Batch: A1	**Roll No.: 16010120015**
Experiment / assignment / tutorial No. 3	
Grade: AA / AB / BB / BC / CC / CD /DD	
Signature of the Staff In-charge with date	

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: (Mention CO /CO's attained here)

CO 2-Detail working of the arithmetic logic unit and its sub modules
CO 3-Understand the Central processing unit with addressing modes and working of control unit

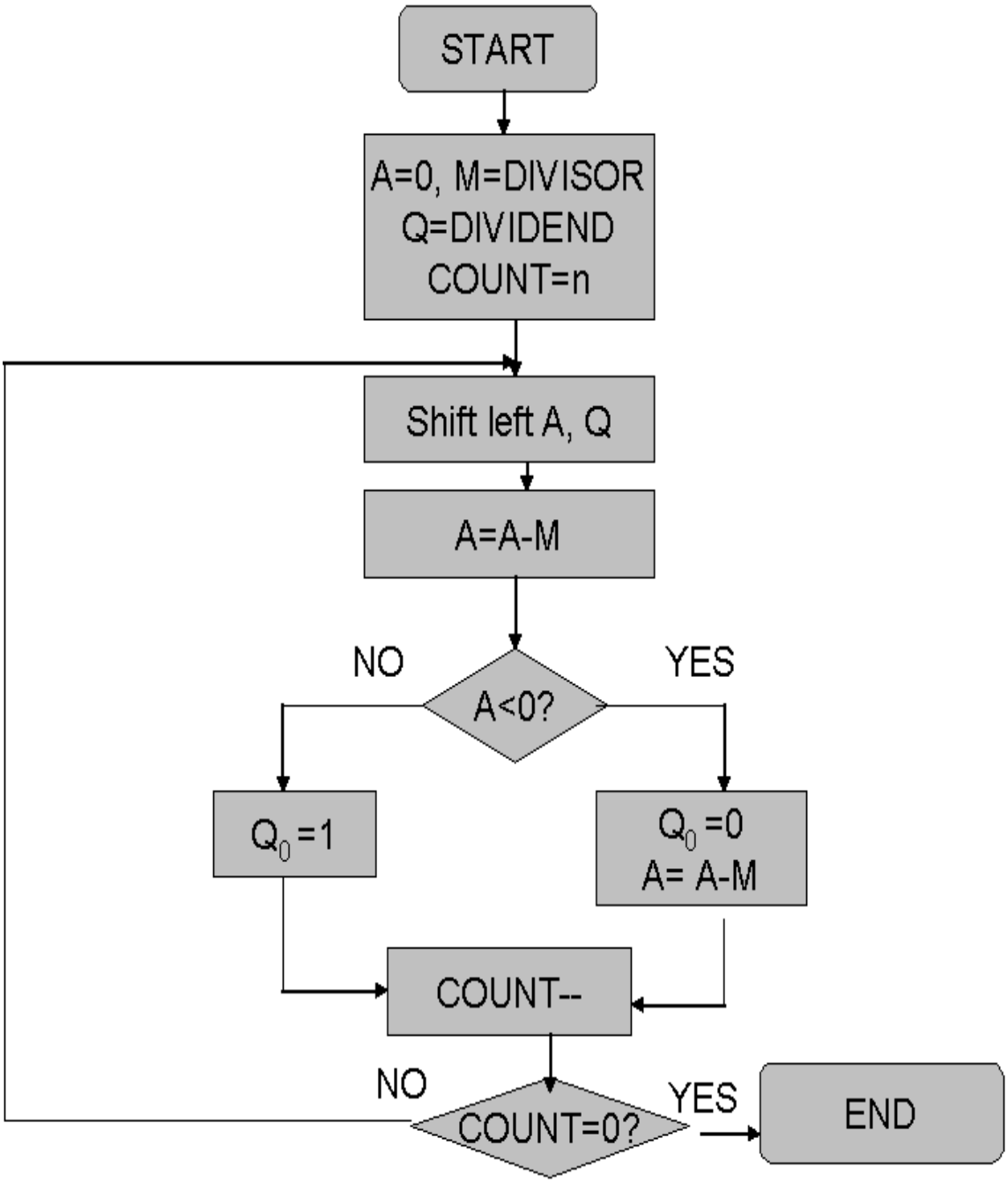
Books/ Journals/ Websites referred:

1. Carl Hamacher, Zvonko Vranesic and Safwat Zaky, "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:



Design Steps:

1. Start
2. Input variables ONE , A , M , Q negM, bits
3. To initialize , the values constructor is used
4. If MSB of A and M are same
5. Then A=A-M
6. Else A=A+M
7. If MSB of previous A and present A are same
8. Q=0 & store present A
9. Else Q=0 & restore previous A
10. Start from MSB(A) -> LSB(A) -> MSB(Q) -> LSB(Q)
11. fliping all the bits ,and call all the Binary ADD(negM)
12. Output Decimal , quotient , remainder
13. STOP

