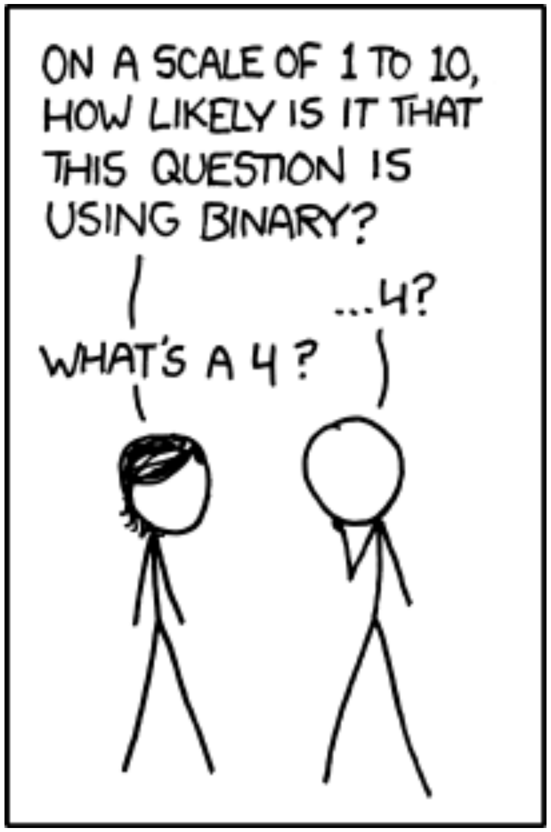
ECSE 425 Lecture 7: Pipeline Hazards

H&P Appendix A

Before We Get Started


ON A SCALE OF 1 TO 10, HOW LIKELY IS IT THAT THIS QUESTION IS USING BINARY?
WHAT'S A 4?
...4?

[Credit: xkcd.com]

