CS250,
Liane Yanglian (net ID:xy48),
Professor Sorin,
Assignment 4

Summary

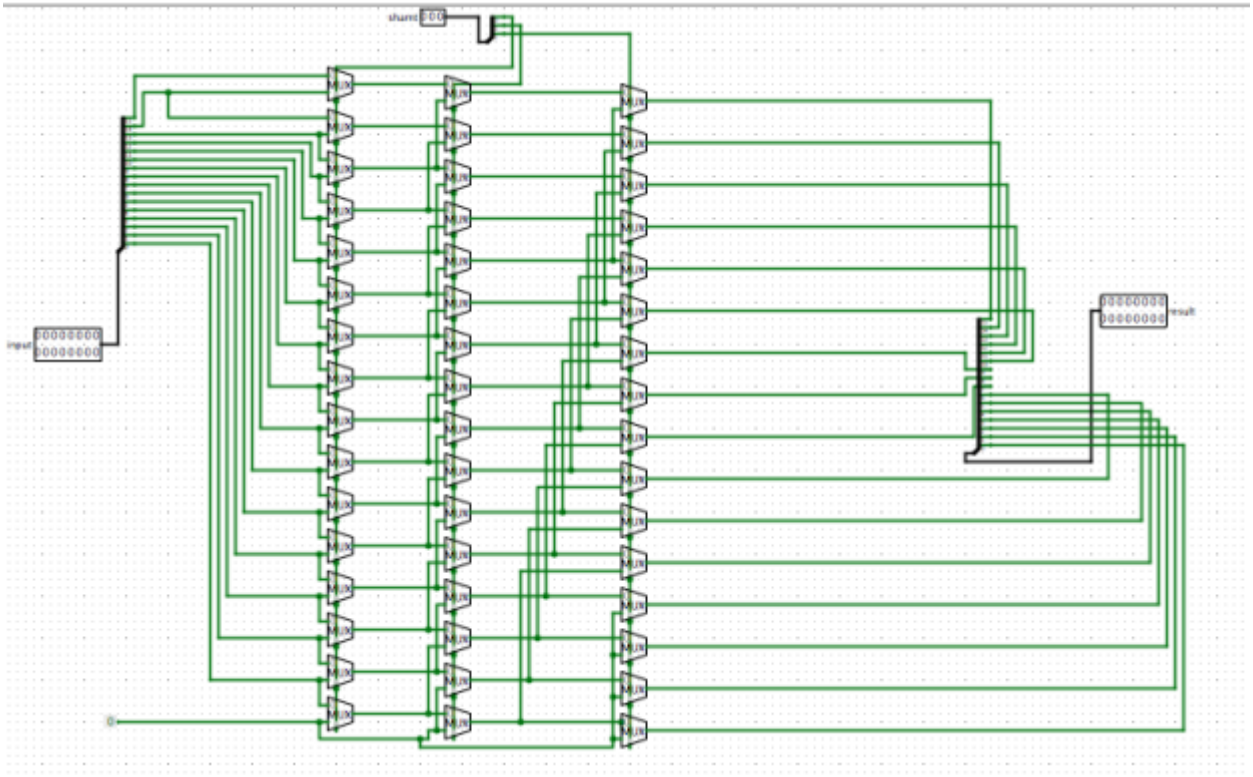
All parts of my processor work. My processor passed all tests, including the one with the file example.s. I also made other test cases to test my processor, including on that tested my jr instruction.

I explained the subscircuits and the main circuit with screenshots of my processor below before delving into the control table and explaining each control, after which I elaborated on how the control subcircuit is built.

Adder/Subtractor

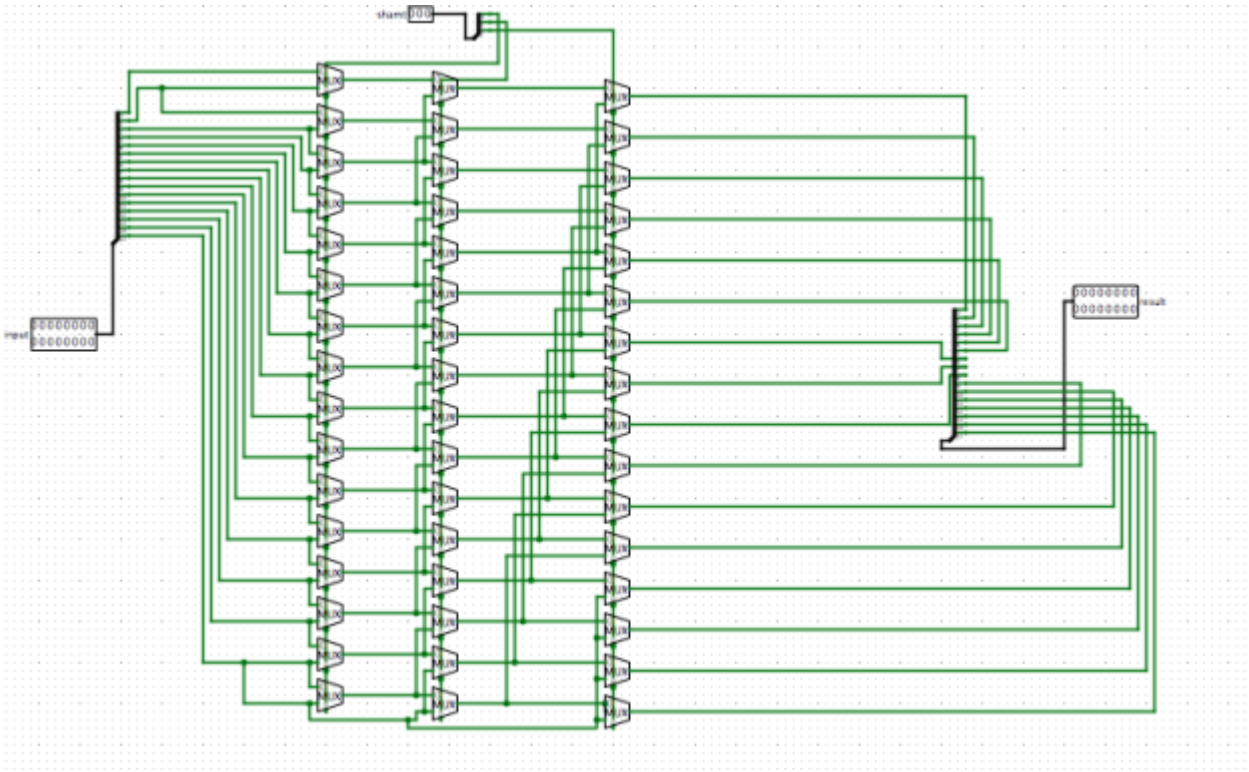
For my 16 bit adder/subtractor, I took the 32 bit adder/subtractor I wrote for the last assignment and reduced the dimension by half.

