

**CS252**  
**Graduate Computer Architecture**  
**Lecture 1**

**Review of Pipelines, Performance, Caches, and  
Virtual Memory(!)**

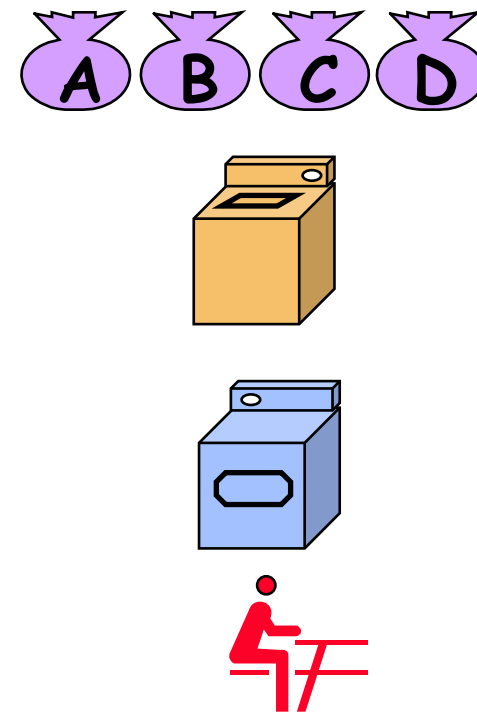
**January 17, 2001**  
**Prof. David A. Patterson**  
**Computer Science 252**  
**Spring 2001**

# Coping with CS 252

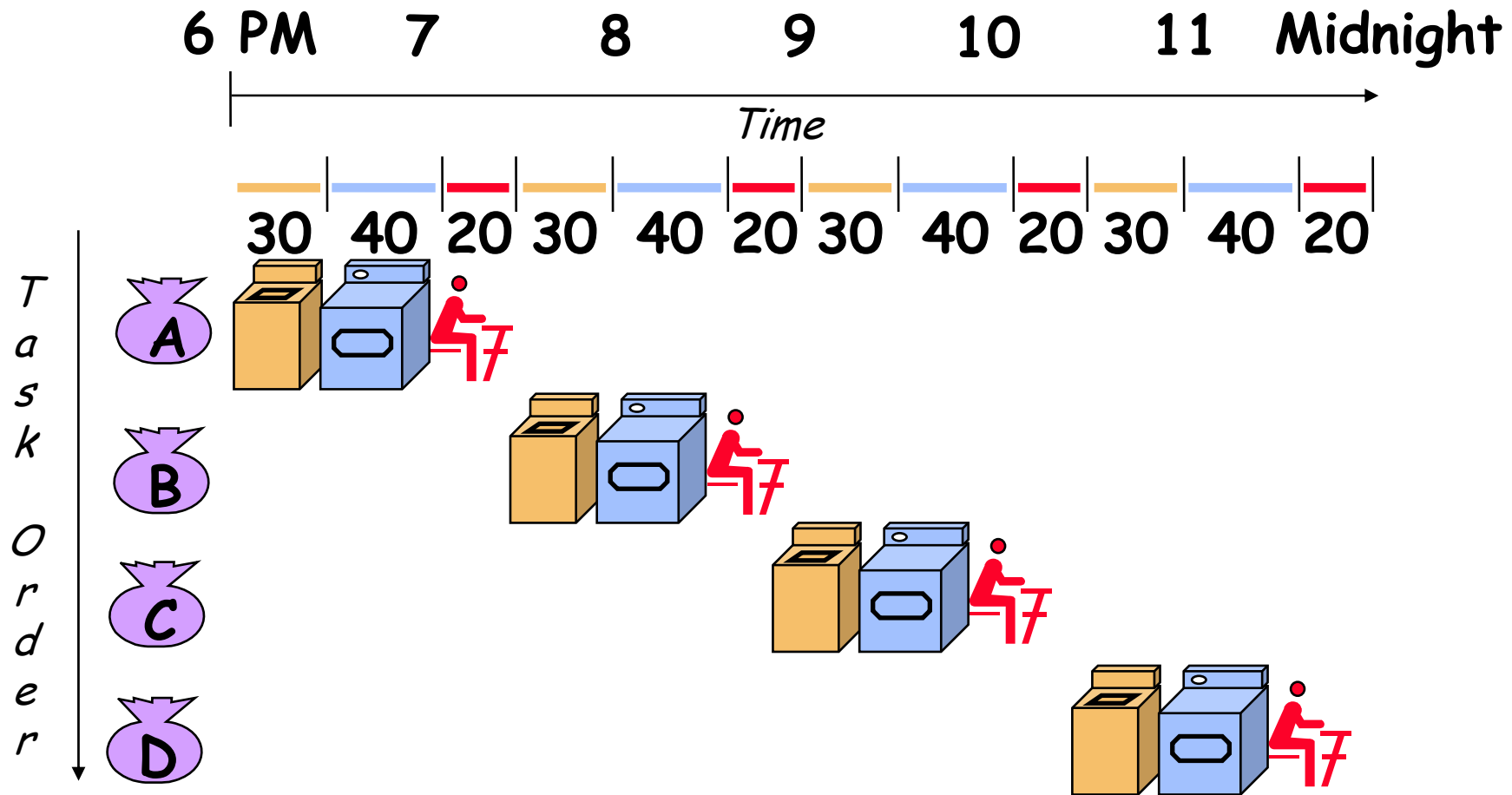
- **Students with too varied background?**
  - In past, CS grad students took written prelim exams on undergraduate material in hardware, software, and theory
  - 1st 5 weeks reviewed background, helped 252, 262, 270
  - Prelims were dropped => some unprepared for CS 252?
- **In class exam on Friday January 19 (30 mins)**
  - Doesn't affect grade, only admission into class
  - 2 grades: Admitted or audit/take CS 152 1st
  - Improve your experience if recapture common background
- **Review: Chapters 1, CS 152 home page, maybe "Computer Organization and Design (COD)2/e"**
  - Chapters 1 to 8 of COD if never took prerequisite
  - If took a class, be sure COD Chapters 2, 6, 7 are familiar
  - Copies in Bechtel Library on 2-hour reserve
- **FAST review today of Pipelining, Performance, Caches, and Virtual Memory**

# Pipelining: Its Natural!

- Laundry Example
- Ann, Brian, Cathy, Dave each have one load of clothes to wash, dry, and fold
- Washer takes 30 minutes
- Dryer takes 40 minutes
- “Folder” takes 20 minutes



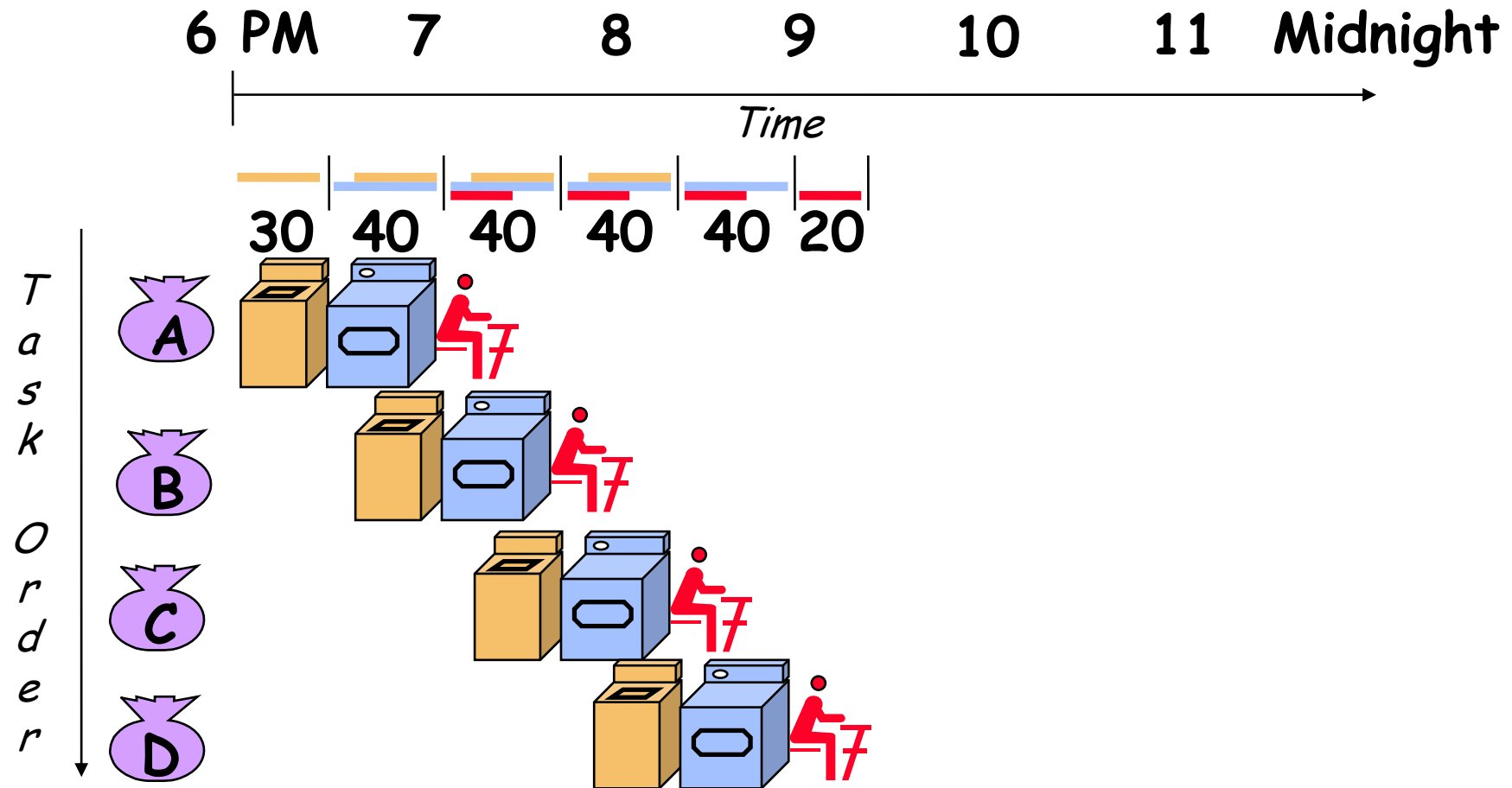
# Sequential Laundry



- Sequential laundry takes 6 hours for 4 loads
- If they learned pipelining, how long would laundry take?

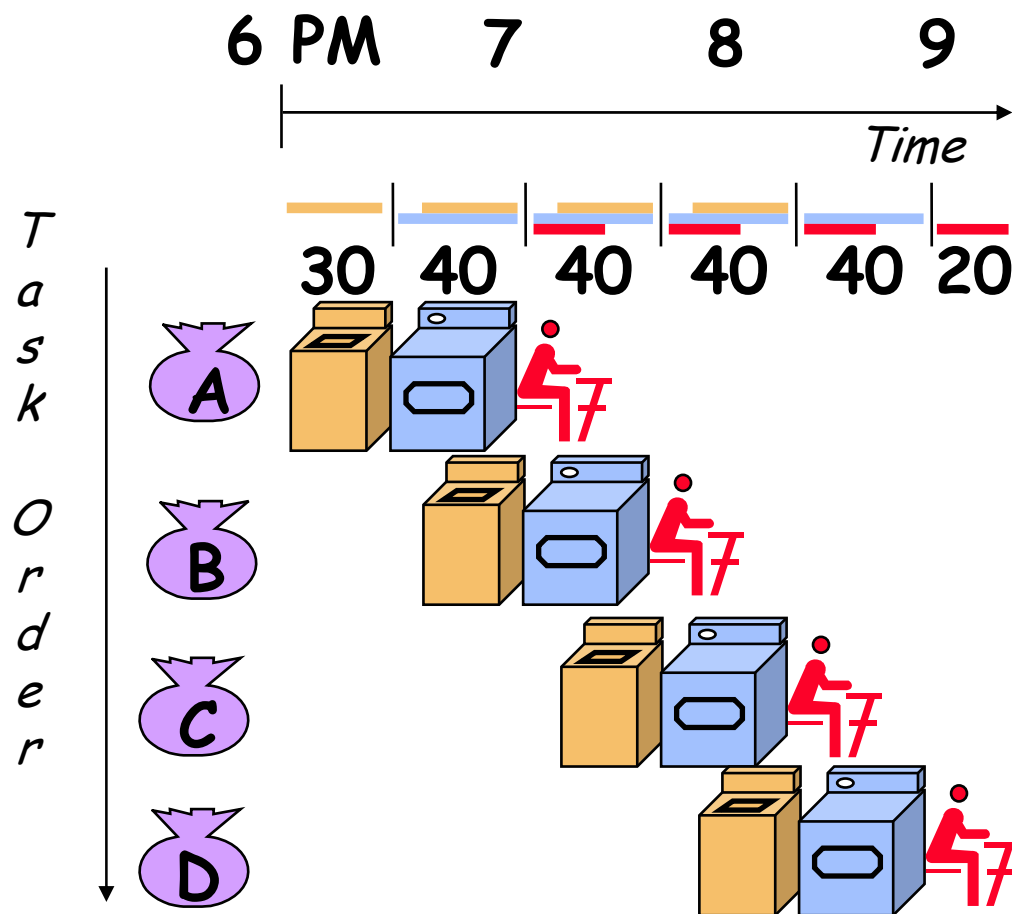
# Pipelined Laundry

Start work ASAP



- Pipelined laundry takes 3.5 hours for 4 loads

# Pipelining Lessons



- Pipelining doesn't help **latency** of single task, it helps **throughput** of entire workload
- Pipeline rate limited by **slowest** pipeline stage
- **Multiple** tasks operating simultaneously
- Potential speedup = **Number pipe stages**
- Unbalanced lengths of pipe stages reduces speedup
- Time to "**fill**" pipeline and time to "**drain**" it reduces speedup

# Computer Pipelines

- Execute billions of instructions, so *throughput* is what matters
- What is desirable in instruction sets for pipelining?
  - Variable length instructions vs. all instructions same length?
  - Memory operands part of any operation vs. memory operands only in loads or stores?
  - Register operand many places in instruction format vs. registers located in same place?

