# ASSIGNMENT NO: A3

## 1. TITLE

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++/ Rubi on Rails. Perform Positive and Negative testing. Use latest open source software modelling, Designing and testing tool/Scrum-it/KADOS and Camel.

## 2. PREREQUISITES

- 64-bit Fedora or equivalent OS with 64-bit Intel-i5/i7
- Java 1.7.0
- Testing Tool: Scrum-it/KADOS/Camel

## 3. OBJECTIVE

- To perform Risk Analysis.
- To learn about designing and testing tools.

## 4. MATHEMATICAL MODELS

Let, S be the System Such that,
A={ S, E, I,O, F, DD, NDD, success, failure }
Where,
S= Start state,
E= End State,
I= Set of Input
O= Set of Out put
F =Set of Function
DD=Deterministic Data
NDD=Non Deterministic Data
Success Case: It is the case when all the inputs are given by system are entered correctly. Failure Case: It is the case when the input does not match the validation Criteria.

## 5. THEORY

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.

**Booth Multiplication algorithm**:

Booth's algorithm examines adjacent pairs of bits of the $N$-bit multiplier $Y$ in signed two's complement representation, including an implicit bit below the least significant bit, $y_{-1} = 0$. For each bit $y_i$, for $i$ running from 0 to $N$-1, the bits $y_i$ and $y_{i-1}$ are considered. Where these two bits are equal, the product accumulator $P$ is left unchanged. Where $y_i = 0$ and $y_{i-1} = 1$, the multiplicand times $2^i$ is added to $P$; and where $y_i = 1$ and $y_{i-1} = 0$, the multiplicand times $2^i$ is subtracted from $P$. The final value of $P$ is the signed product.

The multiplicand and product are specified; typically, these are both also in two's complement representation, like the multiplier, but any number system that supports addition and subtraction will work as well. As stated here, the order of the steps is not determined. Typically, it proceeds from LSB to MSB, starting at $i = 0$; the multiplication by $2^i$ is then typically replaced by incremental shifting of the $P$ accumulator to the right between steps; low bits can be shifted out, and subsequent additions and subtractions can then be done just on the highest $N$ bits of $P$. There are many variations and optimizations on these details.

The algorithm is often described as converting strings of 1's in the multiplier to a high-order +1 and a low-order –1 at the ends of the string. When a string runs through the MSB, there is no high-order +1, and the net effect is interpretation as a negative of the appropriate value.

**Implementation:**

Booth's algorithm can be implemented by repeatedly adding (with ordinary unsigned binary addition) one of two predetermined values $A$ and $S$ to a product $P$, then performing a rightward arithmetic shift on $P$. Let **m** and **r** be the multiplicand and multiplier, respectively; and let $x$ and $y$ represent the number of bits in **m** and **r**.

1. Determine the values of $A$ and $S$, and the initial value of $P$. All of these numbers should have a length equal to $(x + y + 1)$.
    1. A: Fill the most significant (leftmost) bits with the value of **m**. Fill the remaining $(y + 1)$ bits with zeros.
    2. S: Fill the most significant bits with the value of $(-\mathbf{m})$ in two's complement notation. Fill the remaining $(y + 1)$ bits with zeros.
    3. P: Fill the most significant $x$ bits with zeros. To the right of this, append the value of **r**. Fill the least significant (rightmost) bit with a zero.
2. Determine the two least significant (rightmost) bits of $P$.
    1. If they are 01, find the value of $P + A$. Ignore any overflow.
    2. If they are 10, find the value of $P + S$. Ignore any overflow.
    3. If they are 00, do nothing. Use $P$ directly in the next step.
    4. If they are 11, do nothing. Use $P$ directly in the next step.
3. Arithmetically shift the value obtained in the 2nd step by a single place to the right. Let $P$ now equal this new value.
4. Repeat steps 2 and 3 until they have been done $y$ times.
5. Drop the least significant (rightmost) bit from $P$. This is the product of **m** and **r**.

