**Batch: A4 Roll No.: 16010122083**

**Experiment / assignment / tutorial No.\_\_\_\_\_\_\_**

|  |
| --- |
| **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: CO 1**

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

**Books/ Journals/ Websites referred:**

1. Carl Hamacher, ZvonkoVranesic and SafwatZaky, “Computer Organization”, Fifth Edition, TataMcGraw-Hill.
2. William Stallings, “Computer Organization and Architecture: Designing for Performance”, Eighth Edition, Pearson.

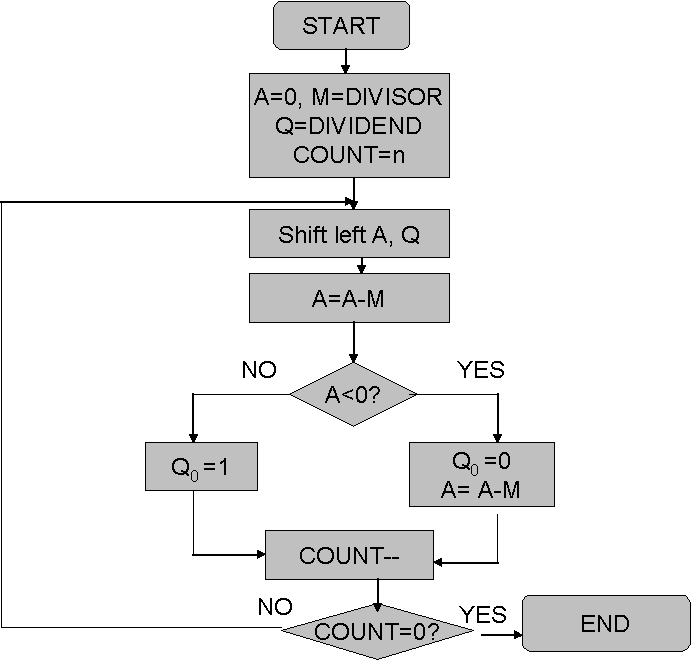
**3**. Dr. M. Usha, T. S. Srikanth, “Computer System Architecture and Organization”, First Edition, Wiley-India.

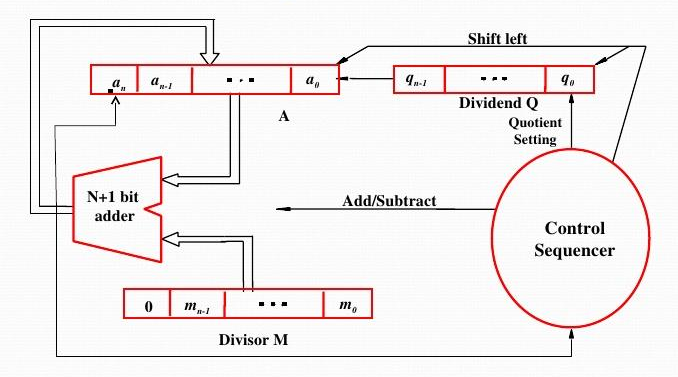
**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Pre Lab/ Prior Concepts:**

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

**Flowchart for Restoring of Division:**



****

**Circuit Arrangement for Binary Division**

**Design Steps**: ( This will take care of any combination of positive and negative numbers. These steps are slightly different than above flowchart)

