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

Batch: B2	**Roll No.: 16010124107**

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)

Books/ Journals/ Websites referred:

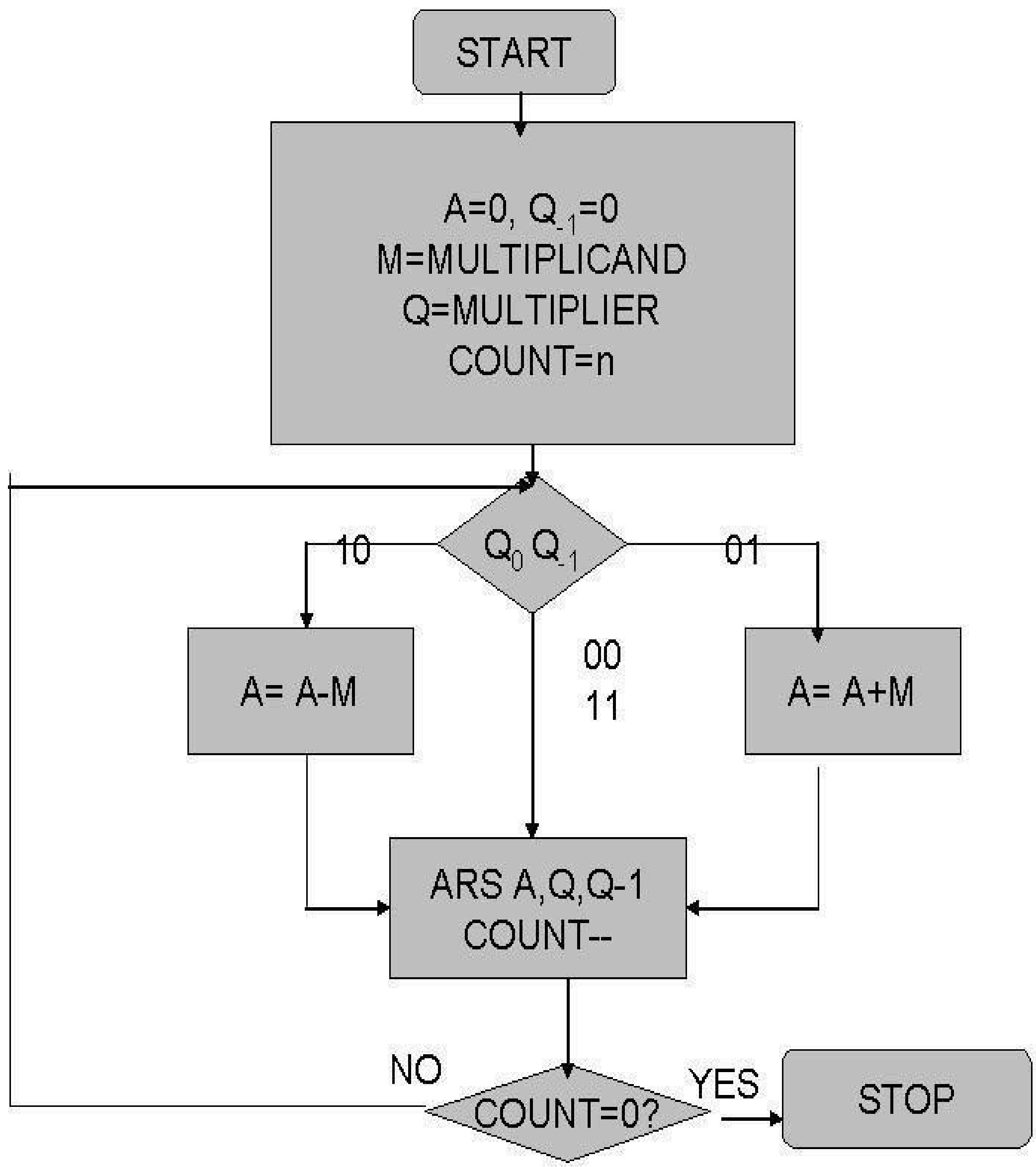
1. Carl Hamacher, Zvonko Vranesic and Safwat Zaky, "Computer Organization", Fifth Edition, TataMcGraw-Hill.

2.
3. William Stallings, "Computer Organization and Architecture: Designing for Performance", Eighth Edition, Pearson.
4.
5. 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 binary
5. Compare Q0 and Q-1 and perform the respective operation.

