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

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

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

**AIM :** The basis of algorithm is based on paper and pencil approach and the operations 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: 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:**

**Design Steps for Non Restoring Division**

Refer circuit arrangement given in write up of Restoring Division

n- Number of bits in dividend (Unsigned number of width n)

Width of Register A is (n+1) bits wherein MS bit is sign bit (during subtract operation this is required)

Width of register M is (n+1) bits wherein MS bit is sign bit & is = 0 (for unsigned divisor)

1. A 🡨 0, M 🡨 Divisor (of n bits), Q 🡨 Dividend, Count 🡨 n
2. Left shift A, Q by 1 bit position
3. A 🡨 A –M
4. If sign of A is 1 then go to step 9 (subtraction is not successful)
5. Else q0 = 1
6. Count 🡨 Count – 1
7. Is count = 0? NO. Then go to step 2
8. Yes. End of program. {Q will have quotient and Remainder in A}
9. Q0 = 0
10. Count 🡨 Count – 1
11. Is count = 0? If yes then go to step 15
12. No. Then Left shift A, Q by 1 bit position
13. A 🡨 A + M
14. If sign of A is 0 then go to step 5 else go to step 9
15. A 🡨 A + M. This is required to restore remainder
16. End of program. {Q will have quotient and Remainder in A}

**Example: (Handwritten following example need to be solved with status of registers A, Q & M in every cycle) Consider these are 4 bit unsigned numbers. Use above design steps.**

1. 11/3

A piece of 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";

// Function to add two binary numbers

string add(string A, string M)

{

    int carry = 0;

    string Sum = ""; // Iterating through the number

    // A. Here, it is assumed that

    // the length of both the numbers

    // is same

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

        // Adding the values at both

        // the indices along with the

        // carry

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

        // If the binary number exceeds 1

        if (temp > 1) {

            Sum += to\_string(temp % 2);

            carry = 1;

        }

        else {

            Sum += to\_string(temp);

            carry = 0;

        }

    }

    // Returning the sum from

    // MSB to LSB

    return string(Sum.rbegin(), Sum.rend());

}

// Function to find the compliment

// of the given binary number

string decimalToBinary(int n)

{

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

    return s;

}

string compliment(string m)

{

    string M = ""; // Iterating through the number

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

        // Computing the compliment

        M += to\_string((m[i] - '0' + 1) % 2);

    }

    // Adding 1 to the computed

    // value

    M = add(M, "000000001");

    return M;

}

// Function to find the quotient

// and remainder using the

// Non-Restoring Division Algorithm

void nonRestoringDivision(string Q, string M, string A)

{

    // Computing the length of the

    // number

    int count = M.length();

    string comp\_M = compliment(M);

    // Variable to determine whether

    // addition or subtraction has

    // to be computed for the next step

    string flag = "successful";

    // Printing the initial values

    // of the accumulator, dividend

    // and divisor

    cout << "Initial Values: A: " << A << " Q: " << Q

        << " M: " << M << endl;

    // The number of steps is equal to the

    // length of the binary number

    while (count) {

        // Printing the values at every step

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

        // Step1: Left Shift, assigning LSB of Q

        // to MSB of A.

        cout << " Left Shift and ";

        A = A.substr(1) + Q[0];

        // Choosing the addition

        // or subtraction based on the

        // result of the previous step

        if (flag == "successful") {

            A = add(A, comp\_M);

            cout << "subtract: ";

        }

        else {

            A = add(A, M);

            cout << "Addition: ";

        }

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

        if (A[0] == '1') {

            // Step is unsuccessful and the

            // quotient bit will be '0'

            Q = Q.substr(1) + "0";

            cout << " -Unsuccessful";

            flag = "unsuccessful";

            cout << " A: " << A << " Q: " << Q

                << " -Addition in next Step" << endl;

        }

        else {

            // Step is successful and the quotient

            // bit will be '1'

            Q = Q.substr(1) + "1";

            cout << " Successful";

            flag = "successful";

            cout << " A: " << A << " Q: " << Q

                << " -Subtraction in next step" << endl;

        }

        count--;

    }

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

        << endl;

}

// Driver code

int main()

{

    int divisor, dividend;

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

    cin >> divisor >> dividend;

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

    return 0;

}

**Output**

**A screenshot of a computer program

Description automatically generated**

**Post Lab Descriptive Question(s)**

**What is/are the advantage(s) of non restoring division over restoring division? Justify the same.**

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