# TRIBHUWAN UNIVERSITY
# INSTITUTE OF ENGINEERING
# PULCHOWK CAMPUS

## A LAB REPORT
## ON

## Booth's multiplication algorithm for signed integers

Lab No: 6
Experiments Date:
Submission Date:

**Submitted By:**
Name: Nabin Khanal
Group: B
Roll No: 076BCT036

**Submitted To:**
Department of
Electronics and computer engineering

TITLE: Booth's multiplication Algorithm.

OBJECTIVE

Booth's multiplication algorithm is a multiplication algorithm that can multiply two signed binary numbers in 2's complement notation. 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 $2^{K+1}-2^m$. For example the binary number 001110 (+14) has a string of 1's from K=3 m=1. The number can be represented as $2^{K+1}-2^m = 2^4-2^1 = 16-2 = 14$. Therefore the multiplication Mx14, where M is the multiplicand and 14 is the multiplier can be done as $M \times 2^4 - M \times 2^1$. Thus the product can be obtained by shifting the binary multiplicand M four times to the left as subtracting M shifted left once.

As in all multiplication schemes, Booth algorithm requires examination of the multiplier bits and shifting of the partial products prior to the shifting, the multiplicand may be added to the partial product, subtracted from the partial product or left unchanged according to the 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 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 previous multiplier bit.

Let Q contains multiplier, M contains multiplicand and A (accumulation) is initialized to 0. The process involves addition, subtraction & shifting.

Algorithm:
The number of steps required is equal to the number of bits in the multiplier.
At the beginning, consider an imaginary "0" beyond LSB of multiplier.

(1) At each step, examine two adjacent multiplier bits from right to left.
(2) If the transition is from 0 to 1, then subtract M from A.
(3) If the transition is from 1 to 0 then add M to A.

(4) Then simply Right shift.

Repeat steps 1 to 5 for all bits of multiplier.

CamScanner

-5 → 1011
-7 → 1001

Example: 5 × 7

| Step | A | Q | $Q_{-1}$ (imag) |
|---|---|---|---|
| Initial | 0000 | 0111 | 0 |
| 1←0 SUBM | 0000 + 1011 = 1011 | | |
| shift | 1101 | 1011 | 1 |
| 1←1 | | | |
| shift | 1110 | 1101 | 1 |
| 1←1 shift | 1111 | 0110 | 1. |
| 0←1 Add M. | 1111 + 0101 = ~~0000~~ 0100 | | |
| | ~~0001~~ | ~~1011~~ | ~~0~~ |
| | 0010 | 0011 | 0 |

∴ 5×7 = ~~00110010~~ 00100011

32 21

= 35 #

SOURCE CODE

```
from sum import add
from difference import subtract

def shift (sum, multiplier):
    last= multiplier[-1]
    multiplier= sum[-1] + multiplier[:-1]
    sum = sum[0] + sum[:-1]
    return sum, multiplier, last


def product(n1, n2, n):
    sum = "".zfill(n)
    last = "0"
    for ~~itr~~ i in range(n):
        if (n2[len(n2)-1]=='1' and last =='0'):
            sum =subtract (sum, n1,n)
        elif (n2[len(n2)-1]=='0' and last =='1'):
            sum = add(sum, n1,n)
            if (len(sum) >n):
                sum=sum[1: ]
        sum, n2, last = shift (sum, n2)
    return sum, n2

def main():
    n= int(input("Enter thenumber ofbits: "))
    n1 = input ("Enter the first number: ")
    n2 = input (" Enter the second number: ")
    n1 =n1.zfill(n)
    n2 = n2.zfill (n)

    print (product (n1, n2,n))
main()
```

Output

```
Enter the number of bits: 4
Enter the first number: 0111
Enter the second number: 0101

('0010', '0011')
```

DISCUSSION AND CONCLUSION

The program was implemented using python programming language. Booth algorithm can perform multiplication for signed numbers using the partial product approach.

In this way we implemented and executed Booth's algorithm for the multiplication of signed binary numbers