Batch: C2	**Roll No.: 16010122267**
Experiment / assignment / tutorial No.3	

TITLE : To study and implement Restoring method of division

AIM : The basis of algorithm is based on paper and pencil approach and the operation involves repetitive shifting with addition and subtraction. So the main aim is to depict the usual process in the form of an algorithm.

Expected OUTCOME of Experiment: (Mention CO /CO's attained here)

Books/ Journals/ Websites referred:

1. Carl Hamacher, Zvonko Vranesic and Safwat Zaky, "Computer Organization", Fifth Edition, TataMcGraw-Hill.

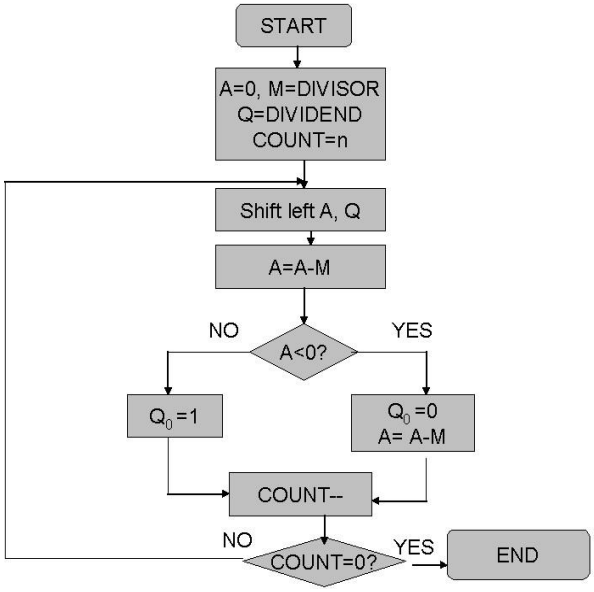
2. William Stallings, "Computer Organization and Architecture: Designing for Performance", Eighth Edition, Pearson.

3. Dr. M. Usha, T. S. Srikanth, "Computer System Architecture and Organization", First Edition, Wiley-India.

Pre Lab/ Prior Concepts:

The Restoring algorithm works with any combination of positive and negative numbers

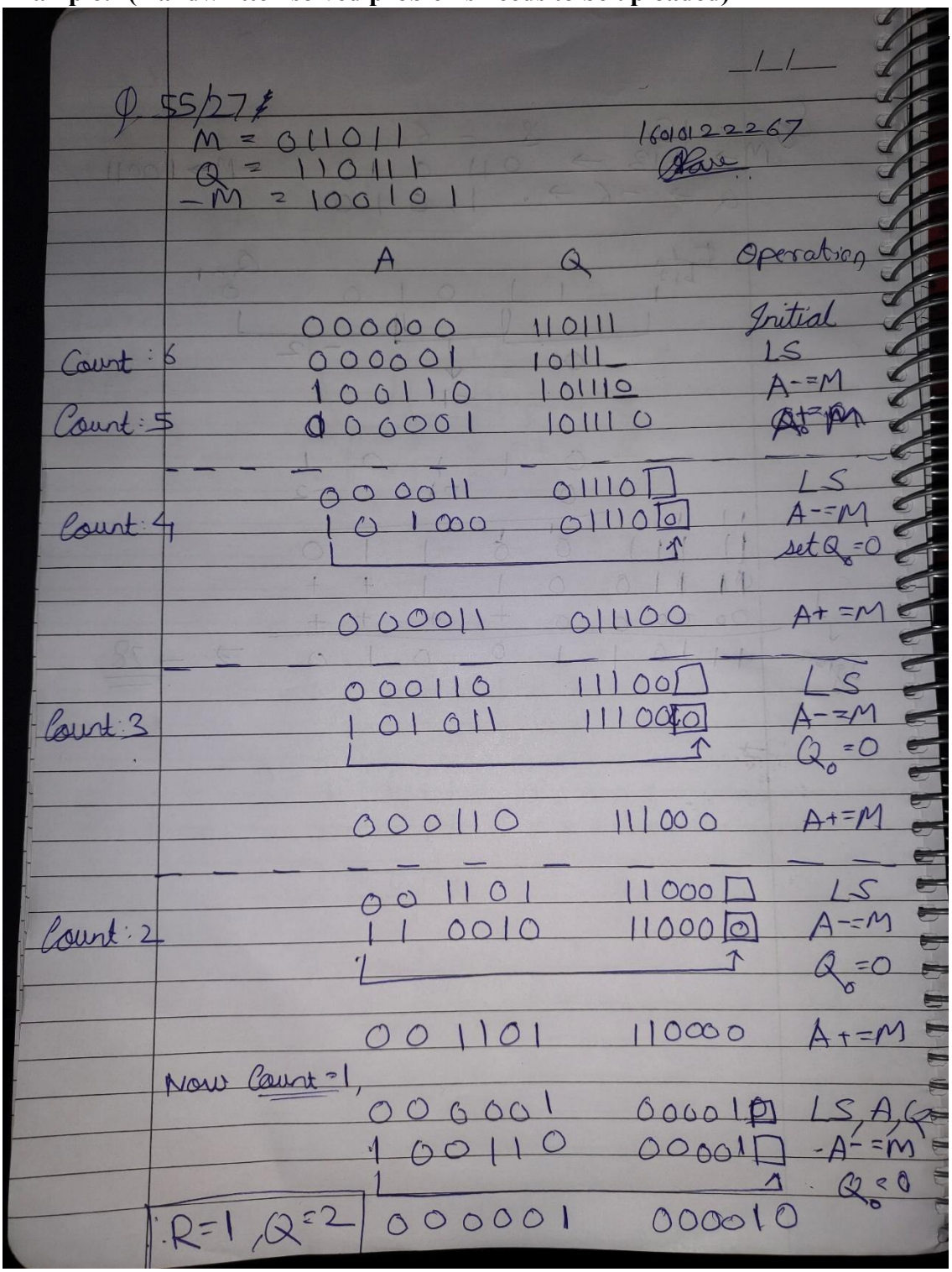
Flowchart for Restoring of Division:



Design Steps:

1. Start
2. Initialize A=0, M=Divisor, Q=Dividend and count=n (no of bits)
3. Left shift A, Q
4. If MSB of A and M are same
5. Then A=A-M
6. Else A=A+M
7. If MSB of previous A and present A are same
8. Q_0=0 & store present A 9. Else Q_0=0 & restore previous A
10. Decrement count.
11. If count=0 go to 11
12. Else go to 3
13. STOP

Example:- (Handwritten solved problems needs to be uploaded)



Implementation Details:

```
#DIVISION RESTORING ALGORITHM

#CONVERTS DECIMAL TO BINARY
def binary(a):
    bits_list=[]

    while(a>0):
        num=a
        b=int(num%2)
        bits_list.append(b)
        num=num//2
        a=int(a/2)
    #print(bits_list)
    bits_list.reverse()
    return bits_list;

#LEFT SHIFT OPERATION
def Shift(shiftA,shiftQ):

    val=shiftA+shiftQ
    l = len(val)
    i=0

    for i in range(0,l-1):
        val[i]=0
        val[i]=val[i+1]
    del val[i]
    return val

#TAKES 2's COMPLIMENT (REQD FOR "-M")
def compliment(value):
    onecomp = []
    twocomp = []
    for i in range(0,len(value)):
        if value[i]==0:
            onecomp.append(1)
        elif value[i]==1:
            onecomp.append(0)

    carry=1
    for j in range(len(value)-1,-1,-1):
```

```
        if(onecomp[j]==0 and carry==1):
            twocomp.append(1)
            carry=0
        elif(onecomp[j]==1 and carry==1):
            twocomp.append(0)
            carry=1
        elif(onecomp[j]==0 and carry==0):
            twocomp.append(0)
            carry=0
        elif(onecomp[j]==1 and carry==0):
            twocomp.append(1)
            carry=0

    twocomp.reverse()
    return twocomp

#ADDING TWO BINARY NOS.
def Add(valA,valM):
    add = []
    ad = valA
    carry = 0
    for i in range(len(ad)-1,-1,-1):
        if(valA[i]==0 and valM[i]==0 and carry==0):
            add.append(0)
            carry=0
        elif(valA[i]==0 and valM[i]==0 and carry==1):
            add.append(1)
            carry=0
        elif(valA[i]==0 and valM[i]==1 and carry==0):
            add.append(1)
            carry=0
        elif(valA[i]==0 and valM[i]==1 and carry==1):
            add.append(0)
            carry=1
        elif(valA[i]==1 and valM[i]==0 and carry==0):
            add.append(1)
            carry=0
        elif(valA[i]==1 and valM[i]==0 and carry==1):
            add.append(0)
            carry=1
        elif(valA[i]==1 and valM[i]==1 and carry==0):
            add.append(0)
            carry=1
        elif(valA[i]==1 and valM[i]==1 and carry==1):
```

```
            add.append(1)
            carry=1
    add.reverse()
    return add

#CONVERTS BINARY TO DECIMAL
def decimal(bin):
    bin.reverse()
    dec=0
    for i in range(0,len(bin)):
        if(bin[i]==1):
            dec=dec+(bin[i]*(2**i))
        elif(bin[i]==0):
            pass
    bin.reverse()
    return dec



print("      DIVISION RESTORING ALGORITHM     ")
print("")
dividend=int(input("Enter value of Dividend -> Q : "))
divisor=int(input("Enter value of Divisor -> M : "))

if (divisor == dividend):
    print("Quotient:  1 ")
    print("Remainder: 0")
elif (divisor == 0):
    print("Infinity!")
elif (dividend < divisor):
    print("Quotient: 0 ")
    print("Remainder: " + str(dividend))
else:
    q=binary(dividend)
    m=binary(divisor)

    print("Q : ",*q)

    #SETTING THE M VALUE : i.e. len(M) should be 1 more than len(Q)
    if len(m) < len(q):
        diff=len(q)-len(m)
        for i in range(0,diff+1):
            m.insert(0,0)
    print("M : ",*m)
```

```
    #ASSIGNING VALUE OF A to 0
    ACC = []
    for i in range(0,len(m)):
        ACC.append(0)
    print("A : ",*ACC)

    #VALUE OF -M
    negM = compliment(m)
    print("-M : ",*negM)
    print("")


    print("          "," |  "," A  "," |  "," Q  "," |
   ")
    print("-----------------------------------------------------------------
---------------------------------")
    n=1
    #No of iterations
    counter = len(q)
    while counter > 0:


        #LEFT SHIFT A, Q
        a = Shift(ACC,q)


        #TAKING THE "A" AND "Q" PART FROM THE "a"
        newA=a[0:len(ACC)]
        newQ=a[len(ACC):]

        #A<-A-M
        sumAM = Add(newA,negM)


        b=len(newQ)+1
        if(sumAM[0]==1):#MSB(A)=1
            newQ.insert(b,0)
            sumAM=Add(sumAM,m)
        elif(sumAM[0]==0):#MSB(A)=0
            newQ.insert(b,1)
```

```
        ACC=sumAM
        q=newQ

        print("Step : ",n,"    |    ",*ACC,"    |    ",*q,"  |  ")

        print("-------------------------------------------------")
        n=n+1
        counter=counter-1


        print("Quotient: ",*q,"  ->  ",decimal(q))
        print("Remainder: ",*ACC,"  ->  ",decimal(ACC))
```

