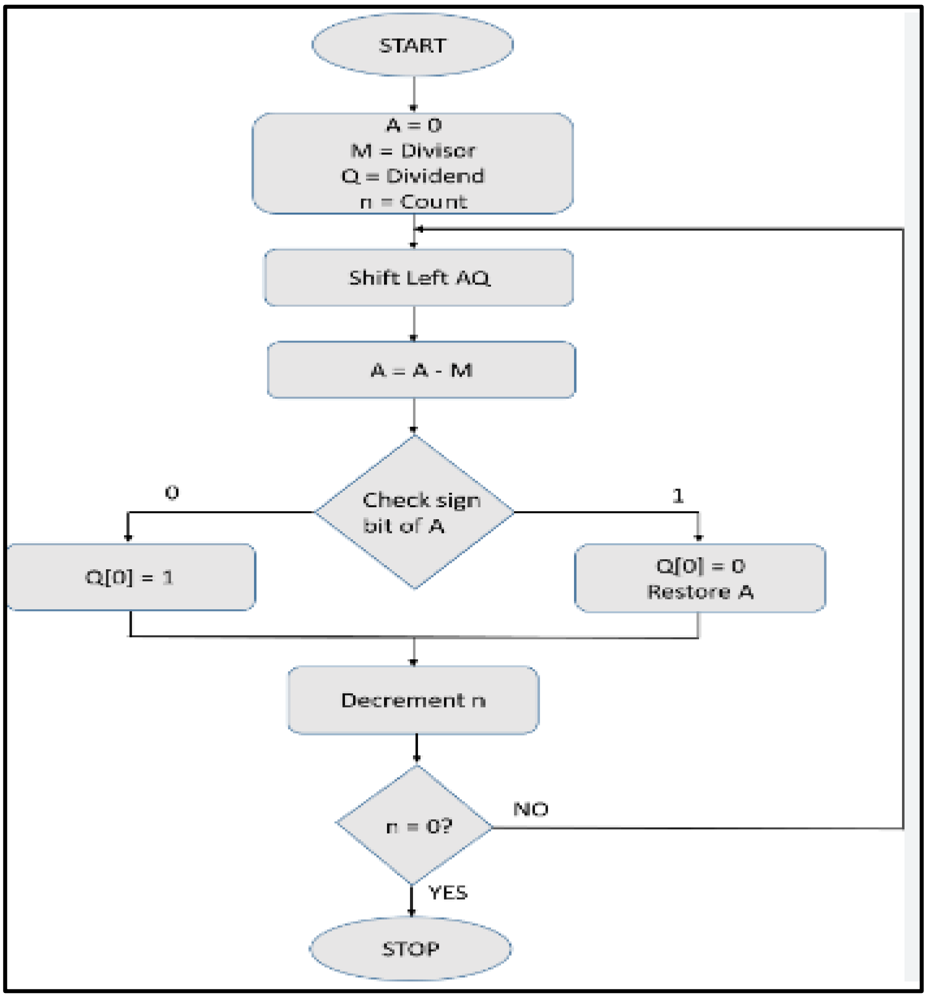
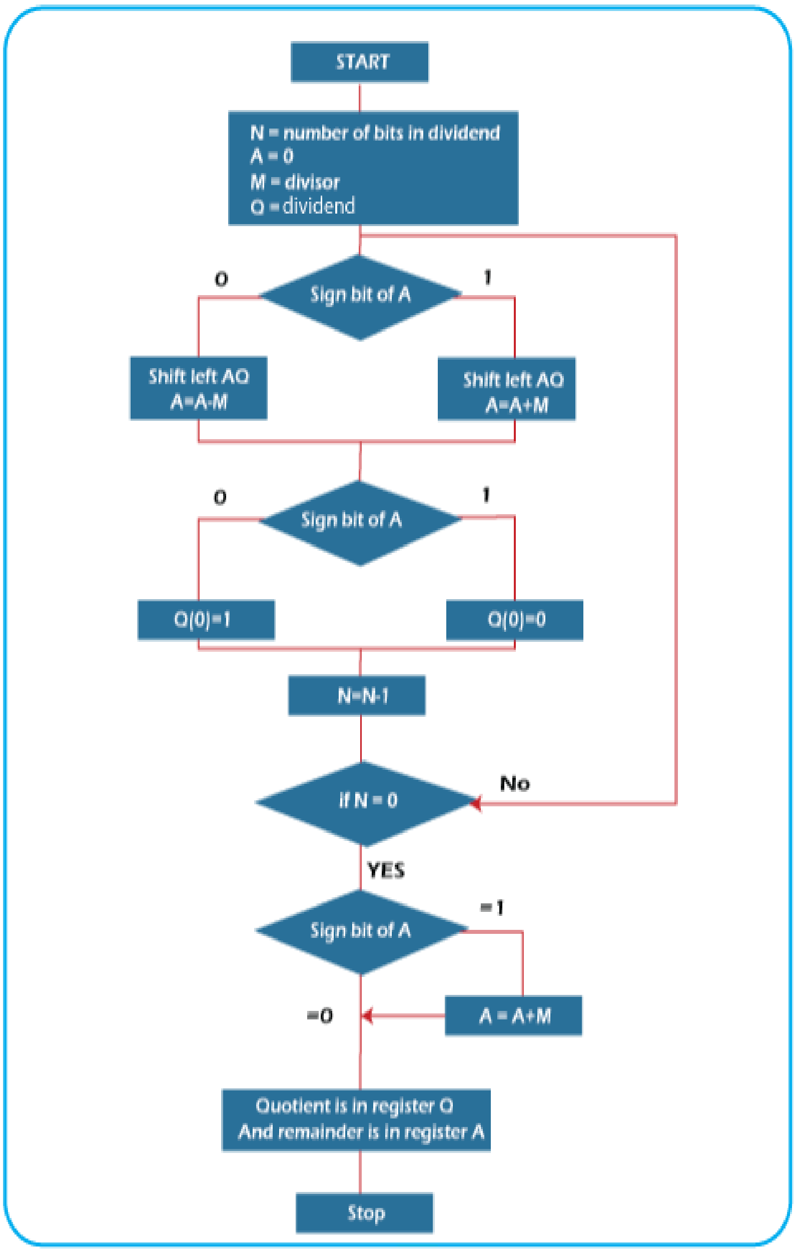
NAME :-	SUJAL SANDEEP DINGANKAR
UID :-	2024301005
SUBJECT :-	COA
EXPERIMENT NO :-	8
DATE OF PERFORMANCE :-	25/10/2024
AIM :-	**Program for Restoring & Non-Restoring Division..**
FLOWCHART :-	Restoring & Non-Restoring Division Flowchart.
	 START A = 0 M = Divisor Q = Dividend n = Count Shift Left AQ A = A - M Check sign bit of A 0 1 Q[0] = 1 Q[0] = 0 Restore A Decrement n n = 0? NO YES STOP



