Experiment No. 7
Implement Booth's algorithm using c-programming
Name: James Lewis
Roll Number: 28
Date of Performance: 04-09-2024
Date of Submission:

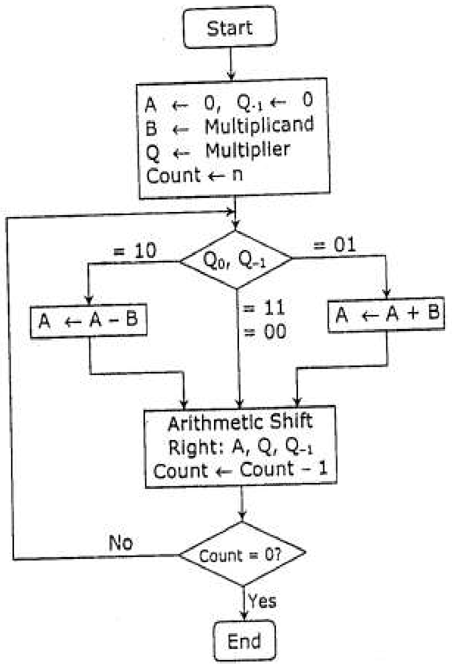
Aim: To implement Booth's algorithm using c-programming.

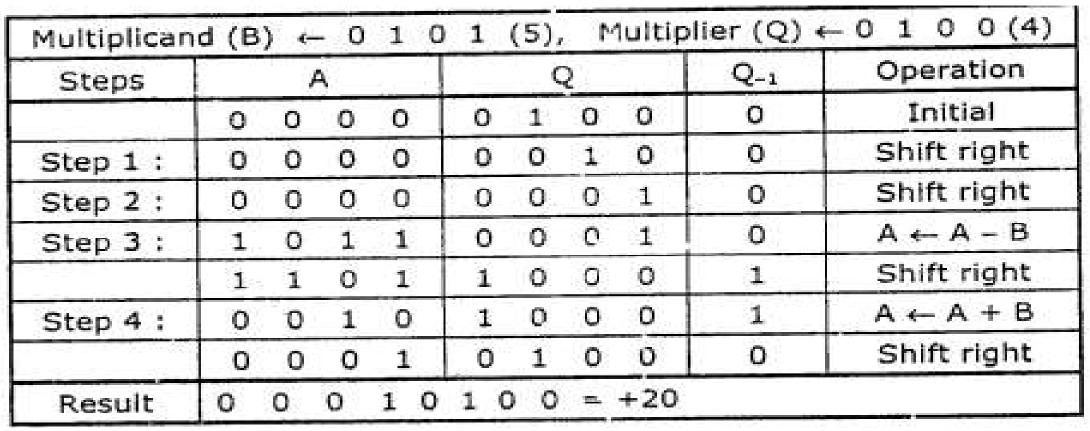
Objective -

1. To understand the working of Booths algorithm.
2. To understand how to implement Booth's algorithm using c-programming.

Theory:

Booth's algorithm is a multiplication algorithm that multiplies two signed binary numbers in 2's complement notation. Booth used desk calculators that were faster at shifting than adding and created the algorithm to increase their speed.
The algorithm works as per the following conditions :
1. If Qn and Q_{-1} are same i.e. 00 or 11 perform arithmetic shift by 1 bit.
2. If Qn Q_{-1} = 10 do A= A - B and perform arithmetic shift by 1 bit.
3. If Qn Q_{-1} = 01 do A= A + B and perform arithmetic shift by 1 bit.





