CP2 Writeup

Progress Report

John: For this checkpoint, John developed most of the hazard detections and forwarding logic and worked collaboratively with Gally to implement branch prediction

Gally: For this checkpoint, Gally was mainly responsible for tracing out the numerous signals and doing the verification. He identified issues which we as a group would then fix. Gally also worked collaboratively with John to implement branch prediction.

Arjun: For this checkpoint, Arjun was responsible for all things cache related. He mainly focused on modifying his mp3 cache to be one cycle hit. He then made the necessary changes to stop using magic memory. As a group, we then debugged the timing issues we were having with the cache

The testing strategies we used was creating test code and seeing if the intended results were loaded into registers and tracing signals back if registers weren't as expected.

Roadmap

John: For the next checkpoint, John will focus on implementing hardware prefetching which predicts the data that will be used next and prefetches that data. John will also work on a better branch predictor.

Gally: For the next checkpoint, Gally will focus on implementing the riscv m extension which adds new modules for multiplication/division and will potentially need to implement more stall logic. Gally will also attempt to implement the logic for clock gating to keep wasted power to a minimum.

Arjun: For the next checkpoint, Arjun will focus on creating a unified L2 cache so that hopefully many misses that occur in the split L1 cache are hits in the L2 cache and require less physical memory accesses. He will also modify the cache module itself so that it is parameterized and can be reused. Arjun will work on the eviction cache buffer which hopefully cuts down on time taken to retrieve valid data from memory.

CP2 Writeup

Optimization 1: RISC-V M-Extension

The RISC-V M-Extension adds the following instructions:

MUL performs a 32 x 32 register multiply and stores the lower 32 bits of the 64-bit result in a register

MULH performs a signed register multiply and stores the upper 32 bits of the 64 bit result in a register

MULHU performs a unsigned register multiply and stores the upper 32 bits of the 64 bit result in a register

MULSHU performs a signed x unsigned register multiply and stores the upper 32 bits of the 64-bit result in a register

DIV performs signed division of two 32 bit registers, rounding the result towards 0

DIVU performs unsigned division of two 32 bit registers, round the result towards 0

REM returns the remainder corresponding to a DIV instruction

REMU return the remainder corresponding to a DIVU instruction

Implementation

Our hope is to implement an advanced multiplier, specifically the Wallace Tree method. We will implement division in its own module using elementary riscv operations. We foresee that we may have to add logic to stall the pipeline while these computations are made as they will take more time than typical alu ops. We may have to reconsider how the pipelines are currently being stalled.

CP2 Writeup

Optimization 2: Unified L2 Cache (Parameterized Cache)

We noticed that the frequency at which we can get a response from our current cache arrangement really can affect how many cycles we have to stall as we wait for data. We therefore want to implement a cache hierarchy. This can be a unified cache because we care less about bandwidth in this second level of the hierarchy. We want this cache to be large enough either in cache line size or way size and most likely fully associative so we can have full utilization of this cache. We hope to significantly decrease access to physical memory. We also hope that by making it a parameterized cache we can play around with different cache arrangements.

Optimization 3: Hardware Prefetching

This seems like it may prove to be the most difficult of the optimizations to implement and actually make worth our while in terms of saving on memory access times. We would like to implement the more advanced stride based prefetching but may choose to do the basic one block lookahead prefetch as it takes advantage of spatial locality and could be easier to implement.

Optimization 4: Eviction Cache Buffer

This is another optimization to try and decrease the delay when accessing physical memory. We will create an eviction register so that on a dirty cache eviction, we can hold the to be evicted value in the register and prioritize serving the correct data into the cache before doing the writeback when the opportunity is available.

Optimization 5: Clock Gating

Clock gating is our effort to make our cpu more power efficient by removing the clock signal when it is not being used. Any clocked module when not in use does not need to be clocked. For example if we have a multiplier that is not using the alu, a multiply instruction doesn't need to have a clocked alu in the EX stage.

Optimization 6: Branch Predictor

We will try and implement a backwards taken/forward not taken implement as we notice the competition code makes use of a lot of loops so the backwards branch at the end of the loop will be taken and all other non-loop branches will be not-taken. As a fall back option we could implement a 2 bit branch predictor that at least has some sense of what happened the previous time a certain pc was encountered.