K. J. Somaiya School of Engineering, Mumbai-77
(Formerly K J Somaiya College of Engineering)
Department of Computer Engineering

Course Name:	**Computer Organization and Architecture Laboratory**	**Semester:**	**III**
Date of Performance:	**01/08/25**	**Batch No:**	**B1**
Faculty Name:	**Prof. Sheetal Pereira**	**Roll No:**	**16010124080**
Faculty Sign & Date:		**Grade/Marks:**	

Experiment No: 02

Title: Booth's Multiplication Algorithm.

Aim and Objective of the Experiment:

Aim: To study and implement **Booth's Multiplication Algorithm** for signed binary number multiplication using two's complement representation.

Objective: To understand and implement Booth's algorithm for efficient multiplication of signed binary numbers.

COs to be achieved:

CO1: Describe and define the structure of a computer with buses structure and detail working of the ALU

Books/Journals/Websites referred:

1. W. Stallings William “Computer Organization and Architecture: Designing for Performance”, Pearson Prentice Hall Publication, 7thEdition. C.

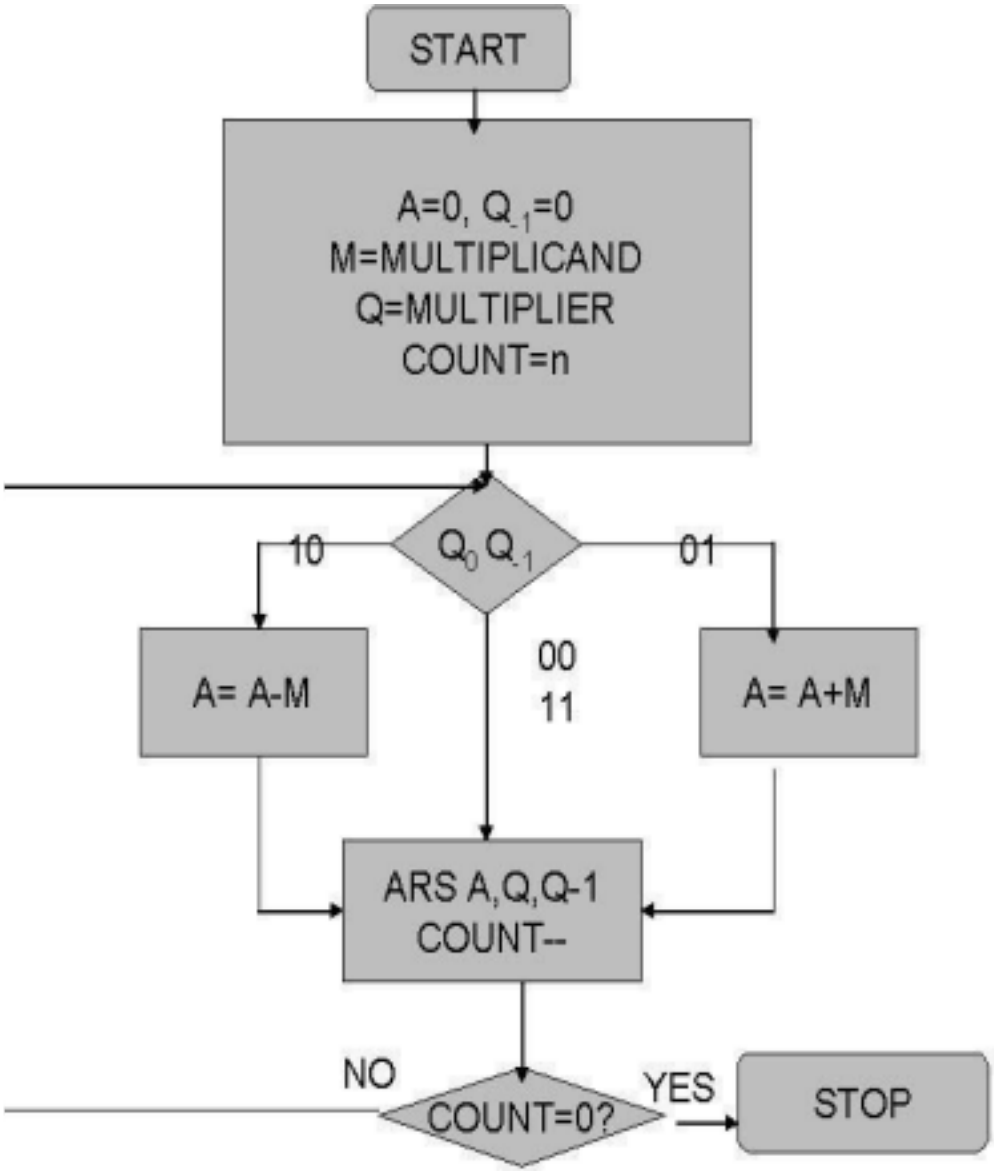
2. Carl Hamacher, Zvonko Vranesic and Safwat Zaky, “Computer Organization”, Fifth Edition, TataMcGraw-Hill.
3. Dr. M. Usha, T. S. Srikanth, “Computer System Architecture and Organization”, First Edition, Wiley-India.

