

# Pipeline Hazards

## A.1 What is Pipelining?

## A.2 The Major Hurdle of Pipelining-

- Structural Hazards
- Data Hazards
- Control Hazards

## A.3 How is Pipelining Implemented

## A.4 What Makes Pipelining Hard to Implement?

## A.5 Extending the MIPS Pipeline to Handle Multi-cycle Operations

**Limits to pipelining: Hazards prevent next instruction from executing during its designated clock cycle**

- **Structural hazards**: Multiple instructions use a single HW resource.
- **Data hazards**: Instruction depends on result of prior instruction still in the pipeline
- **Control hazards**: Pipelining of branches & other instructions that change the PC
- Common solution is to **stall** the pipeline until the hazard is resolved, inserting one or more “**bubbles**” in the pipeline

# Pipeline Hazards

## Definition

- **Conditions (or situations in pipelined architecture) that lead to incorrect behavior if not fixed**
  - Structural hazard
    - **two different instructions use same h/w in same cycle**
  - Data hazard
    - **two different instructions use same storage**
    - **must appear as if the instructions execute in correct order**
  - Control hazard
    - **one instruction affects which instruction is next**

## Resolution

- **Pipeline interlock logic detects hazards and fixes them**
- **simple solution: stall**
  - increases CPI, decreases performance
- **better solution: partial stall**
  - some instruction stall, others proceed better to stall early than late
- **Best solution: remove hazard sources**

# An example: Structural Hazards

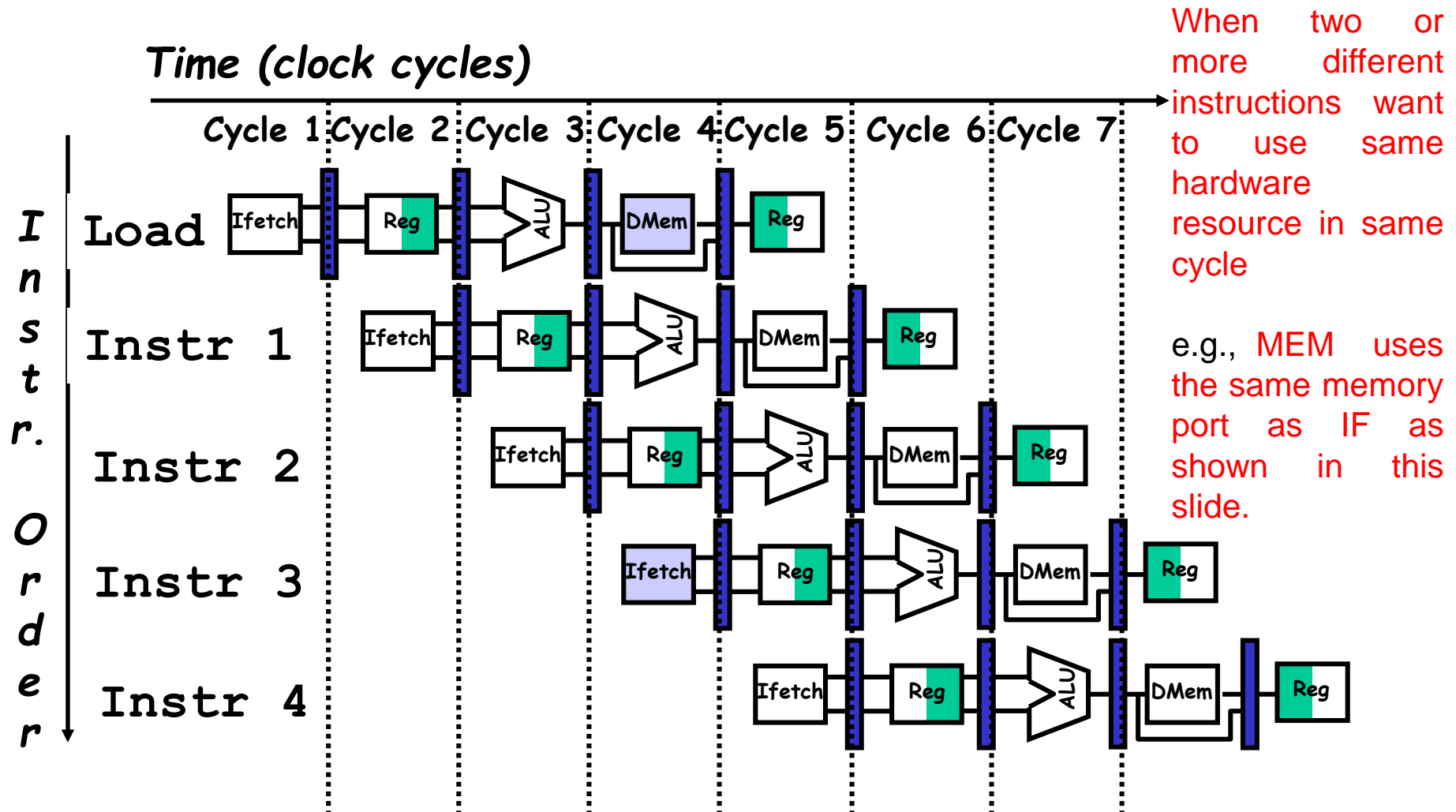


Figure 3.6

# A simple solution: Structural Hazards

Time (clock cycles)

