LECTURE - 06

DLX Unpipelined Implementation

- Five cycles: IF, ID, EX, MEM, WB
  - Branch and store instructions: 4 cycles only
  - What is the CPI?

$$F_{branch} = 0.12, F_{store} = 0.05$$

$$CPI = 0.17 \times 4 + 0.83 \times 5 = (5 - 0.17) = 4.83$$

  - Further reduction in CPI (without pipelining)
  - ALU instructions can finish in 4 cycles too

$$F_{ALU} = 0.47 \qquad CPI = (4.83 - 0.47) = 4.36$$

$$Speedup = 4.83 \div 4.36 = 1.1$$

Some Remarks

- Any further reduction in CPI will likely increase cycle time
- Some hardware redundancies can be eliminated
  - Use ALU for (PC+4) addition also
  - Same I-cache and D-cache
- These are minor improvements...

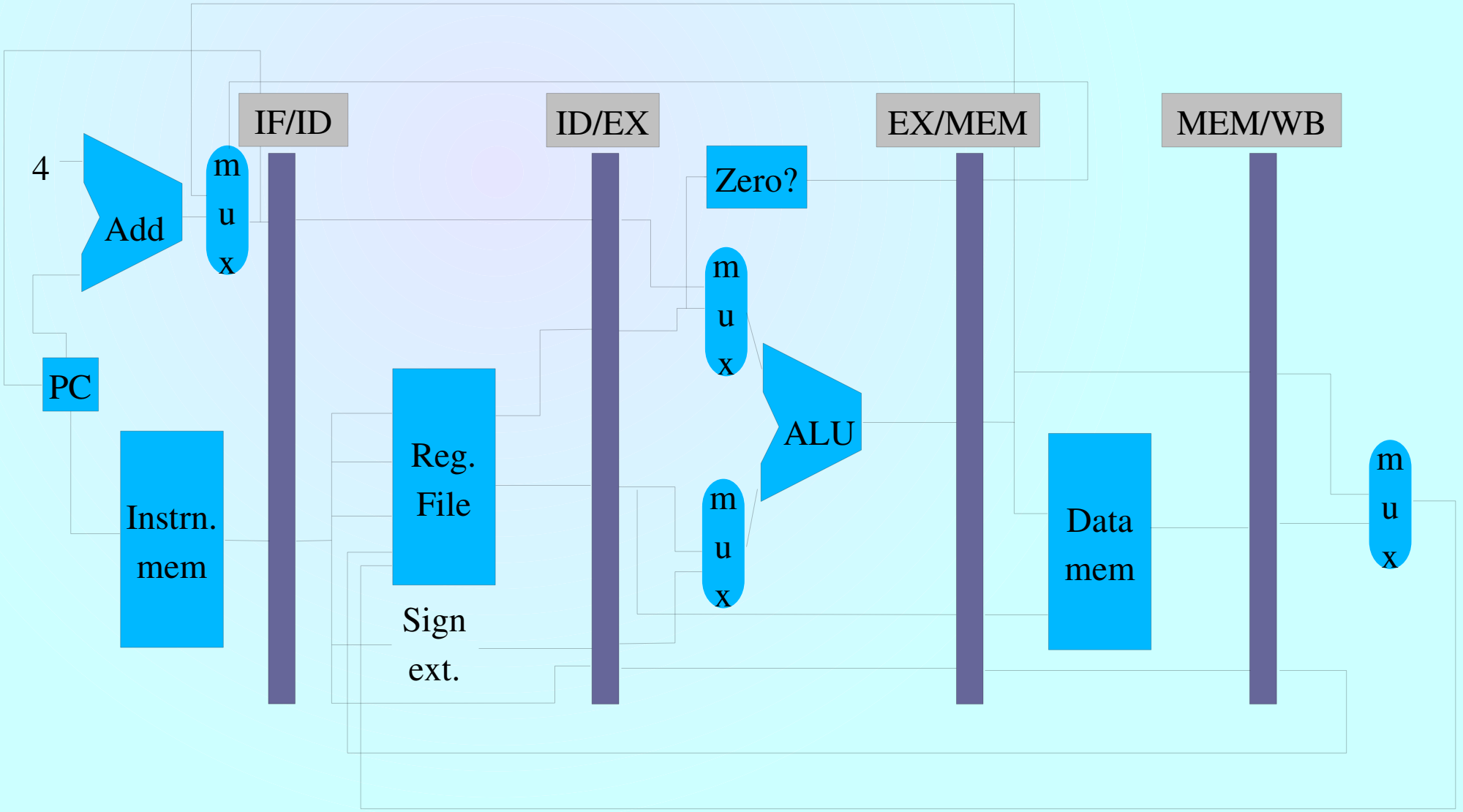
- An alternative single-cycle implementation:
  - Variation in amount of work ==> higher cycle time
  - Hardware unit reuse is not possible

