This project really helped me learn about the design of RISC processors. I elected to design a multicycle architecture and aimed for minimum area overhead. My instruction set architecture (ISA) contains 16 instructions, of which 5 are arithmetic instructions (ADD, SUB, AND, OR, NOR). To me, the most interesting part of the project is seeing how the design of the instruction set architecture directly impacts the complexity of the processor’s hardware implementation. With a very simple processor like the one I designed, each additional instruction in the ISA required some additional hardware in the datapath and an additional state in the control logic. For example, I included a jump-and-link instruction, and this required an additional input to the multiplexor which selects the destination register address. This additional input is tied to “1111” so that the PC address of the next instruction can be stored in register $15 when this instruction is executed. It was important to include this instruction in my ISA because there is no other way to store the contents of PC in the register file.

Overall, I am satisfied with the design of my multicycle processor. It is small and has sufficient instructions to make assembly language programming straightforward. If I were to repeat the project, I would try to implement a pipelined processor. It does not seem much more difficult to design a pipelined processor than the single-cycle processor. However, the additional pipeline registers, forwarding logic, and hazard detection logic will add to the area impact of the design.

For future students working on a similar project, I would advise them to note early in the design process that the RAM in the Altera MegaFunction library is a synchronous read/write RAM. The block diagrams of the MIPS architecture from the lecture slides assumes that the RAM read is asynchronous. This led to a problem in my multicycle architecture, and I had to use a dual-port RAM model to fix it. Also, leave plenty of time for simulation with ModelSim. Only a small portion of your time will be spent designing the architecture and writing HDL. Most ( > 70%) will be spent writing testbenches or force files and running simulations. Take some time to write a good testbench for the datapath. This will help you to not only to verify and debug the datapath, but also to understand how the control logic needs to behave. And, as always, get started on the project early, and try to get each section turned in on time. There will be some unexpected bugs (even if you’ve had some experience modeling hardware). Try to catch as many of the bugs as possible very early in the design process, because by the time you incorporate the datapath, control logic, and memory, it will be much harder to locate the source of a bug in the design. There’s nothing worse than downloading your design to the FPGA only to find that it doesn’t produce correct results because of a functional bug that was missed early in the design process.