**Batch: C-2 Roll No.: 16010122267**

**Experiment / assignment / tutorial No. 4**

|  |
| --- |
| **TITLE :** To study and implement Non Restoring method of division |

**AIM :** The basis of algorithm is based on paper and pencil approach and the operation involve 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)**

CO1-Describe and define the structure of a computer with buses structure and detail

working of the arithmetic logic unit and its sub modules.

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Books/ Journals/ Websites referred:**

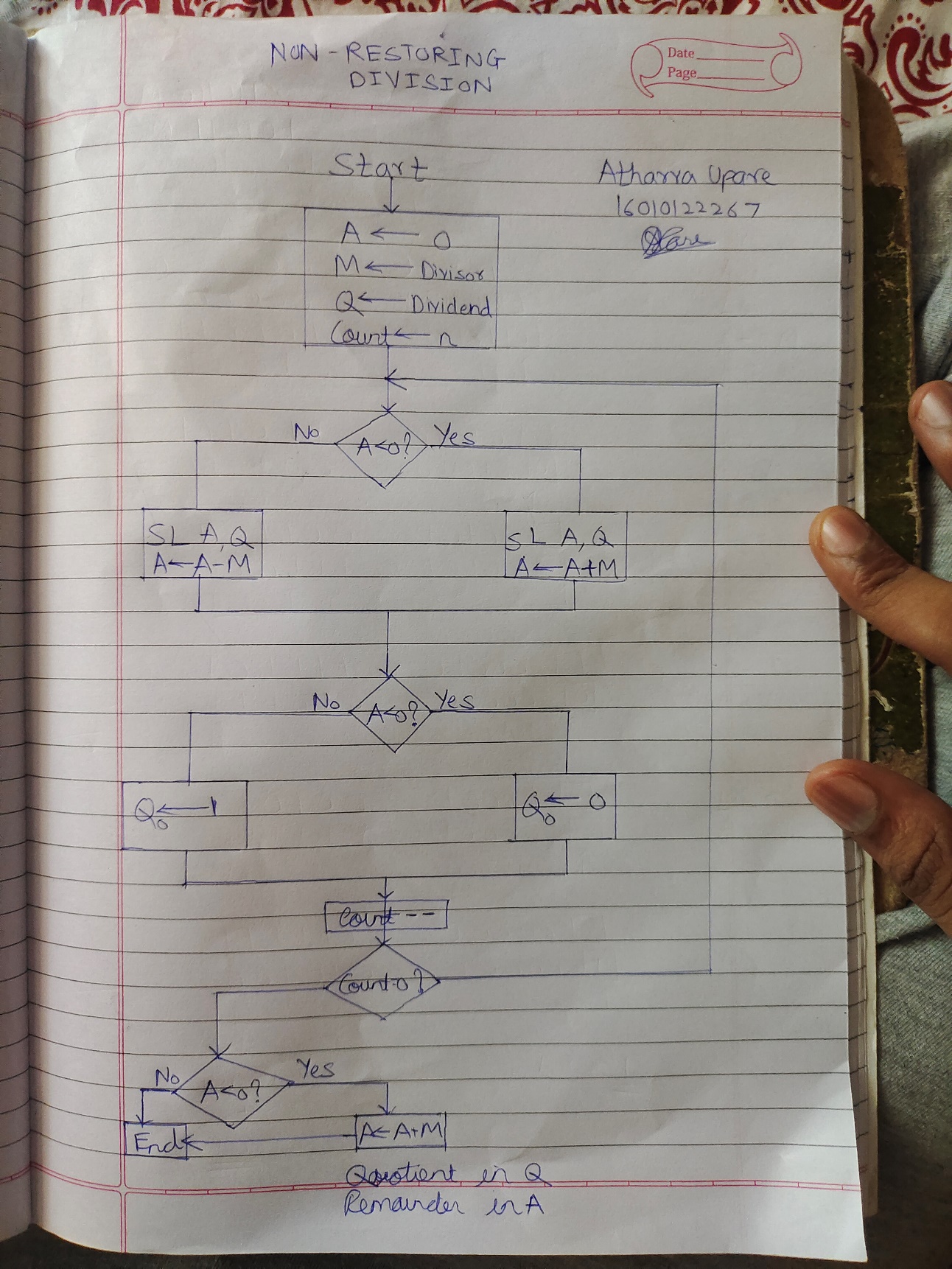
1. Carl Hamacher, Zvonko Vranesic and Safwat Zaky, “Computer Organization”, Fifth Edition, TataMcGraw-Hill.
2. William Stallings, “Computer Organization and Architecture: Designing for Performance”, Eighth Edition, Pearson.
3. Dr. M. Usha, T. S. Srikanth, “Computer System Architecture and Organization”, First Edition, Wiley-India.

