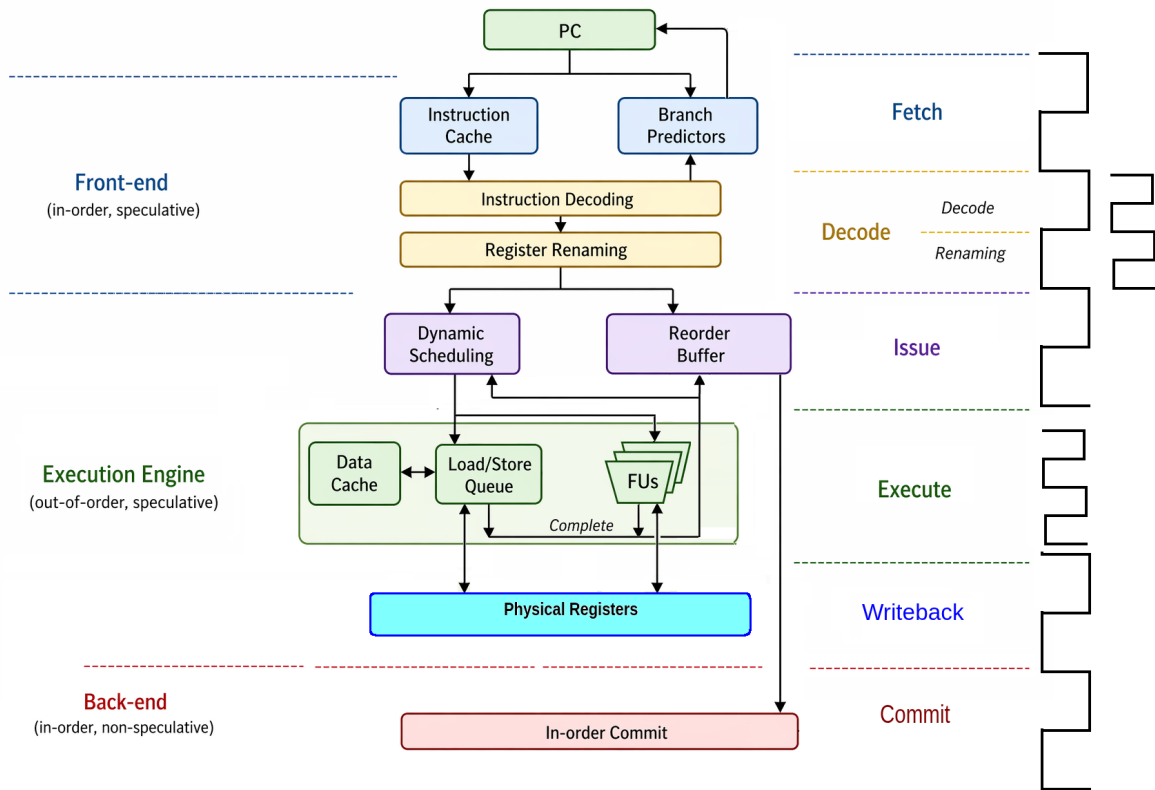


# Instruction Execution Flow in Superscalar OoO Processors

## Pipeline Stage Breakdown

### 1 Introduction

Modern superscalar processors combine multiple issue, deep pipelining, speculative execution, and dynamic scheduling. The execution process is divided into three distinct sections: the Front-End, the Execution Engine, and the Back-End.



### 2 Stage 1: Front-End (Speculative, In-Order)

The processor front-end fetches and decodes multiple instructions per cycle. It operates in a speculative, in-order manner as it uses branch prediction to determine the PC used in each cycle, but it does not reorder instructions.

After each instruction is decoded, its register operands are renamed, an ROB (reorder buffer) entry is allocated for its results, and the instruction is issued to the hardware structures for dynamic scheduling. If there is no free physical register, no free ROB entry, or no free scheduling entry, the front end stops fetching new instructions until resources are available.

Instruction bundle is the term used to describe the group of instructions that are issued in parallel within a clock cycle. In most superscalar processors the instructions in a bundle are also fetched and decoded in parallel.

1. **Fetch & Branch Prediction:** Multiple instructions, according to the superscalar width, are fetched per cycle. The processor uses branch prediction to speculate on the next Program Counter (PC).

Instr	Operation	1	2	3
I1	lw x1, 0(x10)	IF	ID	IS
I2	lw x2, 4(x10)	IF	ID	IF
I3	add x3, x1, x2		IF	ID
I4	mul x4, x3, x5		IF	ID
I5	add x6, x4, x2			IF
I6	add x3, x7, x5			IF

Figure 1: Fetch and Decode in clock cycles #1 and #2

2. **Parallel Decode:** Instructions are decoded as a "bundle."
3. **Register Renaming:** Architectural register operands are mapped to physical registers to eliminate false data dependencies.

