

Exercise walk-through

6.3

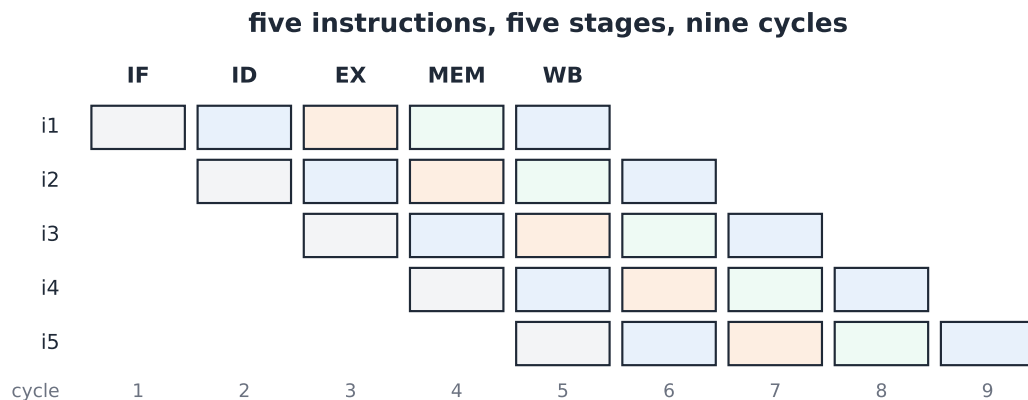
Pipelining

University Computer Science

Pipelining: $n + k - 1$ cycles

- 1 **Formula** n instructions in k stages take $n + k - 1$ cycles: five instructions take 9, not 25; at $n = 100$, 104.
- 2 **Stages** Slowest stage rules: IF 250, ID 150, EX 200, MEM 250, WB 100 - every stage 250 ps against the 900 ps single cycle.
- 3 **CPI** 1000 instructions, 25 percent loads, half load-use: $1000 + 125 \text{ stalls} + 4 \text{ fill} = 1129$ cycles, **CPI** 1.129.

Pipeline timeline



each instruction starts one cycle after its predecessor; one retires per cycle once the pipe is full

n instructions take $n + k - 1$ cycles: here $5 + 5 - 1 = 9$, not 25

Five instructions through five stages: 9 cycles total, not 25 - the pipe fills once and drains once.

Exercises 2.1-2.3

2.1 Twenty instructions on a five-stage pipeline take how many cycles? [3]

2.2 Classify: `add R1,...` followed by `sub R4, R1, R1`. [3]

2.3 What does forwarding do, and when does it fail? [3]

Exercises 2.1-2.3

2.1 Twenty instructions on a five-stage pipeline take how many cycles? [3]

2.1

- $20 + 5 - 1 = 24$ cycles ✓.
-

2.2 Classify: `add R1, ...` followed by `sub R4, R1, R1`. [3]

2.2

- A read-after-write hazard on R1 ✓.
-

2.3 What does forwarding do, and when does it fail? [3]

2.3

- Routes an ALU result straight to the next ALU's inputs ✓. It fails when the producer is a load (data exists only after MEM): one stall ✓.

Exercise 3.1

3.1 A five-stage pipeline runs a program of 2000 instructions; 30% are loads and 40% of those are load-use pairs (one stall each); taken branches are 10% of instructions and each costs one bubble; forwarding covers everything else.

Compute the stall and bubble counts, the total cycles, and the CPI. Compute the same program's cycles unpipelined (single 900 ps cycle) and the speedup of the pipelined 250 ps machine, stating what the speedup tends to as n grows. [8]

Exercise 3.1

3.1 A five-stage pipeline runs a program of 2000 instructions; 30% are loads and 40% of those are load-use pairs (one stall each); taken branches are 10% of instructions and each costs one bubble; forwarding covers everything else.

Compute the stall and bubble counts, the total cycles, and the CPI. Compute the same program's cycles unpipelined (single 900 ps cycle) and the speedup of the pipelined 250 ps machine, stating what the speedup tends to as n grows. [8]

3.1

- Loads: 600; load-use stalls: $600 \times 0.4 = 240$ ✓.
- Bubbles: $2000 \times 0.10 = 200$ ✓.
- Total cycles = $2000 + 240 + 200 + 4 = 2444$; $\text{CPI} = 2444/2000 = 1.222$ ✓.
- Unpipelined: $2000 \times 900 \text{ ps} = 1.8 \text{ ms}$; pipelined: $2444 \times 250 \text{ ps} = 611 \text{ } \mu\text{s}$ – speedup $\approx 2.94\times$ ✓.
- As n grows the fill fades and the speedup tends to $900/250 = 3.6\times$ minus the stall tax ✓.

Exercise 3.2

3.2 These instructions run in order:

```
lw    R1, 0(R2)
add   R3, R1, R4
sub   R5, R6, R7
or    R8, R3, R5
```

State which pair creates a hazard that forwarding cannot remove, and the stall it forces. State which pairs forwarding covers at no cost. Give the extra cycles this sequence adds to the ideal pipeline count, and what changes if the `lw` and `add` swap places with an independent instruction between them. [7]

Exercise 3.2

3.2 These instructions run in order:

```
lw    R1, 0(R2)
add   R3, R1, R4
sub   R5, R6, R7
or    R8, R3, R5
```

State which pair creates a hazard that forwarding cannot remove, and the stall it forces. State which pairs forwarding covers at no cost. Give the extra cycles this sequence adds to the ideal pipeline count, and what changes if the `lw` and `add` swap places with an independent instruction between them. [7]

3.2

- `lw R1` feeding `add R3` is a load-use pair: forwarding cannot remove it – one stall ✓.
- `add R3` feeding `or R8` (ALU to ALU) forwards free; `sub R5` to `or` likewise; `lw/sub` are independent ✓.
- Extra cycles: 1 (the single load-use bubble) ✓.
- An independent instruction between them fills the bubble: zero extra cycles – scheduling is the cheaper fix ✓.