ANIRISC-V

By: Anirudh Kashyap, Vishnu Jagan, Ethan John

Advanced Features

- Cache
  - L2 Cache
    - 2 way 16 set L2 cache acts as an intermediate between given cache and memory
  - Eviction Write Buffer
    - Buffer holds stores dirty evicted lines, allows memory to be read from first
- Branch Prediction
  - Tournament Branch Predictor
    - Chooses between local and global table to decide which one is better using two bit counter
- Prefetching
  - PC Based Stride Prefetching
    - Built into the arbiter, the stride prefetcher remembers distance to previous memory access and prefetches data using that stride
    - Falls back on prefetching the next instruction cache line if there is a hit for the given stride forward

Quantitative Analysis of advanced feature code

- Branch predictor above 60% for competition code. 77% on cp3 code.
  - To track, used perf counter on testbench by tracking cmp results when branch in ex phase
  - 5% performance increase compared to static not taken
  - Static taken performs better by 2%. Loops with 200 iterations (on test 3 for example) will be taken 99% of the time, so just static taken performs better.
- The stride prefetcher with fallback on next instruction prefetching
  - This hybrid prefetching scheme got around an average of 10% reduction in runtime across the competition codes
  - The prefetcher does best with regular memory accesses in a loop in the case of stride prefetcher and code with no branches for the instruction prefetching
- *L2 cache was the biggest performance booster. ~60% reduction in execution time*
  - *Parameterized*
- *Eviction Write Buffer*
  - *Works better for some code (comp1), worse for comp3*

Possible Improvements

- Biggest: Understanding of what CP3 test code entailed
  - Did not realize CP3 test code was much more in depth than CP2 Test Code
    - Decent amount of time spent on this, we built a grading harness that printed out register writes, memory loads and stores, and jumps and branches
    - We diffed this output with a working mp2 cpu and had another python script that outputted the simulation time of the error on both mp2/mp4
    - Knowing the exact time allowed us to easily run and see all signals to compare from mp2/4
    - If we had a week to do advanced features, we would look into implementing a trace cache, and potentially a fully associative L2 cache
  - Due to a few bugs with our CP3 test code, we had to focus on that more than building extravagant features
- CP1 and CP2 *{Things we would have changed in CP1 and CP2}*
  - RVFI MON from beginning, had we figured this out initially, our lives would have been easier as we wouldn't need to build a grading harness
- *Design Comp*
  - Had we a bit more time, we would have made several levels of cache and implemented full associativity as well as a large number of sets, we found this in our L2 cache to have a big decrease in runtime