We will explain Booth's Multiplication with the help of following **example**
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
The Multiplication Process is explained below:

| **Step** | **Multiplicand** | **Action** | **Multiplier**<br>**upper 5-bits 0,**<br>**lower 5-bits multiplier,**<br>**1 "Booth bit" initially 0** |
|---|---|---|---|
| **0** | 01110 | Initialization | 00000 11011 0 |
| 1 | 01110 | 10: Subtract Multiplicand | 00000+10010=10010<br>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<br>(Carry ignored because adding a positive and negative number cannot overflow.)<br>01010 10110 1 |
| | | Shift Right Arithmetic | 00101 01011 0 |
| **4** | 01110 | 10: Subtract Multiplicand | 00101+10010=10111<br>10111 01011 0 |
| | | Shift Right Arithmetic | 11011 10101 1 |
| **5** | 01110 | 11: No-op | 11011 10101 1 |
| | | Shift Right Arithmetic | 11101 11010 1 |

Booth's Multiplication algorithm is 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.

0010two
x 0111two
--------
\+ 0010 multiplicand shifted by 0 bits left
\+ 0010 multiplicand shifted by 1 bit left
\+ 0010 multiplicand shifted by 2 bits left
\+ 0000
--------

00001110two

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

2i - 2i-j = 2i-j x (2j - 1)

= 2i-j x (2j-1 + 2j-2 + ... + 20)

= 2i-1 + 2i-2 + ... + 2i-j

For example 0111two = 23ten - 20ten. So 0010two x 0111two can be written as:

0010two x (1000two - 0001two)

Or to make it look like the multiplication before:

0010two

x 0111two

---

\- 0010 = 0010two x - 0001two

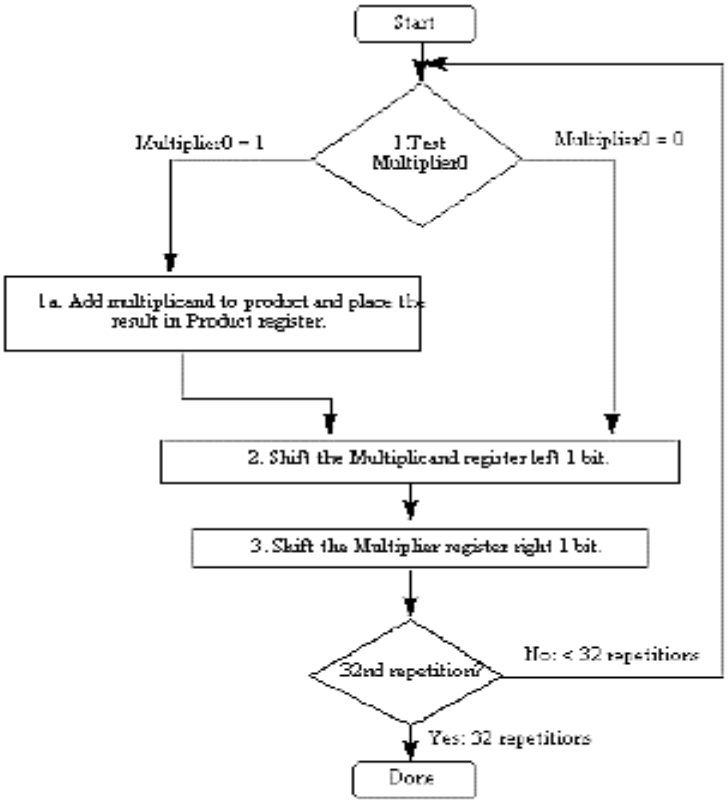
\+ 0000

\+ 0000

\+ 0010 = 0010two x + 1000two

00001110two

Flowchart of booth's multiplication Algorithm :




**Multiply Algorithm**

**KADOS** is a web tool for managing Agile projects (Scrum more specifically) through visual boards on which are displayed post-its representing User Stories, Tasks, Activities, Issues, Actions, Bugs and any objects you wanted your project to manage.