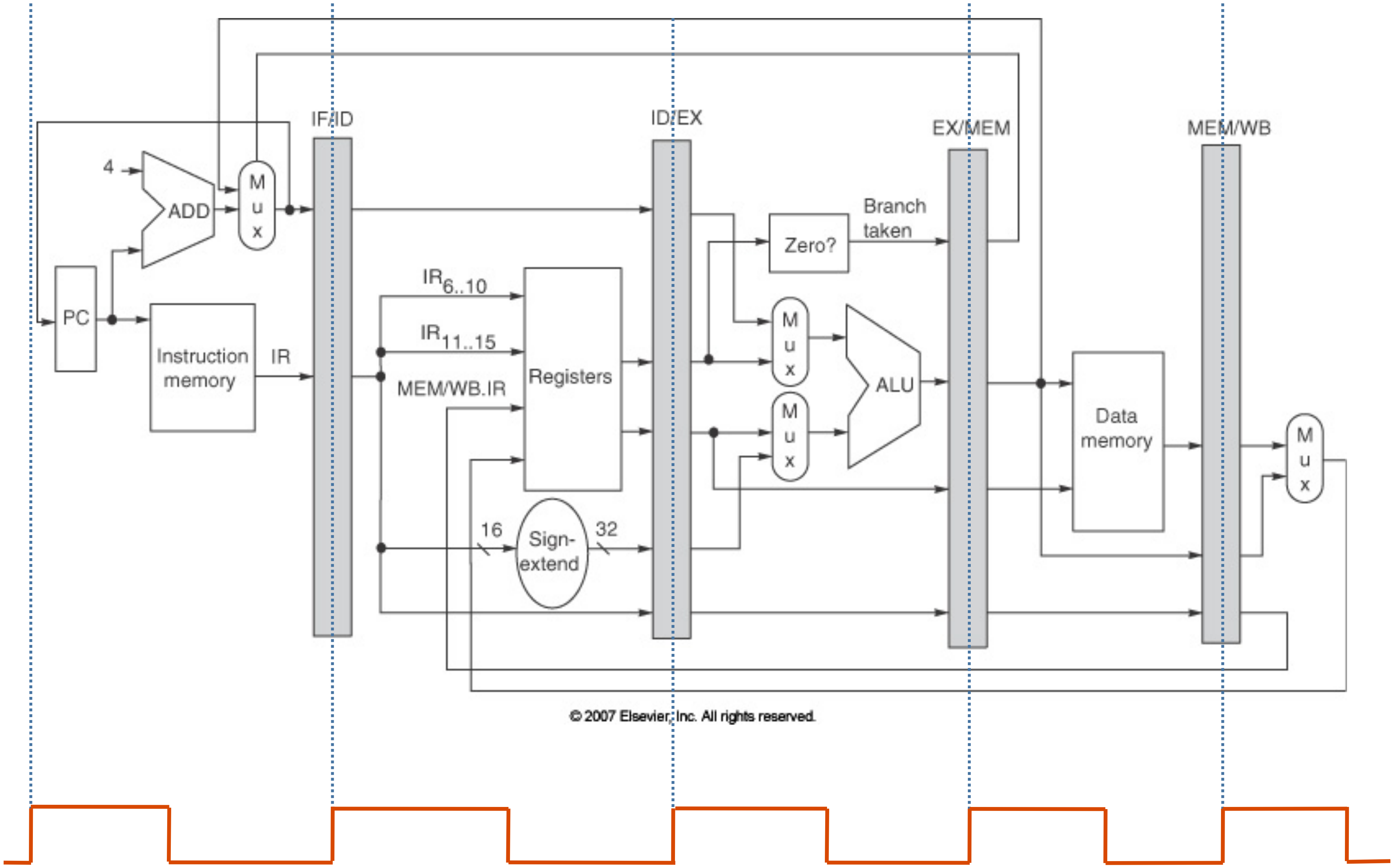
Last Time

- Ideal pipelining
  - *n* stages increases throughput by *n* times!
- RISC Instruction Set
- Unpipelined RISC implementation

Today

- Homework 1 is due!
- Homework 2 is out, due 10/3
- Pipeline Performance Issues
- Pipeline Hazards
- Looking ahead …
  - Next week, ILP
  - Start reading Chapter 2!

Basic MIPS RISC Pipeline



Pipeline Performance Issues

- Ideal pipelining: *n* stages ⇒ *n* times more throughput
- Realistic pipelining: instruction latency increases
  - Pipeline registers: set up time and propagation delay
  - Clock skew: delay between clock arriving at two different registers
- Pipeline length limited by overhead, imbalance
  - If CC = skew + register delay ⇒ no useful time
  - Work can't be divided evenly among all stages

Pipeline Performance Example

- Unpipelined processor:
  - 1 ns clock cycle
  - 4 cycles for ALU operations (40%) and branches (20%)
  - 5 cycles for memory operations (40%)
- Pipelined processor:
  - 0.2 ns overhead due to skew and setup time
- How much speedup in instruction execution rate?

Pipeline Hazards

- The relationship of instructions nearby in the pipeline can result in *hazards*
  - Hazards increase instruction latency
  - Hazards reduce instruction throughput
- Structural Hazards
  - Contention for a resource (e.g., memory, register file)
  - Pipelining stalls while instructions take turns
- Data Hazards
  - Instruction A consumes the result of instruction B
  - Pipelining stalls while A waits for B to finish
- Branch Hazards
  - Pipelining stalls when the result while branches resolve

Performance of Pipelines with Stalls

- Assume
  - No overheads (ignored)
  - Perfectly balanced design
  - Instructions with identical numbers of stages
- Performance with hazards:

$$\text{SpeedUpPipe} = \frac{\text{AverageInstrTimeNoPipe}}{\text{AverageInstrTimePipe}} = \frac{\text{CPI\_NoPipe}}{\text{CPI\_Pipe}} \times \frac{\text{CC\_NoPipe}}{\text{CC\_Pipe}}.$$

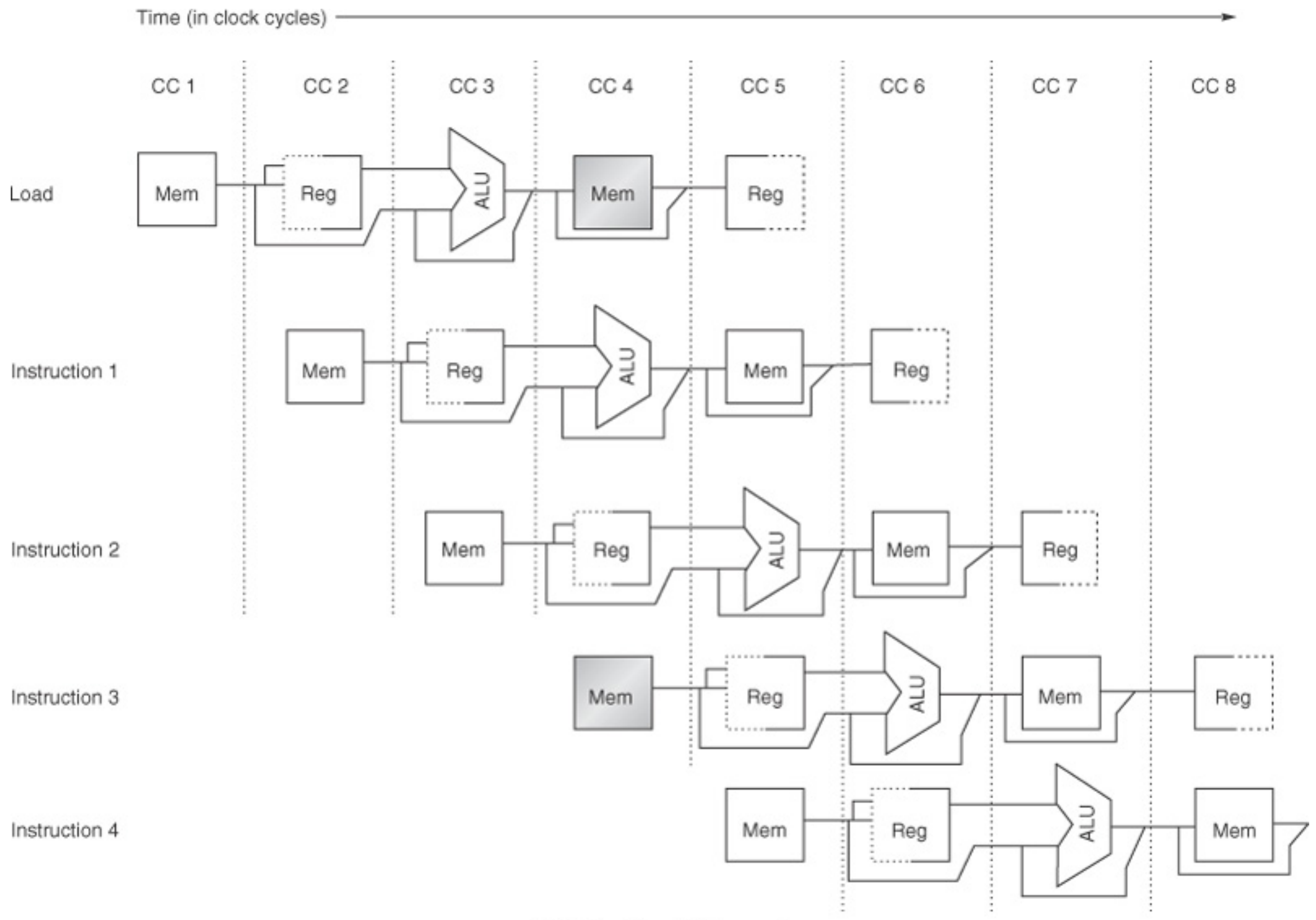
But, $\text{CPI\_Pipe} = \text{CPI\_Ideal} + \text{StallsPerInstr} = 1 + \text{StallsPerInstr},$

