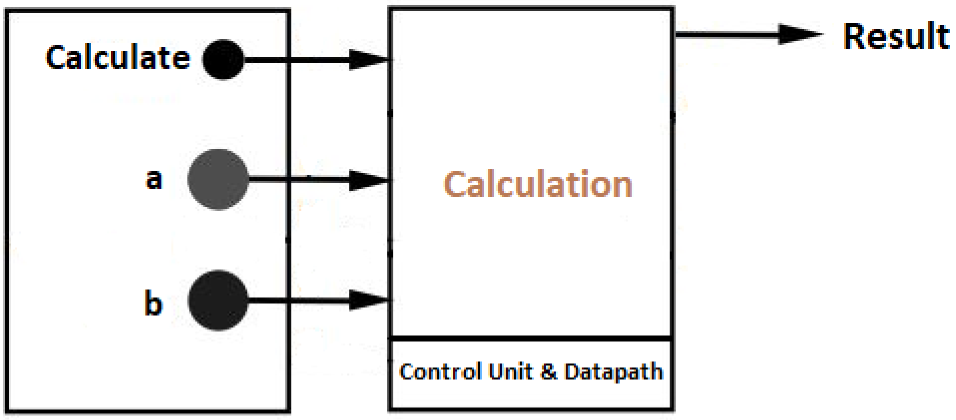
GTU Department of Computer Engineering CSE 232 - Spring 2020 Project 2 Report

Akif KARTAL
171044098

Problem Definition

The problem is to compute the multiplication of two 16-bit numbers without using multiplier.
A basic diagram to show the problem is;



Inputs: Calculate (Button), a and b (16-bit singed numbers)

Outputs: 16-bit singed number (between −32,768 and 32,767)

C language version of problem;

```
while(!calculate);
if (a < 0)
{
     a = -1*a;
     b = -1*b;
}
mult = 0;
while( a > 0 ){
     mult = mult + b;
     a = a - 1;
}
result = mult;
```

Note that this version of C code performs **negative number** multiplications.

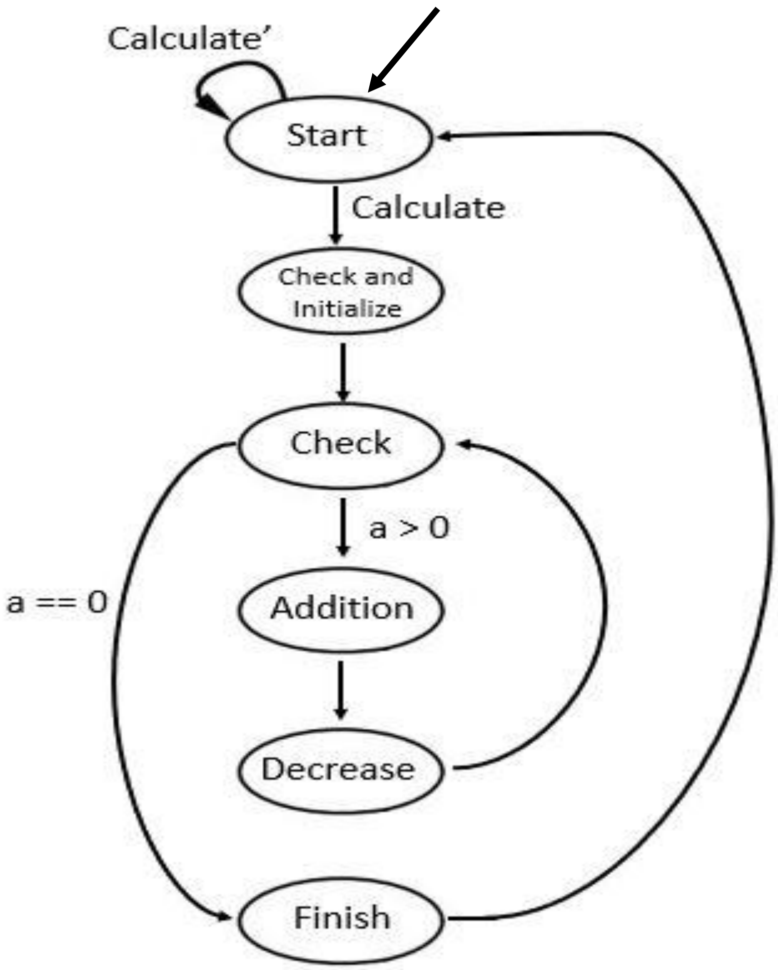
Solution Step by Step

1) Decide states and draw the state diagram for your FSM controller.

