LAB-1

NUMERIC CONVERSION

OBJECTIVES

1. To convert binary number into decimal number.
2. To convert decimal number into binary number.

THEORY

A number system is defined as the representation of numbers by using digits or other symbols in a consistent manner. The value of any digit in a number can be determined by a digit, its position in the number and the base of the number system.

Binary number system

It is the number system with base value 2. It uses two digits i.e. 0 and 1 for the creation of numbers. The numbers formed by using these two digits are termed as binary numbers. Binary number system is very useful in electronic devices and computer systems because it can be easily performed using just two states ON and OFF i.e. 0 and 1.

Decimal number system

It is the number system with base value 10, i.e from 0 to 9 is used for creation of numbers. Here, each digit in the number is at a specefic place with place value and product of different powers of 10.

PROGRAM

1. to convert binary number into decimal.

```
#include <iostream>
# include<conio.h>
# include <math.h>

  using namespace std;
int convert_binary_to_decimal (int);
int main()
   {
   int binary, decimal;
   cout <<"Enter the binary number :"<<endl;
   cin>> binary ;
   decimal = convert_binary_to_ decimal (binary);
   cout <<" The decimal number of the given binary
            number is :" << endl << decimal ;

  getch ();
  return 0;
 }
int convert_binary_to_decimal (int binary)
    {
    int decimal = 0 , power = 0, lsb ;
    while (binary!=0)
    {
    lsb = binary % 10;
    decimal = decimal + lsb * pow (2, power);
    binary = binary / 10;
    power ++;
    }
    return decimal ;
    }
```

OUTPUT

Output 1

```
Enter the binary number:
11110
The decimal number of the given binary number is:
30
```

Output 2

```
Enter the binary number:
1 1 1 1 1 1 1 1 1 1
The decimal number of the given binary number is:
1023
```

Output 3

```
Enter the binary number:
1111110010
The decimal number of the given binary number is:
1602
```

2. to convert decimal number into binary.

```
#include <iostream>
# include <conio.h>
# include <math.h>
   using namespace std;
int convert_decimal_to_binary (int);
int main()
   {
   int binary, decimal;
   cout << "Enter the decimal number :" <<endl;
   cin >> decimal;
   binary = convert_decimal_to_binary(decimal);
   cout << "The binary number of the given decimal
            number is:" <<endl<< binary;
getch();
return 0;
}
```

```
int convert_decimal_to_binary(int decimal)
{
int binary=0, reminder, multiplier=1;
while (decimal != 0)
{
reminder = decimal%2;
binary += reminder * multiplier;
decimal = decimal/2;
multiplier = multiplier * 10;
}
return binary;
}
```

OUTPUT

Output 1

```
Enter the decimal number:
17
The binary number of the given decimal number is:
10001
```

Output 2

```
Enter the decimal number:
1023
The binary number of the given decimal number is:
1111111111
```

Output 3

```
Enter the decimal number:
1024
The binary number of the given decimal number is:
1410065408
```

LAB 2

BINARY ADDITION

OBJECTIVE

- To implement addition algorithms for two binary numbers in C/C++

THEORY

Full Adder

Full adder is the adder that adds three inputs and produces two outputs. The first to inputs are A and B and the ~~tiro~~third ~~out~~ input is an input carry as C-IN. The output carry is designated as C-out and the normal output is designated as S which is sum. The ~~co~~C-out is also known as majority 1's detector, whose output value goes high when more than one input is high. A full adder logic is designed in such a manner that it can take eight inputs together to create a byte-wide adder and cascade the carry bit from one adder to another.