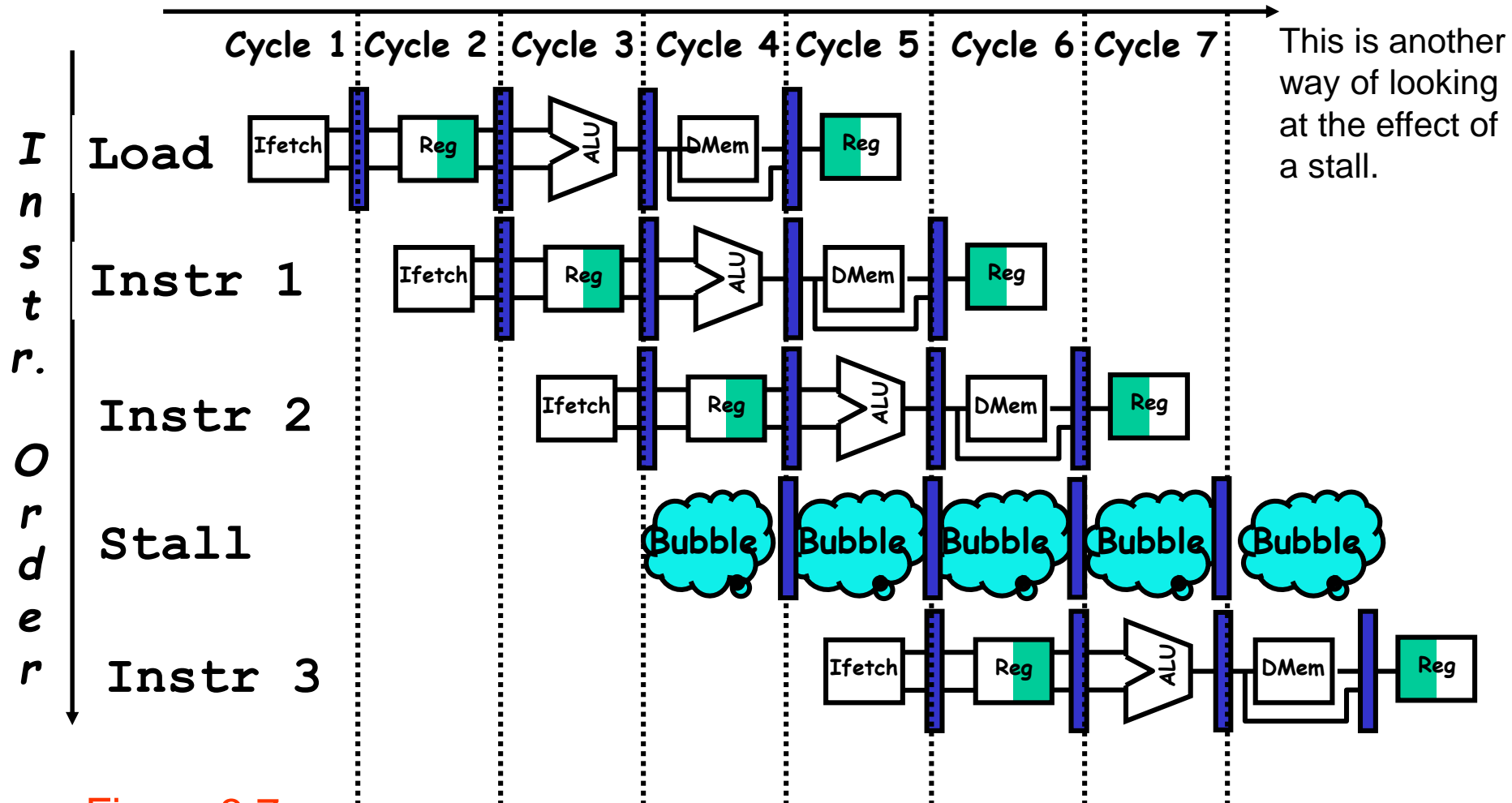


Figure 3.7

# Structural Hazards

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

This is another way to represent the stall we saw on the last few pages.

# Structural Hazards

## Dealing with Structural Hazards

### **Stall**

- low cost, simple
- Increases CPI
- use for rare case since stalling has performance effect

### **Pipeline hardware resource**

- useful for multi-cycle resources
- good performance
- sometimes complex e.g., RAM

### **Replicate resource** (e.g. Harvard Architecture)

- good performance
- increases cost (+ maybe interconnect delay)
- useful for cheap or divisible resources

# Structural Hazards

**Structural hazards are reduced with these rules:**

- 1. Each instruction uses a resource at most once**
- 2. Always use the resource in the same pipeline stage**
- 3. Use the resource for one cycle only**

**Many RISC ISA's designed with this in mind**

**Some common Structural Hazards:**

- Memory - we've already mentioned this one.**
- Floating point - Since many floating point instructions require many cycles, it's easy for them to interfere with each other.**

# Data Hazards

## A.1 What is Pipelining?

## A.2 The Major Hurdle of Pipelining- Structural Hazards

- Structural Hazards
- Data Hazards
- Control Hazards

## A.3 How is Pipelining Implemented

## A.4 What Makes Pipelining Hard to Implement?

## A.5 Extending the MIPS Pipeline to Handle Multi-cycle Operations

These occur when at any time, there are **instructions active in pipeline** that need to access the same data (memory or register) locations.

In the pipeline architecture, instructions sometimes **could not use right data** for its processing without proper handling.



# Data Hazards

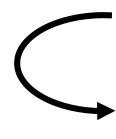
Execution Order is:

**Instr<sub>i</sub>**

**Instr<sub>j</sub>**

## Read After Write (RAW)

**Instr<sub>j</sub> tries to read operand before Instr<sub>i</sub> writes it**

 I: add **r1**, r2, r3  
J: sub r4, **r1**, r3

- Caused by a “**Dependence**” (in compiler nomenclature). This hazard results from an actual need for communication.

