CS152/252A
Spring 2026

# Out of Order Execution

Discussion 6 Solutions

## 1 Out of Order Execution

1.1 Why is out of order execution useful?

With Out of Order Execution, we can exploit instruction-level parallelism (ILP) to keep processor busy. For example, we can schedule around long-latency instructions, such as in the following example:

```
ld x2, 0(x1)       # cache miss: 200 cycles
add x5, x3, x4
ld x7, 4(x6)
```

In this example, we can begin executing the following instructions without waiting for the first instruction to fully complete. We can also execute instructions without waiting for dependencies as in the following example:

```
lw x2 0(x1)
add x4 x2 x3     # dependent on the previous instruction
add x5 x6 x7     # no dependencies, can be issued before the previous instruction
```

1.2 Consider the following code-snippet. What limits its out of order performance, assuming there are only four architectural registers?

```
A: fmul f1, f0, f2
B: fadd f0, f3, f1
C: fmul f3, f2, f3
D: fadd f3, f3, f1
```

We want to issue instruction `C` before right after `A` because there exists a RAW hazard from A $\longrightarrow$ B and C $\longrightarrow$ D. However, there is a WAR hazard between B $\longrightarrow$ C, so we risk computing the wrong value for `f0` by reordering instruction `C` unless we add some support/detection/resolution for this

# 2 Register Renaming

Rewrite the code snippet to use infinite physical registers called P0, P1, P2, ... and update the rename table accordingly.

**Old code snippet:**

```
A: fmul f1, f0, f2
B: fadd f0, f3, f1
C: fmul f3, f2, f3
D: fadd f3, f3, f1
```

**New code snippet:**

```
A: fmul P4, P0, P2
B: fadd P5, P3, P4
C: fmul P6, P2, P3
D: fadd P7, P6, P4
```

| **Register Rename Table** | | |
|---|---|---|
| Floating Point Register | Initial Physical Register | Updated Physical Register |
| f0 | P0 | P5 |
| f1 | P1 | P4 |
| f2 | P2 | P2 (no change) |
| f3 | P3 | ~~P6~~ P7 |

# 3 Tomasulo's Algorithm

**A Brief Overview of Tomasulo's Algorithm**

On instruction dispatch (in program order):

- Allocate reservation station (RS) entry
- If source register has "present" (P) bit set in register file (RF) entry
  - Copy value into tag/data field in RS and set P bit for operand
  - Otherwise, copy tag from RF into RS and clear P bit for operand
- Replace RF entry for destination register with tag assigned to RS entry (tagdest)

Prior to execution:

- For missing operands, monitor result bus for tag match; replace tag with value; set P
- When all operands are present, issue to functional unit

On completion:

- Broadcast <tagdest, result> on result bus for RF and other RS entries to consume
- Deallocate RS entry

**Other notes for this question**:

- At most one instruction is dispatched per cycle
  - Can begin execution the same cycle as dispatch if all operands are present
- Fully-pipelined functional units
  - 2-cycle floating-point add latency
  - 3-cycle floating-point multiply latency
- Broadcasting result takes another cycle
  - Result bus can broadcast two results simultaneously

3.1 Simulate execution of the code in Q2 using Tomasulo's Algorithm. The code is copied for convenience.

```
A: fmul f1, f0, f2    # V3
B: fadd f0, f3, f1    # V4
C: fmul f3, f2, f3    # V5
D: fadd f3, f3, f1    # V6
```

**Cycle 1**

Instruction: A

**Register File**

| | p | tag/ data |
|---|---|---|
| f0 | 1 | V0 |
| f1 | 0 | T3 |
| f2 | 1 | V1 |
| f3 | 1 | V2 |

**Adder Reservation Stations**

| | p | tag/data | p | tag/data |
|---|---|---|---|---|
| T0 | | | | |
| T1 | | | | |
| T2 | | | | |

**Adder**

| Stage | Destination Tag |
|---|---|
| 1 | |
| 2 | |

**Multiplier Reservation Stations**

| | p | tag/data | p | tag/data |
|---|---|---|---|---|
| T3 | 1 | V0 | 1 | V1 |
| T4 | | | | |

**Multiplier**

| Stage | Destination Tag |
|---|---|
| 1 | T3 |
| 2 | |
| 3 | |

## Cycle 2

Instruction: B

**Register File**

| | p | tag/ data |
|---|---|---|
| f0 | 0 | T0 |
| f1 | 0 | T3 |
| f2 | 1 | V1 |
| f3 | 1 | V2 |

**Adder Reservation Stations**

| | p | tag/data | p | tag/data |
|---|---|---|---|---|
| T0 | 1 | V2 | 0 | T3 |
| T1 | | | | |
| T2 | | | | |

**Adder**

| Stage | Destination Tag |
|---|---|
| 1 | |
| 2 | |

**Multiplier Reservation Stations**

| | p | tag/data | p | tag/data |
|---|---|---|---|---|
| T3 | 1 | V0 | 1 | V1 |
| T4 | | | | |

**Multiplier**

| Stage | Destination Tag |
|---|---|
| 1 | |
| 2 | T3 |
| 3 | |

---

## Cycle 3

Instruction: C

**Register File**

| | p | tag/ data |
|---|---|---|
| f0 | 0 | T0 |
| f1 | 0 | T3 |
| f2 | 1 | V1 |
| f3 | 0 | T4 |

