# COMPUTER ORGANISATION

# Implementation of Booth's Algorithm for Multiplication

**Submitted By-**

**Bhaskar Gupta(2019237)**

# MULTIPLICATION

I used Booth's Algorithm to multiply two numbers. Firstly, the numbers are converted to their respective signed binary strings (for negative numbers, I converted them to twos complement) and then I performed booth's algorithm to find the result.

## HELPER FUNCTIONS

I also made various helper functions like right_shift(), twos_negative(), binary(), add_binary(), twos_complement() and the multiplication() which is the main function for performing multiplication.

The operations carried out by functions are as follows-

1. binary()- This function takes two integers num and n where num is the integer whose binary number of length n is to be returned but if the integer n is negative its twos_complement is to be returned.

```
'''
Input Type- Int,Int
Return Type- String
This function takes two integers num and n where num is the integer whose binary number
of length n is to be returned but if the integer n is negative its twos_complement is to be returned.
'''
def binary(num,n):
```

.

2. twos_complement- This function takes a binary string and returns a string containing its two's complement.

```
'''
Input Type- String
Return Type- String
This function takes a binary string and returns a string containing its twos complement.
'''
def twos_complement(b):
```

.

3. add_binary()- This function uses 2 strings as an argument then return the final string made using these 2 binary strings. This also neglects the extra bit in case of overflow.

```
'''
Input Type- String,String
Return Type- String
This function uses 2 strings as an argument then return the final string made using
these 2 binary strings. This also neglects the extra bit in case of overflow.
'''
def add_binary(s1,s2):
```

4. right_shift- This function performs arithmetic right shift operation on the input given string and returns the new string.

```
'''
Input Type- String
Return Type- String
This function performs arithmetic right shift operation on the input given string and returns the new string.
'''
def right_shift(s):
```

5. twos_negative()- This function takes two integers num and n where num is a negative integer whose twos_complement of length n is to be returned.

```
'''
Input Type- Int,Int
Return Type- String
This function takes two integers num and n where num is a negative integer whose twos_complement
of length n is to be returned.
'''
def twos_negative(num,n):
```

6. multiplication()- This function multiples two integers n1 and n2 using booth's algorithm for multiplication and returns the answer in its integer and binary form.

```
'''
Input Type- Int,Int
Return Type- Int, String
This function multiples two integers n1 and n2 using booth's algorithm for multiplication
and returns answer in its integer and binary form.
'''
def multiplication(n1,n2):
```

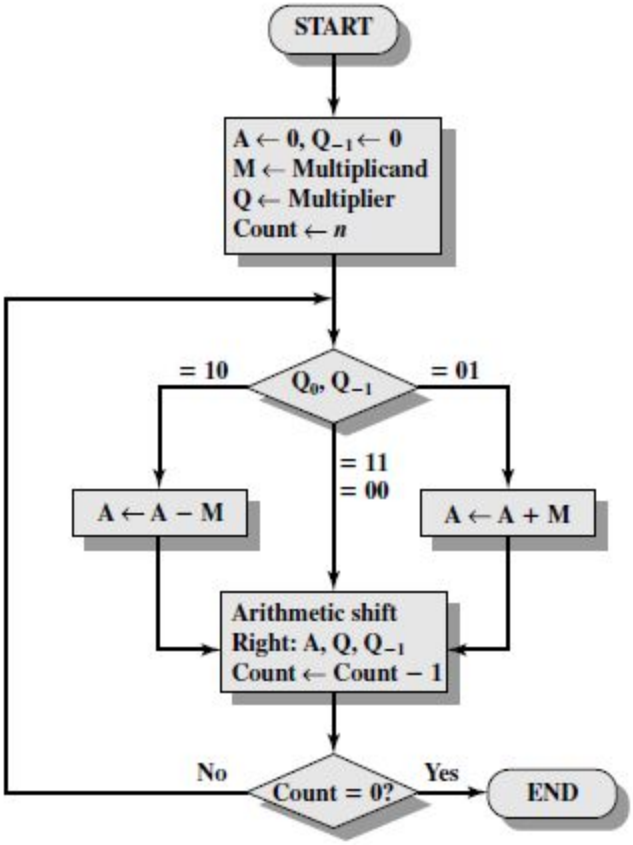
## INPUT / OUTPUT FORMAT

```
Enter two numbers -
8 9
Result of Multiplication as Integer - 72
Result of Multiplication as binary - 0000000001001000
```

```
Enter two numbers -
-10 10
Result of Multiplication as Integer - -100
Result of Multiplication as binary - 1111111110011100
```

## ALGORITHM AND ITS TIME COMPLEXITY

The algorithm I used is given below-




**Time Complexity of the Algorithm-** As evident from flowchart I loop over this flowchart for n (count) times (which is the number of bits in binary). And in every loop, I do a right shift operation which iterates over all the bits and shifts them one place to the right. So, basically in every loop (outer loop), it performs another n operations (for shifting) which makes it's time complexity $O(n_2)$. Or, in terms of decimal number m (i.e., multiplier/multiplicand) it becomes $O((\log_2 m)_2)$.

## TESTING THE PROGRAM

I have also implemented a method to test the program by taking a range for both multiplicand and the multiplier. And, then running the program for each value of multiplicand in the range with every value of the multiplier in its range and then comparing the result with its normal multiplication in Python.

```
Do you want to test the program?(Y/N)
Y
This will take time if range is large.
Enter range for multiplicand:
-500 500
Enter range for multiplier:
-500 500
```

```
---Time taken is 64.07144021987915 seconds---
Passed
```

**Result -** All Test Cases Passed. No wrong output.

**Reference Used**- https://en.wikipedia.org/wiki/Booth%27s_multiplication_algorithm