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.

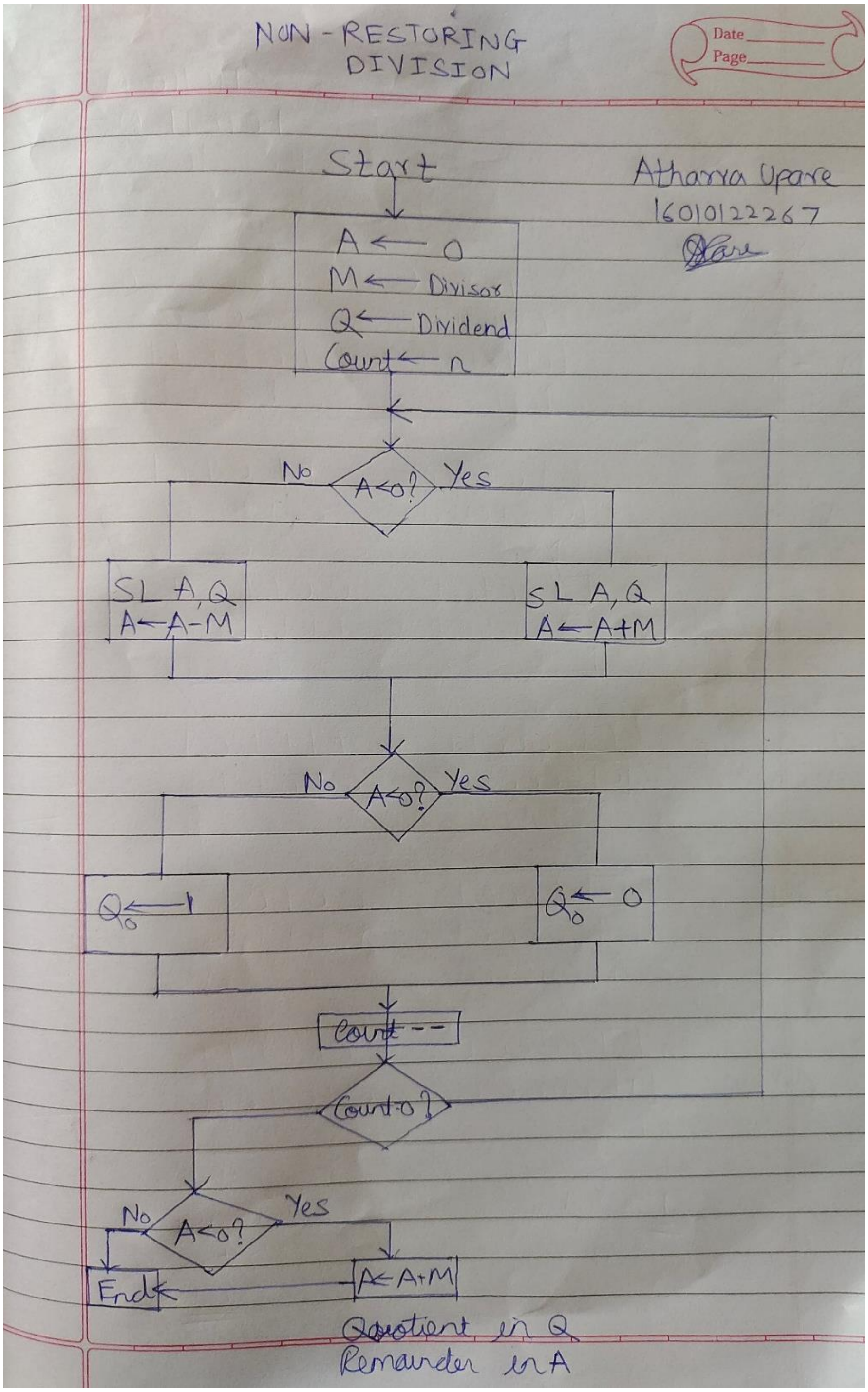
Books/ Journals/ Websites referred:

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)   : ")
```

```python
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:

```
Enter the dividend(Q)  : 21
Enter the divisor(M)   : 11
Initial C value is     :  0
Initial AC value is    :  00000
Initial Q value is     :  10101
Value of M is          :  001011
Two's complement of M  :  110101

-------------------------------------------------------------------
         C     AC       Q        Operation done
-------------------------------------------------------------------
         0    00000    10101     Initial values

step 1 :
         0    00001    0101_     After shift left operation
         1    10110    0101_     AC_equals_AC_minus_M operation

step 2 :
         1    01100    1010_     After shift left operation
         1    10111    1010_     AC_equals_AC_plus_M operation

step 3 :
         1    01111    0100_     After shift left operation
         1    11010    0100_     AC_equals_AC_plus_M operation

step 4 :
         1    10100    1000_     After shift left operation
         1    11111    1000_     AC_equals_AC_plus_M operation

step 5 :
         1    11111    0000_     After shift left operation
         0    01010    0000_     AC_equals_AC_plus_M operation

Final step:
No final step as AC is positive.

Final values
         0    01010    00001

Remainder=(C,AC)       :  10
Quotient=(Q)           :  1
```

Test Case 2:

```
Enter the dividend(Q)  : 10
Enter the divisor(M)   : 10
Quotient: 1
Remainder: 0
```

Test Case 3:

```
Enter the dividend(Q)  : 7
Enter the divisor(M)   : 0
Infinity
```

Test Case 4:

```
Enter the dividend(Q)  : 20
Enter the divisor(M)   : 117
Quotient: 0
Remainder: 20
```

Example: (Handwritten solved problem needs to uploaded)

Q = 010101 ⇒ 21
M = 001011 ⇒ 11
−M = 110101

Atharva Upare
16010122267

Step	A	Q	Operation
	000000	010101	Initial
1	000000	10101□	LS
	110101	10101 $\boxed{0}$	A ← A − M
			$Q_0 = 0$
2	101011	01010□	LS
	110110	01010 $\boxed{0}$	A ← A + M
			$Q_0 = 0$
3	101100	10100□	LS
	110111	10100 $\boxed{0}$	A ← A + M
			$Q_0 = 0$
4	101111	01000□	LS
	111010	01000 $\boxed{0}$	A ← A + M
			$Q_0 = 0$
5	110100	10000□	LS
	111111	10000 $\boxed{0}$	A ← A + M
			$Q_0 = 0$
6	111111	00000□	LS
	001010	00000 $\boxed{1}$	A ← A + M
			$Q_0 = 1$

Q = 000001 ⇒ 1
R = 001010 ⇒ 8

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: _____________