Anthony Nguyen – alnguyn2

Hemang Nehra - hnehra2

CP2: Progress Report

Functionalities

In this checkpoint we integrated the provided caches for the d-cache and i-cache with a mediating arbiter that prefers to respond to i_cache misses in case both i_cache and d_cache miss. Next, we implemented a hazard detection unit that flags data hazards in MEM -> EX, WB -> MEM, and WB -> EX stages of the pipeline. It is done so for by providing a combinational logic that checks RaW, RaW, WaW by checking the equality of rd register in stage_wb, rd register in stage_mem, and ALU inputs in stage_exe for different hazards. We also stall the decode and fetch stage in case the current opcode in stage_exe is op_load (lw). This is to prevent parallel access to the regfile in stage decode. The last thing that we implemented for this checkpoint is the static Branch Not Taken for the branch hazard resolution. For this, we modified our data path and added an extra squash logic to reset the decode stage in case branch is taken.

Contributions

Anthony worked on integrating the d_cache and i_cache to the previously implemented cpu. Anthony completed the decided control logic and Hemang completed the datapath. Hemang completed the hazard detection unit and Anthony continued that part by creating muxes for the forwarding logic. Updated top-level datapath together. Worked on branch hazard resolution together. Each team member worked on the implementation and debugging of the design.

Testing Strategies

To verify the design, we looked at the values coming through the spectating stages. We also kept a constant look at different values of PC register, regfile, and other signals and made sure that they were moving as expected. We changed the testcode at several instances to make sure that our pipeline was able to cover all the foreseeable cases. For example, once we had built upon the entire checkpoint we checked if our previous checkpoint's expected values were still correct, and then accordingly changed the code to test the different forwarding paths.

CP2: Roadmap

Group contributions

The advanced features that we plan on implementing are Basic hardware prefetching, Tournament branch predictor, and victim cache. For this checkpoint, Anthony will work on prefetching idea and design. Hemang will do banked L1 cache. We both will do victim cache at the last. Then we will add and discuss ideas to each other's designs. Writing code and debugging will be mostly done together.

Required functionalities

Given that both our caches were non-functional in MP3, we will take this opportunity to deepen our current understanding of the caches. For this, we will be implementing victim cache. Apart from this, we

will implement a tournament branch predictor to improve the branch prediction and prefetching of instructions as well on our basic 5 staged pipeline.

CP2: Advanced Features

The three advanced functionalities that we aim to include into our CPU design are

1) Basic hardware Prefetching:
   We are implementing the basic hardware prefetching using the One Block Lookahead, prefetch on cache miss algorithm. For this, whenever we are accessing line i and go through a cache miss then we prefetch line i+1, else, we do not. We must make sure that i+1 is not already in the cache, otherwise we would be wasting cycles.

2) Victim Cache:
   We have not yet decided on how we would go about solving the problem, but we assume that in the starting it would be something similar to the eviction write buffer. Wehenever there is a miss in the L1 cache the clean or dirty data that is to be written back to pmem is now stored into the victim cache. Now later, if there is a L1 cache miss for the data thatwas placed in victim cache, the L1 cache asks the victim cache for the data (and victim cache has it). Thus, a victim acache hit takes place and we swap the contents of the L1 cacheline and the matching victim cacheline. Therefore, the path from L1 cache to pmem is now modified to include the victim cache as well.

3) Tournament Branch Predictor:
   As the documentation mentions, we will need to branch prediction models first to use the tournament branch predictor model. For our case, we will be using the local branch history table and a global predictor. Whichever predicting model provides more accuracy at the current time of the branch prediction will get to provide the prediction to the pipeline. For implementing this, we can keep a relative count between the two models and see which predictor's true answers exceed the other predictor's number of true values. The tournament branch predictor could then store 0 or 1 in a register like a LRU.