Assignment A – 3

Aim:
To implement Booth's multiplication algorithm through a web tool.

Problem Statement
A Web Tool for Booth's multiplication algorithm is used to multiply two numbers located in distributed environment. Use software design client-server architecture and principles for dynamic programming. Perform Risk Analysis. Implement the design using HTML-5/Scala/python/Java/C++/ Ruby on Rails. Perform Positive and Negative testing. Use latest open source software modeling, Designing and testing tool/Scrum-it/KADOS and Camel.

Objectives
1 Implementation of the problem statement using Object oriented programming.
2 Multiply two numbers located in distributed environment Using Booth's multiplication algorithm.

Theory
Introduction

Booth's multiplication algorithm is a multiplication algorithm that multiplies two signed binary numbers in two's complement notation. The algorithm was invented by Andrew Donald Booth in 1950 while doing research on crystallography at Birkbeck College in Bloomsbury, London. Booth used desk calculators that were faster at shifting than adding and created the algorithm to increase their speed.

Booth's algorithm is of interest in the study of computer architecture.

- Multiply 14 times -5 using 5-bit numbers (10-bit result).
- 14 in binary: 01110
- -14 in binary: 10010 (so we can add when we need to subtract the multiplicand)
- -5 in binary: 11011
- Expected result: -70 in binary: 11101 11010

Step	**Multiplicand**	**Action**	**Multiplier** **Upper 5 bits – 0,** **Lower 5 bits – multiplier,** **1 'booth bit' initially 0**
0	0110	Initialization	00000110110

Step	Multiplicand	Action	Multiplier Upper 5 bits – 0, Lower 5 bits – multiplier, 1 'booth bit' initially 0
1	01110	10: Subtract multiplicand	00000 + 10010 = 10010 10010 11011 0
		Shift right arithmetic	11001 01101 1
2	01110	11: no-op	11001 01101 1
		Shift right arithmetic	11100 10110 1
3	01110	01: Add multiplicand	11100 + 01110 = 01010 (carry ignored because adding a positive and negative number cannot overflow) 01010 10110 1
		Shift right multiplicand	00101 01011 0
4	01110	10: subtract multiplicand	00101 + 10010 = 10111 10111 01011 0
		Shift right arithmetic	11011 10101 1
5	01110	11: no-op	11011 10101 1
		Shift right arithmetic	11101 11010 1

An elegant approach to multiplying signed numbers.

Using the standard multiplication algorithm, a run of 1s in the multiplier in means that we have to add as many successively shifted multiplicand values as the number of 1s in the run.

```
  0010
*0111
----------
+ 0010 multiplicand shifted by 0 bits left
+ 0100 multiplicand shifted by 1 bit left
+ 1000 multiplicand shifted by 2 bits left
+ 0000
----------
00001110
```

We can rewrite 2i - 2i-j as:

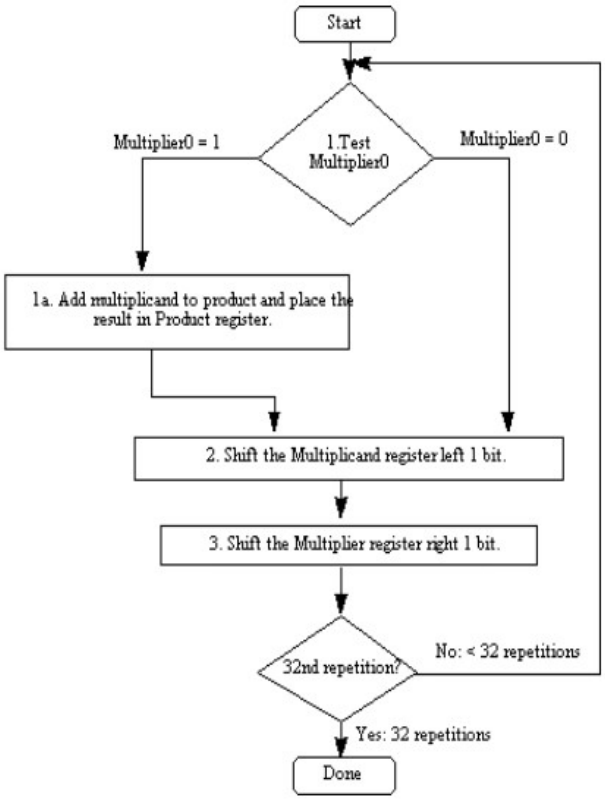
$2i - 2i-j = 2i-j \times (2j - 1)$
$= 2i-j \times (2j-1 + 2j-2 + ... + 20)$
$= 2i-1 + 2i-2 + ... + 2i-j$

For example $0111_{two} = 23_{ten} - 20_{ten}$. So $0010_{two} \times 0111_{two}$ can be written as:

0010_{two} x (1000_{two} – 0001_{two})
Or to make it look like the multiplication before:
0010_{two} x 0111_{two}

- 0010 = 0010_{two} x - 0001_{two}
+ 0000
+ 0000
+ 0010
= 0010_{two} x + 1000_{two}

00001110_{two}



Risks:
1. Server down.
2. Category mismatch.
3. Redundant entry.

Conclusion
Booth's algorithm has been implemented by taking numbers from distributed sources.

