Batch: C-2	**Roll No.: 16010122267**
Experiment / assignment / tutorial No. 2	

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

AIM: Booth's Algorithm for Multiplication

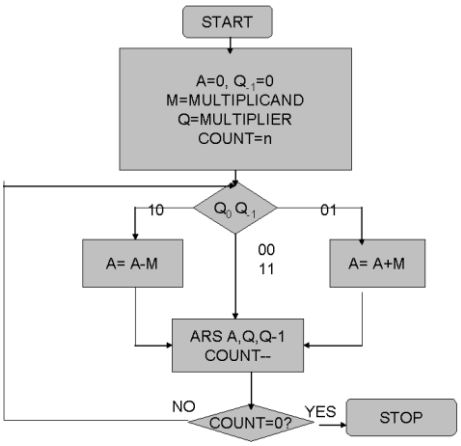
Expected OUTCOME of Experiment: (Mention CO/CO's attained here)
To understand and learn the functioning of computers by learning how it performs arithmetic operations such as multiplication.

Books/ Journals/ Websites referred:

1. Carl Hamacher, Zvonko Vranesic and Safwat Zaky, "Computer Organization", Fifth Edition, Tata McGraw-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 for Booth's algorithm in Python:

```
def binary(n,x):
    revlst=[]
    lst=[]
    for i in range(x):
        revlst.append(n%2)
        n=n//2
    for i in range(x):
        lst.append(revlst[x-1-i])
    return lst

#Function to add two binary numbers
def add(a,b):
    c=a
    for i in range(len(a)-1,-1,-1):
        if b[i]==1:
            for j in range(i,-1,-1):
```

```
                if c[j]==0:
                    c[j]+=1
                    break

                elif c[j]==1:
                    c[j]=0
                    continue
    return c

#Function to get 2s complement of binary number
def comp(arr):
    temp=arr
    #1s complement
    for i in range(len(arr)):
        if temp[i]==0:
            temp[i]=1
        elif temp[i]==1:
            temp[i]=0
    #add 1
    for i in range(len(arr)-1,-1,-1):
        if temp[i]==0:
            temp[i]+=1
            break
        elif temp[i]==1:
            temp[i]=0
            continue
    return temp

#Function to rightshift
def rightShift(arr):
    for i in range(len(arr)-1,0,-1):
        arr[i]=arr[i-1]
    return arr

# Main Code
num1=int(input("Enter 1st number: "))
num2=int(input("Enter 2nd number: "))

for i in range(0,10000):
    if num1<2**i:
        x=i
        break
```

```
for i in range(0,10000):
    if num2<2**i:
        y=i
        break
bits=x+y

M=binary(num1,bits)
Mc=comp(binary(num1,bits))

A=[0]*bits
Q=binary(num2,bits)
Q1=[0]
count=bits

R=A+Q+Q1

print("\nQ = ",Q)
print("\nA = ",A)
print("\nM = ",M)
print("\n-M = ",Mc)
print("\ncount = ",bits)
print("\n")

print("\nStart")
print(R[:bits],R[bits:2*bits],R[2*bits:],count)
for i in range(bits):

    if R[-2:]==[0,0] or R[-2:]==[1,1]:
        print("\nRight Shift")
        R=rightShift(R)
        count-=1

    elif R[-2:]==[1,0]:
        print("\nA=A-M")
        temp=R[:bits]
        temp=add(temp,Mc)
        R[:bits]=temp
        print(R[:bits],R[bits:2*bits],R[2*bits:],count)
        print("\nRight Shift")
        R=rightShift(R)
        count-=1

    elif R[-2:]==[0,1]:
        print("\nA=A+M")
```

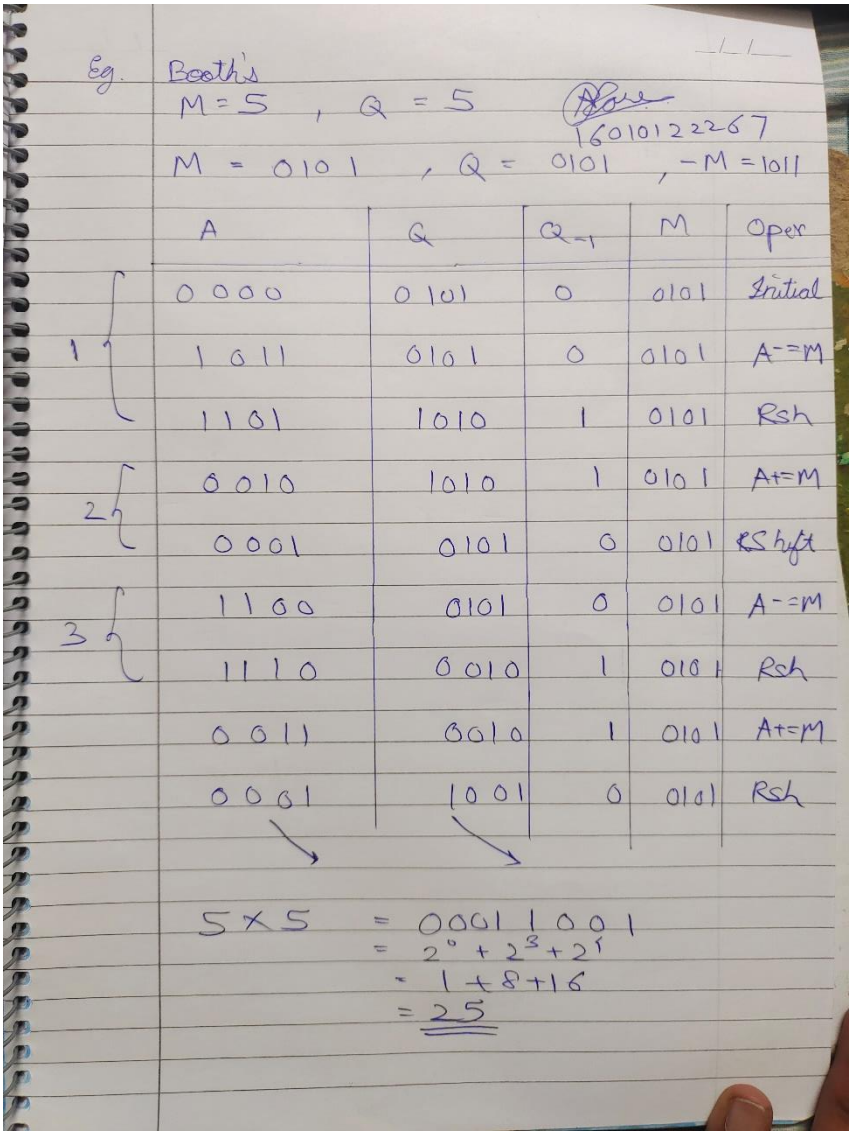
```
        temp=R[:bits]
        temp=add(temp,M)
        R[:bits]=temp
        print(R[:bits],R[bits:2*bits],R[2*bits:],count)
        print("\nRight Shift")
        R=rightShift(R)
        count-=1
    print(R[:bits],R[bits:2*bits],R[2*bits:],count)

answer=R[bits:2*bits]
print(f"\nAnswer in binary is {answer}")

sum=0
for i in range(bits):
        sum+=answer[i]*2**(bits-1-i)
print(f"\nAnswer in decimal is {sum}")
```

Output:

```
Enter 1st number: 4
Enter 2nd number: 5

Q =  [0, 0, 0, 1, 0, 1]

A =  [0, 0, 0, 0, 0, 0]

M =  [0, 0, 0, 1, 0, 0]

-M =  [1, 1, 1, 1, 0, 0]

count =  6
```

```
Start
[0, 0, 0, 0, 0, 0] [0, 0, 0, 1, 0, 1] [0] 6

A=A-M
[1, 1, 1, 1, 0, 0] [0, 0, 0, 1, 0, 1] [0] 6

Right Shift
[1, 1, 1, 1, 1, 0] [0, 0, 0, 0, 1, 0] [1] 5

A=A+M
[0, 0, 0, 0, 1, 0] [0, 0, 0, 0, 1, 0] [1] 5

Right Shift
[0, 0, 0, 0, 0, 1] [0, 0, 0, 0, 0, 1] [0] 4

A=A-M
[1, 1, 1, 1, 0, 1] [0, 0, 0, 0, 0, 1] [0] 4

Right Shift
[1, 1, 1, 1, 1, 0] [1, 0, 0, 0, 0, 0] [1] 3

A=A+M
[0, 0, 0, 0, 1, 0] [1, 0, 0, 0, 0, 0] [1] 3

Right Shift
[0, 0, 0, 0, 0, 1] [0, 1, 0, 0, 0, 0] [0] 2

Right Shift
[0, 0, 0, 0, 0, 0] [1, 0, 1, 0, 0, 0] [0] 1

Right Shift
[0, 0, 0, 0, 0, 0] [0, 1, 0, 1, 0, 0] [0] 0

Answer in binary is [0, 1, 0, 1, 0, 0]

Answer in decimal is 20
```