# The Basic Pipeline For MIPS

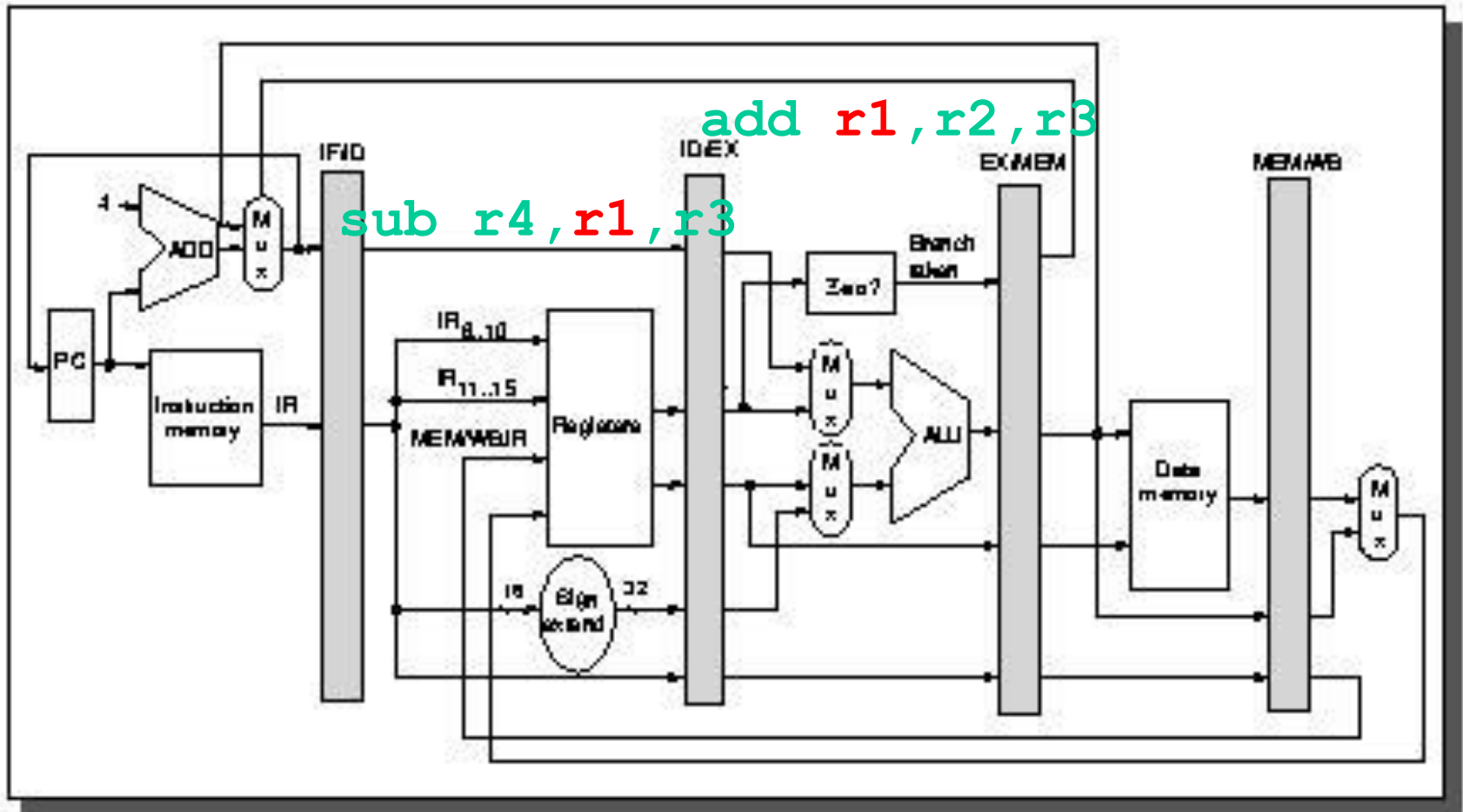


FIGURE 3.4 The datapath is pipelined by adding a set of registers, one between each pair of pipe stages.

# Data Hazards

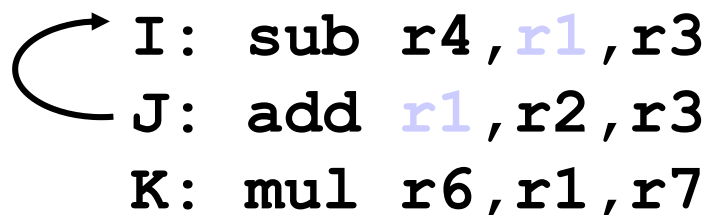
Execution Order is:

**Instr<sub>i</sub>**  
**Instr<sub>j</sub>**

## Write After Read (WAR)

Instr<sub>j</sub> tries to write operand before Instr<sub>i</sub> reads it

- Gets wrong operand



```
I:  sub  r4, r1, r3
J:  add  r1, r2, r3
K:  mul  r6, r1, r7
```

- Called an “anti-dependence” by compiler writers. This results from reuse of the name “r1”.

- **Can't happen in MIPS 5 stage pipeline because:**
  - All instructions take 5 stages, and
  - Reads are always in stage 2, and
  - Writes are always in stage 5

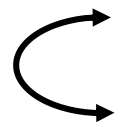
# Data Hazards

Execution Order is:  
Instr<sub>i</sub>  
Instr<sub>j</sub>

## Write After Write (WAW)

Instr<sub>j</sub> tries to write operand before Instr<sub>i</sub> writes it

- Leaves wrong result ( Instr<sub>i</sub> not Instr<sub>j</sub> )



```
I:  sub  r1, r4, r3
J:  add  r1, r2, r3
K:  mul  r6, r1, r7
```

- Called an “**output dependence**” by compiler writers  
This also results from the reuse of name “**r1**”.
- **Can't happen in MIPS 5 stage pipeline because:**
  - All instructions take 5 stages, and
  - Writes are always in stage 5
- **Will see WAR and WAW in later more complicated pipes**