Program:

```
import java.util.Arrays;
import java.util.Scanner;

public class RestoringDiv {
    int ONE[];
    int A[];
    int M[];
    int Q[];
    int negM[];
    int bits;


    RestoringDiv(int bits) {
        this.A = new int[bits];
        this.M = new int[bits];
        this.Q = new int[bits];
        this.negM = new int[bits];
        this.ONE = new int[bits];
        this.ONE[0] = 1;
        this.bits = bits;
    }
    void toBinary(int num, char flag)
    {
```

```
        int temp[] = new int[this.bits];
        String binary = Integer.toBinaryString(num);
        int len = binary.length(), i;


        for (i = 0; i < len; i++) {

            temp[i] = Character.getNumericValue(binary.charAt(len - 1 - i
));
        }


        switch (flag) {
            case 'M':
                this.M = temp.clone();
                break;

            case 'Q':
                this.Q = temp.clone();
                break;

            default:
                break;
        }
    }


    void binaryAdd(int[] add1, int[] add2, char flag) {
        int temp1[] = add1.clone();
        int temp2[] = add2.clone();
        int tempSum[] = new int[this.bits];
        int sum = 0, carry = 0, i;


        for (i = 0; i < temp1.length; i++) {

            sum = (temp1[i] ^ temp2[i]) ^ carry;
            carry = (temp1[i] & temp2[i]) | (temp1[i] & carry) | (carry &
 temp2[i]);

            tempSum[i] = sum;
        }
```

```
        switch (flag) {
            case 'A':
                this.A = tempSum.clone();
                break;

            case 'm':
                this.negM = tempSum.clone();
                break;

            default:
                break;
        }
    }

    int toDecimal(int[] convert) {
        int result = 0, multiplier = 1;
        int temp[] = convert.clone();

        for (int i = 0; i < temp.length; i++) {
            result += (multiplier * temp[i]);
            multiplier *= 2;
        }

        return result;
    }


    void shiftLeft() {
        int i;


        for (i = (this.bits - 1); i > 0; i--) {
            this.A[i] = this.A[i - 1];
        }

        this.A[0] = this.Q[this.bits - 1];

        for (i = (this.bits - 1); i > 0; i--) {
            this.Q[i] = this.Q[i - 1];
        }

        this.Q[0] = 0;
    }
```

```
void calcNegM() {
    int i;


    for (i = 0; i < this.M.length; i++) {
        if (this.M[i] == 0)
            this.negM[i] = 1;
        if (this.M[i] == 1)
            this.negM[i] = 0;
    }

    binaryAdd(this.negM, this.ONE, 'm');
}

void displayArr(int[] arr) {
    int len = arr.length;
    for (int i = len - 1; i >= 0; i--) {
        System.out.print(arr[i]);
    }
}

public static void main(String[] args) {
    int bits = 6, i;
    Scanner scanner = new Scanner(System.in);

    System.out.print("Enter the dividend: ");
    int dividend = scanner.nextInt();

    System.out.print("Enter the divisor: ");
    int divisor = scanner.nextInt();

    for (i = 1; i <= 8; i++) {
        if (Math.abs(dividend) < Math.pow(2, i)) {
            bits = i + 1;
            break;
        }
    }

    RestoringDiv div = new RestoringDiv(bits);

    div.toBinary(divisor, 'M');
    div.toBinary(dividend, 'Q');
    div.calcNegM();
```

```
        System.out.print("\n M: ");
        div.displayArr(div.M);
        System.out.print(" -M: ");
        div.displayArr(div.negM);
        System.out.print(" Q: ");
        div.displayArr(div.Q);
        System.out.println("\n");

        while (bits > 0) {

            div.shiftLeft();
            div.binaryAdd(div.A, div.negM, 'A');

            div.displayArr(div.A);
            System.out.print("   ");
            div.displayArr(div.Q);
            System.out.println("   " + bits);

            if (div.A[div.bits - 1] == 0) {
                div.Q[0] = 1;
            }
            if (div.A[div.bits - 1] == 1)
            {
                div.Q[0] = 0;
                div.binaryAdd(div.A, div.M, 'A');
            }

            div.displayArr(div.A);
            System.out.print("   ");

            div.displayArr(div.Q);
            System.out.println("   " + bits);

            bits--;
        }

        System.out.print("\nIn Binary:");
        System.out.print("\nQuotient: ");
        div.displayArr(div.Q);
        System.out.print("\nRemainder: ");
        div.displayArr(div.A);

        System.out.print("\n\nIn Decimal:");
```

```
        System.out.print("\nQuotient: " + div.toDecimal(div.Q));
        System.out.print("\nRemainder: " + div.toDecimal(div.A));

        scanner.close();
    }
}
```

Output:

```
PS D:\COMPUTER SCIENCE\LANGUAGES\JAVA\Coa\Restoringdivision>  & 'c:\
Files\Eclipse Foundation\jdk-11.0.12.7-hotspot\bin\java.exe' '-agent
'-cp' 'C:\Users\YASH\AppData\Roaming\Code\User\workspaceStorage\bd4a
Enter the dividend: 5
Enter the divisor: 15

 M: 1111 -M: 0001 Q: 0101

0001   1010   4
0001   1011   4
0100   0110   3
0100   0111   3
1001   1110   2
1000   1110   2
0010   1100   1
0010   1101   1

In Binary:
Quotient: 1101
Remainder: 0010

In Decimal:
Quotient: 13
Remainder: 2
PS D:\COMPUTER SCIENCE\LANGUAGES\JAVA\Coa\Restoringdivision>
```

