Experiment No. 9
Implement Non-Restoring algorithm using c-programming
Name:Ayush Mayekar
Roll No:28
Date of Performance:
Date of Submission:

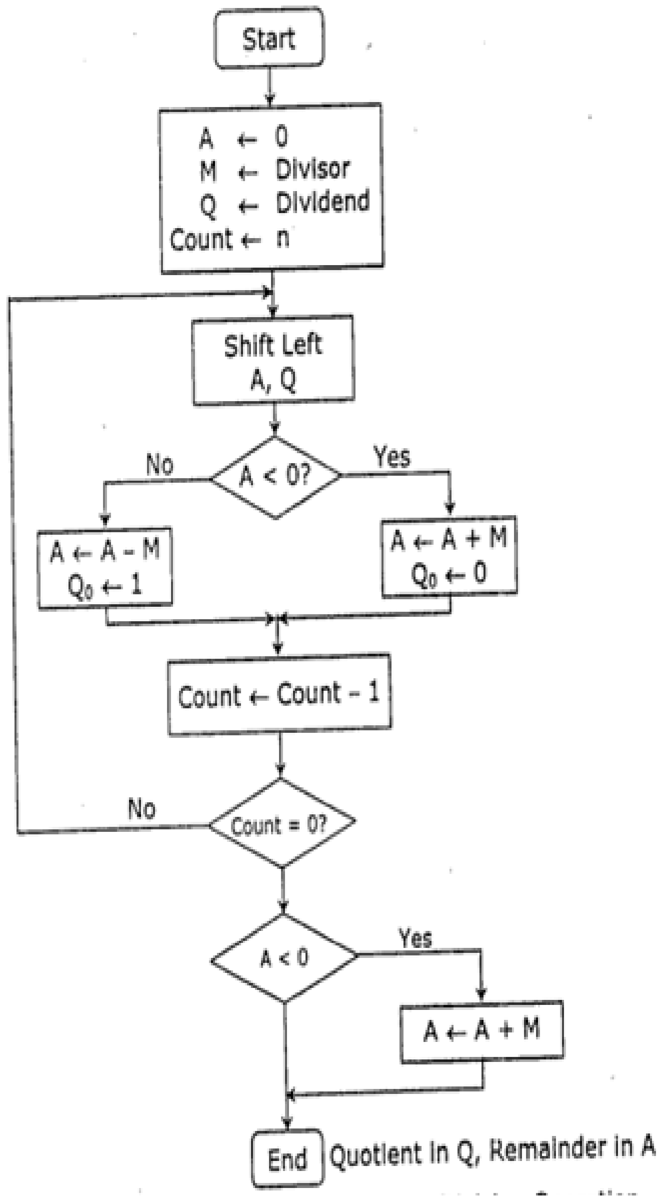
Aim - To implement Non-Restoring division algorithm using c-programming.

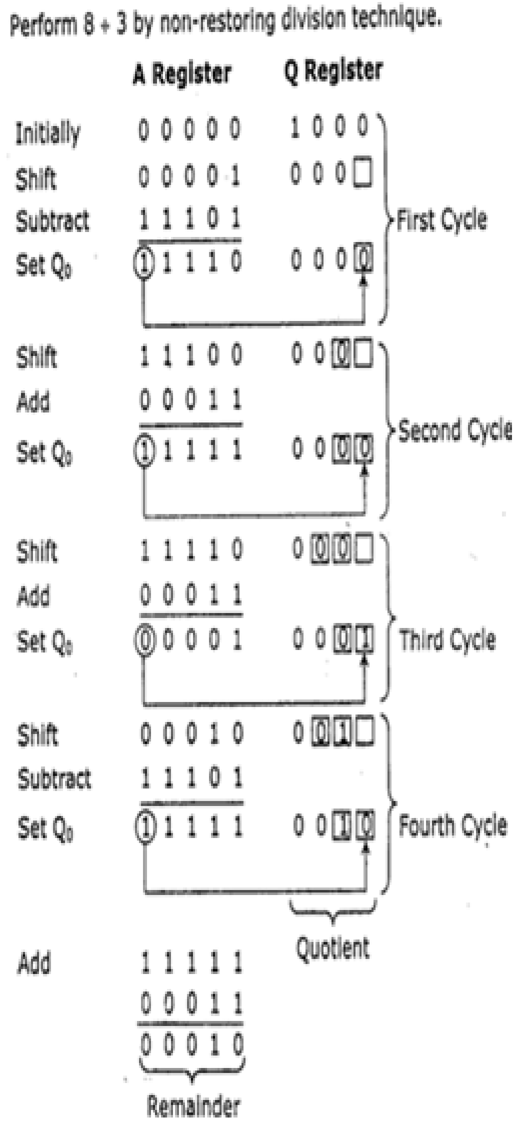
Objective -

1. To understand the working of Non-Restoring division algorithm.
2. To understand how to implement Non-Restoring division algorithm using c-programming.

Theory:

In each cycle content of the register, A is first shifted and then the divisor is added or subtracted with the content of register A depending upon the sign of A. In this, there is no need of restoring, but if the remainder is negative then there is a need of restoring the remainder. This is the faster algorithm of division.