# A "Typical" RISC

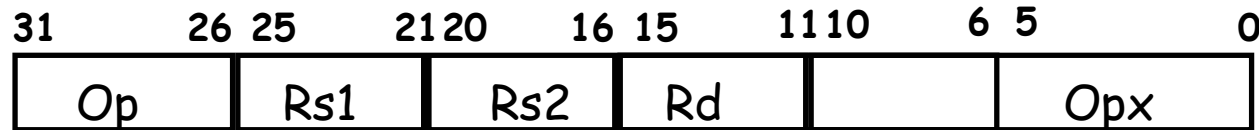
- 32-bit fixed format instruction (3 formats)
- Memory access only via load/store instructions
- 32 32-bit GPR (R0 contains zero, DP take pair)
- 3-address, reg-reg arithmetic instruction; registers in same place
- Single address mode for load/store:  
base + displacement
  - no indirection
- Simple branch conditions
- Delayed branch

see: SPARC, MIPS, HP PA-Risc, DEC Alpha, IBM PowerPC,  
CDC 6600, CDC 7600, Cray-1, Cray-2, Cray-3

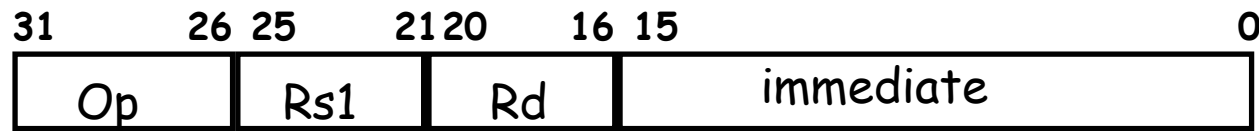


# Example: MIPS (Note register location)

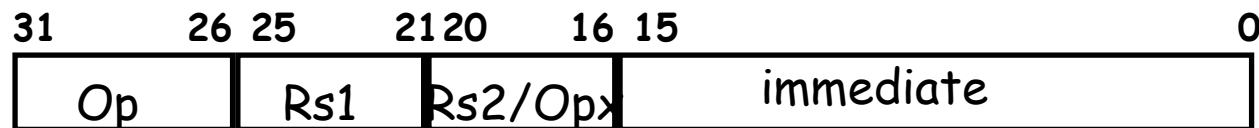
## Register-Register



## Register-Immediate



## Branch

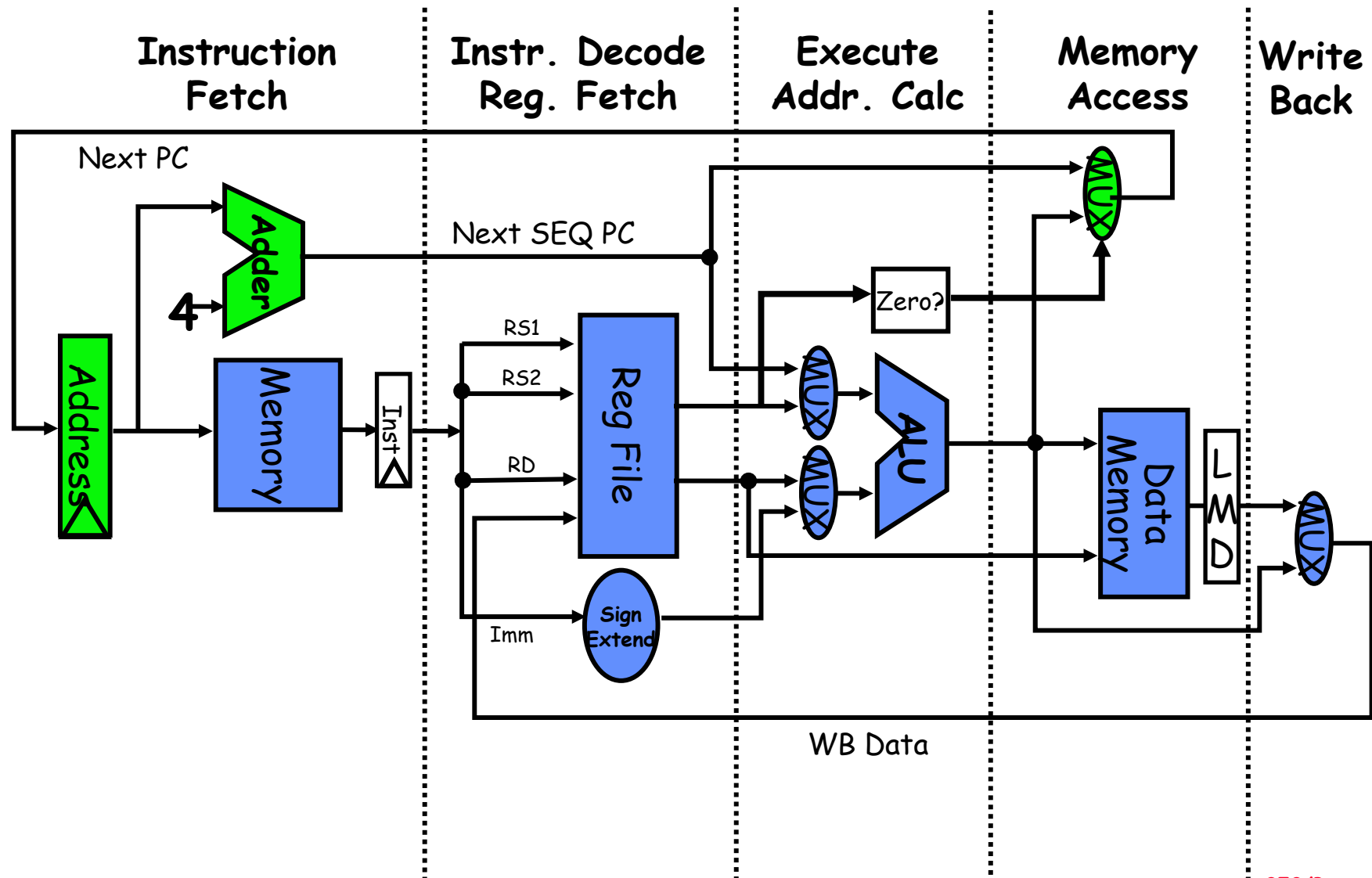


## Jump / Call



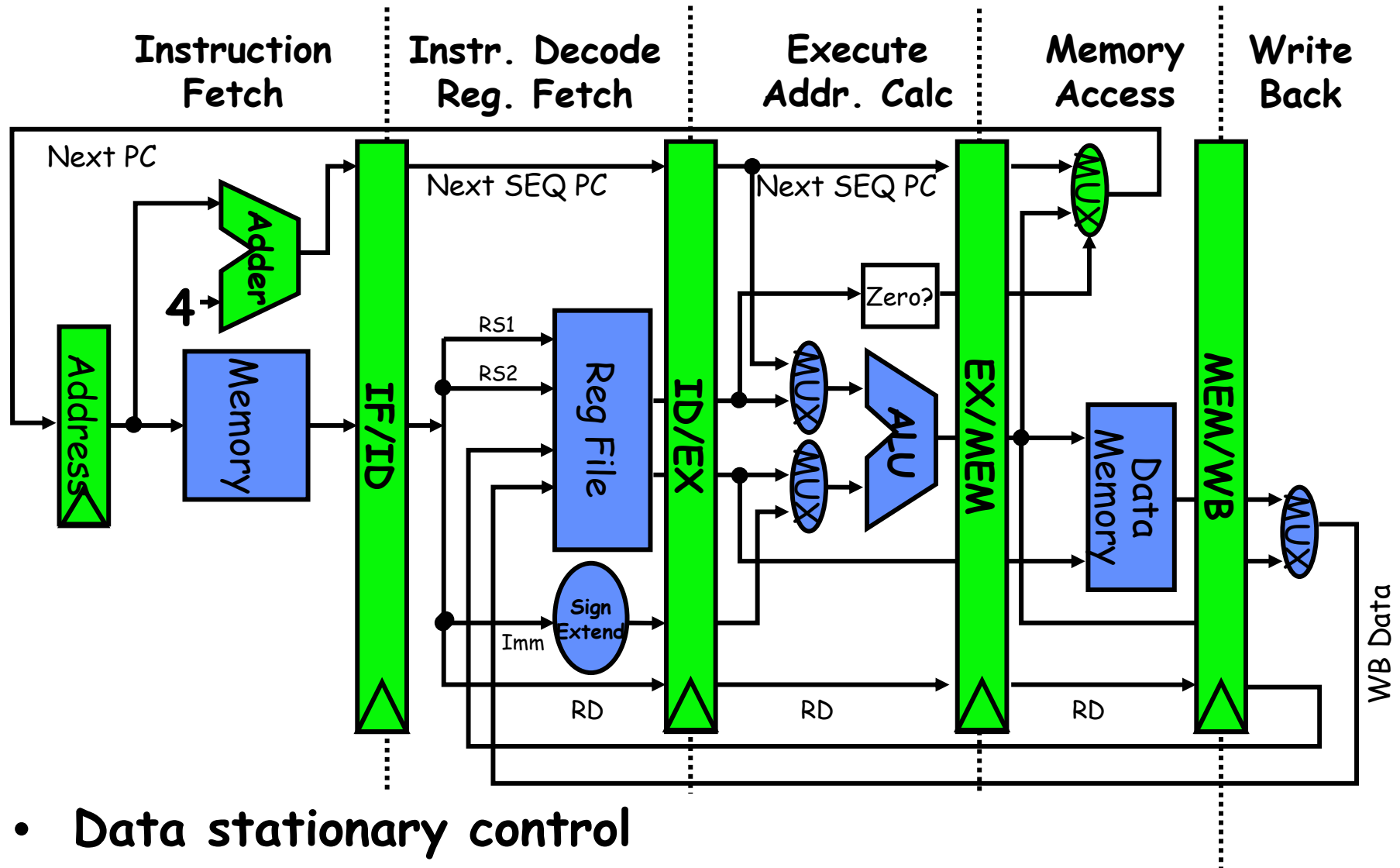
# 5 Steps of MIPS Datapath

Figure 3.1, Page 130, CA:AQA 2e



