CP2 Progress Report

Work breakdown and implementation of functionalities

Lakshya worked on the RVFI monitor connections, as well as the code for the forwarding unit. Peter worked on hazard detection and arbiter, both with Mridul. Peter also worked on the branch predictor and L1 cache as well. Mridul worked on the design for the next checkpoint as well. All three attempted their share of debugging the code.

Testing Strategy

The testing strategy was to compare the outputs of the CPU with the outputs of the CPU from previous MPs. Because of the use of the RVFI monitor, MP2's code was the main code used for comparison.

CP3 Roadmap

Projected work breakdown

We have chosen many advanced features, and we most likely will spend a lot of time working with each other on individual features. However, here are the main features that each person will be mostly responsible for: Mridul will work on the eviction write buffer (4 points) and the RISC-V M extension (the base will be 3 points, but he will try to get 5 points). Lakshya will work on the pipelined L1 caches (6 points) and the branch target buffer (1 point). Peter will work on the local branch history table (2 points), and the parameterized and associative caches. For the associative cache, he will start with the basic 4-way cache (2 points), but if he has time, he will expand to 8-way (3 extra points). For the parameterized cache, he will start with parameterizing the LRU (1.5 points), but if possible he will also parameterize the set and ways of the cache (3 extra points). This is a total of 19.5 for the basic features, and 27.5 points with all possible extensions completed.

Potential issues

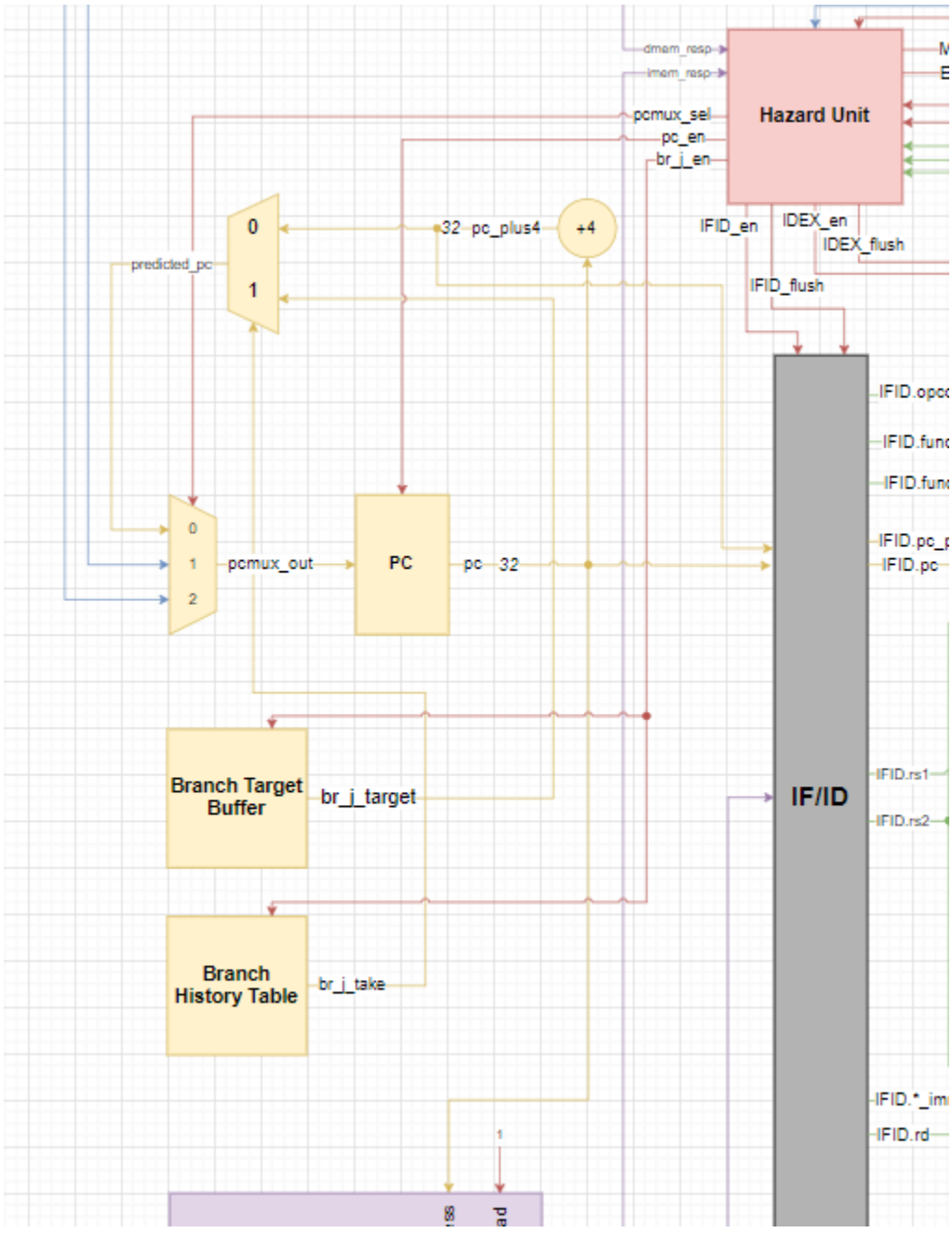
Possible issues include some of the advanced features somewhat clashing with each other or simply being unable to get a feature to work – in such a

case we will attempt to use different features that will finalize with the TA during the meeting as a backup option. Such features could be some form of prefetching (the basic is 4 points, and the advanced is +2 points). These would also be attempted for extra credit if all work was finished before the deadline.

Advanced Feature Designs

- Parameterized Cache
    - Go through entire cache code and replace any usage of constants with parameterized inputs
    - Parameters include:
        - Number of Ways
        - Number of Sets
        - Cacheline size
    - To be able to parameterize for a variable number of ways, need to use loops to instantiate multiple array and data_array blocks.
    - Will need to generalize LRU logic to be parameterizable.
- 4-Way Set Associative (potentially 8-way)
    - Implement this after doing parameterized cache
    - 4-way and 8-way can be adjusted after parameterization
- Pipelined L1 Cache
    - Modify the cache so that it can receive a new request while responding to the previous request.
    - Will need to add registers to store information for the previous request.
- Local Branch History Table
    - 2-bit dynamic prediction
    - Each time we have a memory request in the FETCH stage, we check the BHT for the predicted branch result
        - By default, we will predict Taken
    - We get the target from the BTB at the current PC Address
        - The PC Address will be indexed based on what the size of our BTB/BHT is
- Branch Target Buffer (w/ support for Jumps)

- This module is used to store the target addresses for Branch and Jump instructions
- It will be indexed using the PC value (a smaller portion of the PC depending on the size of the BTB)
- The predicted target addresses will be updated on mispredictions



- M Extension
  - This module specifies the multiplication and division instructions

    - Operations are to be implemented using logic, not the operators
        - Start with using the basic add-shift multiplier from MP1
        - A more advanced multiplier will be implemented after basic 1 is made
    - The object is to test and verify all instructions
- Eviction Write Buffer
    - This module is meant to hold dirty evicted blocks on an eviction
        - Start with storing the block
        - Then read the first subsequent missed address and process it
        - Lastly, check when the next level is free, and write back the evicted block when free
    - The purpose is to make the CPU receive missed data faster instead of waiting on the dirty block
        - We test this normal code and the write buffer code, and make a comparison