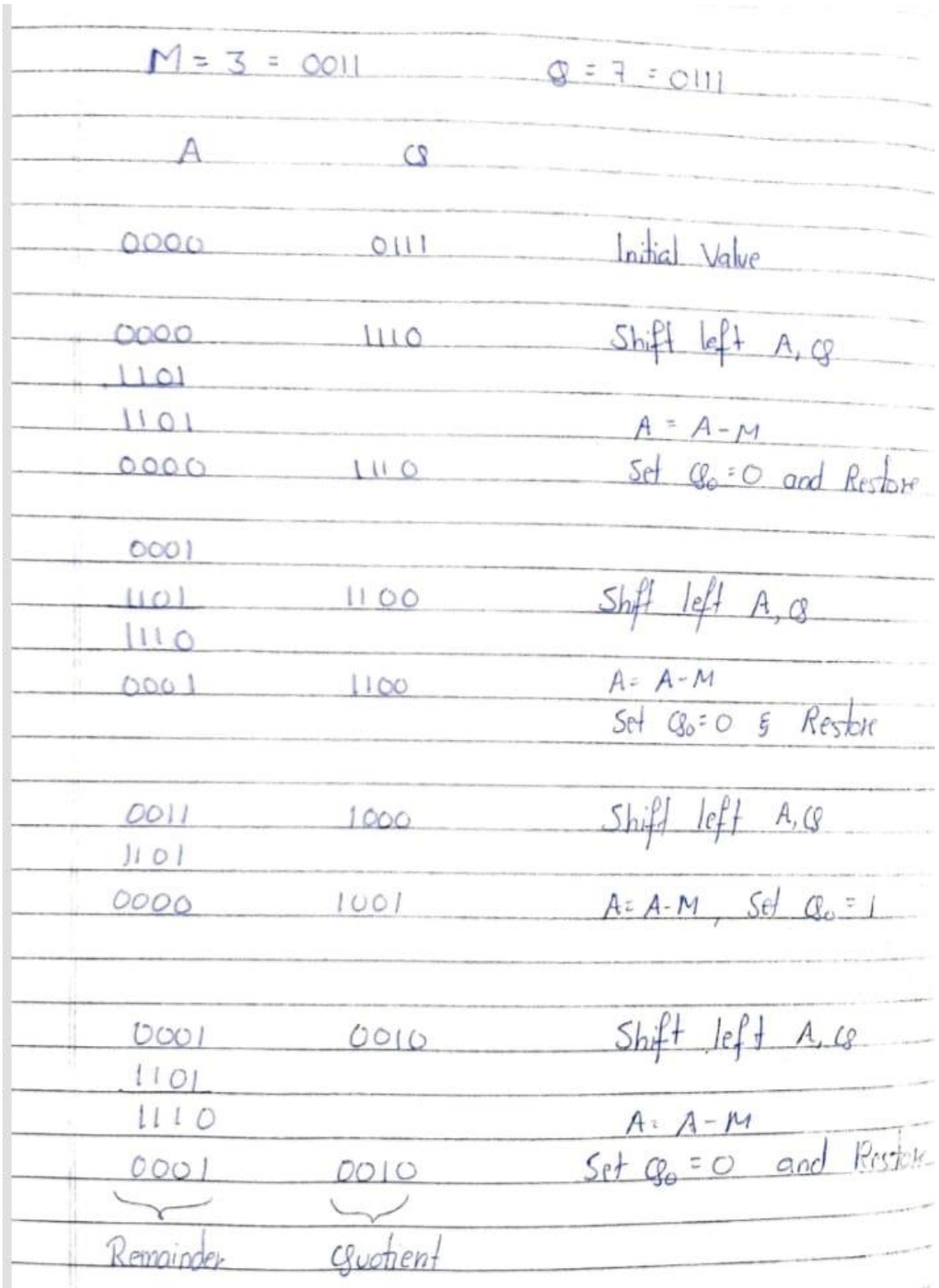
Example:- (Handwritten solved problems needs to be uploaded)

Here we take the example of dividing 7 by 3.

7 = 0111 = Dividend (Q)

3= 0011 = Divisor (M)

-M= 1101



Quotient = 0010= 2

Remainder = 0001 = 1

Conclusion:

Thus, we understood The concept of restoring method of division was used to divide two binary numbers and verified as the desired outputs were achieved.

Post Lab Descriptive Questions

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

ANS-

1. The advantage of using non-restoring arithmetic over the standard restoring division is that a test subtraction is not required
   That is the sign bit determines whether an addition or Subtraction is used.
2. Non-restoring is faster , whereas restoring
   method needs up to 2n-1 steps , we need to do a correction addition per each step,
3. The correction is done only once at the end in the case of non-restoring division and hence it is more efficient.
4. Even No extra bit must be maintained to keep a track of the sign.

Date: 13-09-2021 **Signature of faculty in-charge**