Left Shifter



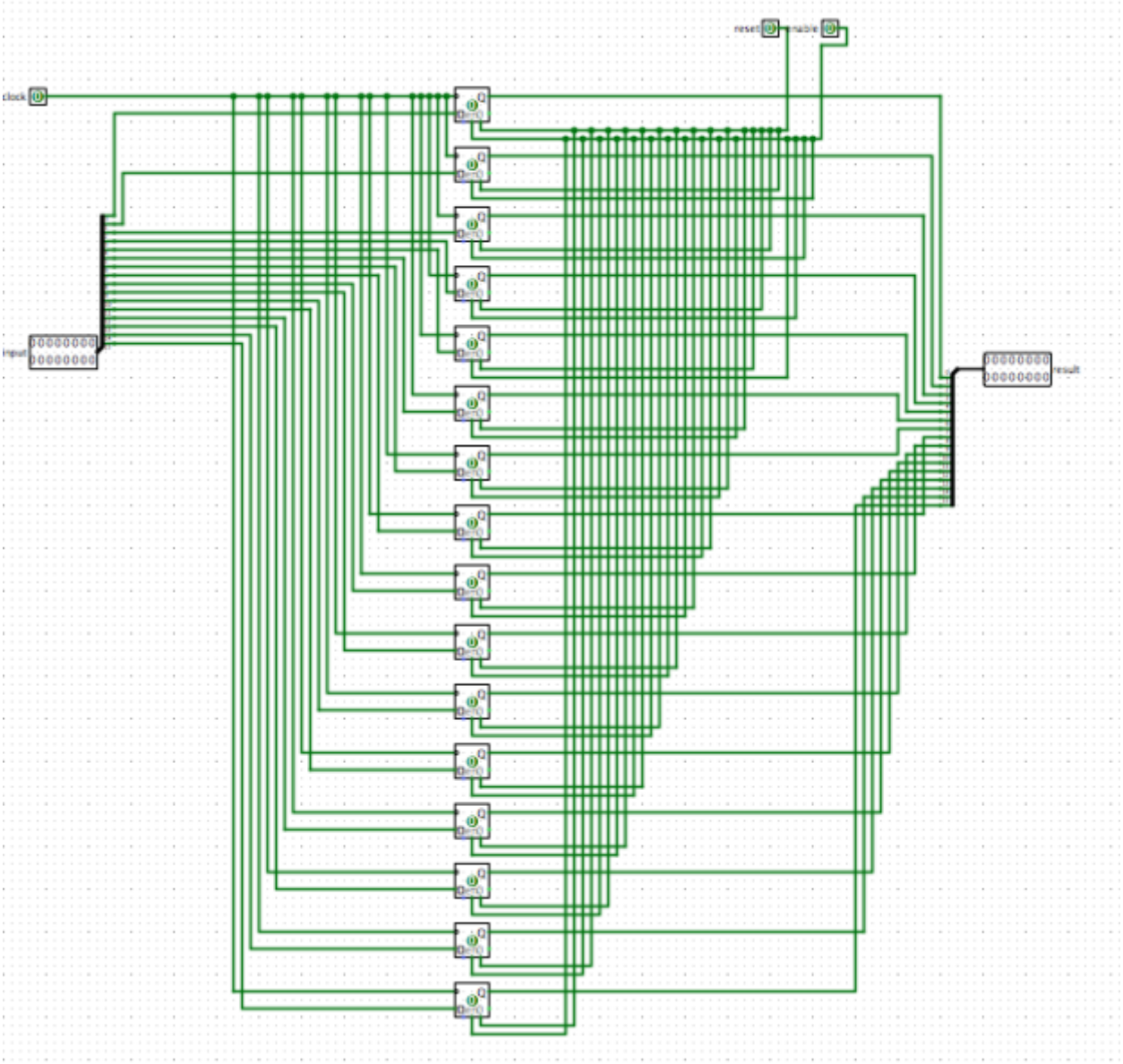
For my left shifter, I split the 16 bit input into 16 bits using a splitter, then I use three muxes. I followed the left shifter example on the lecture slides.

