Progress Report (CP3):

For this checkpoint, we implemented our advanced features from our design from CP2. The advanced features we ended up implementing was a Local Branch Predictor and a Global Branch Predictor to form a Tournament Branch Predictor, a Branch Target Buffer (BTB), a Return Address Stack (RAS), Basic Hardware Prefetching, an L2 Cache with parameterized sets, and an Eviction Write Buffer. We started off this checkpoint by integrating our cache from MP3. We had to modify our cache in order to make it a 1-cycle hit. Using this cache instead of the given cache significantly increased our efficiency in terms of run-time. The first feature we added was the Tournament Predictor. We added our state machines to choose between the prediction levels of the local and global predictors, and then connected the BTB in order to actually go to the address of the predicted branch. When testing our code, we realized that there were various conditions that we did not add for when we mispredict branches, and therefore our BTB was not getting us to the right address. After going through the CP3 code and writing our own test code with loops, we were able to resolve these errors. We then added the EWB, which allows for reads to have priority, and for writes to be done back to memory when we have the time to service them. This feature did not give us too much trouble, so we were able to move on to the prefetching. Within the prefetching unit, we had to modify our cache and the arbiter so that a signal could be sent from the cache to the prefetching unit in order to grab the next instruction. After testing this and seeing if our prefetcher updated states properly (tested by writing our own code of instructions as well as the competition/checkpoint code), we then moved on to the L2 Cache. For now, our L2 cache is the same as our L1 cache with a greater number of sets, but we have a plan to change this. The last feature we added was a RAS. Our RAS holds the addresses of our JALR instructions so that when we return, we can return to the correct address without having a delay.

Throughout testing in this checkpoint, we had errors with shadow memory and rvfi errors, and this helped us find errors within our forwarding paths as well as logic. For example, we noticed that the conditions we chose the BTB and RAS were not always correct, and therefore we had to edit our multiplexer conditions to account for this. Although these features are not the most efficient they can be, we have added these features successfully and can note where they work. We all worked on all these advanced features for debugging, but we all tackled certain advanced features at first attempt. Yash did the Tournament Predictor, L2 Cache, and RAS, Aayush did the EWB and RAS with Yash, and Vishal worked on the Prefetching Unit and BTB. We all looked at the waveforms over screen sharing and also on Live Share to modify the code together, and then modified each of these modules while debugging. Aayush also worked on using the cache from MP3 instead of using the given cache, as well as making it a 1-cycle hit.

Roadmap for CP4:

For checkpoint 4, our plan is to analyze our performance counters and see what modules and features are worth keeping in for the competition. Thankfully, in this checkpoint, we made sure our processor could run all the competition code without any errors. However, our focus in CP3 was for implementation and not necessarily efficiency. Our fmax is not at the level we would have hoped, but this is because we have some long combinational delays and critical paths that are limiting our efficiency. What we hope to do is tackle these paths 1 by 1, so that we can shorten our delay. We also parameterized the BTB and RAS in our code, so we will experiment with the sizes and the tradeoffs with these. We will all look at the features we added and see what delays we can cut down individually, and then once again debug together by looking at the timing analyzer and making changes.

Performance Counters & Analysis:

Note: All code was run with Checkpoint 3

Branch Prediction, BTB, RAS:

This section was the biggest advanced feature we added into our processor. By adding in Branch Prediction, we're able to predict the address of the next instruction instead of waiting for a calculation of the address. Our tournament branch predictor lets us pick the value from the local branch predictor or global predictor based on the correctness of these two. The advantage of this is that if the program is smaller, the local branch predictor does well, and on bigger scales, the global predictors. The RAS lets us have more hits on JALR instructions, while the BTB takes care of BR and JAL instructions, letting us have significantly more hits. We can see that the branch prediction significantly decreased the number of I-Cache Stalls and Flushes, overall improving in every category. The RAS was analyzed using Competition Code 1 because checkpoint 3 did not include any JALR instructions, which means that the RAS was not used.

Prefetching:

The prefetcher grabs the next instruction by grabbing the i+1 line whenever we access the i-th block. We do so by sending a signal as a prefetch request to the cache whenever the next instruction is needed. This decreases the number of icache stalls we have. In our table, when we ran all the advanced features without the prefetcher vs. Prefetcher included, we can see that adding the prefetcher decreases the number of I-Cache Stalls we have and also the number of I-Cache misses. We spent about 80% less time in stalls with I-Cache and had 13 less I-Cache misses.

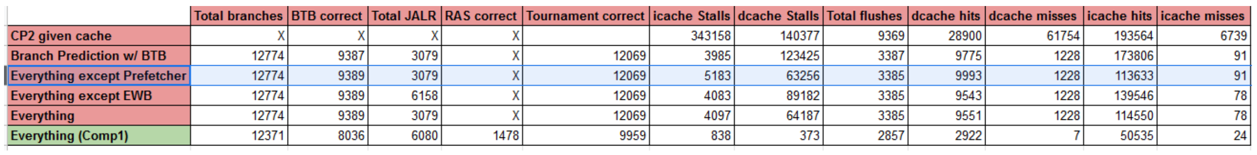
EWB:

The EWB takes care of writes whenever we don't service a read, which decreases the amount of time we spend stalling. In our table we can see that when we ran the code with all advanced features vs. all the advanced features without the EWB connected, we had significantly less time stalling for D-Cache, with about 72% of the amount of cycles in that stage. There were also more hits on the D-Cache, which shows that we were able to grab our data better. Although we did have more I-Cache stalls, the D-Cache improvement shows the importance of having this.

L2 Cache:

The L2 Cache adds a unit between the I-Cache/D-Cache and the Arbiter. Having this unit be a bigger size than our L1 Cache and lay in between these units lets us save time going back to physical memory and therefore decreases the amount of time spent stalling.

Performance Counter Data:



	Total branches	BTB correct	Total JALR	RAS correct	Tournament correct	icache Stalls	dcache Stalls	Total flushes	dcache hits	dcache misses	icache hits	icache misses
CP2 given cache	X	X	X	X		343158	140377	9369	28900	61754	193564	6739
Branch Prediction w/ BTB	12774	9387	3079	X	12069	3985	123425	3387	9775	1228	173806	91
Everything except Prefetcher	12774	9389	3079	X	12069	5183	63256	3385	9993	1228	113633	91
Everything except EWB	12774	9389	6158	X	12069	4083	89182	3385	9543	1228	139546	78
Everything	12774	9389	3079	X	12069	4097	64187	3385	9551	1228	114550	78
Everything (Comp1)	12371	8036	6080	1478	9959	838	373	2857	2922	7	50535	24

Updated Results:

https://docs.google.com/spreadsheets/d/1ZRosALYvTiLPgfbULImkcBphulsBPAGZc3_0WJ0grW0/edit?usp=sharing