K. J. Somaiya College of Engineering, Mumbai-77
(A Constituent College of Somaiya Vidyavihar University)
Department of Computer Engineering

Batch: A1	**Roll No.: 16010120015**
Experiment / assignment / tutorial No.__2_____	
Grade: AA / AB / BB / BC / CC / CD /DD	
Signature of the Staff In-charge with date	

TITLE : To study and implement Non Restoring method of division

AIM : The basis of algorithm is based on paper and pencil approach and the operation 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: (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:

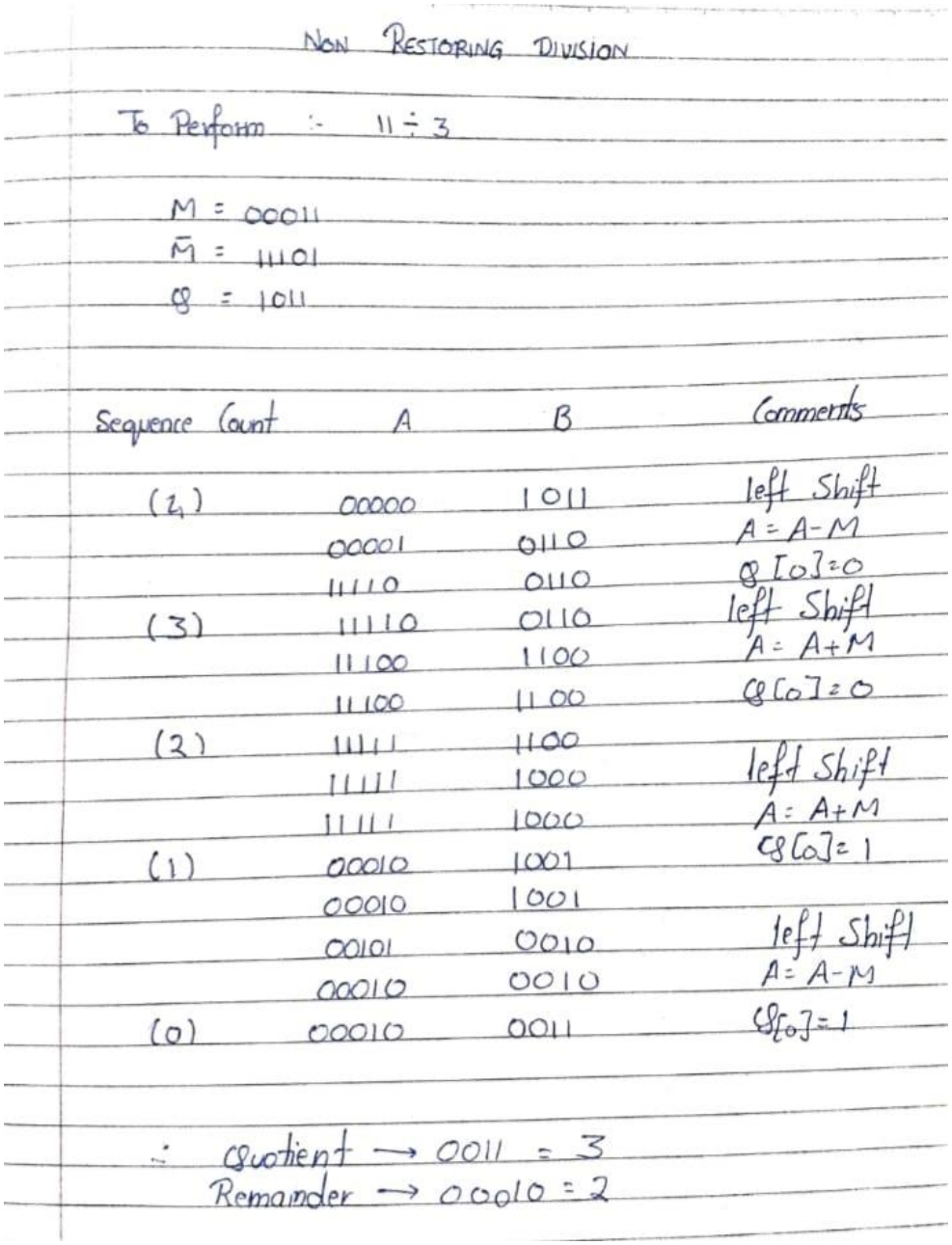
1. Carl Hamacher, Zvonko Vranesic and Safwat Zaky, "Computer Organization", Fifth Edition, TataMcGraw-Hill.
2. William Stallings, "Computer Organization and Architecture: Designing for Performance", Eighth Edition, Pearson.
3. Dr. M. Usha, T. S. Srikanth, "Computer System Architecture and Organization", First Edition, Wiley-India.

