**CS552 Final Project**

Team Name: **xX\_MaYMaY\_Xx**

Team Members: **William Jen, Stephen Eick**

Spring 2016

**Project Overview**

This project is focused around implementing the WISC-SP13 ISA through the development of a processor. Our processor possesses a five-stage pipeline with a two-way set associative cache.

*The Pipeline*

The five stages of our pipeline are as follows: fetch, decode, execute, memory, and writeback. Each stage is broken into separate modules. Placed between each stage is a buffering module which ensures the inter-stage signals as well as the pipeline control signals are sent through properly

The fetch stage contains the instruction memory module, a CLA to increment the PC, and registers to buffer the combinational logic used to determine PC and EPC states. The computed PC for a branch instruction is sent back to this stage. For the stage to know when to use the branch PC, a signal telling the fetch stage that branching should occur is sent back.

The decode stage contains the combinational logic to translate the provided opcode into the appropriate data processing control signals. All but one of the control signals are a single statement of hand-optimized combinational logic, with the outlier being a case statement to assign the three-bit ALU control signal. The decode stage also contains the register file for design organizational purposes.

The execute stage will use the control signals and values provided by the decode stage to perform an operation. Living within the execute stage are the ALU, ALU operand forwarding logic, ALU output logic, and branching logic.

The memory stage contains the cache module, which is described in the next section.

The writeback stage sends either memory or ALU output data back to the register file.

*The Cache*

The implemented cache is a two-way set-associative cache. The state machine—is comprised of thirteen stages

**--**

*Brief discussion of optimizations implemented (maximum 0.5 pages)*

**Optimizations**

One of our optimizations lies with the instruction decoding logic. Since a number of data processing control signals from the decode stage have similar combinational logic, we hand-optimized the logic to minimize the amount of transistors required for computation.

**--**

*Discussion about failures, if any (Required for partial credit): A discussion of what does not work and why. Also include what you would have liked to implement given more time. For each part of the implementation that does not work, turn in an annotated output in the form of a trace or script run that clearly shows the error. Give your thoughts as to why the error occurs and what could be done to fix it. (without counting traces this section should not exceed half a page)*

**Project Failures**

**--**

*A table listing the possible hazards that arise in your pipelined design and the number of stall cycles that each hazard incurs (max 0.5 pages)*

**Hazards and Stalling**

**--**

*A brief discussion of your cache design that explains the number of cycles for a cache-hit, cache-miss (with eviction of a line), cache-miss (without eviction) (max 0.5 pages***)**

**Cache design**

Our cache is

**--**

*A conclusion outlining what you learned by doing this project and what you would have done differently (max 0.5 pages)*

**Conclusion**