# 5 Steps of MIPS Datapath

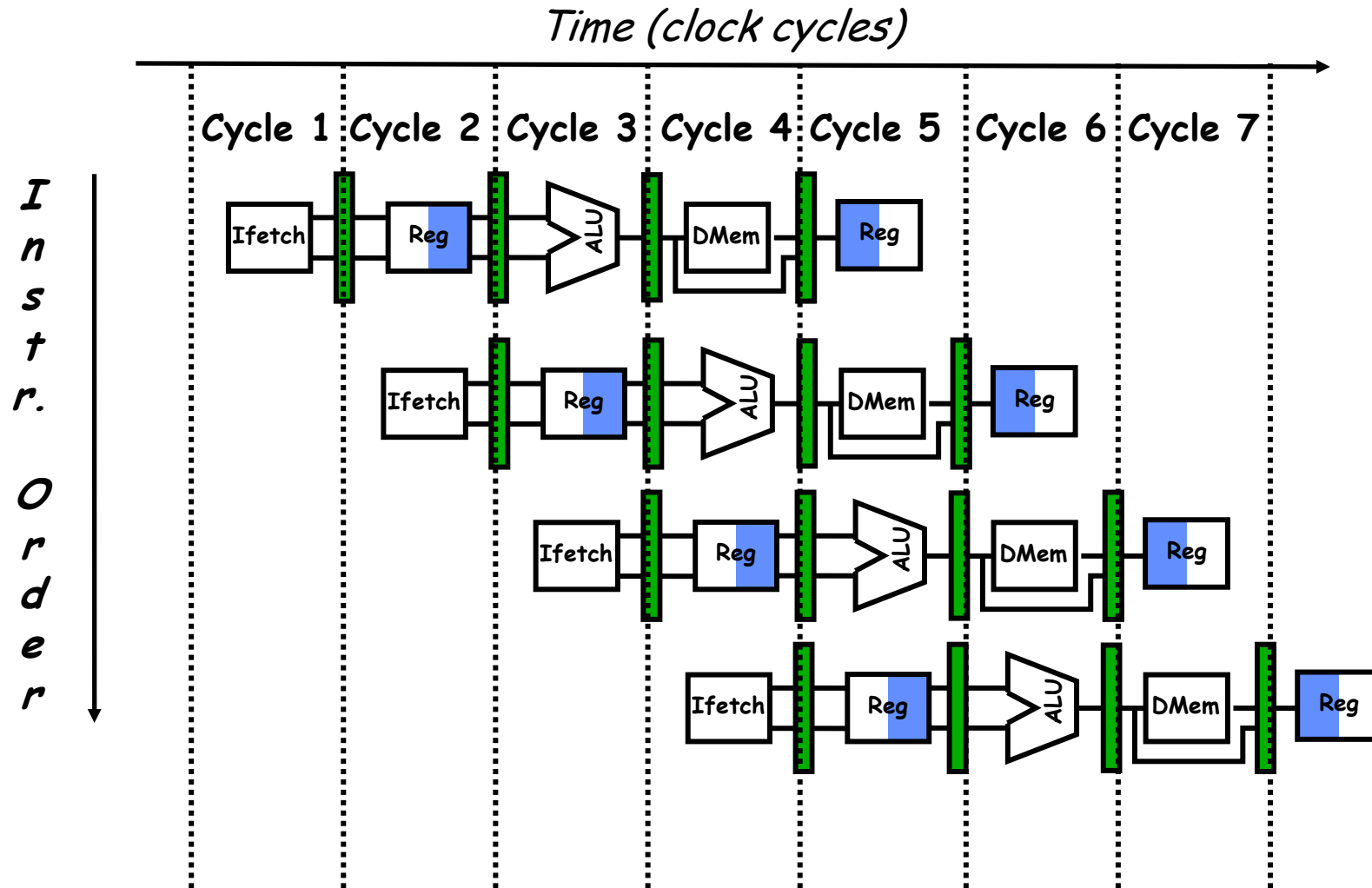
Figure 3.4, Page 134 , CA:AQA 2e



- Data stationary control
  - local decode for each instruction phase / pipeline stage

# Visualizing Pipelining

Figure 3.3, Page 133 , CA:AQA 2e

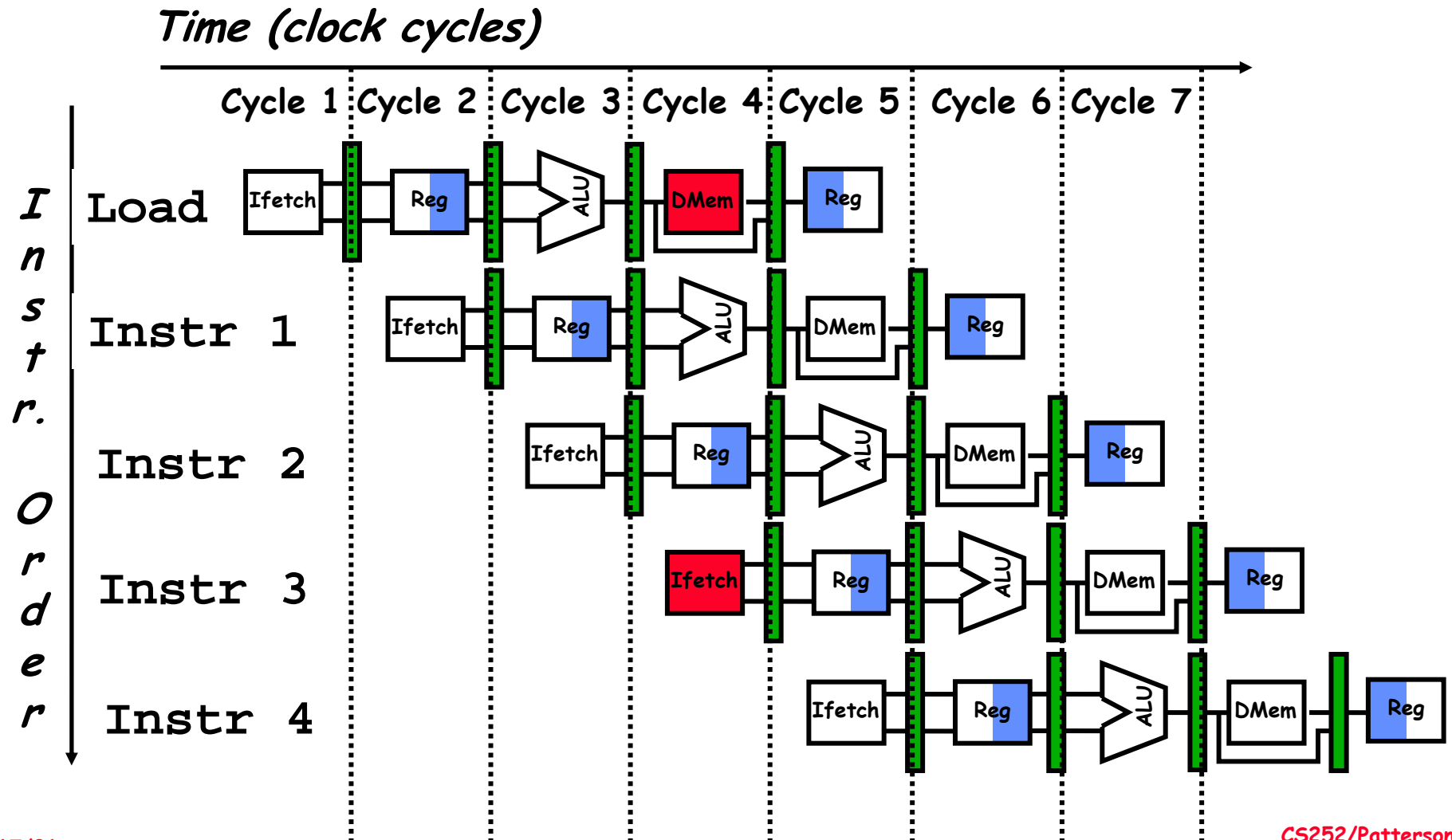


# Its Not That Easy for Computers

- Limits to pipelining: **Hazards** prevent next instruction from executing during its designated clock cycle
  - Structural hazards: HW cannot support this combination of instructions (single person to fold and put clothes away)
  - Data hazards: Instruction depends on result of prior instruction still in the pipeline (missing sock)
  - Control hazards: Caused by delay between the fetching of instructions and decisions about changes in control flow (branches and jumps).

