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

4 tasks
Washer / Dryer / folder / storer

4 loads — 30 minutes

Sequential Laundry
$= (0.5 \times 4) \times 4 = 8$ hours

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

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

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

$$\approx \boxed{4}$$

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.

	C1	C2	C3	C4	C5	C6	C7	
I1	IF	ID	EX	MEM	WB			
I2		IF	ID	EX	MEM	WB		
I3			IF	ID	EX	MEM	WB	
I4				IF	ID	EX	MEM	WB

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 C_4 → Instr_1 access data from memory &
Instr_4 fetch instr from memory
⇒ STRUCTURAL HAZARD

Seperate 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
⇒ READ

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.)

	C1	C2	C3	C4	C5	C6
add $s0, $t0, $t1	IF	ID	EX	MEM	WB	
sub $t2, $s0, $t3		IF	ID	EX	MEM	WB

# of Clock cycles = 6

Load-Use Data Hazard

	1	2	3	4		
lw $s0, 20($t1)	IF	ID	EX	MEM	WB	
sub $t2, $s0, $t3		IF	ID	EX	MEM	WB

	1	2	3	4			
lw $s0, 20($t1)	IF	ID	EX	MEM	WB		
sub $t2, $s0, $t3		IF	ID	S	EX	MEM	WB

	C_1	C_2	C_3	C_4	C_5 -	C_6	C_7
lw $s0, 20($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 Addr → 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)

	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	S	IF	ID	EX	

Solution 2: Predict that branch will not be taken

Solution 3: Predict that branch will be taken

*** Study the examples given in the textbook.