THEORY :-

<u>Restoring Division Algorithm :-</u>

Key Concepts:

- **Binary Representation**: Uses binary numbers and two's complement for signed values.
- **Registers**: Involves an accumulator (A), a quotient (Q), and a divisor (M).

Steps:

1. **Initialization**:
   - Set A to 0, Q to the dividend, and M to the divisor.
   - Determine the number of bits based on the divisor's binary representation.
2. **Iteration**:
   - For each bit:
     - **Left Shift** A and Q.

        - **Subtract** M from A.
            - If A ≥ 0, write '1' in Q and do not restore A.
            - If A < 0, restore A by adding M back and write '0' in Q.
3. **Finalization**:
    - Q contains the quotient, and A contains the remainder.

<u>Non-Restoring Division Algorithm :-</u>

Key Concepts:

- **Similar Structure**: Like restoring division, it uses binary representation and maintains A, Q, and M.

Steps:

1. **Initialization**:
    - Set A to 0, Q to the dividend, and M to the divisor.
2. **Iteration**:
    - For each bit:
        - **Left Shift** A and Q.
        - **Subtract** M from A.
            - If A ≥ 0, write '1' in Q (no restoration).
            - If A < 0, write '0' in Q (do not restore immediately).
        - **Add** M back to A if '0' was written in Q.
3. **Finalization**:
    - Q contains the quotient, and A contains the remainder.