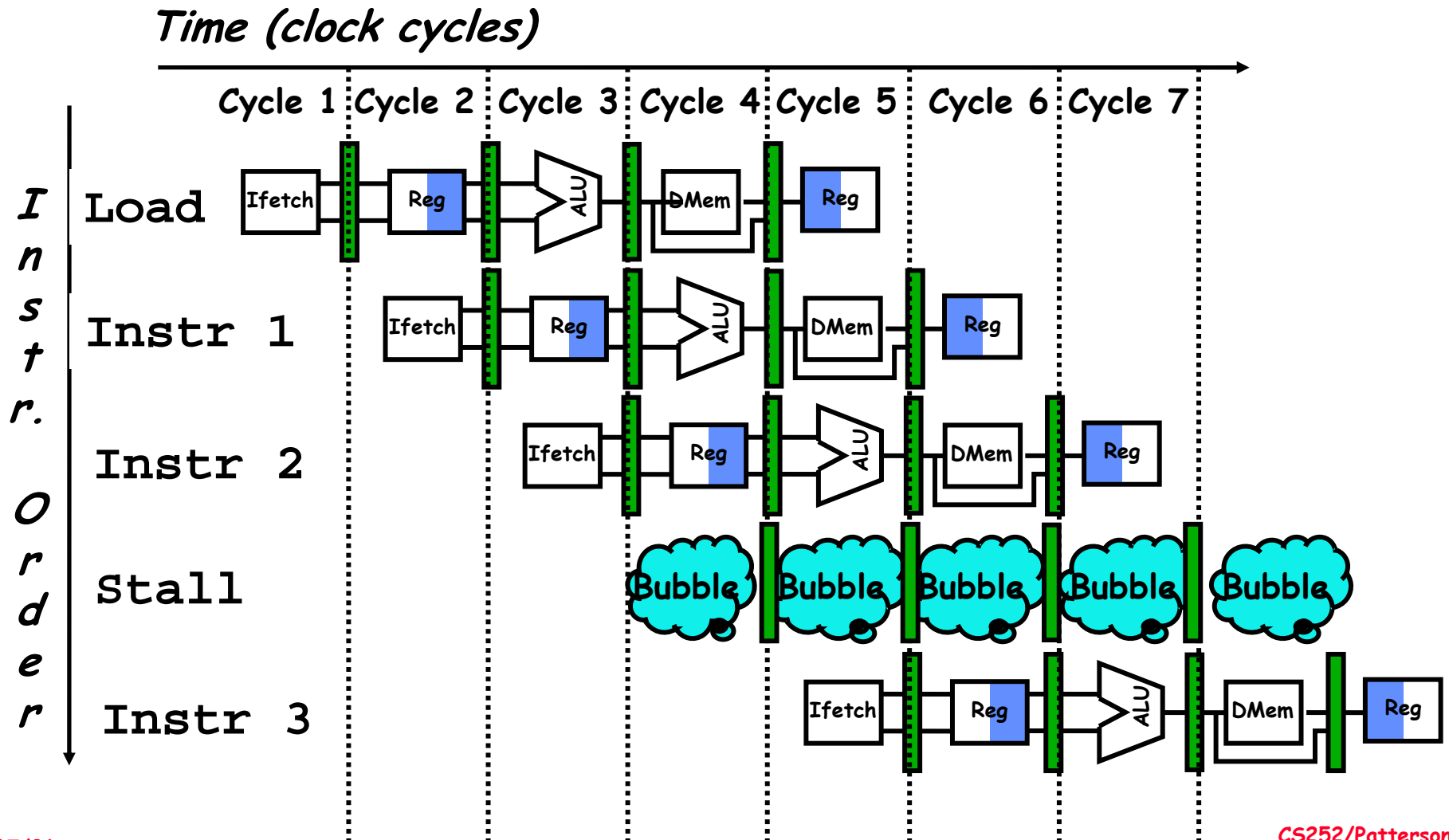
# One Memory Port/Structural Hazards

Figure 3.6, Page 142 , CA:AQA 2e



# One Memory Port/Structural Hazards

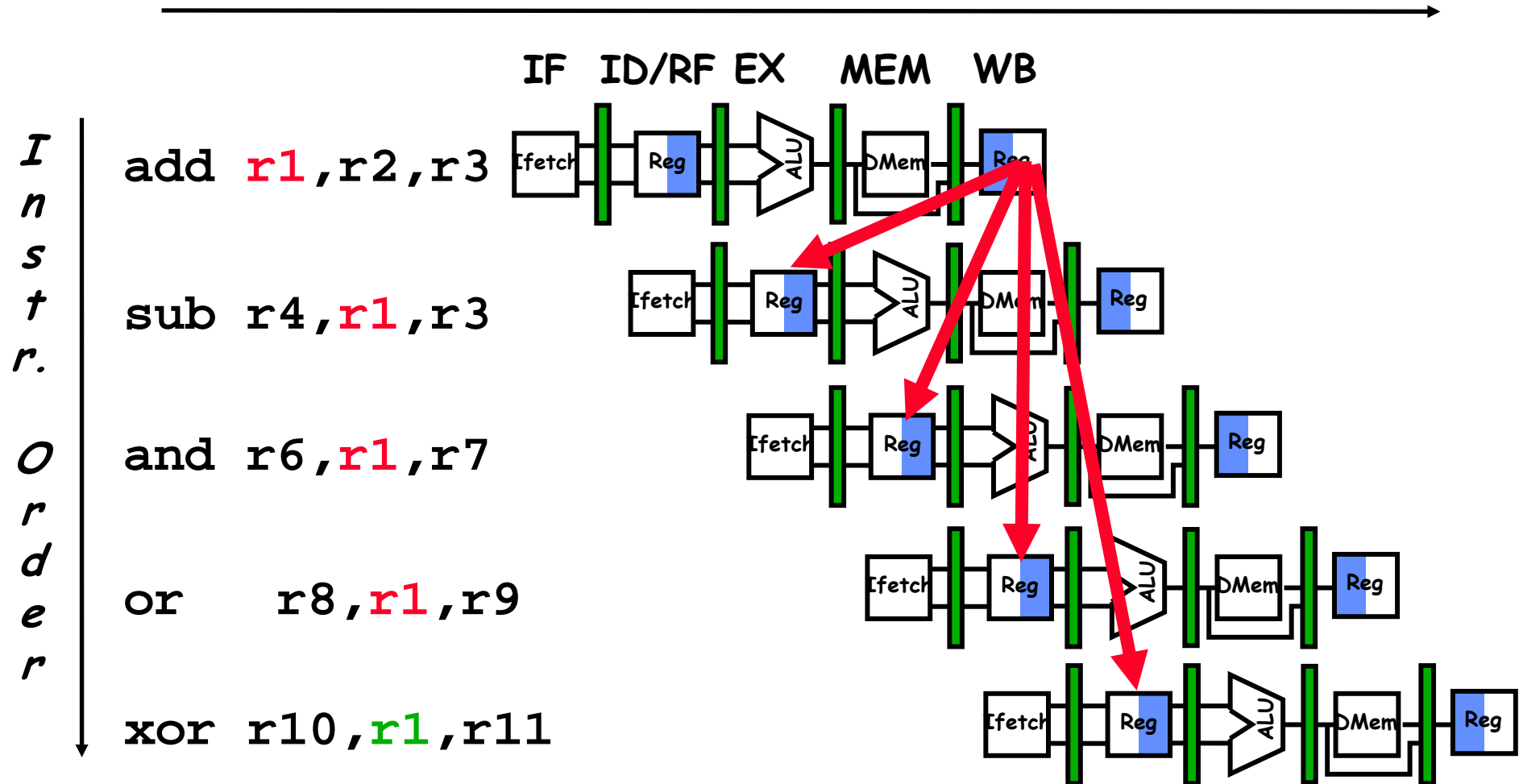
Figure 3.7, Page 143 , CA:AQA 2e



# Data Hazard on R1

Figure 3.9, page 147 , CA:AQA 2e

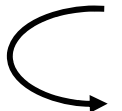
Time (clock cycles)





# Three Generic Data Hazards

- **Read After Write (RAW)**  
Instr<sub>J</sub> tries to read operand before Instr<sub>I</sub> writes it

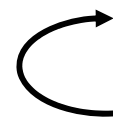
 I: add **r1**, r2, r3  
J: sub r4, **r1**, r3

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

# Three Generic Data Hazards

- **Write After Read (WAR)**

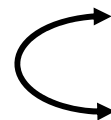
Instr<sub>J</sub> writes operand before Instr<sub>I</sub> reads it

 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

# Three Generic Data Hazards

- **Write After Write (WAW)**  
Instr<sub>J</sub> writes operand before Instr<sub>I</sub> writes it.

 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

## CS 252 Administrivia

- All assignments, lectures via WWW page:  
<http://www.cs.berkeley.edu/~pattrsn/252S01/>
- 2 Quizzes (given evenings in 8th and 14th week)
- Text: Beta copy of 3rd edition of Computer Architecture: A Quantitative Approach + "Readings in Computer Architecture" by Hill et al
- In class exam on Friday Jan 19, last 30 minutes
  - Improve 252 experience if recapture common background
  - Bring 1 sheet of paper with notes on both sides
  - Doesn't affect grade, only admission into class
  - 2 grades: Admitted or audit/take CS 152 1st
  - Review: Chapters 1, CS 152 home page, maybe "Computer Organization and Design (COD)2/e"
  - If did take a class, be sure COD Chapters 2, 5, 6, 7 are familiar
  - Copies in Bechtel Library on 2-hour reserve

# Research Paper Reading

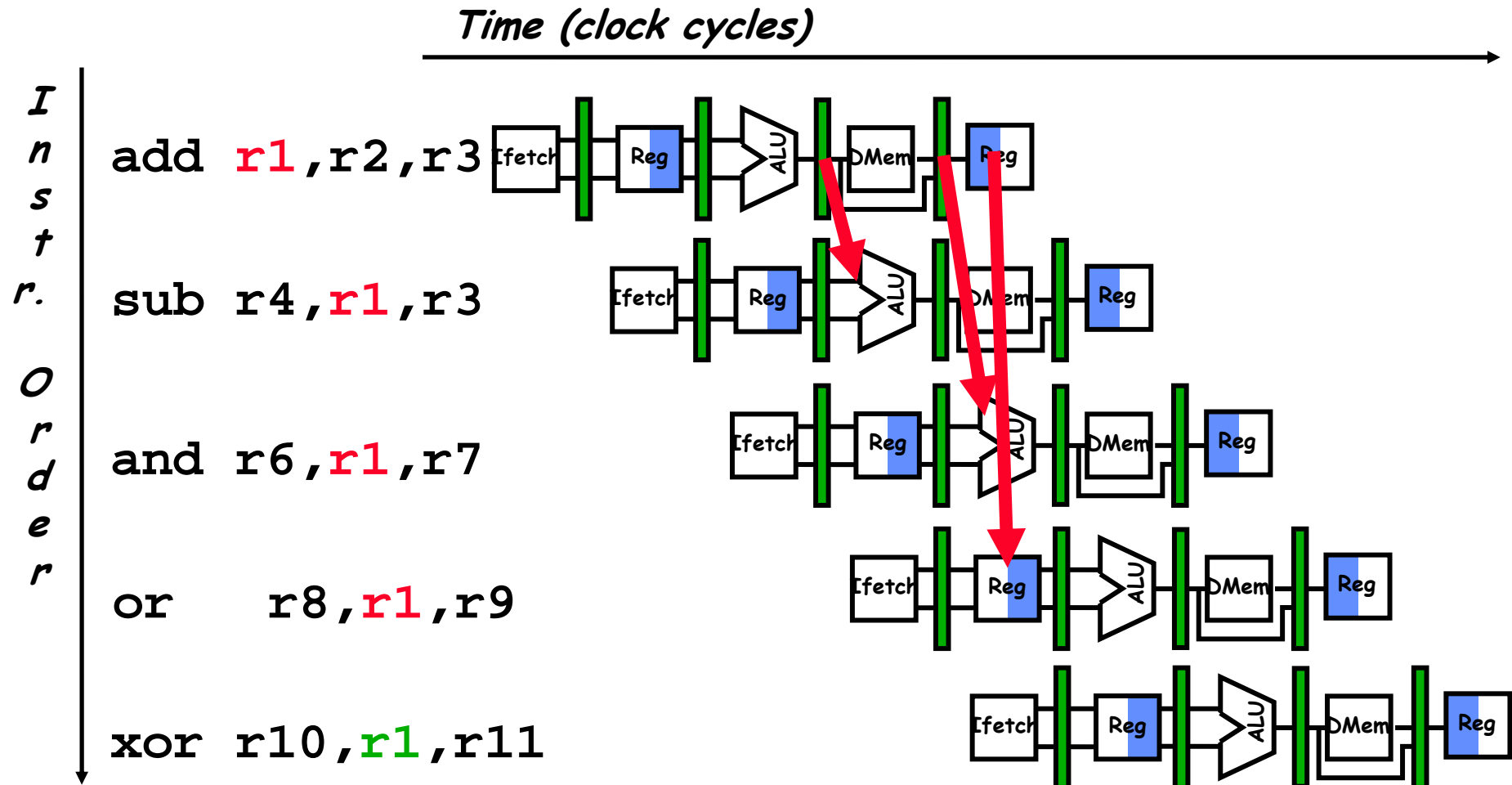
- As graduate students, you are now researchers.
- Most information of importance to you will be in research papers.
- Ability to rapidly scan and understand research papers is key to your success.
- So: 1 paper / week in this course
  - Quick 1 paragraph summaries will be due in class
  - Important supplement to book.
  - Will discuss papers in class
- Papers “Readings in Computer Architecture” or online
- First assignment (before Friday): Read p. 56-59 “Cramming More Components onto Integrated Circuits” by G.E. Moore, 1965 (“Moore’s Law”)

# Grading

- 10% Homeworks (work in pairs)
- 40% Examinations (2 Quizzes)
- 40% Research Project (work in pairs)
  - Transition from undergrad to grad student
  - Berkeley wants you to succeed, but you need to show initiative
  - pick topic
  - meet 3 times with faculty/TA to see progress
  - give oral presentation
  - give poster session
  - written report like conference paper
  - 3 weeks work full time for 2 people
  - Opportunity to do “research in the small” to help make transition from good student to research colleague
- 10% Class Participation