Also, $\frac{\text{CC\_NoPipe}}{\text{CC\_Pipe}} = 1,$ and $\text{CPI\_NoPipe} = \text{PipeDepth}.$

So $\text{SpeedUpPipe} = \frac{\text{PipeDepth}}{1 + \text{StallsPerInstr}}$

Structural Hazards

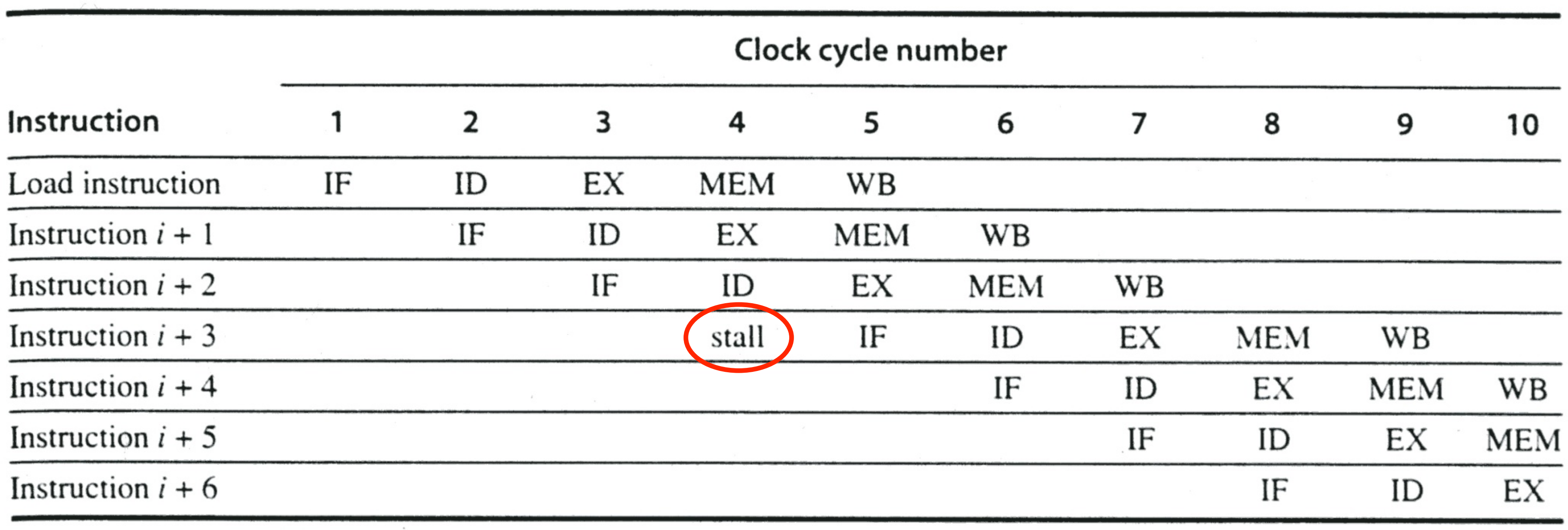
- Occur when FUs aren't pipelined or sufficiently replicated
- Example: One memory—MEM stage of load inst. stall IF



© 2007 Elsevier, Inc. All rights reserved.

Structural Hazards: IF and Load Inst.

- Can’t load data and fetch instructions at the same time
- Solve the problem by
  - Adding ports to a single memory—expensive!
  - More typical: independent instruction and data caches
    - Not free, but there are other benefits, too!



