Batch: B1 **Roll No.: 1711072**

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:

CO 2-Detail working of the arithmetic logic unit and its sub modules

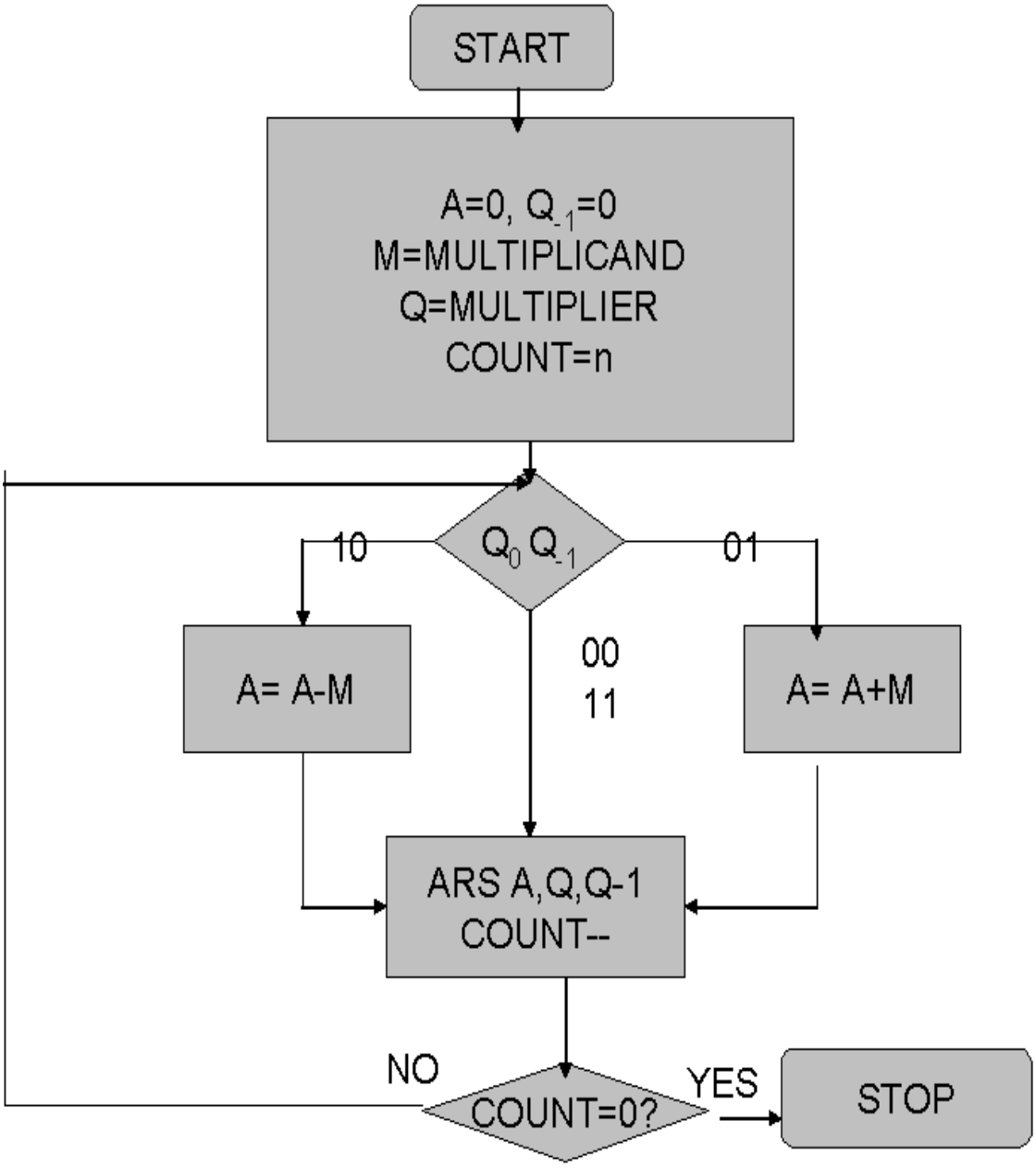
Books/ Journals/ Websites referred:

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:


START
A=0, Q-1=0
M=MULTIPLICAND
Q=MULTIPLIER
COUNT=n
Q0 Q-1
10
01
00
11
A= A-M
A= A+M
ARS A,Q,Q-1
COUNT--
NO
COUNT=0?
YES
STOP

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 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

Implementation (in Java):

```
import java.util.*;
class Main {
  public static void main(String[] args) {
    int M,Q, Q_1=0;
    int BITS=5;
    int count=BITS;
    //int A[]=new int[BITS];
    int M_BIN[]=new int[BITS];
    int Q_BIN[]=new int[BITS];
    int answer[]=new int[BITS];
    int M_TEMP[]=new int[BITS];
    Scanner sc=new Scanner(System.in);
    System.out.println("BOOTH'S ALGORITHM");
    System.out.print("Enter Multiplicand(M): ");
    M=sc.nextInt();
    System.out.print("Enter Multiplier(Q): ");
```

```
Q=sc.nextInt();

M_BIN=binary(M, BITS);
Q_BIN=binary(Q, BITS);
if(M<0){
  M_BIN=complement(M_BIN, BITS);
}
if(Q<0){
  Q_BIN=complement(Q_BIN, BITS);
}
System.out.print("\nM in binary is: ");
Arrays.stream(M_BIN).forEach(System.out::print);
System.out.print("\nQ in binary is: ");
Arrays.stream(Q_BIN).forEach(System.out::print);
System.out.println("\n\nExpected product: "+(M*Q));
Arrays.stream(answer).forEach(System.out::print);
System.out.print(" : ");
Arrays.stream(Q_BIN).forEach(System.out::print);
System.out.print("  "+Q_1);
while(count>0){
    if(Q_BIN[BITS-1]==1 && Q_1==0){
        M_TEMP=complement(M_BIN.clone(), BITS);
        answer=add(answer, M_TEMP, BITS);
    }
    else if(Q_BIN[BITS-1]==0 && Q_1==1){
      M_TEMP=M_BIN.clone();
      answer=add(answer, M_TEMP, BITS);
    }
System.out.println("\n\nCOUNT= "+count);
System.out.println("Array after comparing digits: ");
Arrays.stream(answer).forEach(System.out::print);
System.out.print(" : ");
Arrays.stream(Q_BIN).forEach(System.out::print);
System.out.print("  "+Q_1);
Q_1=Q_BIN[BITS-1];
for(int i=BITS-1;i>0;i--){
  Q_BIN[i]=Q_BIN[i-1];
}
Q_BIN[0]=answer[BITS-1];
for(int i=BITS-1;i>0;i--){
  answer[i]=answer[i-1];
}
//answer[0]=answer[1];
System.out.println("\nArray after right shift: ");
```

```
    Arrays.stream(answer).forEach(System.out::print);
    System.out.print(" : ");
    Arrays.stream(Q_BIN).forEach(System.out::print);
    System.out.print("  "+Q_1);
    count--;
    }
    System.out.print("\nFinal Product: ");
    Arrays.stream(answer).forEach(System.out::print);
    System.out.print(" : ");
    Arrays.stream(Q_BIN).forEach(System.out::print);
  }
  public static int[] binary(int dec, int BITS){
    int[] dec_bin=new int[BITS];
    for(int i=BITS-1;i>=0;i--){
      dec_bin[i]=dec%2;
      dec=dec/2;
    }
    return dec_bin;
  }
  public static int[] add(int[] arr1,int[] arr2, int BITS){
    int carry=0;
    int arr[]=new int[BITS];
    for(int i=BITS-1;i>=0;i--)
    {
        arr[i]=(arr1[i]+arr2[i]+carry)%2;
        carry=(arr1[i]+arr2[i]+carry)/2;
    }
    return arr;
  }
  public static int[] complement(int[] bin, int BITS){
    for(int i=0;i<BITS;i++){
        bin[i]=(bin[i]==0)?1:0;
    }
    int plus_one[]=new int[BITS];
    plus_one[BITS-1]=1;
    bin=add(bin,plus_one,BITS);
    return bin;
  }
}
```

