Team Pied Piper Checkpoint 3 Progress Report and Roadmap

Progress Report:

Who worked on each part of the design:

Unlike the previous two checkpoints, we decided to split the several advanced features that we plan to incorporate between the three of us. We decided on this due to the fact that each advanced feature requires a solid understanding of how it conceptually works and a decent amount of programming which would be difficult for us to attend to all the features together as a group due to time restrictions. Due to this, Ishaan worked on the cache portions of the advanced features, such as parameterized cache, L2 cache, and 4-way set associativity while Steven worked on prefetching and the eviction buffer while Megh worked on the M extension and implementing performance counters.

Functionalities we implemented:

For checkpoint 3, there were several advanced features that we implemented. The first is the dirty eviction buffer which holds dirty data that is to be written to physical memory. This implementation should speed up our design as we can return the new data from memory and continue operations as normal while the eviction buffer will hold the dirty data that is to be written and manage that write on its own time. Upon implementation, we noticed a slight improvement in our speeds. Another advanced feature that we implemented was prefetching. We implemented prefetching solely with regards to the instruction cache as we are guaranteed to fetch new instructions from memory every 8 clock cycles. Due to this, we knew that instruction prefetching will always get a lot of use due to spatial locality. We did not implement prefetching with data cache since the memory locations of the data we access are often all over the place. Furthermore, we implemented the M extension instructions. We did this by creating a multiply and division module and an upper-tier muldiv module that takes in which M extension instruction we want to execute and passes the data along to the proper module within it. We also modified the hazard detection unit and forwarding to account for these new instructions as we will have data dependency as well as the need to stall the pipeline since the M extension instructions take multiple clock cycles. Last, but not least, we implemented several cache-related advanced features. We added an L2 cache to have more cache space available, 4-way set associativity to increase the data held by cache, and parameterized the number of sets to make our cache design more flexible and efficient as it can adapt the needs of the program.

Testing Strategy:

Regarding testing, we used the same testing strategy that we have been using for the past two checkpoints. Essentially, to confirm that our baseline design worked as intended, we ran all three checkpoint codes simultaneously on our MP4 design as well as our working MP2 design. We would then signal search and compare the register values between these two designs. If there was an unintended value in one of the registers in our MP4 design, we would look at the PC value of that instruction, locate it in the assembly file, and run that snippet of code separately to determine exactly what instruction, or combination of instructions, was causing our design to have incorrect values. We would then look at the part of the datapath that corresponds to the "bad" instruction(s) and modify the necessary modules or logic elements as needed. Once we confirmed all checkpoint codes worked, we repeated this process with the competition code as those test codes contained several edge cases that the checkpoint codes did not account for. When we implemented our advanced features, we knew that our final regfile values should not change, but rather only our design should be faster, or in some cases slower. When we would incorporate a new advanced feature, we would ensure that the final regfile values matched the values before we added that feature to ensure that the CPU was still functioning properly.

<u>Performance Counters:</u>
Below is a table regarding the metrics measured by our performance counters. We implemented performance counters in the form of physical registers that would hold the number of times a certain advanced feature was used by the end of program execution. We also recorded the runtime of several programs to showcase whether the selected advanced feature improved or decreased our design's performance.

Advanced Feature	**MP4-CP1.s Runtime**	**MP4-CP2.s Runtime**	**MP4-CP3.s Runtime**
None	7,695ns	4,945ns	5.196 million ns
Dirty Eviction Buffer	7,705ns	4,955ns	5.118 million ns
Performance Counter (# of times used)	1	2	2,465
Instruction Prefetching	15,025ns	7,975ns	7.175 million ns
Performance Counter (# of times used)	16	8	6,739
M Extension	7,695ns	4,945ns	5.196 million ns
Performance Counter (# of times used)	0	0	0
L2 Cache + 4-Way	8,025ns	5,115ns	1.776 million ns

Set Associativity + Parametrized Sets			
Performance Counter (# of times used)	Cache Inst L1 Miss: Inst L1 Hit: 204 Data L1 Miss: 1 Data L1 Hit: 16 Inst L2 Miss: 16 Inst L2 Hit: 220 Data L2 Miss: 1 Data L2 Hit: 17	Cache Inst L1 Miss: 16 Inst L1 Hit: 204 Data L1 Miss: 1 Data L1 Hit: 16 Inst L2 Miss: 16 Inst L2 Hit: 220 Data L2 Miss: 1 Data L2 Hit: 17	Cache Inst L1 Miss: 16 Inst L1 Hit: 204 Data L1 Miss: 1 Data L1 Hit: 16 Inst L2 Miss: 16 Inst L2 Hit: 220 Data L2 Miss: 1 Data L2 Hit: 17
All Advanced Features together	15,055ns	8,005ns	1,683,555ns
Performance Counter (# of times used)	Eviction Counter:1 Prefetch counter:16 Multiplier counter:0 Cache Inst L1 Miss: 16 Inst L1 Hit: 204 Data L1 Miss: 1 Data L1 Hit: 16 Inst L2 Miss: 16 Inst L2 Hit: 220 Data L2 Miss: 1	Eviction Counter:2 Prefetch counter:8 Multiplier counter:0 Cache Inst L1 Miss: 8 Inst L1 Hit: 151 Data L1 Miss: 2 Data L1 Hit: 10 Inst L2 Miss: 8 Inst L2 Hit: 159 Data L2 Miss: 2	Eviction Counter:1947 Prefetch counter:91 Multiplier counter:0 Cache Inst L1 Miss: 91 Inst L1 Hit: 160972 Data L1 Miss: 828 Data L1 Hit: 11052 Inst L2 Miss: 91 Inst L2 Hit: 161063 Data L2 Miss: 1274

	Data L2 Hit: 17	Data L2 Hit: 12	Data L2 Hit: 16539

Roadmap:

<u>Who is going to implement and verify each feature and functionality:</u>
Since the next checkpoint is the design competition, our full focus will be shifting on ensuring that our design meets the timing and power requirements. In order to do this, we will regroup and work on optimizing our design together as it will become difficult to split up the optimization and parameter tuning. Our plan on optimizing our design will be to modify our advanced features to speed up our design. This can include anything from reducing the number of states in our state machines, changing parameters such as changing the number of sets in our cache, or even removing an advanced feature if we have difficulty modifying it. Furthermore, we will begin preparations for our final report. These preparations will include updating our paper design to include all advanced features as well as contain any modifications we made throughout our design process and begin writing a rough draft of the report. We will also begin preparations for our presentation.

<u>What are those features or functionalities:</u>
Features & Functionalities to be implemented for next checkpoint:

- Optimize design to ensure maximum speed and efficiency
    - Modify/Tune advanced features, if applicable
    - Remove advanced features, if necessary
    - Look at coherent design of our baseline CPU and see if there are any improvements that can be made
- Presentation and Final Report
    - Update paper design to include advanced features and all modifications
    - Begin rough draft of final report
    - Prepare and rehearse for presentation