Batch: A3	**Roll No.: 16010121045**
Experiment / assignment / tutorial No. 4	
Grade: AA / AB / BB / BC / CC / CD /DD	
Signature of the Staff In-charge with date	

TITLE : To study and implement Non Restoring method of division

AIM : The basis of algorithm is based on paper and pencil approach and the operation involve repetitive shifting with addition and subtraction. So the main aim is to depict the usual process in the form of an algorithm.

Expected OUTCOME of Experiment: (Mention CO/CO's attained here)

To better understand the non-restoring algorithm and executing it using a programming language. To find the advantage of non-restoring over restoring division.

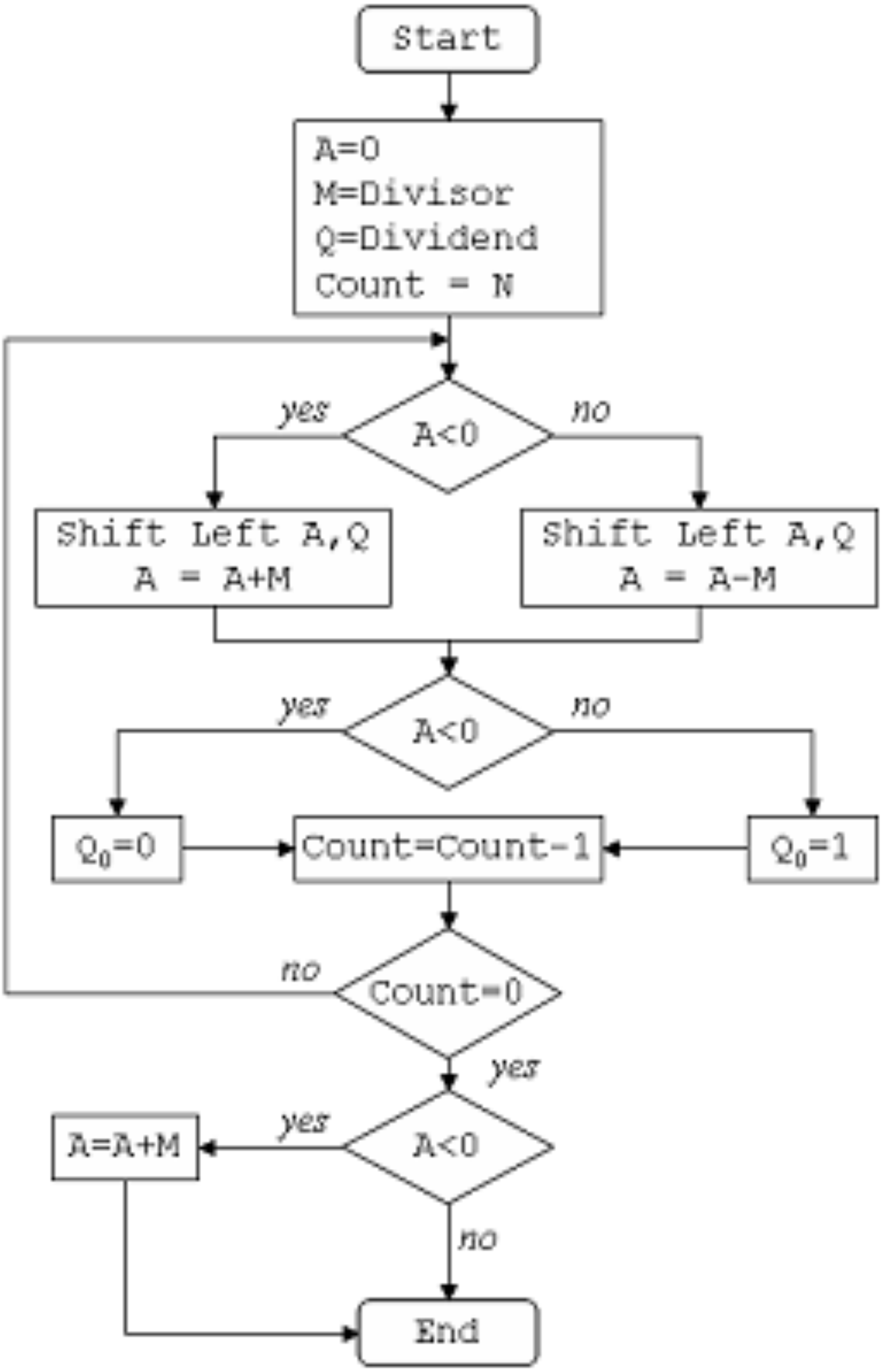
Books/ Journals/ Websites referred:

3. Carl Hamacher, Zvonko Vranesic and Safwat Zaky, "Computer Organization", Fifth Edition, TataMcGraw-Hill.
4. William Stallings, "Computer Organization and Architecture: Designing for Performance", Eighth Edition, Pearson.
3. Dr. M. Usha, T. S. Srikanth, "Computer System Architecture and Organization", First Edition, Wiley-India.

