Experiment No. 7
Implement Booth's algorithm using c-programming
Name: Sumit V. Patel
Roll Number: 40
Date of Performance: 04-09-2024
Date of Submission:

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

Objective -

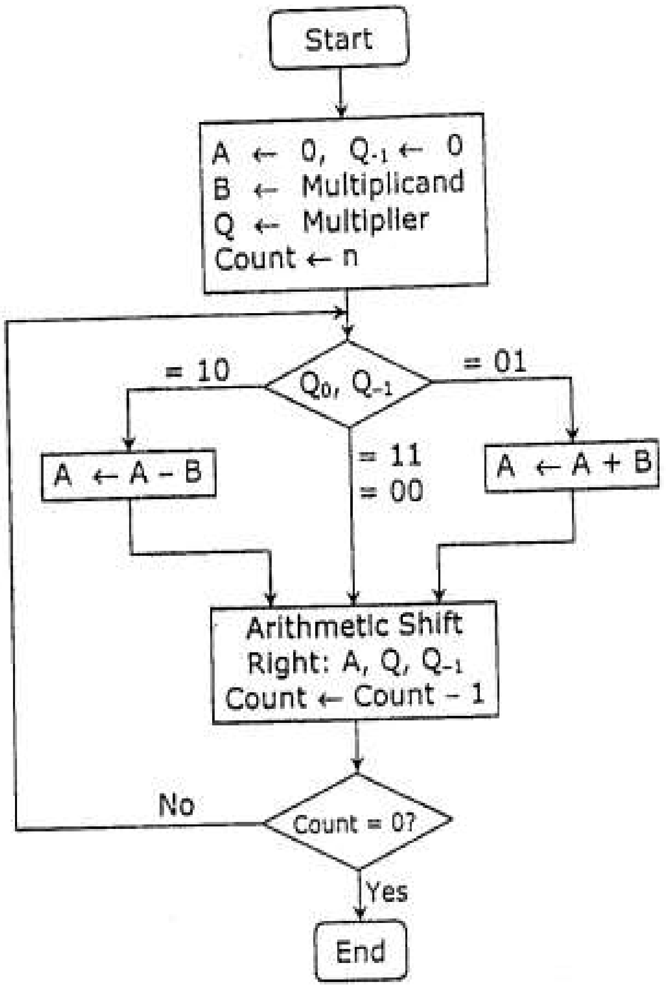
1. To understand the working of Booths algorithm.
2. To understand how to implement Booth's algorithm using c-programming.

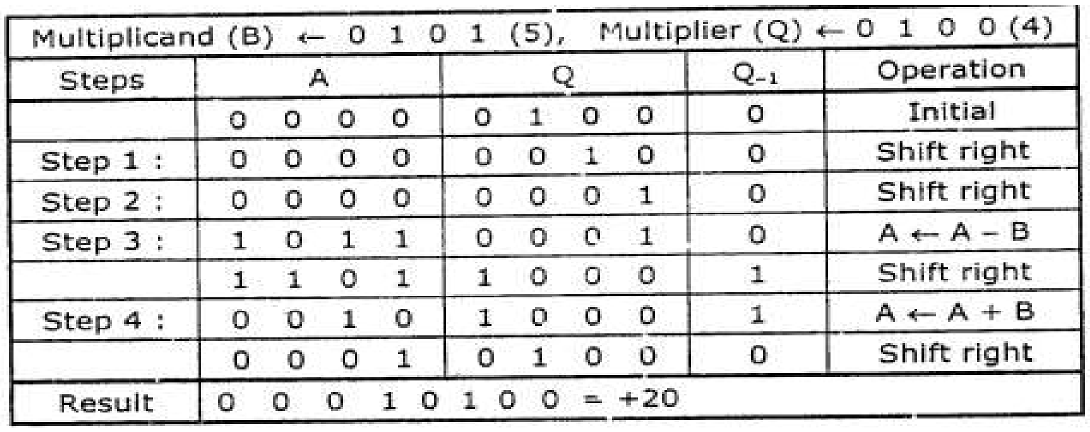
Theory:

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			

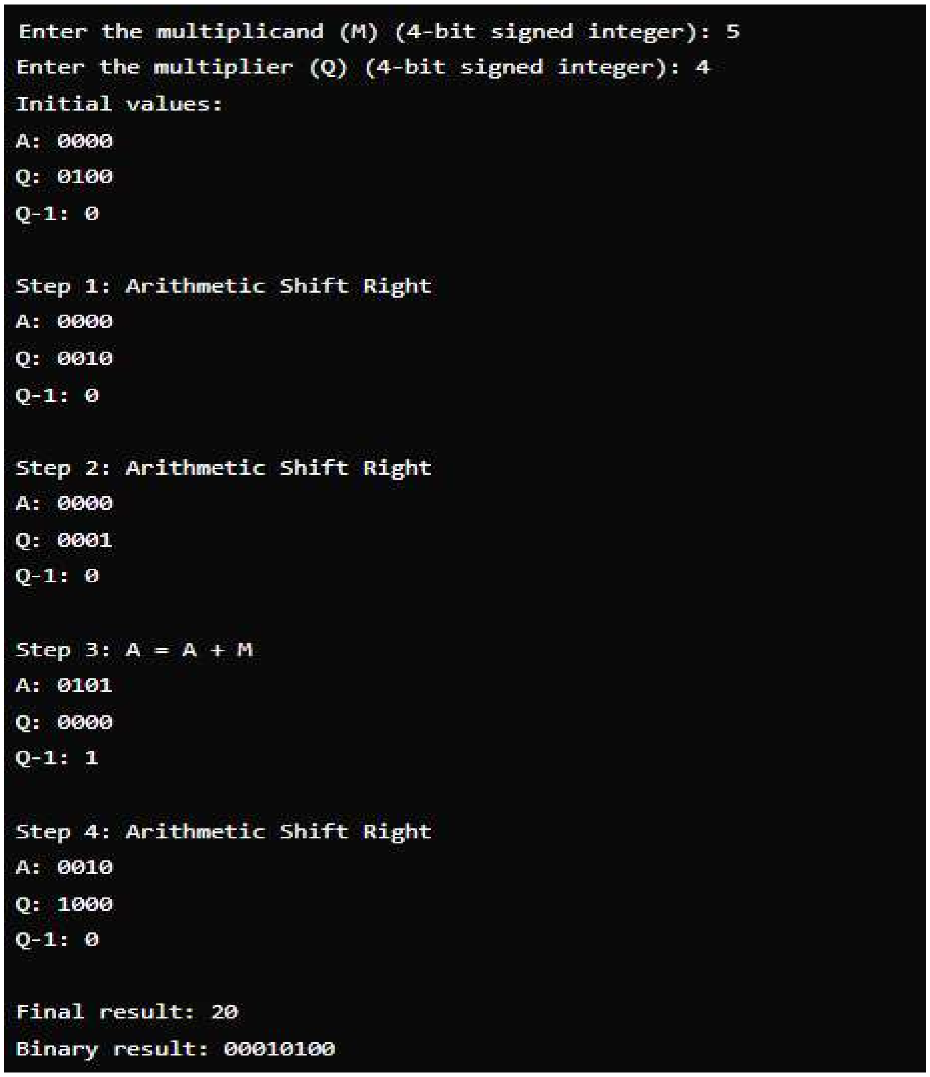
Program:
```
#include <stdio.h>
#include <stdint.h>

#define BIT_SIZE 4

void printBinary(int num, int size) {
   for (int i = size - 1; i >= 0; i--) {
      printf("%d", (num >> i) & 1);
   }
   printf("\n");
}

void boothMultiplication(int8_t M, int8_t Q) {
   int8_t A = 0;
   int8_t Q_1 = 0;
   int8_t M_neg = -M;

   printf("Initial values:\n");
   printf("A: ");
   printBinary(A, BIT_SIZE);
   printf("Q: ");
   printBinary(Q, BIT_SIZE);
   printf("Q-1: %d\n", Q_1);
   printf("\n");

   for (int i = 0; i < BIT_SIZE; i++) {
      int Qn = Q & 1;

      if (Qn == Q_1) {
         printf("Step %d: Arithmetic Shift Right\n", i + 1);
      } else if (Qn == 0 && Q_1 == 1) {
         printf("Step %d: A = A - M\n", i + 1);
         A = A + M_neg;
      } else if (Qn == 1 && Q_1 == 0) {
         printf("Step %d: A = A + M\n", i + 1);
         A = A + M;
      }

      Q_1 = Q & 1;
      Q = (A & 1) << (BIT_SIZE - 1) | (Q >> 1);
```

```
        A = (A >> 1) & ((1 << BIT_SIZE) - 1);

        printf("A: ");
        printBinary(A, BIT_SIZE);
        printf("Q: ");
        printBinary(Q, BIT_SIZE);
        printf("Q-1: %d\n", Q_1);
        printf("\n");
    }

    int result = ((int)A << BIT_SIZE) | (Q & ((1 << BIT_SIZE) - 1));
    printf("Final result: %d\n", result);
    printf("Binary result: ");
    printBinary(result, BIT_SIZE * 2);
}

int main() {
    int8_t M, Q;

    printf("Enter the multiplicand (M) (4-bit signed integer): ");
    scanf("%hhd", &M);
    printf("Enter the multiplier (Q) (4-bit signed integer): ");
    scanf("%hhd", &Q);

    boothMultiplication(M, Q);

    return 0;
}
```

Output:

```
Enter the multiplicand (M) (4-bit signed integer): 5
Enter the multiplier (Q) (4-bit signed integer): 4
Initial values:
A: 0000
Q: 0100
Q-1: 0

Step 1: Arithmetic Shift Right
A: 0000
Q: 0010
Q-1: 0

Step 2: Arithmetic Shift Right
A: 0000
Q: 0001
Q-1: 0

Step 3: A = A + M
A: 0101
Q: 0000
Q-1: 1

Step 4: Arithmetic Shift Right
A: 0010
Q: 1000
Q-1: 0

Final result: 20
Binary result: 00010100
```

Screenshot -



```
Enter the multiplicand (M) (4-bit signed integer): 5
Enter the multiplier (Q) (4-bit signed integer): 4
Initial values:
A: 0000
Q: 0100
Q-1: 0

Step 1: Arithmetic Shift Right
A: 0000
Q: 0010
Q-1: 0

Step 2: Arithmetic Shift Right
A: 0000
Q: 0001
Q-1: 0

Step 3: A = A + M
A: 0101
Q: 0000
Q-1: 1

Step 4: Arithmetic Shift Right
A: 0010
Q: 1000
Q-1: 0

Final result: 20
Binary result: 00010100
```

Conclusion -

Booth's algorithm is a powerful technique for performing binary multiplication, particularly useful for handling both positive and negative integers. By employing a combination of arithmetic shifts and conditional additions or subtractions, it efficiently computes the product while managing sign extensions. The algorithm's strength lies in its ability to handle signed numbers through two's complement representation, making it versatile for various computing applications. Its iterative process ensures that the multiplication is performed correctly within a fixed number of steps. Overall, Booth's algorithm optimizes binary multiplication and is a valuable tool in digital arithmetic.