Expermiment 1

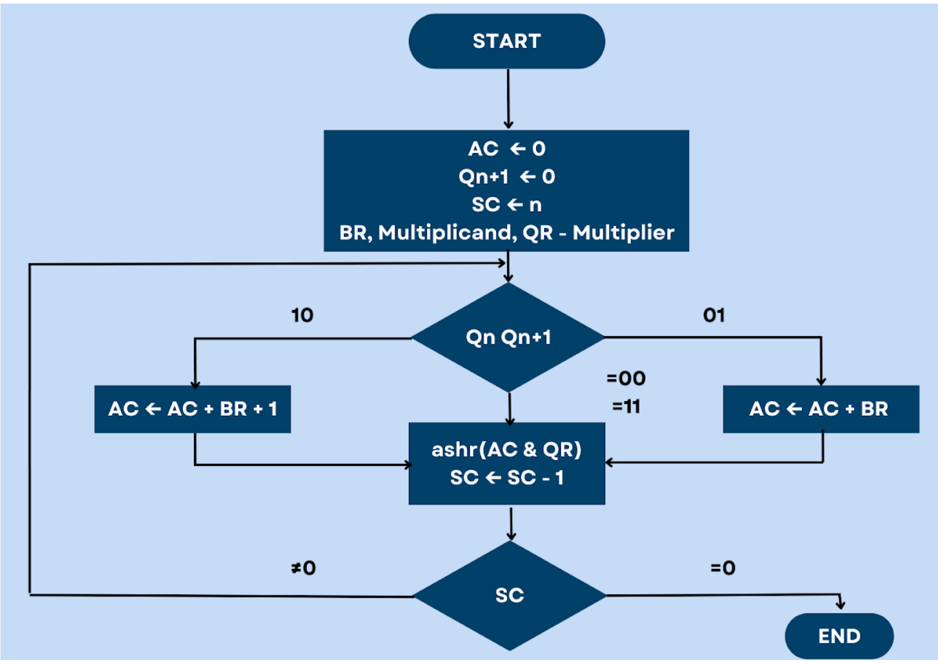
Aim : To Implement Booth’s (Signed) Algorithm & Unsigned Multiplication

Booth’s Algorithm

Theory :

Booth's algorithm is a computer algorithm used for multiplying two signed binary numbers in two's complement notation. It is particularly efficient for performing binary multiplication in hardware. The main idea behind Booth's algorithm is to reduce the number of additions and subtractions required to compute the product of two binary numbers, resulting in faster multiplication operations.

Flowchart :



Solved Problem :

Booth's Algorithm

M = 7 , Q = 3

A	Q	Q_1	Operation
0000	0011	0	A ← A − M
1001	0011	0	ARS
1100	1001	1	ARS
1110	0100	1	A ← A + M
0101	0100	1	ARS
0010	1010	0	ARS
0001	0101	0	

∴ 7 × 3 = 00010101 = 21

Code :

```
def twosComplement(num):
    onesComp=""
    for i in num:
        if i == "0":
            onesComp += "1"
        else:
            onesComp +="0"
    return bin(int(onesComp,2) + int("1",2)).replace('0b',"")

num1 = int(input('Enter number: '))
num2 = int(input('Enter 2nd number: '))
binNum1 = bin(abs(num1)).replace("0b",'')
binNum2 = bin(abs(num2)).replace("0b",'')

if len(binNum1) >= len(binNum2):
    maxlen = len(binNum1)
else:
    maxlen = len(binNum2)
maxlen +=1

binNum1 = binNum1.zfill(maxlen)
binNum2 = binNum2.zfill(maxlen)
if num2 < 0:
    binNum2 = twosComplement(binNum2)
if num1 < 0:
    binNum1 = twosComplement(binNum1)

binCompNum1 = twosComplement(binNum1)
binCompNum1 = binCompNum1.zfill(maxlen)
print(binNum1)
print(binNum2)

count = maxlen
m = binNum1
minusm = binCompNum1
q = binNum2
q1 = '0'
a = "0"
a = a.zfill(maxlen)
rightshift=""
while count > 0:
    if q1 == '1' and q[maxlen-1] == '0':
        a = bin(int(a,2) + int(m,2)).replace('0b','')
        if(len(a) > maxlen):
            a = a[1:]
        a = a.zfill(maxlen)

    elif q1=='0' and q[maxlen-1] == '1':
        a = bin(int(a,2) + int(minusm,2)).replace('0b','')
        if(len(a) > maxlen):
            a = a[1:]
        a = a.zfill(maxlen)

    merged = a+q+q1
    rightshift = merged[0]
    for i in range(len(merged)-1):
        rightshift += merged[i]
```

```
    a = rightshift[:maxlen]
    q = rightshift[maxlen:maxlen*2]
    q1 = rightshift[-1]
    count -=1

ans = a+q
minus = False
if ans[0] == '1':
    ans = twosComplement(ans)
    minus = True
print(ans)
if minus:
    print(int(ans,2) * -1)
else:
    print(int(ans,2))
```

Output :

