Lab 4 Report

313553054 陳冠霖

1. Introduction

The purpose of this lab is to extend the riscv-long design from Lab 2 by implementing a Reorder Buffer (ROB) into the I2O2 pipeline, transforming it into an I2OI pipeline. Additionally, several test cases should be designed to observe the effects of out-of-order and in-order commits. Finally, benchmark tests will be conducted to compare and analyze the performance impact.

2. Design

Internal Components

1. Parameters and Registers:
    - ROB_SIZE: Fixed size of the ROB (16 entries).
    - valid: Tracks whether each ROB entry is valid.
    - pending: Indicates if an instruction in a slot is still pending execution.
    - rob_preg: Stores the physical register for each entry.
    - head, tail: Pointers for commit and allocation operations, respectively.
2. Signals:
    - is_full: Asserts when the ROB is full and cannot accept new entries.
    - is_empty: Asserts when the ROB is empty, meaning no valid entries exist.

Key Functional Blocks

1. Allocation Logic:
    - Ensures allocation is only allowed if the ROB is not full (is_full signal).
    - Writes the valid and pending flags along with the physical register to the tail slot.
    - Updates the tail pointer to point to the next slot after allocation.
2. Writeback Logic:
    - Clears the pending flag for a specific ROB slot (rob_fill_slot) when the instruction has completed execution (indicated by rob_fill_val).
3. Commit Logic:
    - Commits an instruction only if the head slot is valid and its pending flag is cleared.
    - Outputs the rob_commit_slot and rob_commit_rf_waddr for the committed instruction.
    - Clears the valid flag of the committed entry and advances the head pointer.

3. Testing Methodology

```
li   x1,1;
li   x2,2;
li   x3,3;
li   x4,4;
li   x5,5;
mul x3,x1,x2;
add x3,x4,x5;
add x6,x4,x5;
add x7,x4,x5;
add x8,x4,x5;
add x9,x4,x5;
add x10,x4,x5;
add x11,x4,x5;
add x12,x3,x5;
```

(1)

```
li   x1,1;
li   x2,2;
li   x3,3;
li   x4,4;
li   x5,5;
add x3,x4,x5;
mul x12,x3,x2;
add x6,x4,x5;
add x7,x4,x5;
add x8,x4,x5;
add x9,x4,x5;
add x10,x4,x5;
add x11,x4,x5;
```

(2)

```
li   x1,1;
li   x2,2;
li   x3,3;
li   x4,4;
li   x5,5;
add x3,x4,x5;
mul x3,x1,x2;
add x6,x4,x5;
add x7,x4,x5;
add x8,x4,x5;
add x9,x4,x5;
add x10,x4,x5;
add x11,x4,x5;
add x12,x3,x5;
```

(3)

```
li   x1,1;
li   x2,2;
li   x3,3;
li   x4,4;
li   x5,5;
mul x1,x4,x5;
add x6,x4,x5;
add x7,x4,x5;
add x8,x4,x5;
mul x2,x4,x5;
add x9,x4,x5;
add x10,x4,x5;
add x11,x4,x5;
mul x3,x4,x5;
add x12,x4,x5;
add x13,x4,x5;
add x14,x4,x5;
```

(4)

(1)A test that fails due to out-of-order commits but passes with the reorder buffer (ROB) implemented

A: if no reorder buffer, the x3 used by the last add will be the result from mul instead of add from line2

(2)A test case requiring a value to be bypassed from the ROB

A: when line 2 is using x3,the correct value is in the reorder buffer(not committed yet)

(3)A write-after-write(WAW) scenario that executes correctly on both the original and the final processor.Explain what conditions allow this to occur?

A: comparing to example(1), because this time the first instruction(add) will finish first ,so whether it's in or out-of order commit,the last add will always use the correct x3(in line 2)

(4)A scenario in which the riscvlong processor achieves a higher IPC than the completed riscvooo processor. Why might this happen?

A: this happen because structural hazard occur, for example, when line1 enters reorder buffer(wb stage), line 4's add also need to enter reorder buffer,which causes stall,therefore will have a lower IPC.

4. Evaluation

Benchmark	Processor	Num cycles	Num inst	IPC
ubmark-vvadd.c	riscvooo	581	453	0.77969
ubmark-vvadd.c	riscvlong	471	453	0.961783
ubmark-cmplx-mult.c	riscvooo	2600	1873	0.720385
ubmark-cmplx-mult.c	riscvlong	2750	1873	0.681091
ubmark-masked-filt.c	riscvooo	7447	4764	0.639721
ubmark-masked-filt.c	riscvlong	6553	4764	0.726955
ubmark-bin-search.c	riscvooo	1445	1030	0.712803
ubmark-bin-search.c	riscvlong	1415	1030	0.727915