**RISK ANALYSIS**

Risk can be defined as the potential of losses and rewards resulting from an exposure to a hazard or as a result of a risk event. Risk can be viewed to be a multi-dimensional quantity that includes

- event occurrence probability,
- event occurrence consequences,
- consequence significance, and
- the population at risk.

Risk analysis is the process of defining and analyzing the dangers to individuals, businesses and government agencies posed by potential natural and human-caused adverse events. Risk analysis is the review of the risks associated with a particular event or action. In IT, a risk analysis report can be used to align technology-related objectives with a company's business objectives. A risk analysis report can be either quantitative or qualitative. Risk analysis can be defined in many different ways, and much of the definition depends on how risk analysis relates to other concepts. Risk analysis can be "broadly defined to include risk assessment, risk characterization, risk communication, risk management, and policy relating to risk, in the context of risks of concern to individuals, to public- and private-sector organizations, and to society at a local, regional, national, or global level." A useful construct is to divide risk analysis into two components: (1) risk assessment (identifying, evaluating, and measuring the probability and severity of risks) and (2) risk management (deciding what to do about risks).

**6. CONCLUSION**

Hence we have implemented Booth’s Multiplication algorithm and performed test cases for that.

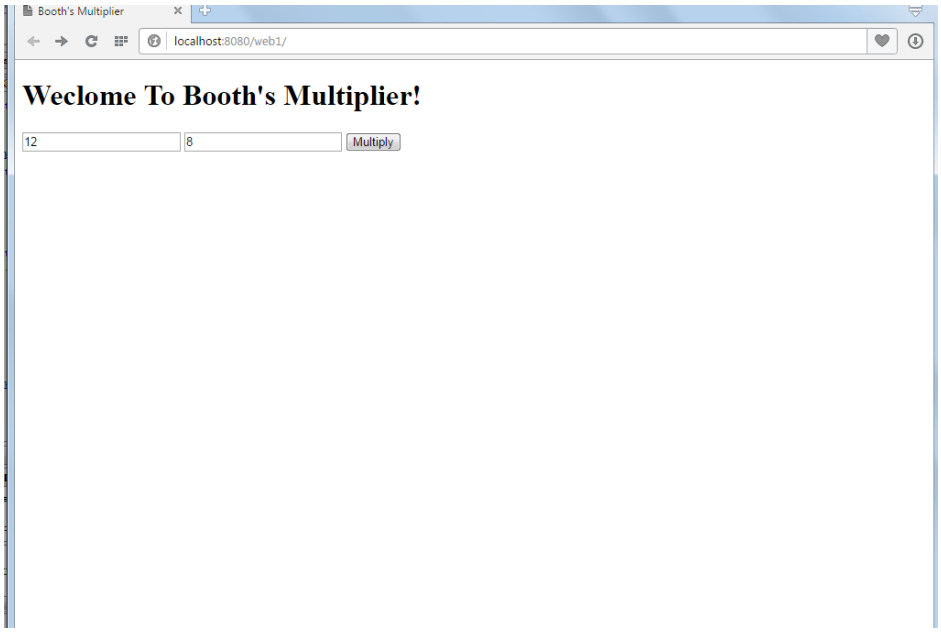
```
<%--
    Document   : index
    Created on : 25 Jan, 2016, 2:13:52 PM
    Author     : Piyush Galphat
--%>

<%@page contentType="text/html" pageEncoding="UTF-8"%>
<!DOCTYPE html>
<html>
    <head>
        <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
        <title>Booth's Multiplier</title>
    </head>
    <body>
        <h1>Weclome To Booth's Multiplier!</h1>
        <form action="newjsp.jsp" method="POST">
            <input type="number" name="m1" value="">
            <input type="number" name="m2" value="">
        <input type="submit" value="Multiply">
        </form>

    </body>
</html>
```

