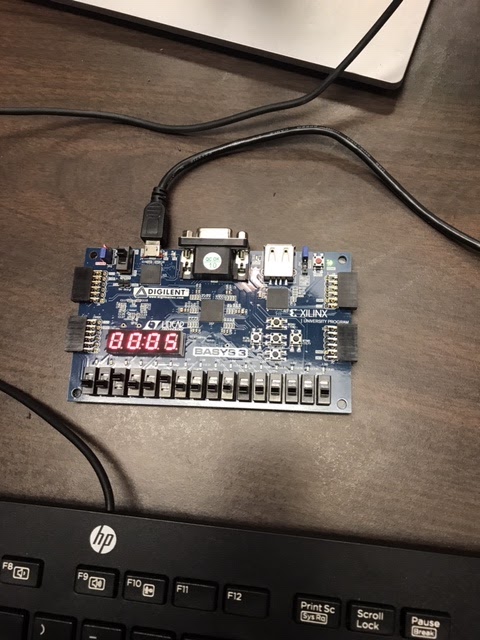
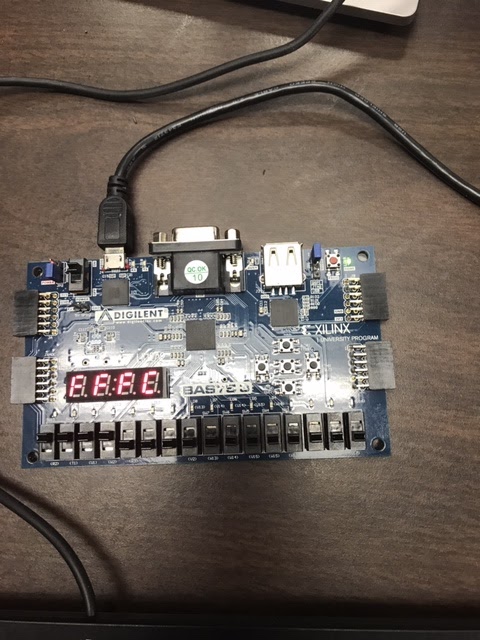
Technical Description

Modifications for Steps 4 through 6:

Step 4: In order to implement step four we first had to add the wire, Instr, in the SingleCycleProcessor an output so we could send that to the top level file. In the top level file we added another ternary statement to create a nested ternary statement where set a,b,c, and d were being assigned. This statement now first checks to see if a switch we declared as RegView, is on or off. If it is on then the 7 segment display on the basys3 board with show the values of the registers. If the switch is off then it will show the value of the instruction that was last executed. The functionality of the TopHalf switch remained unchanged.

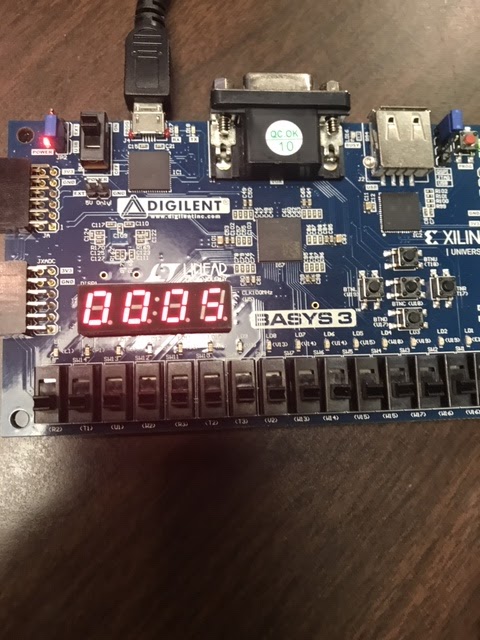
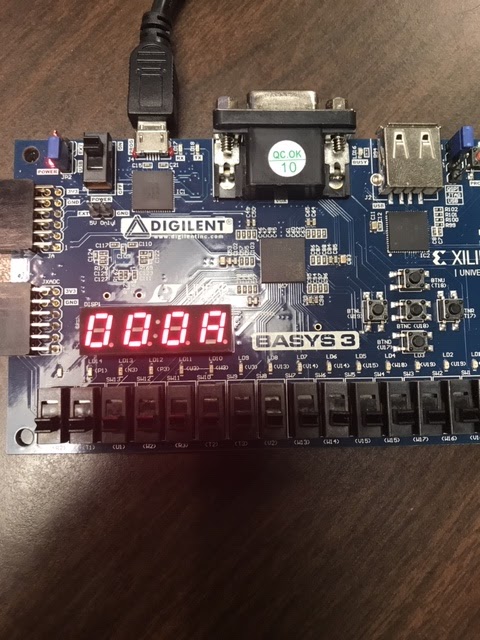


(This shows the register value of R2 at the instruction: 4'b0101: RD = 32'h1AFF\_FFFC)

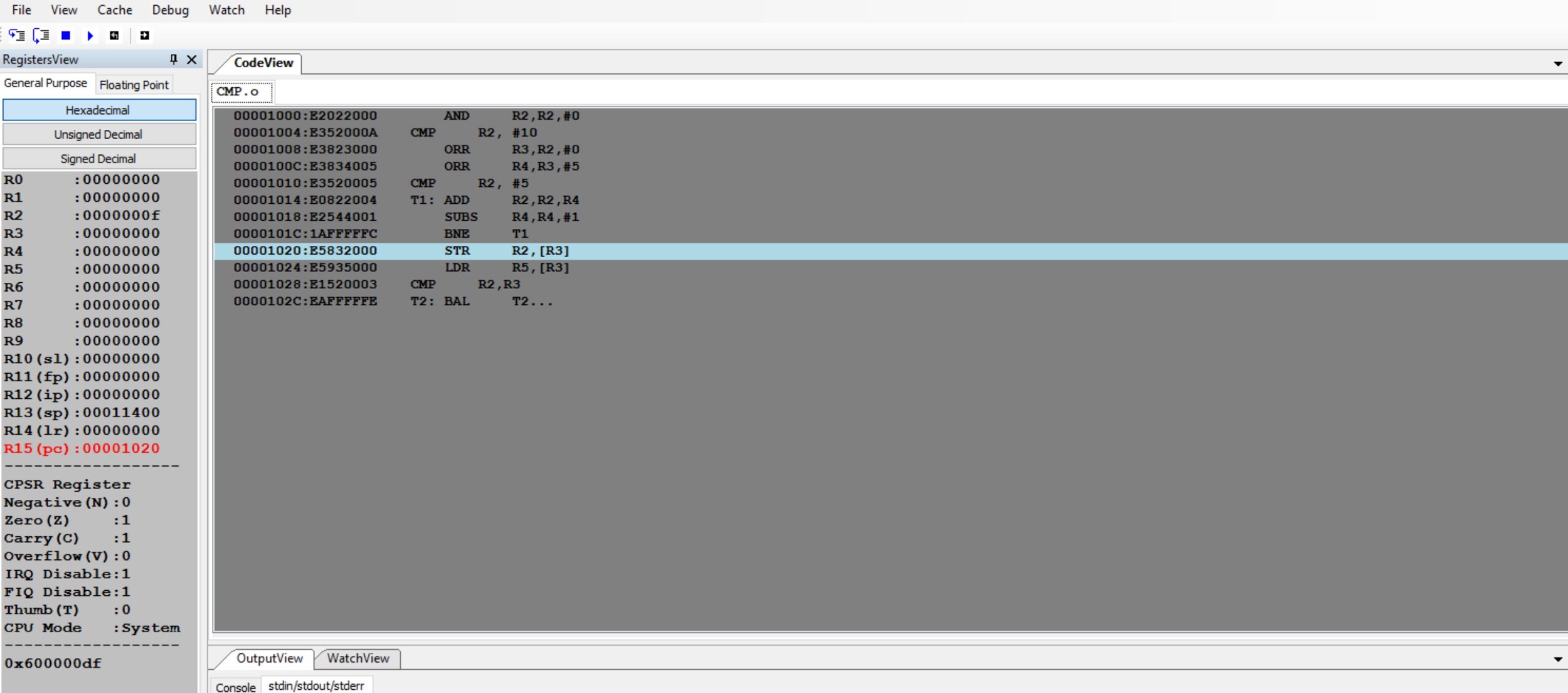


(This shows the last four bits of the instruction value)

Step 5: For step five we had to first add to the controlunit.v file to implement the CMP command. We added a control signal called NoWrite in order to prevent writing Rd during compare. This NoWrite signal was then added to the ALU Decoder table and it produced a 1’b0 at every instruction except CMP, where it produces a 1’b1. We then extended this signal to RegWrite. In order to test this, we added new CMP instructions to the imem.v file. The instructions being run on the basys3 are shown below as well as the results of the program when run through the ARM simulator.

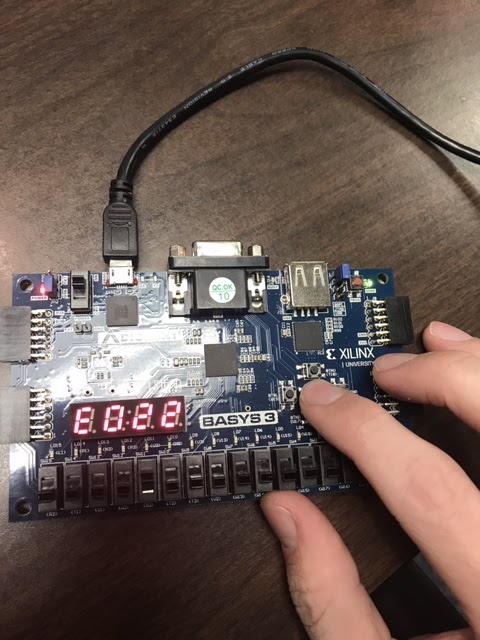


(The images show the last four bits of the instruction value for the first two CMP commands run. The full instruction value for the one on the left is: 32'hE352\_0005 and for the one on the right it is: 32'hE352\_000A)

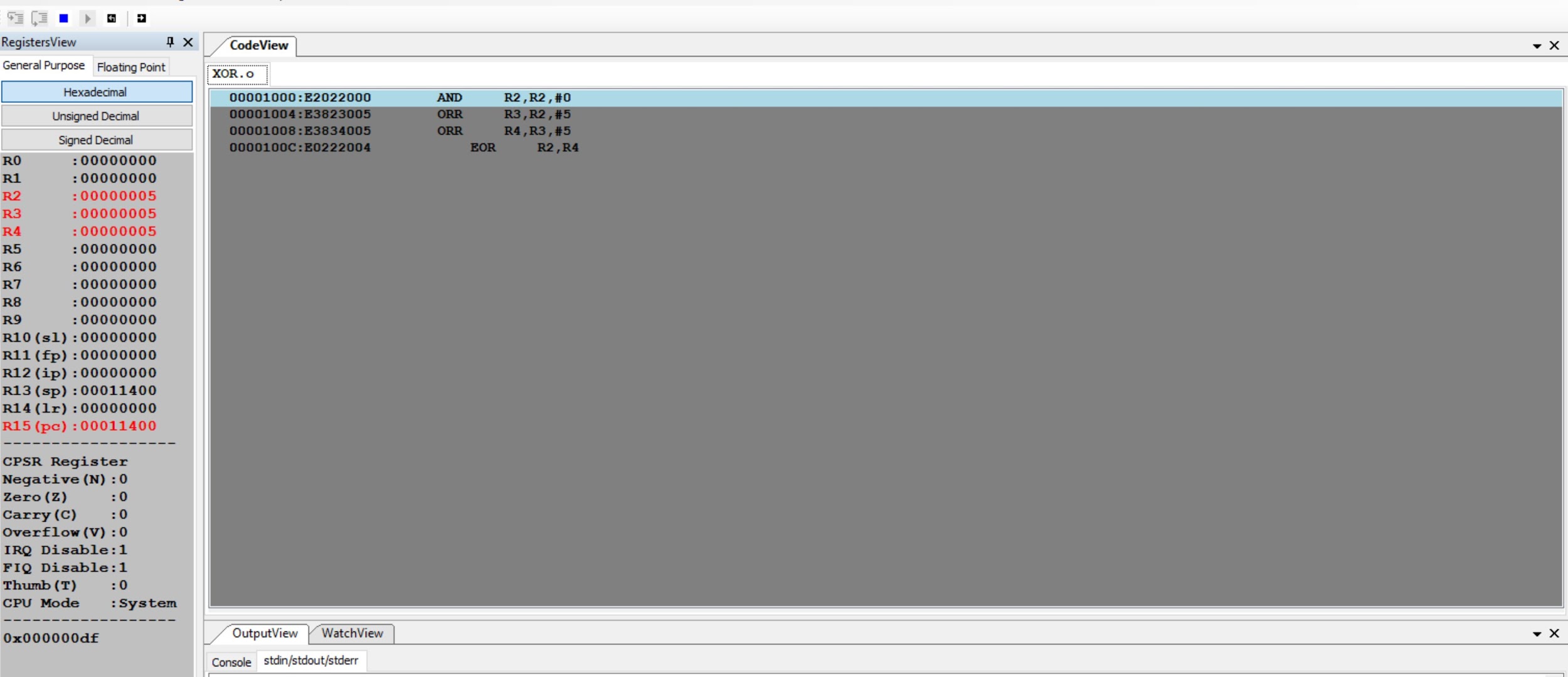


(This shows the results of the CMP program run on the ARM Simulator.)

Step 6: We believe that we need to add an SRC2 to our design that connects to the ALU in order to implement XOR. We did not make any changes to our files but we did write a simple program that implements XOR and were able to see the instruction being run on the basys3. We replaced the imem.v file with this new XOR implementation. We also ran the program through the ARM Simulator.



(This shows the first 4 bits of the EOR instruction value. The full instruction value is: 32'hE022\_2004)



(ARM simulation of our XOR program)

Inventory of Basys3 Functionality:

Switches:

* Sw[12-15]: used to change the value of the register that is currently displayed
* Sw[11]: used to switch between displaying the first four bits and last four bits of the register value or instruction value. If switch is on it will display the first four bits, if it is off it will display the last four.
* Sw[10]: switches between displaying the value of the register or the instruction. If switch is on, then it will display the register value, if switch is off it will display the instruction value.
* Sw[0]: Reset switch. To reset the program turn the switch on, press btn(u18) and then turn the switch off.

Buttons:

* BtnC(U18): pressing the button will run the next instruction of the program.

Verilog Module Modifications:

Imem.v: We created our own imem files to test the CMP and XOR instructions.

Controlunit.v: In order to implement the CMP instruction we added a one bit output called NoWrite to prevent the writing of Rd when a CMP command is used. We also added NoWrite to the ALU Decoder table after ALUControl and FlagW. We added another instruction at the bottom of the table for CMP. NoWrite was 1’b0 for all instructions in the table except for CMP. Finally we added NoWrite to the assigned value of RegWrite at the bottom of the file.

SingleCycleProcessor.v: We added Instr as an output so that the top level file could use it, and we also added RegWrite as a wire.

TopLevelSCUWARM.v: Added Instr as a wire, RegView as an input, and updated the SingleCycleProcessor line to include the new output INSTR. Also added another ternary expression when assigning seta, setb, setc, and setd. This ternary statement first looks to see if RegView is high, and if so displays the reg value with TopHalf functionality remaining unchanged. If RegView is low then it displays the instruction value with TopHalf functionality remaining unchanged.

Schematic:

