Phloating Point PinKY

EE 480

Dr. Dietz

Team 47

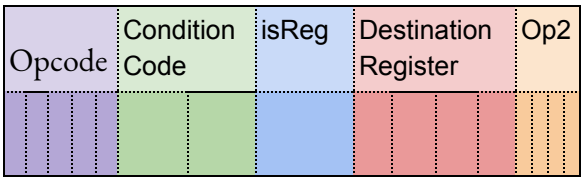
Atanas Golev, Tyler Lawson, Matthew Tarter

Overview

In this project we used Verilog to simulate a floating point pipelined processor based on the PinKY instruction set. This implementation is able to handle dependencies between operations, as well as managing register contents between stages.

Implementation Details

In this particular design, we rely on this bit-pattern for our 16-bit instruction codes:



Stage 0

In our pipelined processor design, we start in Stage 0 in which we determine the value of our program counter (PC), set our 'Z' flag, perform writes to the regfiles, and handle jumps/branching. The PC is managed here based on the 15th register value.

Stage 1

In Stage 1, we fetch the next instruction from memory, set a PRE value if necessary, and check the 'Z' flag to see if the instruction should execute. In this stage we also check if the instruction is a JUMP which would freeze subsequent instructions. Stage 1 is also the stage that sets the ***frz*** flag, which is set when a dependency is noticed.

Stage 2

Stage 2 takes care of loading registers (LDR) and storing (STR) operations, and handles sign extension from PRE. This stage only executes when there are no dependencies present (***frz*** = 0).

Stage 2.5, or, Stage F

Stage 2.5 handles preparation for floating point operations which wouldn't be reasonable to perform all in one phase, such as splitting up shifting registers and determining the number of leading zeroes. Non-floating point operations simply pass this phase without any effect.

Stage 3

In this stage we decide if we will be doing a memory or ALU operation, and then perform said operation. We also output from this stage the DestVal which is the result of the operation taken in this stage.

AIK Specification

We chose to use the AIK specification from one of our group members' previous assignments. It has a few interesting features, including that PRE S and SYS S are detected as inadmissible instructions. They are automatically changed to the machine code of PRE AL and SYS AL. We have attached the specification as pinky.aik. In general, the format for our instructions is:

OP CC OP1, OP2

In the case of LDR and STR instructions, OP2 is enclosed in brackets. In the case of a constant, OP2 is preceded by a #.

Specific Insights/Details

Dr. Dietz' sneaky float tricks were heavily implemented, including the reciprocal lookup table and the binary method for counting leading zeroes. However, we did not use a barrel shifter, as we did not implement the ADDF or SUBF functions.

We added one extra phase in order to split up floating point operations which may slow the clock down too much in a single phase. We tested all the previous tests we had from the previous project to make sure that dependency handling and all other features weren't broken.

In order to deal with ITOF for negative integers, we stored the positive and negative values for the mantissa of op2 through Stage 2.5 and Stage 3 in order to be able to simply use the negation instead of finding it on the fly, which would make this a more reasonable floating point design.

We also decided to use a long 16-bit mantissa for MULF in order to enable the product of the mantissas to be more accurate than it would be if it were stored in an 8-bit field.

Additionally, we did fixed a known bug from the previous project which made the program unable to handle negative integers smaller than -8.

Known Bugs

The ADDF and SUBF functions have not been implemented. Additionally, the RECF goes through only one iteration instead of multiple iterations; therefore, it is not as precise as the actual float approximation.

Additionally, there may be some issues with FTOI. It specifically fails to produce the correct result when converting -50, getting the 14th bit incorrect. However, it does work for most other values, and we weren't able to figure out what about that specific case caused it to fail.

However, there are no other known bugs, based on the testing described.

Test Cases

We have attached a *tests.pdf* document detailing the tests we ran and the expected results.