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 Booth's Multiplication Algorithm.

AIM: Booth's Algorithm for Multiplication

Expected OUTCOME of Experiment: (Mention CO/CO's attained here)

CO 2-Detail working of the arithmetic logic unit and its sub modules

---

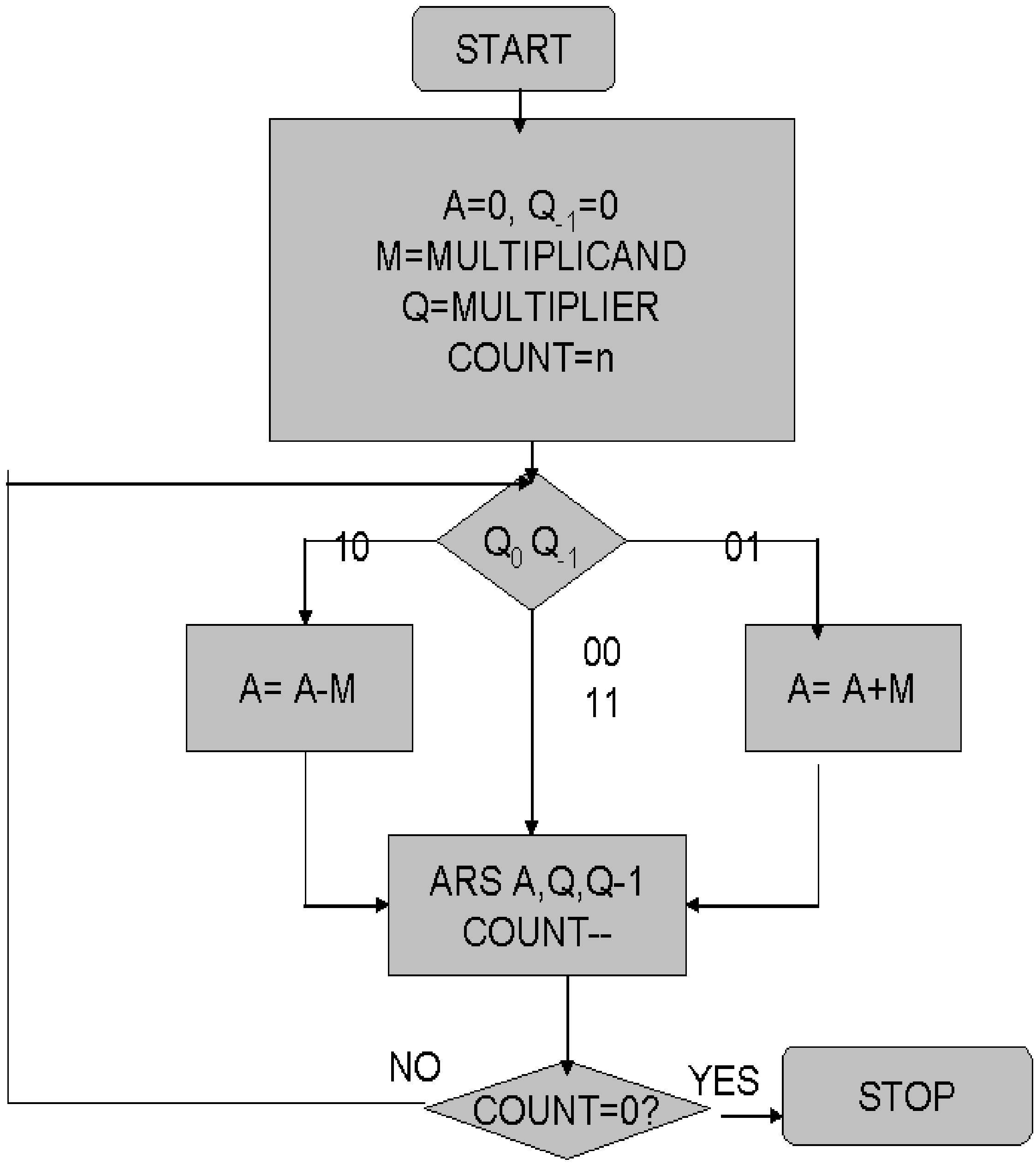
Books/ Journals/ Websites referred:

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:

It is a powerful algorithm for signed number multiplication which generates a 2n bit product and treats both positive and negative numbers uniformly. Also the efficiency of the algorithm is good due to the fact that, block of 1's and 0's are skipped over and subtraction/addition is only done if pair contains 10 or 01

Flowchart:


START
A=0, Q-1=0
M=MULTIPLICAND
Q=MULTIPLIER
COUNT=n
Q0 Q-1
10
01
00
11
A= A-M
A= A+M
ARS A,Q,Q-1
COUNT--
NO
COUNT=0?
YES
STOP

Design Steps:

1. Start
2. Get the multiplicand (M) and Multiplier (Q) from the user
3. Initialize A= Q_{-1} =0
4. Convert M and Q into binar
5. Compare Q_0 and Q_{-1} and perform the respective operation.