Pre Lab/ Prior Concepts:

The Non Restoring algorithm works with any combination of positive and negative numbers.

Flowchart for Non Restoring of Division:

Example: (Handwritten solved problem needs to uploaded)



NON RESTORING DIVISION

To Perform :- 11 ÷ 3

M = 00011
$\bar{M}$ = 11101
Q = 1011

Sequence Count	A	B	Comments
(4)	00000	1011	left Shift
	00001	0110	A = A-M
	11110	0110	Q[0] = 0
(3)	11110	0110	left Shift
	11100	1100	A = A+M
	11100	1100	Q[0] = 0
(2)	11111	1100	
	11111	1000	left Shift
	11111	1000	A = A+M
(1)	00010	1001	Q[0] = 1
	00010	1001	
	00101	0010	left Shift
	00010	0010	A = A-M
(0)	00010	0011	Q[0] = 1

∴ Quotient → 0011 = 3
Remainder → 00010 = 2

```java
import java.util.Scanner;


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

    NonRestoringDiv(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 & temp
2[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 count = 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)) {
                count = i + 1;
                break;
            }
        }
```

```
NonRestoringDiv div = new NonRestoringDiv(count);

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 (count > 0) {
    div.shiftLeft();

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

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

    if (div.A[div.bits - 1] == 0) {
        div.Q[0] = 1;
    }
    if (div.A[div.bits - 1] == 1) // if A is negative
    {
        div.Q[0] = 0;
    }

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

    count--;
}

if (div.A[div.bits - 1] == 1) {
```

```
            div.binaryAdd(div.A, div.M, 'A');
        }

        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();
    }
}
```

```
PS D:\Projects\JAVA\Coa\non restoring>  & 'c:\Users\YASH\.
on\jdk-11.0.12.7-hotspot\bin\java.exe' '-agentlib:jdwp=tra
ppData\Roaming\Code\User\workspaceStorage\00d3ed7005fd7f75
Enter the dividend: 11
Enter the divisor: 3

 M: 00011 -M: 11101 Q: 01011

11101   10110   5
11101   10110   5
11110   01100   4
11110   01100   4
11111   11000   3
11111   11000   3
00010   10000   2
00010   10001   2
00010   00010   1
00010   00011   1

In Binary:
Quotient: 00011
Remainder: 00010

In Decimal:
Quotient: 3
Remainder: 2
PS D:\Projects\JAVA\Coa\non restoring>
```

Conclusion

The concept of non-restoring division was studied and the code for the same was written in python.

Post Lab Descriptive Questions

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

- It is less complex than the restoring one because simpler operation are involved
- i.e. addition and subtraction, also restoring step is performed.
- In the method,A test substractor is not required .the sign bit of the register which initially contain zero named as A.

Example: (difference)

restoring method: we add the divisor back, and put 0 as your next quotient digit

non-restoring method: we don't do that , we keep negative remainder and a digit 1, and basically correct things by a supplementary addition afterwards.

Date: ___20/09/2021___ **Signature of faculty in-charge**