The Basic Pipeline for DLX

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

- That is it?
- Complications:
  - Resource conflicts, Register conflicts, Branch instructions
  - Exceptions, Instruction set issues

The Pipelined Data-path



Pipeline registers, or *pipeline latches* to carry values from one *pipe stage* to the next

Some Performance Numerics...

$$Unpipeline\ clock\ cycle = 10\text{ns}$$

$$CPI_{ALU} = CPI_{Branch} = 4, CPI_{Other} = 5$$

$$F_{ALU} = 0.4, F_{Branch} = 0.2, F_{Other} = 0.4$$

$$Pipelined\ clock\ cycle = 11\text{ns}$$

$$Speedup = \frac{[10 \times (5 - 0.6)]}{[11 \times 1]} = 4$$

$$For\ a\ single\text{-}cycle\ impl.\ T_{IF} = 10\text{ns}, T_{ID} = 8\text{ns}$$

$$T_{EX} = 10\text{ns}, T_{MEM} = 10\text{ns}, T_{WB} = 7\text{ns}$$

$$Speedup\ from\ multi\text{-}cycle\ implementation = \frac{45}{11} = 4.09$$

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

- Usually happen when a unit is not fully pipelined
  - That unit cannot churn out one instruction per cycle
- Or, 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**

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

- What is the slowdown due to stalls caused by such load instructions?

$$CPI\ without\ stalls = 1$$

$$CPI\ with\ stalls = 1 + F_{load}$$

$$Slowdown = 1 + F_{load}$$

Why Allow Structural Hazards?

- Lower Cost:
  - Lesser hardware ==> lesser cost
- Shorter latency of unpipelined unit
  - May have other performance benefits
  - Data hazards may introduce stalls anyway!
- Suppose the FP unit is unpipelined, and the other instructions have a 5-stage pipeline. What percentage of instructions can be FP, so that the CPI does not increase?