# Data Hazards

## Simple Solution to RAW

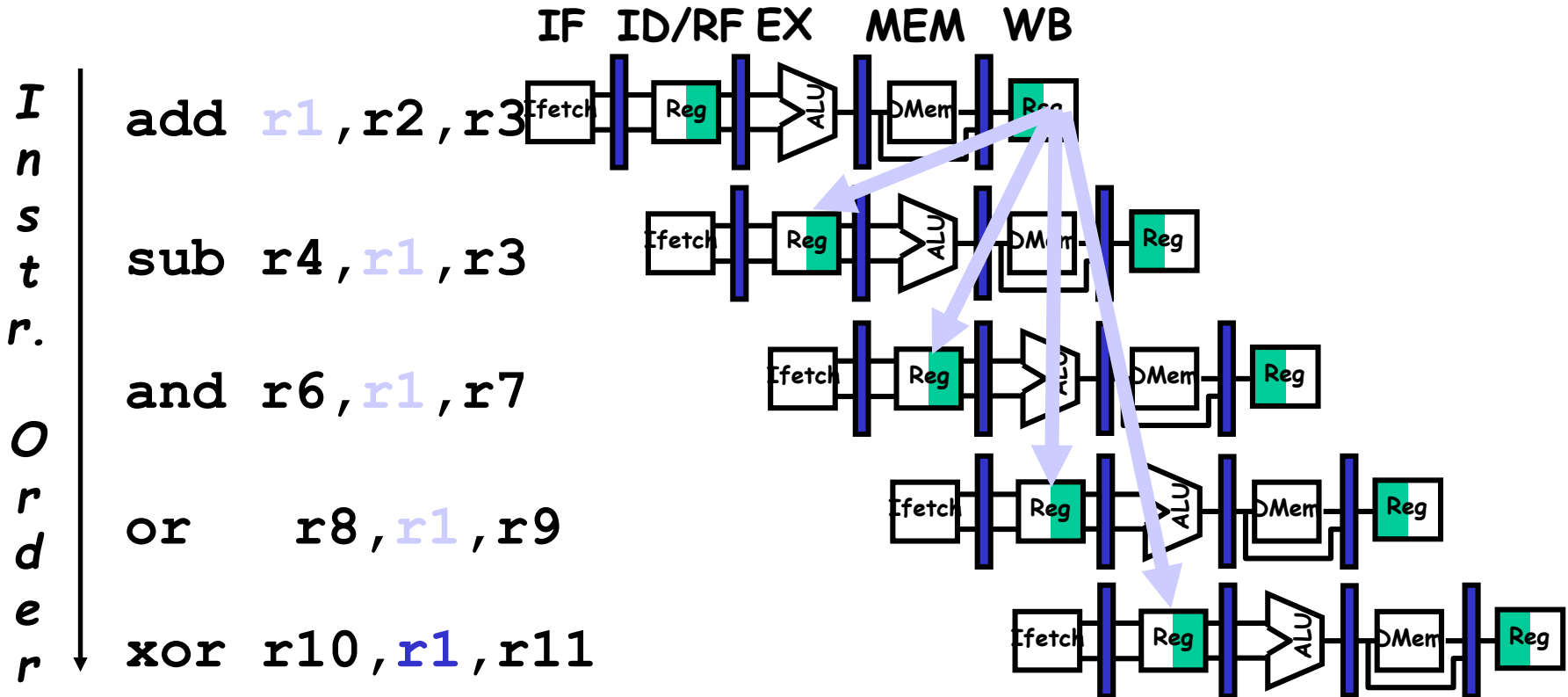
- Hardware detects RAW and stalls
- Assumes register written then read each cycle
  - + low cost to implement, simple
  - reduces IPC
- Try to minimize stalls

## Minimizing RAW stalls

- Bypass/forward/shortcircuit (We will use the word “forward”)
- Use data before it is delivered to the register
  - + reduces/avoids stalls
  - complex
- Crucial for common RAW hazards

# Data Hazards

Time (clock cycles)



The use of the result of the ADD instruction in the next three instructions causes a hazard, since the register is not written until after those instructions read it.

Figure 3.9

# Data Hazards

## Forwarding To Avoid Data Hazard

Forwarding is the concept of making data available to the input of the ALU for subsequent instructions, even though the generating instruction hasn't gotten to WB in order to write the memory or registers.