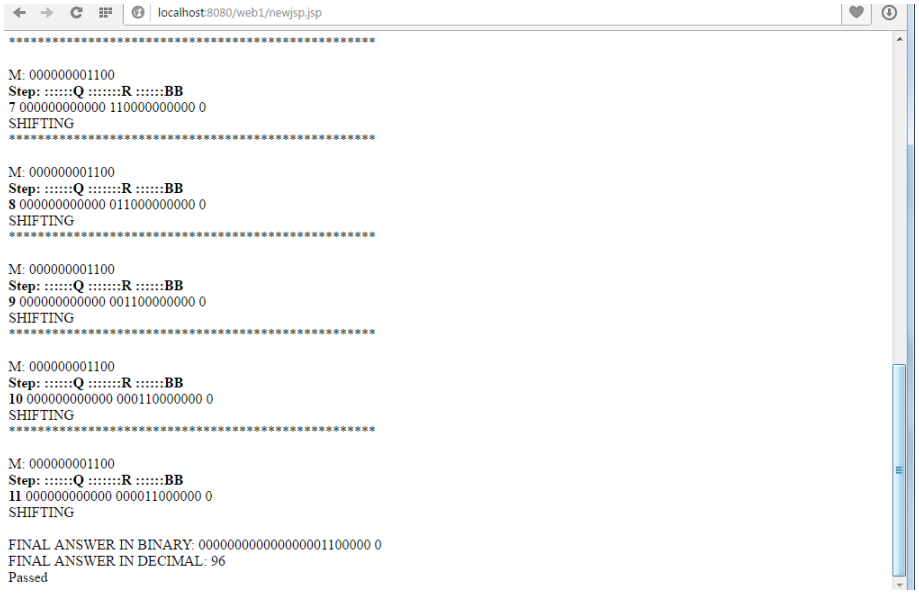
```
<%--
    Document   : mypage
    Created on : 25 Jan, 2016, 4:07:40 PM
    Author     : Piyush Galphat
--%>

<%@page contentType="text/html" pageEncoding="UTF-8"%>
<!DOCTYPE html>
<html>
    <head>
        <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
        <title>JSP Page</title>
    </head>
    <body>
        <h1><u>Answer Page:</u></h1>

        <%!
        public int size=12;//no of steps

        %>

        <%!
        //TO GET 2'S CIMPLEMENT
        public void do_2comp(int m[]){
       int i=size-1;
       while((m[i--]!=1)){
            if(i<0)
                break;
        }
       for(;i>=0;i--)
              m[i]=m[i]==0?1:0;
        }
        //TO GET 2'S CIMPLEMENT
        public void do_2comp(int m[],int s){
       int i=s-1;
       while((m[i--]!=1)){
            if(i<0)
                break;
```

```
        }
       for(;i>=0;i--)
              m[i]=m[i]==0?1:0;
        }

        public int to_decimal(int m[],int lim,int flag){
       int num=0;
       if(flag<1){
              do_2comp(m,lim);
       }
       for(int i=lim-1;i>=0;i--){
              num+=Math.pow(2,lim-1-i)*m[i];
       }
       return flag<1?-num:num;
        }

        public void to_binary(int m[],int m1){
       int flag=0;
       if(m1<0){
              m1*=-1;
              flag=1;
       }
       int i=size-1;
       do{
              if(i<0){
                     //out.println("BITS OVERFLOW::");
                     break;
              }
              m[i--]=(m1%2);
              m1/=2;
       }while(m1!=0);
       if(flag==1){
              do_2comp(m);
       }
        }

        public void bin_add(int a[],int b[]){
       int[] c=new int[size];
       int cr=0;
       for(int i=size-1;i>=0;i--){
              c[i]=(a[i]+b[i]+cr)%2;
              cr=(a[i]+b[i]+cr)/2;
              a[i]=c[i];
       }
       }

        public void bin_sub(int a[],int b[]){
            try{
       do_2comp(b);
       bin_add(a,b);
       do_2comp(b);
            }catch(Exception e){
               e.printStackTrace();
            }
        }
    %>

         <%
              //int out=1,in=1;
 //do{
//      do{
       int m1=Integer.parseInt(request.getParameter("m1"));
        int r1=Integer.parseInt(request.getParameter("m2"));
        int[] m=new int[size];      //multiplicand
```

```
        int r[]=new int[size];       //multiplier
        int q[]=new int[size];
        int bb=0;      //booth bit
        double range=(Math.pow(2, size-1)-1);
         out.println("Range: +-"+(range)+"<br>");
         if(m1>range||r1>range){
             out.println("Exceeds range....<br>");
             return;
         }

        //binary conversion
        to_binary(m,m1);
        to_binary(r,r1);

// booth's multiplication process starts
