EXPERIMENT - 3

Aim: To study and implement non-restoring division algorithms.

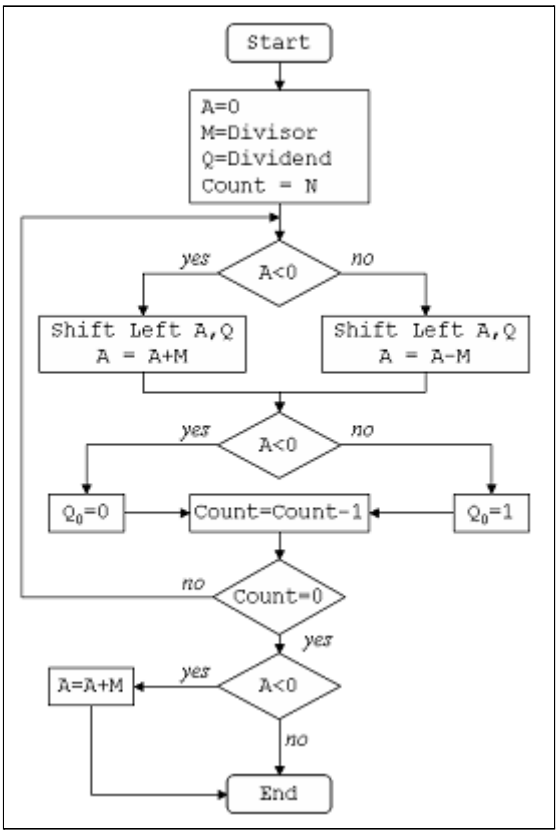
Submission Sheet

SAP ID	Name of Student	Date of Experiment	Date of Submission	Remarks
60004190057	Junaid Girkar	30-10-2021	30-10-2021	

Theory

A division algorithm is an algorithm which, given two integers N and D, computes their quotient and/or remainder, the result of Euclidean division. Division algorithms fall into two main categories: slow division and fast division. Slow division algorithms produce one digit of the final quotient per iteration. Examples of slow division include restoring, non-performing restoring, non-restoring, and SRT division. Fast division methods start with a close approximation to the final quotient and produce twice as many digits of the final quotient on each iteration. Newton–Raphson and Goldschmidt algorithms fall into this category.

Non-Restoring division, it is less complex than the restoring one because simpler operations are involved i.e. addition and subtraction, also now the restoring step is performed. In the method, rely on the sign bit of the register which initially contains zero named as A.
Here is the flow chart given below.



Algorithm

```
1) Set the value of register A as 0 (N bits)
2) Set the value of register M as Divisor (N bits)
3) Set the value of register Q as Dividend (N bits)
4) Concatenate A with Q {A,Q}
5) Repeat the following "N" number of times (here N is no. of bits in
divisor):
    If the sign bit of A equals 0,
      shift A and Q combined left by 1 bit and subtract M from A,
    else shift A and Q combined left by 1 bit and add M to A
    Now if sign bit of A equals 0, then set Q[0] as 1, else set Q[0]
as 0
6) Finally if the sign bit of A equals 1 then add M to A.
7) Assign A as remainder and Q as quotient.
```

Example:

Dividend (A) = 101110, ie 46, and Divisor (B) = 010111, ie 23.

Initialization :
Set Register A = Dividend = 000000
Set Register Q = Dividend = 101110
(So AQ = 000000 101110 , Q0 = LSB of Q = 0)
Set M = Divisor = 010111, M' = 2's complement of M = 101001
Set Count = 6, since 6 digits operation is being done here.

After this we start the algorithm, which I have shown in a table below :
In the table, SHL(AQ) denotes shift left AQ by one position leaving Q0 blank.
Similarly, a square symbol in Q0 position denote, it is to be calculated later

Action	A	Q	Count
Initial	000 000	101 110	6
A > 0 => SHL (AQ)	000 001	011 10□	
A = A-M	101 010	011 10□	
A < 0 => Q0 = 0	101 010	011 100	5
A < 0 => SHL (AQ)	010 100	111 00□	
A = A+M	101 011	111 00□	
A < 0 => Q0 = 0	101 011	111 000	4
A < 0 => SHL (AQ)	010 111	110 00□	
A = A+M	101 110	110 00□	
A < 0 => Q0 = 0	101 110	110 000	3
A < 0 => SHL (AQ)	011 101	100 00□	
A = A+M	110 100	100 00□	
A < 0 => Q0 = 0	110 100	100 000	2
A < 0 => SHL (AQ)	101 001	000 00□	
A = A+M	000 000	000 00□	
A < 0 => Q0 = 1	000 000	000 001	1
A > 0 => SHL (AQ)	000 000	000 01□	
A = A+M	101 001	000 01□	
A < 0 => Q0 = 1	101 001	000 010	0
Count has reached Zero, So final steps			
A < 0 => A = A+M	**000 000**	**000 010**	
	Reminder	**Quotient**	