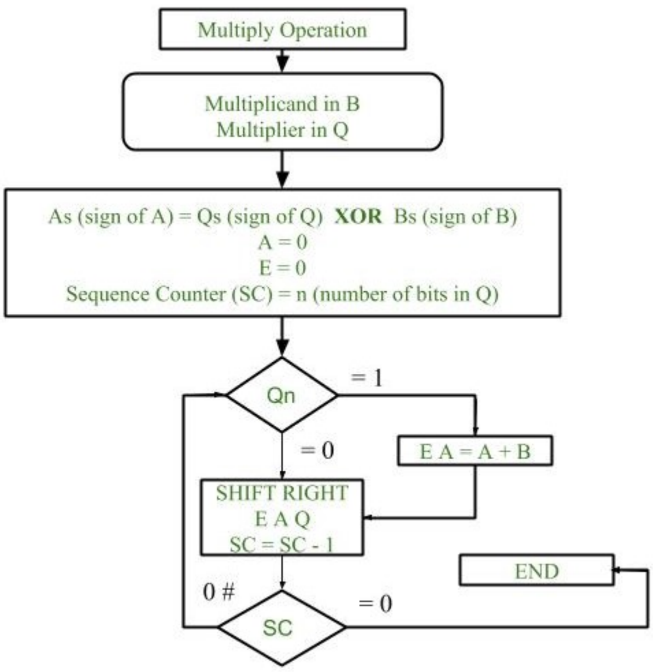
```
Enter multiplier: 7
Enter multiplicand: 4
Binary form of 7 = 0111
Binary form of 4 = 0100
Answer in binary form 00011100
Answer in decimal form 28
```

Unsigned Multiplication

Theory :

Unsigned multiplication is the process of multiplying two unsigned binary numbers. Unsigned binary numbers are non-negative numbers that are represented using only 0's and 1's.Unsigned multiplication is performed using a variety of algorithms.The shift-and-add algorithm works by multiplying each bit of the multiplier with the multiplicand, and then shifting the partial products to the left. The partial products are then added together to produce the final product.

Flowchart :



Solved Problem :

Unsigned Multiplication

13x11

C	A	Q	M	Operation
0	0000	1101	1011	C,A ← A+M
0	1011	1101	1011	RS
0	0101	1110	1011	RS
0	0010	1111	1011	C,A ← A+M
0	1101	1111	1011	RS
0	0110	1111	1011	C,A ← A-M
1	0001	1111	1011	RS
0	1000	1111	1011	

13x11 = 1000 1111 = 143

Code :

```
def binary(a, b):
    a1 = abs(a)
    b1 = abs(b)
    com = [1, 0, 0, 0, 0, 0, 0, 0]
    anum = [0] * 8
    anumcp = [0] * 8
    bnum = [0] * 8
    acomp = [0] * 8
    bcomp = [0] * 8
    pro = [0] * 8
    res = [0] * 8
    for i in range(8):
        r = a1 % 2
        a1 = a1 // 2
        r2 = b1 % 2
        b1 = b1 // 2
        anum[i] = r
        anumcp[i] = r
        bnum[i] = r2
        if r2 == 0:
            bcomp[i] = 1
        if r == 0:
            acomp[i] = 1
    c = 0
    for i in range(8):
        res[i] = com[i] + bcomp[i] + c
        if res[i] >= 2:
            c = 1
        else:
            c = 0
        res[i] = res[i] % 2
```

```
        bcomp[i] = res[i]
    if a < 0:
        c = 0
        for i in range(8):
            res[i] = 0
        for i in range(8):
            res[i] = com[i] + acomp[i] + c
            if res[i] >= 2:
                c = 1
            else:
                c = 0
            res[i] = res[i] % 2
            anum[i] = res[i]
            anumcp[i] = res[i]
    if b < 0:
        for i in range(8):
            temp = bnum[i]
            bnum[i] = bcomp[i]
            bcomp[i] = temp
    return anum, bnum, bcomp, pro, anumcp
def add(num, pro, anumcp):
    res = [0] * 8
    c = 0
    for i in range(8):
        res[i] = pro[i] + num[i] + c
        if res[i] >= 2:
            c = 1
        else:
            c = 0
        res[i] = res[i] % 2
        pro[i] = res[i]
    return pro, anumcp
def arshift(pro, anumcp):
    temp = pro[7]
    temp2 = pro[0]
    for i in range(1, 8):
        pro[i - 1] = pro[i]
    pro[7] = temp
    for i in range(1, 8):
        anumcp[i - 1] = anumcp[i]
    anumcp[7] = temp2

    return pro, anumcp
def booth_multiplication(a, b):
    anum, bnum, bcomp, pro, anumcp = binary(a, b)
    q = 0
    result = []
    for i in range(8):
        if anum[i] == q:
            result.append(pro)
            pro, anumcp = arshift(pro, anumcp)
            q = anum[i]
        elif anum[i] == 1 and q == 0:
            result.append(pro)
            pro, anumcp = add(bcomp, pro, anumcp)
            pro, anumcp = arshift(pro, anumcp)
            q = anum[i]
        else:
            result.append(pro)
            pro, anumcp = add(bnum, pro, anumcp)
```

```
            pro, anumcp = arshift(pro, anumcp)
            q = anum[i]
    final_product = [0] * 16
    for i in range(8):
        final_product[i] = anumcp[i]
    for i in range(8, 16):
        final_product[i] = result[-1][i - 8]
    return final_product
if __name__ == "__main__":
    a = int(input("Enter A: "))
    b = int(input("Enter B: "))
    if abs(a) > 255 or abs(b) > 255:
        print("Both numbers must be integers in the range
             (-256 to 255).")
    else:
        result = booth_multiplication(a, b)
        print("\nProduct is =", end=" ")
        for i in reversed(result):
            print(i, end="")
        print()
```

Output :

Enter A: 3
Enter B: 2
Product is = 0000000000000110

Conclusion : The best algorithm to use for signed or unsigned multiplication depends on the specific requirements of the application. If speed and efficiency are the most important factors, then the Booth algorithm is the best choice. However, if simplicity and ease of implementation are more important, then the shift-and-add algorithm is a good choice.