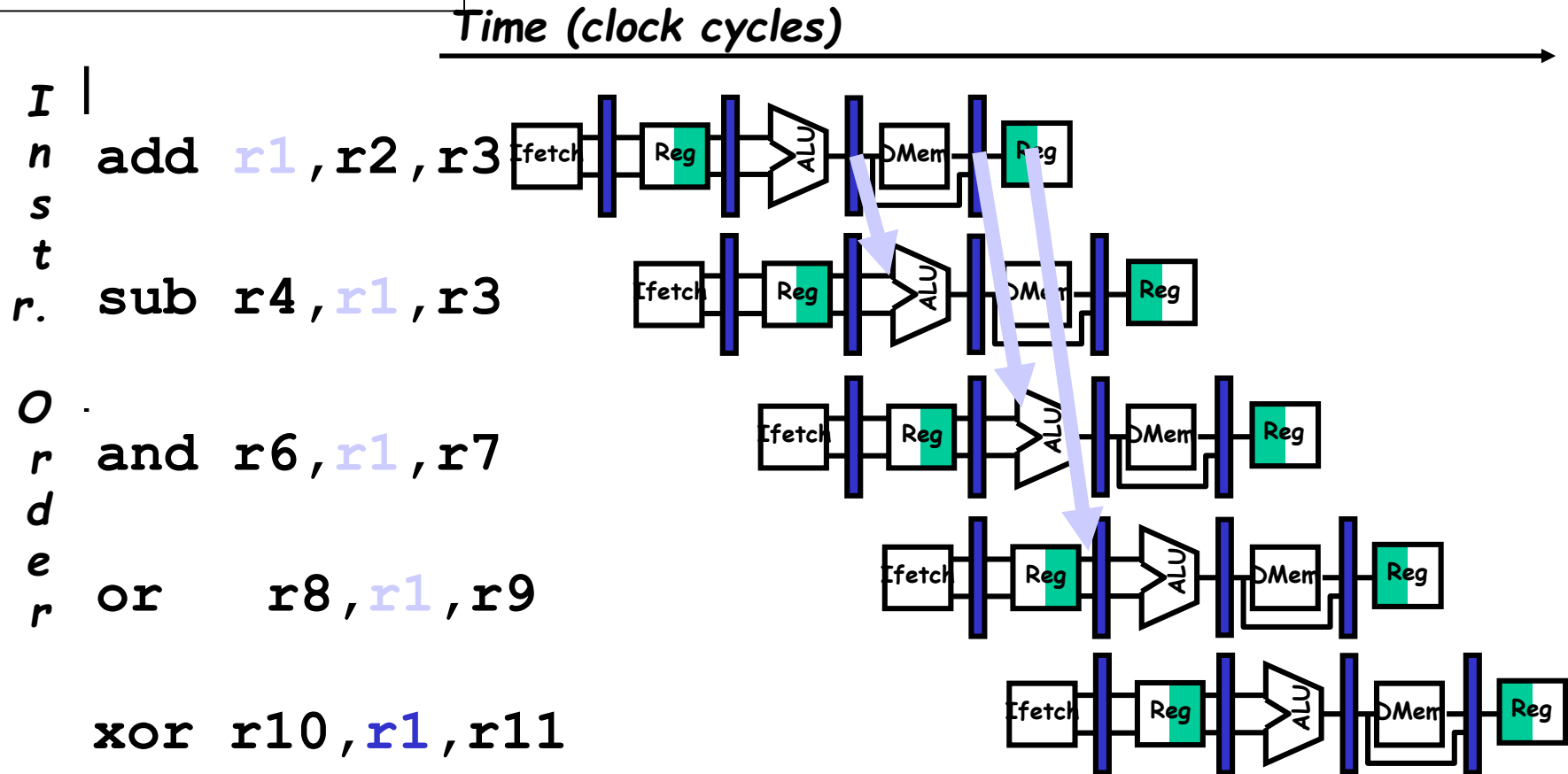
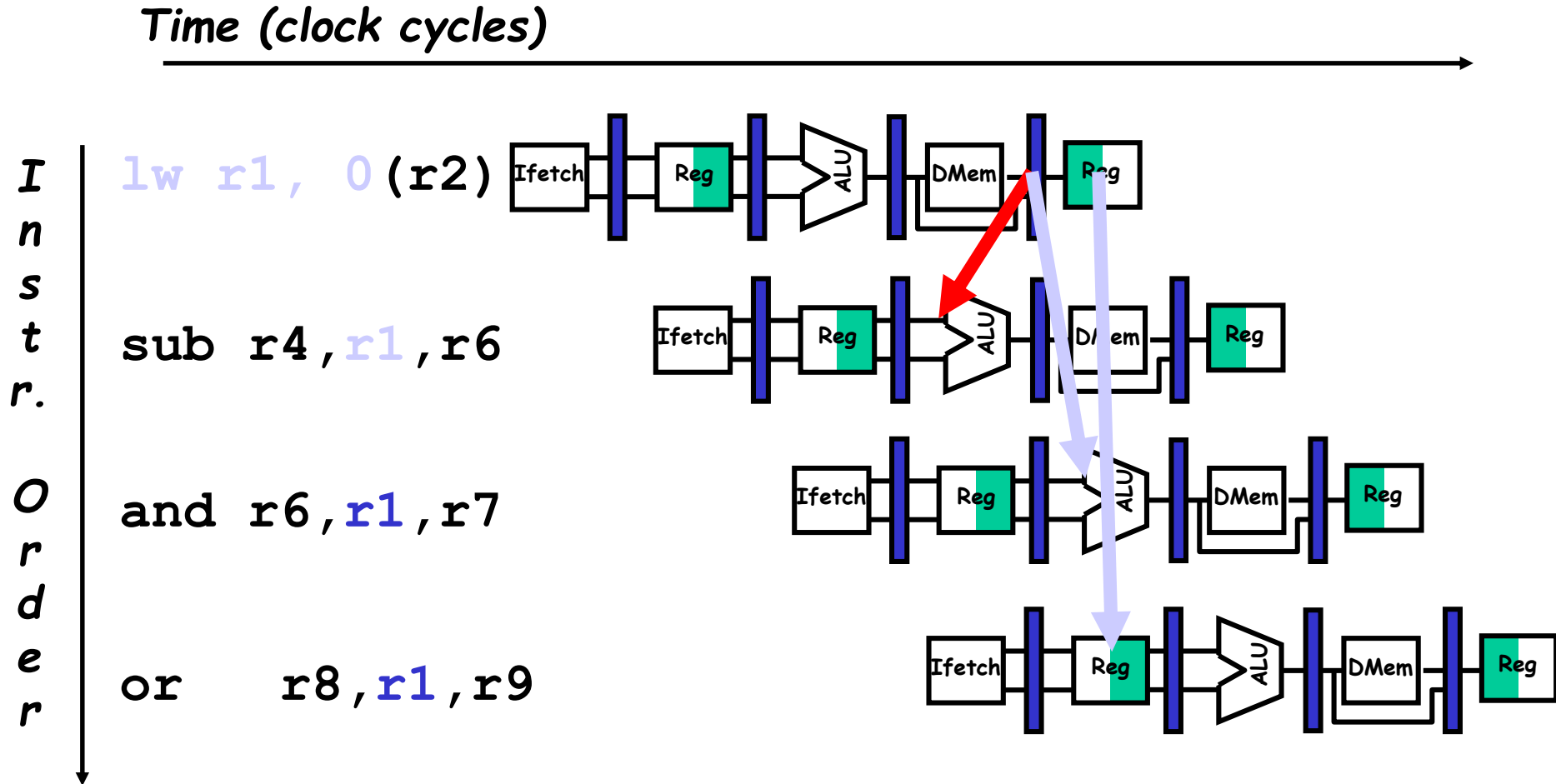


Figure 3.10

# Data Hazards

The data isn't loaded until after the MEM stage.