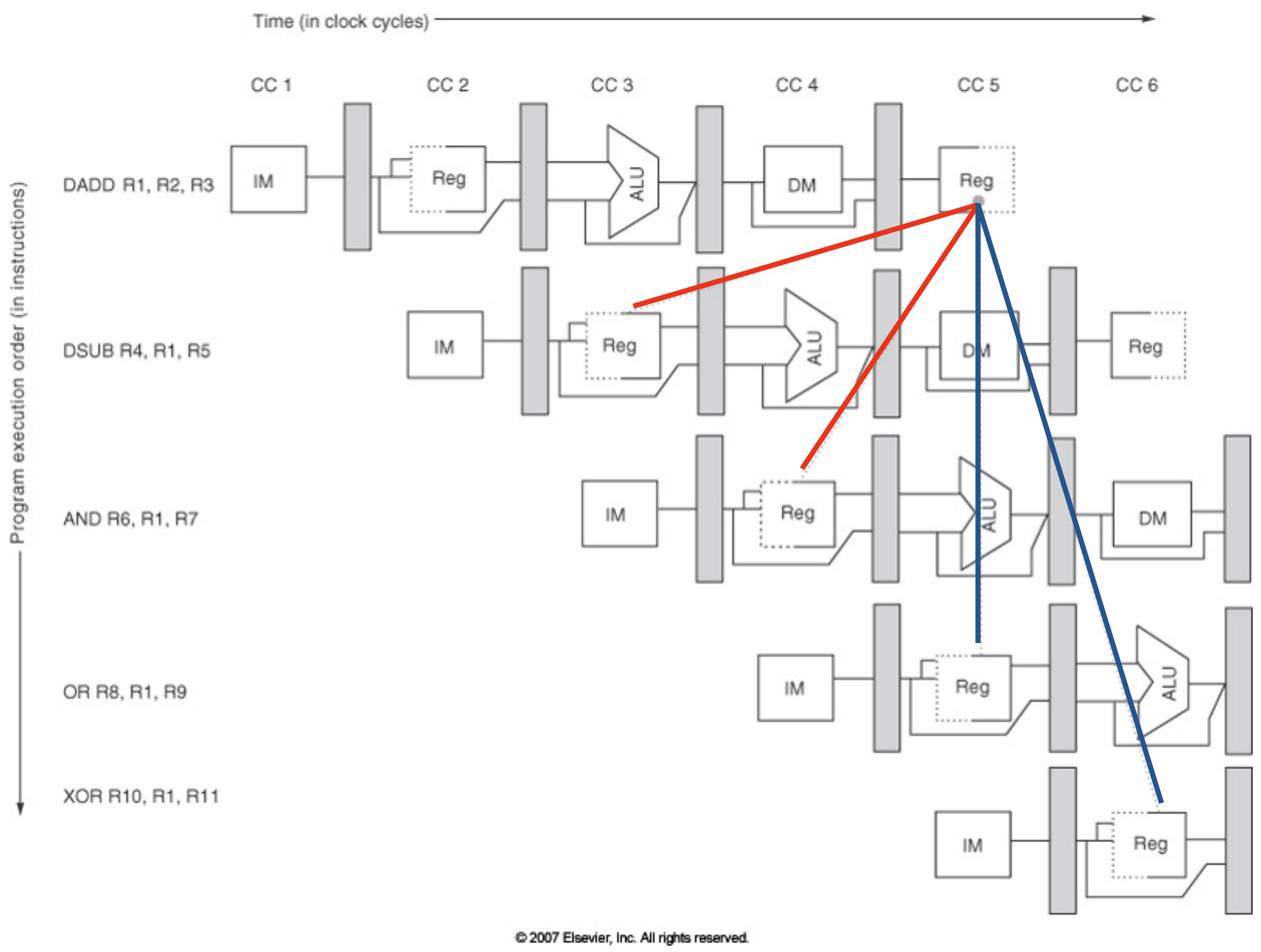
	Clock cycle number									
Instruction	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
Load instruction	IF	ID	EX	MEM	WB					
Instruction $i + 1$		IF	ID	EX	MEM	WB				
Instruction $i + 2$			IF	ID	EX	MEM	WB			
Instruction $i + 3$				stall	IF	ID	EX	MEM	WB	
Instruction $i + 4$						IF	ID	EX	MEM	WB
Instruction $i + 5$							IF	ID	EX	MEM
Instruction $i + 6$								IF	ID	EX

Structural Hazards Example

- Cost of load structural hazard – example:
  - Data references: 40% of instructions
  - CPI_{Ideal} of pipelined processor (no hazards) = 1
  - Machine A: processor with no structural hazard
  - Machine B: processor with structural hazard
  - The clock frequency of machine A, $f_A = 1.05\, f_B$
- Ignoring any other performance losses, which machine is faster?