Information about the states;

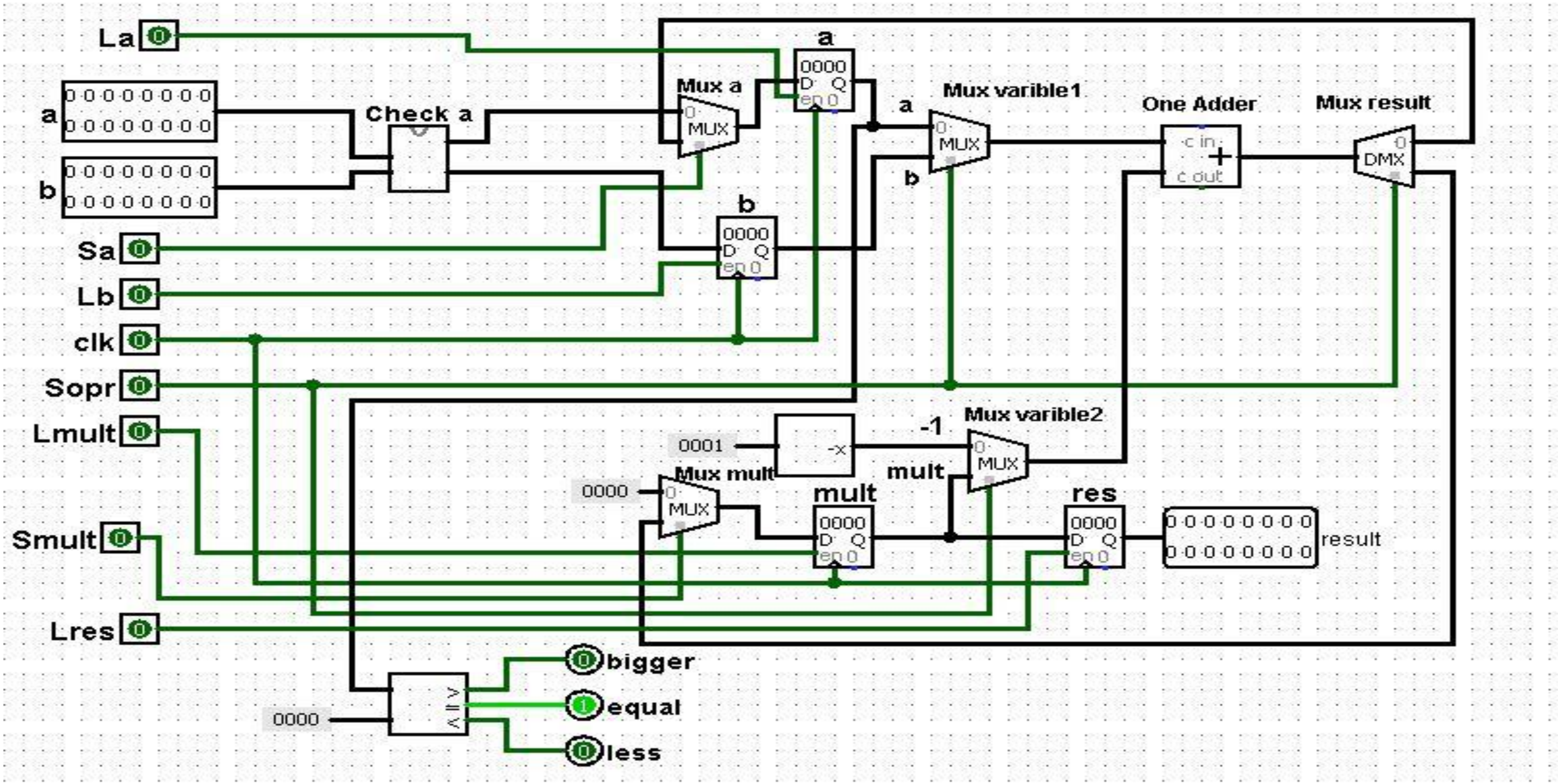
State Name	Description of the State
Start	This is the start state where calculate input is 0. Operation starts from this state.
Check and Initialize	In this state first it checks a. If a is less than 0, then it changes a and b. Also, it initializes mult register to be 0.
Check	In this state, it checks a. According to a, it determines next state.
Addition	In this state, it adds mult and b, after it writes the result in mult.
Decrease	In this state, it decreases the a input.
Finish	This is the finish state where a is 0. It writes mult to result register.

