Expermiment 2

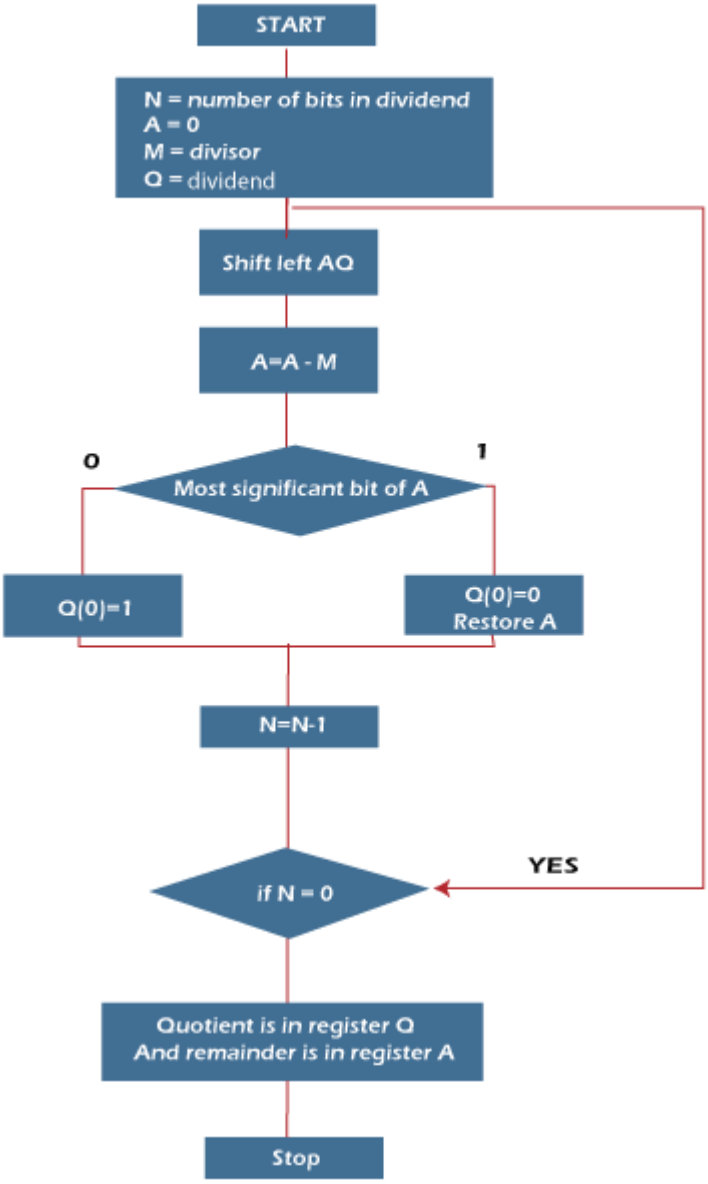
Aim : To Implement Restoring (Signed) & Non-Restoring (Unsigned) Algorithm Division

Restoring (Signed)

Theory :

Restoring division is a signed division algorithm that works by repeatedly subtracting the divisor from the dividend until the remainder is less than the divisor. The quotient is then calculated by counting the number of subtractions performed.

Flowchart :



Solved Problem :

Restoring Division Algorithm

Dividend = 11 (1011)
Divisor = 3 (0011)
A = 0 times 'N+1'
N = 4 {max count (dividend, divisor)}

N	M	A	Q	Operation
4	00011	00000	1011	Initialize
	00011	00001	011_	Shift left AQ
	00011	11110	011_	A ← A-M (A 00001 + (-M) 11101)
	00011	00001	0110	Q_0 ← 0 & restore A
3	00011	00010	110_	Shift left
	00011	11111	110_	A ← A-M (A 00010 + (-M) 11101 = 11111)
	00011	00010	1100	Q_0 ← 1 & restore A
2	00011	00101	100_	Shift left A & Q
	00011	00010	100_	A ← A+(-M) (00101 + 11101 = 00010)
	00011	00010	1001	Q_0 ← 1
1	00011	00101	001_	Shift left A & Q
	00011	00010	001_	A ← A+(-M)
	00011	00010	0011	Q_0 ← 1
		↓	↓	
		Remainder = 2	Quotient = 3	

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 dividend : '))
num2 = int(input('Enter divisor : '))
```

