Batch: A2	**Roll No.: 16010122041**
Experiment / assignment / tutorial No. 4	
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)

To better understand the non-restoring algorithm and executing it using a programming language. To find the advantage of non-restoring over restoring division.

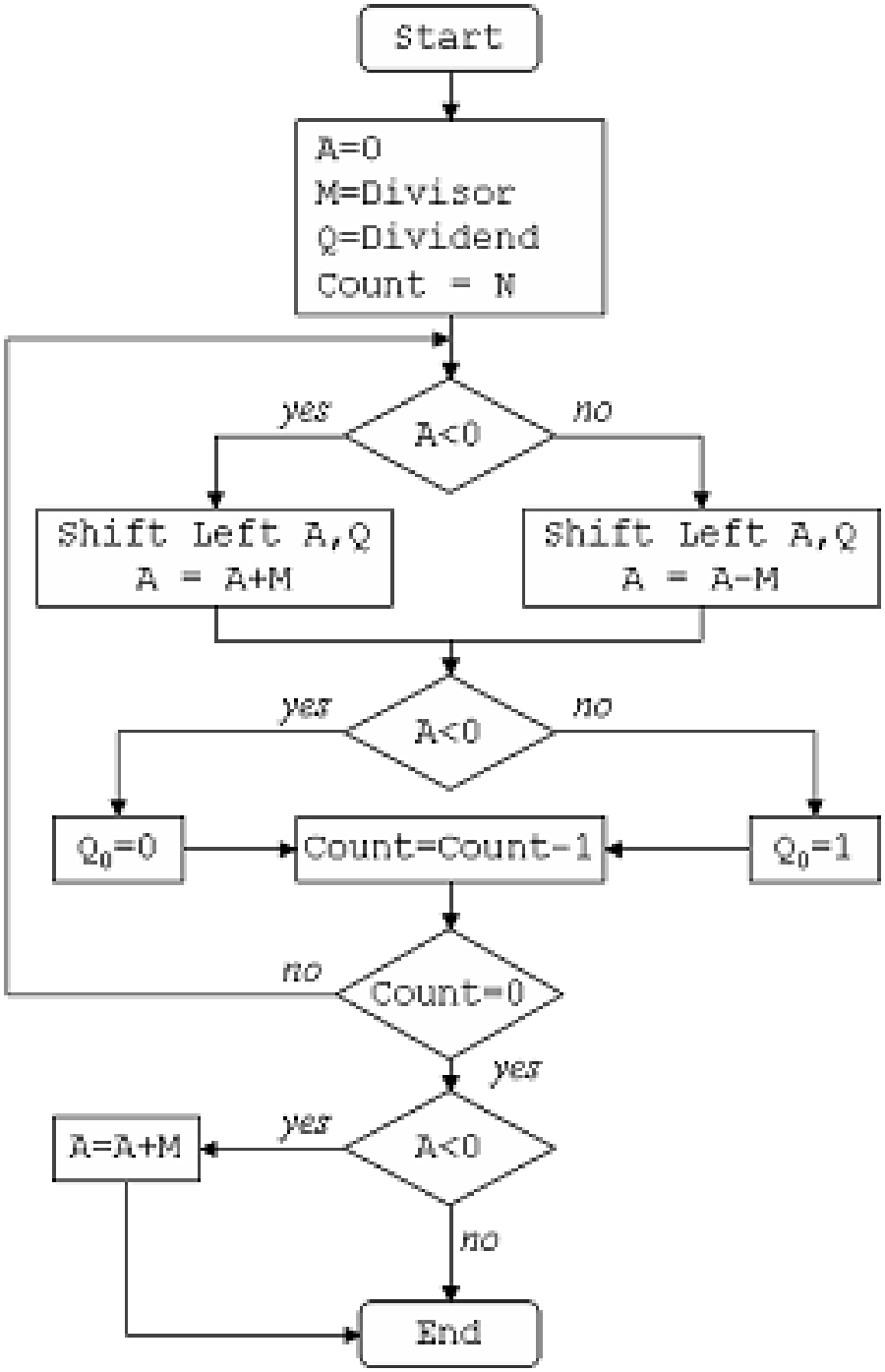
Books/ Journals/ Websites referred:

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.

