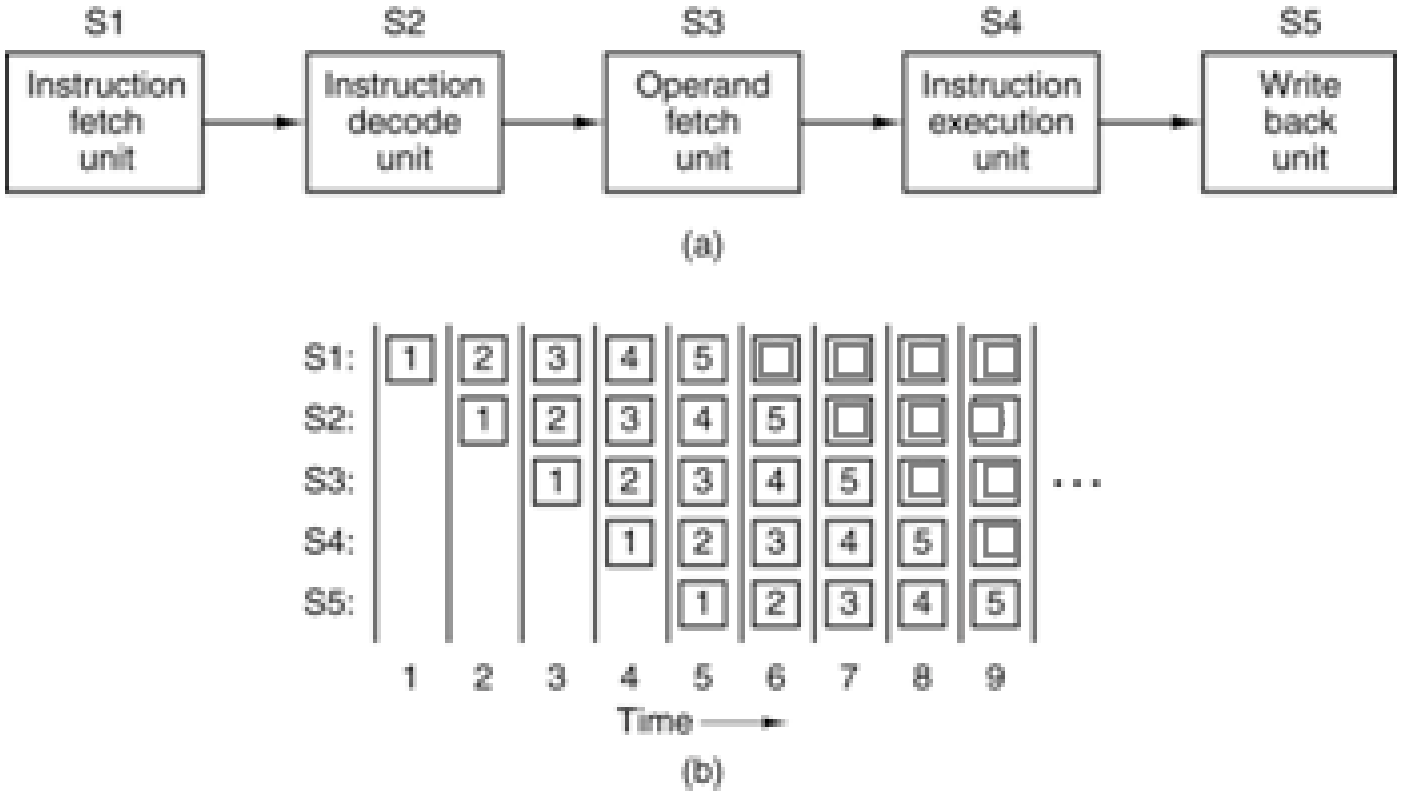
Instruction-Level Parallelism



a) A five-stage pipeline

b) The state of each stage as a function of time. Nine clock cycles are illustrated

Pipeline Hazards

- **Structural Hazards:** resource conflict
  - Example: same cache/memory for instruction and data
- **Data Hazards:** same data item being accessed/written in nearby instructions
  - Example:
    - ADD R1, R2, R3
    - SUB R4, R1, R5
- **Control Hazards:** branch instructions

Structural Hazards

- when a resource has not been duplicated enough
  - Example: same I-cache and D-cache
  - Example: single write-port for register-file
- Usual solution: **stall**
  - Also called **pipeline bubble**, or simply **bubble**

Structural Hazard

Stalling the Pipeline

	CC1	CC2	CC3	CC4	CC5	CC6	CC7	CC8	CC9	CC10
Load	IF	ID	EX	MEM	WB					
I+1		IF	ID	EX	MEM	WB				
I+2			IF	ID	EX	MEM	WB			
I+3				STALL	IF	ID	EX	MEM	WB	
I+4						IF	ID	EX	MEM	WB

Solution : Structural Hazard

Increase number of functional units that are simultaneously needed by more than one instruction

