FR. Conceicao Rodrigues College of Engineering
Department of Computer Engineering

7. Write a program for Binary Multiplication.(Booth's Algorithm)

1. Course, Subject & Experiment Details

Academic Year	2023-24	Estimate d Time	Experiment No. 7.– 02 Hours
Course & Semester	S.E. (Computers) – Sem. III	Subject Name	Digital Logic & Computer Organization and Architecture
Chapter No.	2	Chapter Title	Data Representation and Arithmetic algorithms
Experiment Type	Software	Subject Code	CSC304

Rubrics

Roll No	Date of Performance	Timeline (2)	Practical Skill & Applied Knowledge **(4)**	Output (4)	Total (10)
	Date of Submission				

2. Aim & Objective of Experiment

- Learn to implement multiplication by using addition and shifts (Booth's algorithm).
- To study and implement n bit Binary Multiplication using C/Java/ Python

3. Problem Statement

Write a C/ Java/ Python program to implement Booth's Algorithm for Multiplication..

.

4. Brief Theoretical Description

With unsigned multiplication there is no need to take the sign of the number into consideration. However in signed multiplication the same process cannot be applied because the signed number is in a 2's compliment form which would yield an incorrect result if multiplied in a similar fashion to unsigned multiplication. That's where Booth's algorithm comes in. Booth's algorithm preserves the sign of the result.

Algorithm:

1. Multiplier and multiplicand are placed in the Q and M registers.

2. One bit register(Q_{-1}) placed logically to the right of least significant bit of the register Q0.

3. The result of multiplication will appear in A and Q registers.

4. A and Q_{-1} are initialized to zero.

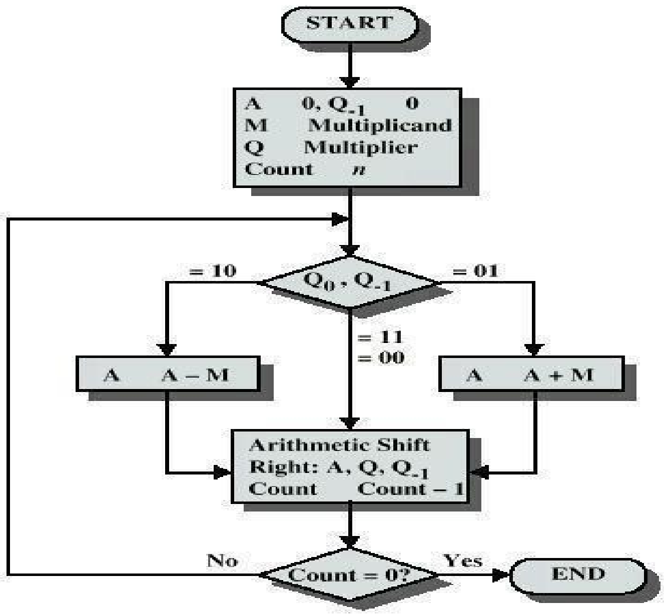
5. If the combination of two bits is same(0-0 or 1-1), then all the bits of A , Q and Q_{-1} registers are shifted to right by 1 bit.

6. If two bits differ , then the multiplicand is added to or subtracted from register A depending upon whether two bits are 0-1 or 1-0.

7. Following the addition or subtraction, the right shift occurs.

8. In either case, the right shift is such that the leftmost bit of A, namely A_{n-1}, not only is shifted into A_{n-2}, but also remains in A_{n-1}. This is called **Arithmetic Shift,** because it preserves the sign bit.

Flowchart of Booth's Algorithm:



5. *Attach the program*

```
#include <stdio.h>
#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];

     }

}
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;
```

```c
    for (i = 4; i >= 0; i--){

          }

    for(i = 4; i >= 0; 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{
        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){
```

```
                arshift();
                q = anum[i];
            }
            else if(anum[i] == 1 && q == 0){


               add(bcomp);
               arshift();
               q = anum[i];
            }
            else{

               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:-

```
Both must be less than 16
Enter A: 4
Enter B: 2

Expected product = 8

Binary Equivalents are:
A = 00100
B = 00010
B'+ 1 = 11110


Product is = 0000001000
PS C:\Users\Mark Lopes\Desktop\New folder (3)>
```

6. Conclusion:

Hence,we have successfully implemented booth's algorithm by C language.

7. Post-lab:

1. Write advantages of Booth's algorithm.

Q.1 Write advantages of Booth's algorithm.

→ 1. Faster multiplication: It speeds up the process by minimizing the computing steps.

2. Simplifies hardware design:- It requires fewer hardware resources than traditional multiplication methods.

3. Efficient for signed numbers:- The algorithm is designed specifically for multiplying signed binary numbers, making it a more efficent method for multiplication of signed numbers.