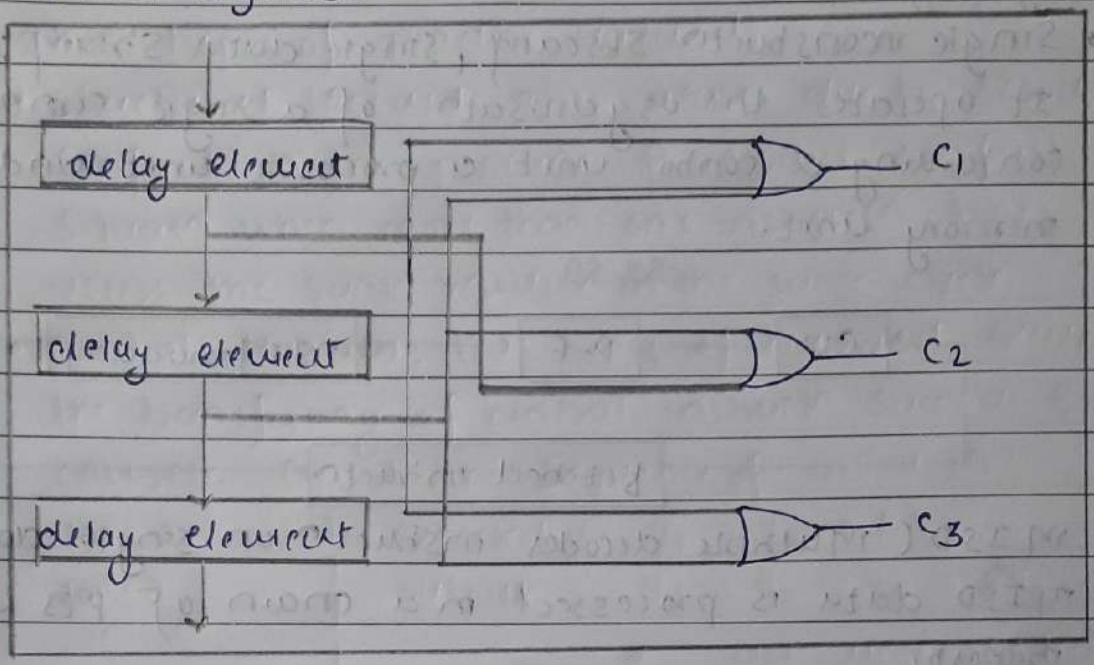
soln:- Hardwired control unit Design is viewed as a sequential an combinatorial logic circuit. It is used to generate a timed sequences of control signals.
It is implemented using standard digital logic circuits. The advantages of hard wired control units are speed and smaller space required of silicon wafer.
The different methods of its implementation are:

- State Table method: In this method, state transition for each instruction is made and a stable table is obtained. Input is formed using a dual input and status bit.

	I	I_2	...	I_m
S_1	$S_{11}Z_{11}$	$S_{12}Z_{12}$		$S_{1m}Z_{1m}$
S_2	$S_{21}Z_{21}$	$S_{22}Z_{22}$		$S_{2m}Z_{2m}$
:	:	:		

- Delay element method: Normally each delay element generates fixed time delay function output after delay are connected to or gates depending on actual of

control signals.



detailed block diagram of Hardwired control unit & explanation of components:

* Step decoder: It decodes control step counter and provides separate line for each step.
* Instruction decoder: It decodes the contents of IR and selects one pathway output as per machine instruction.
* Encoder: It inputs particular time slot from step decoder. it inputs machine instruction from instruction decoder.

---

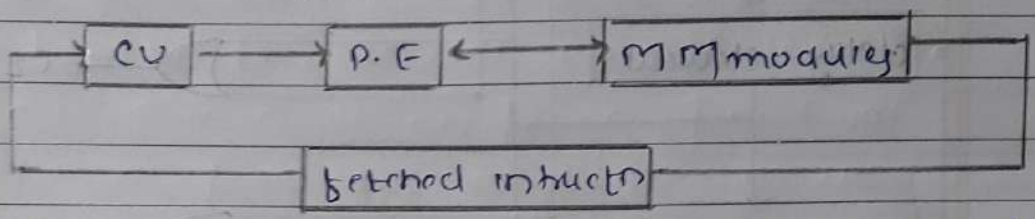
4) Classify the computer system as per the Flynn's classification scheme and explain each in brief.