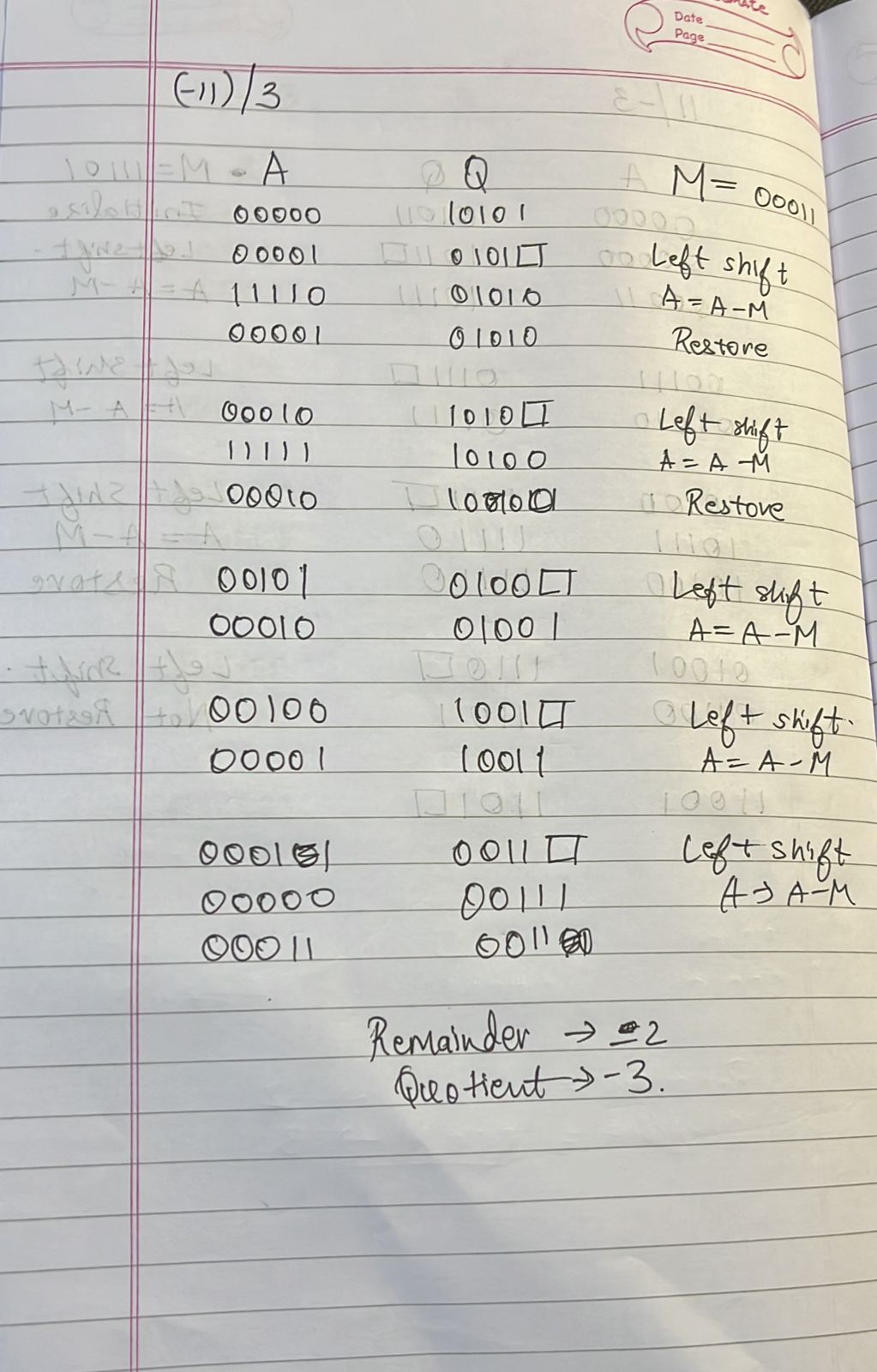
1. Start
2. Initialize M=Divisor and Dividend in A & Q registers. Dividend must be expressed as a 2n-bit twos complement number. Count=n (no of bits in dividend)
3. Left shift A, Q by 1 bit position
4. If MSB of A and M are same i.e sign of both is same
5. Then A=A-M
6. Else A=A+M
7. If MSB of previous A and present A (after subtract/ add operation) are same { This means previous operation is successful}
8. Then Q0=1 & retain present A (after operation)
9. Else Q0=0 & restore previous A ( before operation){ Since previous operation is not successful}
10. Decrement count.
11. If count≠ 0 go to step 3
12. Stop / End
13. The remainder is in A. If dividend is negative then remainder is in 2’s complement form. If signs of the divisor and dividend were the same then quotient is in Q as it is, otherwise the correct quotient is the twos complement of Q.