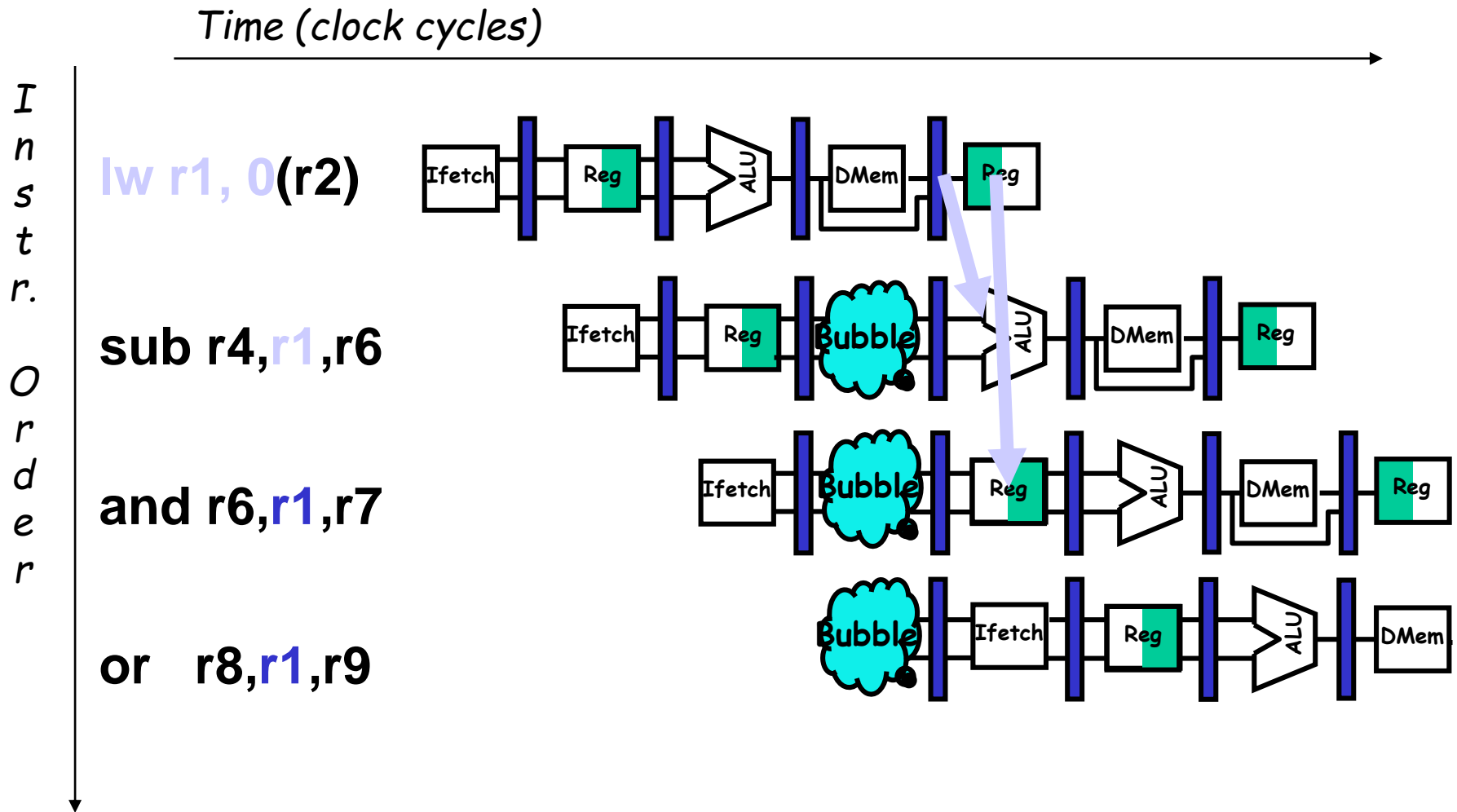
There are some instances where hazards occur, even with forwarding!

Figure 3.12



# Data Hazards

The stall is necessary as shown here.



There are some instances where hazards occur, even with forwarding.

Figure 3.13

# Data Hazards

This is another representation of the stall.

LW	R1, 0(R2)	IF	ID	EX	MEM	WB			
SUB	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

LW	R1, 0(R2)	IF	ID	EX	MEM	WB				
SUB	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

# Data Hazards

## Pipeline Scheduling

**Instruction scheduled by compiler - move instruction in order to reduce stall.**

<b>lw Rb, b</b>	code sequence for <b>a = b+c</b> before scheduling
<b>lw Rc, c</b>	
<b>Add Ra, Rb, Rc</b>	<b>stall</b>
<b>sw a, Ra</b>	
<b>lw Re, e</b>	code sequence for <b>d = e-f</b> before scheduling
<b>lw Rf, f</b>	
<b>sub Rd, Re, Rf</b>	<b>stall</b>
<b>sw d, Rd</b>	

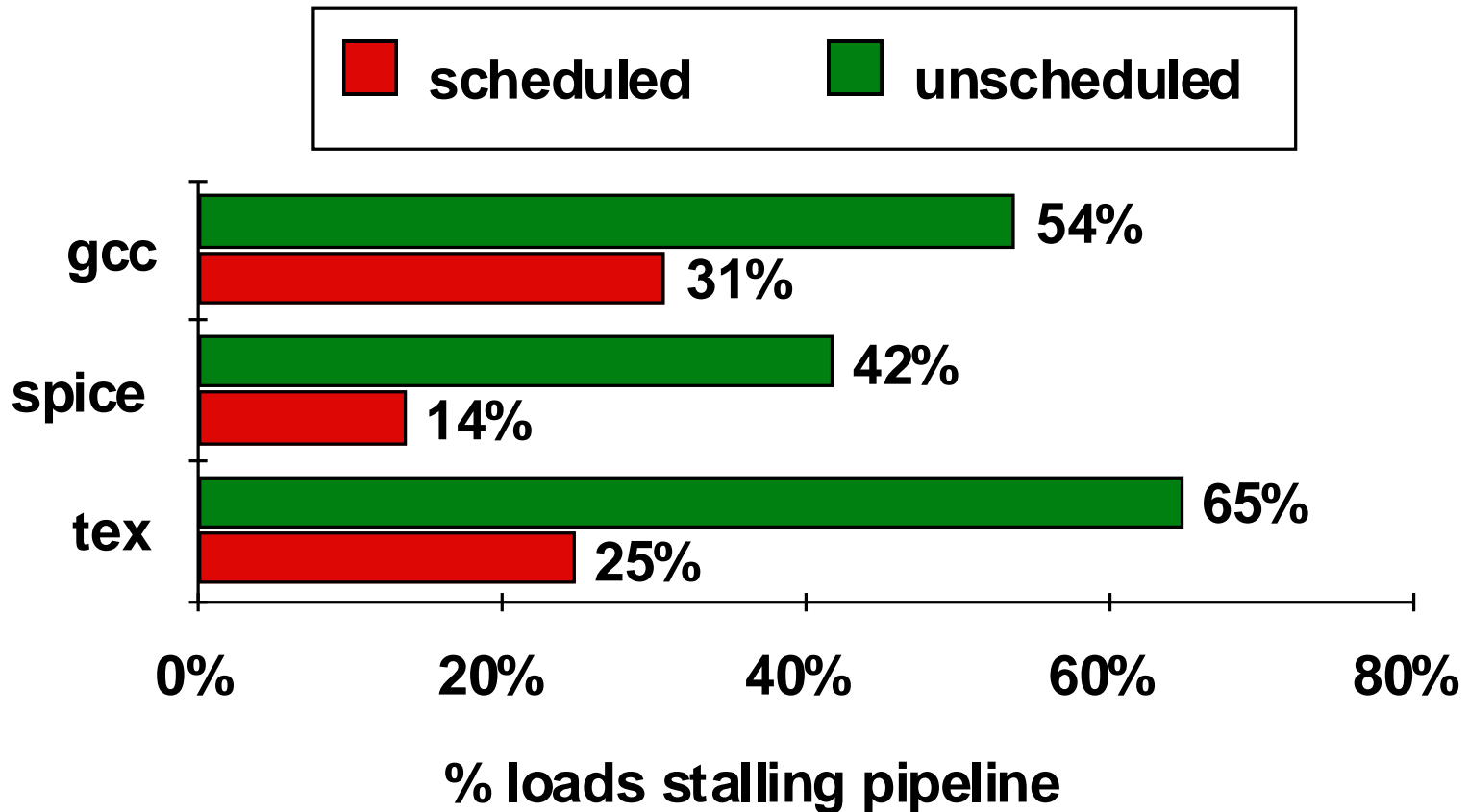
**Arrangement of code after scheduling.**

```
lw Rb, b
lw Rc, c
lw Re, e
Add Ra, Rb, Rc
lw Rf, f
sw a, Ra
sub Rd, Re, Rf
sw d, Rd
```