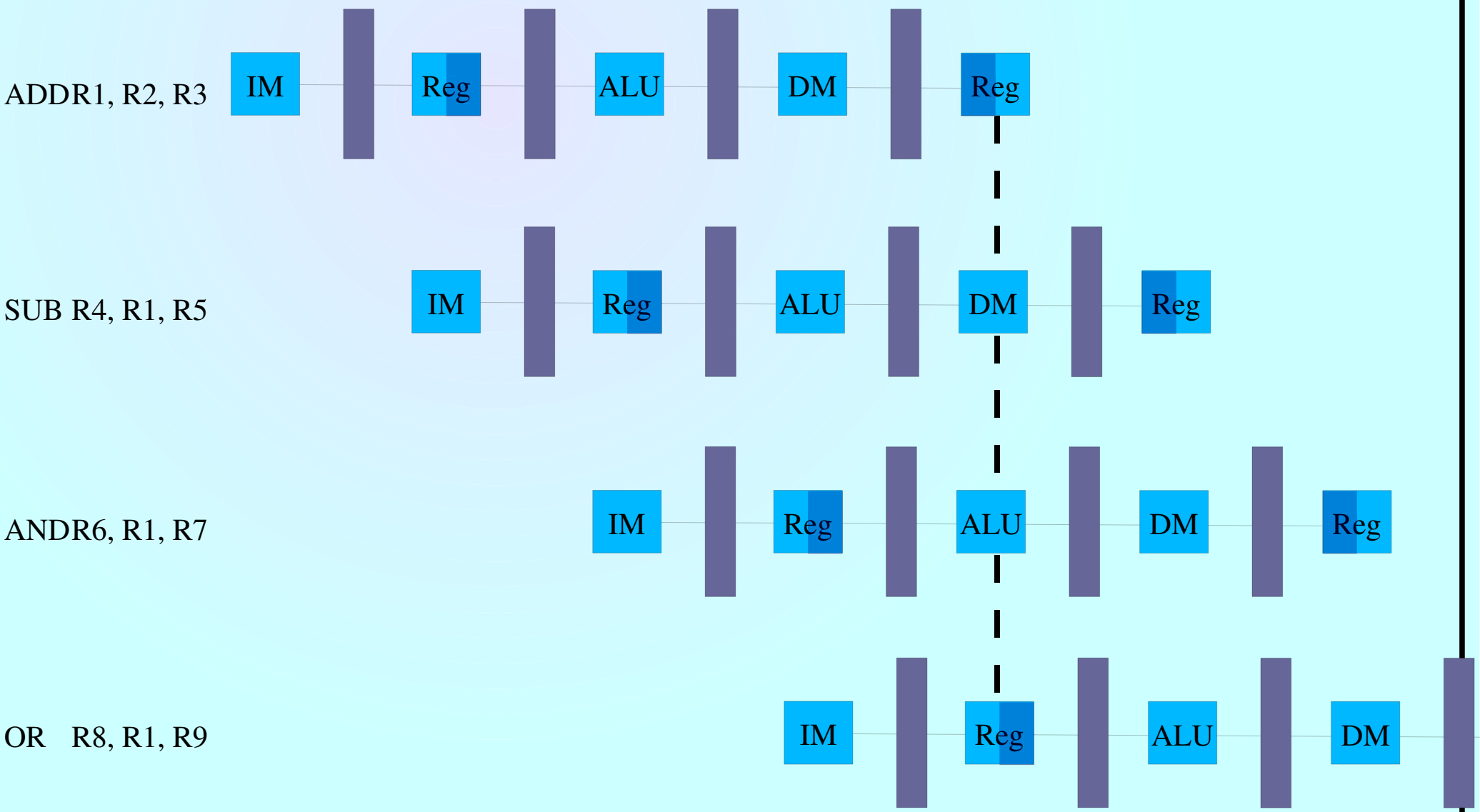
  20% can be FP, assuming no clustering of FP instructions

