###### Experiment Number:

## TITLE: Booth’s Multiplication algorithm

**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 Python. Perform Positive and Negative testing. Use latest open source software modeling, Designing and testing tool (Scrum-it and Camel)

**OBJECTIVES:**

1. To develop problem solving abilities using Mathematical Modeling
2. To implement software design and testing in distributed, concurrent and parallel environments

**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.

**Socket Programming in Python**

**socket.socket():** Create a new socket using the given address family, socket type and protocol number.

**socket.bind(address)**: Bind the socket to address.

**socket.listen(backlog)**: Listen for connections made to the socket. The backlog argument specifies the maximum number of queued connections and should be at least 0; the maximum value is system-dependent (usually 5), the minimum value is forced to 0.

**socket.accept():** The return value is a pair (conn, address) where conn is a new socket object usable to send and receive data on the connection, and address is the address bound to the socket on the other end of the connection.

**socket.send(bytes[, flags])**: Send data to the socket. The socket must be connected to a remote socket. Returns the number of bytes sent. Applications are responsible for checking that all data has been sent; if only some of the data was transmitted, the application needs to attempt delivery of the remaining data.

**socket.close()**: Mark the socket closed. all future operations on the socket object will fail. The remote end will receive no more data (after queued data is flushed). Sockets are automatically closed when they are garbage-collected, but it is recommended to close() them explicitly. Note that the server socket doesn't receive any data. It just produces client sockets. Each client socket is created in response to some other client socket doing a connect() to the host and port we're bound to. As soon as we've created that clientsocket, we go back to listening for more connections**.**

***bitstring*** is a pure Python module designed to help make the creation and analysis of binary data as simple and natural as possible. *bitstrings* can be constructed from integers (big and little endian), hex, octal, binary, strings or files. They can be sliced, joined, reversed, inserted into, overwritten, etc. with simple functions or slice notation. They can also be read from, searched and replaced, and navigated in, similar to a file or stream. *bitstring* is open source software, and has been released under the MIT licence.

We need to install " bitstring " library before executing the code.

To do so navigate to bitstring-3.1.4 directory and execute the following command

***$ sudo python setup.py install*** Or you can simply install it using pip ***$ pip install bitstring***

**MATHEMATICAL MODEL:**

Aim: Let system ’S’ be the solution for the multiplication of two numbers using Booth’s Multiplication Algorithm.

Initialisation:.

Input = Two decimal numbers.

Output = A decimal number with the binary conversions.

Mathematical model using Set theory:

L = {(S, E, I, O, F, SUCCESS, FAILURE)}

S = Initial state

E = End of state

I = input set i.e. 2 decimal numbers.

O = output set i.e. product of the 2 decimal numbers.

F = Functions used in program

F1 : booth()

F2 : arith\_shift\_right()

FAILURE = System reaches desired state i.e multiplication of 2 decimal numbers is calculated successfully

FAILURE = Desired outcome not generated. i.e multiplication of 2 decimal numbers is not calculated successfully

**IMPLEMENTATION DETAILS / DESIGN LOGIC:**

*(Algorithm/Flow Charts/Sequence Diagram)*

**Algorithm 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**.

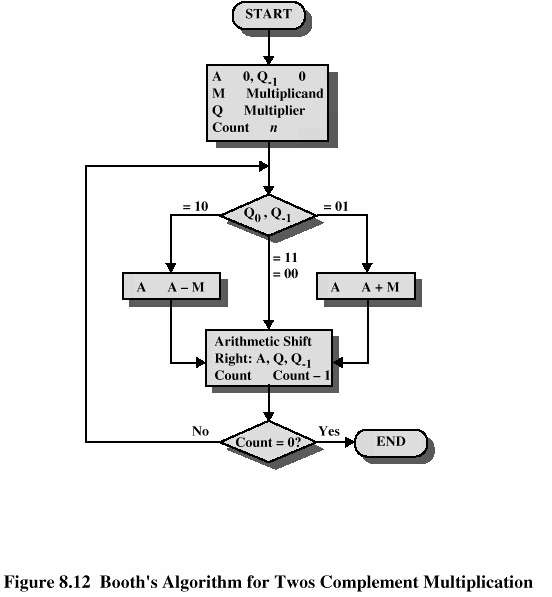
1. Determine the values of *A* and *S*, and the initial value of *P*. All of these numbers should have a length equal to (*x* + *y* + 1).
   1. A: Fill the most significant (leftmost) bits with the value of **m**. Fill the remaining (*y* + 1) bits with zeros.
   2. S: Fill the most significant bits with the value of (−**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**.

## Example :

Find 3 × (−4), with m = 3 and r = −4, and *x* = 4 and *y* = 4:

* m = 0011, -m = 1101, r = 1100
* A = 0011 0000 0
* S = 1101 0000 0
* P = 0000 1100 0
* Perform the loop four times:
  1. P = 0000 1100 0. The last two bits are 00.
     + P = 0000 0110 0. Arithmetic right shift.
  2. P = 0000 0110 0. The last two bits are 00.
     + P = 0000 0011 0. Arithmetic right shift.
  3. P = 0000 0011 0. The last two bits are 10.
     + P = 1101 0011 0. P = P + S.
     + P = 1110 1001 1. Arithmetic right shift.
  4. P = 1110 1001 1. The last two bits are 11.
     + P = 1111 0100 1. Arithmetic right shift.
* The product is 1111 0100, which is −12.

**Flow Chart :**



**Sequence Diagram :**



**State Transition Diagram :**



**Input :**

The client part of the program takes two decimal numbers as an input to perform the multiplication from the terminal and sends them to the server part of the program.

**Expected Output :**

The server part of the program accepts the decimal numbers and converts them to binary bitstring, displays all the operations and gives the correct result.

