Computer Architecture Report for Lab 4

資科工所 張勝迪 312551006

1. Introduction

This lab focuses on implementing and refining a Reorder Buffer (ROB) as part of an out-of-order (OoO) processor pipeline. The processor begins as an I2O2 design, with fetch, decode, and issue stages operating in order, while execute, writeback, and commit stages execute out of order. The primary objective is to convert this processor into an I2OI design by enhancing its handling of out-of-order execution and commitment, introducing functionalities like managing latencies for multiple functional units (ALU, MEM, and MUL), and ensuring correct data flow through a simplified ROB structure. Key challenges addressed include tracking instruction dependencies, resolving hazards, and implementing boundary checks for the ROB's capacity.

2. Design

a. Implementation Methodology

The implementation of the RISC-V pipeline integrates a reorder buffer (ROB) to manage out-of-order execution and in-order commit of instructions. The ROB is a circular buffer with the following primary components:

(1) Entry Structure:

- **Valid bits:** Indicates whether a slot is occupied by an instruction.
- **Pending bits:** Tracks whether the instruction's result is yet to be written back.
- **Physical registers:** Records the destination registers for instructions.

(2) Pointers:

- **Head pointer:** Points to the slot being retired.
- **Tail pointer:** Indicates the slot for the next instruction to be allocated.

(3) Control Logic:

- **Allocation:** Allocates a ROB slot for each instruction entering the pipeline if the tail slot is available (determined by valid[tail]).
- **Writeback:** Clears the pending bit when an instruction completes its execution.
- **Commit:** Retires instructions in order by checking the pending bit and valid[head].

The reorder buffer ensures in-order retirement by maintaining a strict sequence through its head pointer, allowing it to handle hazards such as write-after-read (WAR) and write-after-write (WAW).

b. Design Justification

(1) **Circular Buffer Design:** The circular buffer design simplifies slot management by using modular arithmetic for head and tail increments, minimizing logic complexity.

(2) **Pipeline Hazard Management:** The ROB effectively decouples instruction execution and commit stages. By tracking pending states, the scoreboard prevents early overwrites in the register file and ensures architectural correctness.

(3) **Scalability:** The parameterized ROB_SIZE and SLOT_BITS allow easy scaling for larger pipelines without redesigning the entire structure.

(4) **Separation of Concerns:** Allocation, writeback, and commit logic are segregated, making the design modular and easier to debug or extend.

3. Testing Methodology

Here are the code snippets for the four test cases I wrote to analyze this assignment:

(1) Due to out-of-commit, an earlier issued instruction overwrites the result of a later instruction, resulting in a WAW (write-after-write) hazard.

```
li    x2, 2
mul   x3, x2, x2
li    x3, 2
add   x0, x0, x0
add   x0, x0, x0
add   x0, x0, x0
add   x4, x2, x3
TEST_CHECK_EQ( x4, 4 )
```

(2) Before an instruction commits, its result is bypassed for use by subsequent instructions.

```
li    x2, 2
mul   x3, x2, x2
li    x5, 5
add   x4, x5, x5
TEST_CHECK_EQ( x4, 10 )
```

(3) Although a WAW hazard occurs, it does not lead to execution errors. This is because the instruction about to overwrite the target register has not yet been committed.

```
li    x2, 2
mul   x3, x2, x2
li    x3, 2
add   x4, x2, x3
TEST_CHECK_EQ( x4, 4 )
```

(4) In a scenario where only one instruction result can be written back per cycle, frequent structural hazards lead to multiple stalls in riscvooo, causing performance degradation (IPC is lower than riscvlong).

```
li    x2, 2
mul   x3, x2, x2
addi  x4, x2, 1
addi  x5, x2, 1
addi  x6, x2, 1
addi  x7, x2, 1
```