```
binNum1 = bin(abs(num1)).replace("0b","")
binNum2 = bin(abs(num2)).replace("0b","")

maxlen = len(binNum1)

binNum1 = binNum1.zfill(maxlen)
binNum2 = binNum2.zfill(maxlen + 1)

binCompNum2 = twosComplement(binNum2)
binCompNum2 = binCompNum2.zfill(maxlen)

count = maxlen
m = binNum2
minusm = binCompNum2
q = binNum1
a = "0"
a = a.zfill(maxlen+1)
leftshift=""

while count > 0:
    merged = a+q
    leftshift = merged[1:]
    a = leftshift[:maxlen+1]
    a = bin(int(a,2)+int(minusm,2)).replace("0b","")
    if len(a) > maxlen+1:
        a=a[1:]
    a = a.zfill(maxlen+1)

    if a[0] == "0":
        leftshift = a+q[1:]
        leftshift += "1"
    else:
        a = bin(int(a,2)+int(m,2)).replace("0b","")
        if len(a) > maxlen+1:
            a=a[1:]
        a = a.zfill(maxlen+1)
        leftshift = a+q[1:]
        leftshift += "0"

    a = leftshift[:maxlen+1]
    q = leftshift[maxlen+1:]
    count -=1

if a[0] == "1":

    a = bin(int(a,2)+int(m,2)).replace("0b","")
    if len(a) > maxlen+1:
        a = a[1:]

