Joseph Liba

Prof. Kinsy

EC413

15 November 2017

Project Report

**Implementation Strategy:**

**Register File:** Creating the actual register file involved creating a 2D register consisting of 32x32 size. This meant 32 bit data for each of the 32 addresses. When accessing a random index location in the register file, Verilog does this very elegantly. If reg\_select is a 5 bit input, memory[reg\_select] would give me the data stored at the 5 bit input index. For example, 00100 would access the register data at position 4. Depending on whether write is 0 or 1, I update or read the register accordingly at each clock cycle.

**Memory:** This implementation is the same as the implementation for the register file. The only difference is that because you have a 8 bit address, we have 256 addresses rather than 32 lines. Another key difference is that we have a distinct memRead and memWrite compared to a single write input like the register file. This is because if the memory is not being used at all, you don’t want to either read or write.

**ALU:** Controlling the ALU involved a switch statement depending on the ALU control. Depending on the different values for the ALU control, it would perform different arithmetic operations on the two input values.

**Control Unit:** Some of the outputs for the control unit were easy to access. The ALU\_control, read select a, read select b, and write select all can be wired from the instruction directly. For determining values for mem\_read, mem\_write, ALUsrc, and mem\_to\_reg, I set these as registers and updated them at each clock cycle using a switch statement for the OP code. However, in future optimization, I could easily set this using combinational logic and setting them with wires rather than using registers. This would optimize speed, and if this is a bottleneck in the future, I would optimize this.

**CPU:** Connecting the modules and wires was easy as looking at the diagram. I needed to use combination logic, however, to set up the multiplexers that take data either from the immediate or from the 2nd register, or whether to write from the ALUout or the memory. To test this in the test bench, I needed to add some output wires somewhere, so I put them in the register outs.

**What I learned:** Lots of design questions came down to how Verilog would convert high level language to low level design. For example, using registers and switch statements might be more compact code-wise, but design wise might make for a slower module. However, in a pipelined design, if this is not the slowest cycle, it might not be so important to optimize.

Notes: I uploaded my own testbenches since I changed names and variable orders.