<u>Input:</u> Calculate



Step 1: Capture the FSM

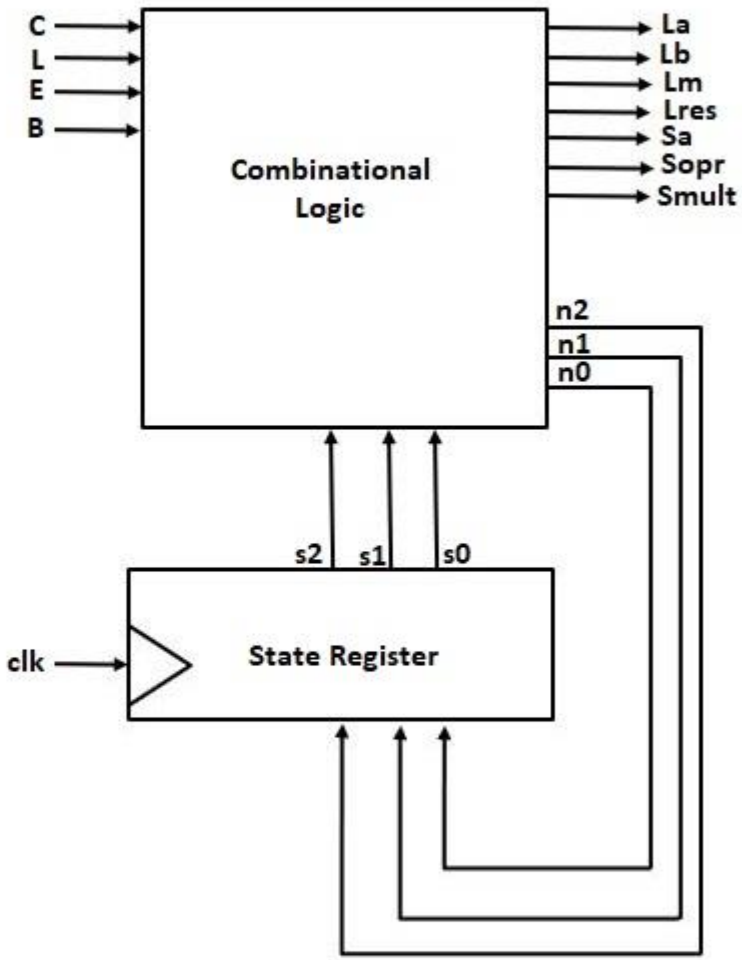
2) Draw datapath.



Optimization on Datapath

To optimize datapath I used one **adder** for addition and subtraction. To perform this I used two Mux for variables. According to current state muxes choose the variables and send them to the adder. For example in Addition state one mux choose b other mux choose mult and after addition result goes mult register by using Demultiplexer. Also, I used **Check a circuit** to change a and b, if a less than zero. **So, this datapath can perform negative number multiplications.**

Control Unit Architecture(FSM)



Inputs	Description of the Inputs
C	Calculate input from Button
L	Less than input from Datapath (a < 0)
E	Equal Input from Datapath (a == 0)
B	Bigger than Input from Datapath (a > 0)
clk	Clock Input
s2 s1 s0	Present State

Outputs	Description of the Outputs
La	Register a Load input from Datapath
Lb	Register b Load input from Datapath
Lm	Register mult Load input from Datapath
Lres	Register result Load input from Datapath
Sa	Mux a Select input from Datapath
Sopr	Mux variable Select input from Datapath
Smult	Mux mult Select input from Datapath
n2 n1 n0	Next State

Step 2A: Set up architecture

Step 2B: Encode states

Encoding ➔ Start=000, Check and Initialize=001, Check=010, Addition=011, Decrease = 100, Finish= 101

3) Draw truth table.