int w=size;
        while(w!=0){   //repeated size times
//checking bits
    out.print("**************************************************\n"+"<br>");
    out.println("<br>"+"\nM: ");
    for(int i=0;i<size;i++){
        out.print(m[i]);
    }

    out.println("<br>"+"\n<b>Step:\t\t::::::Q\t\t:::::::R\t\t::::::BB</b>");

   out.println("<br><b>"+(size-w)+"</b>\t\t");
    for(int i=0;i<size;i++){
     out.print(q[i]);
    }
    out.print("\t\t");
    for(int i=0;i<size;i++){
    out.print(r[i]);
    }

    out.println("\t\t"+bb);
        if(r[size-1]==1&&bb==0){
               //substraction
               bin_sub(q,m);

        out.print("<br>"+"\nSUBSTRACTION\n"+"<br>");
        }
        if(r[size-1]==0&&bb==1){
               //addition
               bin_add(q,m);
        out.print("<br>"+"\nADDITION\n"+"<br>");
        }
        //ASR
        bb=r[size-1];
        int t1;        //temporary variable
        t1=q[size-1];
        for(int i=size-1;i>0;i--){
               q[i]=q[i-1];
        }
        for(int i=size-1;i>0;i--){
               r[i]=r[i-1];
        }
        r[0]=t1;
        out.print("<br>"+"\nSHIFTING\n"+"<br>");
//end ASR
         w--;
        }//end for booths
        out.print("<br>"+"\nFINAL ANSWER IN BINARY: ");
        int alu[]=new int[2*size];
```

```
         int z=0;
        for(int i=0;i<size;i++){
               alu[z++]=q[i];
               //cout<<q[i];
        }
        for(int i=0;i<size;i++){
               alu[z++]=r[i];
               //cout<<r[i];
        }
        for(int i=0;i<2*size;i++){
               out.print(alu[i]);
        }
        out.print("\t\t"+bb);
        int ans;
        out.print("<br>"+"\nFINAL ANSWER IN DECIMAL: ");

        if(!(m1<0&&r1<0)&&(m1<0||r1<0))
               ans=to_decimal(alu,2*size,-2);
        else
               ans=to_decimal(alu,2*size,2);
        out.println(ans);
        if(ans!=(m1*r1)){
               out.println("<br>"+"\n\t\t\tFailed for in:out
|"+m1+":"+r1+"\n\n"+"<br>"+"<br>");
        }
        else{
        out.println("<br>"+"Passed");
        }

         %>
    </body>
</html>
```


Booth's Multiplier
localhost:8080/web1/
Weclome To Booth's Multiplier!
12
8
Multiply


localhost:8080/web1/newjsp.jsp
**********
M: 000000001100
Step: ::::::Q :::::::R ::::::BB
7 000000000000 110000000000 0
SHIFTING
**********
M: 000000001100
Step: ::::::Q :::::::R ::::::BB
8 000000000000 011000000000 0
SHIFTING
**********
M: 000000001100
Step: ::::::Q :::::::R ::::::BB
9 000000000000 001100000000 0
SHIFTING
**********
M: 000000001100
Step: ::::::Q :::::::R ::::::BB
10 000000000000 000110000000 0
SHIFTING
**********
M: 000000001100
Step: ::::::Q :::::::R ::::::BB
11 000000000000 000011000000 0
SHIFTING
FINAL ANSWER IN BINARY: 000000000000000001100000 0
FINAL ANSWER IN DECIMAL: 96
Passed