print("Quotient : ",int(q,2))
print("Remainder : ",int(a,2))
```

Output :

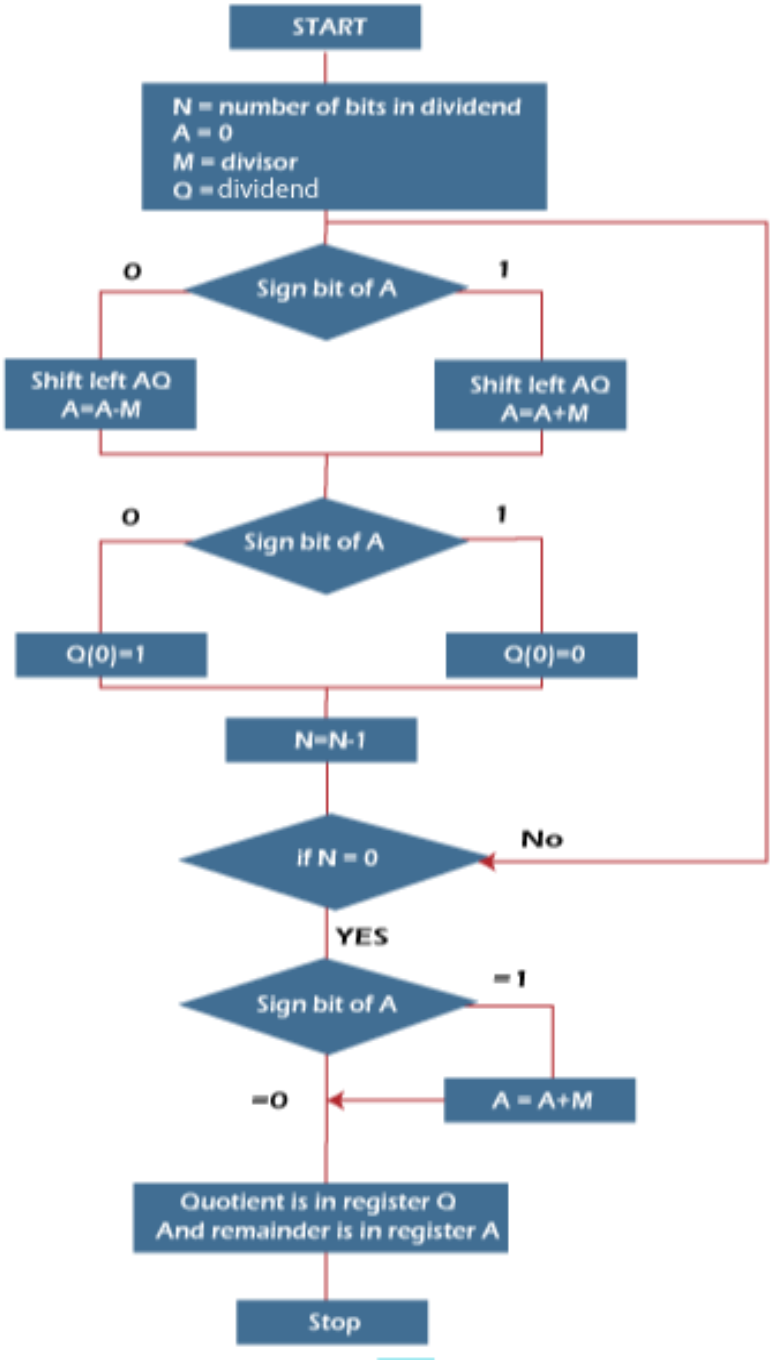
```
Enter dividend : 6
Enter divisor : 4
Quotient :  1
Remainder :  2
```

Non Restoring Division

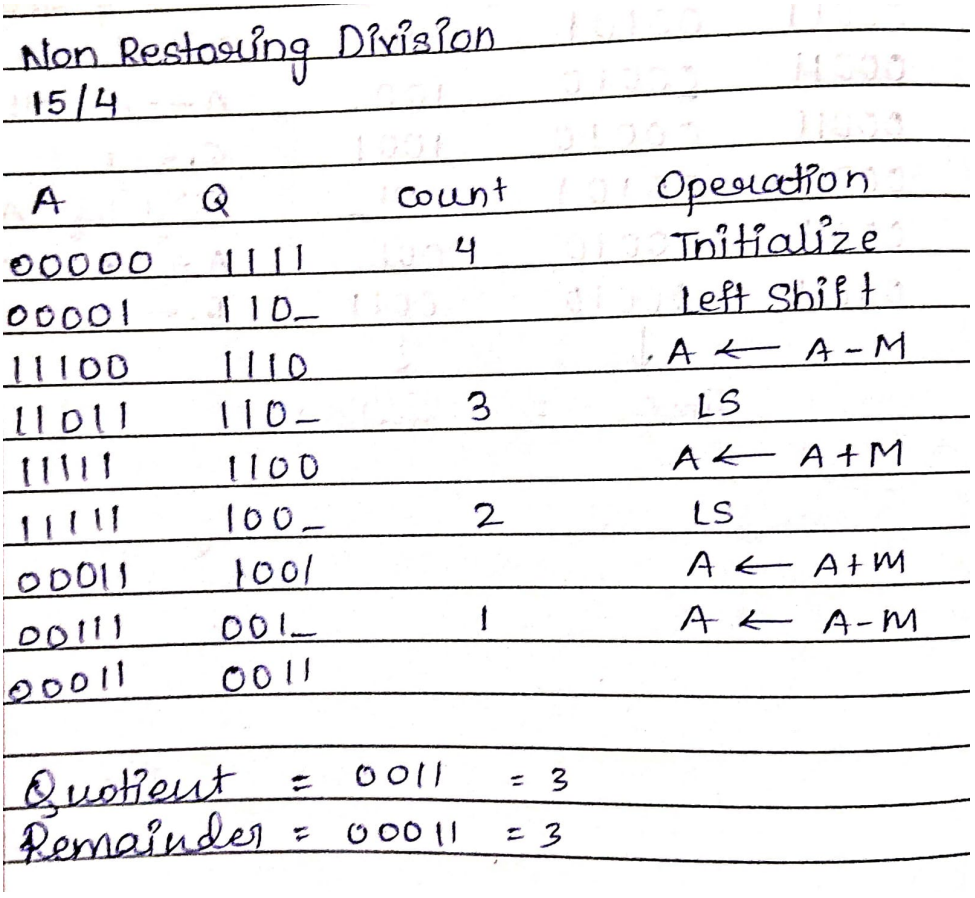
Theory :

Non-Restoring Division is a computer division algorithm that computes quotient and remainder without restoring partial remainders. It begins by shifting the dividend leftward and comparing it to the divisor, marking '0' for subtraction success and '1' for failure (negative result). If a '1' is marked, the dividend is shifted right and the divisor is complemented before subtracting. This process continues for each bit until the dividend is fully shifted. The marked bits collectively form the quotient, and the final result is obtained as the remainder. Non-restoring division is computationally efficient but complex, making it suitable for hardware-based division units, where its speed and simplicity are advantageous.

Flowchart :



Solved Problem :



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 dividend : '))
num2 = int(input('Enter divisor : '))

binNum1 = bin(abs(num1)).replace("0b","")
binNum2 = bin(abs(num2)).replace("0b","")

maxlen = len(binNum1)

binNum1 = binNum1.zfill(maxlen)
binNum2 = binNum2.zfill(maxlen + 1)

binCompNum2 = twosComplement(binNum2)
binCompNum2 = binCompNum2.zfill(maxlen)

count = maxlen
m = binNum2
```

```
minusm = binCompNum2
q = binNum1
a = "0"
a = a.zfill(maxlen+1)
leftshift=""

while count > 0:
    merged = a+q
    leftshift = merged[1:]
    a = leftshift[:maxlen+1]

    if a[0] == "1":
        a = bin(int(a,2)+int(m,2)).replace("0b","")
        if len(a) > maxlen+1:
            a=a[1:]
        a = a.zfill(maxlen+1)
    else:
        a = bin(int(a,2)+int(minusm,2)).replace("0b","")
        if len(a) > maxlen+1:
            a=a[1:]
        a = a.zfill(maxlen+1)


    leftshift = a+q[1:]

    if a[0] == "1":
        leftshift += "0"
    else:
        leftshift +="1"

    a = leftshift[:maxlen+1]
    q = leftshift[maxlen+1:]
    count -=1


if a[0] == "1":

    a = bin(int(a,2)+int(m,2)).replace("0b","")
    if len(a) > maxlen+1:
        a = a[1:]

print("Quotient : ",int(q,2))
print("Remainder : ",int(a,2))
```

Output :

```
Enter dividend : 9
Enter divisor : 2
Quotient :  4
Remainder :  1
```

Conclusion :
Restoring and non-restoring division are two different methods for performing division on binary numbers. Restoring division is a more traditional method that is simpler to understand and implement, but it is slower than non-restoring division. Non-restoring division is a more recent method that is faster than restoring division, but it is more complex to understand and implement.
The best method to use for division depends on the specific requirements of the application. If speed and efficiency are the most important factors, then non-restoring division is the best choice. However, if simplicity and ease of implementation are more important, then restoring division is a good choice.