**Architectural Registers: x0-x31,      Physical Registers: p0-p31**

**An Example** Before execution, we assume the following architectural-to-physical mapping:

$$x1 \rightarrow p1, \quad x2 \rightarrow p2, \quad x3 \rightarrow p3, \quad x10 \rightarrow p10, \quad x5 \rightarrow p5$$

(a) **Clock Cycle 1: Bundle A Instructions:**

1. lw x1, 0(x10)
2. lw x2, 4(x10)

Step	Instruction	Source Renaming	Destination Renaming
1	lw x1, 0(x10)	$x10 \rightarrow p10$	$x1 \leftarrow p20$
2	lw x2, 4(x10)	$x10 \rightarrow p10$	$x2 \leftarrow p21$

**Intra-Bundle Check:** Instruction 2 does not depend on Instruction 1 except that there is only one lw/sw *execute unit* so **I2** has to wait for relinquishing of it by **I1**.

**Renamed Bundle A:**

- lw p20, 0(p10)
- lw p21, 4(p10)

(b) **Clock Cycle 2: Bundle B Instructions:**

3. add x3, x1, x2
4. mul x4, x3, x5

Step	Instruction	Source Renaming	Destination Renaming
3	add x3, x1, x2	$x1 \rightarrow p20, x2 \rightarrow p21$	$x3 \leftarrow p22$
4	mul x4, x3, x5	$x3 \rightarrow p22, x5 \rightarrow p5$	$x4 \leftarrow p23$

**Renamed Bundle B:**

- add p22, p20, p21
- mul p23, p22, p5

(c) **Key Observations**

- **Final State**

$$x1 \rightarrow p20, \quad x2 \rightarrow p21, \quad x3 \rightarrow p22, \quad x4 \rightarrow p23$$

- **True Dependencies (RAW) Persist:** Renaming does not remove the data dependency; p22 correctly links the add and mul.
- **Elimination of Name Hazards:** While not explicitly shown here, subsequent writes to these architectural registers would be assigned new physical IDs, preventing WAW/WAR stalls.
- **Parallel Execution:** Despite the dependency between Instructions 3 and 4, the renaming logic processes both in a single cycle via hardware bypass.

### 3 Stage 2: Execution Engine (Dynamic Scheduling, Out-of-Order)

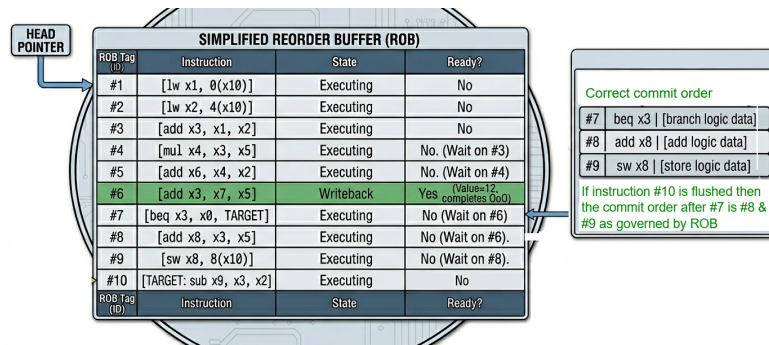
The execution engine uses dynamic scheduling to process multiple instructions per cycle speculatively and out-of-order at the speed allowed by data dependencies. An instruction is issued for execution, when all its input operands are ready, and a proper functional unit is available. When execution completes, the instruction outcomes, including any exceptions (*lw s1, 4(s0)* does not find data in cache or main memory but in virtual memory), are buffered in the ROB entry. The scheduling hardware is notified as some dependent instructions may now be ready to execute. Note that the instructions within a bundle will likely be issued and complete execution on different clock cycles. The hardware will reorder and overlap the execution of instructions from multiple bundles.

1. **Issue:** An instruction is issued to a functional unit only when:

- All input operands are ready.
- A proper functional unit is available.

2. **Execution:** Instructions from different bundles overlap and reorder to maximize throughput.

3. **Result Buffering:** The processor back-end uses the ROB to update the processor states, registers, and memory locations, in program order, and to recover from any mispredictions or exceptions. An instruction *commits* (or *retires*) and its result becomes architecturally visible only after it becomes the oldest instruction in the ROB.



If the oldest instruction led to an exception or misprediction, the ROB is used to undo any instructions past the offending ones: release any physical registers they allocated, free

their ROB entries, and resume execution from the correct PC. Some of these canceled instructions may have completed execution, while others may still be waiting for their operands.

4. **Dependency Notification:** The *dynamic scheduler* is notified that the result is ready, potentially triggering the execution of dependent instructions.

## 4 Stage 3: Back-End (Commitment, In-Order)

The back-end ensures the architectural state is updated correctly and handles recovery from speculation errors.

5. **Commit / Retire:** An instruction becomes architecturally visible (updates registers and memory) only when it is the **oldest** instruction in the ROB. This maintains program order.
6. **Exception & Misprediction Recovery:**
  - If the oldest instruction is a misprediction/exception, all subsequent instructions in the ROB are flushed.
  - Allocated physical registers and ROB entries are freed.
  - Execution resumes from the correct PC.

## 5 Summary of Pipeline Characteristics

Phase	Execution Order	Speculative?	Key Hardware
Front-End	In-Order	Yes	Branch Predictor, Renamer
Execution Engine	Out-of-Order	Yes	Issue Queues, Functional Units
Back-End	In-Order	No (at commit)	Reorder Buffer (ROB)

Table 1: Superscalar Pipeline Operation Summary