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Pre Lab/ Prior Concepts:**

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

**Flowchart for Non Restoring of Division( Students need to draw)**



**Code for Non Restoring Division in Python:**

*#NON RESOTORING DIVIISION ALGORITHM*

def deci\_to\_bin(*n*):

*return* bin(*n*).replace("0b", "")

def shift\_left(*C*,*AC*,*Q*):

*C*=*AC*[0]

    temp\_AC=list(*AC*)

*for* i *in* range(1,len(*AC*)):

        temp\_AC[i-1]=temp\_AC[i]

    temp\_AC[len(*AC*)-1]=*Q*[0]

*AC*=''

*AC*=*AC*.join(temp\_AC)

    temp\_Q=list(*Q*)

*for* i *in* range(1,len(*Q*)):

        temp\_Q[i-1]=temp\_Q[i]

    temp\_Q[len(*Q*)-1]='\_'

*Q*=''

*Q*=*Q*.join(temp\_Q)

*return*(*C*,*AC*,*Q*)

def operation(*C*,*AC*,*M*):

    temp=*C*+*AC*

    temp=bin(int(temp,2)+int(*M*,2))

    temp=temp.replace("0b","")

*# discard the carry while operation is done*

*if*(len(temp)>len(*M*)):

        temp=temp[1::]

*return*(temp[0],temp[1::])

*# Main function*

AC=''

C='0'

Q=input("Enter the dividend(Q)  : ")

Q=deci\_to\_bin(int(Q))

M=input("Enter the divisor(M)   : ")

N = M;

M=deci\_to\_bin(int(M))

*if* (Q == M and M!='0'):

    print("Quotient: 1")

    print("Remainder: 0")

*elif* (deci\_to\_bin(int(N)) == '0' or Q==M==0):

    print("Infinity")

*elif* (int(Q,2) < int(M,2)):

    print("Quotient: 0")

    print("Remainder: " + str(int(Q,2)))

*else*:

*if*(len(Q)>len(M)):

*for* i *in* range(len(Q)):

            AC=AC+'0'

*else*:

*for* i *in* range(len(M)):

            AC=AC+'0'

    print("Initial C value is     : ",C)

    print("Initial AC value is    : ",AC)

    print("Initial Q value is     : ",Q)

*for* i *in* range(len(Q)-len(M)):

        M='0'+M

*# adding one bit extra*

    M='0'+M

    print("Value of M is          : ",M)

*# two's complement*

    M\_array=list(M)

*for* i *in* range(len(M)):

*if*(M[i]=='0'):

            M\_array[i]='1'

*if*(M[i]=='1'):

            M\_array[i]='0'

    M\_negative=''

    M\_negative=M\_negative.join(M\_array)

    M\_negative=bin(int(M\_negative,2)+int('1',2))

    M\_negative=M\_negative.replace("0b","")

    print("Two's complement of M  : ",M\_negative)

    print()

    print("---------------------------------------------------------------------------------------------------------------------")

    print("\t C "," "\*int(len(AC)/2),"AC"," "\*int(len(AC)/2)," "\*int(len(Q)/2),"Q"," "\*int(len(Q)/2),"     Operation done")

    print("---------------------------------------------------------------------------------------------------------------------")

    print("\t",C,"  ",AC,"  ",Q,"   ","Initial values")

    print()

*for* i *in* range(len(Q)):

        print("step",(i+1),":  ")