  Even if clustered, data hazards may introduce stalls anyway

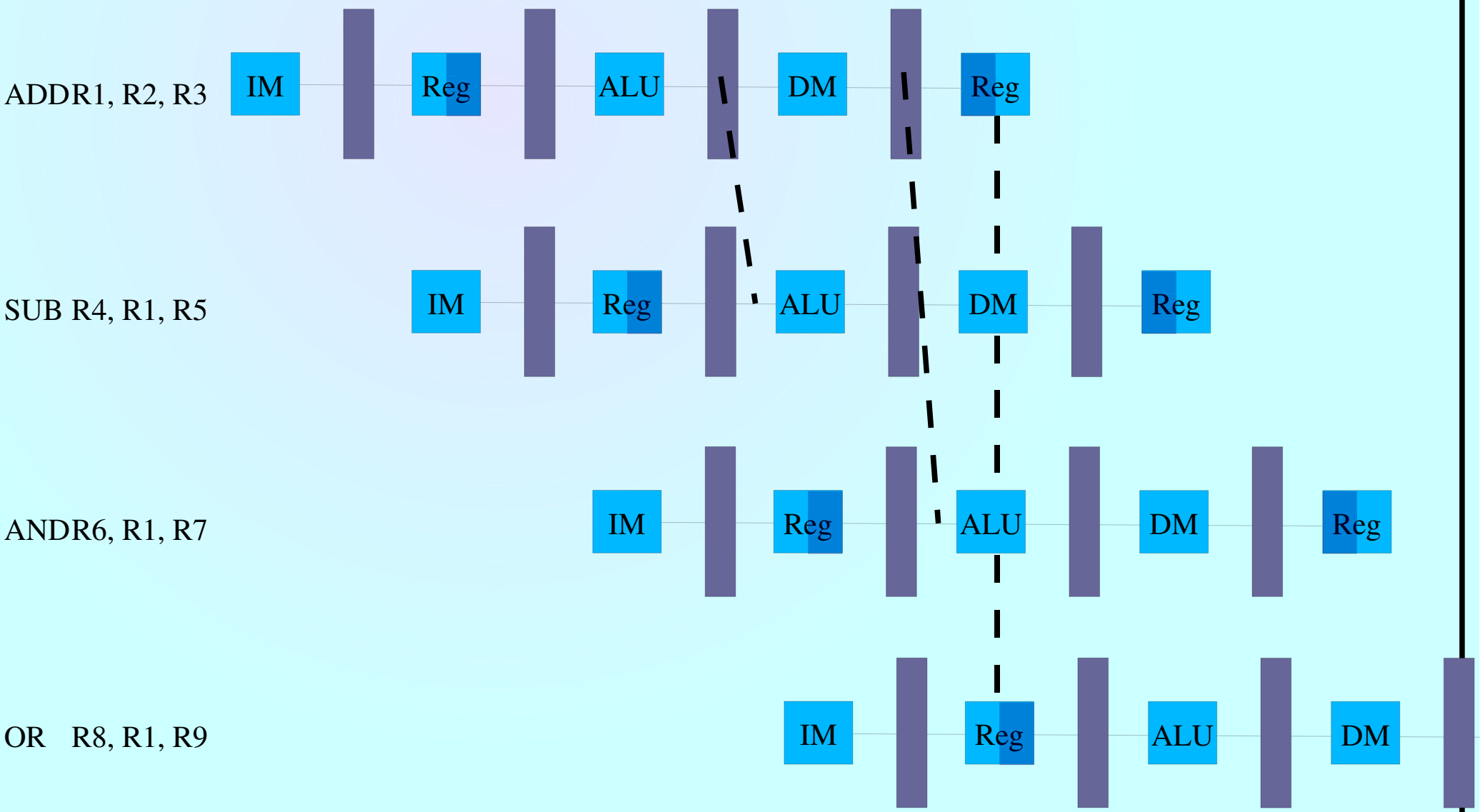
Data Hazards

- Example:
  - ADD **R1**, R2, R3
  - SUB R4, **R1**, R5
  - AND R6, **R1**, R7
  - OR R8, **R1**, R9
  - XOR R10,**R1**, R11
- All instructions after ADD depend on R1
- Stalling is a possibility
  - Can we do better?

