Batch: A1 **Roll No.: 16010123012**

Experiment / assignment / tutorial No.: 2

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

AIM: Booth's Algorithm for Multiplication

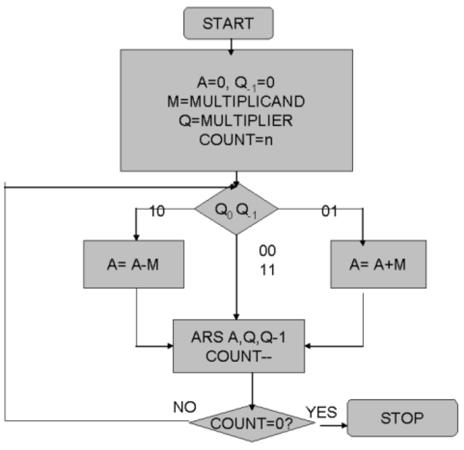
Expected OUTCOME of Experiment: (Mention CO/CO's attained here)

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 binary
5. Compare Q_0 andQ_{-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

<u>Code for Booth's Algorithm in C:</u>

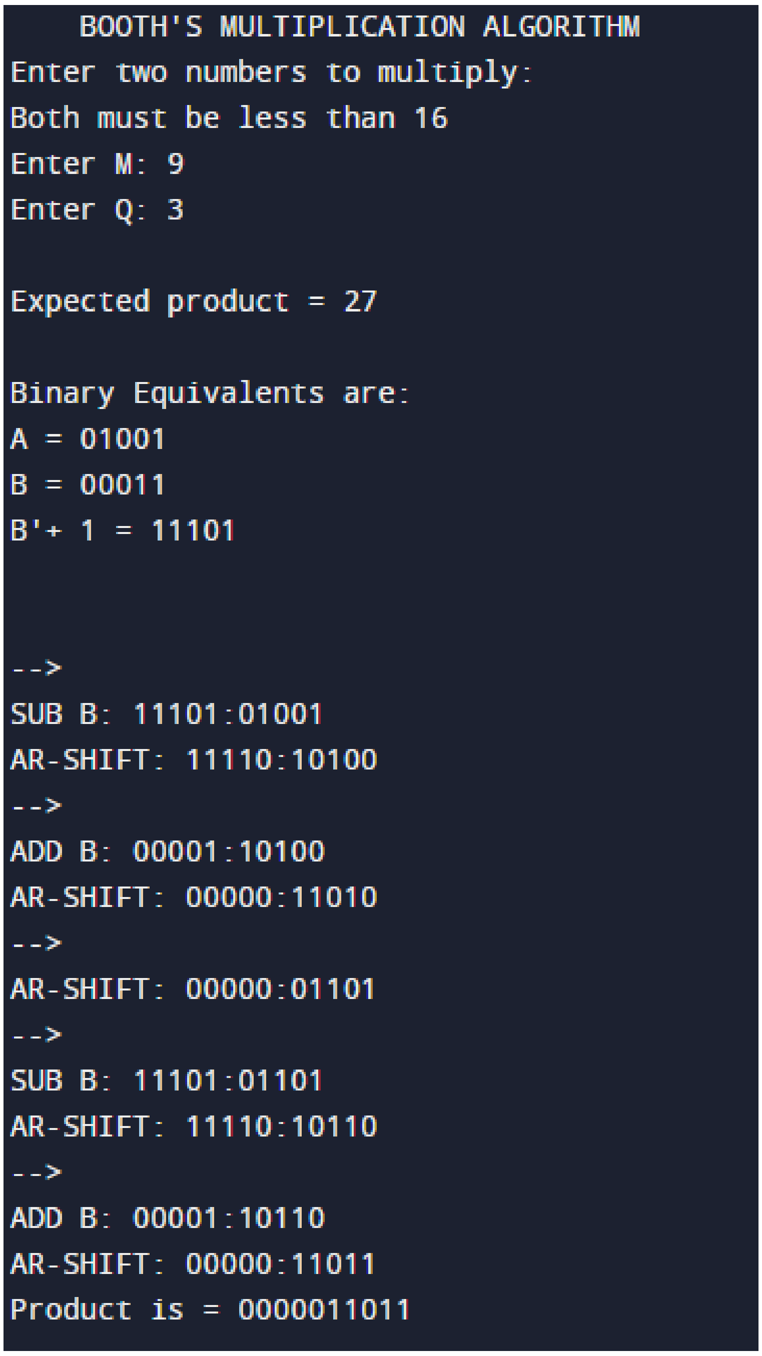
```
#include <stdio.h>
#include <math.h>

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];
            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("\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 M: ");
        scanf("%d",&a);
        printf("Enter Q: ");
        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]);
 }
```