Program -

```
#include <math.h>

#include <stdio.h>

//NON RESTORING DIVISION

int main()

{

int a[50],a1[50],b[50],d=0,i,j;

  int n1,n2, c, k1,k2,n,k,quo=0,rem=0;

          printf("Enter the number of bits\n");

          scanf("%d",&n);

  printf("Enter the divisor and dividend\n");

  scanf("%d %d", &n1,&n2);


  for (c = n-1; c >= 0; c--)//converting the 2 nos to binary

  {

          k1 = n1 >> c;
```

```
        if (k1 & 1)
        a[n-1-c]=1;// M
        else
        a[n-1-c]=0;

        k2 = n2 >> c;

        if (k2 & 1)
        b[2*n-1-c]=1;// Q
        else
        b[2*n-1-c]=0;

}

for(i=0;i<n;i++)//making complement
{
        if(a[i]==0)
        a1[i]=1;
        else
        a1[i]=0;
}

a1[n-1]+=1;//twos complement ie -M
```

```
 if(a1[n-1]==2)
 {
         for(i=n-1;i>0;i--)
         {
         if(a1[i]==2)
         {
         a1[i-1]+=1;
         a1[i]=0;
         }
         }
 }
 if(a1[0]==2)
         a1[0]=0;

 for( i=0;i<n;i++)// putting A in the same array as Q
 {
         b[i]=0;


 }

printf("A\tQ\tPROCESS\n");
```

```
 for(i=0;i<2*n;i++)
{
        if(i==n)
        printf("\t");

        printf("%d",b[i]);
}
printf("\n");

 for(k=0;k<n;k++)//n iterations
 {
        for(j=0;j<2*n-1;j++)//left shift
        {
        b[j]=b[j+1];

        }

        for(i=0;i<2*n -1;i++)
        {
        if(i==n)
                printf("\t");
        printf("%d",b[i]);
        }printf("_");
```

```
printf("\tLEFT SHIFT\n");


if(b[0]==0)
{
        for(i=n-1;i>=0;i--)//A=A-M
        {
                b[i]+=a1[i];


                if(i!=0)
        {
                if(b[i]==2)
                    {
                      b[i-1]+=1;
                      b[i]=0;
                         }
                if(b[i]==3)
                         {
                      b[i-1]+=1;
                      b[i]=1;
                         }
                // printf("%d",b[i]);
                }
```

```
        }
            if(b[0]==2)
            b[0]=0;

            if(b[0]==3)
            b[0]=1;

        for(i=0;i<2*n -1;i++)
        {
            if(i==n)
        printf("\t");



      printf("%d",b[i]);
        }printf("_");

        printf("\tA-M\n");
    }

    else
    {
        for(j=n-1;j>=0;j--)//A=A+M
```

```
            {
            b[j]+=a[j];

            if(j!=0)
        {
            if(b[j]==2)
                {
                b[j-1]+=1;
                b[j]=0;
                }
            if(b[j]==3)
                {
                b[j-1]+=1;
                b[j]=1;
                }
            }

            if(b[0]==2)
            b[0]=0;

            if(b[0]==3)
            b[0]=1;
            }
```

```
  for(i=0;i<2*n -1;i++)
{
        if(i==n)
      printf("\t");




   printf("%d",b[i]);
}printf("_");

     printf("\tA+M\n");

}




{
if(b[0]==0)//A==0?
{
b[2*n-1]=1;
for(i=0;i<2*n ;i++)
{
        if(i==n)
```

```
        printf("\t");




    printf("%d",b[i]);
}

printf("\tQ0=1\n");
}




if(b[0]==1)//A==1?
{
b[2*n-1]=0;
for(i=0;i<2*n ;i++)
{
            if(i==n)
        printf("\t");




    printf("%d",b[i]);
```

```
        }

        printf("\tQ0=0\n");

        }

 }

if(b[0]==1)
{
        for(j=n-1;j>=0;j--)//A=A+M
                {
                b[j]+=a[j];

                if(j!=0)
        {
                if(b[j]==2)
                        {
                    b[j-1]+=1;
                    b[j]=0;
                        }
                if(b[j]==3)
                        {
```

```
                b[j-1]+=1;
                  b[j]=1;
                  }
            }

            if(b[0]==2)
            b[0]=0;

            if(b[0]==3)
            b[0]=1;
            }

        for(i=0;i<2*n;i++)
      {
            if(i==n)
         printf("\t");

        printf("%d",b[i]);
      }

      printf("\tA+M\n");
```

```
}
printf("\n");
for(i=n;i<2*n;i++)
{
        quo+= b[i]*pow(2,2*n-1-i);
}
for(i=0;i<n;i++)
{
        rem+= b[i]*pow(2,n-1-i);
}
printf("The quotient of the two nos is %d\nThe remainder is %d",quo,rem);


printf("\n");
  return 0;
}
```

Output:

Terminal

```
Enter the number of bits
4
Enter the divisor and dividend
1010
0010
A   Q   PROCESS
0000    1010
0001    010_    LEFT SHIFT
1111    010_    A-M
1111    0100    Q0=0
1110    100_    LEFT SHIFT
0000    100_    A+M
0000    1001    Q0=1
0001    001_    LEFT SHIFT
1111    001_    A-M
1111    0010    Q0=0
1110    010_    LEFT SHIFT
0000    010_    A+M
0000    0101    Q0=1

The quotient of the two nos is 5
The remainder is 0
```

Conclusion -
Our experiment and code implementation of the Non-Restoring Division Algorithm have furnished us with significant insights into the realm of binary division. We've effectively demonstrated the algorithm's prowess in dividing binary numbers without necessitating restoring operations, rendering it particularly apt for hardware implementations where efficiency holds paramount importance. This experiment not only highlights the potential of algorithmic optimization in digital computation but also serves as a practical illustration of

non-restoring division's reliability in achieving precise binary division within a hardware context.