Output:

Test Case 1:

```
Enter value of Dividend -> Q : 21
Enter value of Divisor -> M : 11
              |      A        |       Q       |
-------------------------------------------------
Step :  1     |     0 0 0 0 0 1     |     0 1 0 1 0
-------------------------------------------------
Step :  2     |     0 0 0 0 1 0     |     1 0 1 0 0
-------------------------------------------------
Step :  3     |     0 0 0 1 0 1     |     0 1 0 0 0
-------------------------------------------------
Step :  4     |     0 0 1 0 1 0     |     1 0 0 0 0
-------------------------------------------------
Step :  5     |     0 0 1 0 1 0     |     0 0 0 0 1
-------------------------------------------------
Quotient:  0 0 0 0 1   ->   1
Remainder:  0 0 1 0 1 0   ->   10
```

Test Case 2:

```
Enter value of Dividend -> Q : 10
Enter value of Divisor -> M : 10
Quotient:  1
Remainder: 0
```

Test Case 3:

```
Enter value of Dividend -> Q : 7
Enter value of Divisor -> M : 0
Infinity!
```

Test Case 4:

```
Enter value of Dividend -> Q : 14
Enter value of Divisor -> M : 20
Quotient: 0
Remainder: 14
```

Conclusion

In conclusion, we understood and implemented restoring division in Python.

Post Lab Descriptive Questions

1. **What are the advantages of restoring division over non restoring division?**

Ans:

1. Enhanced Error Recovery: Restoring division boasts superior error recovery capabilities. If an error or an incorrect quotient estimate occurs during the division process, restoring division can more effectively rectify the error and

resume the division operation accurately. This advantage stems from the fact that restoring division continually adjusts the remainder towards the correct value.

2. Streamlined Hardware Implementation: Restoring division tends to offer a simpler hardware implementation due to its iterative nature. The hardware design for restoring division is generally more straightforward, making it an attractive option for situations where minimizing hardware complexity is a priority.

3. Versatility Across Radices: Restoring division is adaptable to division in various radix systems, including binary, decimal, and others, with relatively minor modifications. This adaptability makes it a versatile choice for different number representations.

4. Incremental Quotient Computation: Restoring division calculates the quotient incrementally, meaning that at each step, it adds a partial quotient bit to the final quotient. This feature can be advantageous in applications where obtaining the quotient progressively is beneficial, rather than waiting for the entire division to finish.

Date: _____