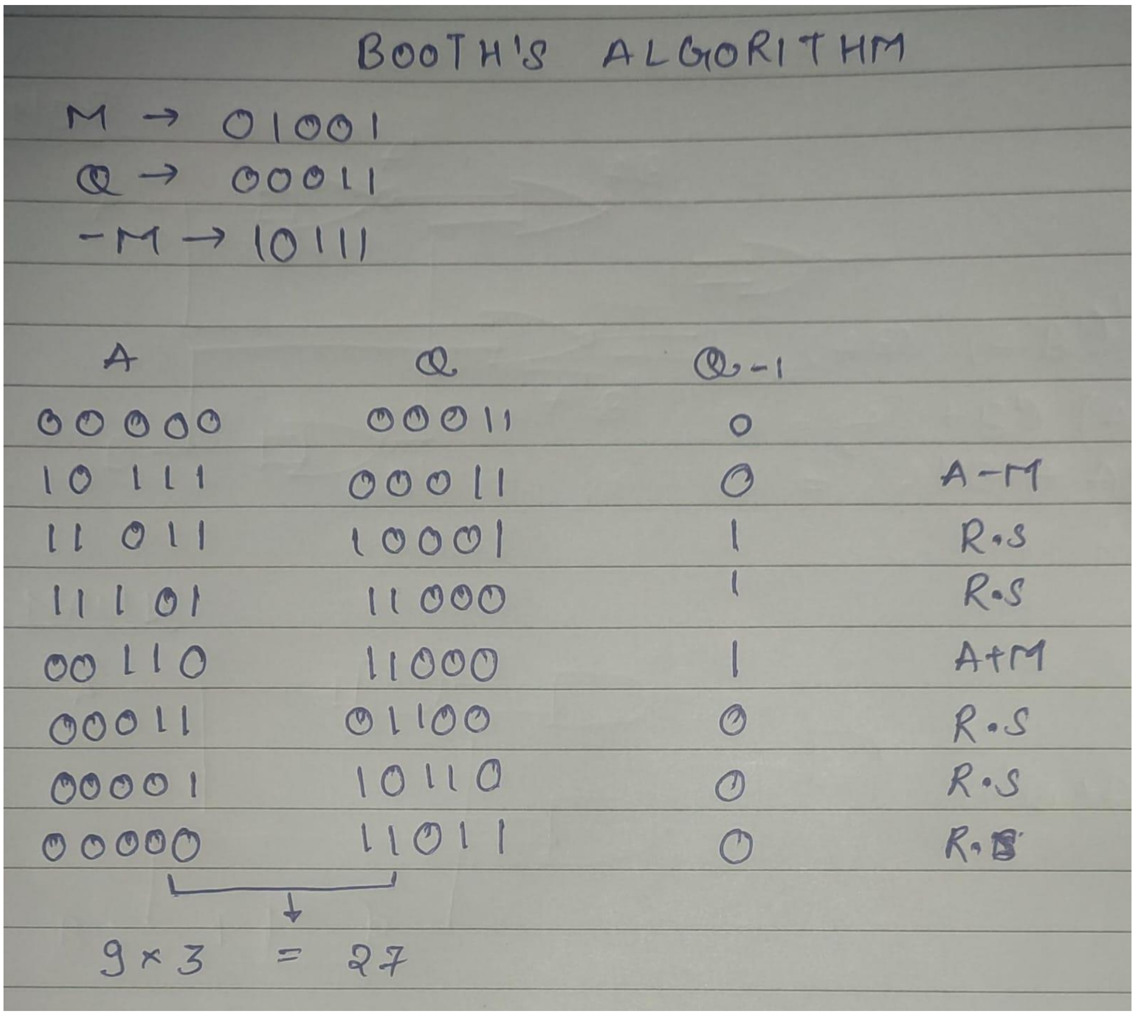
```
    for (i = 4; i >= 0; i--){
        printf("%d", anumcp[i]);
    }
}
```

