Benjamin Kalnbach, Ethan Chow, Tongyu Liu

bak7, ethanc6, tongyul3

10/31/2023

CP1:

**Progress Report:**

The main component of CP1 was the pipeline and the control unit. The pipeline consists of several smaller units: fetch stage, fe\_de pipeline register, decode stage, de\_exe pipeline register, execute stage, exe\_mem pipeline register, memory stage, mem\_wb pipeline register, and the writeback stage. These components had everyone making small corrections and input, but most of the work was split as follows. Benjamin did the execute stage, memory stage, exe\_mem register, mem\_wb register, the control unit, ¼ of the datapath, and the mp4 top level file. Ethan completed the writeback stage, and corrected many of the syntax errors that were abundant throughout the design. Tongyu handled the fetch stage, decode stage, fe\_de register, de\_exe register, and ¾ of the datapath. The functionality of our implementations is a pipeline that supports running multiple instructions at once(not using the same resources), but without support for branch and jump operations. While the pipeline doesn’t support flushing at this point it should support stalling, although we are choosing not to test that since it is not a requirement for CP1. In terms of testing and verification the main strategy is to test the design as a whole rather than unit testing individual components. This method is potentially quicker, but the trade off is it makes it harder to debug individual components. In terms of what the source of the input signals for testing was, we chose to go with driving the design using a testbench since we couldn’t halt the program with an infinite loop within the source file without branching enabled.

**Roadmap:**

CP2 focuses on enabling branch predictions via a branch predictor(static), hazard detecting and forwarding between stages, an arbiter to dictate if I cache or D cache should be interacting with memory, and integrating L1 caches into the design(I cache and D cache). Enabling branching should allow the core to be able to run the full RV32I isa. The main benefits of integrating caches and the arbiter is to do away with “magic memory”. Magic memory was previously handling all memory operations and immediately giving us the data rather than the more realistic wait times one would expect. Hazard detection and forwarding should increase performance, and should make it unnecessary to insert nops into the source code in order to make sure each instruction has the correct data written back by the time the next instruction needs it. A potential issue is that no one in our group designed a fully functional cache, this may lead to integrating it into the design being more difficult than it would otherwise be. The plan is for Ethan to handle the arbiter and static branch predictor, while Tongyu and Benjamin work on getting the L1 caches integrated and implementing hazard detection and forwarding. Another potential issue is that CP1 was completed late, so that leaves less time to implement and test CP2 before the due date. Likely we will not have time to get the extra credit of making sure CoreMark runs. Hopefully for CP2 we can take more time to unit test components and the overall design, but with the current time crunch we are facing that seems to be an unlikely occurrence.