Assignment 4: Floating Implementor's Notes

Abu-Taha Abdulfattah
tat_489@hotmail.com
Casey O'kane
casey.okane@uky.edu
LiangLiang Zheng
lzh229@g.uky.edu

***Abstract*—The goal of this assignment involved implementing the floating point based instructions of the IDIOT instruction set. The newly adapted ALU was then interfaced with the provided pipeline solution and a series of tests were created to ensure module and interface correctness.**

I. General Approach

Similar to the approach suggested by Dr. Dietz in the assignment prompt, the ALU received most of the group's early attention. First, the floating point instructions were implemented using the ALU module of the provided **pipe.v** file. Then after integrating this ALU with floating point this ALU was then placed inside of the **pipe.v** file and a few changes were made to account for the floating point instructions.

II. Implementation

This section describes how each instruction was implemented.

A. Pipeline Solution

B. Floating Point Instructions

1) Integer to Float: For integer to float, the sign bit is first determined and stored in the signBit register. After finding the sign, the mantissa is constructed by creating a buffer of 24 bits, which contains the absolute value of the input with 8 bits of zero.The mantissa is constructed in the slice of the buffer contained from $15-numZero+7$ to $15-numZero$ to account for offset. After the mantissa, the exponent is found by adding the hexadecimal value *0x7f* to 15, which is then subtracted by the number of zeros. After all of these individual portions are collected, they are concatenated together to form the resulting floating point value.

2) Float to Integer:

3) Float Addition:

4) Float Multiply:

5) Float Inverse: After a numerous algorithms were provided by Dr. Dietz, the implementation for the inverse float operation was finally settled after all proposed solutions were presented and it was decided that the ideal tactic would involve the use of a lookup table **reqFiles/recip.vmem**.After finding the sign bit, the reciprocal value of the mantissa is found by indexing the lookup table at the mantissa address. After the mantissa is collected the exponent is found by finding the difference between 254 and the value of input exponent (if the mantissa is equal to zero), if the mantissa bits do not result in the value zero however, the result must be shifted and the difference between 253 is used to record the exponent value. Finally, after all of the components have been constructed, the sign, exponent and mantissa are concatenated together to form the inverse result.

III. Testing

A. Verilog Modules

B. Testing Results

IV. Issues

A. Notable Workarounds

1) Negative Number Leading Zeros Block: One issue that the group experienced involved finding the leading zeros for numbers that were negative. To combat this, an always block was utilized before the leading zeros module was instantiated so that if the number was negative, that the its positive sign binary representation would have been used to find the number of leading zeros. This was placed in an always block as checking for the sign bit later (inside the instruction implementation) since the **lead0s** module could not be instantiated inside the always block where the instructions are handled.

B. Features Not implemented

C. Known Errors

As it currently stands there are no known issues

V. Theory

"Are these operations, plus the integer operations, sufficient to cover most floating-point needs or is there stuff missing? For example, there are no floating-point compare instructions here... is that really reasonable?"

"Are these operations necessary? How much harder would it have been to implement these floating-point operations using just the integer instructions? Is there a real performance payoff?"

"Would it have been more or less useful if you implemented the IEEE standard binary16 format (which has a 4-bit exponent) instead of implementing the "top half" of binary32?"