EE480 Assignment 3: A SIK Pipeline Implementor's Notes

Austin Langton, Kyle Nelson, Thomas Wheeler
University of Kentucky, Lexington, KY USA
Department of Electrical and Computer Engineering
Austin.Langton@uky.edu, kyle.nelson@uky.edu, thomaswheeler@uky.edu

April 10, 2017

Abstract

This project served as the initial design and implementation of a pipe-lined processor module written in Verilog for the SIK instruction set developed by Dr. Dietz.

1 General Approach

This lab involved designing and developing a pipe-lined processor module for the SIK instruction set architecture written using synthesizable Verilog code. The specifications for the SIK instruction set can be found on the course webpage.

The first task in completing this assignment was to start with a single cycle design created in the last assignment. We chose to use Kyle's implementation and also his AIK encoding.

The next task involved modifying Kyle's implementation to comply with the project guidelines and implement a pipelined design. We did this by splitting the parts of the single cycle processor design into multiple stages. This allows us to work on more than one thing at once. We chose to split the single cycle design into 4 stages. The first stage, stage 0, contains the program counters. Later stages are able to write to this. Stage 0 also has the ability to insert Nops into the pipeline for halted threads. Stage 1 chooses source and destination registers if there are any needed depending on what instruction is given. This stage also halts the thread if a Sys instruction is called. Stage 2 performs register operations as well as writing to destination registers. Stage 3 performs ALU operations and writes to main memory.

Our last major task for this project was to test our implementation. We went through the several test cases that are detail below and got mixed results that will be discussed further in the Issues section.

2 Issues

We did our testing in a separate document so that when we added displays to each operation, we were not adding non-synthasizable code to our processor implementation.

As mentioned above, we added displays to every function in every stage so that we could get a visual of where potential hangups were occurring. All of our tests were conducted using simple assembly instructions generated by AIK.

Our first issues was the testbench stopping as soon as soon as a thread halted. In order to fix this we altered the clock 5 or 6 times after the loop in the testbench.

We then had an issue with the Push instruction always using Pre in stage two but this was easy to fix.

We were stuck for a long time when stage one was loading out of the instruction register which didnt have anything loaded into it.

At the submission time, our processor was not fully working. We were having issues where it would sometimes get stuck in an infinite loop if the program input is of uneven length. Also the threads appear to be writing over each other at random.

Also included in the submission tarball is a text document detailing each of our test cases and the full results from each test. It is called TestResults3.txt