Right Shifter



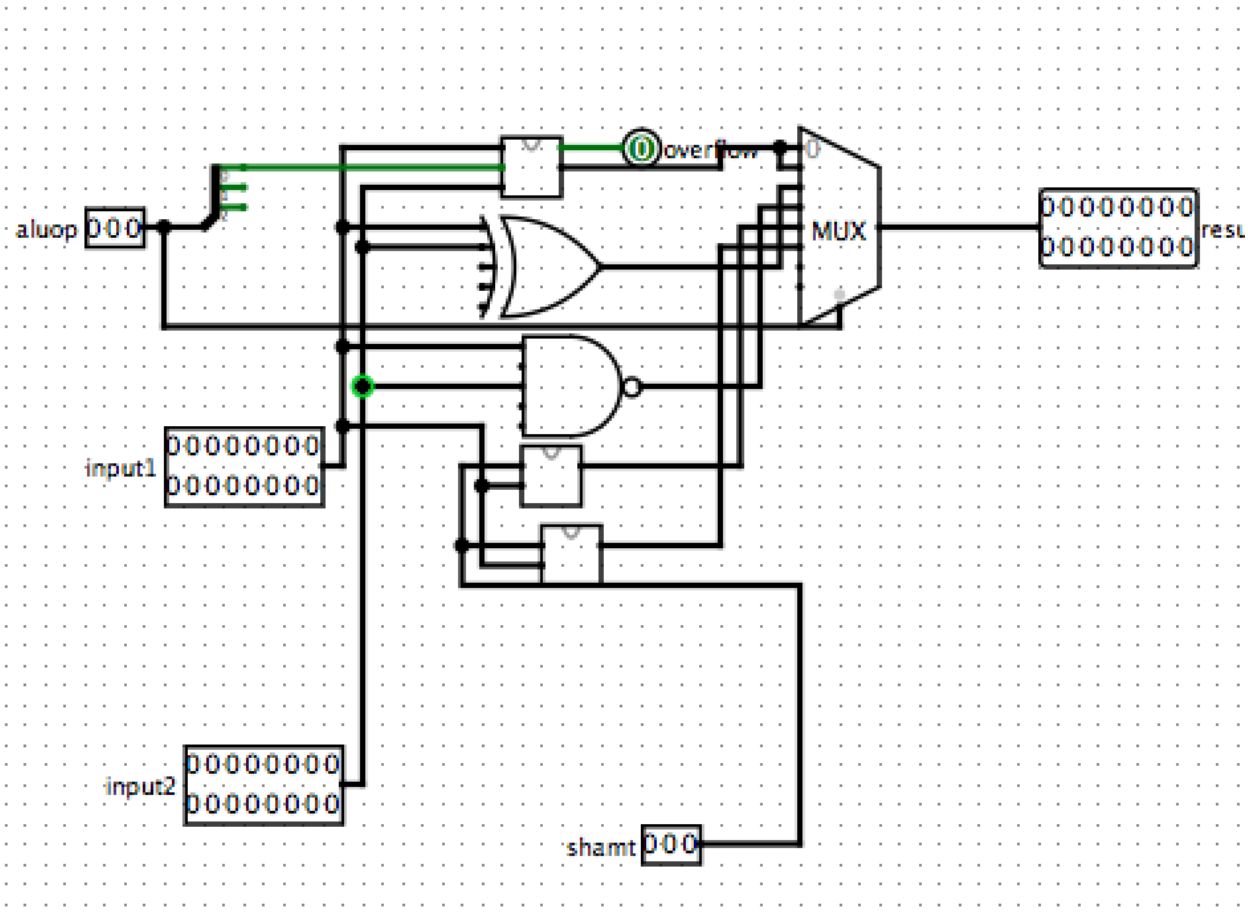
For the right shifter, I did exactly the same thing as I did for left shifter, except for a few changes. Instead of connecting 0 to input 2 of the “last first mux,” last second mux, and the last third mux, I connect input2 to the penultimate first mux to them. Also, I flip the bits for the input and output such that the order of the bits is reversed.