Ans:- M.I Flynn proposed a classification for the organisation of a computer system by the numbers of instruction and data items are manipulated, simultaneously. Flynn classification divides computer into four -

major groups that are:

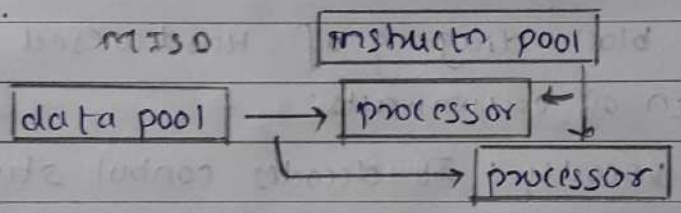
- Single instruction stream, single data stream: It operates the organisation of a single computer containing a control unit a processor unit and a memory unit.

SISD



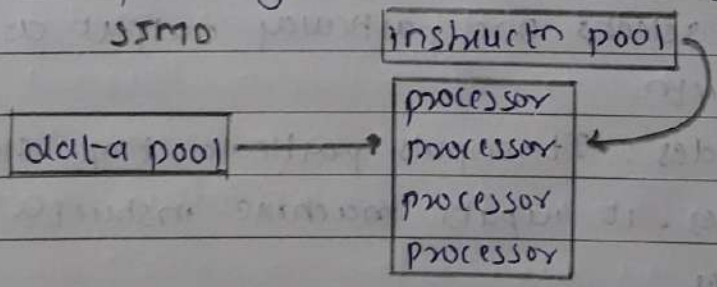
- MISD (Multiple decodes instruction on single data) MISD data is processed in a chain of PEs using property.

MISD



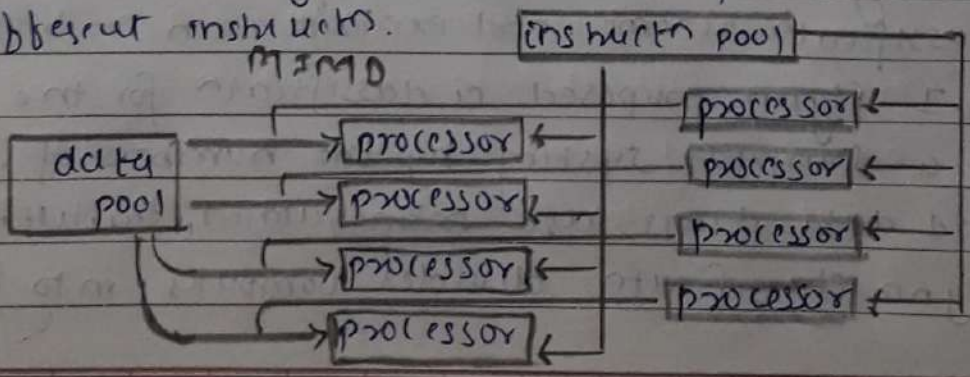
- Single instruction multiple data: Same instruction is given to different processing elements but different data.

SIMD



- Multiple instruction multiple data: Here every processing element is having a different set of data and different instructions.

MIMD



5) Enlist and explain various types of pipeline hazards.

Ans:- Three types of pipeline Hazards

- Structural Hazard: This arises due to resource conflict in the pipeline. A resource conflict is a situation when more than one instruction tries to access the same resource in the same cycle.
- Control Hazard: This types of hazard occurs due the transfering of control instruction such as BRANCH, CALL, etc.. Consider the sequence of instruction:

100 : I_1	Expected :	$I_1 \rightarrow I_2 \rightarrow BI_1$
101 : I_2	output :	$I_1 \rightarrow I_2 \rightarrow I_3 \rightarrow BI_1$
:	$\neq$	Not implemented.
250 : BI_1		

Output sequence : $I_1 \rightarrow I_2 \rightarrow$ delay $\rightarrow BI_1$

Total number of stalls

branch instruction = Branch frequency * Branch penalty.

- Data hazard: Data hazard occurred when instruction that exhebit data dependence, modify data in different stages of a pipeline. Hazard cause delays in pipeline. There are three mainly types of data Hazards.
  - RAW (read after write): Occurs when instruction I tries to read data before instruction I writes it.
  - WAR (write after read): Occurs when instruction I tries to write data before instruction I reads it.
  - WAW (write after write): Occurs when instruction I tries to write output before instruction I writes it.
  - WAR and WAW hazards occurs during the out of order execution of the instructions.