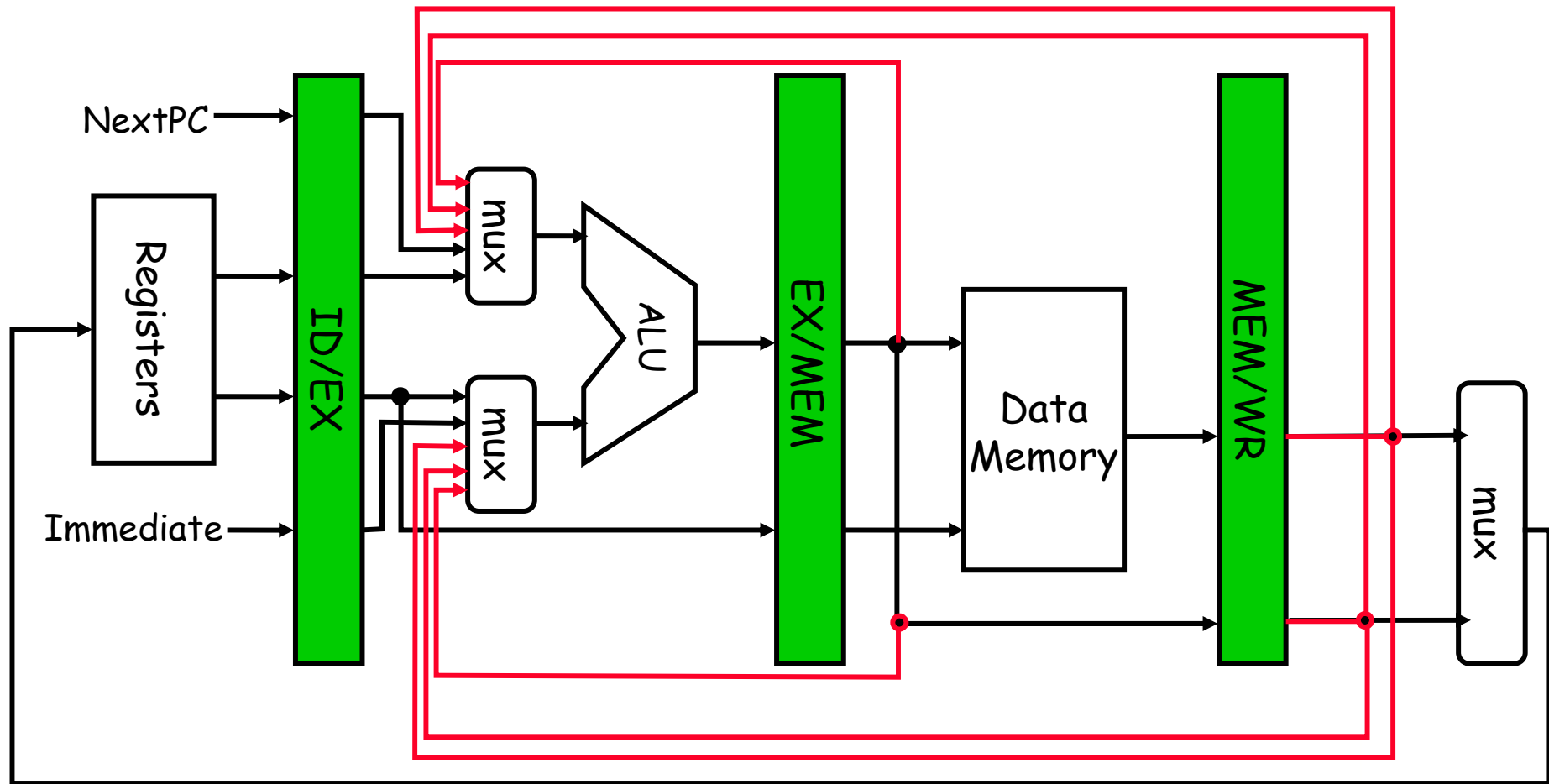
# Forwarding to Avoid Data Hazard

Figure 3.10, Page 149 , CA:AQA 2e



# HW Change for Forwarding

Figure 3.20, Page 161, CA:AQA 2e

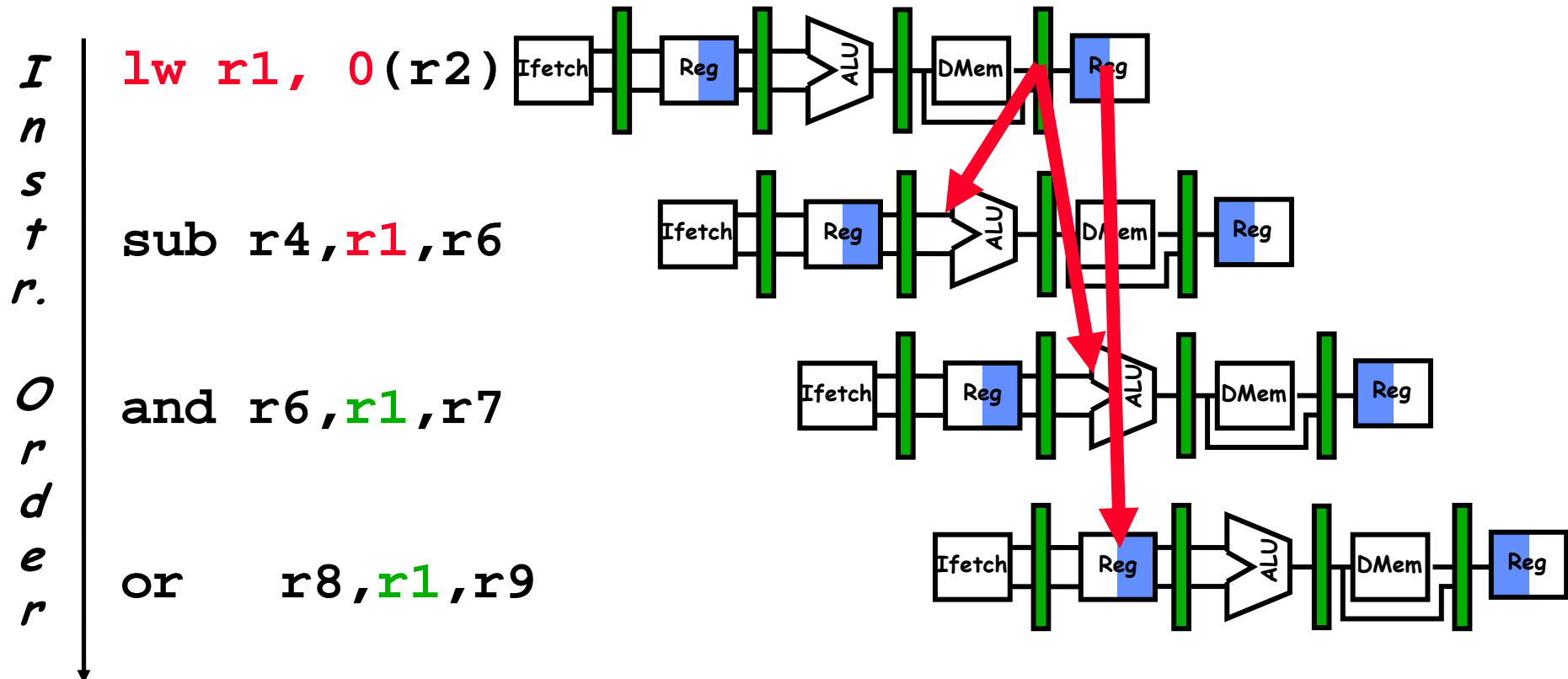




# Data Hazard Even with Forwarding

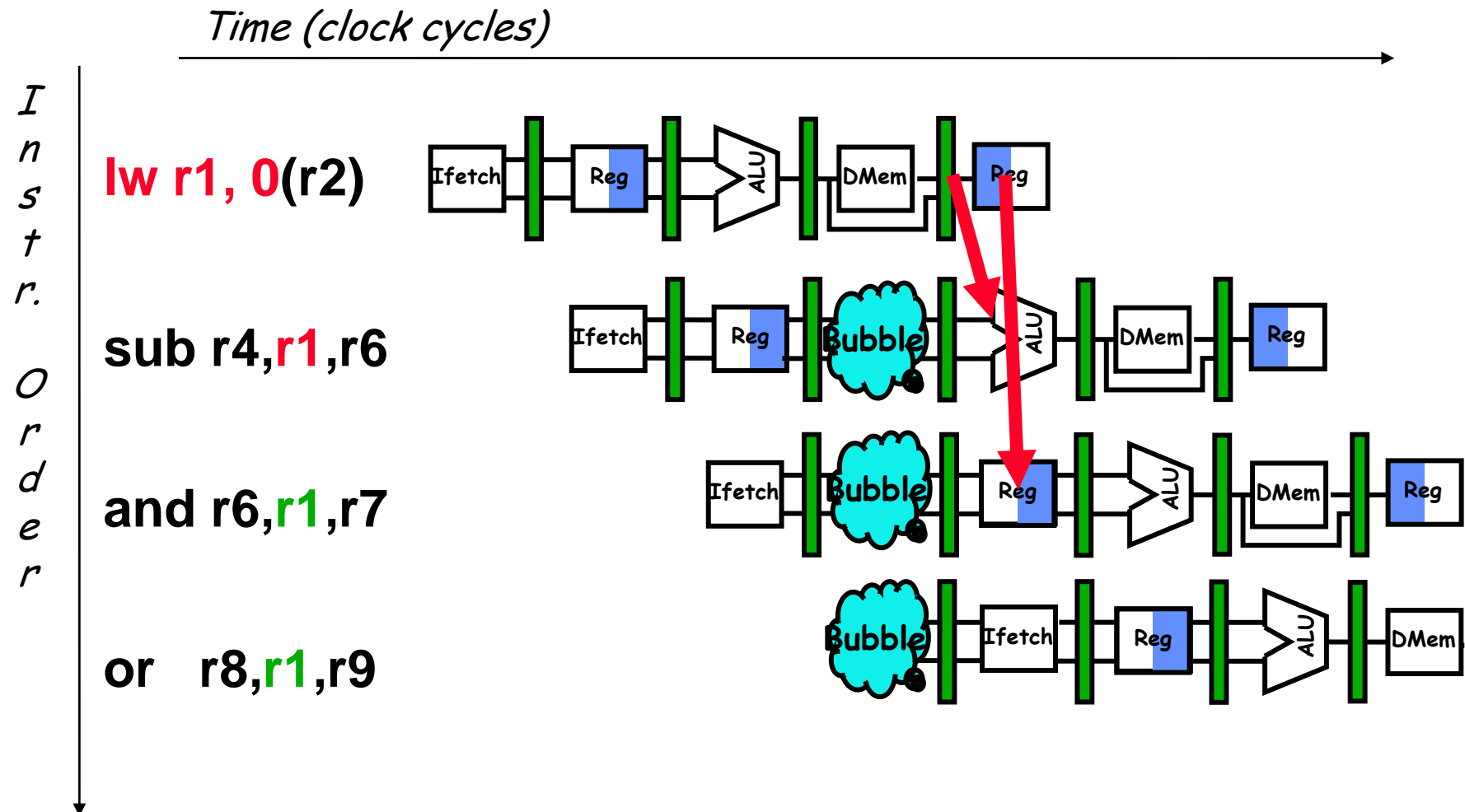
Figure 3.12, Page 153 , CA:AQA 2e

Time (clock cycles)



# Data Hazard Even with Forwarding

Figure 3.13, Page 154 , CA:AQA 2e



# Software Scheduling to Avoid Load Hazards

Try producing fast code for

$a = b + c;$

$d = e - f;$

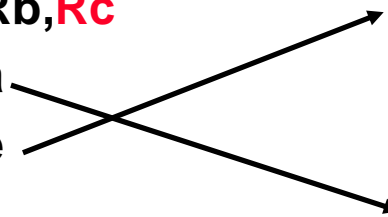
assuming a, b, c, d, e, and f in memory.

Slow code:

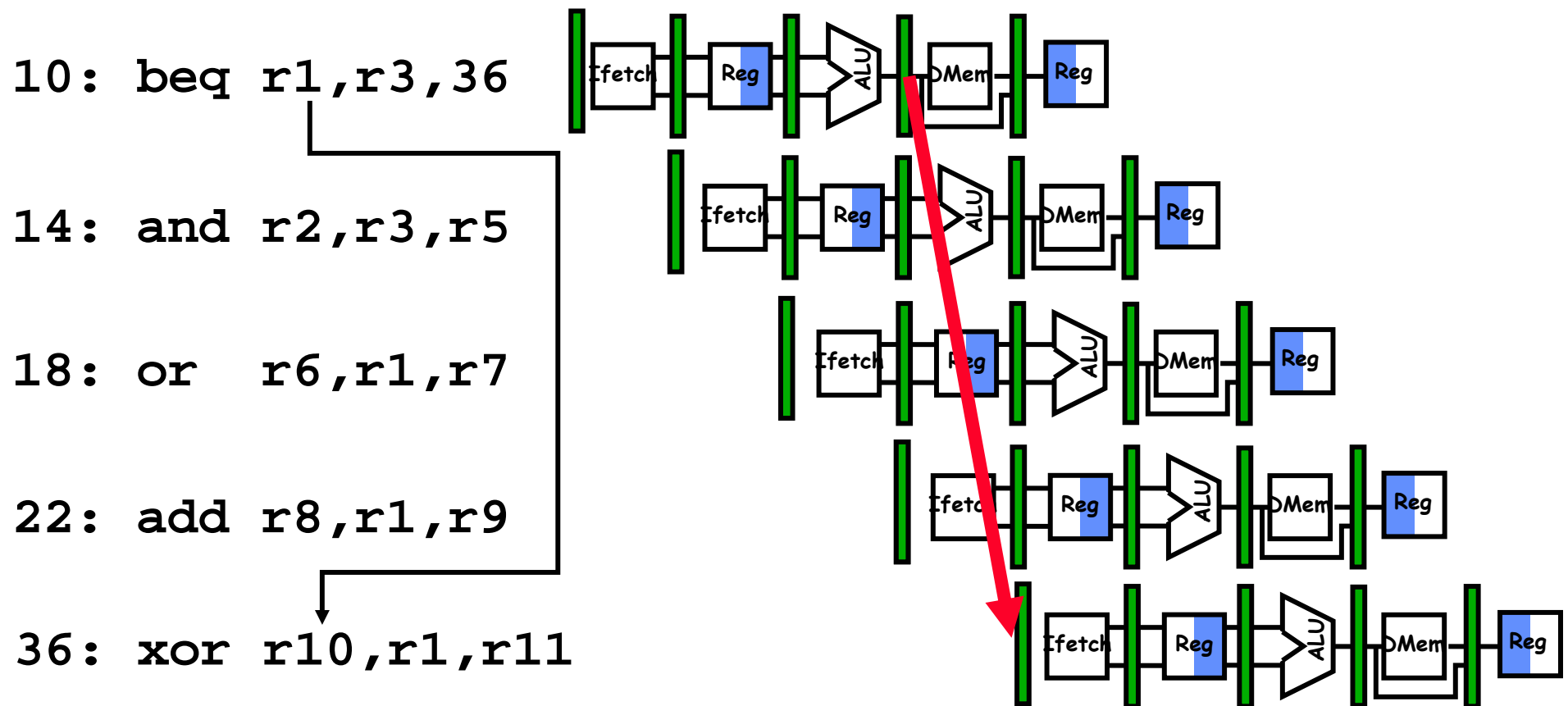
```
LW    Rb,b
LW    Rc,c
ADD   Ra,Rb,Rc
SW    a,Ra
LW    Re,e
LW    Rf,f
SUB   Rd,Re,Rf
SW    d,Rd
```

