10/21/22:
Progress on the pipeline datapath
We have implemented the five stages of our pipeline. Fetch reads the instruction data using pc from memory. Decode uses the instruction data to form the control word, which is used to control subsequent stages. The execute stage computes the decoded instruction via ALUs and comparators. We also included a branch comparison unit to properly handle branch and jump instructions. Memory access is used for loads and stores by reading from or writing to memory. Writeback writes the correct value into the register. During writeback, if it detects the halt condition, it will stop the entire pipeline.

For testing strategy, we went through the register values after running the cp1 test code using our pipeline design and compared it to the register values using our mp2 cpu design to ensure that it produces the correct values.

Jing: Decode, Fetch, Writeback
Albert: Execute, forwarding unit, designed hazard detection
Justin: Control word Rom

Our timing analysis returns a slack (MET) of 6.34, which indicates that it can be synthesized.

Next Checkpoint:
Static hazard detection: we will check for branches in ID and stall for 2 cycles afterwards; we will also check for read after writes where there is a load in Ex and a read in ID that depends on that load.

Arbiter: we will prioritize the D-cache reads first because there is no point in stalling the end of our pipeline in favor of the front.

L1-cache: we will make use of our set associative cache we designed in MP3 and turn it into a 1 cycle hit by merging the idle and check states so that it is continuously checking for hits regardless if a read or load instruction is being performed. In the future, we want to turn it to a pipeline cache.

Advanced features: We will begin brainstorming advanced features and start on working on the design in our datapath.

Jing: L1-cache
Albert: Static hazard detection
Justin: Arbiter