PROGRAM :-

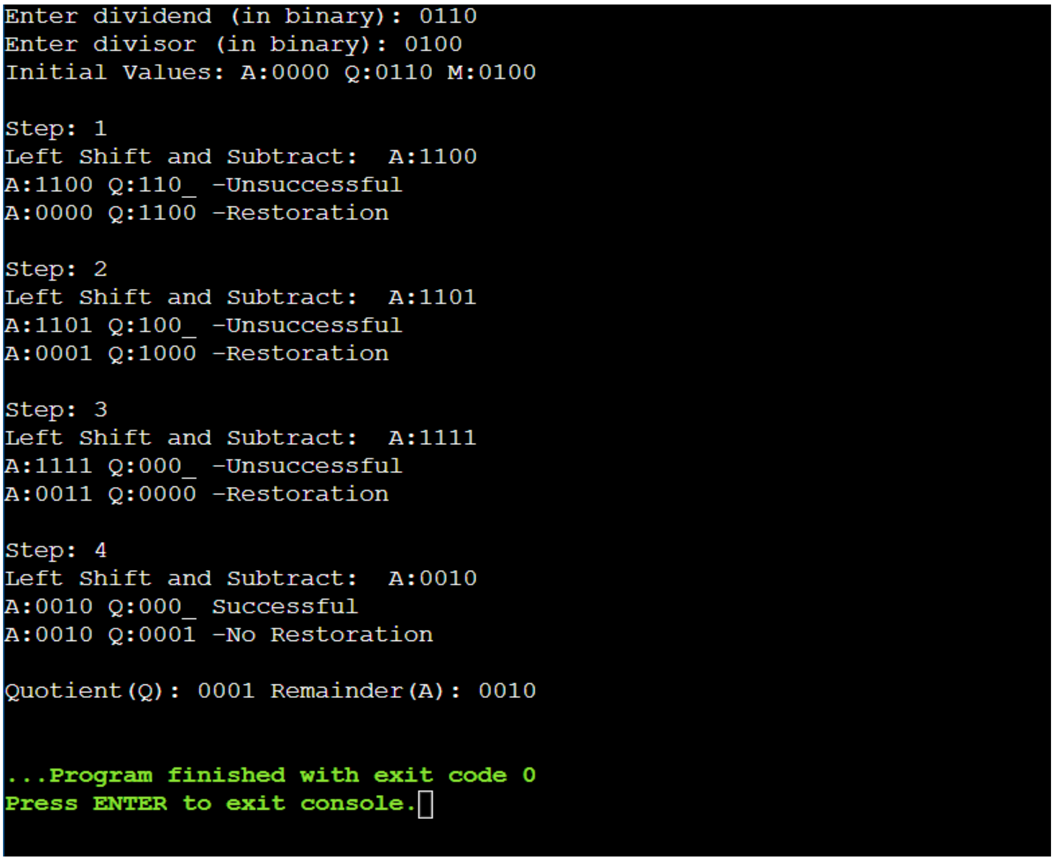
<u>Restoring Division Code :-</u>

```
#include <iostream>
#include <string>
#include <algorithm>

using namespace std;

// Function to add two binary numbers
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;
                }
        }
```

```
    reverse(Sum.begin(), Sum.end());
    return Sum;
}

// Function to find the complement of the binary number
string complement(string m) {
    string M;

    for (int i = 0; i < m.length(); i++) {
        M.push_back('0' + ((m[i] - '0' + 1) % 2));
    }
    M = add(M, "0001");
    return M;
}

// Function to find the quotient and remainder using Restoring Division
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) {
        cout << "\nStep: " << (M.length() - count + 1) << endl;
        A = A.substr(1) + Q[0];

        string comp_M = complement(M);
        A = add(A, comp_M);

        cout << "Left Shift and Subtract: ";
        cout << " A:" << A << endl;
        cout << "A:" << A << " Q:" << Q.substr(1) << "_";

        if (A[0] == '1') {
            Q = Q.substr(1) + '0';
            cout << " -Unsuccessful" << endl;
            A = add(A, M);
            cout << "A:" << A << " Q:" << Q << " -Restoration" << endl;
        } else {
            Q = Q.substr(1) + '1';
            cout << " Successful" << endl;
            cout << "A:" << A << " Q:" << Q << " -No Restoration" <<
endl;
        }
        count--;
    }
    cout << "\nQuotient(Q): " << Q << " Remainder(A): " << A << endl;
}

int main() {
    string dividend, divisor;
```

```
        // Taking input for dividend and divisor
        cout << "Enter dividend (in binary): ";
        cin >> dividend;
        cout << "Enter divisor (in binary): ";
        cin >> divisor;

        // Initialize accumulator as zeros with the same length as dividend
        string accumulator = string(dividend.length(), '0');

        // Call the restoring division function
        restoringDivision(dividend, divisor, accumulator);

        return 0;
}
```

