4. For our implementation of alu-control.circ, we focused on each group of instructions individually and then got them to work together. Getting them to work together was definitely the tricky part. For the loader instruction, we put those into their own multiplexer with the ‘sh’ instruction as the select, since they all have the same ALU Control number. Then, we took that into the main big multiplexer, where all of the other instructions were plugged into their corresponding ALU Control number using constants, using the middle four of the Funct binary number to choose which one was needed. Then, one last multiplexer was added for the branch instruction. Using the Opcode as the select, if that number was 00, then we would use the main multiplexer number; if op was any other, use binary 7.

5. debugging story

a. One time we observed an unexpected behavior in our circuit was when we were working in implementing the flags for the ALU. When using the adder or subtracter, the oVerflow and Carry flags were not acting as expected. At one time, we even got the Carry flag to be e! Of course, the expected behavior was for it to be either 0 or 1, as the tests showed us.

b. To investigate this problem, we first went into the test file and went to the test number that it kept stopping on when we were running the tests. Then, we went into the ALU from there to really see what was going on. For this specific test, it looked like our oVerflow flag was set correctly, but the Carry flag was not. We noticed right away that the ALUControl number was for addition, so we investigated that. At this point, our solution for the Carry flag was just to use the c out for both the subtracter and adder. We noticed that even though the adder wasn’t giving a Carry flag, the subtracter was, and since we had both the wires connected to one another, they were able to mess each other up.

c. To fix this, we made a new multiplexer in the main view of the ALU, using the ALUControl as the select. We attached the adder and subtracter to the multiplexer in their respective spots and made every other slot a constant 0. Now, when one outputted the Carry flag, but the other didn’t, they wouldn’t ‘contaminate’ each other. We reran the test (alu-add) and saw that everything passed!