Progress Report

Eric

- Linking signals properly from new CP2 modules to datapath
- Switching from magic memory to L1 cache + debugging it – modified top level files and set-up regfile connections with memory
- Traced signals into cache until instructions were being loaded properly.
- Worked on debugging arbiter to properly send signals through either end.

Gerardo

- Cleaned up design to show more clear view of forwarding and hazard detection unit
- Debugged forwarding and hazard detection
- Worked on stall and branch prediction logic within datapath
- Worked on arbiter file and logic

Bryan

- Worked on hazard detection unit and forwarding unit.
- Worked on the static branch predictor
- Set up most of RVFI
- Set up transparent regfile

Functionalities

Correctly handles data hazards (via forwarding and transparent register file), read after load stalls, and branch hazards. CPU properly communicates with two caches, and the arbiter handles which cache gets serviced at what time. Static branch prediction predicts not-taken, and properly flushes the pipeline in the case of branch taken.

Testing Strategy

We first implemented the majority of the checkpoint requirements except the cache, and verified that it functioned as expected before introducing code pertaining to the L1 cache and arbiter. When debugging the cache, we traced all the way down to the cache control signals to determine where signals are being muddled and setting them properly.

Timing/Energy Analysis

See .rpt files

Roadmap

Advanced Features Proposal

- L2 Cache (2) – Additional cache block to speed up memory operations. Should be bigger than L1 cache and sits between our L1 caches and arbiter.
- Bimodal branch predictor (2) – 4 state branch prediction model outlined in class.
    - Potentially expand it to be a tournament branch predictor, since it utilizes the bimodal predictor anyways. (additional 3 pts)

- Basic hardware prefetching (4) – if we anticipate a cache miss, we will prefetch the data so that we will have it in cache before we actually have a miss.
- RISC-V M Extension (5) – add native CPU support for M-extension instructions. Create wallace_tree file to handle multiplication from scratch
- Eviction write buffer (4) – allows faster cache operations to be processed first before writing back dirty blocks, which are more timely.
- Return address stack (2) – Stores address of next instruction onto a stack in preparation for a jump

Roadmap

We will each independently tackle an advanced design feature that interests us and are about the same amount of work. The assignments are as follows:

Eric – bimodal branch predictor/tournament predictor
Gerardo – RISC-V M Extension
Bryan – basic hardware prefetching

We will get each design working on top of our current CP2 design before combining them. Combining all our individual features will likely lead to some more bugs, so we will debug together to ensure all 3 features can work at once. Only once this is done will we tackle augmenting the cache.

We will start by connecting two L2 caches in between the existing L1 caches and arbiter. The L2 caches should be bigger than the L1, so we will have to modify the caches to have different number of cachelines depending on the level. The functionality should be the same as that in CP2, with improvement on cache misses down the line. At the same time, we'll implement support for the eviction write buffer.

Our last proposed feature, the return address stack, should be straightforward to implement and handled once our cache features are complete.