4. Evaluation

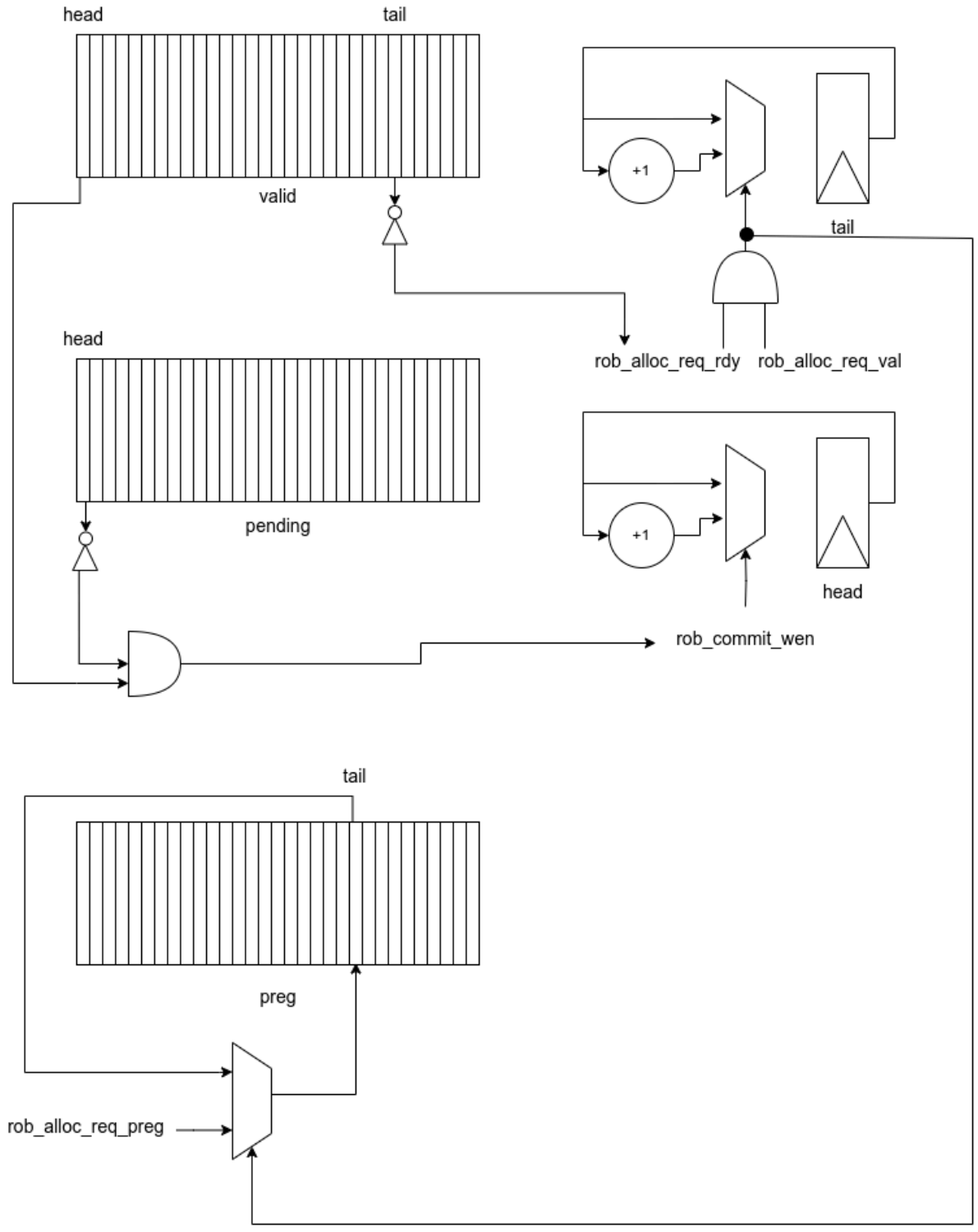
Benchmark	Processor	num_cycles	ipc
ubmark-bin-search.c	riscvlong	1415	0.727915
	riscvooo	1445	0.712803
ubmark-cmplx-mult.c	riscvlong	2750	0.681091
	riscvooo	2600	0.720385
ubmark-masked-filter.c	riscvlong	6553	0.726995
	riscvooo	7447	0.639721
ubmark-vvadd.c	riscvlong	471	0.961783
	riscvooo	581	0.779690

5. Discussion

From the evaluation results, it can be seen that riscvlong generally spends fewer cycles in the benchmarks. This is because, in the I2O2 architecture, the potential conflict of two instructions needing to write back simultaneously is not considered, and there is a lack of corresponding stall mechanisms. **Among the benchmarks, in the case of ubmark-bin-search.c, the IPC difference between the two is the smallest.** This is because the program has very few structural hazards, so the presence or absence of the wb_stall mechanism does not make a significant difference.

However, despite riscvlong's better performance in these benchmarks, the absence of the aforementioned mechanism can lead to potential program errors, such as the WAW issues observed in my testing methodology. Therefore, we can conclude that, from the perspective of the Iron Law of Processor Performance, **although riscvooo increases clock cycles and may cause a rise in cycle time due to additional checks,** leading to a decrease in IPC, this is still necessary because the correctness of program execution is the top priority.

6. Figure


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