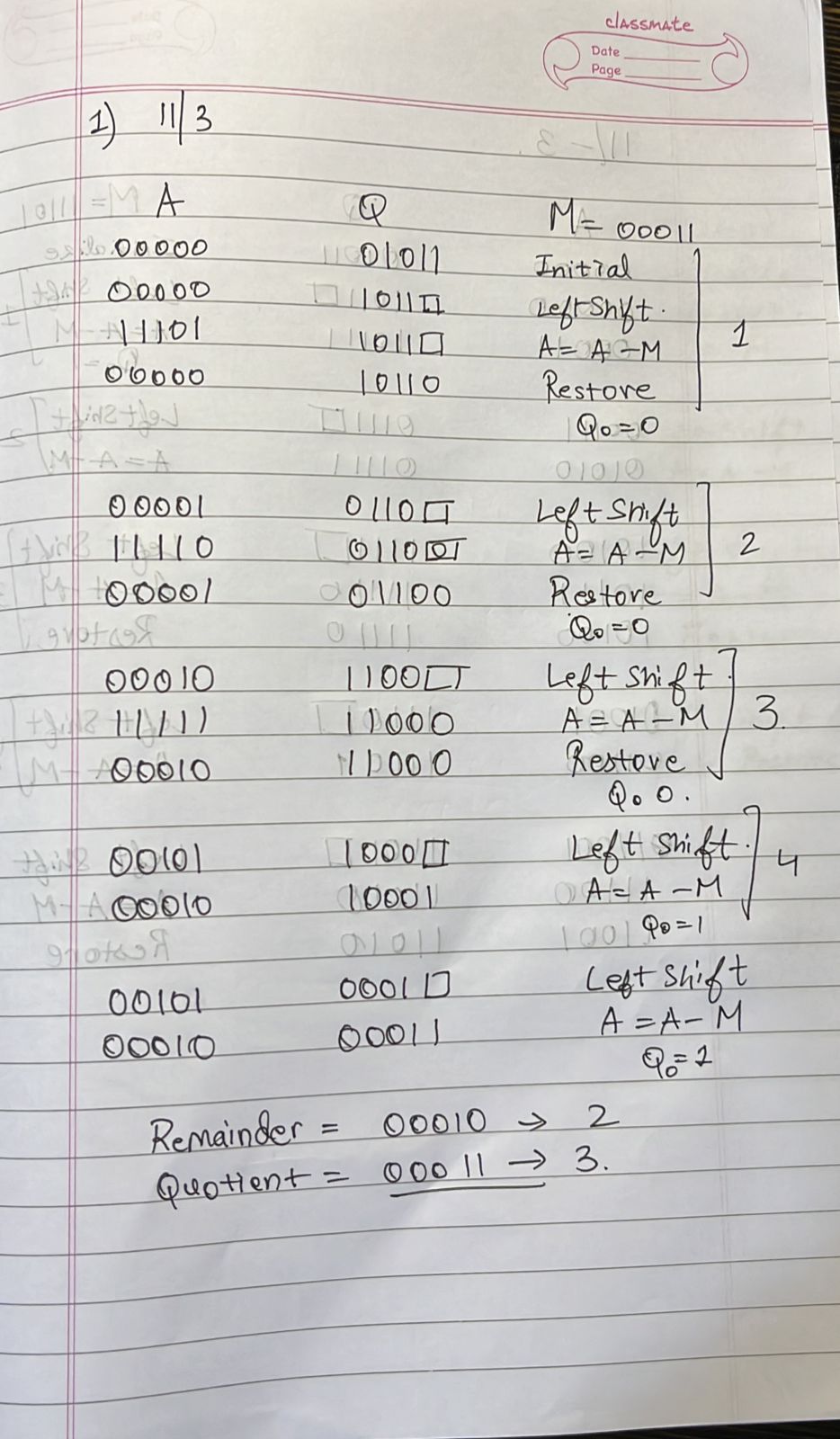
**Example:- (Handwritten following examples need to be solved with status of registers A, Q & M in every cycle) Consider these are 5 bit signed numbers. Use above design steps.**