**Adder Reservation Stations**

| | p | tag/data | p | tag/data |
|---|---|---|---|---|
| T0 | 1 | V2 | 0 | T3 |
| T1 | | | | |
| T2 | | | | |

**Adder**

| Stage | Destination Tag |
|---|---|
| 1 | |
| 2 | |

**Multiplier Reservation Stations**

| | p | tag/data | p | tag/data |
|---|---|---|---|---|
| T3 | 1 | V0 | 1 | V1 |
| T4 | 1 | V1 | 1 | V2 |

**Multiplier**

| Stage | Destination Tag |
|---|---|
| 1 | T4 |
| 2 | |
| 3 | T3 |

---

## Cycle 4

Instruction: D

**Register File**

| | p | tag/ data |
|---|---|---|
| f0 | 0 | T0 |
| f1 | 1 | V3 |
| f2 | 1 | V1 |
| f3 | 0 | T1 |

**Adder Reservation Stations**

| | p | tag/data | p | tag/data |
|---|---|---|---|---|
| T0 | 1 | V2 | 1 | V3 |
| T1 | 0 | T4 | 1 | V3 |
| T2 | | | | |

**Adder**

| Stage | Destination Tag |
|---|---|
| 1 | T0 |
| 2 | |

**Multiplier Reservation Stations**

| | p | tag/data | p | tag/data |
|---|---|---|---|---|
| T3 | | | | |
| T4 | 1 | V1 | 1 | V2 |

**Multiplier**

| Stage | Destination Tag |
|---|---|
| 1 | |
| 2 | T4 |
| 3 | |

## Cycle 5

Instruction: N/A

**Register File**

| | p | tag/ data |
|---|---|---|
| f0 | 0 | T0 |
| f1 | 1 | V3 |
| f2 | 1 | V1 |
| f3 | 0 | T1 |

**Adder Reservation Stations**

| | p | tag/data | p | tag/data |
|---|---|---|---|---|
| T0 | 1 | V2 | 1 | V3 |
| T1 | 0 | T4 | 1 | V3 |
| T2 | | | | |

**Adder**

| Stage | Destination Tag |
|---|---|
| 1 | |
| 2 | T0 |

**Multiplier Reservation Stations**

| | p | tag/data | p | tag/data |
|---|---|---|---|---|
| T3 | | | | |
| T4 | 1 | V1 | 1 | V2 |

**Multiplier**

| Stage | Destination Tag |
|---|---|
| 1 | |
| 2 | |
| 3 | T4 |

---

## Cycle 6

Instruction: N/A

**Register File**

| | p | tag/ data |
|---|---|---|
| f0 | 1 | V4 |
| f1 | 1 | V3 |
| f2 | 1 | V1 |
| f3 | 0 | T1 |

**Adder Reservation Stations**

| | p | tag/data | p | tag/data |
|---|---|---|---|---|
| T0 | | | | |
| T1 | 1 | V5 | 1 | V3 |
| T2 | | | | |

**Adder**

| Stage | Destination Tag |
|---|---|
| 1 | T1 |
| 2 | |

**Multiplier Reservation Stations**

| | p | tag/data | p | tag/data |
|---|---|---|---|---|
| T3 | | | | |
| T4 | | | | |

**Multiplier**

| Stage | Destination Tag |
|---|---|
| 1 | |
| 2 | |
| 3 | |

---

## Cycle 7

Instruction: N/A

**Register File**

| | p | tag/ data |
|---|---|---|
| f0 | 1 | V4 |
| f1 | 1 | V3 |
| f2 | 1 | V1 |
| f3 | 0 | T1 |

**Adder Reservation Stations**

| | p | tag/data | p | tag/data |
|---|---|---|---|---|
| T0 | | | | |
| T1 | 1 | V5 | 1 | V3 |
| T2 | | | | |

**Adder**

| Stage | Destination Tag |
|---|---|
| 1 | |
| 2 | T1 |

**Multiplier Reservation Stations**

| | p | tag/data | p | tag/data |
|---|---|---|---|---|
| T3 | | | | |
| T4 | | | | |

**Multiplier**

| Stage | Destination Tag |
|---|---|
| 1 | |
| 2 | |
| 3 | |

**Cycle 8**

Instruction: N/A

**Register File**

| | p | tag/ data |
|---|---|---|
| f0 | 1 | V4 |
| f1 | 1 | V3 |
| f2 | 1 | V1 |
| f3 | 1 | V6 |

**Adder Reservation Stations**

| | p | tag/data | p | tag/data |
|---|---|---|---|---|
| T0 | | | | |
| T1 | | | | |
| T2 | | | | |

**Adder**

| Stage | Destination Tag |
|---|---|
| 1 | |
| 2 | |

**Multiplier Reservation Stations**

| | p | tag/data | p | tag/data |
|---|---|---|---|---|
| T3 | | | | |
| T4 | | | | |

**Multiplier**

| Stage | Destination Tag |
|---|---|
| 1 | |
| 2 | |
| 3 | |

3.2 Why can't the reservation station entry for an instruction be deallocated immediately on issue?

```
A: fmul f4, f0, f1    # Dispatched and issued immediately; RS is freed
B: fmul f5, f2, f3    # Allocated same RS as A before A has written back
```

`f4` and `f5` now assigned the same tag in regfile, causing instruction `A` to incorrectly clobber `f5` on writeback

3.3 Why are exceptions imprecise in this implementation?

The register file is irrevocably modified on dispatch when we replace a value with a tag. There is also no mechanism to recover this original value of a destination register if an instruction causes an exception.