Pre Lab/ Prior Concepts:

The Non Restoring algorithm works with any combination of positive and negative numbers.

Flowchart for Non Restoring of Division:



Example: (Handwritten solved problem needs to uploaded)

Q = 11	01011	
M = 7	00111	
−M	11001	
A	Q	
00000	01011	~~initial~~ initial
00000	1011☐	shift left
11001	1011[0]	A ← A − M ; $Q_0 = 0$
10011	0110☐	shift left
11010	0110[0]	A ← A + M , $Q_0 = 0$
10100	1100☐	shift left
11011	1100[0]	A ← A + M , $Q_0 = 0$
10111	1000☐	shift left
11110	1000[0]	A ← A + M , $Q_0 = 0$
11101	0000☐	shift left
00100	0000[1]	A ← A + M

Q = 42   101010

M = 17   010001

−M   101111

A	Q	
000 000	101 010	initial value.
000 001	010 10□	shift left
110 000	01010[0]	A ← A − M, $Q_0 = 0$
100 000	10100□	shift left
110001	10100[0]	A ← A + M, $Q_0 = 0$
100011	01000□	shift left
110100	01000[0]	A ← A + M, $Q_0 = 0$
101000	10000□	shift left
111001	10000[0]	A ← A + M, $Q_0 = 0$
110011	00000□	shift left
000100	00000[1]	A ← A + M, $Q_0 = 1$
001000	00001□	shift left
110111	00001[0]	A ← A − M, $Q_0 = 1$
001000	000010	A ← A + M
Remainder	Quotient	

Code:

```
#include <bits/stdc++.h>
using namespace std;
int findbit(int m,int q){
    m=max(abs(m),abs(q));
    for(q=0;pow(2,q)<m;q++);
    return (max((q),4));
}
int* binary(int a,int num){
    int* ptr=(int*)malloc(num*sizeof(int));
    int acopy=abs(a),check=1;
    for (int i = 0;i<num; i++){
        ptr[i] = acopy % 2;
        acopy = acopy/2;
    }
    if (a < 0){
        for (int i = 0; i <num; i++){
            if (ptr[i] == 1 && check==1)
                check=0;
            else if(ptr[i] == 1 && check==0)
                ptr[i]=0;
            else if(ptr[i] == 0 && check==0)
                ptr[i]=1;
        }
    }
    return ptr;
}
void printbinary(int* ans,string s,int num){
    for(int i=(2*num)-1;i>num-1;i--)
        cout<<ans[i]<<" ";
    cout<<"\t";
    for(int i=num-1;i>=0;i--)
        cout<<ans[i]<<" ";
    cout<<"\t"<<s<<endl;
}
void binaryadd(int* ans,int* n,int num){
```

```
    int carry=0;
    for(int i=num;i<2*num;i++){
        if(ans[i]+n[i-num]+carry==1){
            ans[i]=1;
            carry=0;
        }
        else if(ans[i]+n[i-num]+carry==2){
            ans[i]=0;
            carry=1;
        }
        else if(ans[i]+n[i-num]+carry==3){
            ans[i]=1;
            carry=1;
        }
    }
}
int main()
{
    int m,q;
    cout<<"Enter Q and M: ";
    cin>>q>>m;
    int num=findbit(m,q);
    int ans[2*num]={0};
    int *arr=binary(q,num);
    for(int i=num-1;i>=0;i--)
        ans[i]=arr[i];
    cout<<endl<<"A\t\tQ\t\tOperation"<<endl<<endl;
    printbinary(ans,"Initial Value",num);
    for(int i=0;i<num;i++){
        if(ans[2*num-1]==1){
            for(int i=2*num;i>0;i--)
                ans[i]=ans[i-1]; // left Shifting
            printbinary(ans,"Shift Left",num);
            binaryadd(ans,binary(m,num),num);
            printbinary(ans,"A <- A + M",num);
        }
```

```
        else{
            for(int i=2*num;i>0;i--)
                ans[i]=ans[i-1]; // left Shifting
            printbinary(ans,"Shift Left",num);
            binaryadd(ans,binary(-m,num),num);
            printbinary(ans,"A <- A - M",num);
        }

        if(ans[2*num-1]==1){
            ans[0]=0;
            printbinary(ans,"Qo = 0",num);
        }
        else{
            ans[0]=1;
            printbinary(ans,"Qo = 1",num);
        }
    }
    if(ans[2*num-1]==1){
            binaryadd(ans,binary(m,num),num);
            printbinary(ans,"A <- A + M",num);
        }
    printbinary(ans,"Final Answer",num);
}
```

