Batch: B2 **Roll No:-** 16010122151

Experiment / Assignment / Tutorial No:-2

TITLE: To study and implement Booth's Multiplication Algorithm.

AIM: Booth's Algorithm for Multiplication

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

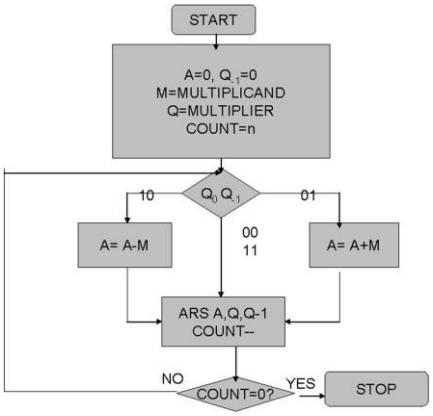
Books/ Journals/ Websites referred:

1. Carl Hamacher, Zvonko Vranesic and Safwat Zaky, "Computer Organization", Fifth Edition, TataMcGraw-Hill.
2. 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:

It is a powerful algorithm for signed number multiplication which generates a 2n bit product and treats both positive and negative numbers uniformly. Also the efficiency of the algorithm is good due to the fact that, block of 1's and 0's are skipped over and subtraction/addition is only done if pair contains 10 or 01

Flowchart:



Design Steps:

1. Start
2. Get the multiplicand (M) and Multiplier (Q) from the user
3. Initialize A= Q_{-1} =0
4. Convert M and Q into binary
5. Compare Q_0 and Q_{-1} and perform the respective operation.

Q_0 Q_{-1}	**Operation**
00/11	Arithmetic right shift
01	A+M and Arithmetic right shift
10	A-M and Arithmetic right shift

6. Repeat steps 5 till all bits are compared
7. Convert the result to decimal form and display
8. End

Code:

```
#include<math.h>

int acc[4]={0};
void right_shift(int a[],int q[],int*q_1)
{
  int i;
  *q_1=q[3];
  for(i=2;i>=0;i--)
  q[i+1]=q[i];

  q[0]=a[3];
  for(i=2;i>=0;i--)
  a[i+1]=a[i];
}

void swap(int a[],int b[])
{
  int i,temp;
  for(i=0;i<4;i++) //Swap 2 arrays
```

```
  {
    temp=a[i];
    a[i]=b[i];
    b[i]=temp;
  }
}

void add(int a[],int b[])
{
  int  carry[4];
  int i;
  for(i=3;i>=0;i--)
  {
    if(i==3)
    {
    carry[i]=a[i]&b[i];
    a[i]=a[i]^b[i];
    }
    else
    {

      carry[i]=(a[i]&carry[i+1])||(b[i]&carry[i+1])||(a[i]&b[i]);
      a[i]=a[i]^b[i]^carry[i+1];
    }
  }
}

void twoscomp(int a[],int n)
{
    int i;
  for(i=0;i<n;i++) //To find ones compliment
  {
    if(a[i]==1)
      a[i]=0;
    else
      a[i]=1;
  }
  int r;
  for(i=n-1;i>0;i--)
  {
    if(i==n-1)
    {
      r=a[i]&1;
      a[i]=a[i]^1;
      continue;
    }
    if(r==1)  //Only calculate if carry is one
    {
      int t=a[i]&r;
      a[i]=a[i]^r;
      r=t;
```

```
    }
  }
}

void print(int a[],int n)
{
  int i;
  for(i=0;i<n;i++)
  {
    printf("%d",a[i]);
  }
}

int main()
{
    int n1,n2,i,j;
    printf("\nEnter multiplicand and multiplier");
    scanf("%d,%d",&n1,&n2);
    int a=n1,b=n2;

    int n1b[4],n2b[4],n1c[4],n2c[4];

    for(i=4;i>=1;i--)
    {
        n1c[i-1]=n1b[i-1]=abs(a%2);
        a=a/2;
    }

      for(j=4;j>=1;j--)
      {
        n2c[j-1]=n2b[j-1]=abs(b%2);
        b=b/2;
      }
    twoscomp(n1c,4);
    printf("\nCompliments of +ve nos:");
    print(n1c,4);
    if(n1<0)
    swap(n1b,n1c);

    twoscomp(n2c,4);
    printf("\nCompliments of +ve nos:");
    print(n2c,4);
    if(n2<0)
    swap(n2b,n2c);

    printf("\nIn binary:\n");
    printf "\nn1:");
    print(n1b, 4);
    printf "\n");
    printf "\nn2:");
    print(n2b, 4);
```

```
    printf("\n\n");
    int c,t=0;
    printf("\n A    Q   Q(-1) count\n");
    for(c=4;c>=1;c--)
    {
      if(c==4){
      print(acc,4);
      printf(" ");
      print(n2b,4);
      printf("   ");
      printf("%d    %d\n",t,c);
      }
      if(n2b[3]==1&&t==0)
      {
      add(acc,n1c); //A=A-M (-M is in n1c)
      printf("\n");
      print(acc,4);
      printf(" ");
      print(n2b,4);
      printf("   ");
      printf("%d    %d\n",t,c);
      }
      else if(n2b[3]==0&&t==1)
      {
      add(acc,n1b); //A=A+M (M is in n1b)
      printf("\n");
      print(acc,4);
      printf(" ");
      print(n2b,4);
      printf("   ");
      printf("%d    %d\n",t,c);
      }
      right_shift(acc,n2b,&t); //Arithematic right shift
      printf("\n");
      print(acc,4);
      printf(" ");
      print(n2b,4);
      printf("   ");
      printf("%d    %d\n",t,c);
    }
    printf("\n");
    int fin[8];

    //Merge acc and n2b
    for(i=8,j=0;i>0;i--,j++)
    {
      if(j<4)
      fin[i-1]=n2b[3-j];
      else fin[i-1]=acc[3-j+4];
    }
```

```
    print(fin,8);
    if((n1<0&&n2>0)||(n1>0&&n2<0))//Code for shifting and then again 2's
complement if one no. is -ve
    {
    twoscomp(fin,8);
    printf("\n");
    print(fin,8);
    }
    //**Convert fin to decimal
    int sum=0;
    for(i=0;i<8;i++)
    {
      sum=sum+fin[7-i]*pow(2,i);
    }
    printf("\nEquivalent positive answer:%d",sum);
    if((n1<0&&n2>0)||(n1>0&&n2<0))
    sum=-sum;
    printf("\nFinal answer:%d",sum);
    printf("\nPerformed by Jigyassa Lamba,1811014,A2 batch");
    return 0;
}
```