        C,AC,Q=shift\_left(C,AC,Q)

        print("\t",C,"  ",AC,"  ",Q,"   ","After shift left operation")

*# AC is positive*

*if*(C=='0'):

            C,AC=operation(C,AC,M\_negative)

            print("\t",C,"  ",AC,"  ",Q,"   ","AC\_equals\_AC\_minus\_M operation")

*# AC is negative*

*else*:

            C,AC=operation(C,AC,M)

            print("\t",C,"  ",AC,"  ",Q,"   ","AC\_equals\_AC\_plus\_M operation")

*# AC is negative*

*if*(C=='1'):

            temp\_Q=list(Q)

            temp\_Q[len(Q)-1]='0'

            Q=''

            Q=Q.join(temp\_Q)

*# AC is positive*

*else*:

            temp\_Q=list(Q)

            temp\_Q[len(Q)-1]='1'

            Q=''

            Q=Q.join(temp\_Q)

        print()

*# AC is negative*

    print("Final step: ")

*if*(C=='1'):

        print("Finally AC is negative. So, ")

        C,AC=operation(C,AC,M)

        print("\t",C,"  ",AC,"  ",Q,"   ","AC\_equals\_AC\_plus\_M operation")

*else*:

        print("No final step as AC is positive.")

    print('\nFinal values')

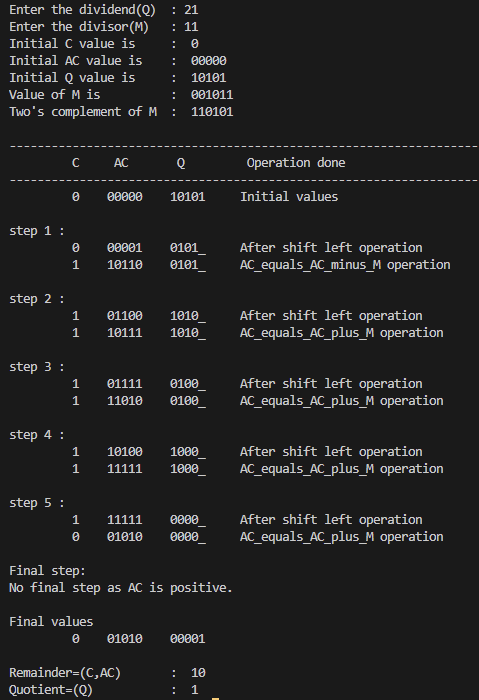
    print("\t",C,"  ",AC,"  ",Q)

    print()

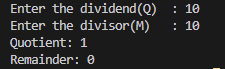
    print("Remainder=(C,AC)       : ",int(C+AC,2))

    print("Quotient=(Q)           : ",int(Q,2))