Output:

```
Enter Q and M: 11 7

A               Q               Operation

0 0 0 0         1 0 1 1         Initial Value
0 0 0 1         0 1 1 1         Shift Left
1 0 1 0         0 1 1 1         A <- A - M
1 0 1 0         0 1 1 0         Qo = 0
0 1 0 0         1 1 0 0         Shift Left
1 0 1 1         1 1 0 0         A <- A + M
1 0 1 1         1 1 0 0         Qo = 0
0 1 1 1         1 0 0 0         Shift Left
1 1 1 0         1 0 0 0         A <- A + M
1 1 1 0         1 0 0 0         Qo = 0
1 1 0 1         0 0 0 0         Shift Left
0 1 0 0         0 0 0 0         A <- A + M
0 1 0 0         0 0 0 1         Qo = 1
0 1 0 0         0 0 0 1         Final Answer
pargat@Router Programs %
```

```
pargat@Router Programs % cd "/Users/pargat/Docume
Enter Q and M: 42 17

A               Q               Operation

0 0 0 0 0 0     1 0 1 0 1 0     Initial Value
0 0 0 0 0 1     0 1 0 1 0 0     Shift Left
1 1 0 0 0 0     0 1 0 1 0 0     A <- A - M
1 1 0 0 0 0     0 1 0 1 0 0     Qo = 0
1 0 0 0 0 0     1 0 1 0 0 0     Shift Left
1 1 0 0 0 1     1 0 1 0 0 0     A <- A + M
1 1 0 0 0 1     1 0 1 0 0 0     Qo = 0
1 0 0 0 1 1     0 1 0 0 0 0     Shift Left
1 1 0 1 0 0     0 1 0 0 0 0     A <- A + M
1 1 0 1 0 0     0 1 0 0 0 0     Qo = 0
1 0 1 0 0 0     1 0 0 0 0 0     Shift Left
1 1 1 0 0 1     1 0 0 0 0 0     A <- A + M
1 1 1 0 0 1     1 0 0 0 0 0     Qo = 0
1 1 0 0 1 1     0 0 0 0 0 0     Shift Left
0 0 0 1 0 0     0 0 0 0 0 0     A <- A + M
0 0 0 1 0 0     0 0 0 0 0 1     Qo = 1
0 0 1 0 0 0     0 0 0 0 1 1     Shift Left
1 1 0 1 1 1     0 0 0 0 1 1     A <- A - M
1 1 0 1 1 1     0 0 0 0 1 0     Qo = 0
0 0 1 0 0 0     0 0 0 0 1 0     A <- A + M
0 0 1 0 0 0     0 0 0 0 1 0     Final Answer
pargat@Router Programs %
```

Conclusion

Successfully executed and coded the algorithm for non-restoring division. In this experiment, Non-Restoring Division Algorithm is executed with the help of C++ programming.

The advantage of Non-Restoring Division over Restoring Division is better understood.

Post Lab Descriptive Questions

1. What are the advantages of non-restoring division over restoring division?

Non-restoring division uses the digit set {−1, 1} for the quotient digits instead of {0, 1}. Non-Restoring Division when implemented in hardware, there is only one decision and addition/subtraction per quotient bit; there is no restoring step after the subtraction, which potentially cuts down the numbers of operations by up to half and lets it be executed faster.
Restoring method: you add the divisor back, and put 0 as your next quotient digit
Non-restoring method: you don't do that - you keep negative remainder and a digit 1, and basically correct things by a supplementary addition afterwards.

Date: _____________ **Signature of faculty in-charge**