Experiment No. 8
Implement Restoring algorithm using c-programming
Name: Aryan Gaikwad
Roll Number: 01
Date of Performance:
Date of Submission:

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

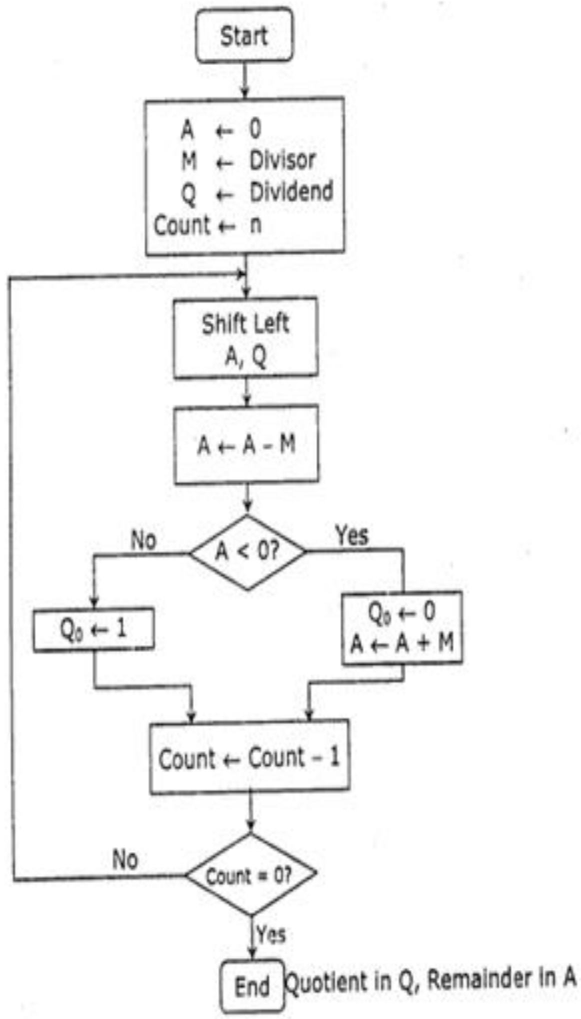
Objective -

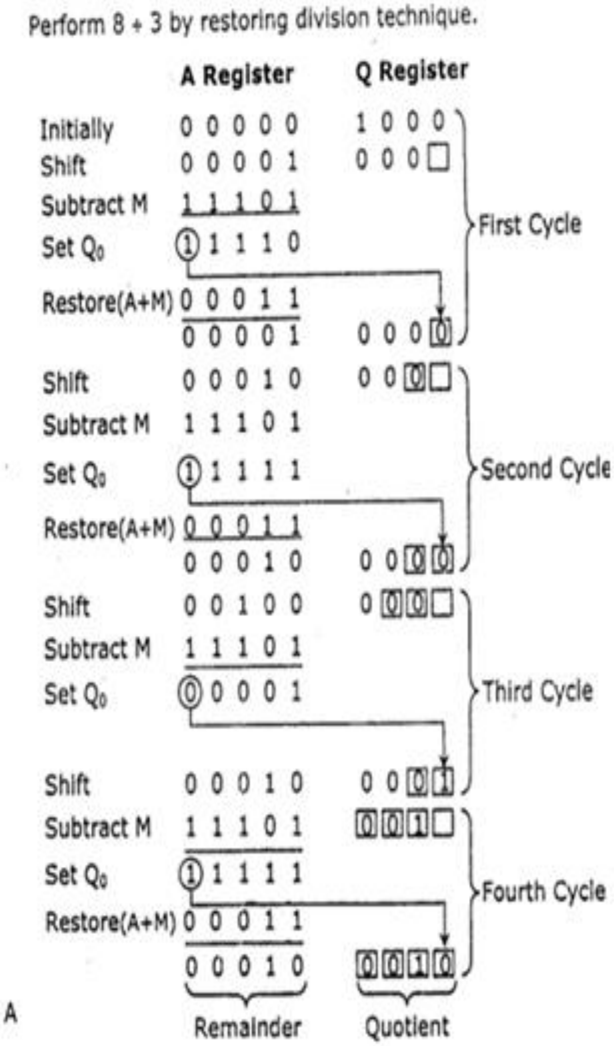
1. To understand the working of Restoring division algorithm.
2. To understand how to implement Restoring division algorithm using c-programming.

Theory:

1) The divisor is placed in M register, the dividend placed in Q register.
2) At every step, the A and Q registers together are shifted to the left by 1-bit
3) M is subtracted from A to determine whether A divides the partial remainder. If it does, then Q0 set to 1-bit. Otherwise, Q0 gets a 0 bit and M must be added back to A to restore the previous value.
4) The count is then decremented and the process continues for n steps. At the end, the quotient is in the Q register and the remainder is in the A register.

