Nicholas Allaire, A10639753
Nathaniel Perkins, A11727588
CSE 141L
5/12/17

Lab 2 Write-up

1. **Introduction:**
    a. We modeled our implementation of the different processor elements after the examples given on TED. The main differences were in our alu and register file. For our alu, we have specialized instructions based on our ISA from lab 1. Our register file is very different, as we used one of the registers as an index into which block of registers we were accessing.

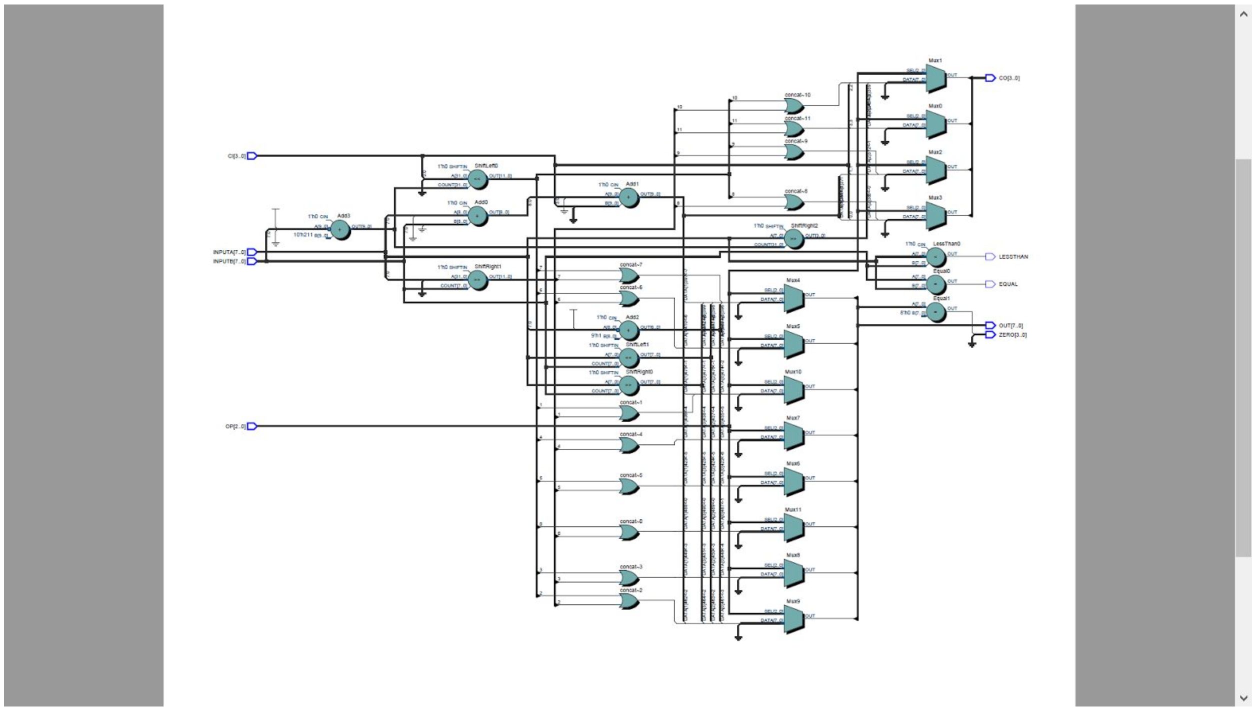
2. **Summarize ISA From Lab 1**
    a. Our ISA from lab 1 applies the concept of an accumulator ISA by using reg0 as an implicit register. Reg1 is also a special register such that it allows us to switch between different register blocks by changing the value between 0, 1, or 2. Reg1 allows us to support 16 registers instead of 8 registers. The operations supported by our ISA are: branch less than(blt), branch equal (beq), shift left (sl), shift right (sr), shift right pad with overflow (sro), set (set), set immediate (sti), halt (halt), load word (lw), store word (sw), add (add), 2's complement (comp), move (mov), jump (j), and clear overflow (clr).
        i. blt regX, regY: if regX < regY then go to the label stored in reg0
        ii. beq regX, regY: if regX == regY then go to the label stored in reg0
        iii. sl regX, regY: shift regX left by the value in regY
        iv. sr regX, regY: shift regX right by the value in regY
        v. sro regX, regY: shift regX right by the value in regY and pad regX with the value stored in overflow
        vi. set #: put the number in the instruction into reg0
        vii. sti regX, #: put the number in the instruction into regX
        viii. halt: signifies the end of the program and stops program counter from incrementing
        ix. lw regX: store the memory value of reg0 into regX
        x. sw regX: Sets the memory value of reg0 with regX
        xi. add regX: reg0 = reg0 + regX + overflow
        xii. comp regX; regX = -regX
        xiii. mov regX: Assign regX with value in reg0
        xiv. j: Jump to label stored in reg0
        xv. clr: set overflow to 0

3. ALU Operations

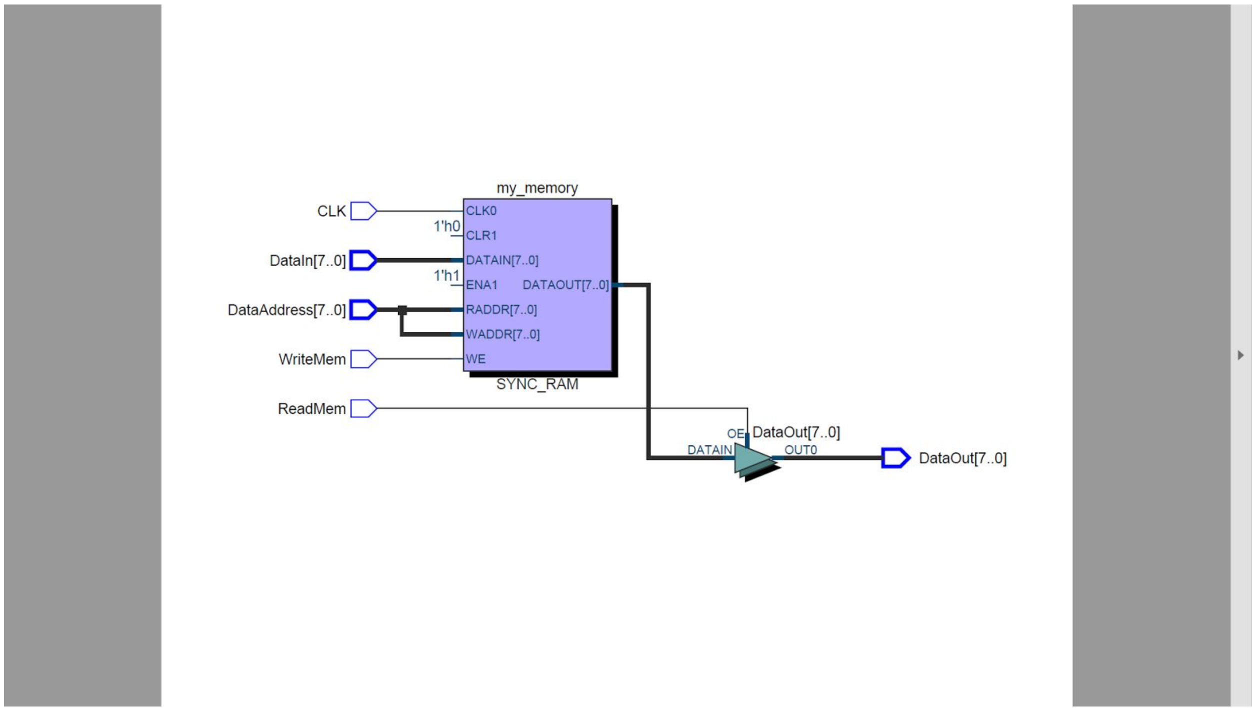
a. ALU: Comp, Assembly Instruction: comp
b. ALU: Add, Assembly Instruction: add
c. ALU: Shift right, Assembly Instruction: sr
d. ALU: Shift right with overflow, Assembly Instruction: sro
e. ALU: Shift left, Assembly Instruction: sl
f. The register file has 2 address inputs, 1 write address input, a write bit and 2 address outputs. All of the ALU instructions require the 2 address outputs.

4. Verilog Models

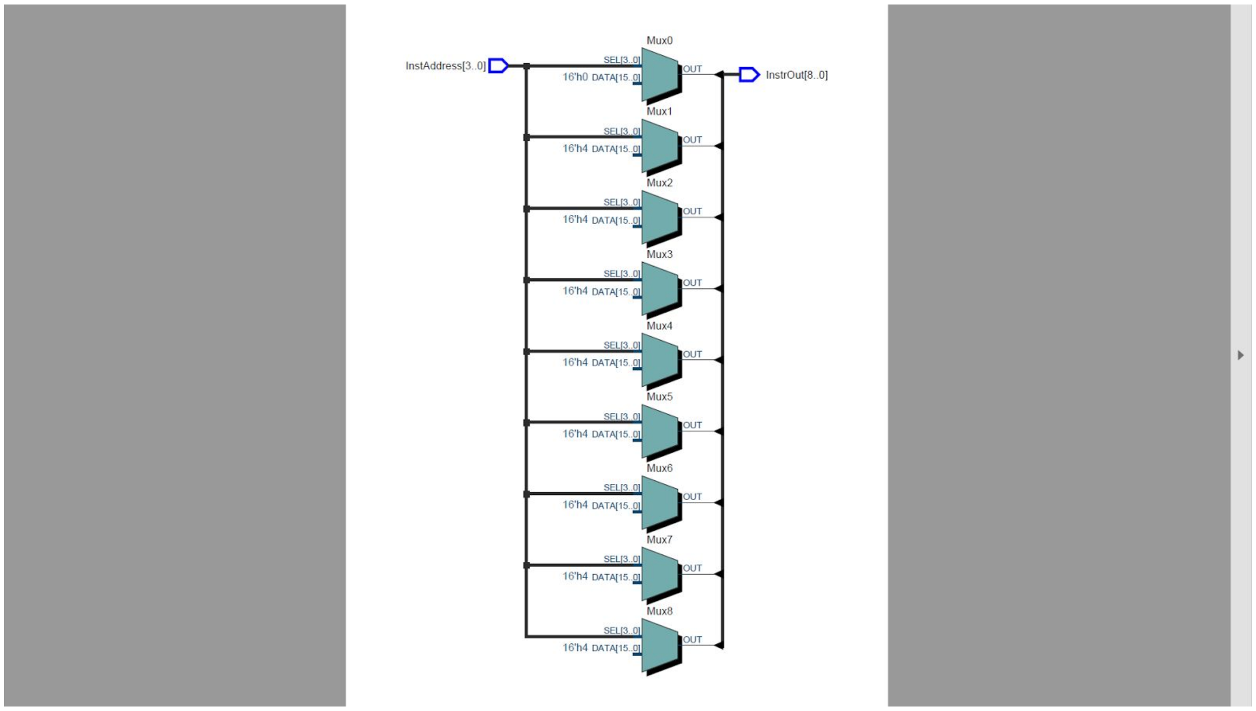
ALU



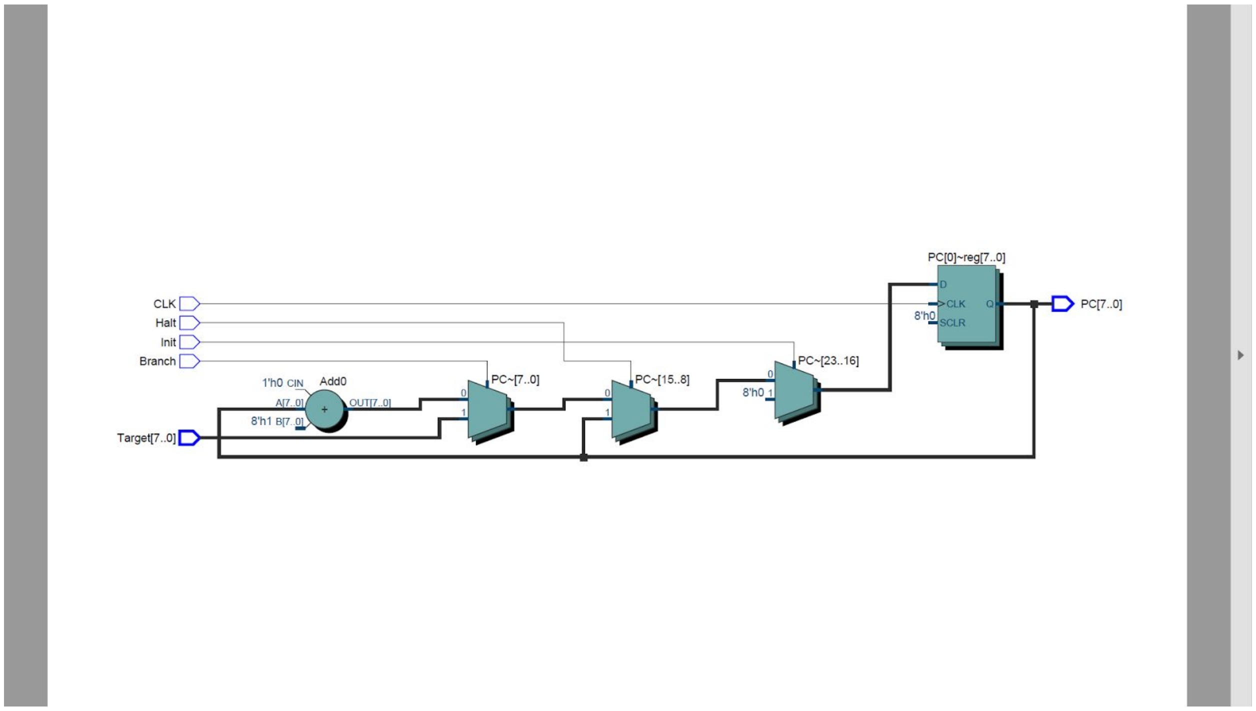
Data_mem



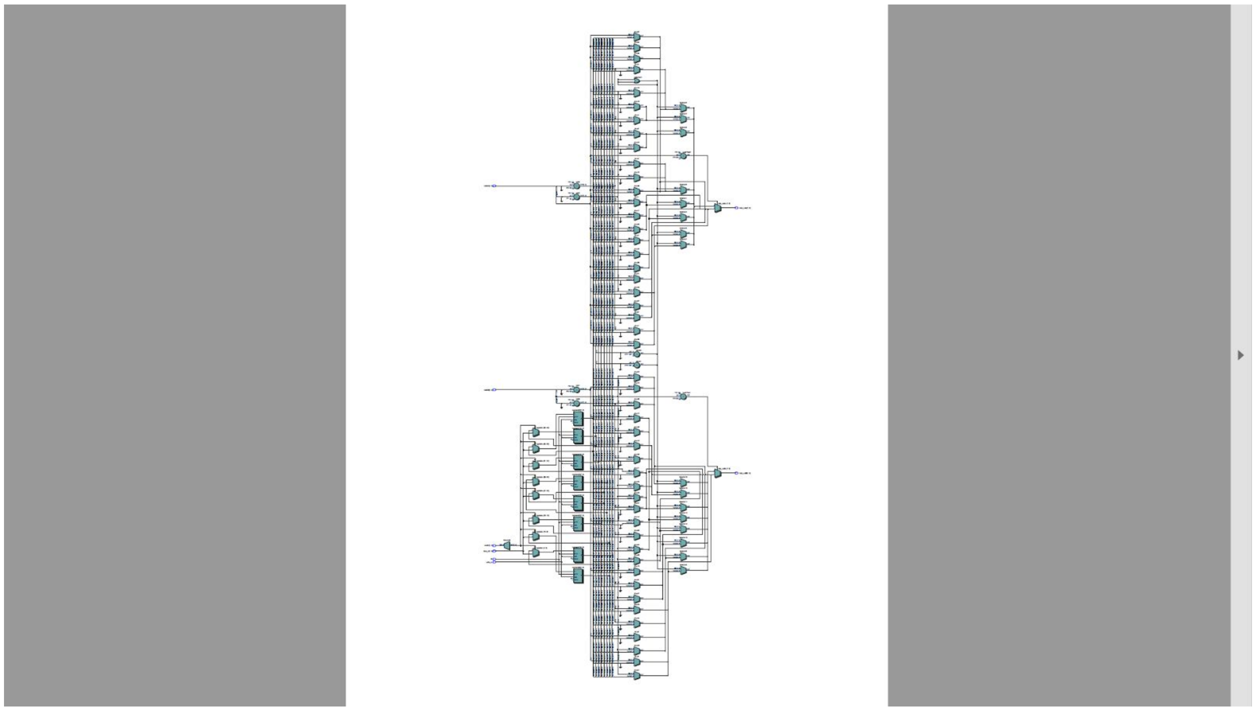
inst_ROM



PC


CLK
Halt
Init
Branch
Target[7..0]
1'h0 CIN
Add0
A[7..0]
8'h1 B[7..0]
OUT[7..0]
PC~[7..0]
PC~[15..8]
PC~[23..16]
8'h0
PC[0]~reg[7..0]
D
CLK
8'h0
SCLR
Q
PC[7..0]

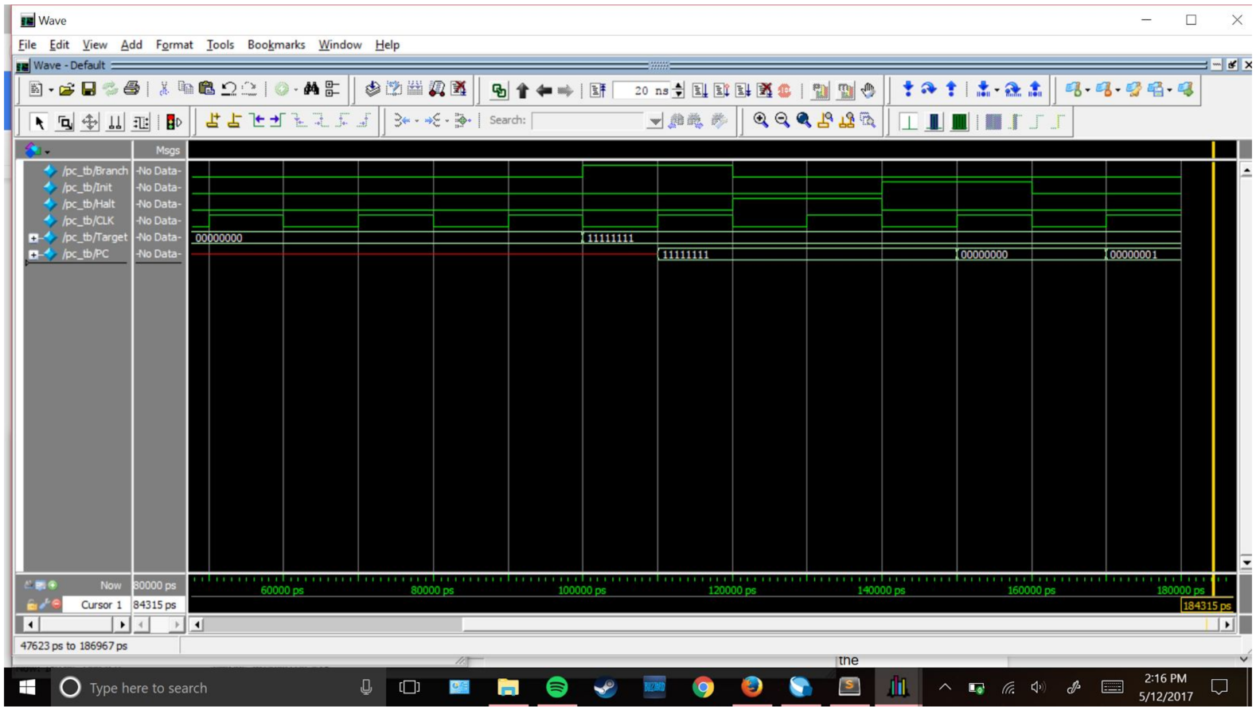
Reg_file



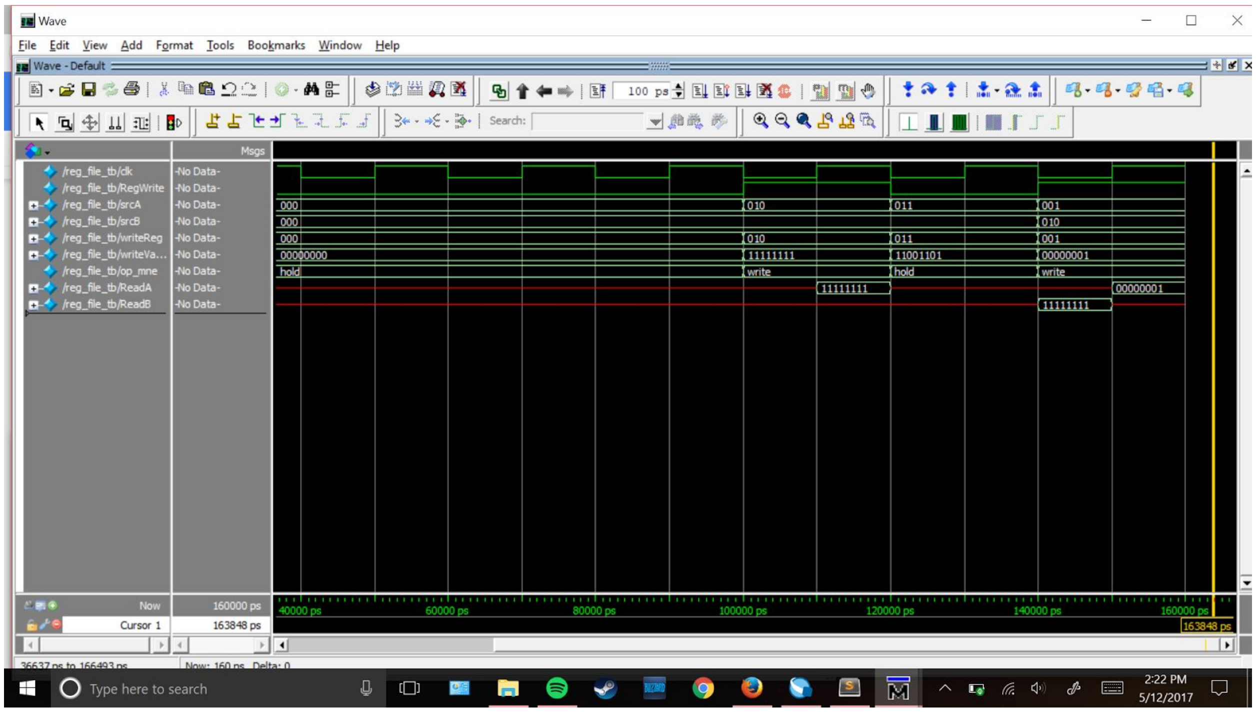
5. Timing Diagrams

a. Our tests are in the testbench files. We begin with 100ns to initialize everything, then every 20ns after that represents a different test case. (Start at 100000 ps)

PC



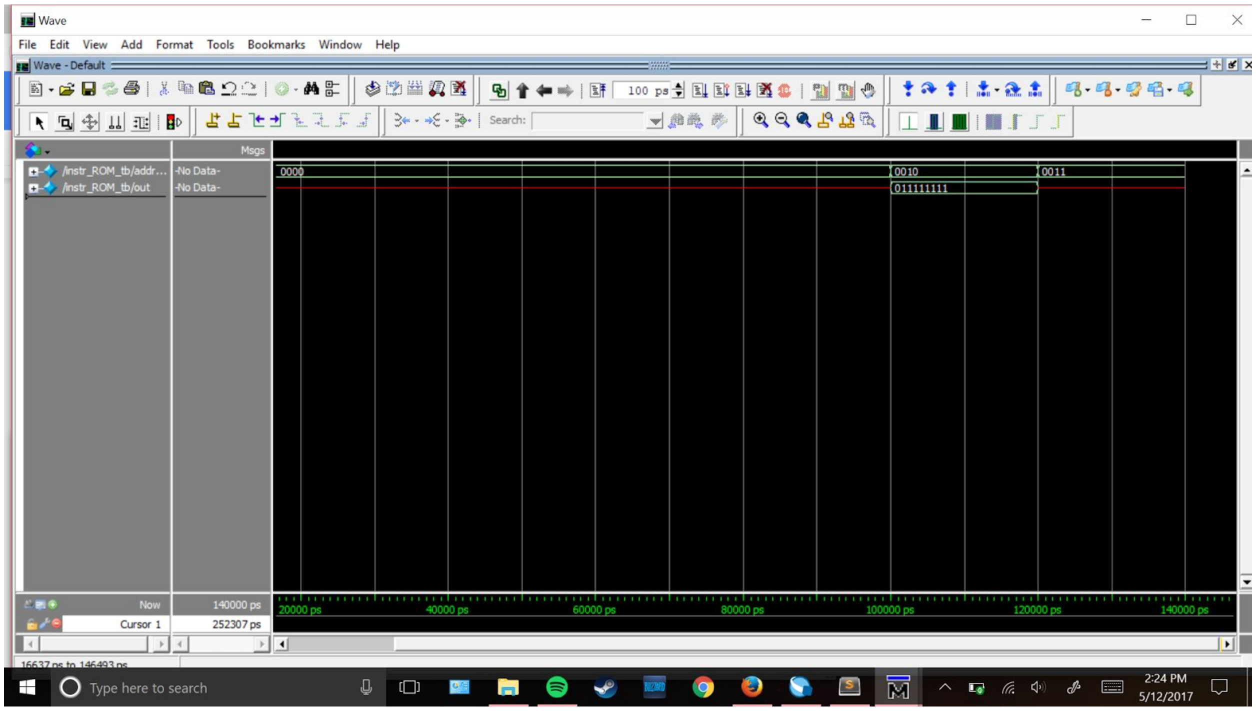
reg_file



```
                            Msgs
/reg_file_tb/clk           -No Data-
/reg_file_tb/RegWrite      -No Data-
/reg_file_tb/srcA          -No Data-   000        010        011        001
/reg_file_tb/srcB          -No Data-   000                              010
/reg_file_tb/writeReg      -No Data-   000        010        011        001
/reg_file_tb/writeVa...    -No Data-   00000000   11111111   11001101   00000001
/reg_file_tb/op_mne        -No Data-   hold       write      hold       write
/reg_file_tb/ReadA         -No Data-              11111111              00000001
/reg_file_tb/ReadB         -No Data-                         11111111

Now        160000 ps
Cursor 1   163848 ps
40000 ps   60000 ps   80000 ps   100000 ps   120000 ps   140000 ps   160000 ps   163848 ps
36637 ps to 166493 ps      Now: 160 ps  Delta: 0
```

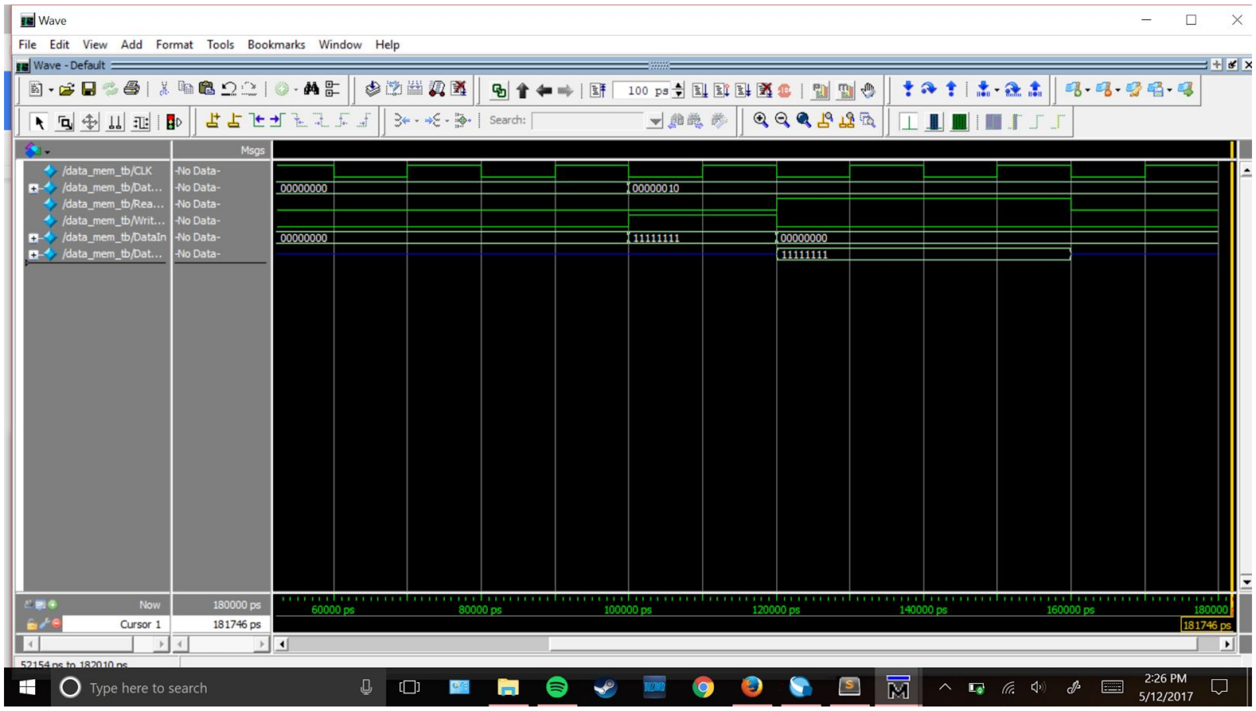
instr_ROM



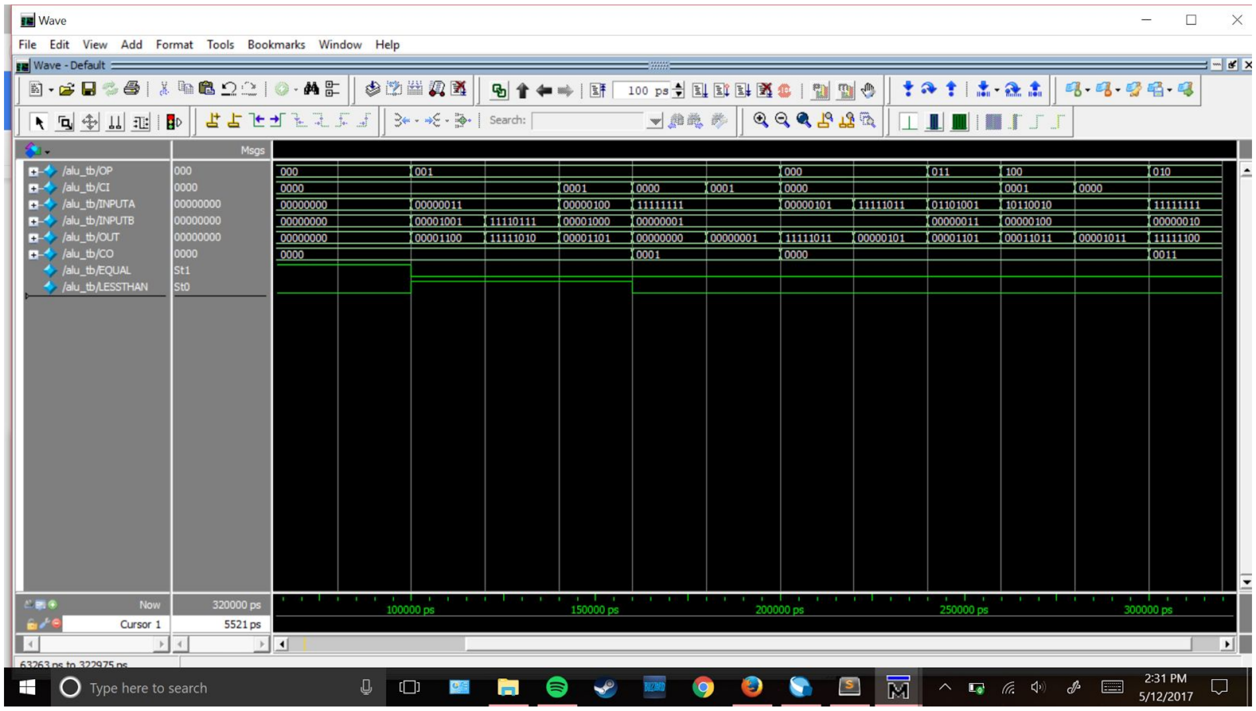
```
                            Msgs
/instr_ROM_tb/addr...      -No Data-   0000       0010        0011
/instr_ROM_tb/out          -No Data-              011111111

Now        140000 ps
Cursor 1   252307 ps
20000 ps   40000 ps   60000 ps   80000 ps   100000 ps   120000 ps   140000 ps
16637 ps to 146493 ps
```

data_mem


Wave
File Edit View Add Format Tools Bookmarks Window Help
Wave - Default
/data_mem_tb/CLK
/data_mem_tb/Dat...
/data_mem_tb/Rea...
/data_mem_tb/Writ...
/data_mem_tb/DataIn
/data_mem_tb/Dat...
-No Data-
00000000
00000010
11111111
00000000
11111111
Now 180000 ps
Cursor 1 181746 ps
60000 ps
80000 ps
100000 ps
120000 ps
140000 ps
160000 ps
180000
52154 ps to 182010 ps
Type here to search
2:26 PM
5/12/2017

alu


Wave
File Edit View Add Format Tools Bookmarks Window Help
Wave - Default
/alu_tb/OP
/alu_tb/CI
/alu_tb/INPUTA
/alu_tb/INPUTB
/alu_tb/OUT
/alu_tb/CO
/alu_tb/EQUAL
/alu_tb/LESSTHAN
000 001 000 011 100 010
0000 0001 0000 0001 0000 0001 0000
00000000 00000011 00000100 11111111 00000101 11111011 01101001 10110010 11111111
00000000 00001001 11110111 00001000 00000001 00000011 00000100 00000010
00000000 00001100 11111010 00001101 00000000 00000001 11111011 00000101 00001101 00011011 00001011 11111100
0000 0001 0000 0011
St1
St0
Now 320000 ps
Cursor 1 5521 ps
100000 ps
150000 ps
200000 ps
250000 ps
300000 ps
63263 ps to 322975 ps
Type here to search
2:31 PM
5/12/2017

6. **Non-arithmetic Instructions**
    a. No we do not use the ALU for non-arithmetic instructions. Our branch and jump addresses are stored in reg0 and when a branch is called the program counter will be replaced with whatever is in reg0.

7. **Make Fetch Unit Design Easier**
    a. The fetch unit consists of the program counter and the instruction memory. To make the program counter design easier, we could take away jumps and branches so the only thing the program counter will do is increment by 1 until a halt is encountered. The instruction memory is just an array that holds all of the instructions so the design is already quite simple in our case.

8. **Make ALU Design Easier**
    a. The ALU design currently makes heavy use of a carry out, so if we eliminate the carry out our ALU will be a lot simpler. But this would severely impact the different types of complex programs our ISA could perform.

9. **Make Register File Design Easier**
    a. To make our register file design easier, we could decrease the amount of registers our ISA supports by changing the functionality of reg1. Currently if reg1 = 0 we have access to reg0-reg7, if reg1 = 1 we have access to reg0, reg1, and reg8-reg13, and if reg1 = 2 we have access to reg0, reg1, and reg14-reg15. So we could change reg1 to be a standard register, but this would decrease the amount of registers from 16 to 8.

10. **Most Complex Instruction**
    a. Add: For our add instruction in the ALU, we add together 2 8-bit registers and a 4-bit Carry In and store it in a 12 bit temp register. We then take the upper 4 MSB of temp and store in Carry Out, and take the lower 8 LSB of temp and store in OUT.

11. **Adding Instructions to ALU**
    a. Currently to perform a subtract you would have to do a 2's comp and then add the two registers, so instead we could just add a subtract instruction that performs the subtraction in 1 cycle instead of 2. Also we could add an increment instruction to handle the looping variables.