Data Hazards

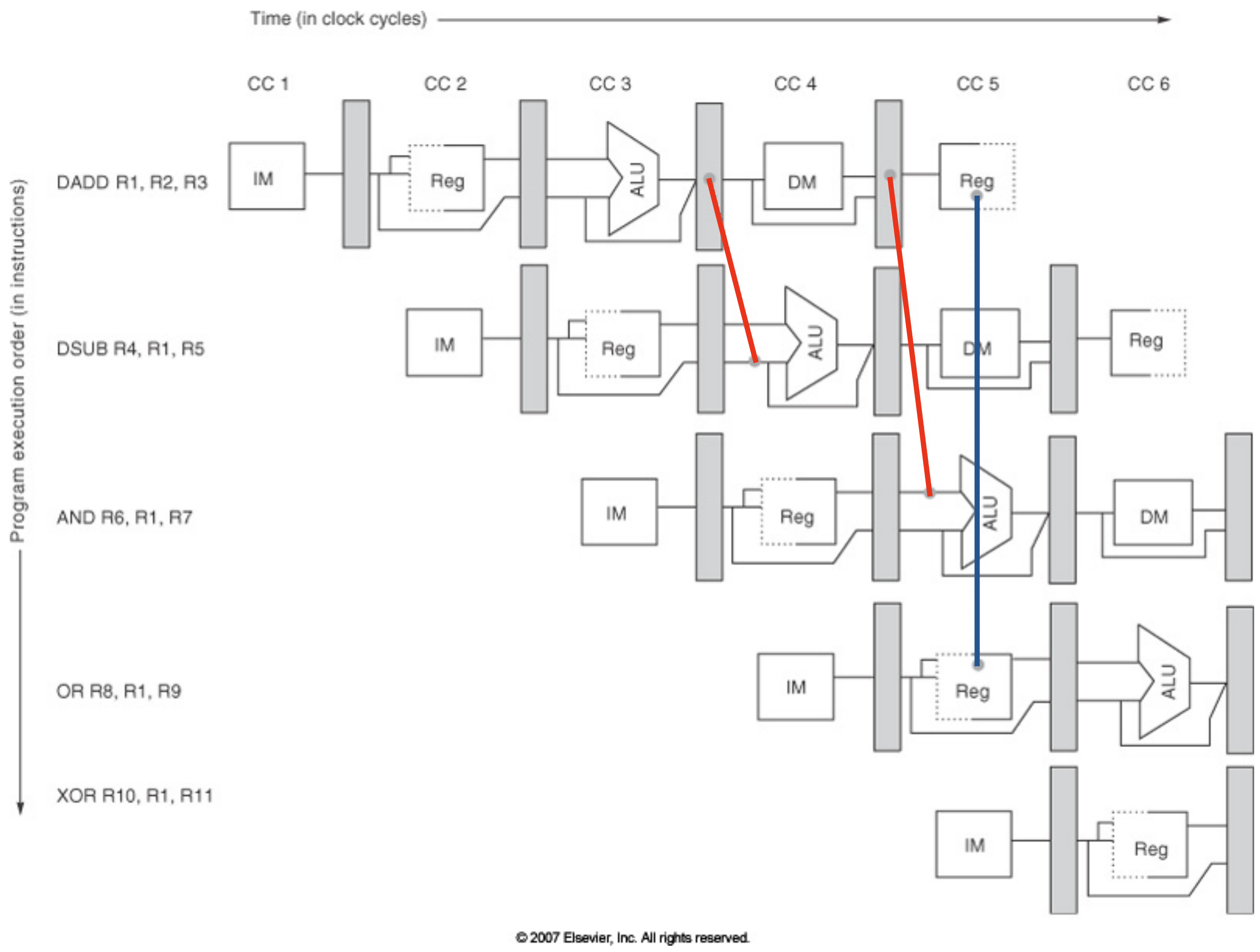
- Occur when instructions need a result before it is ready
- Example: result of DADD (R1) is consumed by four later instructions