Full Adder truth table

Inputs			Output	
A	B	C-IN	Sum (S)	C-out
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

Here,

$Sum = A \oplus B \oplus C\text{-}IN$

$C\text{-}OUT = AB + BC\text{-}IN + AC\text{-}IN$

Program

→to implement binary addition

```
#include <iostream>
#include<conio.h>
  using namespace std;
  int binary_add(int* , int* , int* , int);
  int main()
  {
  int n;
  cout<<"Enter the size of binary number :";
  cin>>n;
  int *b1= new int [n];
  int *b2= new int [n];
  int *sum= new int[n];
  cout<<"Enter 1st binary num:"<<endl;
  for(int i=0 ; i<n; i++)
  {
  cin>> b1[i] ;
  }
  cout<<"Enter 2nd binary num:"<<endl;
  for (int i=0; i<n; i++)
  {
   cin>> b2[i];
  }
  int carry = binary_add(b1,b2,sum,n);
  cout<<"The result is:"<<endl;
  cout<< carry;
  for(int i=0;i<n;i++)
  {
  cout<<sum[i];
  }
  getch();
  return 0;
  }
```

```
int binary_add (int *b1, int *b2, int *sum, int n)
{
int carry = 0;
for (int i=n-1; i>=0; i--)
{
sum[i] = (b1[i] + b2[i] + carry) %2 ;
carry = (b1[i] + b2[i] + carry) /2 ;
}
return carry;
}
```

OUTPUT

Output 1

```
Enter the size of binary number : 6
Enter 1st binary num:
1 1 1 1 1 1
Enter 2nd binary num:
0 0 0 0 1 1
The result is :
1 0 0 0 0 1 0
```

Output 2

```
Enter the size of binary number : 10
Enter 1st binary num:
1 0 1 1 0 0 1 0 1 0
Enter 2nd binary num:
1 0 0 1 1 0 1 0 1 1
The result is:
1 0 1 0 0 1 1 0 1 0 1
```

DISCUSSION

In this lab session, we have learnt about binary addition and its implementation using C++ program. We have implement-ed the addition of two binary numbers using the concept of full adder circuit. As full adder circuit consists of three input and gets two output. The three input includes the n^{th} digit/bit of one binary number as second input consists of n^{th} digit/bit of next binary numbers and the third input is carry which can be obtained from ~~[illegible] n^{th} bi~~ result of addition of $(n-1)^{th}$ digit. For the addition of first digits the carry is always zero. And in full adder if there is more than one 1 in input then carry is produced or carry will be 1 else zero. We implement same concept in C++ for binary addition.

CONCLUSION

Hence, implementation of addition algorithm for two binary numbers in C++ was successful.

LAB 3

BINARY SUBTRACTION

OBJECTIVE

- To implement subtraction algorithms for two binary numbers in C/C++.

THEORY

Binary subtraction is performed in the same manner as decimal subtraction. However, there are some specetic rules regarding the subtraction among the binary digits 0 and 1 which we need to follow while performing binary subtraction. There are various ways to perform binary subtraction: i) subtraction without borrowing. ii) Subtraction with borrowing ii) Subtraction using 1's complement iv) subtraction using 2's complement.

Subtraction using 2's complement:

Binary subtraction with 2's complement entails adding the complement of the subtrahend to the minuend.
The steps for subtraction of binary number using 2's complement is as:

Step 1: Identify the minuend and the subtrahend.

Step 2: Find the 1's complement of the subtrahend and add 1 to it. (i.e making it 2's complement)

Step 3: Now add 2's complement of subtrahend to the minuend.

Step 4: If there is carry on addition (additional bit on result) then ignore the carry bit, the result is ~~negative or minus~~ remaining bits after ignoring carry.
If there is no carry then add 1 to the result of the addition of minuends ~~and~~ of subtrahend 2's complement, then the result obtained is the difference, in negative.

Here, when there occurs no carry in the addition of 2's complement of subtrahend and minuend then the difference is negative (i.e minuend is greater than subtrahend) else the difference is positive.