Q0 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:

```
#include <bits/stdc++.h>
using namespace std;

// Add function
void add(int ac[], int x[], int qrn) {
    int c = 0;
    for (int i = 0; i < qrn; i++) {
        ac[i] = ac[i] + x[i] + c;
        if (ac[i] > 1) {
            ac[i] %= 2;
            c = 1;
        } else c = 0;
    }
}
```

```cpp
// One's complement + 1 (Two's complement)
void complement(int a[], int n) {
    int x[8] = {1}; // binary 1
    for (int i = 0; i < n; i++)
        a[i] = (a[i] + 1) % 2;
    add(a, x, n);
}

// Right shift
void rightShift(int ac[], int qr[], int &qn, int qrn) {
    int temp = ac[0];
    qn = qr[0];
    cout << "\t\trightShift\t";
    for (int i = 0; i < qrn - 1; i++) {
        ac[i] = ac[i + 1];
        qr[i] = qr[i + 1];
    }
    qr[qrn - 1] = temp;
}

// Display
void display(int ac[], int qr[], int qrn) {
    for (int i = qrn - 1; i >= 0; i--) cout << ac[i];
    cout << "\t";
    for (int i = qrn - 1; i >= 0; i--) cout << qr[i];
}

// Booth's algorithm
void boothAlgorithm(int br[], int qr[], int mt[], int qrn, int sc) {
    int qn = 0, ac[10] = {0};
    int temp = 0;
    cout << "qn\tq[n+1]\t\tBR\t\tAC\tQR\t\tsc\n";
    cout << "\t\t\tinitial\t\t";
    display(ac, qr, qrn);
    cout << "\t\t" << sc << "\n";

    while (sc != 0) {
        cout << qr[0] << "\t" << qn;
```

```
        if ((qn + qr[0]) == 1) {
            if (temp == 0) {
                add(ac, mt, qrn);
                cout << "\t\tA = A - BR\t";
                for (int i = qrn - 1; i >= 0; i--) cout << ac[i];
                temp = 1;
            } else if (temp == 1) {
                add(ac, br, qrn);
                cout << "\t\tA = A + BR\t";
                for (int i = qrn - 1; i >= 0; i--) cout << ac[i];
                temp = 0;
            }
            cout << "\n\t";
            rightShift(ac, qr, qn, qrn);
        } else if (qn - qr[0] == 0)
            rightShift(ac, qr, qn, qrn);

        display(ac, qr, qrn);
        cout << "\t";
        sc--;
        cout << "\t" << sc << "\n";
    }
}

int main() {
    int brn, qrn;
    string mStr, qStr;

    cout << "Enter number of bits: ";
    cin >> brn;
    qrn = brn;

    int br[10], mt[10], qr[10];

    cout << "Enter multiplicand (" << brn << "-bit binary): ";
    cin >> mStr;
    reverse(mStr.begin(), mStr.end());
    for (int i = 0; i < brn; i++) br[i] = mStr[i] - '0';
    for (int i = 0; i < brn; i++) mt[i] = br[i];
```

```cpp
    complement(mt, brn);
    reverse(br, br + brn);

    cout << "Enter multiplier (" << qrn << "-bit binary): ";
    cin >> qStr;
    reverse(qStr.begin(), qStr.end());
    for (int i = 0; i < qrn; i++) qr[i] = qStr[i] - '0';

    boothAlgorithm(br, qr, mt, qrn, qrn);

    cout << "\nResult = ";
    for (int i = qrn - 1; i >= 0; i--) cout << qr[i];
    cout << endl;

    return 0;
}
```

Output:

Input:

```
5 01010 00010
```

Expected Output:

```
10100
```

Received Output:

```
Enter number of bits: Enter multiplicand (5-bit binary): Enter multiplier (5-bit binary): qn     q[n+1]
BR              AC      QR              sc
                        initial         00000   00010           5
0       0               rightShift      00000   00001           4
1       0               A = A - BR      10110
                        rightShift      11011   00000           3
0       1               A = A + BR      00101
                        rightShift      00010   10000           2
0       0               rightShift      00001   01000           1
0       0               rightShift      00000   10100           0

Result = 10100
```

Example: (Handwritten solved problem needs to be uploaded)

Page No.
Date

Q) 10 X 2

10 = 01010 = Q

2 = 00010 = M

00010
11101
+1
11110 = 2's Cof

A	Q	Q_{-1}	count	
00000	01010	0	5	
00000	00101	0	4	10, A-M
11110	00101			
11111	00010	1	3	01, A+M
00001	00010	1		
00000	10001	0	2	10, A-M
11110	10001	0		
11111	01000	1	1	01, A+M
00001	01000	1		
00000	10100	0	0	stop

Ans : 10100

Conclusion:

Booth’s algorithm is used to quicky operate on bit multiplication for both negative and positive integers. It implements right shifting and arithmetic alteration of multiplicand to generate correct output.

Post Lab Descriptive Questions:

Question: Explain advantages and disadvantages of Booth’s algorithm.

Advantages:

1. Efficient for both positive and negative numbers using 2’s complement.
2. Reduces operations for runs of 1s in multiplier (e.g., 1111 needs only one subtraction and one

addition).
3. Shifting is easy to implement in hardware circuits.

Disadvantages:

1. For multipliers with alternating 0s and 1s (e.g., 101010), no reduction in operations.
2. Needs extra logic to handle q-1, decision-making, and 2’s complement.
3. Requires tracking of extra bit (q-1) and sign extension during shifts.

Question: Is Booth’s recoding better than Booth’s algorithm? Justify

Booth’s recoding is better than Booth’s algorithm because it reduces the number of partial products in multiplication, leading to faster and more efficient hardware implementation. While Booth’s algorithm is simpler, recoding improves speed and performance, especially in high-speed multipliers.

Date: 25/07/2025