Multiplicand (B) ← 0 1 0 1 (5), Multiplier (Q) ← 0 1 0 0 (4)

Steps	A	Q	Q_{-1}	Operation
	0 0 0 0	0 1 0 0	0	Initial
Step 1 :	0 0 0 0	0 0 1 0	0	Shift right
Step 2 :	0 0 0 0	0 0 0 1	0	Shift right
Step 3 :	1 0 1 1	0 0 0 1	0	A ← A – B
	1 1 0 1	1 0 0 0	1	Shift right
Step 4 :	0 0 1 0	1 0 0 0	1	A ← A + B
	0 0 0 1	0 1 0 0	0	Shift right
Result	0 0 0 1 0 1 0 0 = +20			

Program:

```
#include <stdio.h>
#include <stdlib.h>

void toBinary(int num, int *arr, int size) {
     for (int i = size - 1; i >= 0; i--) {
          arr[i] = num & 1;
          num >>= 1;
     }
}

void arithmeticShiftRight(int *A, int *Q, int *Q_minus_1, int
size) {
     *Q_minus_1 = Q[size - 1];
     for (int i = size - 1; i > 0; i--) {
          Q[i] = Q[i - 1];
     }
     Q[0] = A[size - 1];

     for (int i = size - 1; i > 0; i--) {
          A[i] = A[i - 1];
     }
     A[0] = A[1];
}

void addBinary(int *A, int *B, int size) {
     int carry = 0;
```

```
    for (int i = size - 1; i >= 0; i--) {
        int sum = A[i] + B[i] + carry;
        A[i] = sum % 2;
        carry = sum / 2;
    }
}

void subtractBinary(int *A, int *B, int size) {
    int borrow = 0;
    for (int i = size - 1; i >= 0; i--) {
        int sub = A[i] - B[i] - borrow;
        if (sub < 0) {
            A[i] = sub + 2;
            borrow = 1;
        } else {
            A[i] = sub;
            borrow = 0;
        }
    }
}

void twosComplement(int *arr, int size) {
    for (int i = 0; i < size; i++) {
        arr[i] = arr[i] == 0 ? 1 : 0;
    }
    int carry = 1;
    for (int i = size - 1; i >= 0; i--) {
        int sum = arr[i] + carry;
        arr[i] = sum % 2;
        carry = sum / 2;
    }
}

void printBinary(int *arr, int size) {
    for (int i = 0; i < size; i++) {
        printf("%d", arr[i]);
    }
}

int main() {
    int multiplier, multiplicand;
    printf("Enter multiplicand (decimal): ");
    scanf("%d", &multiplicand);
    printf("Enter multiplier (decimal): ");
    scanf("%d", &multiplier);

    int size = 8;
    int A[size], Q[size], M[size], M_neg[size], Q_minus_1 =
0;

    toBinary(abs(multiplier), Q, size);
    toBinary(abs(multiplicand), M, size);
    for (int i = 0; i < size; i++) A[i] = 0;

    for (int i = 0; i < size; i++) M_neg[i] = M[i];
    twosComplement(M_neg, size);

    for (int step = 0; step < size; step++) {
        if (Q[size - 1] == 1 && Q_minus_1 == 0) {
            addBinary(A, M, size);
        } else if (Q[size - 1] == 0 && Q_minus_1 == 1) {
            addBinary(A, M_neg, size);
        }
        arithmeticShiftRight(A, Q, &Q_minus_1, size);
    }

    printf("Result (binary): ");
    printBinary(A, size);
    printBinary(Q, size);
    printf("\n");

    int result = 0;
    int isNegative = A[0];

    if (isNegative) {
        for (int i = 0; i < size; i++) {
            A[i] = A[i] == 0 ? 1 : 0;
            Q[i] = Q[i] == 0 ? 1 : 0;
        }
        addBinary(Q, (int[]) {
            0, 0, 0, 0, 0, 0, 0, 1
        }, size);
```

```
          addBinary(A, (int[]) {
               0, 0, 0, 0, 0, 0, 0, 1
          }, size);
     }

     for (int i = 0; i < size; i++) {
          result = result * 2 + Q[i];
     }

     if (isNegative) result = -result;

     printf("Result (decimal): %d\n", result);

     return 0;
}
```

Output:

```
Enter multiplicand (decimal): 5
Enter multiplier (decimal): 4
Result (binary): 000000010100
Result (decimal): 20
```

Conclusion -

The program successfully implements Booth's algorithm for multiplying two signed binary numbers using 2's complement notation. It efficiently handles multiplication by performing arithmetic shifts and conditional additions or subtractions, resulting in the correct output for both binary and decimal forms.