Output:

```
/tmp/VKrQEAcWp2.o
Enter multiplicand and multiplier7,3
Compliments of +ve nos:1001
Compliments of +ve nos:1101
In binary:

n1:0111

n2:0011


 A    Q   Q(-1) count
0000 0011   0    4

1001 0011   0    4

1100 1001   1    4

1110 0100   1    3

0101 0100   1    2

0010 1010   0    2
```

```
00010101
Equivalent positive answer:21
Final answer:21
```

Conclusion:

Hence we are able to study and implement Booth's Multiplication Algorithm.

Post Lab Descriptive Questions

1. Explain advantages and disadvantages of Booth's algorithm.

Booth algorithm provides the **procedure of multiplication of binary integers** with 2's complement representation, hence uses of additions and subtractions would be reduced.

Advantages of booth's multiplication:

- Easy calculation of multiplication problem.
- Consecutive additions will be replaced.
- Less complex and ease scaling.

Disadvantages of booth's multiplication:

- This algorithm will not work for isolated 1's.
- It is time consuming.
- If digital gates are more, chip area would be large.

If there would be more blocks of **1's and 0's booth will work great.** It will require more calculation if there would be more multiples.

2. Is Booth's recoding better than Booth's algorithm? Justify

Booth's algorithm and the modified Booth's algorithm are both used for fast multiplication of binary numbers. The main difference between the two is in the way they handle partial products during the multiplication process.

Booth's algorithm operates by scanning the multiplier (which is usually smaller than the multiplicand) from right to left, and adding or subtracting the multiplicandto a running product depending on the value of the current multiplier bit. In other words, Booth's algorithm takes advantage of the fact that consecutive bits in the multiplier have either the same value or differ by one, allowing the multiplication to be performed with fewer partial products.

Modified Booth's algorithm is a variant of Booth's algorithm that handles partial products differently. Specifically, it uses a modified encoding scheme to reduce the number of partial products that need to be generated. In modified Booth's algorithm, three consecutive bits in the multiplier are examined at a time. If the bits are '001' or '110', the multiplicand is added to a running product. If the bits are '000' or '111', no operation is performed, and if the bits are '010' or '101', the multiplicand is subtracted from the running product.

The main advantage of modified Booth's algorithm over Booth's algorithm is that it generates fewer partial products, leading to faster multiplication. However, modified Booth's algorithm is more complex to implement than Booth's algorithm due to the additional encoding scheme. As a result, Booth's algorithm may be preferable in cases where simplicity and ease of implementation are important considerations.

Date: __10/08/23___