For verification, my code is available on:

https://repl.it/@ARGHYADEEPDAS/COAExpt2

Output Screen:

```
BOOTH'S ALGORITHM
Enter Multiplicand(M):  -6
Enter Multiplier(Q):  -4

M in binary is: 11010
Q in binary is: 11100

Expected product: 24
00000 : 11100  0

COUNT= 5
Array after comparing digits:
00000 : 11100  0
Array after right shift:
00000 : 01110  0

COUNT= 4
Array after comparing digits:
00000 : 01110  0
Array after right shift:
00000 : 00111  0

COUNT= 3
Array after comparing digits:
00110 : 00111  0
Array after right shift:
00011 : 00011  1

COUNT= 2
Array after comparing digits:
00011 : 00011  1
Array after right shift:
00001 : 10001  1

COUNT= 1
Array after comparing digits:
00001 : 10001  1
Array after right shift:
00000 : 11000  1
Final Product: 00000 : 11000
```

Example:

M=5 (0101)

Q=4 (0100)

M's 2's compliment: 1011

Count	A	Q	Q_{-1}	Operation
4	0000	0100	0	
	0000	0010	0	Arithmetic Right Shift
3	0000	0001	0	Arithmetic Right Shift
2	1011	0001	0	A=A-M
	1101	1000	1	Arithmetic Right Shift
1	0010	1000	1	A=A+M
	0001	**0100**	0	Arithmetic Right Shift

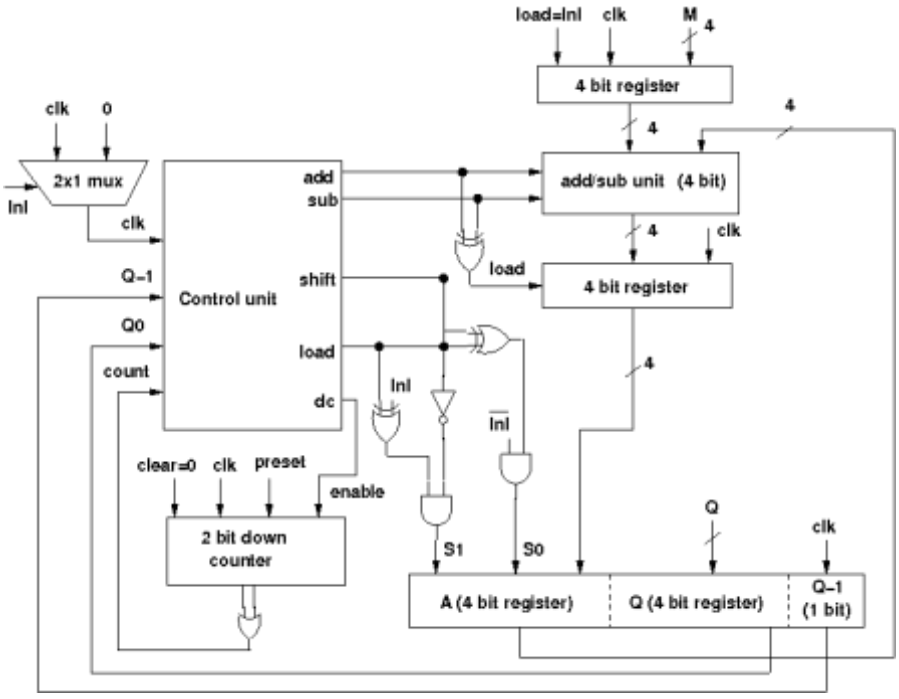
AQ=$(00010100)_2$=$(20)_{10}$.

Conclusion: The Booth's Algorithm for multiplication was implemented in Java and the program ran successfully, giving desired output.

Post Lab Descriptive Questions (Add questions from examination point view)

1. Draw the Circuit diagram for booths algorithm.

Ans.



2. Explain advantages and disadvantages of Booth's Algorithm.

Ans. Advantages:

1. It reduces the number of partial products to be compressed in a carry-save added tree.
2. It is very fast for multiplications having long operands (>16 bits).
3. Not many conversions are required (only 2's complement used).

Disadvantages:

1. A lot of complexity is involved in the circuit to generate a partial product bit in Booth's encoding.

2. Algorithm is inefficient for isolated 1's.
3. Inconvenient for designing a synchronous multiplier.

Date: 01/08/2018 **Signature of faculty in-charge**