Theory:

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 Q0 and Q-1 and perform the respective operation.

Q0 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:

```
def compute_bits(num1,num2):
    BIT=0
    a= max(num1,num2)
    if a<=7:
        BIT = [0,0,0,0]
    elif 8<a<=15:
        BIT=[0,0,0,0,0]
    elif 15<a<=21:
```

```
        BIT=[0,0,0,0,0,0]
    elif 21<a<=28:
        BIT = [0,0,0,0,0,0,0]
    elif 28<a<=35:
        BIT = [0,0,0,0,0,0,0,0]
    elif 35<a<42:
        BIT=[0,0,0,0,0,0,0,0,0]
    elif 42<a<=49:
        BIT=[0,0,0,0,0,0,0,0,0,0]
    else:
        print("NOT ENOUGH TO COMPUTE YOUR RESULT")
    return BIT


def to_binary(num: int,bit_l:list) -> list:
    if num == 0:
       return [0] * len(bit_l)
    i=len(bit_l)
    binary_num = bit_l.copy()


    while num > 0:
        if i!=0:
            binary_num[i-1]=(num % 2)
            num = num // 2
```

```
            i=i-1
    return binary_num


def binary_addition(b1, b2):
    carry = 0
    final = [0] * len(b1)

    for i in range(len(b1) - 1, -1, -1):
        total = b1[i] + b2[i] + carry
        final[i] = total % 2
        carry = total // 2


    return final

def to_negative(binary):
    inverted = [1 - bit for bit in binary]
    negative = binary_addition(inverted, ([0]*(len(binary)-1))
    +[1]) # Trim possible carry at front to keep same length
    if len(negative) > len(binary):
        negative = negative[1:]
    else:
        pass
```

```
    return negative


def to_set_initials():
    question = input("Enter multiplication (format m*q): ")
    nums = question.split('*')


    num1, num2 = int(nums[0]), int(nums[1])
    n1=int(nums[0])
    n2=int(nums[1])
    bits_required = compute_bits(num1,num2)
    M,M_neg,Q=0,0,0
    if num1>=0:
        M=to_binary(num1,bits_required)
        M_neg =to_negative(M)
    else:
        num1 = abs(num1)
        M_neg=to_binary(num1,bits_required)
        M = to_negative(M_neg)
    if num2>=0:
        Q=to_binary(num2,bits_required)
    else:
        num2 = abs(num2)
```

```
        x=to_binary(num2,bits_required)
        Q= to_negative(x)
    A= bits_required.copy()
    n= len(A)
    Q_0= 0
    print(f"M: {M} M_neg: {M_neg} Q: {Q} A:{A}")

    return (M,M_neg,Q,Q_0,A,n,n1,n2)




def calculation_booth(M,M_neg,Q,Q_0,A,n):
    bits=len(A)
    while n!=0:
        checking= [Q[bits-1]] + [Q_0]

        if (checking==[0,1]):
            A=binary_addition(A,M)

        elif(checking==[1,0]):
            A=binary_addition(A,M_neg)
```

K. J. Somaiya School of Engineering, Mumbai-77

(Formerly K J Somaiya College of Engineering)

Department of Computer Engineering

```
        else:
            pass


        a_first=[(A[0])]
        a_last=[(A[bits-1])]
        A.pop()
        A=a_first+A
        q_first=[(Q[0])]
        q_last=(Q[bits-1])
        Q.pop()
        Q=a_last+Q
        Q_0=q_last
        n=n-1


    final = A+Q
    return (final)


def to_decimal(n1,n2,final):
    if n1<0 or n2<0:
        final = [1 - i for i in final]
        myans=0
        final.reverse()
```

```
        t=0
        for i in final:
            myans= myans + ((2**t)*i)
            t=t+1

        myans=(myans*-1)-1

    else:
        myans=0
        final.reverse()
        t=0
        for i in final:
            myans= myans + ((2**t)*i)
            t=t+1


    return means
```