DADD	R1, R2, R3
DSUB	R4, **R1**, R5
AND	R6, **R1**, R7
OR	R8, **R1**, R9
XOR	R10, **R1**, R11

Data Hazards: Forwarding

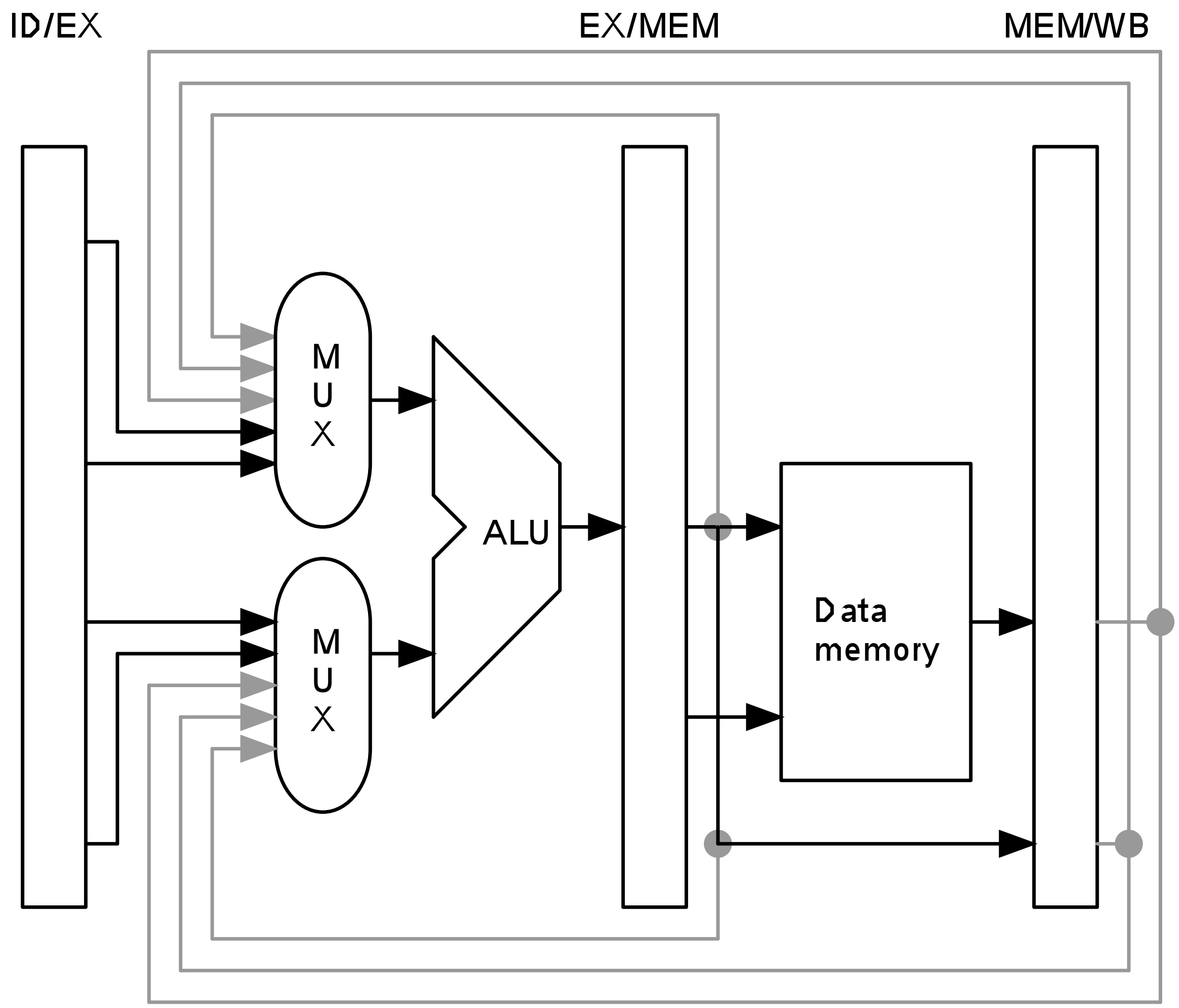
- The result of DADD is ready at the end of CC3
  - With additional circuitry and control, no stalls!



This is *forwarding*, or *bypassing*

Ex., control logic selects ALU inputs from pipeline registers
- ID/EX
- EX/MEM
- MEM/WB

