Batch: D-2 Roll No.: 16010122323

Experiment / assignment / tutorial No._2_____

Grade: AA / AB / BB / BC / CC / CD /DD

Signature of the Staff In-charge with date

TITLE: To study and implement Booth's Multiplication Algorithm.

AIM: Booth's Algorithm for Multiplication

Expected OUTCOME of Experiment: (Mention CO/CO's attained here)

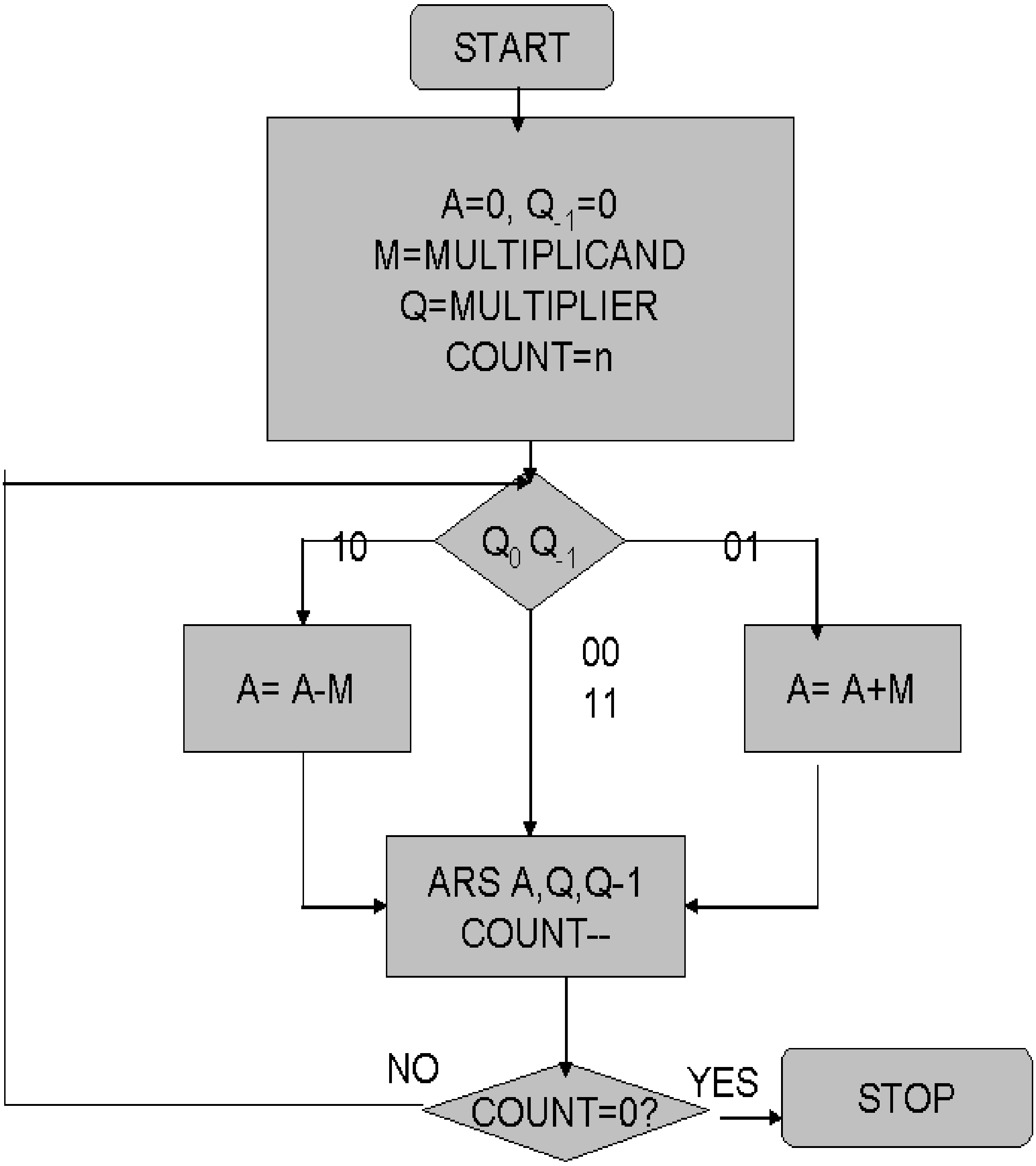
Books/ Journals/ Websites referred:

1. Carl Hamacher, Zvonko Vranesic and Safwat Zaky, "Computer Organization", Fifth Edition, TataMcGraw-Hill.
2. William Stallings, "Computer Organization and Architecture: Designing for Performance", Eighth Edition, Pearson.
3. Dr. M. Usha, T. S. Srikanth, "Computer System Architecture and Organization", First Edition, Wiley-India.