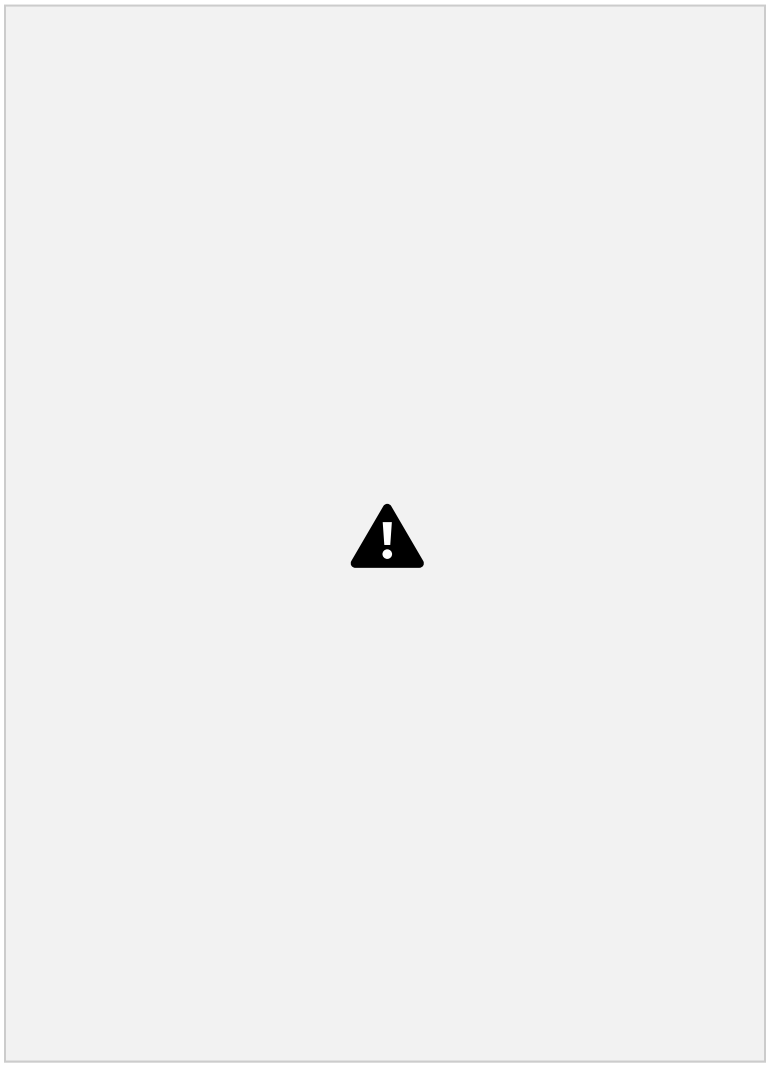
```
M,M_neg,Q,Q_0,A,n,n1,n2 = to_set_initials()

ans=calculation_booth(M,M_neg,Q,Q_0,A,n)

ans_decimal=to_decimal(n1,n2,ans)

print("Answer for your question is :", ans_decimal)
```

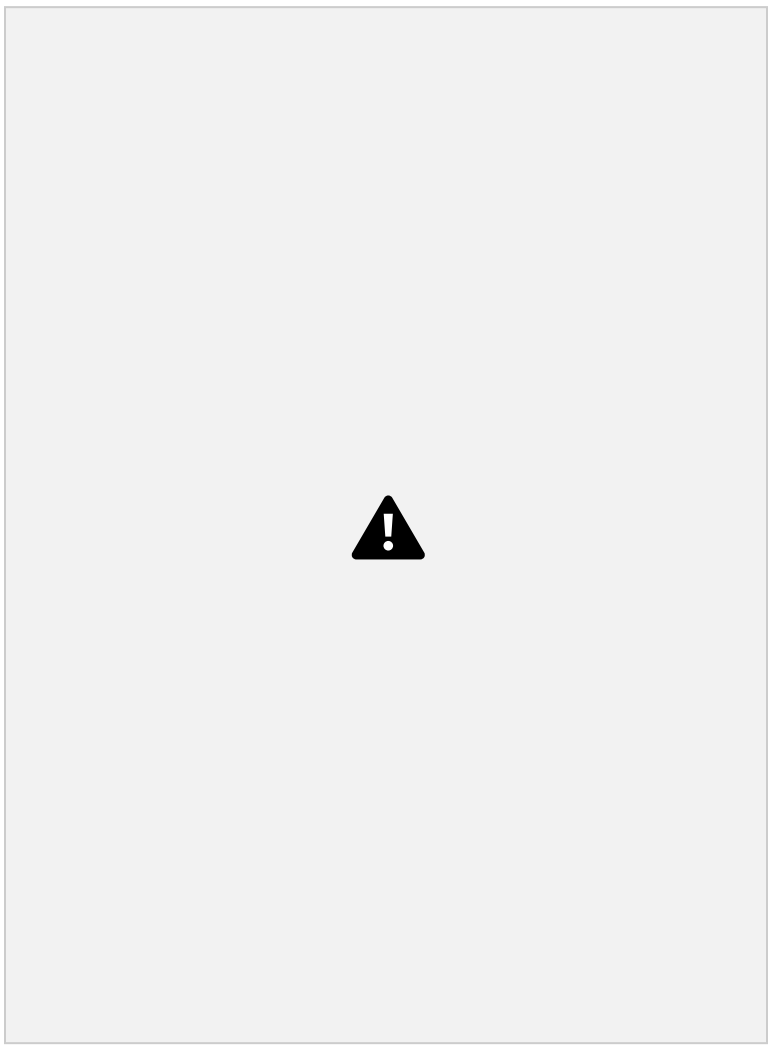
K. J. Somaiya School of Engineering, Mumbai-77