Inputs								Outputs										
S2	S1	S0	C	L	E	B		La	Lb	Lm	Lres	Sa	Sopr	Smult	n2	n1	n0	**Operation**
0	0	0	0	x	x	x		0	0	0	0	0	0	0	0	0	0	Start
0	0	0	1	x	x	x		0	0	0	0	0	0	0	0	0	1	Start
0	0	1	x	x	x	x		1	1	1	0	0	0	0	0	1	0	Initialize
0	1	0	x	1	0	0		0	0	0	0	0	0	0	1	0	1	Check
0	1	0	x	0	1	0		0	0	0	0	0	0	0	1	0	1	Check
0	1	0	x	0	0	1		0	0	0	0	0	0	0	0	1	1	Check
0	1	1	x	x	x	x		0	1	1	0	0	1	1	1	0	0	m=m+b
1	0	0	x	x	x	x		1	0	0	0	1	0	0	0	1	0	a=a-1
1	0	1	x	x	x	x		0	0	0	1	0	0	0	0	0	0	Finish

Step 2C: Fill in truth table

4) Derive Boolean expressions from the truth table.

- General Form

La = s2’s1’s0 + s2s1’s0’

Lb= s2's1's0 + s2's1s0

Lm = s2's1's0 + s2's1s0

Lres = s2s1's0

Sa = s2s1's0'

Sopr = s2's1s0

Smult = s2's1s0

n2 = s2's1s0'LE'B' + s2's1s0'L'EB' + s2's1s0

n1 = s2's1's0 + s2's1s0'L'E'B + s2s1's0'

n0 = s2's1's0'C + s2's1s0'LE'B' + s2's1s0'L'EB' + s2's1s0'L'E'B

- Simplifier form

  La = s2’s1’s0 + s2s1’s0’

  Lb= s2’s0

  Lm = s2’s0

  Lres = s2s1's0

  Sa = s2s1's0'

  Sopr = s2's1s0

  Smult = s2's1s0

  n2 = s2's1s0'LE'B' + s2's1s0'L'EB' + s2's1s0

  n1 = s2's1's0 + s2's1s0'L'E'B + s2s1's0'

  n0 = s2's1's0'C + s2's1s0'LE'B' + s2's1s0'L'EB' + s2's1s0'L'E'B

5) Draw the circuit on Logisim.

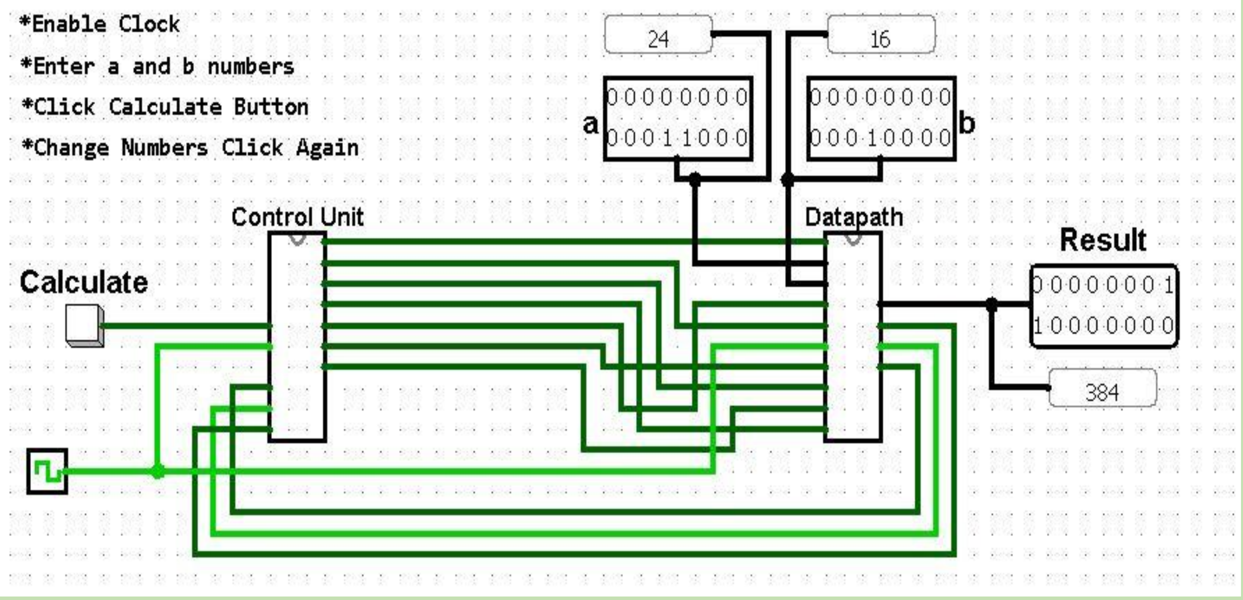
- Check **Multiplier** circuit in **171044098.circ** file to see circuit.

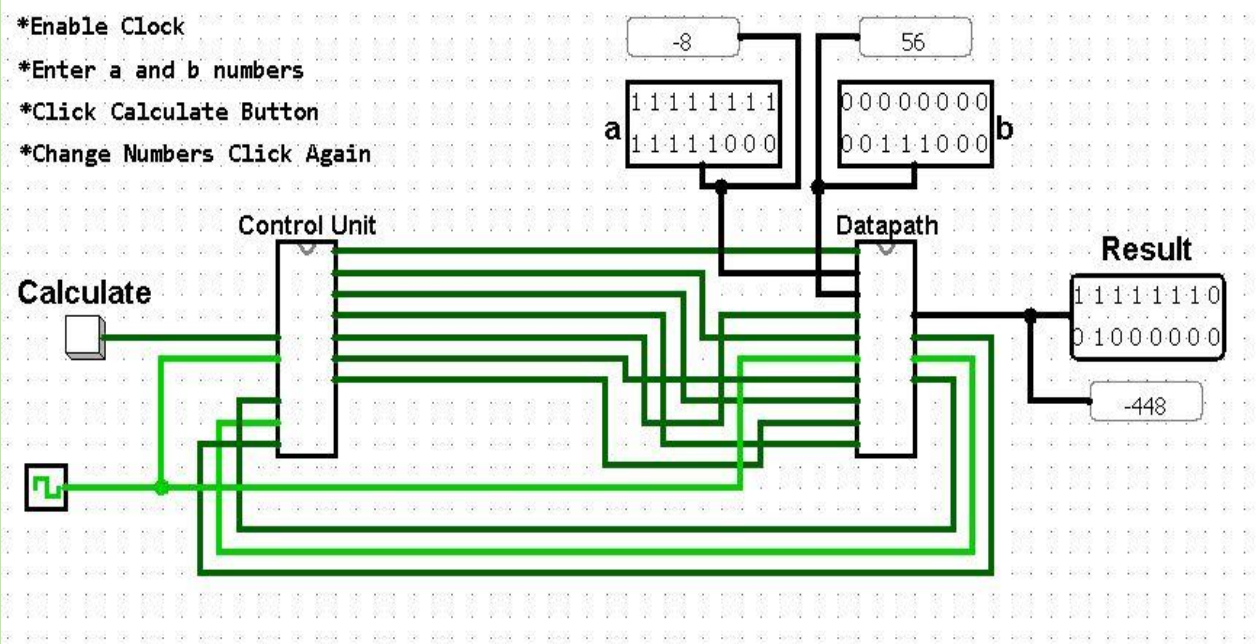
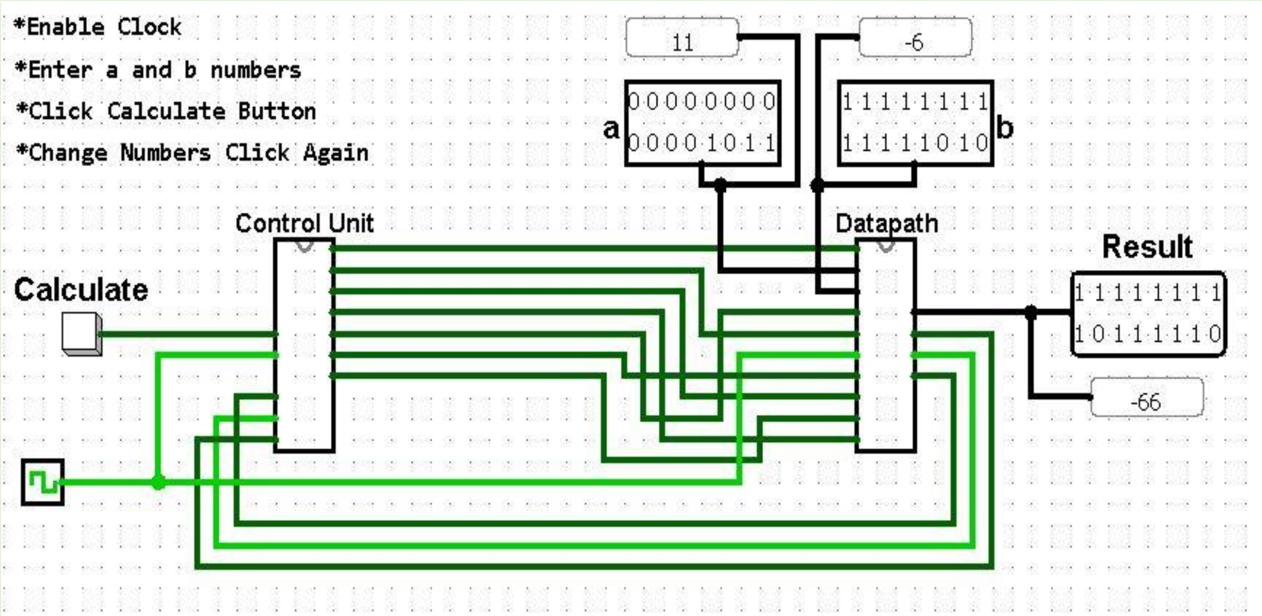
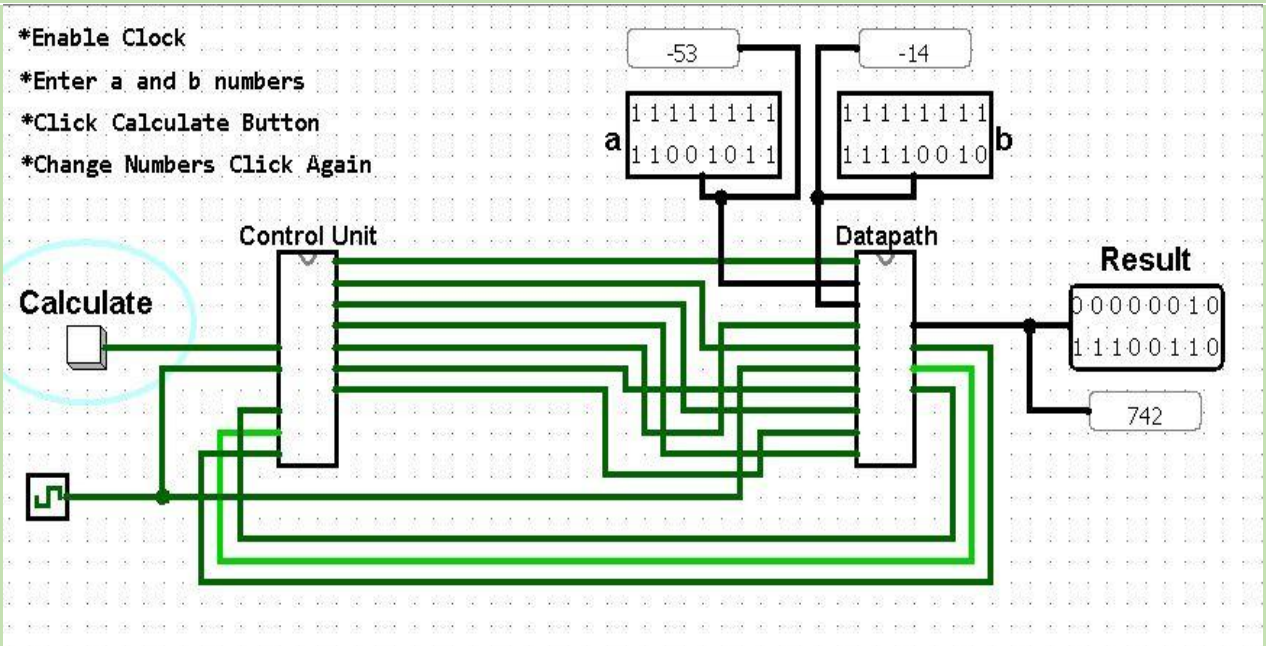
6) Simulate and see whether it works.

* This machine can perform negative number multiplications.

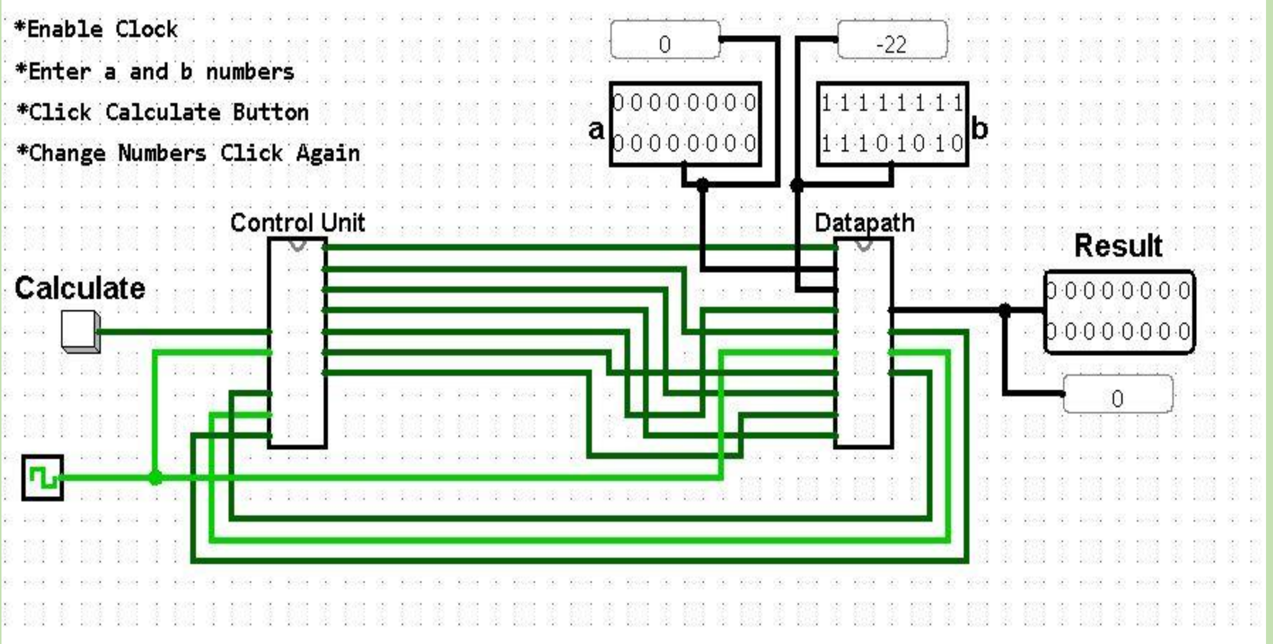
Note: I used 16-bit registers for inputs and outputs therefore, a 16-bit register can store 2^{16} different values. The signed range of integer values that can be stored in 16 bits is −32,768 (-1×2^{15}) through 32,767 ($2^{15} - 1$); the unsigned range is 0 through 65,535 ($2^{16} - 1$). (From Wikipedia) So, if you pass the range, the result might be wrong.

- Check following test results.

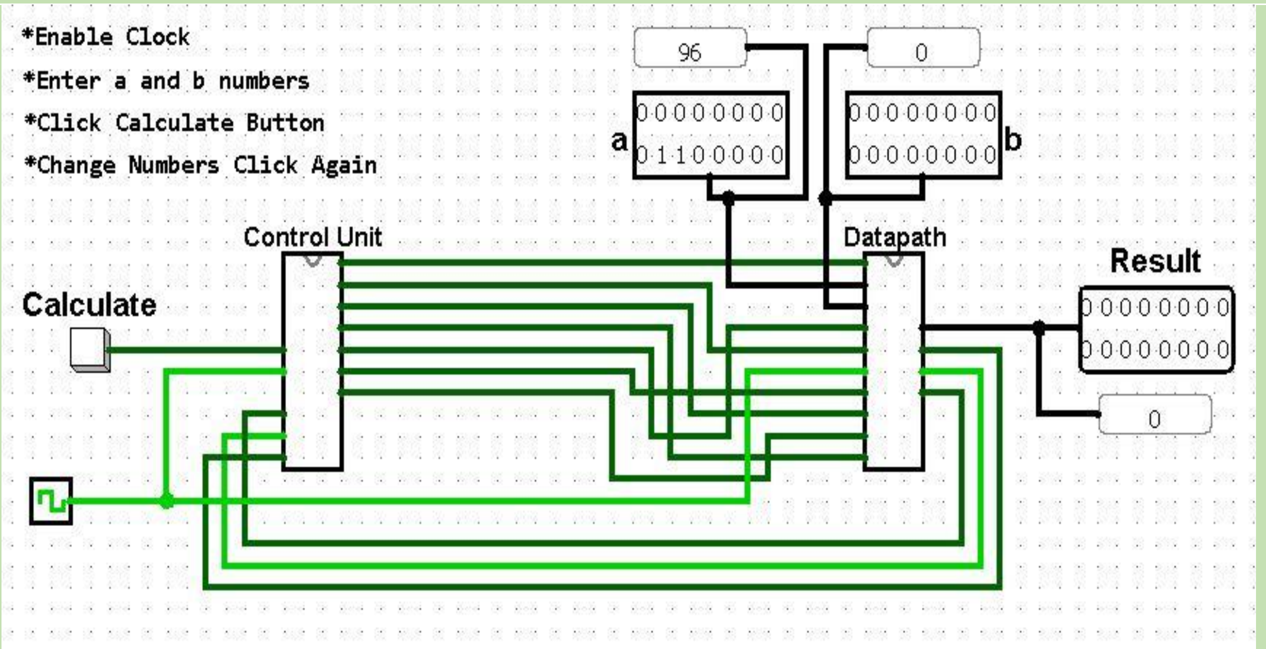
Description	Test Result
Two Positive Number	

One Positive One Negative Number	
One Positive One Negative Number (vice versa)	
Two Negative Number	

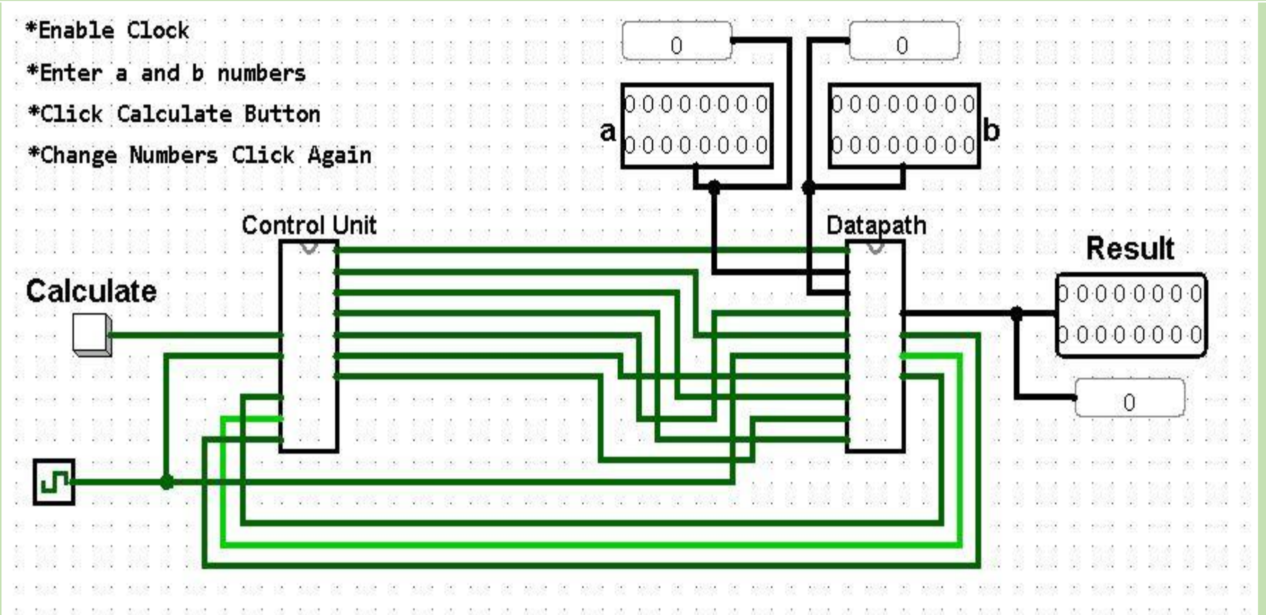
One zero number



Oner zero number(vice versa)



Two Zero Number



BONUS:

A checker circuit to perform negative number multiplications.