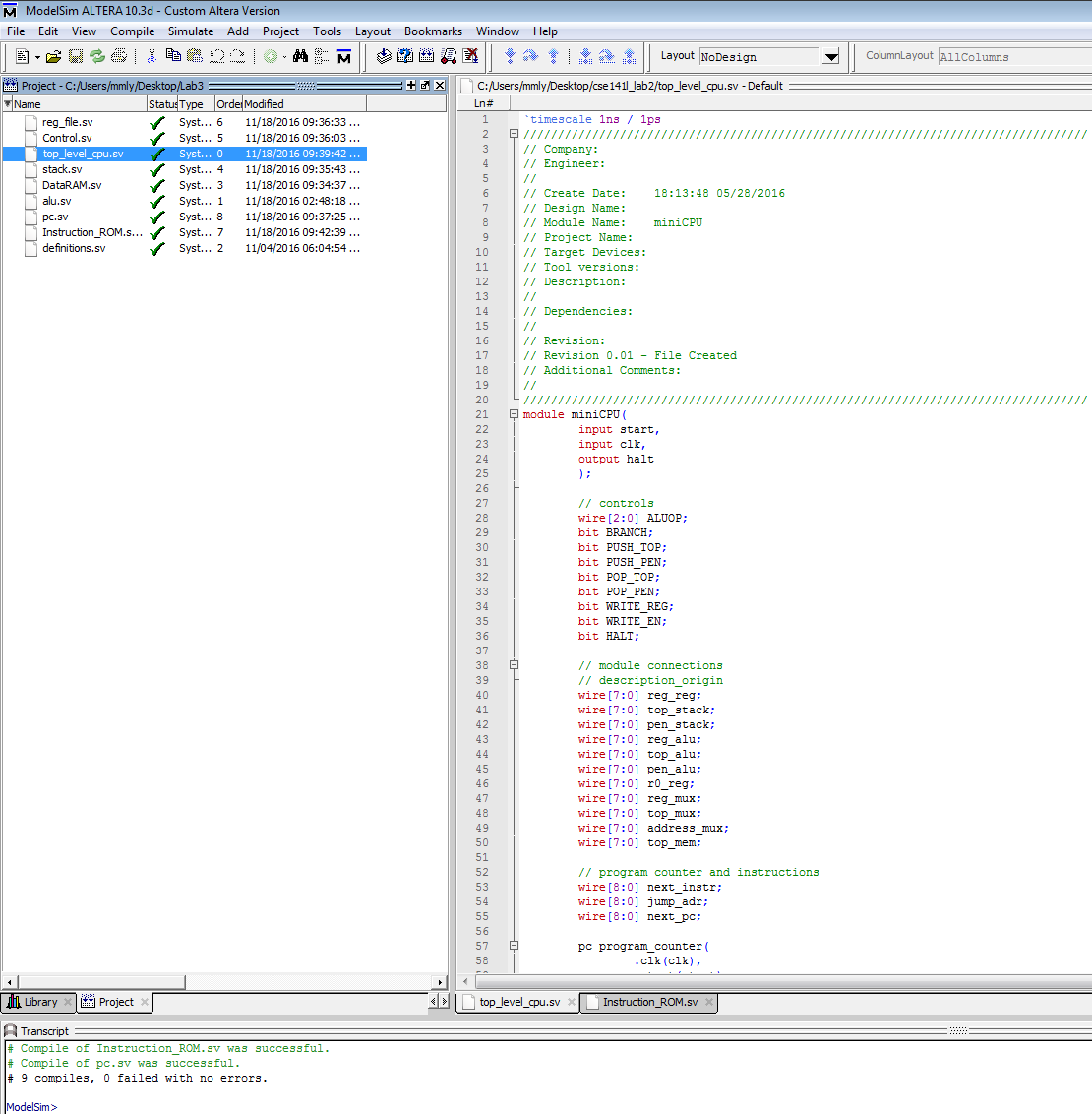
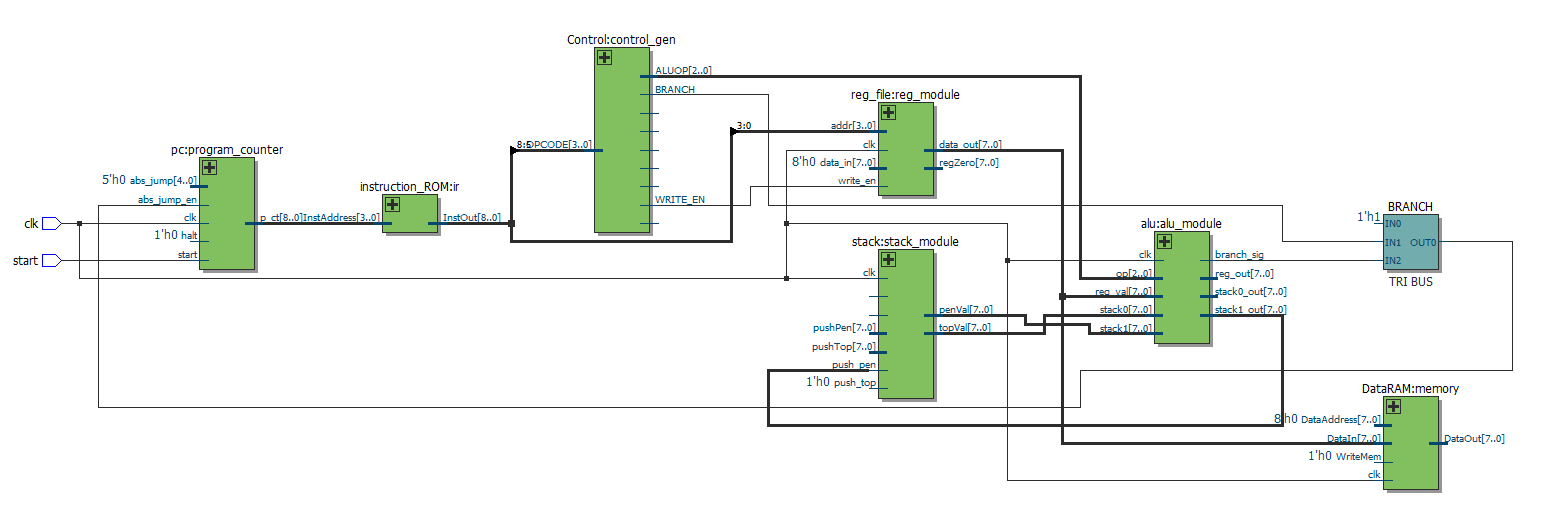
Lab 3

Melissa Ly, Sawandeep Kaur, Alan Chen

A11908353, A11601962, A12045305

To connect the modules, we began by declaring variables for every wire that would connect between modules. Then, we instantiated each module and assigned their inputs and outputs to the variables declared.





**Assembly Code:**

**Multiplier:**

push\_immediate 0

set r1 // i = 0

set r2 // lowSumAB = 0

set r3 // upperSumAB = 0

set r4 // sumLowerLowerABC = 0

set r5 // sumUpperLowerABC = 0

set r6 // sumUpperABC = 0

set\_memory\_ptr 1 // push A onto stack

set\_memory\_ptr 2 // push B onto stack

// ----------------------------------------------------------------------------------------------------------------

A\*B: // B is on the top of the stack, with A right below it

and\_and\_shift r1 // sum = A[i] & B; push sum >> 8-1, push sum << i

push r2

add r2// sumLowerAB = sum << i + r2; pop sum<<i and r2 and set overflow

add\_overflow r3// if sumLower += results in overflow, add it and sum >> 8-i to upper sum, pop

inc r1 // i++

// check for branch

push r1

push\_immediate 7

// state of the stack: 7, i, B, A, 0

blt A\*B // branch to beginning of loop unless i > 7 (i == 8)

// -------------------------------------------------------------------------------------------------------------------

push 0

set r1 // i = 0

set\_memory\_ptr 3 // push C onto stack

push r2 // push the lower bits of A\*B onto stack

// current state of the stack: lowerAB, C, 0

// there are more, but we'll forget about them for now because there are no requirements for memory leaks

// -----------------------------------------------------------------------------------------------------------------

lowerAB\*C: // same routine as before

and\_and\_shift r1 // sum = lowerAB[i] & C; push sum >> 8-1, push sum << i

push r4

add r4// sumLowerLowerABC = sum << i + r4; pop sum<<i and r4 and set overflow

add\_overflow r5 // if sumUpLowABC += results in overflow, add it and sum >> 8-i to upper sum, and pops

inc r1 // i++

// check for branch

push r1

push\_immediate 7

// state of the stack: 7, i, C, lowerAB, 0

blt lowerAB\*C // branch to beginning of loop unless i > 7 (i == 8)

// --------------------------------------------------------------------------------------------------------------------

push 0

set r1 // i = 0

set\_memory\_ptr 3 // push C back onto stack instead of popping; fewer instructions and less chance

//for mistakes

push r3 // push the upper bits of A\*B onto stack

// current state of the stack: upperAB, C, 0

// -------------------------------------------------------------------------------------------------------------------------

upperAB\*C: // same routine as before

and\_and\_shift r1 // sum = upperAB[i] & C; push sum >> 8-1, push sum << i

push r6

add r6// sumLowerLowerABC = sum << i + r6; pop sum<<i and set overflow

// don't care about the rest because it's overflow

inc r1 // i++

// check for branch

push r1

push\_immediate 7

// state of the stack: 7, i, B, A, 0

blt upperAB\*C // branch to beginning of loop unless i > 7 (i == 8)

// r7 will be the upper bits in location 4

push r5

push r6

add r7 // r7 = r5 + r6, or upper + upperLower

// r4 will remain the lower bits

set\_memory\_ptr 4 // move r0 to 4

store r7 // stores r7 into [r0]

set\_memory\_ptr 5

store r4 // stores r4 into [r0]

**Machine Code**

0101\_00000

0100\_00001

0100\_00010

0100\_00011

0100\_00100

0100\_00101

0100\_00110

0000\_00000

0000\_00001

1001\_00001

0010\_00010

0011\_00010

1010\_00011

1000\_00001

0010\_00001

0101\_00111

0111\_00000

0010\_00000

0100\_00001

0000\_00010

0010\_00010

1001\_00001

0010\_00100

0011\_00100

1010\_00101

1000\_00001

0010\_00001

0101\_00111

0111\_00001

0010\_00000

0100\_00001

0000\_00010

0010\_00011

1001\_00001

0010\_00110

0011\_00110

1000\_00001

0010\_00001

0101\_00111

0111\_00010

0010\_00101

0010\_00110

0011\_00111

0000\_00011

0001\_00111

0000\_00100

0001\_00100

**String Match**

push\_immediate 0

set r1 //bit index

push\_immediate 0

set r2 //number of matches in the array of strings

//push\_immediate 32

push\_immediate 0

set r3 // i = 0

set\_memory\_ptr 6 // 4-bit string

set\_memory\_ptr 32 //arr[0]

set\_memory\_ptr 64 //array length

push r3

loop:

pop 2

contains next

inc r2

next:

set\_memory\_ptr 6

set\_memory\_ptr 32

inc r0

set\_memory\_ptr 64 //array length

inc r3

push r3

blt loop

set\_memory\_ptr 7

store r4

**Machine Code**

0101\_00000

0100\_00001

0101\_00000

0100\_00010

0101\_00000

0100\_00011

0000\_00101

0000\_01001

0000\_01010

0010\_00011

0110\_00010

1011\_00100

1000\_00010

0000\_00101

0000\_01001

1000\_00000

0000\_01010

1000\_00011

0010\_00011

0111\_00011

0000\_00110

0001\_00100

**Closest Pair**

set\_memory\_ptr 255

set r1 //r1 = closest

push\_immediate 0

set r2 //r2 = i

push\_immediate 1

set r3 //r3 = 1

push\_immediate 19

set r4 //r4 = 19 = arr.length - 1

push\_immediate 20

set r5 //r5 = 20 = arr.length

set\_memory\_ptr 128

set r6 //r6 = a[0]

outerloop:

push r3

push r2

//branch if i < 1, so that if i = 0 the memory pointer is not incremented since it is already at 128 and //a[0] is at 128

blt incj

//set the memory pointer to the right location by incrementing the pointer until it reaches index i so //that it is at a[i]

set\_memory\_ptr 128 //reset memory pointer location to a[0]

push\_immediate 0

set r7 // r7 = 0 (counter)

incarray:

inc r0 //increment memory location to the appropriate location by looping

set r6 //until r7 = r2(counter = i)

inc r7

push r2

push r7

blt incarray //branch if r7 < r2 (counter < i )

incj:

push r2

push r3

add r8 //r8 = i + 1 = j

//loop through the array subtracting a[j] from a[i] and if it is less than the value stored in r1 (closest) //set r1 to this difference, continue looping while j < array length

innerloop:

push r6 //r6 = a[i]

inc r0 //r0 = a[j]

sub r9 //r9 = a[j] - a[i]

abs r9 //r9 = abs(a[j] - a[i])

push r1 //r1 = closest

blt endinner //branch if r1 < r9 (closest < abs(a[j] - a[i]))

push r9

set r1 //set r1 to r9 (closest = abs(a[j] - a[i])

endinner:

push r5

inc r8 //j++

push r8

blt innerloop //branch if r8 < r4 (j < 20)

//increment i and continue looping through the array if it is less than array length - 1 since the last //element in the array will already have been compared to the rest of the elements

push r4

inc r2 //i++

push r2

blt outerloop //branch if r2 < r4 ( i < 19)

//put the calculated closest distance into memory location 127

set\_memory\_ptr 127 //sets memory pointer to 127

store r1 //stores r1 to location 127 (r1 = closest)

**Machine Code**

0000\_01101

0100\_00001

0101\_00000

0100\_00010

0101\_00001

0100\_00011

0101\_10011

0100\_00100

0101\_10100

0100\_00101

0000\_01100

0100\_00110

0010\_00011

0010\_00010

0111\_00101

0000\_01100

0101\_00000

0100\_00111

1000\_00000

0100\_00110

1000\_00111

0010\_00010

0010\_00111

0111\_00110

0010\_00010

0010\_00011

0011\_01000

0010\_00110

1000\_00000

1100\_01001

1101\_01001

0010\_00001

0111\_00111

0010\_01001

0100\_00001

0010\_00101

1000\_01000

0010\_01000

0111\_01000

0010\_00100

1000\_00010

0010\_00010

0111\_01001

0000\_01011

0001\_00001