Q_0 Q_{-1}	**Operation**
00/11	Arithmetic right shift
01	A+M and Arithmetic right shift
10	A-M and Arithmetic right shift

6. Repeat steps 5 till all bits are compared
7. Convert the result to decimal form and display
8. End

CODE:

```
import java.util.*;

public class BoothAlgo {
    int ONE[];
    int A[];
    int M[];
    int Q[];
    int negM[];
    int q, bits;

    BoothAlgo (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.q = 0;
        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;
        }
    }

    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 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;
        }
    }

    void shiftRight ()
    {
        this.q = this.Q [0];

        for (int i = 0; i < (Q.length-1); i++) {
            this.Q[i] = this.Q[i+1];
        }

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

        for (int i = 0; i < (A.length-1); i++) {
            this.A[i] = this.A[i+1];
        }
    }

    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 i, bits = 0;
        Scanner scanner = new Scanner (System.in);

        System.out.println ("Enter two numbers to multiply using Booth's Algor
ithm");
        int n1 = scanner.nextInt ();
        int n2 = scanner.nextInt ();

        int max = n1 > n2 ? n1 : n2;

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

        System.out.println("bits : " + bits);
        BoothAlgo booth = new BoothAlgo (bits);

        scanner.close();

        booth.toBinary(n1, 'M');
        booth.toBinary(n2, 'Q');
        booth.calcNegM();

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

        for (i = bits; i >= 1; i--)
        {
            if (booth.Q[0] == 1 && booth.q == 0)
            {
```

```
                booth.binaryAdd(booth.A, booth.negM, 'A');

                booth.displayArr(booth.A);
                System.out.print("    ");
                booth.displayArr(booth.Q);
                System.out.println("    " + booth.q + "    " + i);
            }
            else if (booth.Q[0] == 0 && booth.q == 1)
            {
                booth.binaryAdd(booth.A, booth.M, 'A');

                booth.displayArr(booth.A);
                System.out.print("    ");
                booth.displayArr(booth.Q);
                System.out.println("    " + booth.q + "    " + i);
            }

            booth.shiftRight();

            booth.displayArr(booth.A);
            System.out.print("    ");
            booth.displayArr(booth.Q);
            System.out.println("    " + booth.q + "    " + i);
        }

        int result[] = new int [(2*bits)];

        for (i = 0; i < (2*bits); i++)
        {
            if (i < bits)
                result [i] = booth.Q [i];
            else
                result [i] = booth.A [i-bits];
        }

        int res = booth.toDecimal(result);

        System.out.print("\nResult in binary: ");
        booth.displayArr(result);
        System.out.println("\nResult in decimal: " + res);
    }
}
```

OUTPUT :

```
PS D:\Projects>  & 'c:\Users\YASH\.vscode\extensions\vscjava.vscode-java
-11.0.12.7-hotspot\bin\java.exe' '-agentlib:jdwp=transport=dt_socket,ser
sers\YASH\AppData\Roaming\Code\User\workspaceStorage\b5faf066d3a39646bea
o'
Enter two numbers to multiply using Booth's Algorithm
5
7
bits : 4
 Q: 0111
 M: 0101
-M: 1011
1011   0111   0   4
1101   1011   1   4
1110   1101   1   3
1111   0110   1   2
0100   0110   1   1
0010   0011   0   1

Result in binary: 00100011
Result in decimal: 35
PS D:\Projects>
```

Example: (Handwritten solved problem needs to be uploaded)

→ M : 7
Q : 3

M = 0111
Q = 0011
−M = 1001

A	Q	Q_{-1}	M	OPERATIONS	
0000	0011	0	0111		
1001	0011	0	0111	A = A − M	First Cycle
1100	1001	1	0111	Shift	
1110	0100	1	0111	Shift	Second Cycle
0101	0100	1	0111	A = A + M	Third Cycle
0000	1010	0	0111	Shift	
0001	0001	0	0111	Shift	Forth Cycle

Answer :- A and Q → $(00010101)_2$
$= (21)_{10}$

Conclusion:

With the help of this experiment we were able to learn:

1. Booth's Algorithm for Multiplication
2. Its flowchart and working
3. How to make different functions such as decimal to binary, complement, adding to binary numbers ,displaying it, right shift etc
4. Booths Algorithm was studied and implemented in JAVA programming language.

Post Lab Descriptive Questions

1. **Explain advantages and disadvantages of Booth's algorithm.**

Advantages:

- It reduces the number of partial products to be compressed in a carry-save
- added tree.
- It is very fast for multiplications having long operands (>16 bits).
- Not many conversions are required (only 2's complement used).

Disadvantages:

- A lot of complexity is involved in the circuit to generate a partial product
- bitin Booth's encoding.
- Algorithm is inefficient for isolated 1's.
- Inconvenient for designing a synchronous multiplier.

2. **Is Booth's recoding better than Booth's algorithm? Justify**

Booth's Recoding:

1. Booth multiplication is a technique that allows for smaller,
2. faster multiplication circuits, by recoding the numbers that are multiplied. It is
3. the standard technique used in chip design, and provides significant

4. improvements over the "long multiplication" technique.
5. Booth’s Algorithm: Booth's multiplication algorithm is a multiplication
6. algorithm that multiplies two signed binary numbers in two's complement
7. notation

Booth’s recoding is better because

1. • When a partial product of 0 occurs can skip addition and just shift
2. • Doesn’t help multiplier where datapaths go through adder
3. • Does help design for asynchronous implementations or
4. microprogramming since shift is faster than addition

Date: 04/09/2021 **Signature of faculty in-charge**