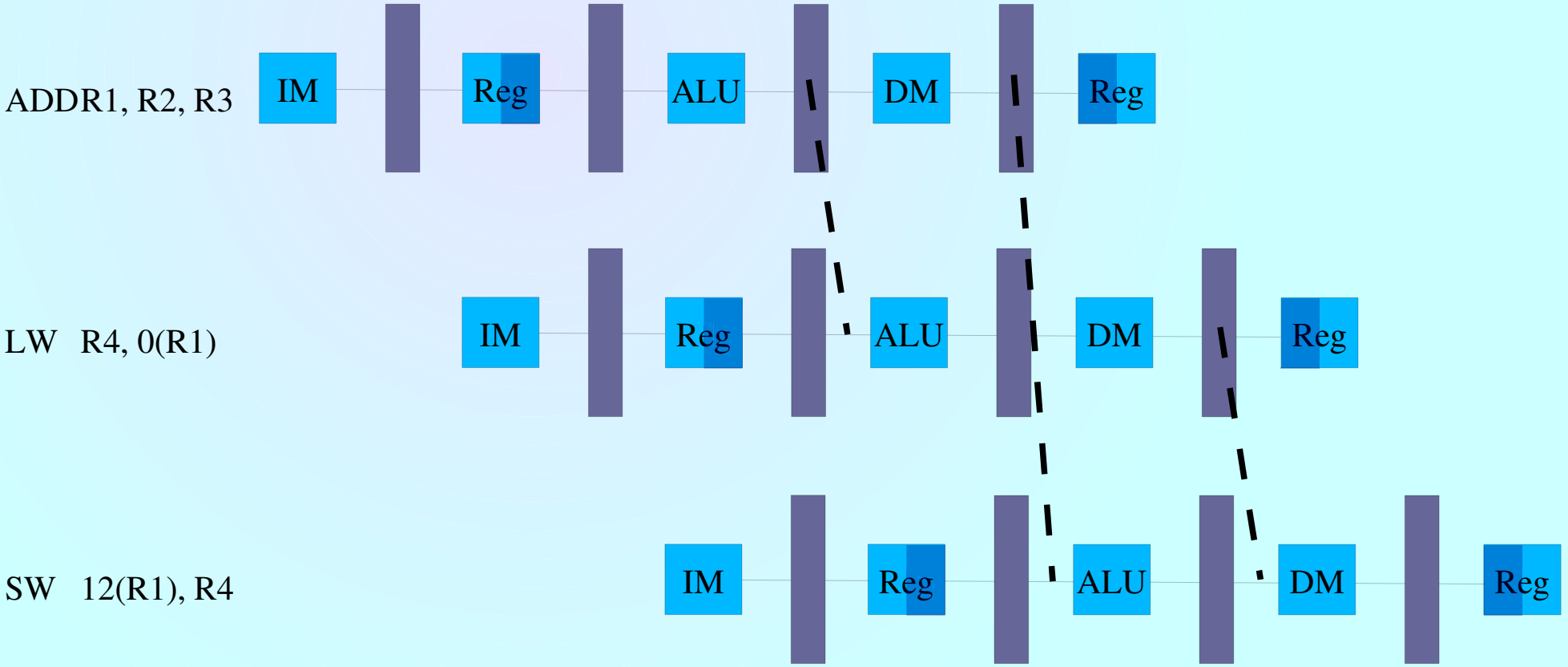
Register File: Reads after Writes



Minimizing Stalls via Forwarding



Data Forwarding for Stores



Note: no data hazards on memory locations in DLX, since memory references are always in order