Fast code:

```
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
```



## Control Hazard on Branches Three Stage Stall

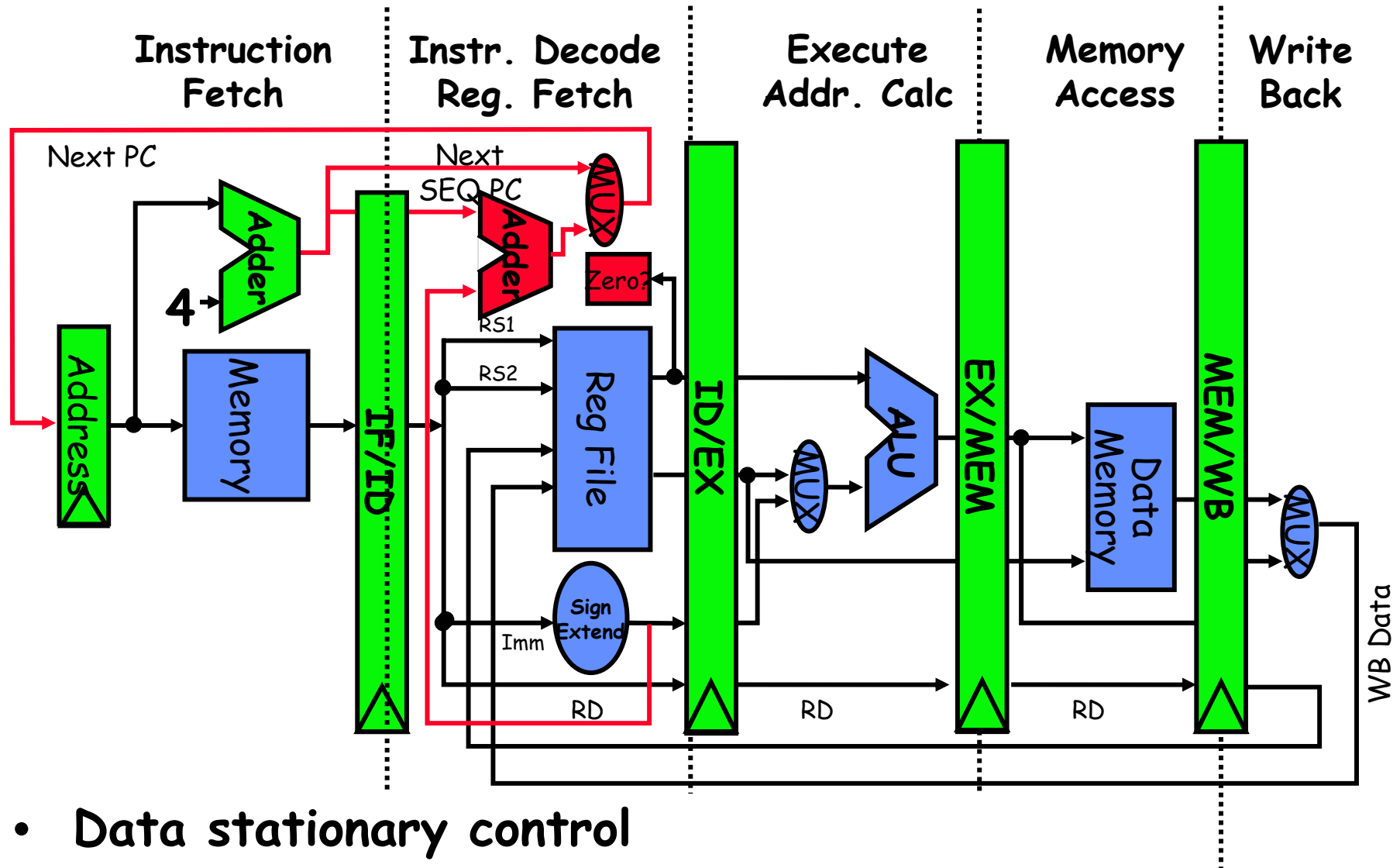


## Example: Branch Stall Impact

- If 30% branch, Stall 3 cycles significant
- Two part solution:
  - Determine branch taken or not sooner, AND
  - Compute taken branch address earlier
- MIPS branch tests if register = 0 or  $\neq$  0
- MIPS Solution:
  - Move Zero test to ID/RF stage
  - Adder to calculate new PC in ID/RF stage
  - 1 clock cycle penalty for branch versus 3

# Pipelined MIPS Datapath

Figure 3.22, page 163, CA:AQA 2/e



- Data stationary control
  - local decode for each instruction phase / pipeline stage

# Four 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

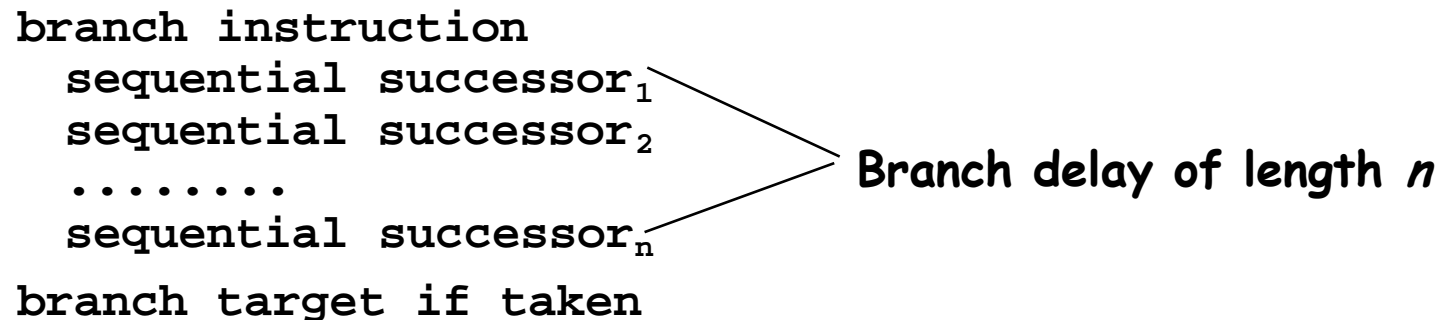
**#3: Predict Branch Taken**

- 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

# Four Branch Hazard Alternatives

## #4: Delayed Branch

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



- 1 slot delay allows proper decision and branch target address in 5 stage pipeline
- MIPS uses this



# Delayed Branch

- 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
  - Canceling branches allow more slots to be filled
- Compiler effectiveness for single branch delay slot:
  - Fills about 60% of branch delay slots
  - About 80% of instructions executed in branch delay slots useful in computation
  - About 50% ( $60\% \times 80\%$ ) of slots usefully filled
- Delayed Branch downside: 7-8 stage pipelines, multiple instructions issued per clock (superscalar)

# Now, Review of Performance

## Which is faster?

Plane	DC to Paris	Speed	Passengers	Throughput (pmph)
Boeing 747	6.5 hours	610 mph	470	286,700
BAD/Sud Concorde	3 hours	1350 mph	132	178,200

- Time to run the task (ExTime)
  - Execution time, response time, latency
- Tasks per day, hour, week, sec, ns ... (Performance)
  - Throughput, bandwidth

# Definitions

- Performance is in units of things per sec
  - bigger is better
- If we are primarily concerned with response time

$$\text{performance}(x) = \frac{1}{\text{execution\_time}(x)}$$

"X is n times faster than Y" means

$$n = \frac{\text{Performance}(X)}{\text{Performance}(Y)} = \frac{\text{Execution\_time}(Y)}{\text{Execution\_time}(X)}$$

# Aspects of CPU Performance (CPU Law)

$$\text{CPU time} = \frac{\text{Seconds}}{\text{Program}} = \frac{\text{Instructions}}{\text{Program}} \times \frac{\text{Cycles}}{\text{Instruction}} \times \frac{\text{Seconds}}{\text{Cycle}}$$

	Inst Count	CPI	Clock Rate
<b>Program</b>	X		
<b>Compiler</b>	X	(X)	
<b>Inst. Set.</b>	X	X	
<b>Organization</b>		X	X
<b>Technology</b>			X

# Cycles Per Instruction (Throughput)

## “Average Cycles per Instruction”

$$\begin{aligned}\text{CPI} &= (\text{CPU Time} * \text{Clock Rate}) / \text{Instruction Count} \\ &= \text{Cycles} / \text{Instruction Count}\end{aligned}$$

$$\text{CPU time} = \text{Cycle Time} \times \sum_{j=1}^n \text{CPI}_j \times I_j$$

## “Instruction Frequency”

$$\text{CPI} = \sum_{j=1}^n \text{CPI}_j \times F_j \quad \text{where } F_j = \frac{I_j}{\text{Instruction Count}}$$

## Example: Calculating CPI

Base Machine (Reg / Reg)

Op	Freq	Cycles	CPI(i)	(% Time)
ALU	50%	1	.5	(33%)
Load	20%	2	.4	(27%)
Store	10%	2	.2	(13%)
Branch	20%	2	.4	(27%)
			<hr/> 1.5	

Typical Mix of  
instruction types  
in program

## Example: Branch Stall Impact

- Assume  $CPI = 1.0$  ignoring branches
  - Assume solution was stalling for 3 cycles
  - If 30% branch, Stall 3 cycles
- | Op     | Freq | Cycles | $CPI(i)$ | (% Time) |
|--------|------|--------|----------|----------|
| Other  | 70%  | 1      | .7       | (37%)    |
| Branch | 30%  | 4      | 1.2      | (63%)    |
- $\Rightarrow$  new  $CPI = 1.9$ , or almost 2 times slower



## Example 2: Speed Up Equation for Pipelining

$$CPI_{\text{pipelined}} = \text{Ideal CPI} + \text{Average Stall cycles per Inst}$$

$$\text{Speedup} = \frac{\text{Ideal CPI} \times \text{Pipeline depth}}{\text{Ideal CPI} + \text{Pipeline stall CPI}} \times \frac{\text{Cycle Time}_{\text{unpipelined}}}{\text{Cycle Time}_{\text{pipelined}}}$$

For simple RISC pipeline,  $CPI = 1$ :

$$\text{Speedup} = \frac{\text{Pipeline depth}}{1 + \text{Pipeline stall CPI}} \times \frac{\text{Cycle Time}_{\text{unpipelined}}}{\text{Cycle Time}_{\text{pipelined}}}$$

## Example 3: Evaluating Branch Alternatives (for 1 program)

$$\text{Pipeline speedup} = \frac{\text{Pipeline depth}}{1 + \text{Branch frequency} \times \text{Branch penalty}}$$

<i>Scheduling scheme</i>	<i>Branch penalty</i>	<i>CPI</i>	<i>speedup v. stall</i>
Stall pipeline	3	1.42	1.0
Predict taken	1	1.14	1.26
Predict not taken	1	1.09	1.29
Delayed branch	0.5	1.07	1.31

Conditional & Unconditional = 14%, 65% change PC

## Example 4: Dual-port vs. Single-port

- Machine A: Dual ported memory (“Harvard Architecture”)
- Machine B: Single ported memory, but its pipelined implementation has a 1.05 times faster clock rate
- Ideal CPI = 1 for both
- Loads are 40% of instructions executed

$$\begin{aligned}\text{SpeedUp}_A &= \text{Pipeline Depth} / (1 + 0) \times (\text{clock}_{\text{unpipe}} / \text{clock}_{\text{pipe}}) \\ &= \text{Pipeline Depth}\end{aligned}$$

$$\begin{aligned}\text{SpeedUp}_B &= \text{Pipeline Depth} / (1 + 0.4 \times 1) \times (\text{clock}_{\text{unpipe}} / (\text{clock}_{\text{unpipe}} / 1.05)) \\ &= (\text{Pipeline Depth} / 1.4) \times 1.05 \\ &= 0.75 \times \text{Pipeline Depth}\end{aligned}$$

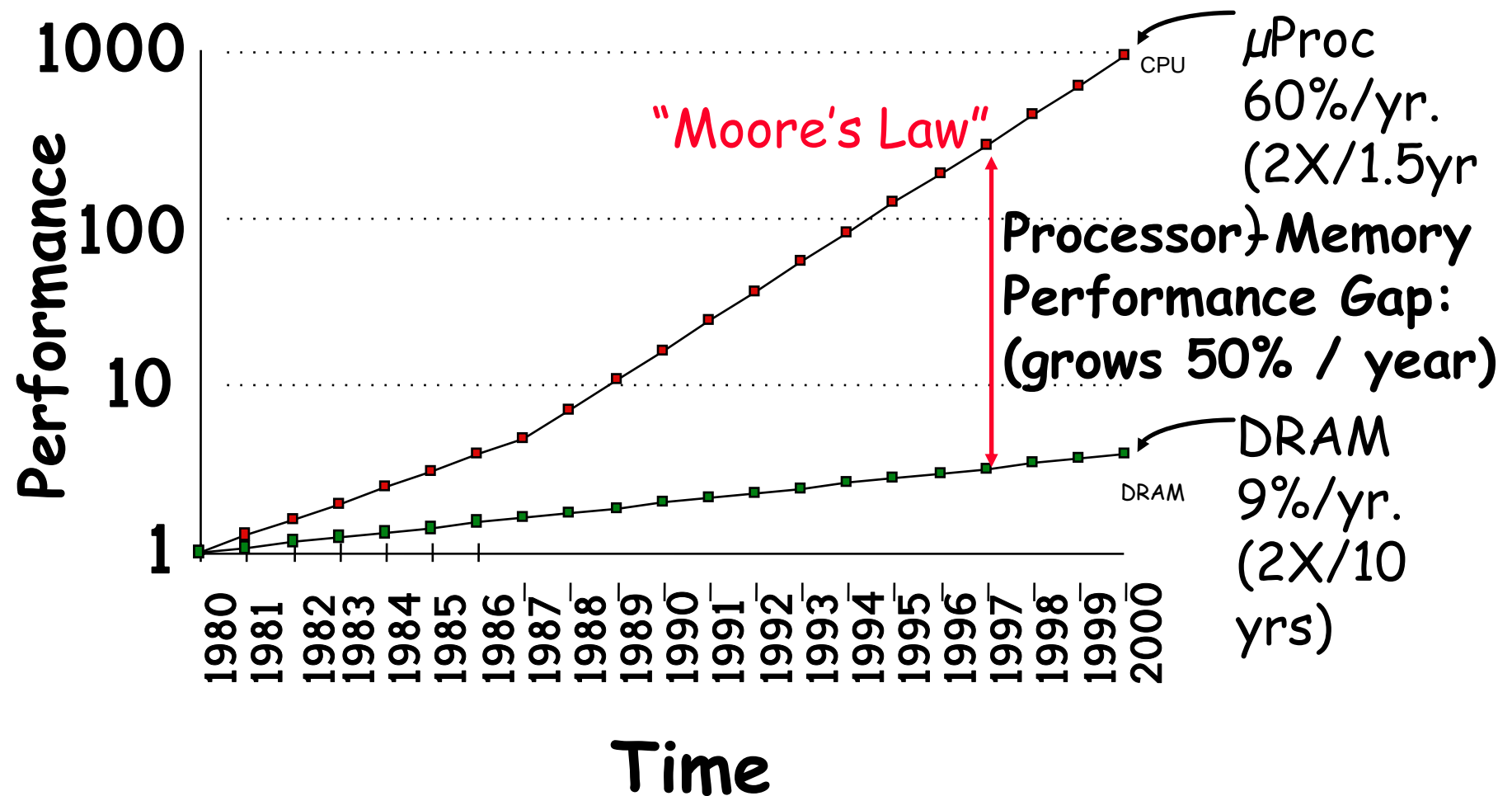
$$\text{SpeedUp}_A / \text{SpeedUp}_B = \text{Pipeline Depth} / (0.75 \times \text{Pipeline Depth}) = 1.33$$

