**Checkpoint 2 Progress Report:**

Due to increasing complexity in superscalar, we've temporarily halted implementing it to focus on a working processor first.

Cameron:
Worked on regfile and preliminary scoreboard. The regfile will be intended to have 2 read 1 write ports per superscalar, with a priority order to resolve conflict write. The scoreboard was modified by Zhihao to track instructions in ROB that are written to a specified register.

Bobby & Zhihao:

1. For the fetch module, we send fetch signals when the instruction_queue is full or not. We ensure that only one memory reading request is sent per PC..
2. For the instruction_queue, we improved checkpoint 1's design such as combining decode logic. The decoded information is used by an "issue" module for assigning to reservation stations.
3. For the issue module, we send issue signals based on the instruction type to be issued and reservation station status. ROB and regfile scoreboard are also marked accordingly for an issue.
4. For the ALU reservation station, we have one ALU unit attached for execution. Since ALU contains only combinational logic, it can be finished in one cycle. So the cycle after execution the computed output is written to a dedicated slot in the CDB. The reservation station also has logic snooping the CDB to resolve input data dependency, as specified in Tomasulo's algorithm.
5. For the multiplication reservation station, we have one given shift_add_multiplier attached for execution. Internally we have some registers keeping track of the status of the multiplier, such as whether it is busy and which station is executing. The computed result is also passed to another dedicated slot in the CDB once it's finished.
6. For CDB, we have decided to have one slot for a type of reservation station. This design choice is made to save the trouble of saving computed results, as well as to speed up the process of writing to ROB. Since we have two types of reservations for now, there are two slots in the CDB. This design choice means that the components reading the CDB need to read all the slots, resulting in increased area. This increase in area is justified because of the speed benefit and less complication in saving computed results.
7: For the register file and scoreboard, we can keep track of each register's value and the status of any in-flight instruction that will be written to a specific register, as described in Tomasulo's algorithm.
8: For the ROB module, we created an in-order commit queue that could update the computed value at specific ROB by snooping from the CDB. We also have most of the RVFI signals here, which are initialized during the issue stage.
9. For the commit module, we wrote some simple logic to drive the commit signals based on CDB and ROB status. We ask the ROB to pop when it's ready to commit, and this signal is used as RVFI's valid signal.

The processor is tested by running dependency_test.s and ooo_test.s on it and verify the results by observing the waveform and signals. For timing, we were able to achieve a maximum of frequency with a clock period = 1.8ns.

**Checkpoint 2 Roadmap:**

Bobby & Zhihao: (to be divided later between us)

- Load/store queue
- Branch instruction support
- Overall functionality with coremark

Cameron:

- Cache integration

All three of us:

- Make sure synth, lint, and compile give no warnings