**No Stall!**

# Data Hazards

## Pipeline Scheduling



# Control Hazards

## **A.1 What is Pipelining?**

## **A.2 The Major Hurdle of Pipelining- Structural Hazards**

- Structural Hazards
- Data Hazards
- Control Hazards

## **A.3 How is Pipelining Implemented**

## **A.4 What Makes Pipelining Hard to Implement?**

## **A.5 Extending the MIPS Pipeline to Handle Multi-cycle Operations**

**A control hazard is when we need to find the destination of a branch, and can't fetch any new instructions until we know that destination.**

# Control Hazard on Branches Two Stage Stall



# Control Hazards

## Branch Stall Impact

- If  $CPI = 1$ , 30% branch, Stall 3 cycles  $\Rightarrow$  new  $CPI = 1.9$ !  
(Whoa! How did we get that 1.9???)
- Two part solution to this dramatic increase:
  - Determine branch taken or not sooner, AND
  - Compute taken branch address earlier

# Control Hazards

## Five Branch Hazard Alternatives

**#1: Stall until branch direction is clear**

**#2: Predict Branch Not Taken**

- Execute successor instructions in sequence
- “Squash” instructions in pipeline if branch actually taken
- Advantage of late pipeline state update
- **47% MIPS branches not taken on average**
- PC+4 already calculated, so use it to get next instruction

**#3: Predict Branch Taken**

- **53% MIPS branches taken on average**
- But haven't calculated branch target address in MIPS
  - MIPS still incurs 1 cycle branch penalty
  - Other machines: branch target known before outcome



# Control Hazards

## Five Branch Hazard Alternatives

### #4: Execute Both Paths

### #5: Delayed Branch

- Define branch to take place **AFTER** a following instruction

branch instruction

sequential successor<sub>1</sub>

sequential successor<sub>2</sub>

.....

sequential successor<sub>n</sub>

branch target if taken

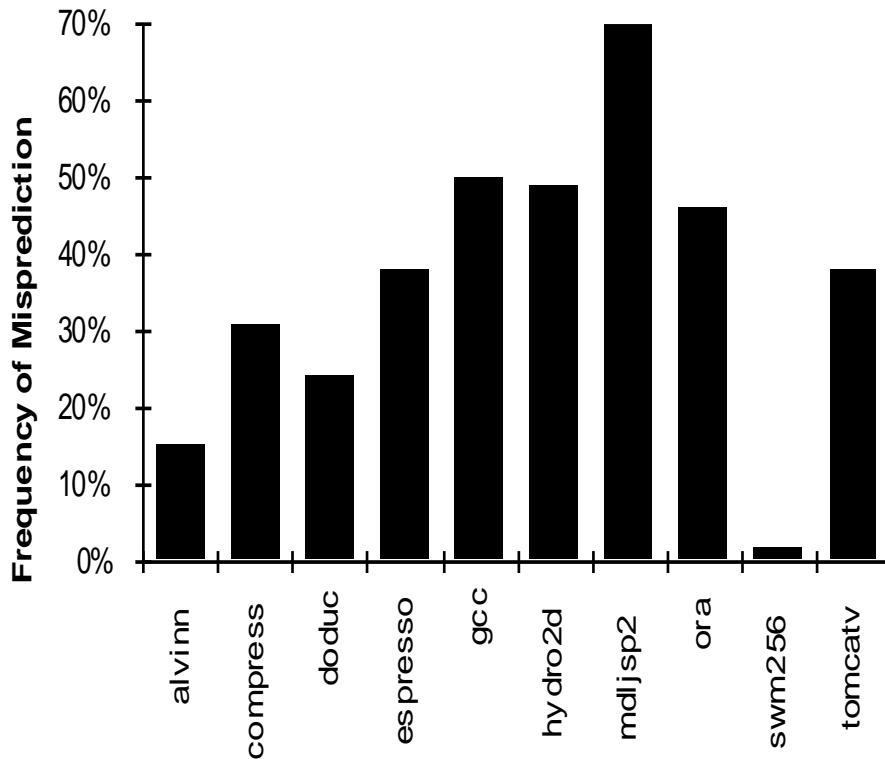


**Branch delay of length  $n$**

- Where to get instructions to fill branch delay slot?
  - **Before branch instruction**
  - From the target address: only valuable when branch taken
  - From fall through: only valuable when branch not taken

