Documentation

Project 2 - Booth's Algorithm Implementation

Course: CSE112 Computer Organization

Project Members:

Vaibhav Gupta - 2019341

Assumptions

- The algorithm is only implemented for multiplication of integers.
- There is no limit as to how large an integer can be.
- User enters the decimal ($base_{10}$) form of the integer.
- The multiplicand and multiplier can be both positive and negative.

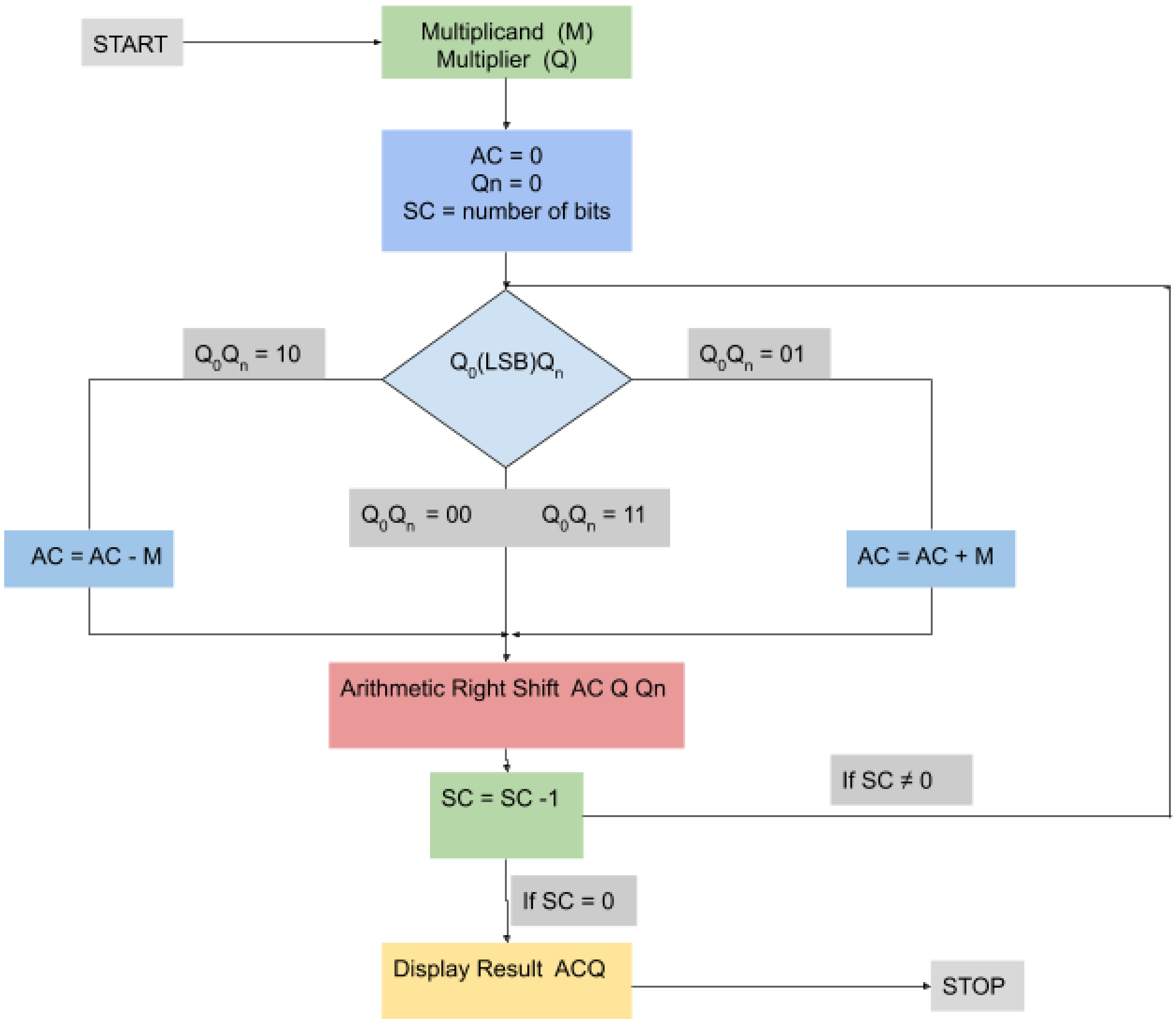
Booth's Algorithm Explanation

Booth algorithm gives a procedure for **multiplying binary integers** in signed 2's complement representation **in an efficient way**, i.e., less number of additions/subtractions required. It operates on the fact that strings of 0's in the multiplier require no addition but just shifting and a string of 1's in the multiplier from bit weight 2^k to weight 2^m can be treated as 2^(k+1) to 2^m.

As in all multiplication schemes, Booth's algorithm requires examination **of the multiplier bits** and shifting of the partial product. Prior to the shifting, the multiplicand may be added to the partial product, subtracted from the partial product, or left unchanged according to following rules:

1. The multiplicand is subtracted from the partial product upon encountering the first least significant 1 in a string of 1's in the multiplier
2. The multiplicand is added to the partial product upon encountering the first 0 (provided that there was a previous '1') in a string of 0's in the multiplier.
3. The partial product does not change when the multiplier bit is identical to the previous multiplier bit.