Program

→ to implement binary subtraction

```
#include <iostream>
#include <conio.h>
  using namespace std;
  int binary_add (int *, int *, int *, int);
  void get_2c (int *, int *, int);
  int subtract (int *, int *, int *, int);
  int main()
  {
  int n;
  cout <<"Enter the size of binary num:";
  cin >>n;
  int *minuend = new int [n];
  int *subtrahend = new int [n];
  int *difference = new int [n];
  cout <<"Enter minuend binary num:"<<endl;
  for (int i=0; i<n; i++)
  {
   cin >> minuend[i];
  }
  cout <<"Enter subtrahend binary num:"<<endl;
  for (int i=0; i<n; i++)
  {
   cin >> subtrahend [i];
   }
  int carry= subtract (minuend, subtrahend, difference, n);
  cout << "The result is:"<<endl;
  if (carry ==0)
  {
  cout <<" - ";
  int *twoscomp = new int [n];
  get_2c (difference, twoscomp, n);
 for (int i=0; i<n; i++)
 {
 cout << twoscomp[i];
 }}
 else
 {
 for (int i=0 ; i<n ; i++)
 {
 cout << difference [i];
 }}
```

```
getch();
return 0;
}
int binary_add (int *b1, int *b2, int *sum, int n)
{
int carry=0;
for (int i=n-1;i>=0;i--)
{
sum[i]=(b1[i]+b2[i]+carry)%2;
carry = (b1[i]+b2[i]+carry)/2;
}
return carry;
}
int subtract (int *m, int *s, int *diff, int n)
{
int *tc_s =new int[n];
get_2c (s, tc_s, n);
int carry = binary_add (m, tc_s, diff, n);
return carry;
}
void get_2c (int *bin, int *tc, int n)
{
int *oc = new int [n];
for (int i=0; i<n-1; i++)
{
oc[i]=bin[i]==0?1:0;
}
int *one = new int [n];
for (int i=0; i<n-1; i++)
{
one[i]=0;
}
one [n-1]=1;
binary_add (oc, one, tc, n);
}
```

OUTPUT

Output 1

Enter the size of binary num: 4
Enter minuend binary num: 1101

Enter subtrahend binary num: 1010

The result is:
0011

Output 2

Enter size of binary num: 5
Enter minuend binary num:
11110
Enter subtrahend binary num:
10010

The result is:
01100

Output 3

Enter size of binary num: 3
Enter minuend binary num: ~~11~~
111

Enter subtrahend binary num:
000

The result is:
~~=001~~ 111

DISCUSSION

In this lab session, we have learnt about binary subtraction and its implementation in C++. We implemented the subtraction using 2's complement method. At first we took minuend and subtrahend as input. Then we complemented every bit of subtrahend and add 1 to it (i.e. 2's complement of subtrahend) then add it to minuend. If the result of the addition occurs additional carry bit then ignore the carry bit, the remaining bit is the result. If there is no carry of carry is zero then add 1 to the result then negative of obtained result is the difference.

CONCLUSION

Hence, subtraction algorithm for two binary numbers in C++ was successful.

LAB 4

BINARY MULTIPLICATION

OBJECTIVES

To implement multiplication algorithm for two binary numbers in C/CH.

THEORY

The multiplication of two fixed point binary numbers is very similar to the usual multiplication method of integers. First, we need to multiply each digit of one binary number to each binary number.
As binary multiplication consist of base multiplication as: $1 \times 1 = 1$,

$1 \times 0 = 0$
$0 \times 1 = 0$
$0 \times 0 = 0$

Let us consider ~~cons~~ a multiplicand 101 and multiplier be 110 then binary multiplication is performed as.

Multiplier $\Rightarrow b_0 = 0, b_1 = 1, b_2 = 1$

Multiplicand $\Rightarrow B_0 = 1, B_1 = 0, B_2 = 1$

then,

$B_2 B_1 B_0 \times b_0 = 0\ 0\ 0$
$B_2 B_1 B_0 \times b_1 = 1\ 0\ 1$
$B_2 B_1 B_0 \times b_2 = 1\ 0\ 1$

then the result is added by shifting left ~~ev~~ as increase in every bit of multiplier (going towards MSB from LSB)

```
   000
+ 101
  1010
+ 101
-------
  11110
```

