Benjamin Kalnbach, Ethan Chow, Tongyu Liu

bak7, ethanc6, tongyul3

11/06/2023

CP2:

**Progress Report:**

CP2 involved implementing data hazard detection, forwarding, a static branch predictor(enabling branching and flushing as well), L1 instruction and data caches, and an arbiter to arbitrate which cache should interact with pmem at any given point. Benjamin handled data hazard detection and forwarding. Tongyu handled branching and the L1 caches. Finally, Ethan handled the arbiter. With these implementation several functionalities were enabled. Forwarding and hazard detection enabled the pipeline to not stall(or just put a bunch of nops between instructions) in order to wait for data to be written to the regfile. While implementing the L1 caches and the arbiter didn’t really add any functionality, it did do away with magic memory and made debugging memory operations easier as a result. Branching was not supported in the CP1 design, in CP2 that is now a working instruction allowing more complex source code to be ran(or just the ability to halt…). Most of the testing was done by Benjamin, in part this was due to forwarding being highly dependent on timing. The testing methodology this checkpoint was similar to that of CP1, that is run source code that checks general and corner cases. This required multiple source codes, mostly due to the timing of the CP1 design not being quite correct, requiring testing with and without nops to avoid timing issues. In order to do some of the debugging we used a working MP2 processor to run the same source code and compare results to make sure the pipelined CPU works correctly.

**Roadmap:**

The plan moving forward is to start implementing features beyond just those necessary to run source code adhering to the RV32i ISA. The main goal of this is to increase performance, primarily by decreasing memory latency. To achieve this we will implement multilevel caches, local branch history table, table victim caches, branch target buffer, basic hardware prefetching, and basic implementations for the RISC-V M extension. Benjamin will work on the RISC-V M and basic hardware prefetching. Tongyu will work on multilevel cache and victim cache. Ethan will work on local branch history table and the branch table buffer. Most of these will speed up memory accesses and branching operations. That is with the exception of the RISC-V M extension being enabled, which will allow further functionality allowing for multiplication and division to be used in source code. Some issues potentially are mostly going to be dealing with timing issues. None of us are particularly good at addressing timing issues once the design is complete, but buggy. The aim of our picks is to mostly side step this issue, but we can’t really avoid it entirely. Another issue is making time to do all of these implementations, it may require we put in work even during the break. One last issue is our caches are still a little iffy. So taking on further cache related features may pose a potential issue going forward with CP3.

**Advanced Features Proposal:**

1. Multilevel Cache System[2]

-It’s a fairly easy feature to enable, you just add another cache in between the L1 cache and the pmem

-It should cut down of memory latency improving performance

2. Local Branch History Table[2]

-Will increase branch prediction accuracy for repeat branches

2. Victim Cache[4]

-Will keep some of the evicted blocks around for awhile

-Main benefit will be avoiding conflict misses

3. BTB[4]

-Worth Points, and we were running out of good options

-It’s a buffer so it shouldn’t be too bad, but using it will require some potentially tricky logic

4. Basic hardware prefetching [4]

-Fairly simple to implement/design and worth a moderate amount of points

-Also, conceptually easy to understand

3. RISC-V M Extension[4]

-Fairly Basic, but worth a decent amount of points

-One draw back is it means that exe stage will now take longer than one cycle if it’s doing a multiply