Data Hazard Classification

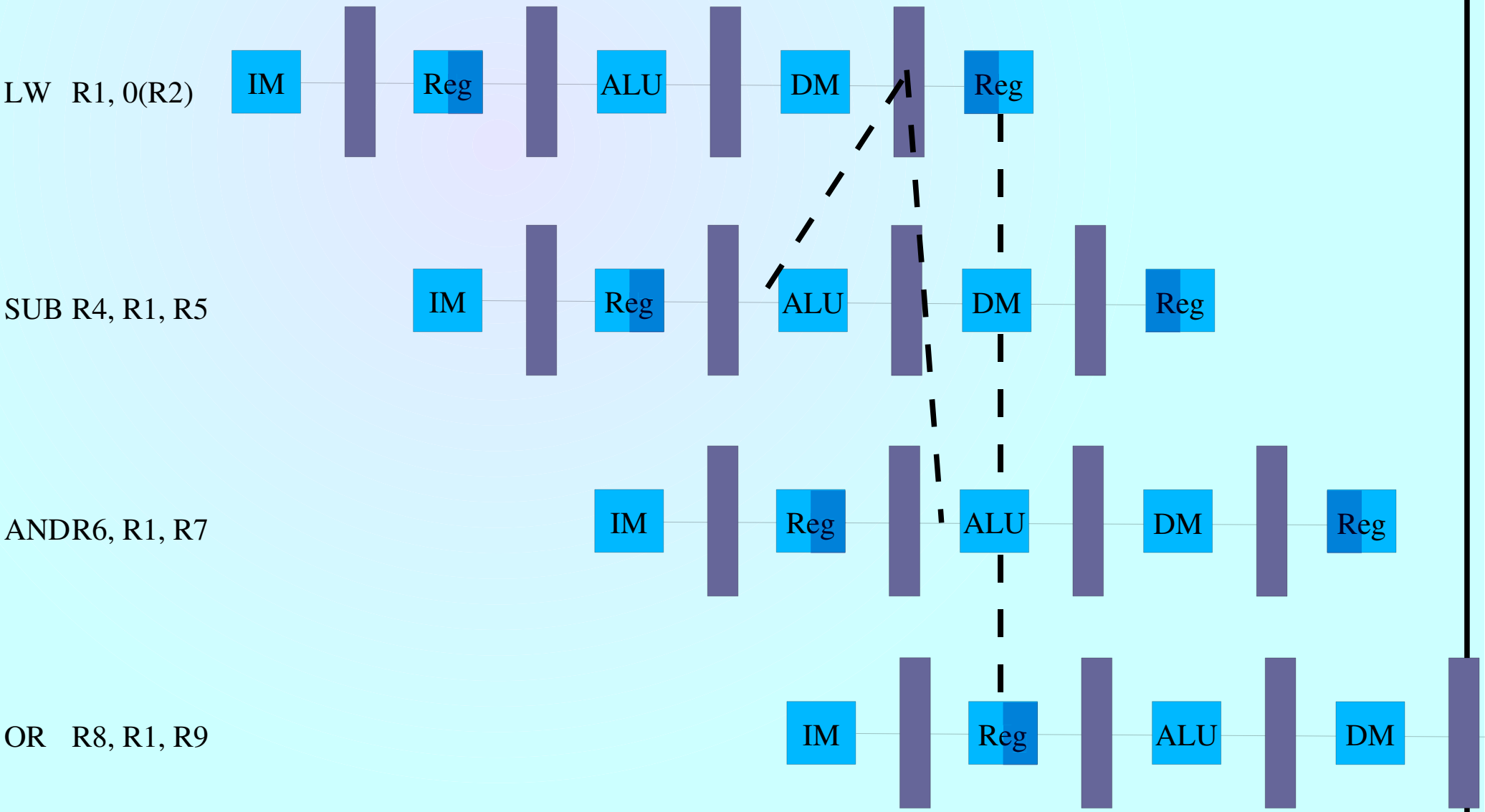
- **Read after Write (RAW):** use data forwarding to overcome
- **Write after Write (WAW):** arises only when writes can happen in different pipeline stages

	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**	

Stalls due to Data Hazard



Pipeline interlock is required: to detect hazard and stall

Avoiding such Stalls

- Compiler scheduling:
  - Example: a = b + c; d = e + f;
    - LW R1, b
    - LW R2, c
    - LW R10, e
    - ADD R4, R1, R2
    - LW R11, f
    - SW a, R4
    - ADD R12, R10, R11
    - SW d, R12

Without such scheduling, what is the slow-down?

$$1 + F_{\text{loads causing stalls}}$$

Topics for Next Lecture

- Control hazards
- Exceptions during a pipeline
  - More difficult to deal with
  - Cause more damage