Execute the program with the following commands.

Start a terminal and run

$ python booth\_server.py

Open another terminal and run

$ python booth\_client.py

**TEST CASES: write down 3 positive and 3 negative test cases**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Test ID | Description | Input | Expected output | Actual output |
| 1 | Client Server session establishment | Client sends request to server for commumnication | Server should accept the request | Server accepts the request and establishes a session with Client |
| 2 | Multiplication of 2 positive decimal numbers | 2 positive deciamal numbers | Should display correct result of multiplication with binary conversion | Successfully displays correct result of multiplication with binary conversion. |
| 3 | Multiplication of 2 negative decimal numbers | 2 negative deciamal numbers | Should display correct result of multiplication with binary conversion | Successfully displays correct result of multiplication with binary conversion |
| 4 | Multiplication of two decimal numbers (one of the number is positive and another is negative) | 2 deciamal numbers one of the number is positive and another is negative | Should display correct result of multiplication with binary conversion | Successfull displays correct result of multiplication with binary conversion |
| 5 | Multiplication of two numbers with 5 bit binary represntation | 2 decimal numbers | Should display correct result of multiplication with binary conversion | Successfull displays correct result of multiplication with 5 bit binary conversion |
| 6 | Multiplication of two numbers with 8 bit binary represntation | 2 decimal numbers | Should display correct result of multiplication with binary conversion | Successfull displays correct result of multiplication with 8 bit binary conversion |

|  |  |
| --- | --- |
|  |  |
|  |  |
|  |  |

**CONCLUSION:**

Hence we have studied and implemented Web Tool for Booth’s multiplication algorithm is used to multiply two numbers using Client Server architecture.

**COURSE OUTCOMES ACHIEVED:**

1. To use software design methods and testing.
2. To solve problems for distributed environments

**FAQ’s**

Q. 1 What is Booth's Multiplication Algorithm? Explain with an example.

Q. 2 Multiply 14 times -5 using 5-bit numbers (10-bit result) using Booth's Algorithm for multiplication. 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

Q. 3 Explain the working principle of Booth's Algorithm using Client Server Architecture.

Q. 4. What are the applications of Booth's Multiplication Algorithm.

## Server.py

#!/usr/bin/python # This is server.py file

from bitstring import BitArray

import socket # Import socket module

def booth(m, r, x, y):

# Initialize

totalLength = x + y + 1

mA = BitArray(int = m, length = totalLength)

rA = BitArray(int = r, length = totalLength)

A = mA << (y+1)

S = BitArray(int = -m, length = totalLength) << (y+1)

P = BitArray(int = r, length = y)

P.prepend(BitArray(int = 0, length = x))

P = P << 1

print "Initial values"

print "A", A.bin

print "S", S.bin

print "P", P.bin

print "Starting calculation"

for i in range(1,y+1):

if P[-2:] == '0b01':

P = BitArray(int = P.int + A.int, length = totalLength)

print "P + A:", P.bin

elif P[-2:] == '0b10':

P = BitArray(int = P.int +S.int, length = totalLength)

print "P + S:", P.bin

P = arith\_shift\_right(P, 1)

print "P >> 1:", P.bin

P = arith\_shift\_right(P, 1)

print "P >> 1:", P.bin

return P.int

def arith\_shift\_right(x, amt):

l = x.len

x = BitArray(int = (x.int >> amt), length = l)

return x

s = socket.socket() # Create a socket object

host = socket.gethostname() # Get local machine name

port = 12345 # Reserve a port for your service.

s.bind((host, port)) # Bind to the port

s.listen(5) # Now wait for client connection.

while True:

c, addr = s.accept() # Establish connection with client.

print 'Got connection from', addr

m[=c.recv(1024)](http://piratepad.net/ep/search?query=c.recv(1024))

print('m :',m)

c.send('m recvd')

r[=c.recv(1024)](http://piratepad.net/ep/search?query=c.recv(1024))

print('r :',r)

c.send('r recvd')

print c.recv(1024)

# Sample usage: find 86 \* 41

b = booth(int(m),int(r),8,8)

print b

c.send(str(b))

c.close() # Close the connection

**Client.py**

#!/usr/bin/python # This is client.py file

import socket # Import socket module

# Accept input

multiplicant[=raw\_input(](http://piratepad.net/ep/search?query=raw_input()"Multiplicant :")

multiplier[=raw\_input(](http://piratepad.net/ep/search?query=raw_input()"Multiplier :")

s = socket.socket() # Create a socket object

host = socket.gethostname() # Get local machine name

port = 12345 # Reserve a port for your service.

s.connect((host, port))

s.send(multiplicant) [#send](http://piratepad.net/ep/search?query=send) multiplicant

print s.recv(1024)

s.send(multiplier) [#send](http://piratepad.net/ep/search?query=send) multiplier

print s.recv(1024)

s.send("From client: Do the calculations for me")

b[=s.recv(1024)](http://piratepad.net/ep/search?query=s.recv(1024))

print("Answer: " + b)

s.close # Close the socket when done

**OUTPUT :**

**Server side**

rajat@y500  ~/Desktop/booth  python s1.py

Got connection from ('127.0.0.1', 57250)

('m :', '2')

('r :', '-3')

From client: Do the calculations for me

Initial values

A 001000000

S 111000000

P 00011010

Starting calculation

P + S: 111011010

P >> 1: 111101101

P + A: 000101101

P >> 1: 000010110

P + S: 111010110

P >> 1: 111101011

P >> 1: 111110101

P >> 1: 111111010

**Client Side**

rajat@y500  ~/Desktop/booth  python c1.py  ✓

Multiplicant :2

Multiplier :-3

m recvd

r recvd

Answer: -6