Project 4 Implementor's Notes

Computer Engineering
University of Kentucky

Tyler Cultice tacu229@uky.edu
Michael Kamb mgka224@uky.edu
Daniel Palmer dmpa227@uky.edu

I. Picking the Right Assignment 3

In order to begin working on this assignment, we first had to decide which of our three previous pipelined processors we would be working with for this assignment. Initially, we elected to go with the processor from Michael's last group because the other two processors did not handle branching and jumping instruction correctly. However, we quickly realized that implementing new ALU functionality on that processor was going to be troublesome because the ALU was a separate module apart from the processor module. We decided to bite the bullet and take the extra time to fix the jumping issues of one of the other processors so that we would potentially have an easier time implementing the new ALU functions and the table lookups without having to create many more control signals. If we would have stuck with our original choice for the processor, control signals would have become a huge nightmare. Ultimately, we went with Tyler's pipelined processor because it was fairly neat, and Tyler offered to take the lead on writing the new Verilog code, so it was helpful that he was very familiar with it from the get-go.

II. Aik Spec Modification

Because of the change form 16-bit posits to 16-bit floats, and the addition of several previously left out instructions, the AIK specification that we had all previously used (Dr. Dietz's solution to assignment 1) needed to be modified. The was a very simple process of simply renaming some encodings from posit, p, to float, f. Other instructions were entirely new, such as dup, so we added them where appropriate based on their number of operands and similarities with other instructions. These changes can be observed in the gr8bond.aik file.

III. Floating Point Operations

Floating point operations were easy enough to implement because of the modules given to us in the assignment description. We placed the modules that we needed at the top of our gr8bond.v file above the processor module. From there we were able to reference these modules from inside of our Read Stage and then set the appropriate results inside of the neg-edge triggered ALU inside of that same stage. The Verilog can get a little confusing at times because some of the floating point modules have the same name as the corresponding instruction. Also, we did encounter some difficulty in using the lookup table for the frecip module because we were running out of vmem slots on the CGI interface. To overcome this issue, and to get some extra error message information, we switched to using the actual Icarus Verilog compiler. Switching to this compiler brought its own set of challenges initially, but we were able to figure out how to use it eventually.

IV. Posit Operations

The posit operations were also not too terribly hard to implement. As with the floating point instructions, we were able to simply add the new functionality into the existing instructions in the ALU. These instructions had, for the most part, existed in our processor in the previous assignments, but they were identical to the integer operations. Changing to their real functionality was as simple as figuring out how to perform table lookups on the vmem files provided in the assignment description. These table lookups were done in the Read Stage of our processor. The index of the lookups was assigned before the ALU and the results from the lookups were assigned within the ALU, but before the case statement. As a result, within the case statement of the ALU, whenever an instruction requires one of the table lookups, the appropriate bits of the result are assigned to the corresponding table lookup. For example, separate table lookup results exist for the low and high bits of each of the two tables. Lookup24low and lookup24high are always indexed with a different number, unless the to 8-bit posits are the same, but when an addpp instruction comes along, the same bits from that lookup are used for both the high and low result. This is because add, multiply and float to posit all have data at the same index in the table.

V. Testing

As with the previous assignments, we were able to mostly recycle previous testing suites and use them again here. Obviously, several instructions have been changed slightly from 16-bit posit to 16-bit float and some are entirely new, but after adding tests specifically for those instructions and removing tests for instructions that no longer exist, we were satisfied with our processor and believe it has been tested sufficiently. Exhaustive testing of the tables could have been performed, but Dr. Dietz said this was not necessary.