- Split Instruction/Data caches:
  - Pro: No structural hazard between IF & MEM stages
    - A single-ported unified cache stalls fetch during load or store
  - Con: Static partitioning of cache between instructions & data
    - Bad if working sets unequal: *e.g.*, $_{code}$/DATA or CODE/$_{data}$

Instruction or Control Hazard

Control Hazard

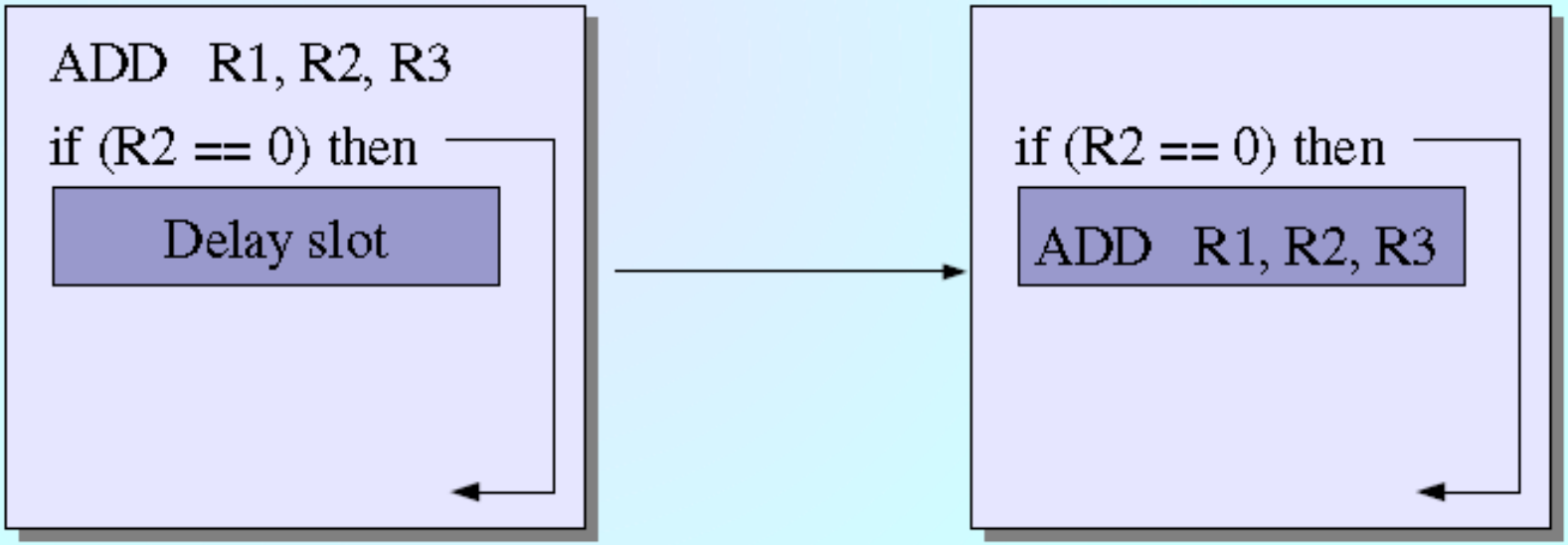
- Result of **branch instruction** not known until end of MEM stage
- Naïve solution: stall until result of branch instruction is known
  - That an instruction is a branch is known at the end of its ID cycle
  - Note: "IF" may have to be repeated

	CC1	CC2	CC3	CC4	CC5	CC6	CC7	CC8	CC9
Branch	IF	ID	EX	MEM	WB				
Branch succ		IF	STALL	STALL	IF	ID	EX	MEM	WB
Branch succ + 1						IF	ID	EX	MEM

One way to Reduce Control Hazard Delay

- **Delayed branch:**
  - Instruction(s) after branch are executed anyway!
  - Sequential successors are called *branch-delay-slots*

Filling the Delay-Slot: Option 1 of 3



- Fill the slot **from before** the branch instruction
- **Restriction:** branch must not depend on result of the filled instruction
- **Improves performance:** always

Data Hazard

Data Hazard Classification

- **Read after Write (RAW):** use data forwarding to overcome
- **Write after Write (WAW):** arises only when writes can happen in different pipeline stages

	CC1	CC2	CC3	CC4	CC5	CC6
LW R1, 0(R2)	IF	ID	EX	MEM1	MEM2	WB
ADD R1, R2, R3		IF	ID	EX	WB	

  - Has other problems as well: structural hazards
- **Write after Read (WAR):** rare

	CC1	CC2	CC3	CC4	CC5	CC6
SW 0(R1), R2	IF	ID	EX	MEM1	MEM2	WB
ADD R2, R3, R4		IF	ID	EX	WB	

Destination source format of instruction

One Data Hazard Solutions

- Compiler is made to change the order of execution of statements such that the out is not affected but the data dependency among consecutive statements is removed