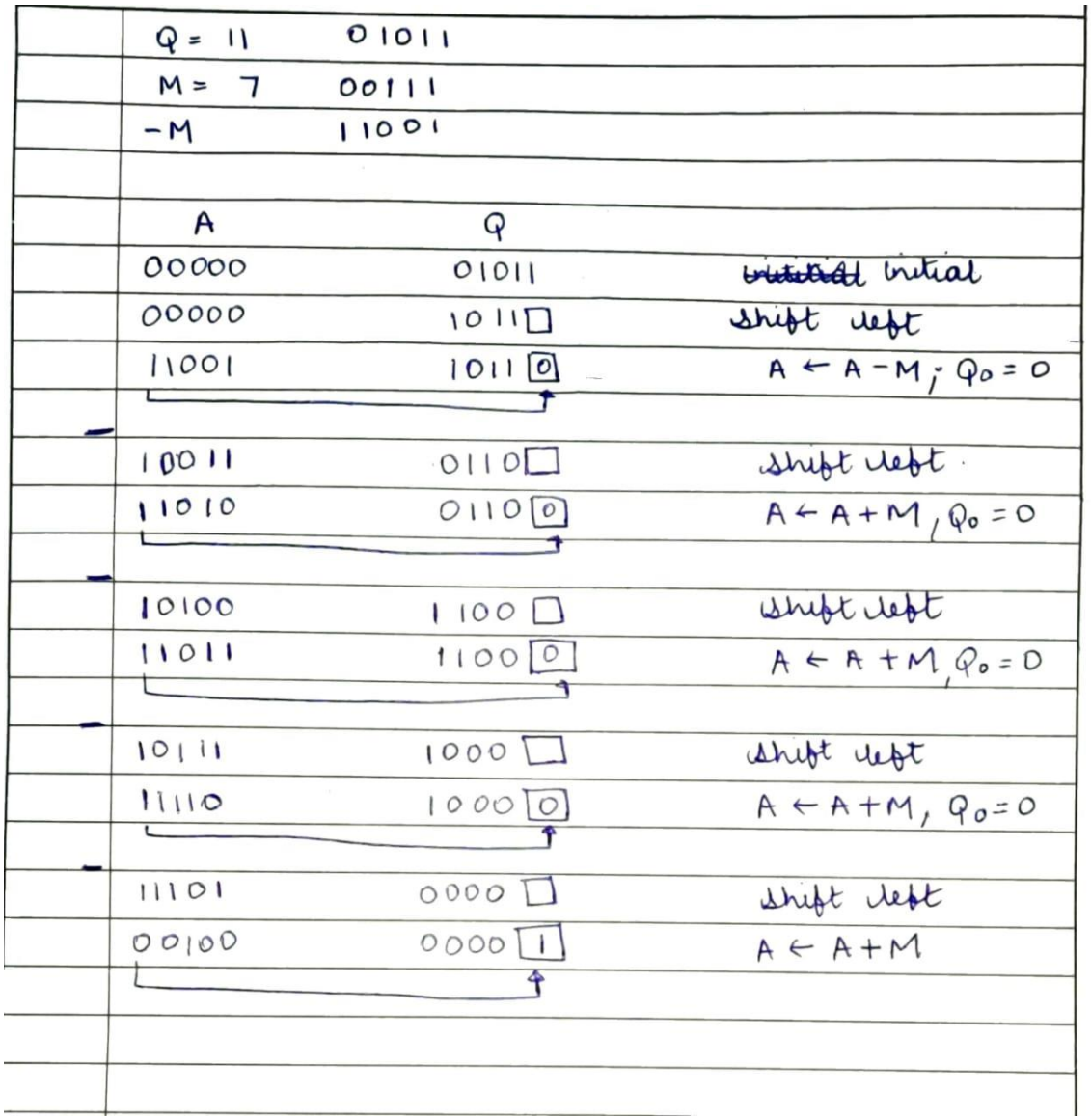
Pre Lab/ Prior Concepts:

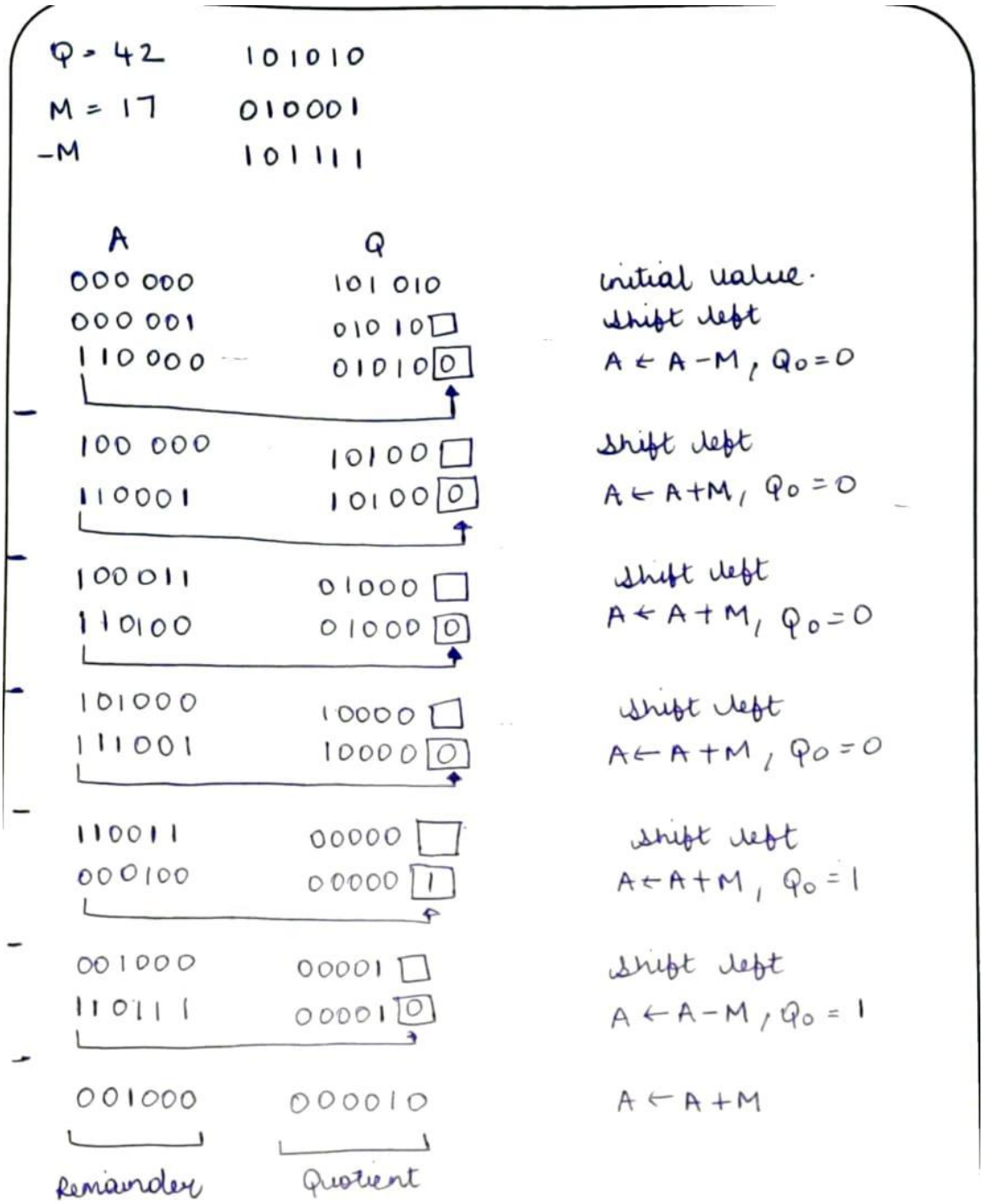
The Non Restoring algorithm works with any combination of positive and negative numbers.

