



**Vidyavardhini's College of Engineering and Technology**

**Department of Artificial Intelligence & Data Science**

Experiment No. 7
Implement Booth's algorithm using c-programming
Name: Tarun Premnarayan Pathak
Roll Number: 38
Date of Performance:
Date of Submission:



# Vidyavardhini's College of Engineering and Technology

## Department of Artificial Intelligence & Data Science

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

**Objective -**

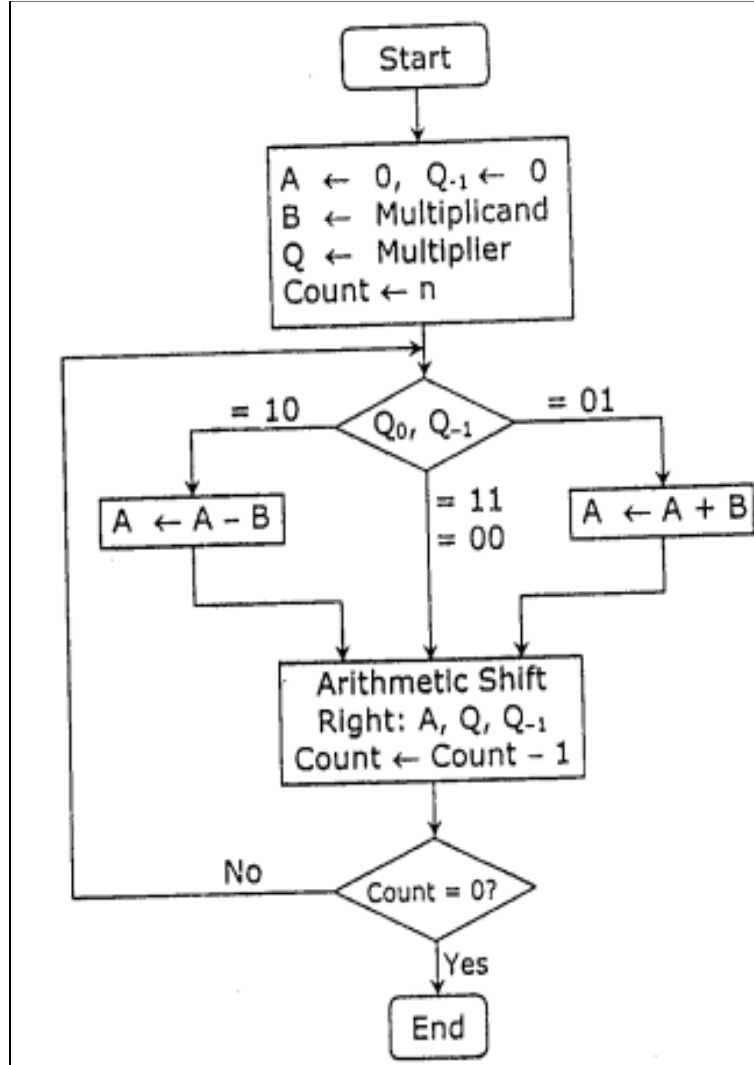
1. To understand the working of Booths algorithm.
2. To understand how to implement Booth's algorithm using c-programming.

**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  $Q_n$  and  $Q_{-1}$  are same i.e. 00 or 11 perform arithmetic shift by 1 bit.
2. If  $Q_n Q_{-1} = 10$  do  $A = A - B$  and perform arithmetic shift by 1 bit.
3. If  $Q_n Q_{-1} = 01$  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:**