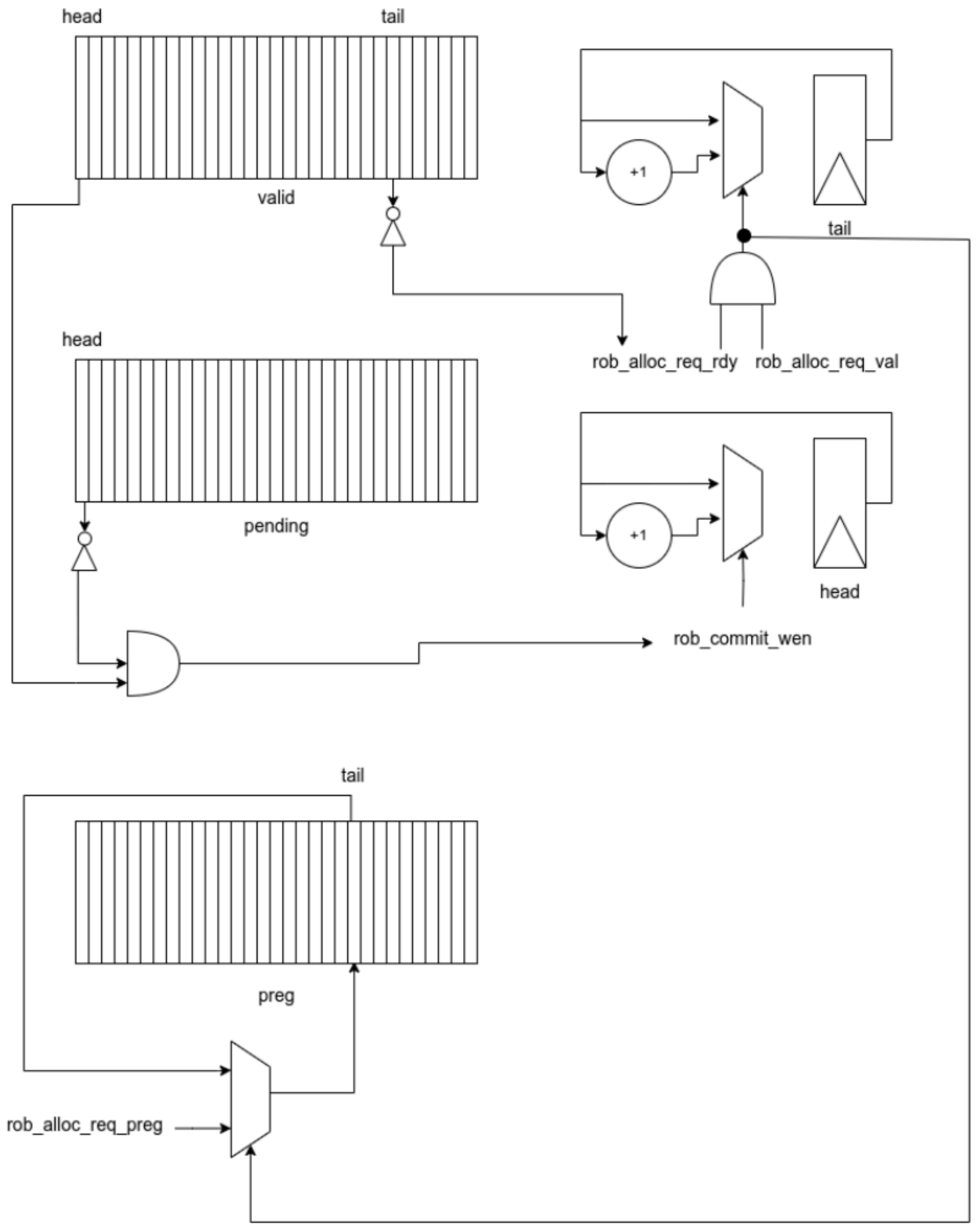
5. Discussion

Benchmark Analysis:

- Cycle Count: riscvlong generally spends fewer cycles across benchmarks, particularly excelling in simpler tasks like ubmark-vvadd.c and ubmark-masked-filt.c. Its I2O2 design avoids handling simultaneous writebacks, leading to fewer stalls but risking errors like WAW hazards.
- IPC: riscvlong achieves higher IPC in simpler workloads, while riscvooo performs better in complex tasks like ubmark-cmplx-mult.c. For benchmarks with minimal hazards, such as ubmark-bin-search.c, the IPC difference is negligible.

While riscvlong demonstrates better efficiency in these tests, its lack of stall mechanisms can lead to correctness issues. In contrast, riscvooo prioritizes execution correctness by addressing hazards, even at the cost of increased cycle count and lower IPC. This highlights the trade-off between simplicity and reliability in pipeline designs.

6. Figures


head
tail
valid
+1
tail
rob_alloc_req_rdy
rob_alloc_req_val
head
pending
+1
head
rob_commit_wen
tail
preg
rob_alloc_req_preg