1. 11/3 B) 11/ (-3) C) (-11)/ 3 D) (-11)/ (-3)

A piece of paper with writing on it

Description automatically generated

A paper with writing on it

Description automatically generated

**Code**

#include <bits/stdc++.h>

using namespace std;

string a = "000000000";

string q = "00000000";

string m = "000000000";

string m\_comp = "000000000";

string add(string A, string M) {

    int carry = 0;

    string Sum;

    for (int i = A.length() - 1; i >= 0; i--) {

        int temp = A[i] - '0' + M[i] - '0' + carry;

        if (temp > 1) {

            Sum.push\_back('0' + (temp % 2));

            carry = 1;

        }

        else {

            Sum.push\_back('0' + temp);

            carry = 0;

        }

    }

    // Return the sum from Most Significant to Low Significant

    reverse(Sum.begin(), Sum.end());

    return Sum;

}

string decimalToBinary(int n)

{

    string s = bitset<9> (n).to\_string();

    return s;

}

string complement(string m) {

    string M;

    // Iterating through the number

    for (int i = 0; i < m.length(); i++) {

        // Computing the Complement

        M.push\_back('0' + ((m[i] - '0' + 1) % 2));

    }

    // Adding 1 to the computed value

    M = add(M, "000000001");

    return M;

}

void restoringDivision(string Q, string M, string A) {

    int count = M.length();

    cout << "Initial Values: A:" << A << " Q:" << Q << " M:" << M << endl;

    while (count > 0) {

        // Printing the values at every step

        cout << "\nstep:" << (M.length() - count + 1) << endl;

        A = A.substr(1) + Q[0];

        string comp\_M = complement(M);

        A = add(A, comp\_M);

        // Left shift,assigning LSB of Q to MSB of A.

        cout << "Left Shift and Subtract: ";

        cout << " A:" << A << endl;

        cout << "A:" << A << " Q:" << Q.substr(1) << "\_";

        if (A[0] == '1') {

            // Unsuccessful and Quotient bit will be zero

            Q = Q.substr(1) + '0';

            cout << "  -Unsuccessful" << endl;

            // Restoration is required for A

            A = add(A, M);

            cout << "A:" << A << " Q:" << Q << " -Restoration" << endl;

        }

        else {

            // Quotient bit will be 1

            Q = Q.substr(1) + '1';

            cout << " Successful" << endl;

            // No restoration

            cout << "A:" << A << " Q:" << Q << " -No Restoration" << endl;

        }

        count--;

    }

    cout << "\nQuotient(Q):" << Q << " Remainder(A):" << A << endl;

}

int main() {

    int divisor, dividend;

    cout << "Enter the divisor and dividend in decimal\n";

    cin >> divisor >> dividend;

    restoringDivision(decimalToBinary(dividend), decimalToBinary(divisor), "000000000");

    return 0;

}

**Output**

A screenshot of a computer

Description automatically generated

**A screenshot of a computer

Description automatically generated**

**Post Lab Descriptive Question(s)**

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

Certainly, here's a concise list of the advantages of restoring division over non-restoring division:

1. Simplicity: Restoring division is simpler to implement in hardware due to its basic addition and subtraction operations.

2. Regular Timing: It offers a more predictable and uniform processing time for each step.

3. Fewer Components: Generally requires fewer hardware components, making it area-efficient.

4. Interrupt Handling: Easier to interrupt and resume without significant progress loss.

5. Division by Powers of 2: Well-suited for division by powers of 2 due to its compatible shift operations.

6. Unsigned Division: More straightforward for unsigned division due to simpler sign handling.

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