Register



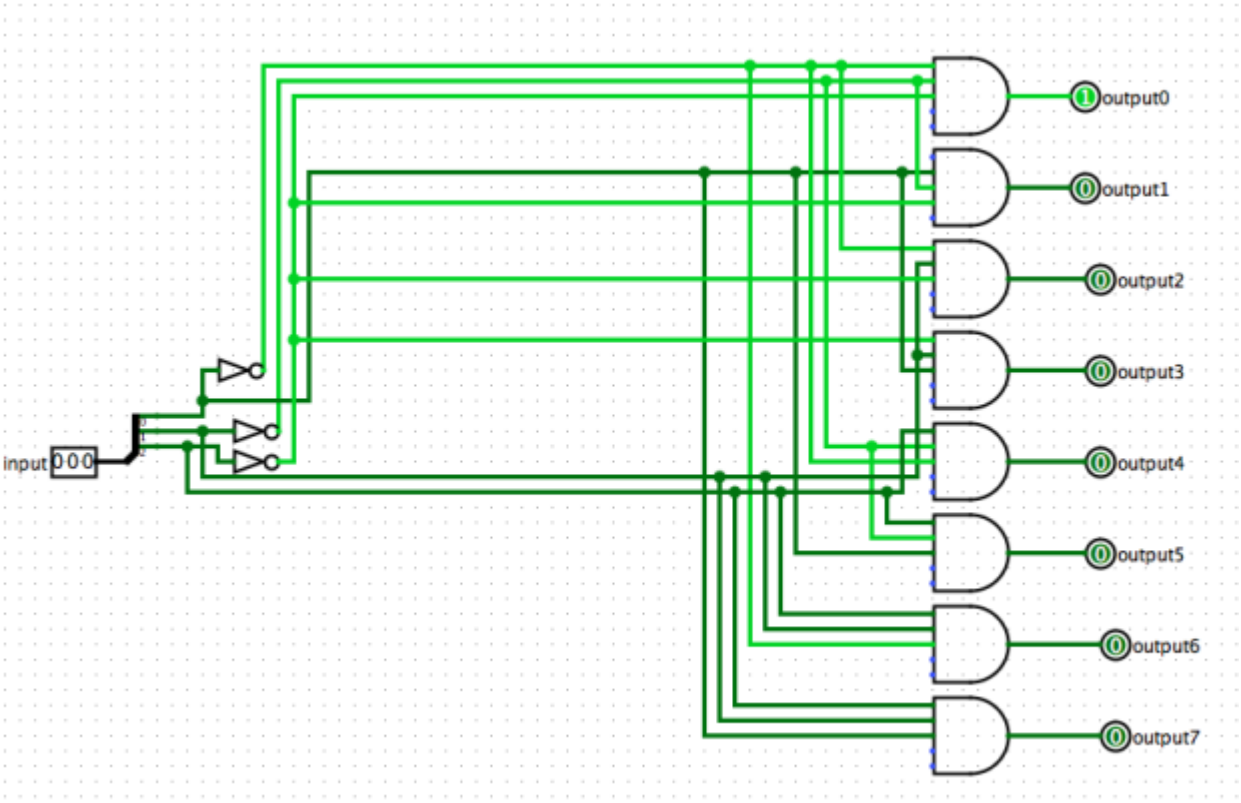
I split the input into 16 bits and used 16 D flip flops, and have a clock, reset, and enable input for each of them.

ALU



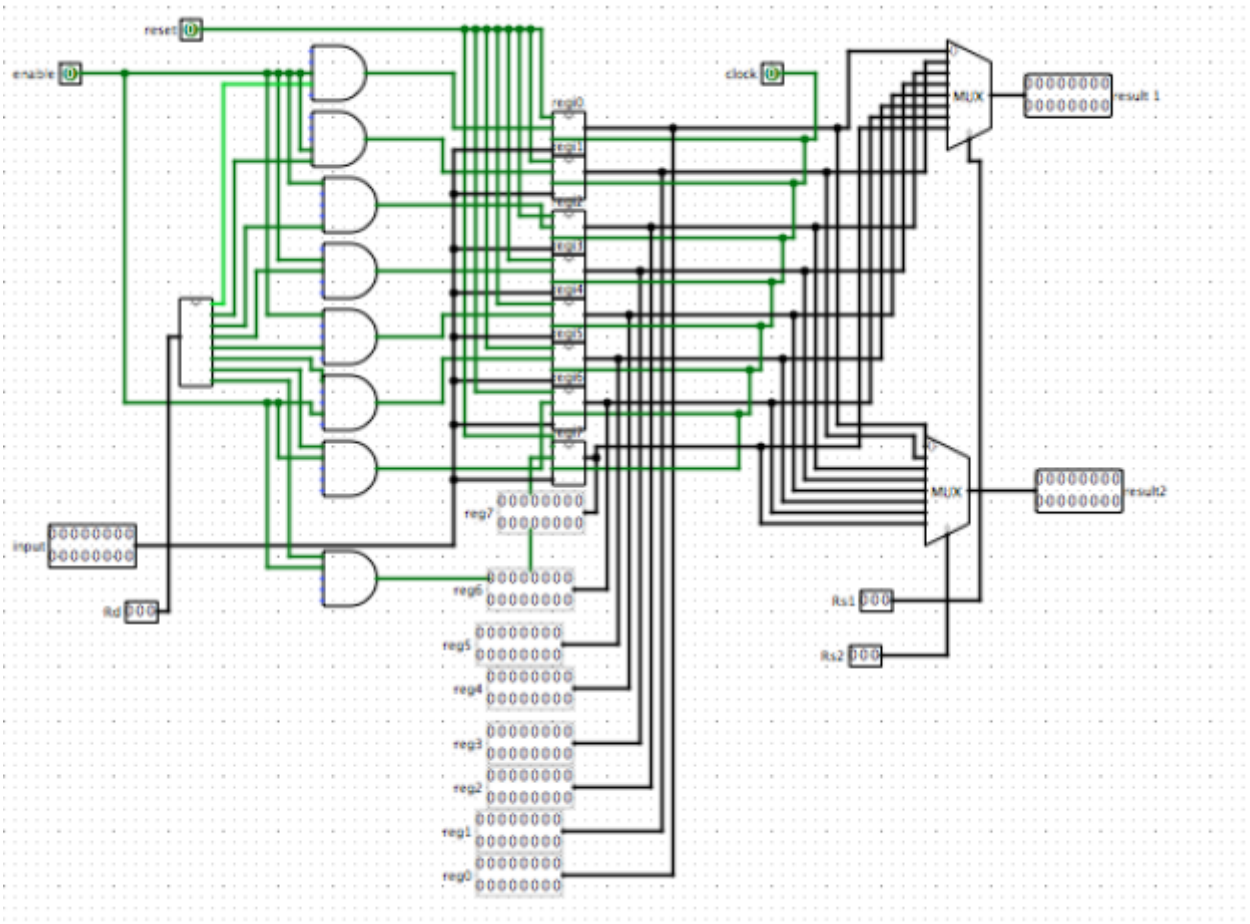
For ALU, I included the 16 bit adder/subtractor, xor, nand, and left and right shifters. I used the aluop's oth bit to control whether the adder/subtractor is doing addition or subtraction. The ALU takes in two 16 bit inputs.

Decoder



My decoder takes in a 3 bit input and then use and gates to produce 8 outputs such that the output corresponds to the input (when the input is 0, the output is 0, and so on).

Regfile



For regfile, I take in a 16 bit input, a Rd, Rs1, and Rs2, and use two muxes to produce a result1 and result2. I used the decoder here.

Main Circuit

Fetch

I used the register I built as the PC and the built-in instruction memory. I connected them according to the method covered in class. I used my adder/subtractor for PC+1.

Datapath

For the datapath, I used the register file I built and muxes as required by the instructions in the instruction set. I used an ALU to carry out the instructions **nand, xor, addi, add, sub, shra, shl, lw,** and **sw**.

For **bgt** and **beqz**, I followed the slide on beq but did not include a shifter. I also built subcircuits for bgt and beqz to see if $rs>$rt, in the first case, or if $rs = 0, in the second case.

For **j, jr, jal**, I followed the slides and added a mux for each instruction. I also have a subcircuit to extend the 12 bit address to 16 bit by adding the first four bits of the current PC.

For **input**, I used a keyboard input and used a mux. The control of the mux is my input control, where if the data is invalid, the input is 128; if the data is valid, the input is as required by the assignment.

For **output**, I used a TTY display and used the rs1 value as the input before changing it to 7 bits.

Controls

Controls Table

Instructions	Controls													
	aluop	rdst	we	alu in b	s/lwselect	swselect	beqz	bgt	beqz/bgt	j	jr	jal	input	output
nand	011	0	1	0	x	x	x	x	x	x	x	x	x	x
xor	010	0	1	0	x	x	x	x	x	x	x	x	x	x
addi	000	1	1	1	x	x	x	x	x	x	x	x	x	x
add	000	0	1	0	x	x	x	x	x	x	x	x	x	x
sub	001	0	1	0	x	x	x	x	x	x	x	x	x	x
shra	100	0	1	x	x	x	x	x	x	x	x	x	x	x
shl	101	0	1	x	x	x	x	x	x	x	x	x	x	x
bgt	001	x	0	0	x	x	0	1	1	x	x	x	x	x
beqz	x	x	0	0	x	x	1	0	1	x	x	x	x	x
lw	000	1	1	1	1	0	x	x	x	x	x	x	x	x
sw	000	1	0	1	1	1	x	x	x	x	x	x	x	x
j	x	x	0	x	x	x	x	x	x	1	0	0	x	x
jr	x	x	0	x	x	x	x	x	x	0	1	0	x	x
jal	x	x	1	x	x	x	x	x	x	0	0	1	x	x
output	x	x	0	x	x	x	x	x	x	x	x	x	x	1
input	x	x	1	x	x	x	x	x	x	x	x	x	1	x

nand

- Aluop is 011 according to how I built my ALU.
- Rdst is 0 so $rd is chosen.
- We (write enable) is 1 so registers can be written to.

- Alu in b is 0 so immediate is not chosen.
- The other controls are not relevant.

xor

- Aluop is 010 according to how I built my ALU.
- Rdst is 0 so $rd is chosen.
- We (write enable) is 1 so registers can be written to.
- Alu in b is 0 so immediate is not chosen.
- The other controls are not relevant.

addi

- Aluop is 000 according to how I built my ALU.
- Rdst is 1 so $rt is chosen.
- We (write enable) is 1 so registers can be written to.
- Alu in b is 1 so immediate is chosen.
- Since I need to extend the 6 bit immediate to 16 bits, I use a subcircuit named sign extender that take the 0-5th bits from the immediate as its 0-5th bits and
- The other controls are not relevant.

add

- Aluop is 000, same as that for addi, because both operations need to perform addition.
- Rdst is 0 so $rd is chosen.
- We (write enable) is 1 so registers can be written to.
- Alu in b is 0 so immediate is not chosen.
- The other controls are not relevant.

sub

- Aluop is 001 according to how I built my ALU.
- Rdst is 0 so $rd is chosen.
- We (write enable) is 1 so registers can be written to.
- Alu in b is 0 so immediate is not chosen.
- The other controls are not relevant.

shra

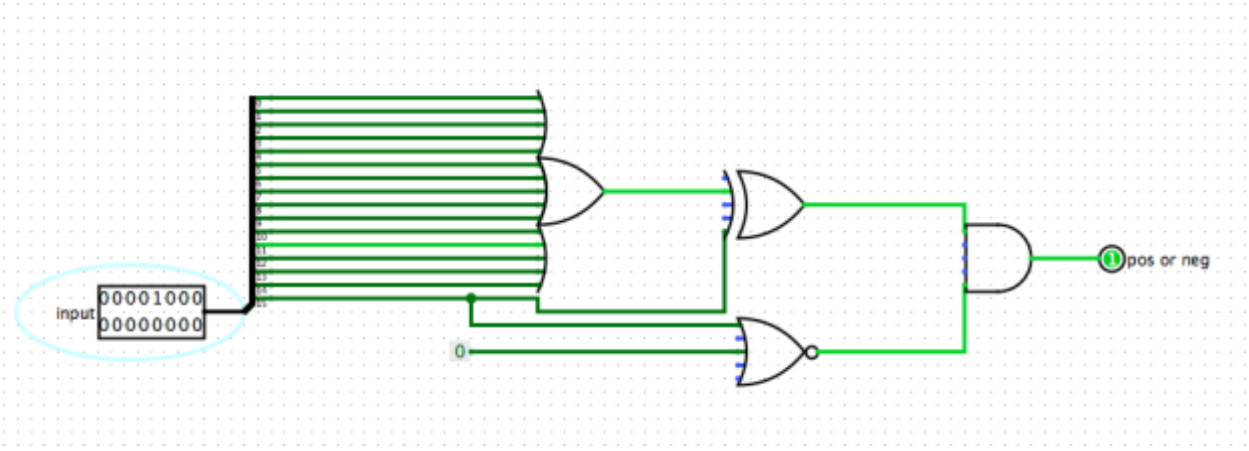
- Aluop is 100 according to how I built my ALU.
- Rdst is 0 so $rd is chosen.
- We (write enable) is 1 so registers can be written to.
- The other controls are not relevant.

shl

- Aluop is 101 according to how I built my ALU.
- Rdst is 0 so $rd is chosen.
- We (write enable) is 1 so registers can be written to.
- The other controls are not relevant.

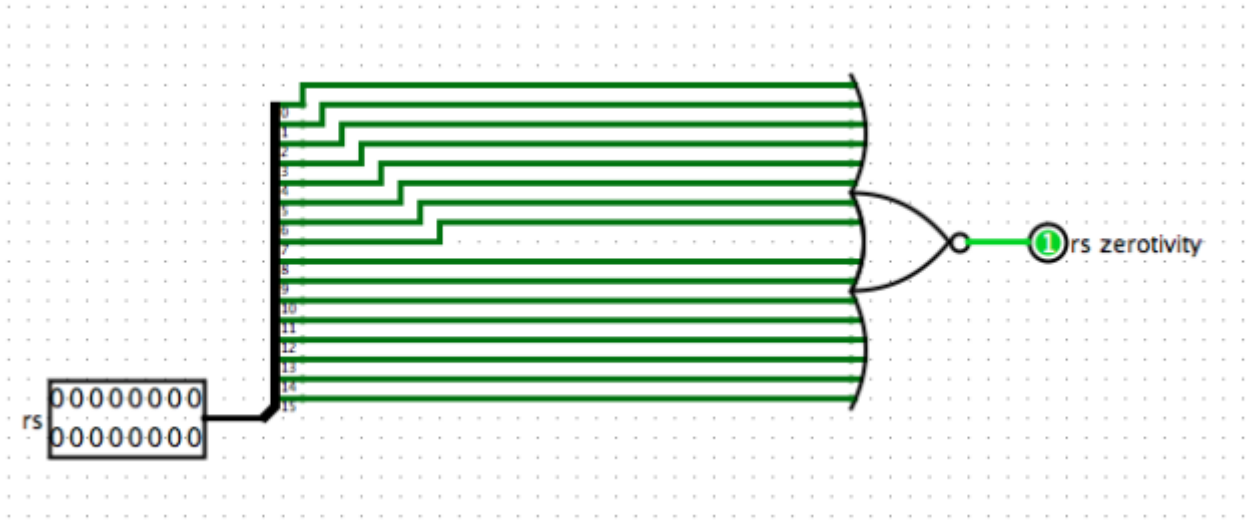
bgt

- Aluop is 001 because that's the aluop for the instruction sub, and here I need to do subtraction: $rs-$rt to see if the difference is greater than 0.
- We (write enable) is 0 because no registers should be written to.
- Alu in b is 0 so immediate is not chosen.
- Beqz signal is 0 so beqz is not carried out.
- Bgt signal is 1 to indicate that the instruction is bgt.
- Beqz/bgt signal is 1 to indicate one of the two instructions is being carried out.
- A subcircuit named bgt difference calculator is built so that the output is 1 when the 15th bit is 0 and at least one bit in the rest of the 15 bits is 1, as shown below.
- The other instructions are irrelevant.



beqz

- We (write enable) is 0 because no registers should be written to.
- Alu in b is 0 so immediate is not chosen.
- Beqz signal is 1 to indicate that the instruction is beqz.
- bgt signal is 0 so bgt is not carried out.
- Beqz/bgt signal is 1 to indicate one of the two instructions is being carried out.
- I built a subcircuit named beqz output which outputs 1 if all bits of the 16 bit input is 0, as shown below.
- The other instructions are irrelevant.



lw

- Aluop is 000 because that is the aluop for add and I need to do addition: $rs+Imm.
- Rdst is 1 so $rt is chosen.
- We (write enable) is 1 so registers can be written to.
- Alu in b is 1 so immediate is chosen.
- S/lwselect is 1 so I know either lw or sw is being executed.
- Swselect is 0 so I know sw is not the instruction so I carry out lw.
- The other instructions are irrelevant.

sw

- Aluop is 000 because that is the aluop for add and I need to do addition: $rs+Imm.
- Rdst is 1 so $rt is chosen.
- We (write enable) is 0 because registers should not be written to.
- Alu in b is 1 so immediate is chosen.
- S/lwselect is 1 so I know either lw or sw is being executed.
- Swselect is 1 so I know sw is the instruction I carry out, not lw.
- The other instructions are irrelevant.

j

- j is 1 to indicate the instruction is j.
- jr and jal are 0 so they do not interfere with j.
- I use a mux, with input 0 as the ouput from the mux I used for beqz/bgt and input 1 as the extended address. The control is j, so when j is 1, the extended address is chosen, i.e. PC=L.
- To obtain the extended address, I used a subcircuit named address&PC extender. It used the 12-15 bits from the current PC as the 12-15 bits of my extended address and the other 12 bits is the address I obtained from the fetch portion of my circuit.
- We (write enable) is 0 because no registers should be written to.
- The other instructions are irrelevant.

jr

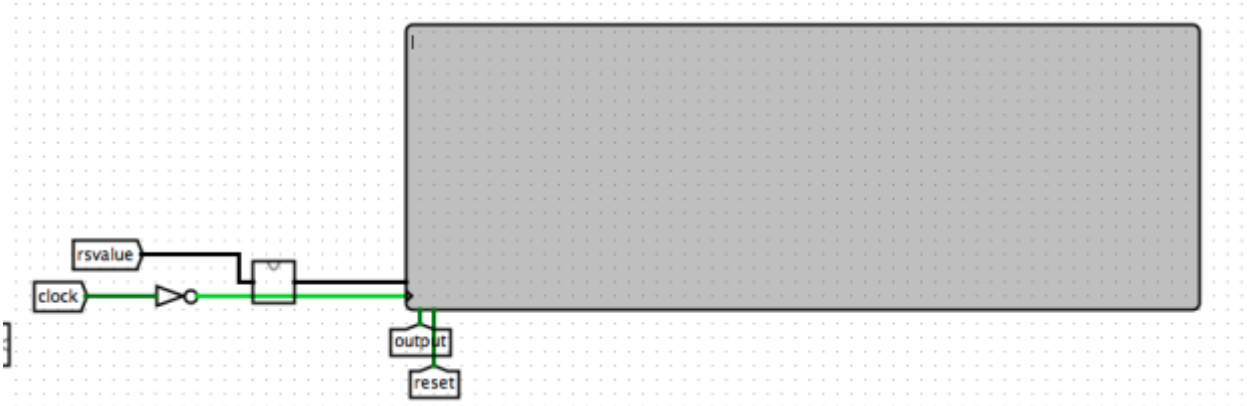
- jr is 1 to indicate the instruction is jr.
- j and jal are 0 so they do not interfere with jr.
- I use a mux, with input 0 as the ouput from the mux I used for j and input 1 as the rsvalue I obtained as the first ouput from my regfile. The control is jr, so when jr is 1, the rsvalue is chosen, i.e. PC=$rs.
- We (write enable) is 0 because no registers should be written to.
- The other instructions are irrelevant.

jal

- jal is 1 to indicate the instruction is jal.
- j and jr are 0 so they do not interfere with jal.
- I use a mux, with input 0 as the ouput from the mux I used for jr and input 1 as the extended address with jal as the control such that when jal is 1, the extended address is chosen, i.e. PC=L.
- I use a mux for the input to regfile, with jal as the control, input 0 as the original input to regfile and input 1 as PC+1 such that when jal is 1, $r7 = PC+1.
- I used a mux before the mux controlled by rdst, with jal as the control, input0 as rd and input1 as the constant 7 such that when jal is 1, $r7 is written to.
- We (write enable) is 1 so I can write to registers.
- The other instructions are irrelevant.

output

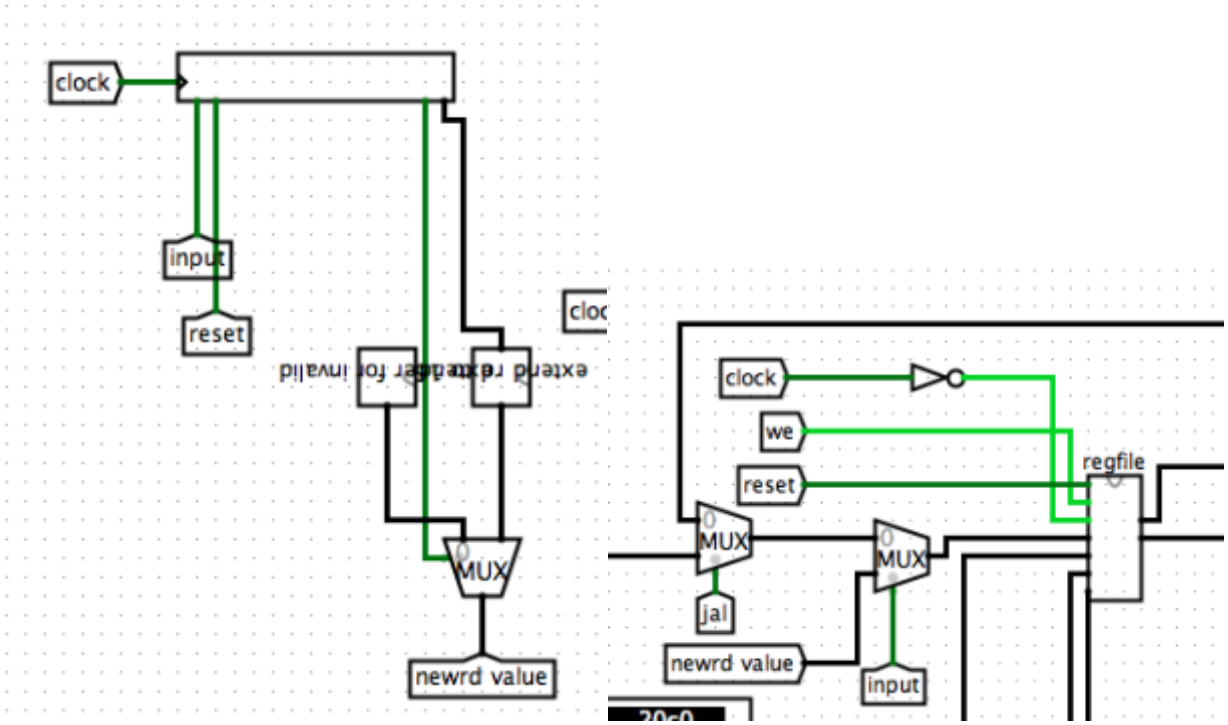
- Output is 1 to indicate the instruction is output.
- As shown below, I used a TTY display, with falling edge, and output as the control. I connect the first ouput of my regfile, named rsvalue, to a subcircuit named from 16 to 7
- We (write enable) is 0 because no registers should be written to.



input

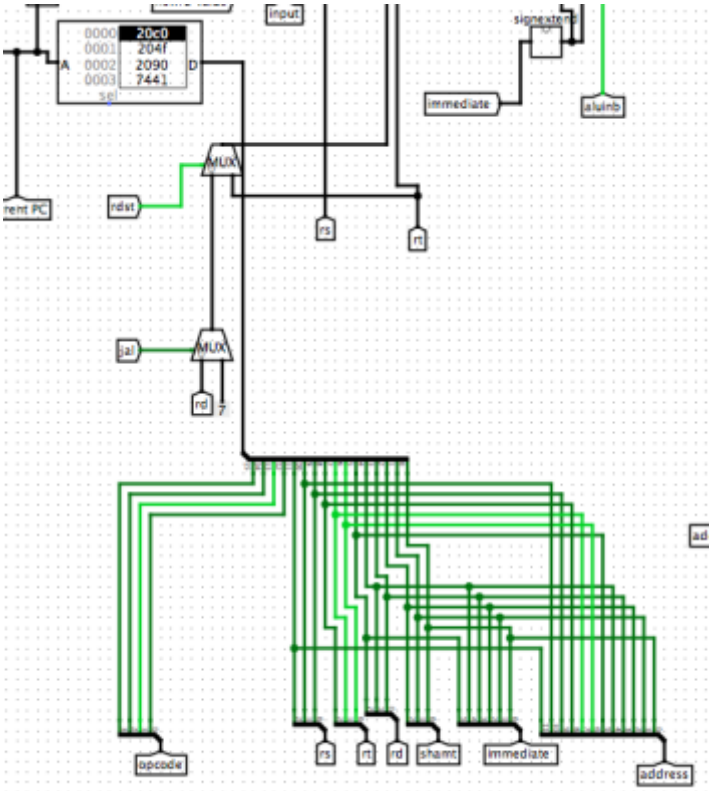
- Input is 1 to indicate the instruction is input.
- We (write enable) is 1 so registers can be written to.
- As shown below, I set the keyboard to rising edge, with input as the control. I used the available output from the keyboard as the control for a mux which takes as input0 the constant 128, a subcircuit I constructed, and takes as input1 the 7 bit output from the keyboard now extended to 16 bits.

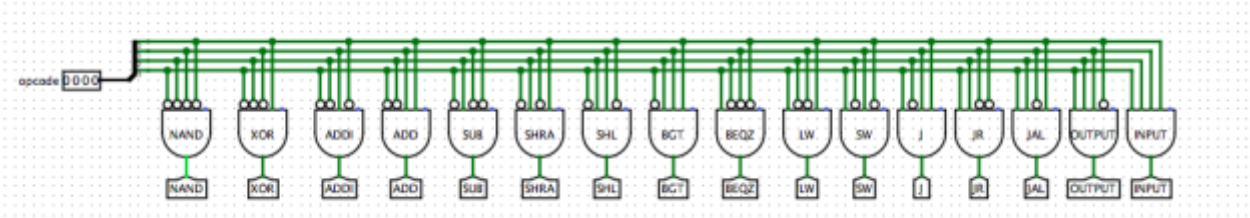
- I constructed a subcircuit to extend the output from the keyboard from 7 bit to 16 bits named extend rd to 16, which takes the 0-7th bits from the keyboard output and make 8-15th bits 0.
- The output from my mux is named newrd value.
- Also shown below, I added a mux in front of regfile so that the control is input, input0 is the output from the jal mux, and input1 is the newrd value such that when input is 1, newrd value will be selected as the input to regfile.



The Subcircuit Control

- As shown below, I used a splitter in the main circuit to get the opcode, rs, rt, rd, shamt, immediate, and address.
- With the opcode obtained, in the subcircuit named control, I then split the opcode into 4 bits and use 16 and gates, with each one representing 1 instruction, to connect the 4 bits to the 16 gates so that each instruction's opcode matches that given by the assignment instructions.
- The outputs from the 16 add gates are 16 tunnels labeled as the name of the 16 instructions. Then, I used an or gate for each control that I have. If, for an instruction that control should be 1, then I connect the instruction to that control. If the control should be 0 or does not matter, I do not connect the two, as shown below.


input
0000 20c0
0001 204f
0002 2090
0003 7441
A
D
sel
signextend
immediate
alulnb
MUX
rdst
rs
rt
jal
MUX
rd
7
ad
opcode
rs
rt
rd
shamt
immediate
address


opcode 0 0 0 0
NAND
XOR
ADDI
ADD
SUB
SHRA
SHL
BGT
BEQZ
LW
SW
J
JR
JAL
OUTPUT
INPUT