Phase 4- Recovery and Branch Prediction

Deadline: Sun., Dec. 14, 11:59 PM

Fall 2025

(Upload it to Gradescope.)

Notes

- You will be implementing these modules and reusing them throughout the period of the project

- You are allowed to use AI tools for writing code as long as you understand what you're implementing and properly indicate what has been written by the tool.

- This phase (and all future phases) can be done in groups. Your group should have been listed under this document: Honors Group

Important Note: If you decide not to complete this phase, but all your other phases work fine, you will receive an A- in this course and 100% on your CAs for the regular class. If your code does not work correctly, then you might lose some points for both the CAs and this class. If you decide to complete this phase, you will get an A with an opportunity to get a bonus.

Steps:

Recovery:

- You must fully implement recovery logic across all stages of the pipeline to handle mispredictions. A successful implementation ensures that the processor always uses *not-taken* prediction for all branches and jumps. When the processor detects that a misprediction has occurred, it must recover and resume execution at the correct branch or jump destination.

- Your recovery mechanism must correctly restore the ROB, RS, register file, and functional units, and it must update the PC to the proper branch/jump target. After recovery, the processor should be able to continue execution normally from that point.

Bonus: Branch prediction logic: You will receive a 10% bonus (added to your average CAs) if you implement the following design:

- Your branch prediction module includes an 8-entry BTB (designed as a fully associative structure) that stores the tag (PC), the target address, and a 2-bit BHT (i.e., a bi-modal predictor).

- Upon encountering a branch, your branch prediction, implemented at the fetch stage, should dynamically predict the outcome of the branch (so no longer always not-taken).
- Your recovery logic should be able to capture and fix the mispredictions.
- You should add a branch update logic (e.g., implemented during the complete stage) to update the BTB (i.e., inserting a new instruction, updating the BHT, updating the target address, etc.).

We may also provide an additional bonus (up to 5%) to the best/most efficient designs (with or without the branch prediction part).

What to submit

1. You need to submit your final design files (you can zip all your files and upload the zip folder) on Gradescope. This should include ALL files needed to successfully run your design.
2. You should include a final report with the following information:
    a. A full block diagram of your entire design. Please note that we will run your design separately using our own traces to verify the correctness.
    b. A brief explanation of all modules in your project.
    c. A screenshot showing the final result of your register file and the total number of cycles for the traces in this folder: newtrace. The traces can be from your simulation software (Quartus, Vivado, Modelsim, etc.). If you did not implement the recovery, indicate that in your report and do not run the "jswr" benchmark.
    d. To receive the bonus: A table showing the total number of cycles with and without branch prediction for jswr and test traces.
3. You should also update your GitHub repository and include the link and the commit number in your report.
4. A short report (a separate PDF file) that explains how each person in the group contributed to this phase.