OUTPUT



```
    BOOTH'S MULTIPLICATION ALGORITHM
Enter two numbers to multiply:
Both must be less than 16
Enter M: 9
Enter Q: 3

Expected product = 27

Binary Equivalents are:
A = 01001
B = 00011
B'+ 1 = 11101



-->
SUB B: 11101:01001
AR-SHIFT: 11110:10100
-->
ADD B: 00001:10100
AR-SHIFT: 00000:11010
-->
AR-SHIFT: 00000:01101
-->
SUB B: 11101:01101
AR-SHIFT: 11110:10110
-->
ADD B: 00001:10110
AR-SHIFT: 00000:11011
Product is = 0000011011
```

Example: (Handwritten solved problem needs to be uploaded)



BOOTH'S ALGORITHM

M → 01001
Q → 00011
-M → 10111

A	Q	Q_{-1}	
00000	00011	0	
10111	00011	0	A-M
11011	10001	1	R.S
11101	11000	1	R.S
00110	11000	1	A+M
00011	01100	0	R.S
00001	10110	0	R.S
00000	11011	0	R.S

9 × 3 = 27

Conclusion: In this experiment we learned about Booth's Algorithm and how to multiply numbers using it. We also verified it using a code that multiplies numbers using Booth's Algorithm.

Post Lab Descriptive Questions

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

Booth's algorithm is a multiplication algorithm designed to perform binary multiplication more efficiently, especially when dealing with signed numbers in two's complement representation.

Advantages of Booth's Algorithm -

1. Efficiency with Certain Data Patterns - Booth's algorithm can be more efficient than the standard multiplication algorithm, especially when the multiplier has large blocks of

consecutive 1s or 0s. This is because the algorithm reduces the number of necessary addition and subtraction operations.

2. Handling Signed Multiplication - Booth's algorithm can handle both positive and negative numbers efficiently due to its use of two's complement representation.

3. Hardware Implementation - The algorithm is well-suited for hardware implementation, where shifts and adds/subtracts can be efficiently managed, leading to faster multiplications.

Disadvantages of Booth's Algorithm –

1. Complexity - Booth's algorithm is more complex to understand and implement compared to straightforward binary multiplication algorithms, especially for those unfamiliar with binary arithmetic and two's complement.

2. Irregularity in Operations - The number of operations required by Booth's algorithm can vary depending on the bit pattern of the multiplier. This irregularity can complicate optimization and timing analysis in hardware design.

3. Overhead in Sign Bit Handling - The handling of the sign bit and the need for extra logic to manage two's complement arithmetic add to the complexity, especially in software implementations.

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

Booth's algorithm and Booth's recoding are both efficient methods for multiplying signed binary numbers, but they each have distinct advantages and disadvantages. Booth's algorithm is simpler to implement and requires fewer hardware resources, making it a good choice for general use. However, it can be less efficient when the multiplier contains long strings of zeros or ones. On the other hand, Booth's recoding, while more complex to implement and requiring more hardware resources, excels in efficiency for multipliers with long strings of zeros or ones by converting these strings into sequences of additions and subtractions. Therefore, Booth's recoding is preferred for such specific cases, whereas Booth's algorithm is more suitable for general purposes due to its simplicity.

Date: 04/08/2024