Non restoring division:

```
#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 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 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 = "0111";
    string divisor = "0101";
    string accumulator = string(dividend.size(), '0');
    nonRestoringDivision(dividend, divisor, accumulator);

    return 0;
}
```

```
input
Initial Values: A: 0000 Q: 0011 M: 0001

step: 1 Left Shift and subtract: A: 1111 Q: 011_ -Unsuccessful A: 1111 Q: 0110 -Addition in next Step

step: 2 Left Shift and Addition: A: 1111 Q: 110_ -Unsuccessful A: 1111 Q: 1100 -Addition in next Step

step: 3 Left Shift and Addition: A: 0000 Q: 100_ Successful A: 0000 Q: 1001 -Subtraction in next step

step: 4 Left Shift and subtract: A: 0000 Q: 001_ Successful A: 0000 Q: 0011 -Subtraction in next step

Quotient(Q): 0011 Remainder(A): 0000
```