

Pipelining = an implementation technique in which multiple instr. are overlapped in execution.

4 tasks

Washer / Dryer / folder / storer

4 loads

30 minutes

Sequential Laundry

$$= (0.5 \times 4) \times 4 = 8 \text{ hours}$$

$$\text{Speedup (n loads)} = \frac{(4 \times 0.5) \times n}{(4 \times 0.5) + (n-1) \times 0.5}$$

$$= \frac{2n}{2 + (n-1) \times 0.5}$$

$$\approx \frac{2n}{(n-1) \times 0.5}$$

$$\approx \textcircled{4}$$

pipelining increase Instr. throughput

It does not decrease execution
time of individual
Instr.

Pipeline stage times are limited
by the slowest resource
of the Computer.

| | C ₁ | C ₂ | C ₃ | C ₄ | C ₅ | C ₆ | C ₇ |
|----------------|----------------|----------------|----------------|----------------|----------------|----------------|----------------|
| I ₁ | IF | ID | EX | MBU | WU | | |
| I ₂ | | IF | ID | EX | MBU | WU | |
| I ₃ | | | IF | ID | EX | MBU | WU |
| I ₄ | | | | IF | ID | EX | MBU |

Structural Hazard = When a planned instruction
can not execute in the proper cycle because
hardware does not support the combination of
instructions that are set to execute.

* Suppose we have single memory insted of two
memories (Instr Memory & Data Memory)

| | C ₁ | C ₂ | C ₃ | C ₄ | C ₅ | C ₆ | C ₇ | C ₈ |
|--------------------|----------------|----------------|----------------|----------------|----------------|----------------|----------------|----------------|
| Instr ₁ | IF | ID | EX | MEM | WB | | | |
| Instr ₂ | | IF | ID | EX | MEM | WB | | |
| Instr ₃ | | | IF | ID | EX | MEM | WB | |
| Instr ₄ | | | | IF | ID | EX | MEM | WB |

In C₄ → Instr₁ access data from memory &
 Instr₄ fetch instr from memory
 ⇒ STRUCTURAL HAZARD

Separate memories ⇒ NO STRUCTURAL HAZARD

Data Hazard = When a planned instr. can not execute in the proper clock cycle because data that is needed to execute the instruction is not yet available.

Pipeline Stall / Bubble = for resolving a hazard

| | C ₁ | C ₂ | C ₃ | C ₄ | C ₅ | C ₆ |
|------------------------|----------------|----------------|----------------|----------------|----------------|----------------|
| • add \$s0, \$t0, \$t1 | IF | ID | EX | MEM | WB | |
| • sub \$t2, \$s0, \$t3 | | IF | ID | EX | MEM | WB |

* Shading = elements that is used by the instr.

Shading on right half of register file/memory
⇒ READ₁

Shading on left half of register file/memory
⇒ WRITE

CASE 1

| | C ₁ | C ₂ | C ₃ | C ₄ | C ₅ | C ₆ | C ₇ | C ₈ | C ₉ |
|----------------------|----------------|----------------|----------------|----------------|----------------|----------------|----------------|----------------|----------------|
| add \$s0, \$t0, \$t1 | IF | ID | EX | MEM | WB | | | | |
| sub \$t2, \$s0, \$t3 | | IF | ID S | S | S | ID | EX | MEM | WB |

We waste 3 cycles

of CC = 9

CASE 2 :

Assumption = Write register file in first half of CC & read register file in second half of CC.

| | C ₁ | C ₂ | C ₃ | C ₄ | C ₅ | C ₆ | C ₇ | C ₈ |
|----------------------|----------------|----------------|----------------|----------------|----------------|----------------|----------------|----------------|
| add \$s0, \$t0, \$t1 | IF | ID | EX | MEM | WB | | | |
| sub \$t2, \$s0, \$t3 | | IF | S | S | ID ↑ \$s0 | EX | MEM | WB |

