**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. It was designed for correctness rather than performance. On a cache miss with dirty eviction, the cache will write all entries into memory, and then wait for memory to complete before reading the entire sequence back into the cache.

**Optimizations**

One optimization 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.

**Project Failures**

If we had more time, we would have liked to integrate the cache. We have tested the cache and verified it works, but we were unable to merge it with the processor because of time-constraints. Given more time, we also could have implemented branch prediction and operand forwarding, but due to our massive fustercluck that was the stalling mechanism. we were unable to do so.

If we had more time, we could have also included more optimizations. We know that our stalling logic is one of the slowest parts of our processor, and we're certain there are better ways to stall properly. In fact, we would probably rearchitect the entire processor given more time. It's pretty garbage.

**Hazards and Stalling**

1. Data hazard from Decode -> Execute (2 assuming single cycle memory. if full memory system + cache, min 2, max 2 + (cache miss + dirty write -> 7 + 7))

2. Data hazard from Decode -> Memory (1 cycle assuming single cycle memory, if full memory system + cache, min 1, max 1 + 14)

3. Branch/Jump/Jump register/Jump and link. Results in F/D and D/E pipeline stage flush, resulting in min 2 cycle delay

4. Instr and Data Mem Stall - however long the memory stalls for (see cache)

**Cache design**

There is no penalty for a cache hit. There are two cases for a cache miss, one with no eviction and the other with a dirty eviction. For a miss with no dirty writeback, it takes 7 cycles for the requested data to be available. One cycle is taken for the comparison, and the other six cycles for reading memory. For a miss with dirty writeback, a minimum of 5 cycles are added to a maximum of 7 cycles because we wait for all memory banks to report not busy before reading and allocating cache entries.

**Conclusion**

One of the biggest lessons we learned was the necessity of planning your schedule. Both of us were extremely swamped with work this semester, which adversely affected our final result. Also, we both gained an appreciation for the amount of time and effort required to design a processor. They're incredibly complex devices that are one of the greatest achievements of humanity. One thing we would have done differently is doing more planning and research before diving in. We had many, many issues with our stalling mechanism that would almost certainly been avoided by designing our processor better.