Program

```
#include <iostream>
#include <conio.h>
  using namespace std;
int add(int * ,int *,int* ,int);
void shift_left (int *,int);
void multiply (int *,int * , int *,int);
int main()
{
int n;
cout << "Enter the size of binary num:";
cin>>n;
int *multiplicand = new int[n];
int * multiplier = new int [n];
int * product =new int[n];[2*n];
cout <<"Enter multiplicand :"<<endl;
for(int i=0; i<n; i++)
{
 cin>> multiplicand [i];
}
cout <<"Enter multiplier:"<<endl;
for (int i=0; i<n; i++)
 {
 cin>> multiplier [n];
 }
multiply (multiplicand, multiplier, product, n)
cout << "The product is :"<< endl;
for(int i=0; i< 2*n ; i++)
{
cout <<*Product [i];
}
getch ();
return 0;
}
void multiply (int *md,int *mr, int *prod,int n)
{
int *s_md = new int [2*n];
```

```
for (int i=0; i<2*n ; i++)
{
prod[i] =0;
}
for(int i=0; i< 2*n; i++)
{
s_md[i]= i<n, i++
s_md[i] = i<n ? 0 : md[i-n];
}
for(int i=n-1; i>=0; i--)
{
if (mr[i])
{
add (prod, s_md, prod, 2*n);
}
shift_left (s_md, 2*n);
}
}
Void shift_left ( int *bin, int n)
{
for (int i=0 ; i<=n-2; i++)
{
bin[i]= bin[i+1];
}
bin[n-1] =0;
}
int add (int *b1, int *b2, int *sum , int n)
{
int carry =0, onescount;
for (int i=n-1; i>=0; i--)
{
onescount = b1[i] + b2[i]+carry;
sum[i] = onescount %2;
carry = onescount /2;
}
return carry;
}
```

OUTPUT

```
Enter the size of binary num: 4
Enter multiplicand : 1011
Enter multiplier: 1100
The product is: 10000100
```

DISCUSSION

In this lab session, we learned the process of binary multiplication using C++. Here while doing the binary multiplication we have performed the various operations such as addition, shifting. Here we have taken the no of bits of binary number whose multiplication is to be perforemed and also set is as count. We have taken input as multiplier and multiplicand and set product as prod. Here product is of maximum 2n as multip-lier and multiplicand are of n bits. Then at first we set product from 2n-1 to n bit as multiplicand data and from n-1 bit to 0 as 0 ~~multiplicand data~~. then we check if there is 1 on multiplier the we add multiplicand to product and if there is zero on multiplier no operation is performed and count is decreased if count is not zero the operation is repeated by shifting product left and if count is zero ~~so~~ the result is the product and the program is stopped. Using this algorithm we have implemented binary mult-iplication.

CONCLUSION

Hence, binary multiplication algorithm was successfully implemented

LAB 5

BINARY DIVISION

OBJECTIVES

→ To implement restoring and non restoring division algorithm.

THEORY

Division of two fixed point binary numbers in signed magnitude representation is done with paper by a process of successive compare, shift and subtract operations. Binary division is simpler than decimal division because the quotient digits are either 0 or 1 and there is no need to estimate how many times the dividend or partial remainder fits into the divisor.

When the division is implemented in a digital computer, it is convinient to change the process slightly. Instead of shifting the divisor to right, the dividend or the partial ~~divisor~~ remainder is shifted to the left, thus leaving the two numbers in the relative position. Subtraction may be achieved by the 2's complement method.

There are two methods for implementing binary division in hardware level:

i) Restoring Method

ii) Non-Restoring Method

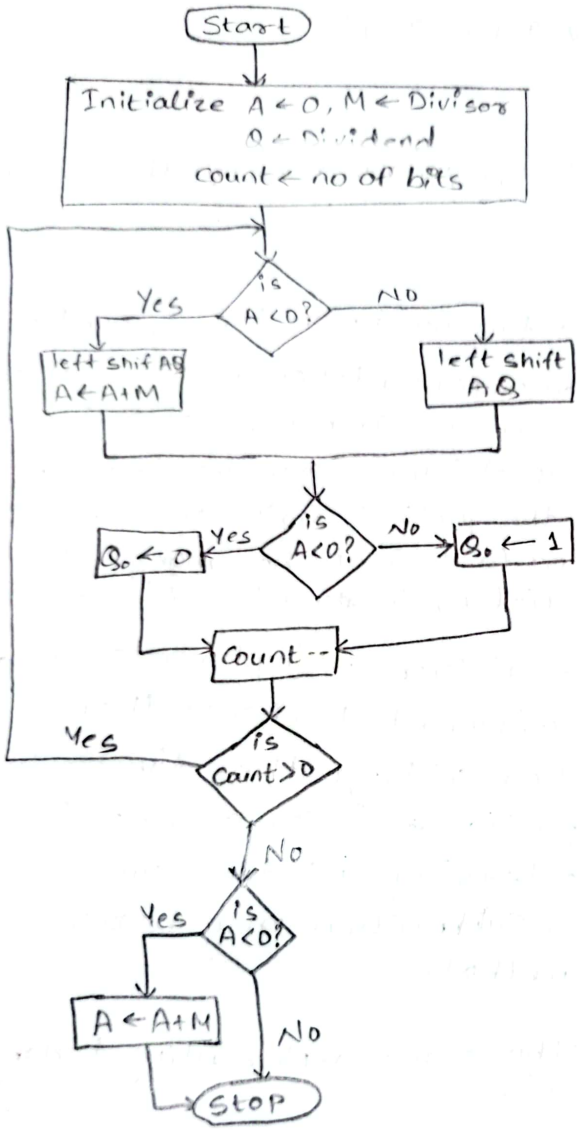

fig: flowchart of non-restoring algorithm

Non-Restoring Division Algorithm

```
#include <iostream>
# include <conio.h>
  using namespace std;
  void shift_left (int *, int *, int);
  int add (int * , int * , int * , int);
  void complement (int * , int * , int) ;
  void divide (int *, int*, int*, int);
  void shift_left (int *, int * , int);
int main ()
 {
 int  n;
 cout << "Enter the number of bits" <<endl;
 cin >> n;
 int * dividend = new int [n];
 int * divisor = new int [n];
 int * A = new int [n];
 int * Q = new int [n];
 cout << "Enter dividend :" <<endl;
 for (int i = 0; i<n; i++)
 {
 cin >> dividend [i];
 }
 cout << "Enter divisor" <<endl;
for (int i=0; i<n; i++)
{
cin >> divisor [i];
}
for (int i=0; i<n; i++)
 {
 Q[i]= dividend [i];
 }
divide (A, Q, divisor, n);
cout << "The quotient is:" <<endl
for(int i=0 ; i<=n; i++)
{
cout << Q[i];
}
```

```
cout<<"Remainder is :"<<endl;
for(int i=0; i<=n; i++)
{
cout<<A[i];
}
getch();
return 0;
}
int add(int *b1, int *b2, int *sum, int n)
{
int carry =0;
for(int i=n-1; i>=0; i--)
{
int onescount=b1[i]+b2[i]+carry;
 sum[i]= onescount %2;
 carry= onescount /2;
}
return carry;
}
void complement (int *bin, int *twoscomp, int n)
{
int *newbin;
newbin= new int[n];
int * ones;
ones=new int[n];
for(int i=0; i<n; i++)
{
ones[i]= (i==n-1)?1:0;
}
add(newbin, ones, twoscomp, n);
}
void divide(int *A, int *B, int *divisor, int n)
{
int *twoscomp;
twoscomp= new int[n+1];
complement(divisor, twoscomp, n+1);
for(int i=n; i>0; i--)
{
if(A[0])
{
shift_left(A,B,n);
add(A, divisor, A, n+1);
```

```
    if (A[0])
    {
    Q[n-1]=0;
    }
    else
    Q[n-1]=1;
  }
  else
  {
   shift_left(A,Q,n);
   add(A, twoscomp, A,n+1);
   if(A[0] Q[n-1]=0;
   else Q[n-1]=1;
  }
  add(A, divisor, A,n+1);
  }
  void shift_left(int *A,int *Q, int n)
  {
  for(int i=0; i<n; i++)
  {
  A[i]=A[i+1];
  }
  A[n]=Q[0];
  for(int i=0; i<=n-2; i++)
  {
  Q[i]=Q[i+1];
  }
  }
```