OUTPUT : -



```
Enter dividend (in binary): 0110
Enter divisor (in binary): 0100
Initial Values: A:0000 Q:0110 M:0100

Step: 1
Left Shift and Subtract:  A:1100
A:1100 Q:110_ -Unsuccessful
A:0000 Q:1100 -Restoration

Step: 2
Left Shift and Subtract:  A:1101
A:1101 Q:100_ -Unsuccessful
A:0001 Q:1000 -Restoration

Step: 3
Left Shift and Subtract:  A:1111
A:1111 Q:000_ -Unsuccessful
A:0011 Q:0000 -Restoration

Step: 4
Left Shift and Subtract:  A:0010
A:0010 Q:000_ Successful
A:0010 Q:0001 -No Restoration

Quotient(Q): 0001 Remainder(A): 0010


...Program finished with exit code 0
Press ENTER to exit console.
```

Non-Restoring Division Code :-

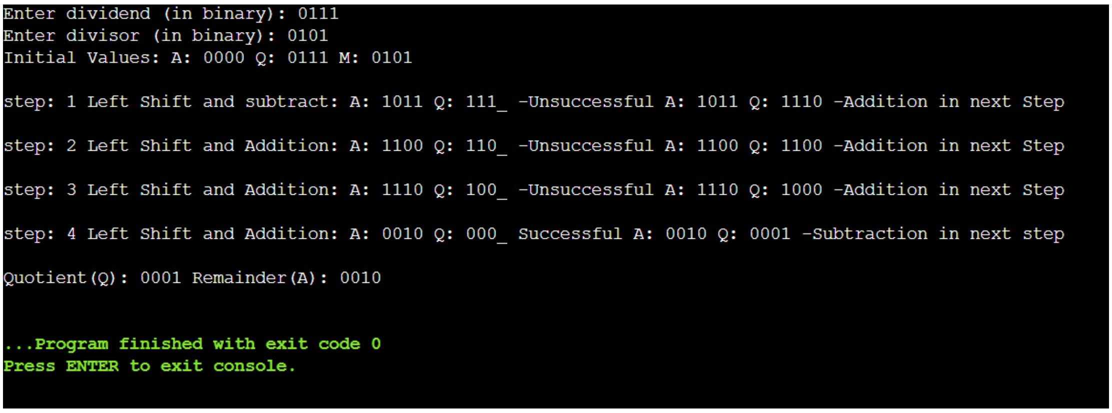
```
// Non-Restoring Division Algorithm
#include <iostream>
#include <string>
using namespace std;

// 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 thea
                // 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 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, "0001");
        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()
{
   string dividend, divisor;

          // Taking input for dividend and divisor
          cout << "Enter dividend (in binary): ";
          cin >> dividend;
          cout << "Enter divisor (in binary): ";
          cin >> divisor;

          string accumulator = string(dividend.size(), '0');
          nonRestoringDivision(dividend, divisor, accumulator);

          return 0;
}
```

OUTPUT :-



```
Enter dividend (in binary): 0111
Enter divisor (in binary): 0101
Initial Values: A: 0000 Q: 0111 M: 0101

step: 1 Left Shift and subtract: A: 1011 Q: 111_ -Unsuccessful A: 1011 Q: 1110 -Addition in next Step

step: 2 Left Shift and Addition: A: 1100 Q: 110_ -Unsuccessful A: 1100 Q: 1100 -Addition in next Step

step: 3 Left Shift and Addition: A: 1110 Q: 100_ -Unsuccessful A: 1110 Q: 1000 -Addition in next Step

step: 4 Left Shift and Addition: A: 0010 Q: 000_ Successful A: 0010 Q: 0001 -Subtraction in next step

Quotient(Q): 0001 Remainder(A): 0010


...Program finished with exit code 0
Press ENTER to exit console.
```

CONCLUSION :-

By performing this experiment, I understood the concept and implementation of restoring and Non-restoring division.