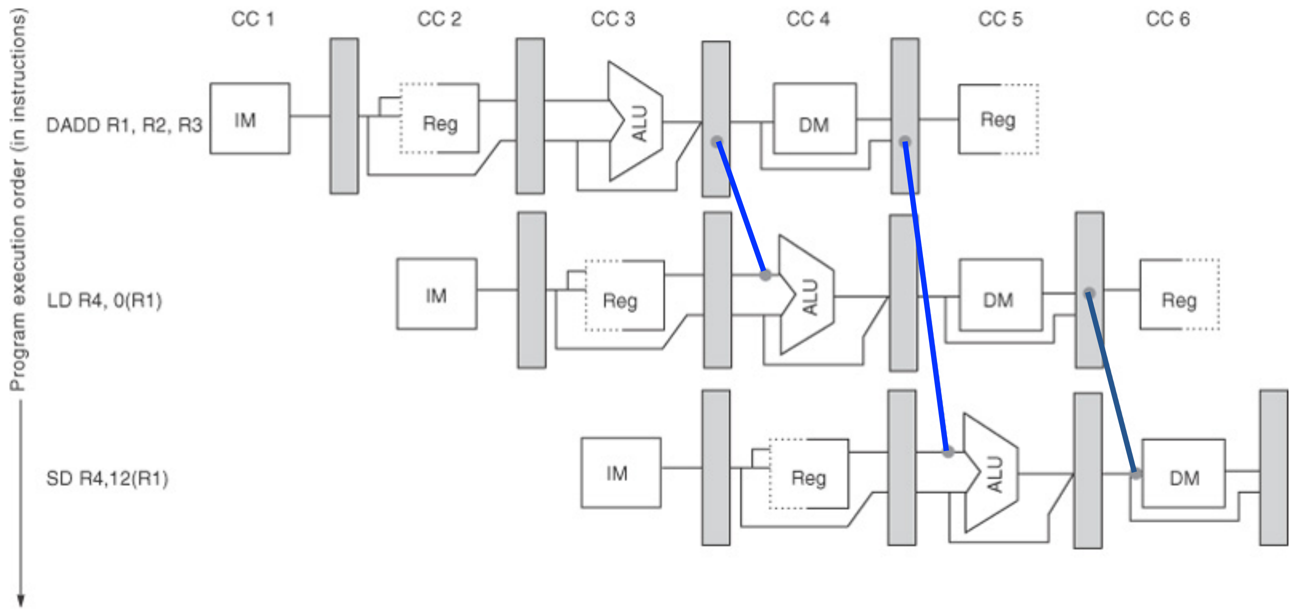
Forwarding Logic



Requirements for Forwarding

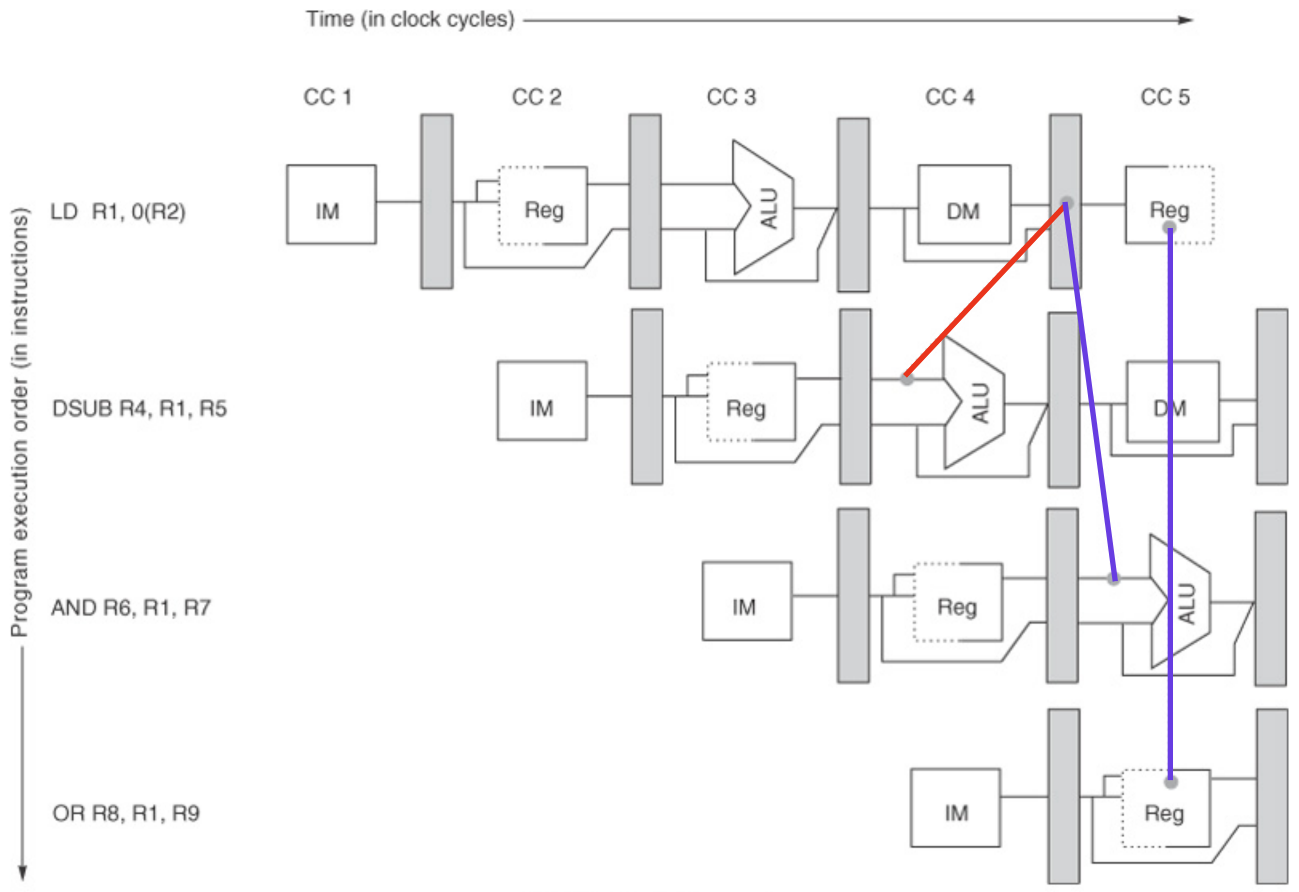
- Distance between the producer consumer is smaller than the distance between corresponding fetches
- Forward directly from output of a pipeline register to the input of another unit

DADD	**R1**,R2,R3	// *i*,	prod. **EX** pos. 3;
LD	**R4**,0(**R1**)	// *i*+1,	cons. **EX** pos. 3, (3–3<1) OK; prod. **MEM** pos. 4
SD	**R4**,12(**R1**)	// *i*+2,	cons. **EX** pos. 3, (3–3<2) OK; cons. **MEM** pos. 4; (4-4<1) OK