Flowchart for Non Restoring of Division:



Example: (Handwritten solved problem needs to uploaded)




Q = 42 101010
M = 17 010001
-M 101111
A
Q
000 000 101 010 initial value.
000 001 010 10□ shift left
110 000 01010 0 A ← A-M, Q0=0
100 000 10100 □ shift left
110001 10100 0 A ← A+M, Q0=0
100011 01000 □ shift left
110100 01000 0 A ← A+M, Q0=0
101000 10000 □ shift left
111001 10000 0 A ← A+M, Q0=0
110011 00000 □ shift left
000100 00000 1 A ← A+M, Q0=1
001000 00001 □ shift left
110111 00001 0 A ← A-M, Q0=1
001000 000010 A ← A+M
Remainder
Quotient

Code:

```c
#include <math.h>
#include <stdio.h>

// NON-RESTORING DIVISION
int main() {
    int a[50], a1[50], b[50], d = 0, i, j;
    int n1, n2, c, k1, k2, n, k, quo = 0, rem = 0;

    // Input the number of bits
    printf("Enter the number of bits\n");
    scanf("%d", &n);

    // Input the divisor and dividend
    printf("Enter the divisor and dividend (n1 n2):\n");
    scanf("%d %d", &n1, &n2);

    // Convert the two numbers to binary representation
    for (c = n - 1; c >= 0; c--) {
        k1 = n1 >> c;
        if (k1 & 1)
            a[n - 1 - c] = 1; // M
        else
            a[n - 1 - c] = 0;

        k2 = n2 >> c;
        if (k2 & 1)
            b[2 * n - 1 - c] = 1; // Q
        else
            b[2 * n - 1 - c] = 0;
    }

    // Compute the two's complement of 'a' (negative M)
    for (i = 0; i < n; i++) {
        if (a[i] == 0)
```

```
            a1[i] = 1;
        else
            a1[i] = 0;
    }

    a1[n - 1] += 1; // Add 1 to get two's complement of 'a'

    // Handle carry propagation in two's complement
    if (a1[n - 1] == 2) {
        for (i = n - 1; i > 0; i--) {
            if (a1[i] == 2) {
                a1[i - 1] += 1;
                a1[i] = 0;
            }
        }
    }
    if (a1[0] == 2)
        a1[0] = 0;

    // Initialize 'A' with zeros
    for (i = 0; i < n; i++) {
        b[i] = 0;
    }

    printf("A\tQ\tPROCESS\n");

    for (i = 0; i < 2 * n; i++) {
        if (i == n)
            printf("\t");

        printf("%d", b[i]);
    }
    printf("\n");

    for (k = 0; k < n; k++) { // n iterations
        for (j = 0; j < 2 * n - 1; j++) // Left shift
        {
```

```
        b[j] = b[j + 1];
    }

    for (i = 0; i < 2 * n - 1; i++) {
        if (i == n)
            printf("\t");
        printf("%d", b[i]);
    }
    printf("_");

    printf("\tLEFT SHIFT\n");

    if (b[0] == 0) {
        for (i = n - 1; i >= 0; i--) // A = A - M
        {
            b[i] += a1[i];
            if (i != 0) {
                if (b[i] == 2) {
                    b[i - 1] += 1;
                    b[i] = 0;
                }
                if (b[i] == 3) {
                    b[i - 1] += 1;
                    b[i] = 1;
                }
            }
        }
        if (b[0] == 2)
            b[0] = 0;
        if (b[0] == 3)
            b[0] = 1;

        for (i = 0; i < 2 * n - 1; i++) {
            if (i == n)
                printf("\t");
            printf("%d", b[i]);
        }
```

```
        printf("_");

        printf("\tA-M\n");
    } else {
        for (j = n - 1; j >= 0; j--) // A = A + M
        {
            b[j] += a[j];
            if (j != 0) {
                if (b[j] == 2) {
                    b[j - 1] += 1;
                    b[j] = 0;
                }