- Machine A is 1.33 times faster

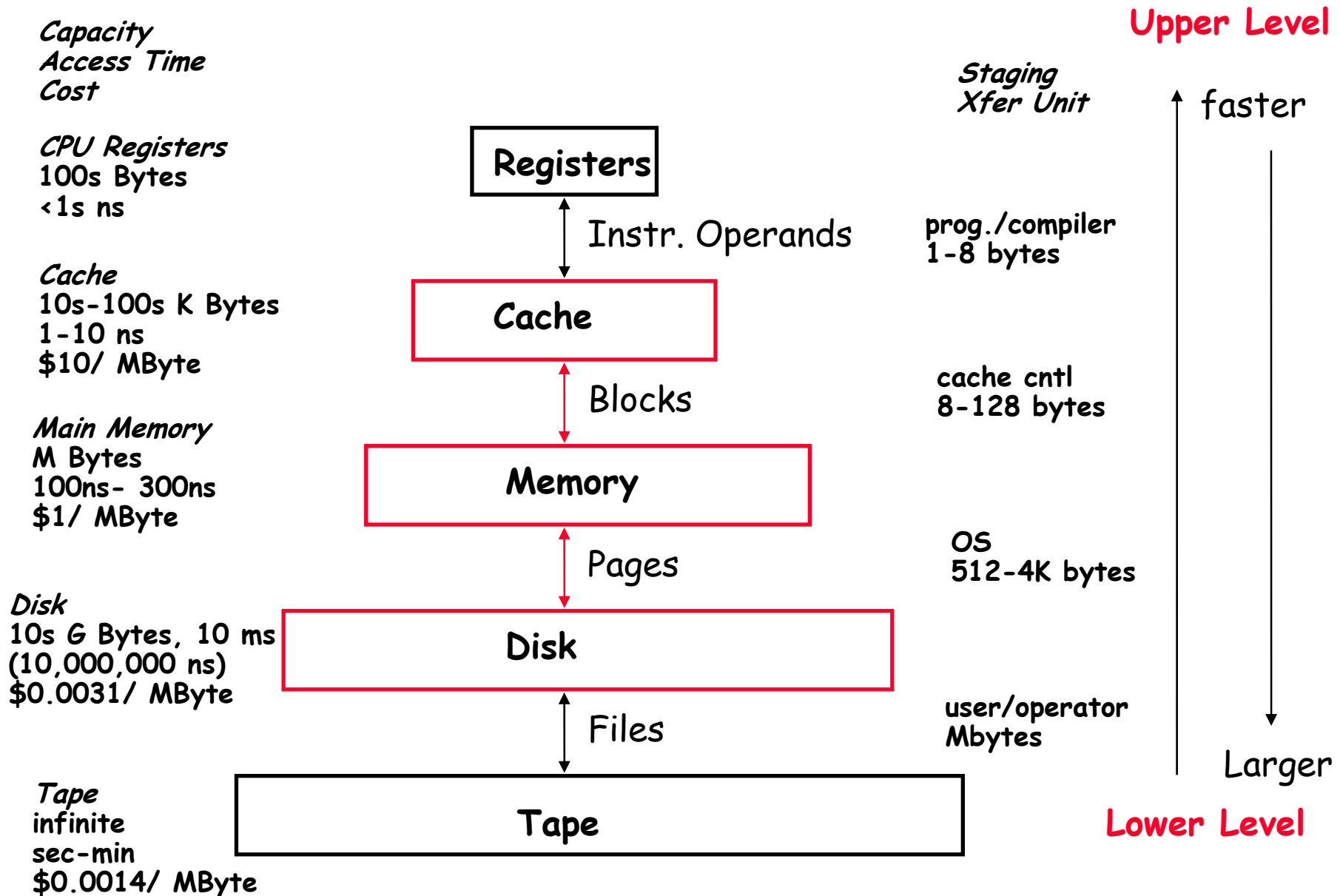
# Now, Review of Memory Hierarchy

## Recap: Who Cares About the Memory Hierarchy?

### Processor-DRAM Memory Gap (latency)



# Levels of the Memory Hierarchy

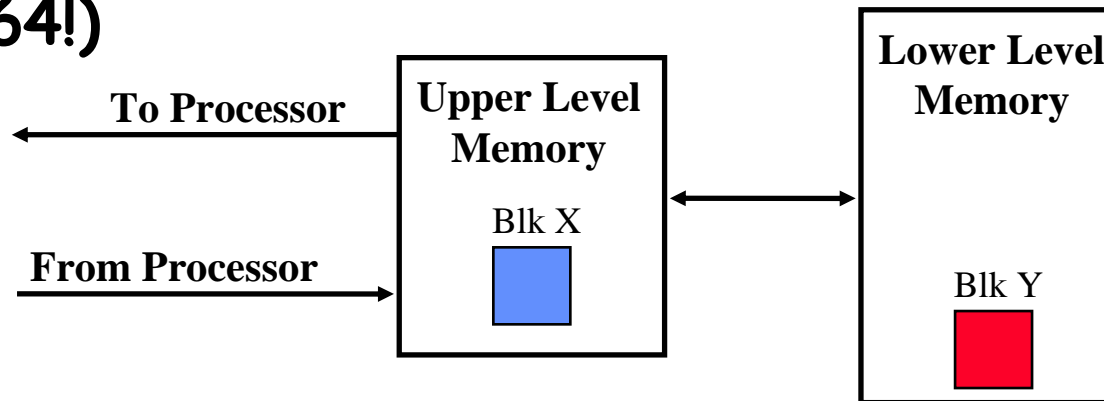


# The Principle of Locality

- The Principle of Locality:
  - Program access a relatively small portion of the address space at any instant of time.
- Two Different Types of Locality:
  - Temporal Locality (Locality in Time): If an item is referenced, it will tend to be referenced again soon (e.g., loops, reuse)
  - Spatial Locality (Locality in Space): If an item is referenced, items whose addresses are close by tend to be referenced soon (e.g., straightline code, array access)
- Last 15 years, HW (hardware) relied on locality for speed

# Memory Hierarchy: Terminology

- **Hit**: data appears in some block in the upper level (example: Block X)
  - **Hit Rate**: the fraction of memory access found in the upper level
  - **Hit Time**: Time to access the upper level which consists of  
RAM access time + Time to determine hit/miss
- **Miss**: data needs to be retrieve from a block in the lower level (Block Y)
  - **Miss Rate** =  $1 - (\text{Hit Rate})$
  - **Miss Penalty**: Time to replace a block in the upper level +  
Time to deliver the block the processor
- **Hit Time** << **Miss Penalty** (500 instructions on 21264!)





# Cache Measures

- *Hit rate*: fraction found in that level
  - So high that usually talk about *Miss rate*
  - Miss rate fallacy: as MIPS to CPU performance, miss rate to average memory access time in memory
- Average memory-access time
  - = Hit time + Miss rate  $\times$  Miss penalty (ns or clocks)
- *Miss penalty*: time to replace a block from lower level, including time to replace in CPU
  - *access time*: time to lower level
    - = f(latency to lower level)
  - *transfer time*: time to transfer block
    - = f(BW between upper & lower levels)

# Simplest Cache: Direct Mapped

Memory Address    Memory

0	
1	
2	
3	
4	
5	
6	
7	
8	
9	
A	
B	
C	
D	
E	
F	

4 Byte Direct Mapped Cache

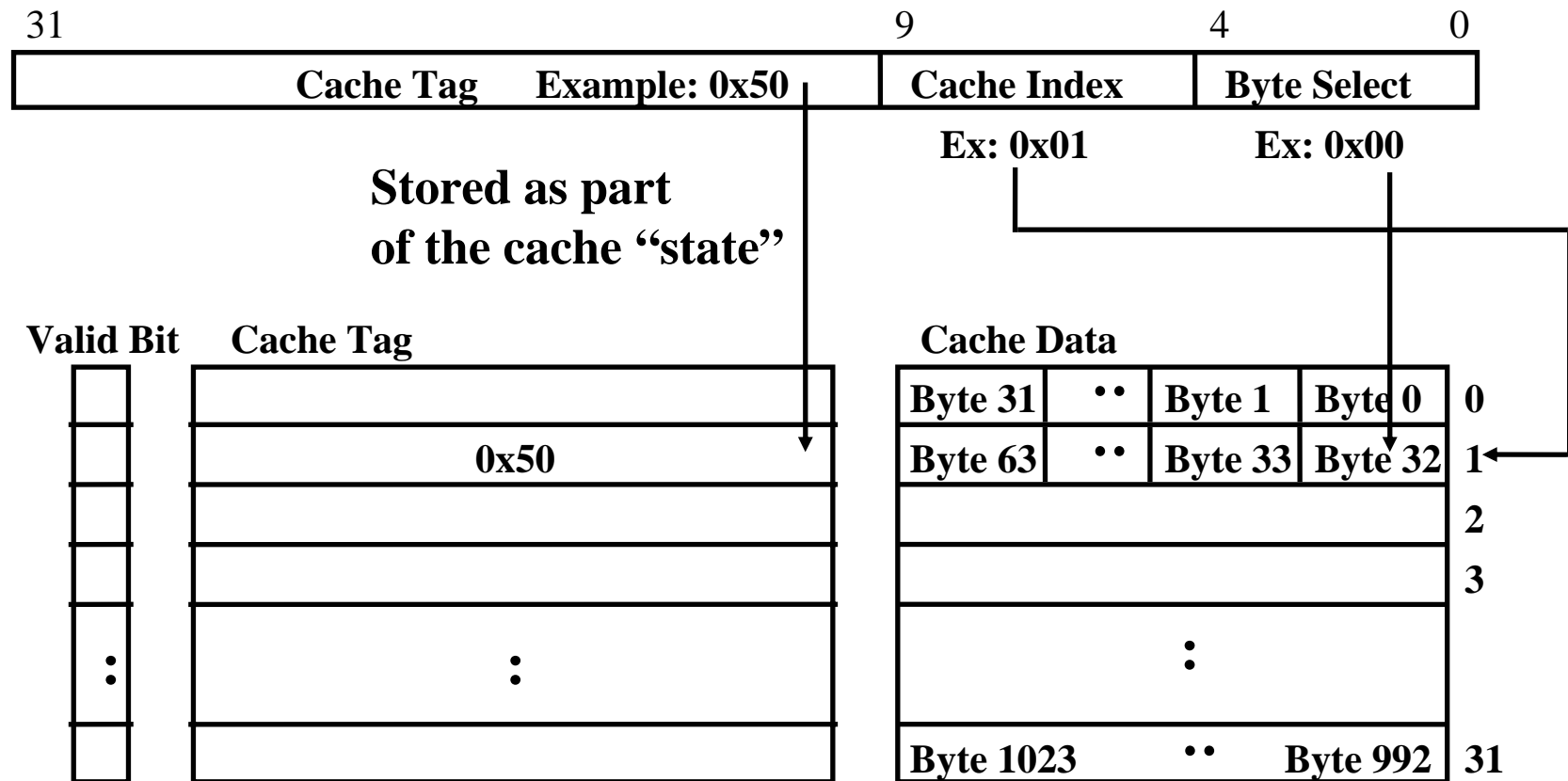
Cache Index

0	
1	
2	
3	

- Location 0 can be occupied by data from:
  - Memory location 0, 4, 8, ... etc.
  - In general: any memory location whose 2 LSBs of the address are 0s
  - $\text{Address} \langle 1:0 \rangle \Rightarrow \text{cache index}$
- Which one should we place in the cache?
- How can we tell which one is in the cache?

