Code Documentation File

Abhinav Gudipati - 2019227

Project - *Booth Algorithm for Multiplication.*
Language - *Python*

Time Complexity
Time Complexity of the algorithm implemented is (N)^2 where N is the maximum length of the binary of the input provided.

Input Format
We are taking input of two integers between the range of -(1000) to (+1000). Keep in mind, Booth Algorithm is only applicable for integers. Therefore even if we try to put floating-point numbers, the algorithm written assumes the math.floor(a) format of the float numbers, because the program takes the floor function of the floating-point number. So for example, if we take the following integers

A = 7.7 || B = 8.9

Output = answer in Binary is : 0000111000 answer in Decimal is : + 56

In the above example, A = 7 and B = 8.

Output Format
The output is stored in outputfile.txt in both binary as well as decimal format is produced inside the outputfile.txt document.

Assumptions of Input:-

I have assumed that the user input can only take integers between the range of -1000 to 1000. With regard to the number of bits. The maximum number of bits the input format can take is 11 bits. We are assuming 10 Numerical bits and 1 Sign Bit, where the sign bit suggests a positive or negative character of the final answer. 0 indicates a positive character whereas 1 indicates negative character.

Assumptions of Output:-

With regard to the output format, I have considered the below two test cases.

1. If the product of both the integers is positive, then the output format is as follows "answer in Binary is: " + answer + " answer in Decimal is: + " + str(a)
2. If the product of both the integers is negative, then the output format is as follows "answer in Binary(2's complement) is: " + answer + " answer in Decimal is: - " + str(a)

If the given output is negative, then the function call for 2's complement is passed, whereby the negative output is again converted to 2's complement.

Dry Run of Code

I have implemented 5 functions in the given code. I have also implemented file handling in python, where the output of the given 2 inputs is shown in a separate .txt file.

Functions Implemented

1) **decimalToBinary -** Converts the given user input to binary format
2) **right_shift_operation -** implements the right shift operation for the given decimal number
3) **onescomplement -** Converts the given negative integer to one's complement, which thereafter is converted to 2's complement
4) **twosComplement -** converts the given ones complement to twos complement.
5) **binaryAddition -**
6) **Main function -** mandatory for seeking inputs for the integers.

1. First Take inputs a and b, and then convert them into binary; a_binary and b_binary
2. For negative numbers, we can store their two's complement in the corresponding binary from the function "twoscomplement".
3. With the way we generate the two's complement of the given negative integer, the negative integer is first passed through "onescomplement" function, where the answer is first stored as a string, if the for loop encounters a 0, it is changed to 1. If it is 1, it is changed to 0.
4. Make the length of both the binaries same. We are doing this in order to make sure that both of the user input binary formats are the same. For example, if a = 7

and b = -199, both must have the same number of bits while performing the booth algorithm.

5. Initialize ac with '0'. Length(ac) = length(a_binary) ; Counter = length(a_binary); q_nPlusOne = 0
6. While (Counter > 0)
    a. last = a_bin[-1]
    b. if last, q_nPlusOne == 1,0
        i. ac = ac + twos_complement(b_binary)
    c. else if last, q_nPlusOne == 1,0
        i. ac = ac + b_binary
    d. RightShift(ac, a_binary)
    e. Counter = Counter-1
7. Answer = ac + a_binary { Note : this is string addition }

Booth Algorithm Explanation.

We need to satisfy 4 conditions; Let's take the example of a 4-bit number

1) If the Q• (Read as Q not) and the Qi-1 Values are both 0 , then we perform right shift operation.
2) If the Q• (Read as Q not) and the Qi-1 Values are both 1, then we perform right shift operation.
3) If the Q• (Read as Q not) and the Qi-1 Values are 0 and 1 respectively, then we perform A + M = X and then X is stored in A, then perform right shift operation.
4) If the Q• (Read as Q not) and the Qi-1 Values are 1 and 0 respectively, then we perform A - M = X and then X is stored in A, then perform right shift operation.

If we take the example of A = 3 and M=7; We would need the answer to be printed as 21 and 10101 as its binary equivalent.

We prepare tables according and perform the above 4 operations. The tables are segregated into 3 columns, A, Q• and Qi-1. M remains the same throughout, which is the 2nd User Input in our case. A keeps changing in accordance with values being stored in the accumulator.

A is the Accumulator, which is 0000 initially for 4-bit operation.

Q• is the index number 0 position of the given binary number. In this case,the first user input a = 0011, where Q• is 1, and Qi-1 is initially 0. We thereafter perform the above operations as mentioned and get the desired answer as 00010101.

Test Cases Used and their Corresponding Outputs :

1)a = 1000 || b = -1000

answer in Binary(2's complement) is : 00111101000001001000000

answer in Decimal is : - 1000000

2) a = 7 || b = -9

answer in Binary(2's complement) is : 0000111111

answer in Decimal is : - 63

3)a = -33 || b = -68

answer in Binary is : 0000100011000100

answer in Decimal is : + 2244

4)a = 1000 || b = 1000

answer in Binary is : 00111101000001001000000

answer in Decimal is : + 1000000

Credits :

1. Professor
2. Internet sources like geeks for geeks
3. Stack exchange
4. Friends