Alex Kahng, George Zhang

CS 141

Simulation Description (Lab 2: Part 1)

**Implementation**

For each of the bitwise operators AND, OR, and XOR, we implemented a module by using the analogous bitwise operator in Verilog. For NOR, we implemented a module by negating the OR output. For ADD, we created a one-bit full adder in a separate module and chained these modules together to create a ripple-carry adder. For the multiplexer (mux), we nested ternary operations that called the appropriate operation (module) depending on the given op\_code.

**Testing**

For each of the bitwise operators, we set X, Y, to all permutations of 0x00000000 and 0xFFFFFFFF. This allowed us to test the operation of each bit and make sure our bitwise operators worked correctly. We also tested over random input to further validate beyond the corner cases. To test the EQUAL and ZERO flags, we checked if X = Y or if Z = 0 and made sure the appropriate flags were set. To test ADD, we wrote a one-bit full adder test and an adder test, both of which we did not include in test\_alu.v. In test\_alu.v, we created a test to make sure that the full adders were correctly hooked up. Specifically, we repeatedly bit-shifted X and Y, added them, and verified that the next significant bit changed correctly due to the carry bit. We tested our mux by varying the op\_codes.

Simulation Description (Lab 2: Part 2)

**Implementation**

For SUB, we simply negated the second input Y and used our ADD module while passing in 1 for Cin. Thus allowed us to easily convert Y to a negative number, and since X – Y = X + (-Y), this is valid. For SLT, we realized that if X < Y, then X – Y < 0. Thus we used our SUB module, and tried to see if the subtraction produced a negative result. If there was no overflow, then we care about the most significant bit, but if there is overflow, then the most significant bit is the opposite of what we want. Thus we took overflow XOR MSB to get SLT. For each of the shifters, we ensured that Y is between 0 and 32 (if not, we return 0) and then use lots of muxes to assign the bits appropriately (according to our diagram). A note on the shifters is that while the diagram shows the usage of SLT modules, we decided in the implementation that it would be easier to simply OR all the bits (31:5), and signal the mux based on that output.

**Testing**

Since our SUB just reused the ADD module, we wrote tests for the edge cases, which we decided were the overflow cases. Thus we had a test where X was very small and Y was very large, which would produce a positive value with an overflow, and vice versa. We also used a random test, which chose two random numbers and performed SUB with them. Our implementation of SLT used the SUB module and took the XOR, so we came up with 4 cases for the XOR: the appropriate combinations of X < Y / X > Y and overflow / no overflow. For the shifters, we tested the cases where Y >= 32 or Y < 0 (in which case the output should be 0), and we tested the appropriate bit shift using the bit shift operator to compare with our result.