NAME: Meith Navlakha
SAP: 60004190068
POA Practical

EXPERIMENT 2

AIM: Write a program to implement Restoring Division Algorithm.

THEORY:

Restoring Division

Restoring Division algorithm is a slow division algorithm. Here, register Q contains quotient and register A contains remainder. The n-bit dividend is loaded into Q and divisor is loaded into M. Value of Register is initially kept 0 and this is the register whose value is restored during iteration due to which it is named Restoring. Left Shift is performed in the algorithm and decision is taken which bit is to be added in the Q_0 bit of Q. When all the iteration is over then the value in A is the remainder and value in Q is the quotient.

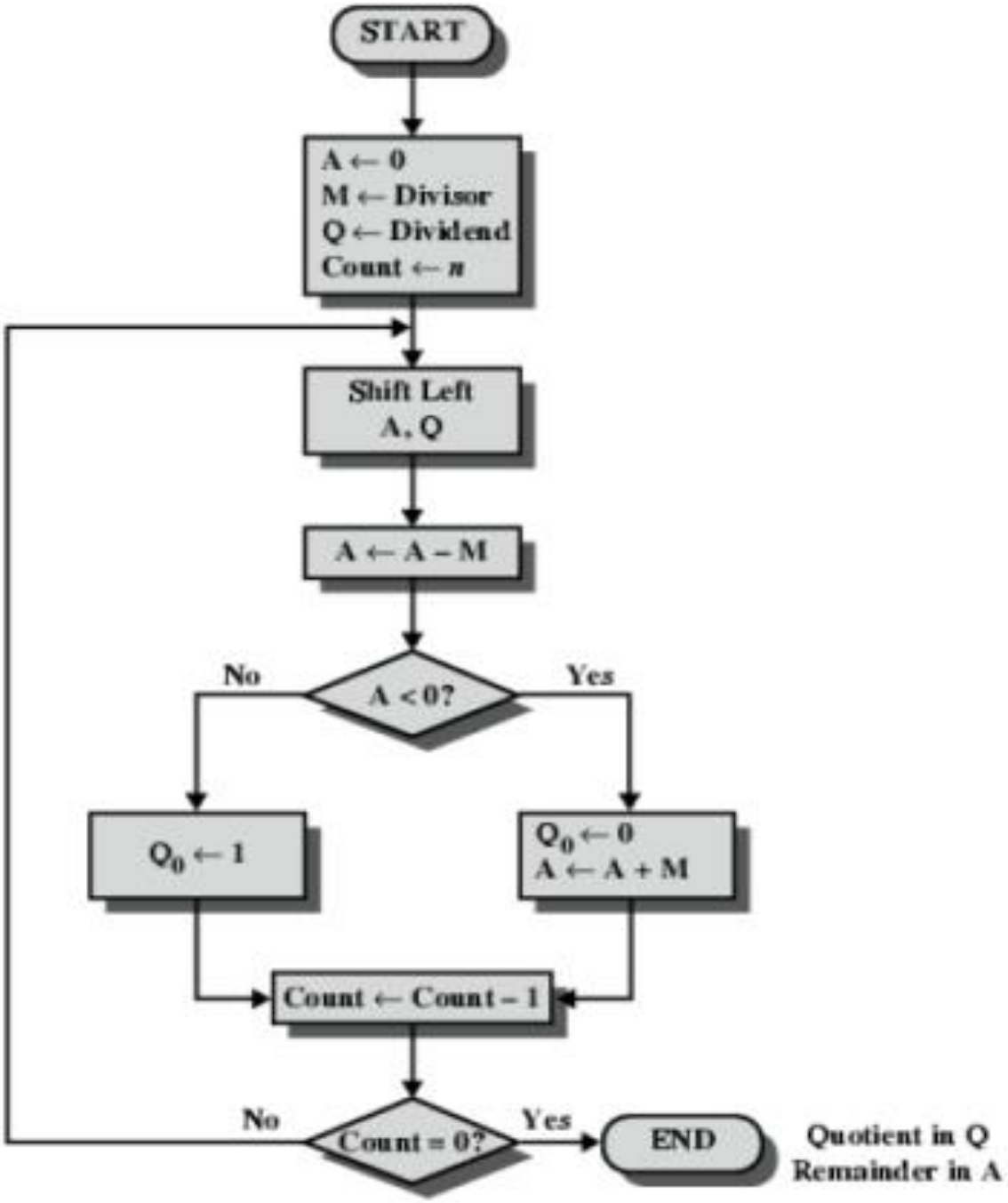
Restoring Division Algorithm:

- **Step-1:** First the registers are initialized with corresponding values (Q = Dividend, M = Divisor, A = 0, n = number of bits in dividend)
- **Step-2:** Then the content of register A and Q is shifted left as if they are a single unit
- **Step-3:** Then content of register M is subtracted from A and result is stored in A
- **Step-4:** Then the most significant bit of the A is checked if it is 0 the least significant bit of Q is set to 1 otherwise if it is 1 the least significant bit of Q is set to 0 and value of register A is restored i.e. the value of A before the subtraction with M
- **Step-5:** The value of counter n is decremented
- **Step-6:** If the value of n becomes zero we get of the loop otherwise we repeat from step 2
- **Step-7:** Finally, the register Q contain the quotient and A contain remainder

Disadvantages: Slower, as requires time because of restoration in each cycle.

NAME: Meith Navlakha
SAP: 60004190068
POA Practical

FLOWCHART



NAME: Meith Navlakha
SAP: 60004190068
POA Practical

CODE:

```
import java.util.*;
import java.util.Arrays;
import java.lang.Integer;
import java.lang.Math;

public class RestoringDiv {

  public static int[] Add(int[] A, int[] M) {

    int c = 0;
    for (int i = A.length - 1; i >= 0; i--) {
      A[i] = A[i] + M[i] + c;

      if (A[i] > 1) {
        A[i] = A[i] % 2;
        c = 1;
      } else {
        c = 0;
      }
    }

    return A;
  }

  public static String displayArray(int[] arr) {

    String s = "";
    for (int i = 0; i < arr.length; i++) {
      s = s + " " + arr[i];
    }

    return s;
  }

  public static int[] twoCompliment(int[] arr, int len) {

    //1's Compliment
    for (int i = 0; i < len; i++) {
      arr[i] = (arr[i] + 1) % 2;
    }

    int[] plus1 = new int[len];
    plus1[len - 1] = 1;
```

```
  // Add 1
  arr = Add(arr, plus1);

  // System.out.print("\n 2's Compliment : " + displayArray(arr)) ;
  return arr;
}

public static int[] tobinary(int num) {

  int m = Math.abs(num);
  int[] arr = new int[10];
  int count = 0;

  while (m > 0) {

    arr[count] = m % 2;
    count++;
    m /= 2;
  }

  int[] a = new int[count];

  for (int i = 0; i < count; i++) {
    a[count - i - 1] = arr[i];
  }

  return a;
}



public static int[][] leftShift(int[] A, int[] Q, int N) {

  int temp = Q[0];
  int[][] res = new int[2][];

  A[0] = A[1];
  for (int k = 0; k < N - 1; k++) {
    A[k + 1] = A[k + 2];
    Q[k] = Q[k + 1];

  }
  A[N] = temp;
  Q[N - 1] = -1;

  res[0] = A;
```

```
    res[1] = Q;

    System.out.print("\n LS : \t\t" + displayArray(A) + "\t" + displayArray(Q));

    return res;
  }

  public static int toDecimal(int[] arr) {

    int num = 0;
    for (int i = 0; i < arr.length; i++) {
      num = num * 2 + arr[i];
    }

    return num;
  }

  public static void restoringDiv(int[] M, int[] minusM, int[] Q, int N) {

    int[] A = new int[N + 1];
    int count = N;

    System.out.print("\n Operation\t     A\t            Q ");
    System.out.print("\n INITIALISE :  \t" + displayArray(A) + "\t" +
displayArray(Q));
    System.out.print("\n\n N = " + count);

    while (count > 0) {

      int[][] res = leftShift(A, Q, N);

      A = res[0];
      Q = res[1];

      A = Add(A, minusM);
      System.out.print("\n A= A-M : \t" + displayArray(A) + "\t" + displayArray(Q));

      if (A[0] == 1) {
        // Negative
        Q[N - 1] = 0;
        System.out.print("\n Q[0] = 0 : \t" + displayArray(A) + "\t" + displayArray(Q));

        A = Add(A, M);
        System.out.print("\n A = A+M : \t" + displayArray(A) + "\t" + displayArray(Q));

      } else {
```

```
        Q[N - 1] = 1;
        System.out.print("\n Q[0] = 1 : \t" + displayArray(A) + "\t" + displayArray(Q));
      }

      count--;
      System.out.print("\n\n N = " + count);

    }

    System.out.print("\n\n*****RESULT*****");
    System.out.print("\n QUOTIENT  : " + displayArray(Q) + " => " + toDecimal(Q));
    System.out.print("\n REMAINDER : " + displayArray(A) + " => " +
toDecimal(A));

  }

  public static void main(String args[]) {

    Scanner sc = new Scanner(System.in);

    System.out.print("\n Enter Dividend(Q) : ");
    int q = sc.nextInt();

    System.out.print("\n Enter Divisor(M) : ");
    int m = sc.nextInt();

    // int q =11 ;
    // int m =  3 ;

    int[] Q = tobinary(q);
    int N = Q.length;

    int[] M = new int[N + 1];
    int[] minusM = new int[N + 1];
    int[] bin_M = tobinary(m);

    // Padding
    int index = bin_M.length;
    while (index > 0) {

      M[N - bin_M.length + index] = bin_M[bin_M.length - index];
      index--;
    }

    System.arraycopy(M, 0, minusM, 0, minusM.length);
```

```
    minusM = twoCompliment(minusM, minusM.length);

    System.out.print("\n  Q = " + q + " : " + displayArray(Q));
    System.out.print("\n  M = " + m + "  : " + displayArray(M));
    System.out.print("\n -M = " + (-1 * m) + " : " + displayArray(minusM));

    System.out.print("\n\n ******RESTORING DIVISION ALGORITHM*****\n");

    restoringDiv(M, minusM, Q, N);

  }
}
```