                if (b[j] == 3) {
                    b[j - 1] += 1;
                    b[j] = 1;
                }
            }
            if (b[0] == 2)
                b[0] = 0;
            if (b[0] == 3)
                b[0] = 1;
        }

        for (i = 0; i < 2 * n - 1; i++) {
            if (i == n)
                printf("\t");
            printf("%d", b[i]);
        }
        printf("_");

        printf("\tA+M\n");
    }

    if (b[0] == 0) {
        b[2 * n - 1] = 1;
        for (i = 0; i < 2 * n; i++) {
            if (i == n)
```

```
                printf("\t");
            printf("%d", b[i]);
        }

        printf("\tQ0=1\n");
    }

    if (b[0] == 1) {
        b[2 * n - 1] = 0;
        for (i = 0; i < 2 * n; i++) {
            if (i == n)
                printf("\t");
            printf("%d", b[i]);
        }

        printf("\tQ0=0\n");
    }
}

// Handle the final iteration when b[0] == 1 (A = A + M)
if (b[0] == 1) {
    for (j = n - 1; j >= 0; j--) // A = A + M
    {
        b[j] += a[j];
        if (j != 0) {
            if (b[j] == 2) {
                b[j - 1] += 1;
                b[j] = 0;
            }
            if (b[j] == 3) {
                b[j - 1] += 1;
                b[j] = 1;
            }
        }
        if (b[0] == 2)
            b[0] = 0;
        if (b[0] == 3)
```

```
                b[0] = 1;
        }

        for (i = 0; i < 2 * n; i++) {
            if (i == n)
                printf("\t");
            printf("%d", b[i]);
        }

        printf("\tA+M\n");
    }

    printf("\n");

    // Calculate the quotient and remainder
    for (i = n; i < 2 * n; i++) {
        quo += b[i] * pow(2, 2 * n - 1 - i);
    }
    for (i = 0; i < n; i++) {
        rem += b[i] * pow(2, n - 1 - i);
    }

    // Output the results
    printf("The quotient of the two numbers is %d\nThe remainder is
%d", quo, rem);

    printf("\n");
    return 0;
}
```

Output:

```
Enter the number of bits
3
Enter the divisor and dividend (n1 n2):
3
6
A       Q       PROCESS
000     110
001     10_     LEFT SHIFT
110     10_     A-M
110     100     Q0=0
101     00_     LEFT SHIFT
000     00_     A+M
000     001     Q0=1
000     01_     LEFT SHIFT
101     01_     A-M
101     010     Q0=0
000     010     A+M

The quotient of the two numbers is 2
The remainder is 0
```

Conclusion

Successfully executed and coded the algorithm for non-restoring division. In this experiment, Non-Restoring Division Algorithm is executed with the help of C++ programming.

The advantage of Non-Restoring Division over Restoring Division is better understood.

Post Lab Descriptive Questions

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

Non-restoring division uses the digit set {−1, 1} for the quotient digits instead of {0, 1}. Non-Restoring Division when implemented in hardware, there is only one decision and addition/subtraction per quotient bit; there is no restoring step after the subtraction, which potentially cuts down the numbers of operations by up to half and

lets it be executed faster.
Restoring method: you add the divisor back, and put 0 as your next quotient digit
Non-restoring method: you don't do that - you keep negative remainder and a digit 1, and basically correct things by a supplementary addition afterwards.

Date: _____ **Signature of faculty in-charge**