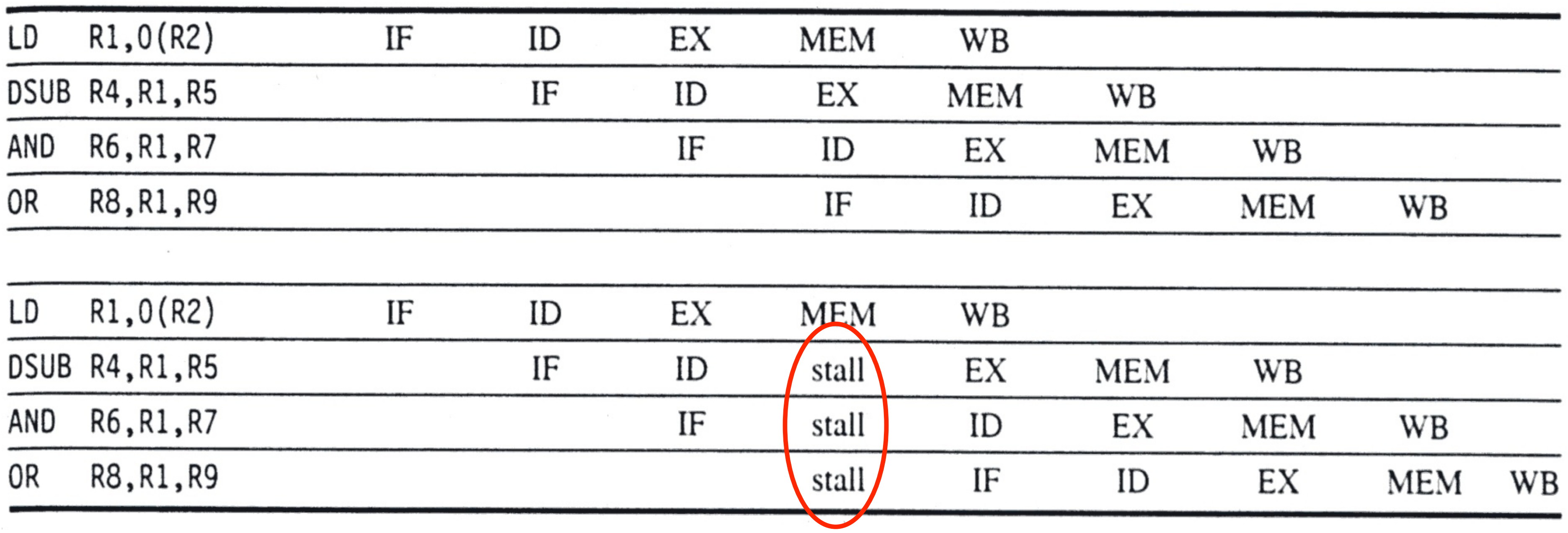


Sometimes Stalls Are Required

```
LD      R1,0(R2)  // i,     prod. MEM pos. 4
DSUB    R4,R1,R5  // i+1,   cons. EX pos. 3; ¬(4-3<1) STALL needed
AND     R6,R1,R7  // i+2,   cons. EX pos. 3; (4-3<2) OK
OR      R8,R1,R9  // i+3,   cons. EX pos. 3; (4-3<3) OK
```



Stalling One Stalls All



LD	R1,0(R2)	IF	ID	EX	MEM	WB			
DSUB	R4,R1,R5		IF	ID	EX	MEM	WB		
AND	R6,R1,R7			IF	ID	EX	MEM	WB	
OR	R8,R1,R9				IF	ID	EX	MEM	WB

LD	R1,0(R2)	IF	ID	EX	MEM	WB				
DSUB	R4,R1,R5		IF	ID	stall	EX	MEM	WB		
AND	R6,R1,R7			IF	stall	ID	EX	MEM	WB	
OR	R8,R1,R9				stall	IF	ID	EX	MEM	WB

- A stalled instruction occupies a stage and prevents all following instructions from proceeding
- A stall is implemented by interlocking pipeline stages
- No forwarding req'd for AND in the stalled pipeline

Static Scheduling for Stall Avoidance

- Re-order instructions at compile-time to avoid stalls
- For example: **a = b + c**
  d = e – f

Raw compilation		Scheduled code	
LW	Rb,…	LW	Rb,…
LW	Rc,…	LW	Rc,…
Stall		*LW*	*Re,…*
DADD	Ra,Rb,Rc	DADD	Ra,Rb,Rc
SW	Ra,…	*LW*	*Rf,…*
LW	Re,…	*SW*	*Ra,…*
LW	Rf,…	DSUB	Rd,Re,Rf
Stall		SW	Rd,…
DSUB	Rd,Re,Rf		
SW	Rd,…		

Summary

- Limits of pipeline performance
  - Overhead and “balance” limits pipeline depth
- Pipeline hazards
  - Structural: not enough resources
  - Data: results needed as inputs aren’t ready
- Hazard mitigation
  - Structural: duplicate resources
  - Data
    - Forward results
    - Static code scheduling
- But sometimes stalls can’t be avoided!

Next Time

- Hazards
  - Branch Hazards
- Implementing Pipelining
  - Pipeline Control
  - Managing Branches