OUTPUT:

Command Prompt

```
C:\Users\meith\Desktop\SEM 5\POA>java RestoringDiv

 Enter Dividend(Q) : 11

 Enter Divisor(M) : 3

  Q = 11 :  1 0 1 1
  M = 3  :  0 0 0 1 1
 -M = -3 :  1 1 1 0 1

 ******RESTORING DIVISION ALGORITHM*****

 Operation            A              Q
 INITIALISE :    0 0 0 0 0       1 0 1 1

 N = 4
 LS :            0 0 0 0 1       0 1 1 -1
 A= A-M :        1 1 1 1 0       0 1 1 -1
 Q[0] = 0 :      1 1 1 1 0       0 1 1 0
 A = A+M :       0 0 0 0 1       0 1 1 0

 N = 3
 LS :            0 0 0 1 0       1 1 0 -1
 A= A-M :        1 1 1 1 1       1 1 0 -1
 Q[0] = 0 :      1 1 1 1 1       1 1 0 0
 A = A+M :       0 0 0 1 0       1 1 0 0

 N = 2
 LS :            0 0 1 0 1       1 0 0 -1
 A= A-M :        0 0 0 1 0       1 0 0 -1
 Q[0] = 1 :      0 0 0 1 0       1 0 0 1

 N = 1
 LS :            0 0 1 0 1       0 0 1 -1
 A= A-M :        0 0 0 1 0       0 0 1 -1
 Q[0] = 1 :      0 0 0 1 0       0 0 1 1

 N = 0

*****RESULT*****
 QUOTIENT   :  0 0 1 1 => 3
 REMAINDER :  0 0 0 1 0 => 2
C:\Users\meith\Desktop\SEM 5\POA>
```

NAME: Meith Navlakha
SAP: 60004190068
POA Practical

CONCLUSION:

In this experiment I implemented Restoring Division Algorithm in Java. Here we take two inputs: Divisor(Q) and Dividend(M) and convert them into binary. Iterating, accumulator(A) and Q till N(the number of bits in binary of Q). Left Shift and Subtraction by 2's Compliment is used during the process. Finally, value of Q is the quotient and value of A is the remainder. Newton–Raphson and Goldschmidt are faster than Restoring Division Algorithm.