CS/ECE 552 Spring 2019 Final Project

Team 26
Siddhartha Edara - Omar Kurosu Jalil

Design Overview

The main idea behind the project is to implement a processor with the following capabilities:

- Two-way set-associative caches with multi-cycle memory
- Register file bypassing
- Forwarding from beginning of the MEM stage to beginning of EX stage
- Forwarding from beginning of the WB stage to the beginning of the EX stage
- Branches predicted not taken

Phase 1: Unpipelined process with RF bypassing

The basic unpipelined processor is 1 instruction in the processor at any given time.

We drew up a rough schematic design of how the processor will be laid out eventually. We took into consideration adding pipelines later down the line. Furthermore, our layout included taking branches from the decode stage instead of the execute stage.

We coded each part of the pipeline stage individually (Ex: Fetch Decode…). We tested and debugged the processor together at each phase. Furthermore. we used HW files for the register file forwarding.

Phase 2: Pipelines with forwarding

The basic pipelined process introduces registers separating each stage from another as to allow a better CPI rate. However, we had to account for read after write stall conditions.

Instead of implementing stalls for all cases of data or control hazards, we decided to add forwarding immediately to save time. Siddhartha added registers between each stage, and we debugged and tested together.

Phase 2.3: Cache design

The cache state machine had to take into consideration cache misses for both loads and stores. Furthermore, we had to handle cases such as valid and dirty bits or not valid and hit and so on.

Siddhartha redid the state machine we designed for the HW as he spotted noticeable problems. After a rough implementation of both direct mapped and 2 way set associative cache, we spent the next few days debugging and testing our designs.

Phase 3: Final processor implementation

The final processor design took the pipelined with forwarding and added our caches with its state machine.

Siddhartha wired all the individual components together for the final stage, while taking into consideration of stalls from our caches. We spent the next two day debugging stalling and halting issues we faced.

Optimizations and Discussions

We decided early on to do our branches in the decode stage, as a result, we had to implement MEM to Decode and EX to decode forwarding to help with stalls.

This means for forwarding we had:

- MEM to EX
- EX to EX
- MEM to Decode
- EX to Decode

If we had more time, we would have wanted to finish our mem to mem forwarding. We had a rough implementation but after some testing it was not behaving as intended and we scrapped it as a result.

Design Analysis

A table listing the possible hazards that arise in your pipelined design and the number of stall cycles that each hazard incurs. (Maximum half a page)

Our forwarding accounted for a lot of our possible hazards on top of our branching in decode implementation.

These are the possible hazards that will result in our stalling (not accounting for stalling from cache)

- A store instruction after a load with read after write dependency will incur one stall.

Our cache design had the following properties:

- A cache hit takes 3 cycles
- A cache miss with eviction of a line takes 17 cycles
- A cache miss without eviction of a line takes 9 cycles

Conclusions and Final Thoughts

A conclusion outlining what you learned by doing this project and what you would have done differently. (Maximum half a page)

We learned how the basic components in a processor interact with each other and how certain optimizations allow for a better performance. The first two phases allowed us to compare the CPI/performance of a pipelined vs non pipelined processor. Furthermore, we learned about the cases a cache (+ memory) must deal with due to misses or hits and how the cache connects with the processor. We learned about all the little tricks and optimizations computer architects had implemented to squeeze out more performance such as branch predictions and forwarding.

It would be interesting if we could observe via our project how a superscalar processor could be implemented and how all the hazards cases are handled.