Flowchart





Program-

```
#include <stdio.h>
#include <stdlib.h>

int dec_bin(int, int []);
int twos(int [], int []);
int left(int [], int []);
int add(int [], int []);

int main()
{
   int a, b, m[4]={0,0,0,0}, q[4]={0,0,0,0}, acc[4]={0,0,0,0}, m2[4], i, n=4;
   printf("Enter the Dividend: ");
```

```
scanf("%d", &a);
printf("Enter the Divisor: ");
scanf("%d", &b);
dec_bin(a, q);
dec_bin(b, m);
twos(m, m2);
printf("\nA\tQ\tComments\n");
for(i=3; i>=0; i--)
{
   printf("%d", acc[i]);
}
printf("\t");
for(i=3; i>=0; i--)
{
   printf("%d", q[i]);
}
printf("\tStart\n");
while(n>0)
{
   left(acc, q);
   for(i=3; i>=0; i--)
   {
      printf("%d", acc[i]);
   }
   printf("\t");
   for(i=3; i>=1; i--)
   {
      printf("%d", q[i]);
   }
   printf("_\tLeft Shift A,Q\n");
   add(acc, m2);
   for(i=3; i>=0; i--)
   {
      printf("%d", acc[i]);
   }
   printf("\t");
   for(i=3; i>=1; i--)
   {
      printf("%d", q[i]);
```

```
        }
        printf("_\tA=A-M\n");
        if(acc[3]==0)
        {
            q[0]=1;
            for(i=3; i>=0; i--)
            {
                printf("%d", acc[i]);
            }
            printf("\t");
            for(i=3; i>=0; i--)
            {
                printf("%d", q[i]);
            }
            printf("\tQo=1\n");
        }
        else
        {
            q[0]=0;
            add(acc, m);
            for(i=3; i>=0; i--)
            {
                printf("%d", acc[i]);
            }
            printf("\t");
            for(i=3; i>=0; i--)
            {
                printf("%d", q[i]);
            }
            printf("\tQo=0; A=A+M\n");
        }
        n--;
    }
    printf("\nQuotient = ");
    for(i=3; i>=0; i--)
    {
        printf("%d", q[i]);
    }
    printf("\tRemainder = ");
```

```
   for(i=3; i>=0; i--)
   {
         printf("%d", acc[i]);
   }
   printf("\n");
   return 0;
}

int dec_bin(int d, int m[])
{
   int b=0, i=0;
   for(i=0; i<4; i++)
   {
      m[i]=d%2;
      d=d/2;
   }
   return 0;
}

int twos(int m[], int m2[])
{
   int i, m1[4];
   for(i=0; i<4; i++)
   {
      if(m[i]==0)
      {
         m1[i]=1;
      }
      else
      {
         m1[i]=0;
      }
   }
   for(i=0; i<4; i++)
   {
      m2[i]=m1[i];
   }
   if(m2[0]==0)
   {
```

```
    m2[0]=1;
  }
  else
  {
    m2[0]=0;
    if(m2[1]==0)
    {
      m2[1]=1;
    }
    else
    {
      m2[1]=0;
      if(m2[2]==0)
      {
        m2[2]=1;
      }
      else
      {
        m2[2]=0;
        if(m2[3]==0)
        {
          m2[3]=1;
        }
        else
        {
          m2[3]=0;
        }
      }
    }
  }
  return 0;
}

int left(int acc[], int q[])
{
  int i;
  for(i=3; i>0; i--)
  {
    acc[i]=acc[i-1];
```

```
    }
    acc[0]=q[3];
    for(i=3; i>0; i--)
    {
        q[i]=q[i-1];
    }
}

int add(int acc[], int m[])
{
  int i, carry=0;
  for(i=0; i<4; i++)
  {
    if(acc[i]+m[i]+carry==0)
    {
      acc[i]=0;
      carry=0;
    }
    else if(acc[i]+m[i]+carry==1)
    {
      acc[i]=1;
      carry=0;
    }
    else if(acc[i]+m[i]+carry==2)
    {
      acc[i]=0;
      carry=1;
    }
    else if(acc[i]+m[i]+carry==3)
    {
      acc[i]=1;
      carry=1;
    }
  }
  return 0;
}
```

Output -

Enter the Dividend: 12
Enter the Divisor: 2

A	Q	Comments
0000	1100	Start
0001	100_	Left Shift A,Q
1111	100_	A=A-M
0001	1000	Qo=0; A=A+M
0011	000_	Left Shift A,Q
0001	000_	A=A-M
0001	0001	Qo=1
0010	001_	Left Shift A,Q
0000	001_	A=A-M
0000	0011	Qo=1
0000	011_	Left Shift A,Q
1110	011_	A=A-M
0000	0110	Qo=0; A=A+M

Quotient = 0110 Remainder = 0000

Conclusion -

The Restoring Division Algorithm is a method for binary division used to divide two binary numbers. It operates by repeatedly subtracting the divisor from the dividend, while maintaining a quotient. If the result of the subtraction is negative, the quotient bit is set to 0, and the divisor is added back. If the result is positive, the quotient bit is set to 1, and the operation continues. The process iterates through the bits of the dividend, resulting in a quotient and a remainder. This algorithm is straightforward but can be slower than non-restoring division, which avoids adding the divisor back in case of a negative result, but it simplifies hardware design.