OUTPUT

```
Enter the number of bits:
3
Enter dividend:
1 0 1
Enter divisor:
0 1 1
The quotrent is:
0 0 1
The remainder is:
0 1 0
```

Restoring Division Algorithm

```
#include <iostream>
#include <conio.h>
using namespace std;
int add(int *b1, int *b2, int *sum, int n)
{
 int carry=0;
 for(int i=n-1; i>=0; i--)
 {
  int onescount = b1[i] + b2[i] + carry;
  sum[i] = onescount %2;
  carry = onescount /2;
 }
 return carry;
}
void complement(int *bin, int *twoscomp, int n)
{
 int *newbin;
 newbin = new int[n];
 int *ones;
 ones = new int[n];
 for(int i=0; i<n; i++)
 {
 newbin[i]= (bin[i]==0)?1:0;
 }
 for(int i=0; i<n; i++)
 {
  ones[i]= (i==n-1)? 1:0;
 }
 add(newbin, ones, twoscomp, n);
}
void divide(int *A, int *Q, int *divisor)
{
int *twoscomp;
twoscomp = new int[n+1];
complement(divisor, twoscomp, n+1);
if(A[0])
{
Q[n-1] = 0;
add(A, divisor, A, n+1);
}
else
Q[n-1] = 1;
}
```