**Test Case 1:**

****

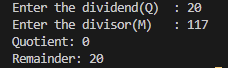
**Test Case 2:**

****

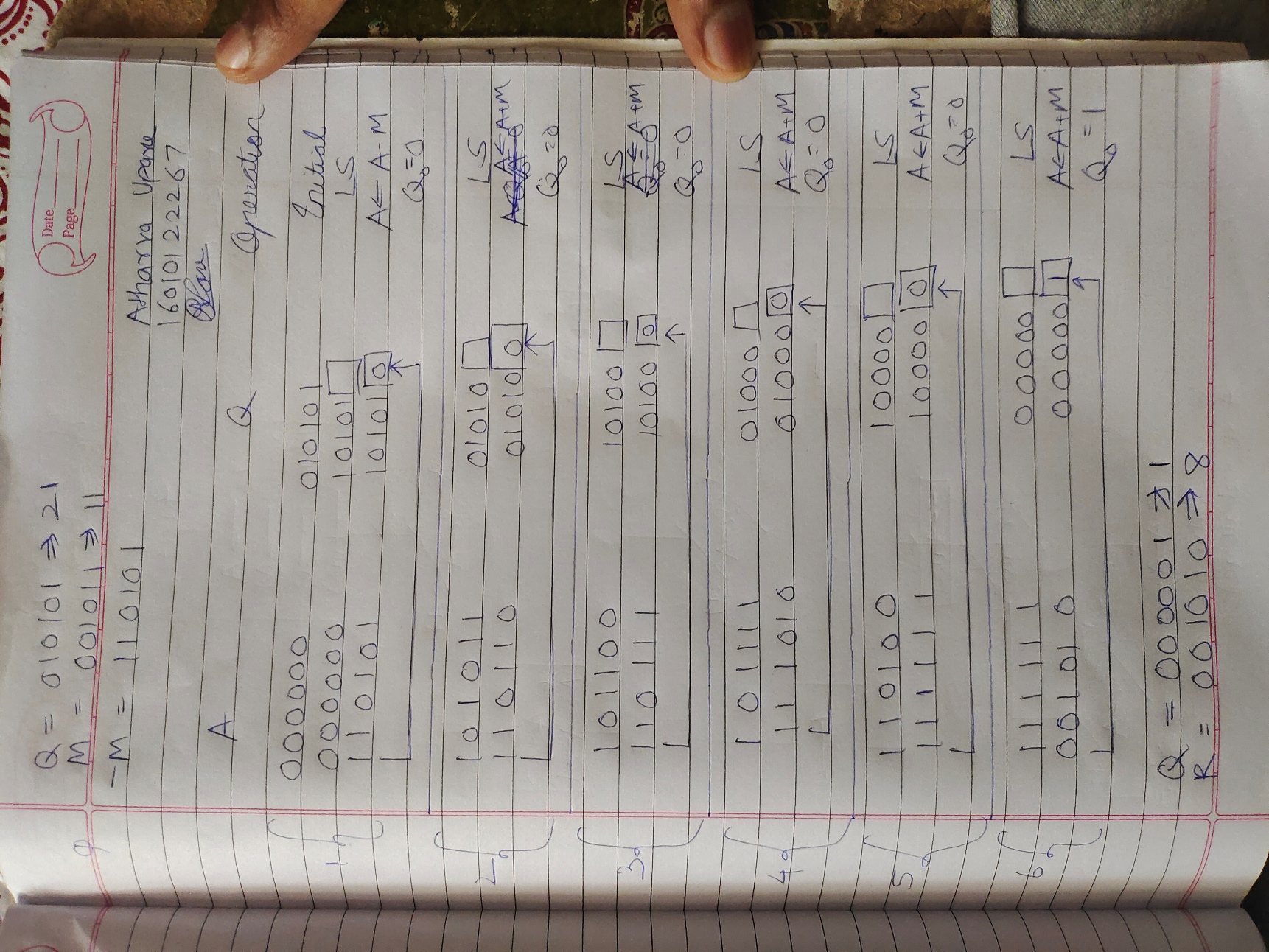
**Test Case 3:**

****

**Test Case 4:**

****

**Example: (Handwritten solved problem needs to uploaded)**

****

**Conclusion:**

In this experiment, we gained knowledge about the non-restoring division algorithm and its application in dividing numbers. Additionally, we validated this algorithm's functionality through the implementation of a code that performs division using the non-restoring division method.

**Post Lab Descriptive Questions**

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

**Ans:**

Non-restoring division and restoring division are distinct algorithms utilized for integer division. Non-restoring division holds several advantages over restoring division:

1. Simplified Implementation: Non-restoring division is often considered more straightforward to implement in hardware as it requires fewer control signals and is less complex to design.
2. Reduced Iterations: Typically, non-restoring division necessitates fewer iterations to complete the division process when compared to restoring division. This efficiency can result in quicker execution, particularly when dealing with large operands.
3. Speedier for Special Cases: Non-restoring division can outperform restoring division in specific input scenarios. For instance, when the divisor is significantly smaller than the dividend, non-restoring division can complete rapidly because it doesn't need to perform as many subtractions.
4. Lower Latency: Due to its decreased iteration count and simplified control logic, non-restoring division can exhibit lower latency in both hardware and software implementations. This reduced latency contributes to faster results.

**Date: \_\_\_\_\_\_\_\_\_\_\_\_\_**