Example: (Handwritten solved problem needs to be uploaded)



Conclusion: Hence, in this experiment we learned about Booth's algorithm and how to multiply numbers using it. We also verified it using a code that multiplies numbers using the booth's algorithm.

Post Lab Descriptive Questions

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

Advantages of Booth's Algorithm:

1. **Reduction in Partial Products**: One of the main advantages of Booth's algorithm is its ability to reduce the number of partial products that need to be generated and added during the multiplication process. This reduction leads to fewer arithmetic operations, which can significantly improve the efficiency of multiplication, especially for larger numbers.

2. **Faster Execution**: Due to the reduction in partial products and the resulting fewer arithmetic operations, Booth's algorithm generally performs multiplication faster than conventional methods, such as the straightforward shift-and-add method.

3. **Efficient for Multiplier Implementation**: Booth's algorithm is particularly well-suited for hardware implementation in digital circuits. It can be efficiently designed using logic gates and flip-flops, making it suitable for use in hardware multipliers.

4. **Saves Hardware Resources**: The reduction in the number of partial products not only speeds up the multiplication process but also reduces the amount of hardware resources required for the multiplier implementation.

Disadvantages of Booth's Algorithm:

1. **Complexity**: Booth's algorithm involves a complex sequence of steps, including the generation of recoded values and multiple shifts and adds. This complexity can make the algorithm harder to understand, implement, and debug, compared to simpler multiplication methods.

2. **Limited Applicability**: Booth's algorithm is most beneficial for larger numbers and scenarios where efficient hardware implementation is a priority. For smaller operands, the overhead of the algorithm's complexity might outweigh the benefits gained.

3. **Not Optimized for Modern Processors**: Modern processors use various optimizations and instruction-level parallelism techniques to accelerate arithmetic

operations. Booth's algorithm doesn't align well with these optimizations, which can make it less suitable for modern high-performance processors.

4. **Increased Latency**: While Booth's algorithm can lead to fewer overall operations, it may introduce additional latency due to the need for recoding and multiple shift-and-add operations.

5. **Non-Trivial Negative Number Handling**: Handling negative numbers in Booth's algorithm requires extra care and considerations, as the recoding process can be more complex for signed numbers.

In summary, Booth's algorithm offers advantages in terms of reducing partial products and speeding up multiplication for larger numbers in hardware implementations. However, its complexity, limited applicability, and lack of alignment with modern processor optimizations can be considered disadvantages. The choice of whether to use Booth's algorithm depends on the specific requirements and constraints of the application or hardware platform.

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

   Booth's recoding is not strictly better than Booth's algorithm. They are both efficient algorithms for multiplying signed binary numbers, but they have different strengths and weaknesses.

   Booth's algorithm is simpler to implement and requires fewer hardware resources. However, it can be less efficient for multiplying numbers with long strings of zeros or ones in the multiplier.

   Booth's recoding is more complex to implement, but it can be more efficient for multiplying numbers with long strings of zeros or ones in the multiplier. This is because Booth's recoding can convert these strings into equivalent sequences of additions and subtractions, which can be performed more efficiently than shifting.

   In general, Booth's recoding is a better choice for multiplying numbers with long strings of zeros or ones in the multiplier. However, Booth's algorithm is a simpler and more efficient choice for other cases.

Date: _____