(Formerly K J Somaiya College of Engineering)

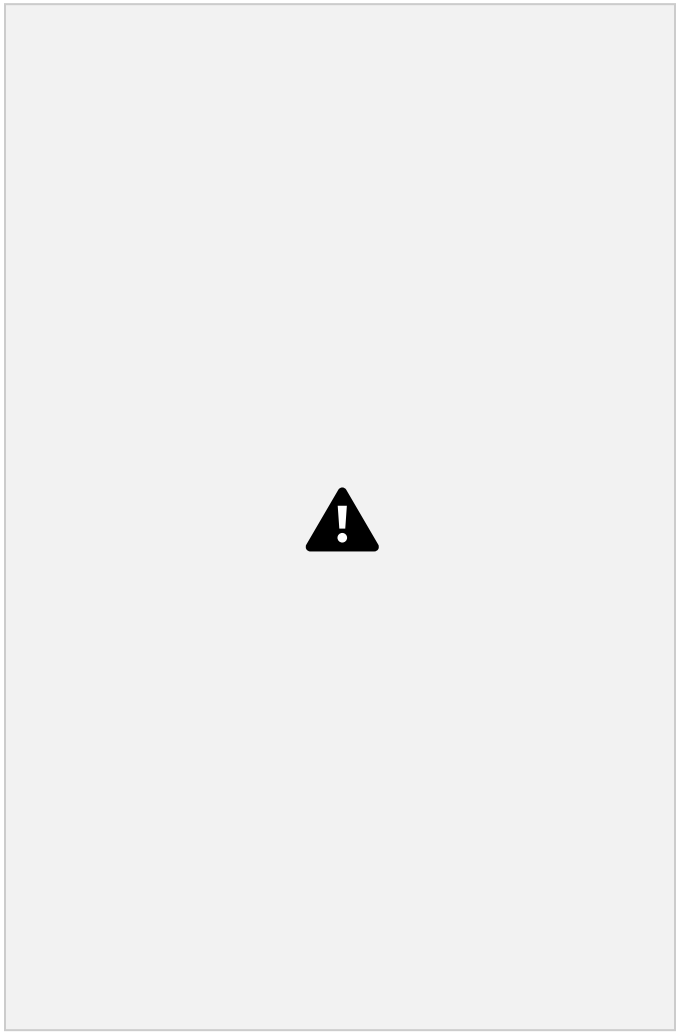
Department of Computer Engineering

Example: (Handwritten solved problem needs to be uploaded)





K. J. Somaiya School of Engineering, Mumbai-77
(Formerly K J Somaiya College of Engineering)
Department of Computer Engineering



Post Lab Subjective/Objective type Questions:

Post Lab Questions:

Q1. Explain advantages and disadvantages of Booth's algorithm. Also justify is Booth's recoding better than Booth's algorithm.

Advantages:

- Handles signed numbers directly using two's complement.
- Reduces the number of addition/subtraction by encoding runs of 1s.
- Simple hardware implementation.
- Works uniformly for positive and negative multipliers.

Disadvantages:

- Execution time varies based on multiplier bit patterns.
- Not efficient for multipliers with alternating 1s and 0s.
- More complex logic than simple shift-and-add.
- Overflow detection can be tricky.

Yes, Booth's recoding (Modified Booth's) processes two bits at a time, reducing partial products and speeding up multiplication. It is generally more efficient than the original algorithm.

Q2. How does Booth's algorithm handle multiplication of negative numbers using two's complement representation?

- Both multiplicand and multiplier are in **two's complement** form.
- Algorithm decides when to add or subtract based on multiplier bits, automatically handling signs.
- Subtraction is done by adding the two's complement, so negative numbers work

naturally. ● Final result is also in two's complement, representing the signed product

correctly.

Conclusion:

Booth's algorithm is an efficient method for multiplying signed binary numbers using two's complement representation. Implementing it in Python helps demonstrate how multiplication can be optimized by reducing additions and handling negative numbers seamlessly. Booth's algorithm, especially its modified version, offers a balance between simplicity and performance, making it useful in both software simulations and hardware designs.

Signature of faculty in-charge with Date: