COMP4300

Lab Assignment 5

Your job: add VHDL code to each state of the controller for the SimpleRisc cpu so that **the entire program** preloaded into the memory (addresses 0 to 6), two loads, an add, and a store)   will execute correctly, and submit screen shots showing the correct execution. Note that you do not need to implement jumps (which are extra credit, worth 1 full lab if you get them working). You will need to use your solutions to Labs 2 and 3 and 4 to complete this assignment, as well as all the files you used in lab 4:

simplerisc_datapath.pdf -- this is a figure showing the data path

SimpleRiscProgrammingCard.pdf -- the instructions and their formats

SimpleRisc_semantics.pdf -- state-by-state description of what the cpu does

In addition to the dlx_types package and bit vector arithmetic files you have already used, the VHDL files you will need are:

SimpleRisc.vhd -- this is the interconnection file. It connects up all the entities that make up the datapath and controller into a single entity. YOU SHOULDN'T NEED TO MAKE ANY CHANGES TO THIS FILE

simpleRisc_datapath_lab4.vhd -- has all the entity declarations for the code you wrote in labs 2 and 3. You will need to copy their architectures into this file, and make sure the architectures are named consistently with what is in SimpleRisc.vhd. The architecture for the memory is given, so you don't have to write that. There is a program preloaded that does two loads, an add, a store, and then some jumps. For full credit just implement the loads, the add, and the store. The jumps are extra credit (a fair amount of it, not sure exactly how much yet).

simpleRisc_control_lab4.vhd -- this is the controller, which I went over in the videos. Your job is to fill in the code for states 3 through 13, or 3 through 18 for the extra credit, as described in the videos.

Deliverables:

As usual, turn in your VHDL code (zip up the whole project, to make it easy for the TA to grade), and screenshots of the simulation, so that the values of the important dlx_word variables can be seen (watch the Simple Risc Videos for more info on what these should look like). Just simulate the one program already loaded in memory. You don't really need a .do file, since the only input to the SimpleRisc controller is the global clock (signal name simple_clock). Give it a 200nS, 50% period with an initially falling edge.

Note:

It is possible, despite weeks of testing, that there remain errors in the SimpleRisc specification. If any are found, I will replace the existing vhdl files with new ones on Canvas, change the names to reflect version numbers, and make an announcement about it.