Experiment No. 7
Implement Booth's algorithm using c-programming
Name: Patil Pranjal Keshav
Roll Number: 45
Date of Performance:
Date of Submission:

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

Objective -

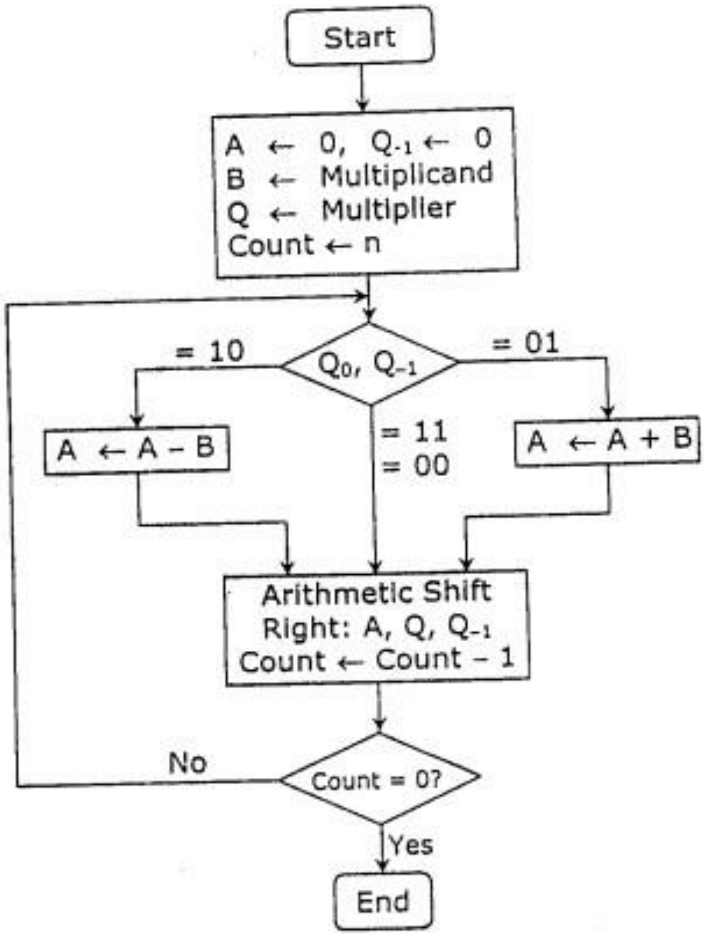
1. To understand the working of Booths algorithm.
2. To understand how to implement Booth's algorithm using c-programming.

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

void booth(int A, int B, int bits)
{
	int M = A;
	// Multiplicand
	int Q = B;
	// Multiplier
	int A_reg = 0;
	 // A register initialized to 0
	int Qn = 0;
	// Previous Q's bit
	int steps = 0;

	// Calculate B' + 1
```

```
int B_complement = (~B + 1) & ((1 << bits) - 1);

printf("Binary Equivalents are:\n");
printf("A = ");
printBinary(A, bits);
printf("\nB = ");
printBinary(B, bits);
printf("\nB' + 1 = ");
printBinary(B_complement, bits);
printf("\n");

while (steps < bits)
{
        printf("-->\n");
        if (Qn == 0 && (Q & 1) == 1)
        {
                // Case 10: A = A - M
                printf("SUB B: ");
                A_reg = (A_reg - M) & ((1 << bits) - 1);
        }
        else if (Qn == 1 && (Q & 1) == 0)
        {
                 // Case 01: A = A + M
                printf("ADD B: ");
                A_reg = (A_reg + M) & ((1 << bits) - 1);
        }
        else
        {
                printf("No operation\n");
        }

        // Print current values
        printf("AR-SHIFT: ");
        printBinary(A_reg, bits);
        printBinary(Q, bits);
        printf("\n");

        // Perform arithmetic right shift
        Qn = Q & 1;
        // Store the least significant bit of Q
```

```
            Q = (A_reg & 1) | (Q >> 1);
            // Shift Q right and insert LSB of A_reg
            A_reg >>= 1;
            // Shift A_reg right

            // Print after shift
            printf("AR-SHIFT: ");
            printBinary(A_reg, bits);
            printBinary(Q, bits);
            printf("\n");

            steps++;
        }

        // Combine A_reg and Q to get the product
        int product = (A_reg << bits) | Q;
        printf("Product is = ");
        printBinary(product, bits * 2);
        printf("\nExpected product = %d\n", A * B);
}

int main()
{
        int A, B;

        printf("Enter two numbers to multiply:\n");
        printf("Both must be less than 16\n");
        printf("Enter A: ");
        scanf("%d", &A);
        printf("Enter B: ");
        scanf("%d", &B);

        if (A < 0 || A >= 16 || B < 0 || B >= 16)
        {
        printf("Both numbers must be less than 16.\n");
        return 1;
        }

        int bits = 5; // Assuming 5 bits for numbers less than 16
        booth(A, B, bits);
```

```
	return 0;
}
```

Output:

```
Enter two numbers to multiply:
Both must be less than 16
Enter A: 3
Enter B: 5
Binary Equivalents are:
A = 00011
B = 00101
B' + 1 = 11011
-->
SUB B: AR-SHIFT: 1110100101
AR-SHIFT: 0111000011
-->
No operation
AR-SHIFT: 0111000011
AR-SHIFT: 0011100001
-->
No operation
AR-SHIFT: 0011100001
AR-SHIFT: 0001100001
-->
No operation
AR-SHIFT: 0001100001
AR-SHIFT: 0000100001
-->
No operation
AR-SHIFT: 0000100001
AR-SHIFT: 0000000001
Product is = 0000000001
Expected product = 15
```

Conclusion –

Booth's algorithm is an efficient way to perform multiplication of binary numbers, especially for signed integers. The provided C implementation demonstrates the algorithm's logic clearly, allowing for both addition and subtraction based on the bits of the multiplier, thereby optimizing the multiplication process. This algorithm is particularly useful in computer architecture and digital systems, enhancing the efficiency of arithmetic operations.