# Vidyavardhini's College of Engineering and Technology

## Department of Artificial Intelligence & Data Science

```
#include <math.h>
```

```
int a = 0, b = 0, c = 0, a1 = 0, b1 = 0, com[5] = { 1, 0, 0, 0, 0};  
int anum[5] = {0}, anumcp[5] = {0}, bnum[5] = {0};  
int acomp[5] = {0}, bcomp[5] = {0}, pro[5] = {0}, res[5] = {0};
```

```
void binary(){  
    a1 = fabs(a);  
    b1 = fabs(b);  
    int r, r2, i, temp;  
    for (i = 0; i < 5; i++){  
        r = a1 % 2;  
        a1 = a1 / 2;  
        r2 = b1 % 2;  
        b1 = b1 / 2;  
        anum[i] = r;  
        anumcp[i] = r;  
        bnum[i] = r2;  
        if(r2 == 0){  
            bcomp[i] = 1;  
        }  
        if(r == 0){  
            acomp[i] = 1;  
        }  
    }  
}
```

```
c = 0;  
for ( i = 0; i < 5; i++){  
    res[i] = com[i] + bcomp[i] + c;  
    if(res[i] >= 2){  
        c = 1;  
    }  
    else  
        c = 0;  
    res[i] = res[i] % 2;  
}  
for (i = 4; i >= 0; i--){  
    bcomp[i] = res[i];  
}
```



```
if (a < 0){
    c = 0;
    for (i = 4; i >= 0; i--){
        res[i] = 0;
    }
    for ( i = 0; i < 5; i++){
        res[i] = com[i] + acomp[i] + c;
        if (res[i] >= 2){
            c = 1;
        }
        else
            c = 0;
        res[i] = res[i]%2;
    }
    for (i = 4; i >= 0; i--){
        anum[i] = res[i];
        anumcp[i] = res[i];
    }
}

if(b < 0){
    for (i = 0; i < 5; i++){
        temp = bnum[i];
        bnum[i] = bcomp[i];
        bcomp[i] = temp;
    }
}

void add(int num[]){
    int i;
    c = 0;
    for ( i = 0; i < 5; i++){
        res[i] = pro[i] + num[i] + c;
        if (res[i] >= 2){
            c = 1;
        }
        else{
            c = 0;
        }
    }
}
```



```
    }
    res[i] = res[i]%2;
}
for (i = 4; i >= 0; i--){
    pro[i] = res[i];
    printf("%d",pro[i]);
}
printf(":");
for (i = 4; i >= 0; i--){
    printf("%d", anumcp[i]);
}
}

void arshift(){
    int temp = pro[4], temp2 = pro[0], i;
    for (i = 1; i < 5 ; i++){
        pro[i-1] = pro[i];
    }
    pro[4] = temp;
    for (i = 1; i < 5 ; i++){
        anumcp[i-1] = anumcp[i];
    }
    anumcp[4] = temp2;
    printf("\nAR-SHIFT: ");
    for (i = 4; i >= 0; i--){
        printf("%d",pro[i]);
    }
    printf(":");
    for(i = 4; i >= 0; i--){
        printf("%d", anumcp[i]);
    }
}

void main(){
    int i, q = 0;
    printf("\t\tBOOTH'S MULTIPLICATION ALGORITHM");
    printf("\nEnter two numbers to multiply: ");
    printf("\nBoth must be less than 16");
    //simulating for two numbers each below 16
    do{
```



# Vidyavardhini's College of Engineering and Technology

## Department of Artificial Intelligence & Data Science

```
printf("\nEnter A: ");
scanf("%d",&a);
printf("Enter B: ");
scanf("%d", &b);
}while(a >=16 || b >=16);

printf("\nExpected product = %d", a * b);
binary();
printf("\n\nBinary Equivalents are: ");
printf("\nA = ");
for (i = 4; i >= 0; i--){
    printf("%d", anum[i]);
}
printf("\nB = ");
for (i = 4; i >= 0; i--){
    printf("%d", bnum[i]);
}
printf("\nB' + 1 = ");
for (i = 4; i >= 0; i--){
    printf("%d", bcomp[i]);
}
printf("\n\n");
for (i = 0; i < 5; i++){
    if (anum[i] == q){
        printf("\n-->");
        arshift();
        q = anum[i];
    }
    else if(anum[i] == 1 && q == 0){
        printf("\n-->");
        printf("\nSUB B: ");
        add(bcomp);
        arshift();
        q = anum[i];
    }
    else{
        printf("\n-->");
        printf("\nADD B: ");
        add(bnum);
    }
}
```



```
        arshift();
        q = anum[i];
    }
}

printf("\nProduct is = ");
for (i = 4; i >= 0; i--){
    printf("%d", pro[i]);
}
for (i = 4; i >= 0; i--){
    printf("%d", anumcp[i]);
}
}
```

**Output:**

**OUTPUT:-**

**BOOTH'S MULTIPLICATION ALGORITHM**

Enter two numbers to multiply:

Both must be less than 16

Enter A: 10

Enter B: 2

Expected product = 20

Binary Equivalentents are:

A = 01010

B = 00010

B'+ 1 = 11110

-->

AR-SHIFT: 00000:00101

-->

SUB B: 11110:00101

AR-SHIFT: 11111:00010

-->

ADD B: 00001:00010

AR-SHIFT: 00000:10001

-->

SUB B: 11110:10001

AR-SHIFT: 11111:01000

-->





# Vidyavardhini's College of Engineering and Technology

## Department of Artificial Intelligence & Data Science

**ADD B: 00001:01000**

**AR-SHIFT: 00000:10100**

**Product is = 0000010100**

### **Conclusion -**

This experiment with Booth's algorithm has highlighted its significance in optimizing binary multiplication. Booth's algorithm efficiently reduces the number of partial products and minimizes the overall number of operations required for multiplication. This not only enhances computational speed but also reduces hardware complexity. Booth's algorithm is a powerful tool for optimizing multiplication processes and is an essential concept in digital arithmetic. Our experiment has successfully demonstrated its practical applicability in computer architecture and digital circuit design.