Booth's Algorithm Flowchart



Imported module

```
from columnar import columnar  # To present operations in form of a table
```

Columnar : A library for creating columnar output strings using data as input.

https://pypi.org/project/Columnar/

Methods & Working

```
"""
:Function Name: Flip_Bits
:Number of Parameters: 1
:Type of Parameters: string
:Return Type: string
:Function Description: flip bits of binary number
"""


def Flip_Bits(String):  # One's complement of binary number
    """
    :param String: String containing binary equivalent of a number
    :return: String with the bits of the binary number flipped
    """
```

Eg:

\>> print(Flip_bits(‘0110’))

\>> ‘1001’

```
"""
:Function Name: Equal_Bits
:Number of Parameters: 2
:Type of Parameters: string
:Return Type: string
:Function Description: returns the two entered numbers with same number of bits
"""


def Equal_Bits(num1, num2):
    """
    :param num1: String containing binary equivalent of a number
    :param num2: String containing binary equivalent of a number
    :return: Two strings with equal number of bits
    """
```

Eg:

```
>>      print(Equal_bits('1101','01'))
>>      '1101' '0001'
```

```
"""
:Function Name: Bin_Add
:Number of Parameters: 2
:Type of Parameters: string
:Return Type: string
:Function Description: Adds two binary numbers
"""


def Bin_Add(num1, num2):
    """
    :param num1: String containing binary equivalent of a number
    :param num2: String containing binary equivalent of a number
    :return: String with sum/difference of two binary numbers [a + b]/[a + (-b)]
    """
```

Eg:

```
>>      print(Bin_Add('0110','0011'))
>>      '01001'
```

```python
"""
:Function Name: Positive_Binary
:Number of Parameters: 1
:Type of Parameters: int
:Return Type: string
:Function Description: convert a positive integer to binary form with sign bit
"""


def Positive_Binary(num):
    """
    :param num: An integer (only positive)
    :return: Binary equivalent with sign bit
    """
```

Eg:
>>	print(Positive_Binary(23))
>>	‘010111’

```python
"""
:Function Name: Twos_Complement
:Number of Parameters: 1
:Type of Parameters: int
:Return Type: string
:Function Description: convert a negative (or positive) integer to its Two's complement representation
"""


def Twos_Complement(num):
    """
    :param num: An integer (can be both positive or negative)
    :return: Two's complement representation of integer with sign bit
    """
```

Eg:
>>	print(Twos_Complement(-13))
>>	‘10011’

```python
"""
:Function Name: Convert_To_Binary
:Number of Parameters: 1
:Type of Parameters: int
:Return Type: string
:Function Description: A pseudo function used to distinguish between negative and positive numbers
                       and call appropriate methods for binary conversion
"""


def Convert_To_Binary(num):
    """
    :param num: An integer
    :return: String containing binary equivalent of integer with sign bit
    """
```

```python
"""
:Function Name: Arithmetic_Right_Shift
:Number of Parameters: 0
:Type of Parameters: -
:Return Type: list
:Function Description: perform arithmetic right shift on AC , QR and Qn
"""


def Arithmetic_Right_Shift():
    """
    :return: A list of arithmetically right shifted accumulator, multiplier and Qn
    """
```

```python
"""
:Function Name: Perform_Operation
:Number of Parameters: 0
:Type of Parameters: -
:Return Type: list
:Function Description: performs appropriate operation based on 'QR(LSB)' + 'Qn'
"""


def Perform_Operation():
    """
    :return: A list containing Accumulator, multiplier and Qn after appropriate operation
    """
```

```python
"""
:Function Name: Booth_Algorithm
:Number of Parameters: 0
:Type of Parameters: -
:Return Type: -
:Function Description: performs Booth's algorithm for binary multiplication
"""


def Booth_Algorithm():
    """
    :return: Nothing
    """
```

Sample Inputs and Outputs

Eg1: Required product 6*2

```
Enter multiplicand: 6
Enter multiplier: 2

 M = 0110
 Q = 0010
-M = 1010

  AC     QR     QN     SC

  0000   0010   0      4
  0000   0001   0      3
  1101   0000   1      2
  0001   1000   0      1
  0000   1100   0      0

Final result :
AC + QR = 00001100
Product =  12
```

Eg2: Required product (-13)*(-9)

```
Enter multiplicand: -13
Enter multiplier: -9

 M = 10011
 Q = 10111
-M = 01101

  AC     QR     QN     SC

  00000  10111  0      5
  00110  11011  1      4
  00011  01101  1      3
  00001  10110  1      2
  11010  01011  0      1
  00011  10101  1      0

Final result :
AC + QR = 0001110101
Product =  117
```

Eg3: Required product 365*(-1248)

```
Enter multiplicand: 365
Enter multiplier: -1248

 M = 000101101101
 Q = 101100100000
-M = 111010010011

  AC            QR            QN     SC

  000000000000  101100100000  0      12
  000000000000  010110010000  0      11
  000000000000  001011001000  0      10
  000000000000  000101100100  0      9
  000000000000  000010110010  0      8
  000000000000  000001011001  0      7
  111101001001  100000101100  1      6
  000001011011  010000010110  0      5
  000000101101  101000001011  0      4
  111101100000  010100000101  1      3
  111110110000  001010000010  1      2
  000010001110  100101000001  0      1
  111110010000  110010100000  1      0

Final result :
AC + QR = 111110010000110010100000 = -000001101111001101100000
Product = -455520
```