CODE:

```
#include<stdio.h>
#include<malloc.h>
int *a,*q,*m,*mc,*c,n,d;
int powr(int x,int y)
{
 int s=1,i;
 for(i=0;i<y;i++)
   s=s*x;
 return s;
}
void print(int arr[],int n)
{
 int i;
 for(i=0;i<n;i++)
  printf("%d ",arr[i]);
}
void bin(int n, int arr[]){
    int r, i = 0;
    do{
        r = n % 2;
        n /= 2;
        arr[i] = r;
        i++;
    }while(n > 0);
 }
void set(int array[], int x){
    int i,tmp[20]={0};
    for(i = x -1; i >=0; i--)
  tmp[x-1-i]=array[i];
 for(i=0;i<x;i++)
  array[i]=tmp[i];
 }
int len(int x)
{
 int i=0;
 while(powr(2,i)<=x) i++;
 return ++i;
}
```

```
void addBinary(int a1[], int a2[])
{
    int bi[2]={0},ca[20]={0};
    int t=len(n),tmp=0;
    int *su=(int*)malloc(sizeof(int)*len(n));
 while(t-->0)
 {
  tmp=a1[t]+a2[t]+ca[t];
  bin(tmp,bi);
  su[t]=bi[0];
  ca[t-1]=bi[1];
  bi[0]=0;bi[1]=0;
 }
 for(t=0;t<len(n);t++)
  a1[t]=su[t];
 free(su);
 }

void twoCom(int arr[]){
    int i;
    int *one=(int*)malloc(sizeof(int)*len(n));
    for(i=0;i<len(n)-1;i++)
  one[i]=0;
 one[i]=1;
    for(i = 0; i < len(n); i++){
        arr[i]=1-arr[i];
    }
    addBinary(arr, one);
    free(one);
}
void ls(int alen,int blen)
{
 int i=0;
 for(i=0;i<alen-1;i++)
  a[i]=a[i+1];
 a[i]=q[0];
 for(i=0;i<blen-1;i++)
  q[i]=q[i+1];
 q[i]=-1;
}
```

```
void printaq()
{
 print(a,len(n));
 printf("\t");
 print(q,len(n)-1);
 printf("\t");
 printf("\n");
}
int main()
{
 int i,cnt=0;
 printf("Enter The Numerator/Denominator: ");
 scanf("%d/%d",&n,&d);
 q=(int*)malloc(sizeof(int)*len(n)-1);
 bin (n,q);
 m=(int*)malloc(sizeof(int)*(len(n)));
 bin(d,m);
 a=(int*)malloc(sizeof(int)*(len(n)));
 for(i=0;i<len(n);i++)
  a[i]=0;
 mc=(int*)malloc(sizeof(int)*(len(n)));
 bin(d,mc);
 set(q,len(n)-1);
 set(m,len(n));
 set(mc,len(n));
 twoCom(mc);
 cnt=len(n)-1;
 printf("\t    A\t\t   Q\t\t    M\t    Count\n");
 printf("\t---------\t-------\t\t--------- ---------\n");
 while(cnt>0)
 {
  printf("\t");
  print(a,len(n));
  printf("\t");
  print(q,len(n)-1);
  printf("\t");
  print(m,len(n));
  printf("\t%d\n",cnt);
  if(a[0]==1)
  {
```

```
   ls(len(n),len(n)-1);
   printf("LSHIFT\t");
   printaq();
   addBinary(a,m);
   printf("A=A+M\t");
   printaq();
  }
  else
  {
   ls(len(n),len(n)-1);
   printf("LSHIFT\t");
   printaq();
   addBinary(a,mc);
   printf("A=A-M\t");
   printaq();
  }
  if(a[0]==1)
  {
   q[len(n)-2]=0;
   addBinary(a,m);
  }
  else
   q[len(n)-2]=1;
  printf("A=A+M\t");
  printaq();
  cnt-=1;
  printf("\n");
 }
 return 0;
}
```

OUTPUT:

```
Enter The Numerator/Denominator: 10/3
            A                 Q                 M          Count
         ---------         -------           ---------  ---------
         0 0 0 0 0         1 0 1 0           0 0 0 1 1          4
LSHIFT   0 0 0 0 1         0 1 0 -1
A=A-M    1 1 1 1 0         0 1 0 -1
A=A+M    0 0 0 0 1         0 1 0 0

         0 0 0 0 1         0 1 0 0           0 0 0 1 1          3
LSHIFT   0 0 0 1 0         1 0 0 -1
A=A-M    1 1 1 1 1         1 0 0 -1
A=A+M    0 0 0 1 0         1 0 0 0

         0 0 0 1 0         1 0 0 0           0 0 0 1 1          2
LSHIFT   0 0 1 0 1         0 0 0 -1
A=A-M    0 0 0 1 0         0 0 0 -1
A=A+M    0 0 0 1 0         0 0 0 1

         0 0 0 1 0         0 0 0 1           0 0 0 1 1          1
LSHIFT   0 0 1 0 0         0 0 1 -1
A=A-M    0 0 0 0 1         0 0 1 -1
A=A+M    0 0 0 0 1         0 0 1 1
```

Conclusion :

The non-restorative division algorithm is an efficient way to perform binary division compared to traditional subtractive based algorithms by using the faster processed bit shift commands in the CPU registers. The algorithm is simple enough to be implemented in hardware in equipment like Arithmometers while also generalising to complex modern day systems. The algorithm serves as a good example in showing that considering lower-level system dependencies and physical limitations can be used to optimize algorithms. Non-restorative algorithm is more efficient than restorative algorithm as it uses simpler commands in terms of addition and subtraction, however it is slower than other algorithms.