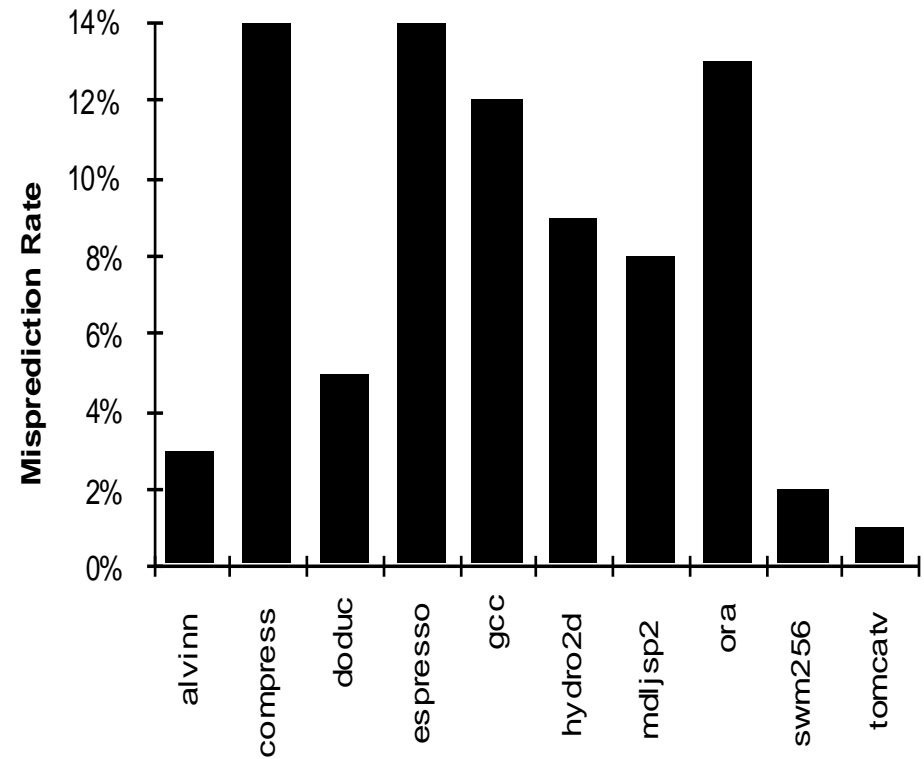
# Control Hazards

The compiler can program what it thinks the branch direction will be. Here are the results when it does so.



Always taken

## Compiler “Static” Prediction of Taken/Untaken Branches



Taken backwards  
Not Taken Forwards

# Control Hazards

## Compiler “Static” Prediction of Taken/Untaken Branches

- Improves strategy for placing instructions in delay slot
- Two strategies
  - Backward branch predict taken, forward branch not taken
  - Profile-based prediction: record branch behavior, predict branch based on prior run

# What Makes Pipelining Hard Except for the Hazards?

Complex  
Instructions

**Complex Addressing Modes and Instructions (require longer clocks to complete the instruction executions)**

- **Address modes: Autoincrement causes register change during instruction execution**
  - Interrupts? Need to restore register state
  - Adds WAR and WAW hazards since writes are no longer the last stage.
- **Memory-Memory Move Instructions**
  - Must be able to handle multiple page faults
  - Long-lived instructions: partial state save on interrupt
- **Condition Codes**

# Handling Multi-cycle Operations

**A.1 What is Pipelining?**

**A.2 The Major Hurdle of Pipelining-  
Structural Hazards**

- Data Hazards
- Control Hazards

**A.3 How is Pipelining Implemented**

**A.4 What Makes Pipelining Hard to  
Implement?**

**A.5 Extending the MIPS Pipeline to  
Handle Multi-cycle Operations**

**Multi-cycle instructions also  
lead to pipeline complexity.**

**A very lengthy instruction  
causes everything else in  
the pipeline to wait for it.**

# Multi-Cycle Operations

## Floating Point

**Floating point gives long execution time.**

**This causes a stall of the pipeline.**

**It's possible to pipeline the FP execution unit so it can initiate new instructions without waiting full latency. Can also have multiple FP units.**

<i>FP Instruction</i>	<i>Latency</i>	<i>Initiation Rate</i>
Add, Subtract	4	3
Multiply	8	4
Divide	36	35
Square root	112	111
Negate	2	1
Absolute value	2	1
FP compare	3	2

# Multi-Cycle Operations

## Floating Point

**Divide, Square Root take 10X to 30X longer than Add**

- Interrupts?
- Adds WAR and WAW hazards since pipelines are no longer same length

	1	2	3	4	5	6	7	8	9	10	11
i	IF	ID	EX	MEM	WB						
i + 1		IF	ID	EX	EX	EX	EX	MEM	WB		
i + 2			IF	ID	EX	MEM	WB				
i + 3				IF	ID	EX	EX	EX	EX	MEM	WB
i + 4					IF	ID	EX	MEM	WB		
i + 5						IF	ID	--	--	EX	EX
i + 6							IF	--	--	ID	EX

# Summary of Pipelining Hazards

- **Hazards limit performance**
  - Structural: need more HW resources
  - Data: need forwarding, compiler scheduling
  - Control: early evaluation & PC, delayed branch, prediction
- **Increasing length of pipe increases impact of hazards; pipelining helps instruction bandwidth, not latency**
- **Instruction Set, FP makes pipelining harder**
- **Compilers may reduce cost of data and control hazards**
  - Load delay slots
  - Branch delay slots
  - Branch prediction



# Summary

## **A.1 What is Pipelining?**

## **A.2 The Major Hurdle of Pipelining-Structural Hazards**

- Data Hazards
- Control Hazards

## **A.3 How is Pipelining Implemented**

## **A.4 What Makes Pipelining Hard to Implement?**

## **A.5 Extending the MIPS Pipeline to Handle Multi-cycle Operations**