Mathematical Model
Mathematical Model Booth Multiplication is basically used for multiplication of two binary numbers.
Following parameters are used for Booth Multiplication:

Let, S be the System Such that,
S= {I, F, O, success, failure}
Where,
I = Input
F = Function
O = Output

Input:

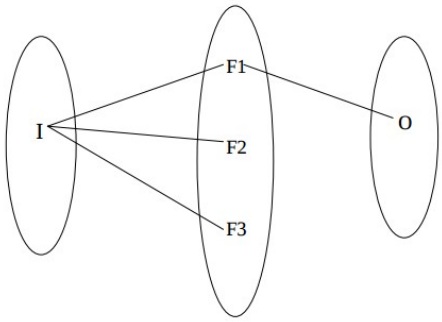
I = any two numbers (here we have used decimal)

Function:

F1= main function (contains even and odd phase)
F2 = Right Shift if last two bits of number are same in p
F3=P+S or P-S when last two digits of p are not same

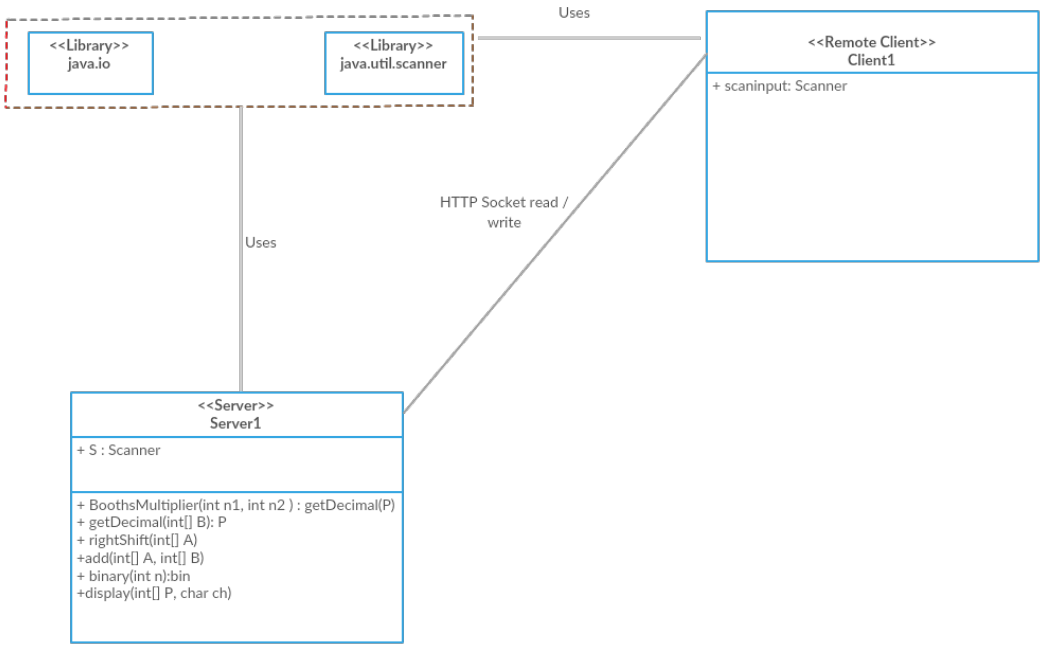
Output:

O = answer of multiplication of given input numbers.

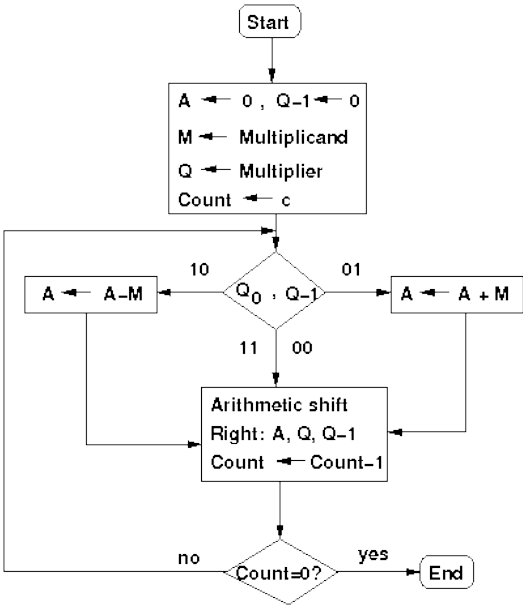


UML Diagrams

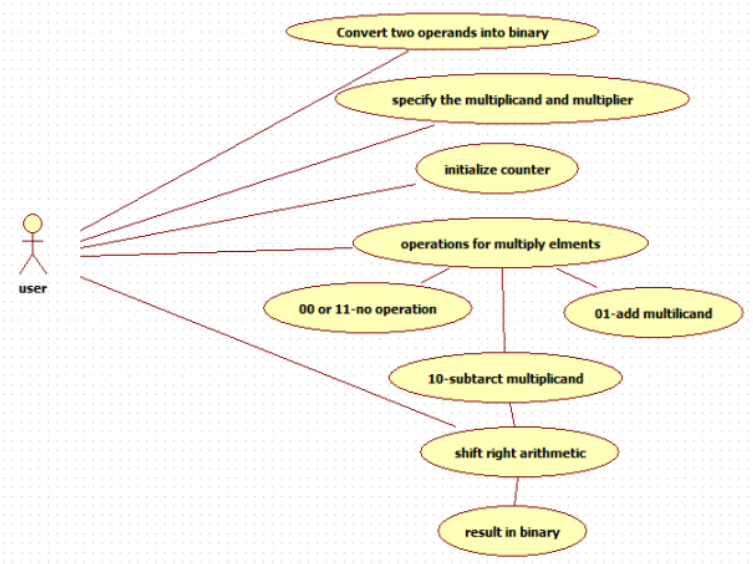
Class Diagram



Activity Diagram



Use Case Diagram



Deployment Diagram