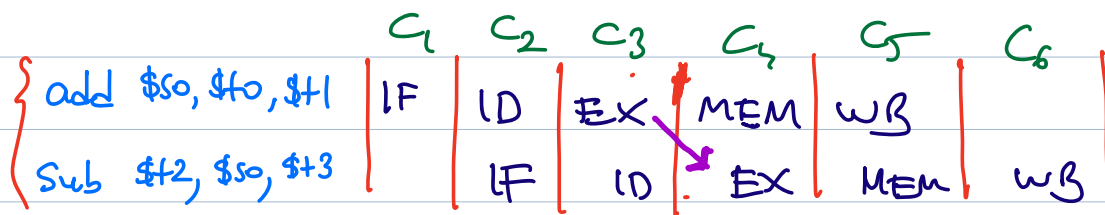
We waste 2 cycles

of CC = 8

CASE 3

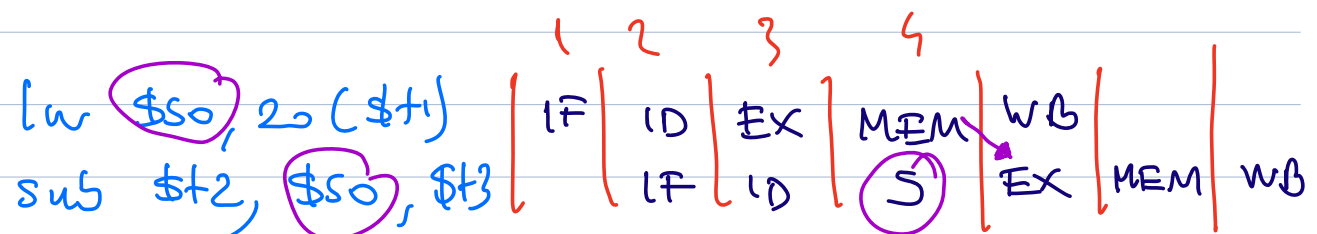
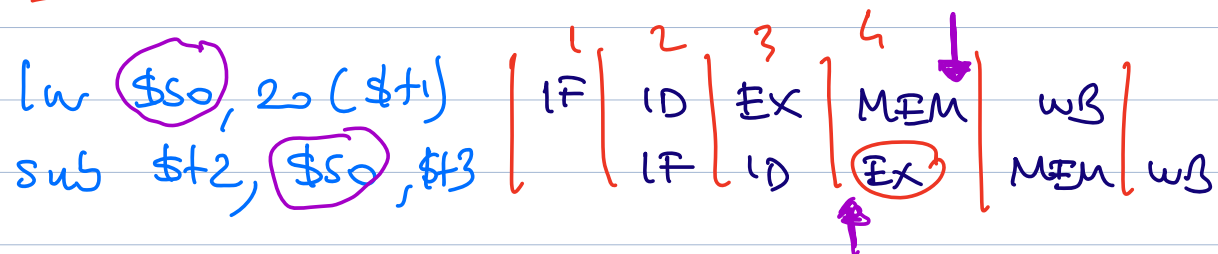
No need to wait for instr. to complete before trying to resolve the data hazard.

(Forwarding / Bypassing = Retrieve missing data element from internal buffers rather than wait for them.)



of clock cycles = 6

Load-Use Data Hazard



| | C ₁ | C ₂ | C ₃ | C ₄ | C ₅ | C ₆ | C ₇ |
|------------------------|----------------|----------------|----------------|----------------|----------------|----------------|----------------|
| lw (\$s0), 2(\$t1) | IF | ID | EX | MEM | WB | | |
| add \$t5, \$t6, \$t7 | | IF | ID | EX | MEM | WB | |
| sub \$t2, (\$s0), \$t3 | | | IF | ID | EX | MEM | WB |

Control Hazards = When the proper instr. cannot execute in the proper pipeline clock cycle because the instruction that was fetched is not the one that is needed

Solution 1: Wait until branch outcome determined before fetching the next instruction.

Assumption: We put extra hardware so that we can test registers calculate branch address & update PC during the second stage.

Branch Decision & Branch Target Address → ID stage

| | C ₁ | C ₂ | C ₃ | C ₄ | C ₅ | C ₆ | C ₇ |
|-----------------------|----------------|----------------|----------------|----------------|----------------|----------------|----------------|
| beq \$s1, \$s2, label | IF | ID | EX | MEM | WB | | |
| or \$t1, \$t2, \$t3 | | (S) | IF | ID | EX | MEM | WB |

(IF assumption
not applied)

beq \$s1, \$s2, label

or \$t1, \$t2, \$t3



Solution 2: Predict that branch will not be taken

Solution 3: Predict that branch will be taken

*** Study the examples given in the textbook.