Phase 2- Rename and with RoB

Deadline: Thu., Nov. 20, 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

Steps:

- You will be implementing the Rename as well as the Dispatch stage. In this design, you will be implementing a read-after-issue design, meaning that we will be using a (unified) physical regfile to store all our data.
- **The Rename Module:**
  The rename module takes in instructions from the Decode and fills up some other data and puts them into the dispatch stage. Hence, it contains multiple submodules as well

  a. Map Table:
     - You can implement this as a simple table. It's supposed to check what the 'new' physical regfile is (similar to what was discussed in the lecture).
     - Tips:
       1. To support speculation, you should also have a 'checkpointed' map table (a cloned map table) for recovery. Remember, we are still in order, hence when a branch instruction comes in, fill up that cloned map table in case of a misspeculation.

  b. Free List:
     - You should keep track of what physical registers are free and assign them accordingly.

- Tips:
  1. Implement this as a circular buffer. HOWEVER, this circular buffer is actually a simple tracker.
     - I.e: instead of storing values, you just need to track what range of PREG is free:
       - E.g: PREG 5 to PREG 10 are currently free.
     - It is a circular buffer, because after the last PREG, it just routes back to the start. But the buffer doesn't actually need to store anything.
  2. To support mispeculation, it needs a recovery pointer.
     - Hints: If there are non-speculative, and speculative instructions in the ROB, will the non-speculative instructions always be "behind" the speculative ones?

c. ROB Tag:
   - Each instruction should have an ROB tag associated so that it can notify the ROB when it commits. This can be implemented as a simple counter.
   - Because we are in order, you can simply use a counter for this.
   - Tips:
     1. Similarly, have a checkpoint for this for mis-speculation.

- **The Dispatch Module.**
  The Dispatch module takes in data from the rename stage and assigns it to the correct reservation station. At the same time, it fills up a slot in the ROB. The Dispatch contains multiple submodules:

  a. Pipeline Buffer / FIFO of size 1:
     - This is to hold the instruction from the rename module. Because some reservation stations can be full, and others not, this pipeline buffer holds the instruction and checks whether their reservation station is 'ready'.
       1. If it is, then in the next clock cycle, push it in
       2. Otherwise, it stalls the rename module.

  b. Reorder Buffer (ROB) Module (size of 16):
     - Tips:
       1. Because instructions arrive in order, we can fill up the reorder buffer as they arrive. Hence, we can implement the reorder buffer as a circular buffer.

2. Figure out the contents of an entry in the ROB and put it into a struct.
    - Hint: Because we are doing a read after issue, the ROB should not store any data, but it should know if an instruction is busy/in-flight.

3. For the 'ready' signal, what conditions are needed for the ROB to stall?

4. Similar to the free list above, it should have a recovery pointer.

c. 3 x Reservation Stations Modules
- One for ALU instructions (size of 8)
- One for Branch instructions (size of 8)
- One for LSU instructions (size of 8)

- Tips:
    1. Figure out the contents of each RS and put them into structs
        - Hint: each should have a valid bit to indicate if a slot has a valid instruction and a ready bit if it is ready.
        - Note: Some reservation stations can be ready while others aren't.

    2. The RS needs to know whether a slot is available to put in instructions. Use a Priority Decoder to read the valid bits, and retrieve the first available index slot.

    3. Similarly, if the execution unit is ready, we can only choose 1 instruction to issue. Use a Priority Decoder for this because it's simple.

        - If you're feeling ambitious, you can also track the 'age' of each instruction and choose the oldest. But be sure to make it area-efficient.

d. Physical Register File (PRF) Module:
- We will implement a simple physical regfile that takes in all ready instructions from all reservation stations. Consider a register file with 128 registers.

What to submit

1. You need to upload all your code to a GitHub repository and share the link on Gradescope (assuming you have already done that). Share the commit number in your report.
2. A short report (a PDF file) that explains how each person in the group contributed to this phase.
3. A short diagram showing your block diagram design (for all stages). Include this in your report!