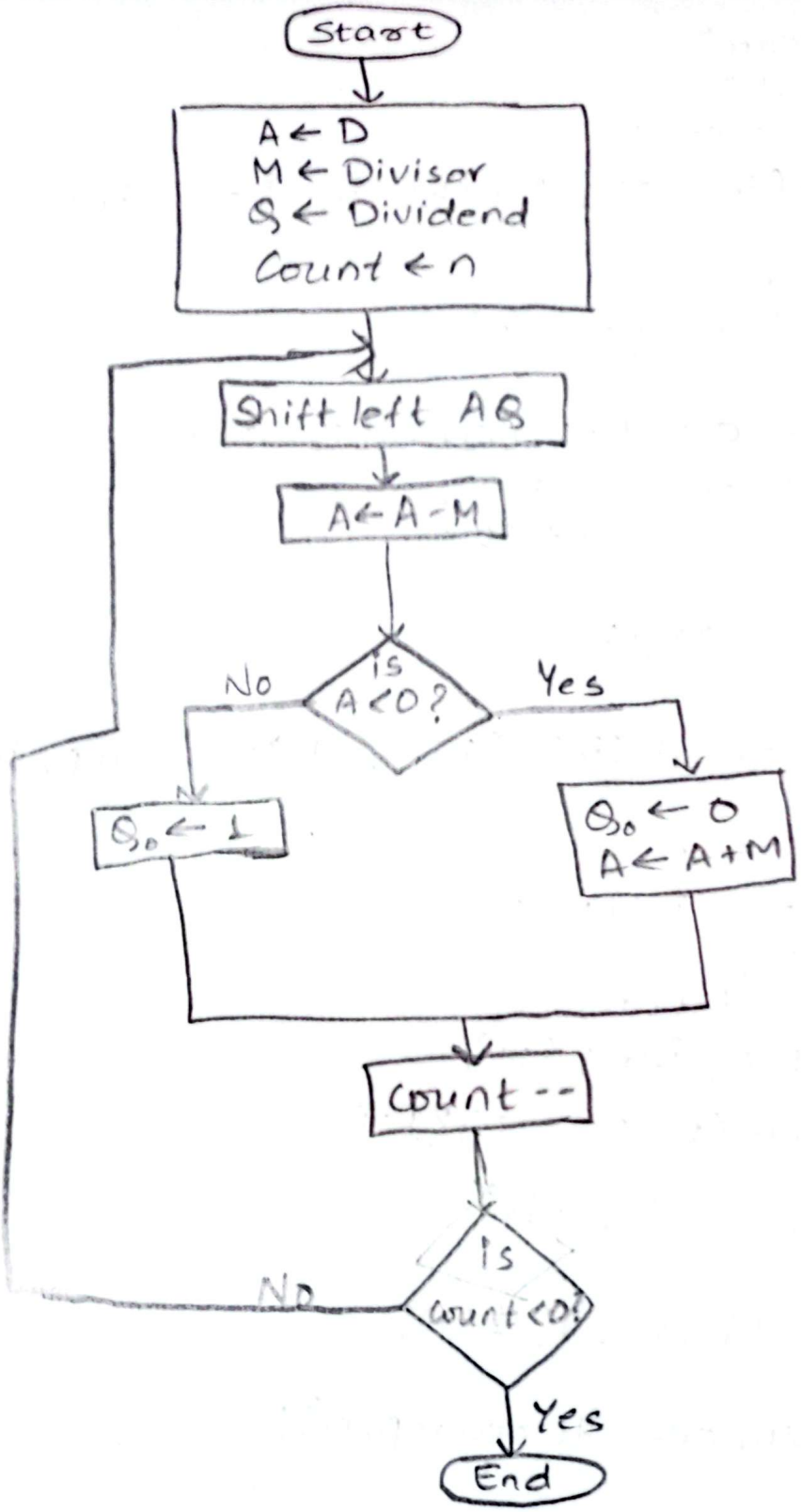

fig: flowchart of restoring division algorithm

```
void shift_left (int *A, int *Q, int n)
{
for (int i=0; i<n; i++)
{
A[i] = A[i+1];
}
A[n]=Q[0];
for(int i = 0; i<n-2; i++)
{
Q[i]=Q[i+1];
}
int main()
{
int n;
cout<<"Enter the number of bits:"<<endl;
cin>>n;
int*dividend= new int[n];
int * divisor = new int[n];
int * A = new int[n+1];
int *Q = new int[n];
cout <<"Enter dividend"<<endl;
for(int i=0; i<n; i++)
{
cin>>dividend[i];
}
cout <<"Enter divisor:"<<endl;
for (int i=0; i<=n; i++)
{
cin>>divisor[i];
A[i]=0
}
divisor[0]= A[0]= 0;
for(int i=0; i<n; i++)
{
Q[i]=dividend[i];
}
divide(A,Q,Divisor,n);
cout<<"The quotient is:"<<endl;
for (i=0; i<n; i++){
cout<<Q[i];
}
cout<<"The remainder is:"<<endl;
for(i=0; i<n; i++) {
cout<<A[i]; }
getch();
return 0;
}
```

OUTPUT

```
Enter the no of bits:
3
Enter the dividend:
1 1 1
Enter the divisor:
0 1 1
The quotient is:
0 1 0
The remainder is:
0 0 1
```

DISCUSSION AND CONCLUSION

In this lab session we have learnt and implemented binary division algorithms. We have used two algorithms i.e restoring algorithm and non-restoring algorithm. Here we have initialize A, Q, dividend, divisor where A gives remainder, Q gives quotient. In non-restoring algorithm variable A and divisor are made of one more bit by placing 0 bit at the start/front of the bits. Here at first all bits of variable A is set to zero. Here dividend value was put in the Q's array. The count was initialized to the no of bits used for operation. Then the value of A is checked, if it is less than zero, the bits are left shifted and A is added with divisor else A is subtracted with divisor and left shifting of A and Q is done. The same process is repeated until count becomes zero. At last then the value of A is again checked if it is negative then the value of A and divisor is added and result is printed if not negative then result is not changed.

Similarly in restoring algorithm no any extra bit is added and the value of A and Q are left shifted and divisor is subtracted from A if there is negative A then Q_0 is set to 0 and divisor is added to A else Q_0 is set to zero and count is decremented and this process is continued until count becomes zero. Hence, we have successfully implemented binary division algorithms.