Pre Lab/ Prior Concepts:

It is a powerful algorithm for signed number multiplication which generates a 2n bit product and treats both positive and negative numbers uniformly. Also the efficiency of the algorithm is good due to the fact that, block of 1's and 0's are skipped over and subtraction/addition is only done if pair contains 10 or 01

Flowchart:



Design Steps:

1. Start
2. Get the multiplicand (M) and Multiplier (Q) from the user
3. Initialize A= Q_{-1} =0
4. Convert M and Q into binar

5. Compare Q_0 and Q_{-1} and perform the respective operation.

Q_0 Q_{-1}	**Operation**
00/11	Arithmetic right shift
01	A+M and Arithmetic right shift
10	A-M and Arithmetic right shift

6. Repeat steps 5 till all bits are compared

7. Convert the result to decimal form and display

8. End

CODE :

```
#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;
         }
    }
  //part for two's complementing
  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];
   }
   //in case of negative inputs
   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(){//for arithmetic shift right
    int temp = pro[4], temp2 = pro[0], i;
    for (i = 1; i < 5  ; i++){//shift the MSB of product
       pro[i-1] = pro[i];
    }
    pro[4] = temp;
    for (i = 1; i < 5  ; i++){//shift the LSB of product
        anumcp[i-1] = anumcp[i];
    }
    anumcp[4] = temp2;
    printf("\nAR-SHIFT: ");//display together
    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{
        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){//just shift for 00 or 11
                printf("\n-->");
                arshift();
                q = anum[i];
            }
            else if(anum[i] == 1 && q == 0){//subtract and shift for 10
               printf("\n-->");
               printf("\nSUB B: ");
               add(bcomp);//add two's complement to implement subtraction
               arshift();
               q = anum[i];
            }
            else{//add ans shift for 01
               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 :

```
input
                BOOTH'S MULTIPLICATION ALGORITHM
Enter two numbers to multiply:
Both must be less than 16
Enter A: 5
Enter B: 6

Expected product = 30

Binary Equivalents are:
A = 00101
B = 00110
B'+ 1 = 11010


-->
SUB B: 11010:00101
AR-SHIFT: 11101:00010
-->
ADD B: 00011:00010
AR-SHIFT: 00001:10001
-->
SUB B: 11011:10001
AR-SHIFT: 11101:11000
-->
ADD B: 00011:11000
AR-SHIFT: 00001:11100
-->
AR-SHIFT: 00000:11110
Product is = 0000011110

...Program finished with exit code 0
Press ENTER to exit console.
```

Example: (Handwritten solved problem needs to be uploaded)

Example: 7x3=21 [10101]

M = 7 = 0111

Q = 3 = 0011

	A	Q	Q_{-1}	Operators
1)	0000	0011	0	A − M ⇒ A+2'comp
				0000
				1001
				1001 ⇒ A
	1001	0011	0	
2)	1100	1001	1	shift operators
	1110	0100	1	
3)	1110	0100	1	
	0101	0100	1	A+M ⇒ 1110
				0111
	0010	1010	0	$\boxed{1}$0101
4)	0010	1010	0	
	$\boxed{0001}$	$\boxed{0101}$	0	shift operator

0 0010101

= 21

Conclusion:

The aim of the experiment is verified.

<u>Post Lab Descriptive Questions</u>

1. **Explain advantages and disadvantages of Booth's algorithm.**

- It handles both positive and negative numbers uniformly.
- It achieves efficiency in the number of additions required when the multiplier has a few large blocks of 1's.

2. **Is Booth's recoding better than Booth's algorithm? Justify**

Booth recoding there are reduced number of partial products as there are reduced number of addition and subtraction as compared to Booth's algorithm. In this case Booth recoding is better but as far as implementation is concerned the algorithm is followed.