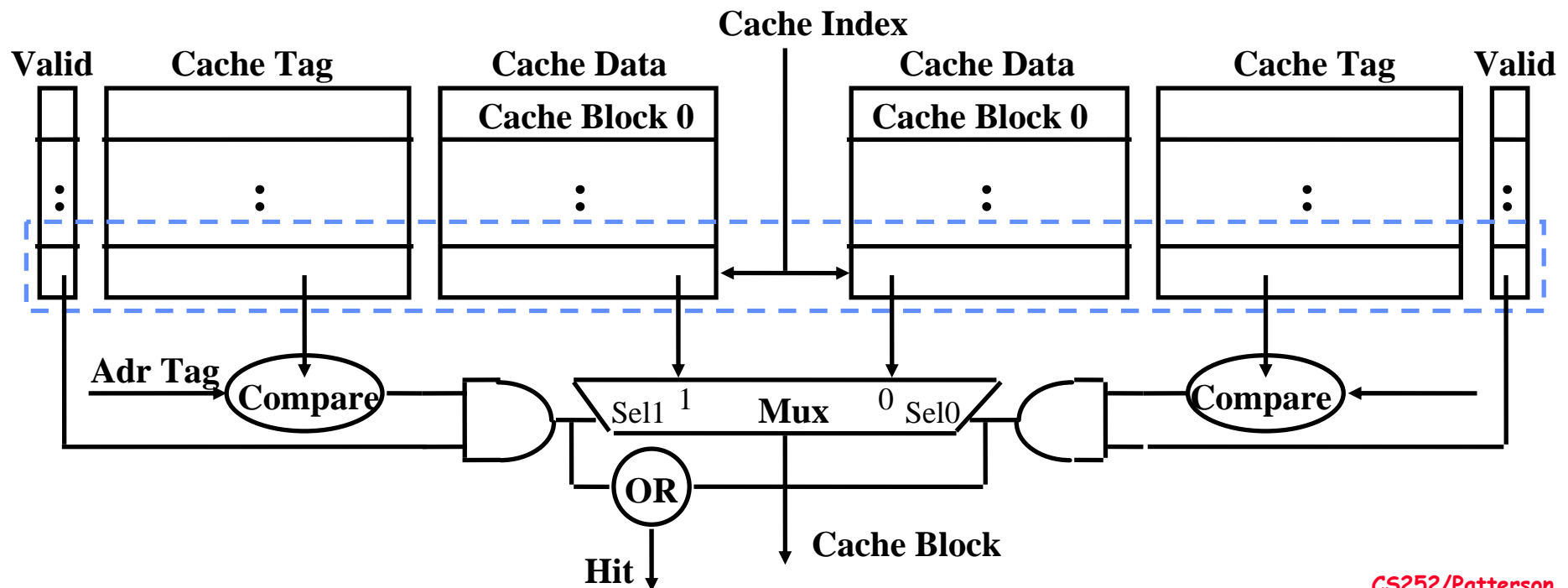
# 1 KB Direct Mapped Cache, 32B blocks

- For a  $2^N$  byte cache:
  - The uppermost  $(32 - N)$  bits are always the Cache Tag
  - The lowest  $M$  bits are the Byte Select (Block Size =  $2^M$ )



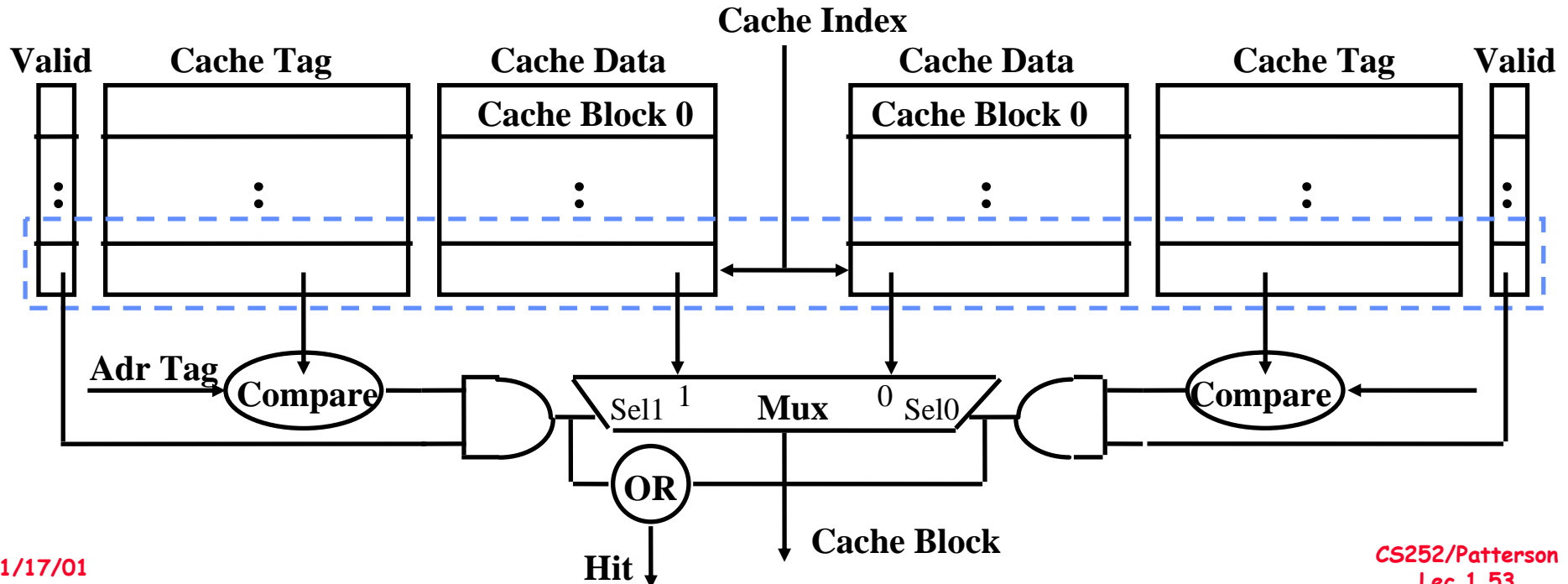
# Two-way Set Associative Cache

- N-way set associative: N entries for each Cache Index
  - N direct mapped caches operates in parallel (N typically 2 to 4)
- Example: Two-way set associative cache
  - Cache Index selects a “set” from the cache
  - The two tags in the set are compared in parallel
  - Data is selected based on the tag result



# Disadvantage of Set Associative Cache

- N-way Set Associative Cache v. Direct Mapped Cache:
  - N comparators vs. 1
  - Extra MUX delay for the data
  - Data comes AFTER Hit/Miss
- In a direct mapped cache, Cache Block is available BEFORE Hit/Miss:
  - Possible to assume a hit and continue. Recover later if miss.

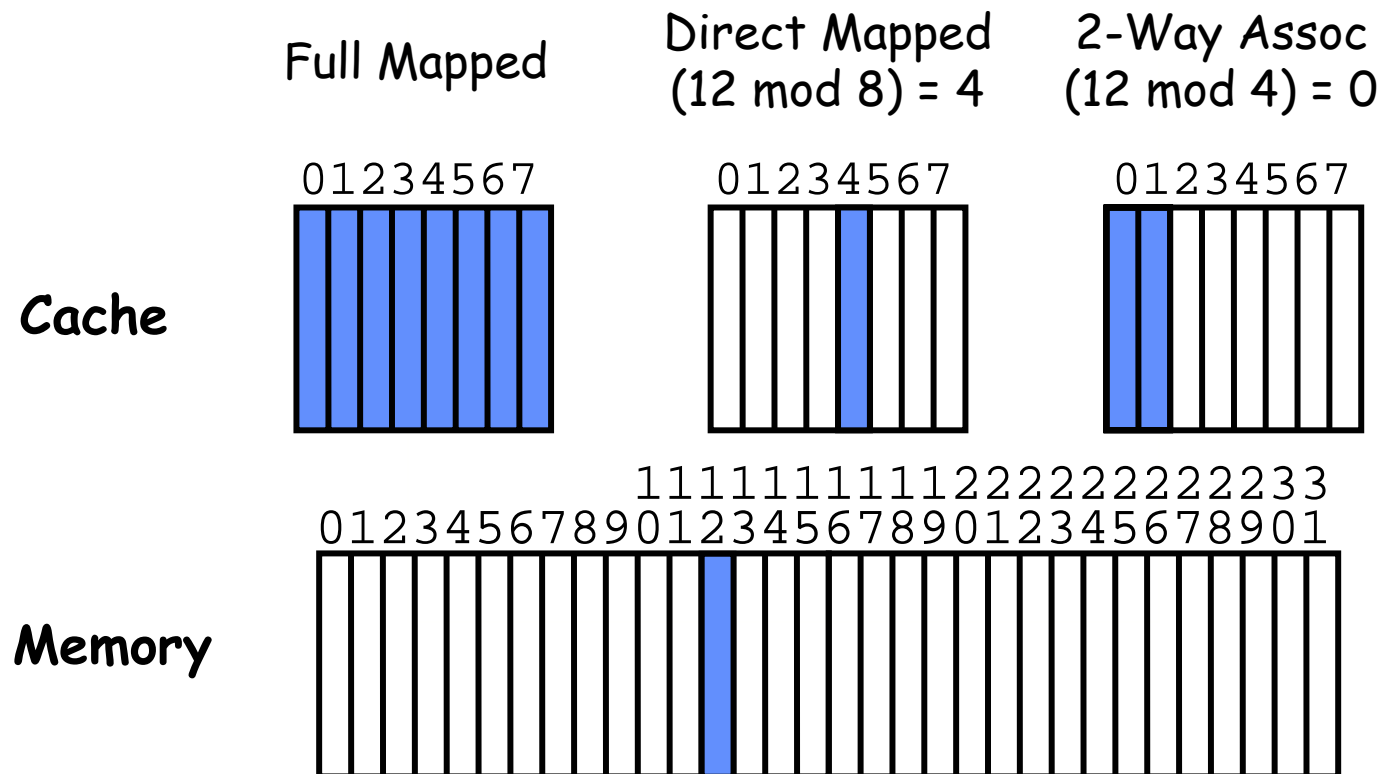


## 4 Questions for Memory Hierarchy

- Q1: Where can a block be placed in the upper level?  
*(Block placement)*
- Q2: How is a block found if it is in the upper level?  
*(Block identification)*
- Q3: Which block should be replaced on a miss?  
*(Block replacement)*
- Q4: What happens on a write?  
*(Write strategy)*

# Q1: Where can a block be placed in the upper level?

- Block 12 placed in 8 block cache:
  - Fully associative, direct mapped, 2-way set associative
  - S.A. Mapping = Block Number Modulo Number Sets



## Q2: How is a block found if it is in the upper level?

- Tag on each block
  - No need to check index or block offset
- Increasing associativity shrinks index, → expands tag →

Block Address		Block Offset
Tag	Index	



## Q3: Which block should be replaced on a miss?

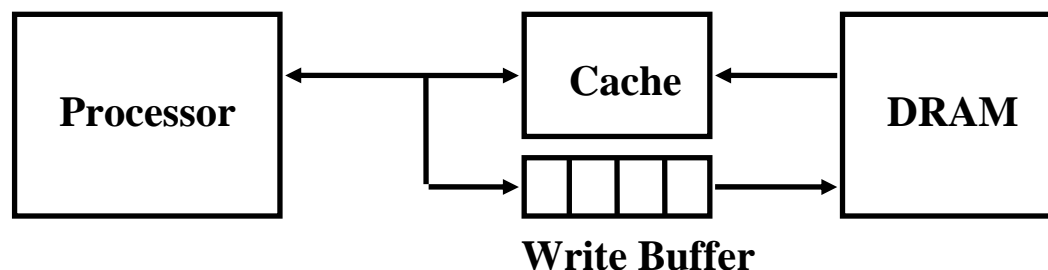
- Easy for Direct Mapped
- Set Associative or Fully Associative:
  - Random
  - LRU (Least Recently Used)

Assoc:	2-way		4-way		8-way	
Size	LRU	Ran	LRU	Ran	LRU	Ran
16 KB	5.2%	5.7%	4.7%	5.3%	4.4%	5.0%
64 KB	1.9%	2.0%	1.5%	1.7%	1.4%	1.5%
256 KB	1.15%	1.17%	1.13%	1.13%	1.12%	1.12%

## Q4: What happens on a write?

- Write through—The information is written to both the block in the cache and to the block in the lower-level memory.
- Write back—The information is written only to the block in the cache. The modified cache block is written to main memory only when it is replaced.
  - is block clean or dirty?
- Pros and Cons of each?
  - WT: read misses cannot result in writes
  - WB: no repeated writes to same location
- WT always combined with write buffers so that don't wait for lower level memory

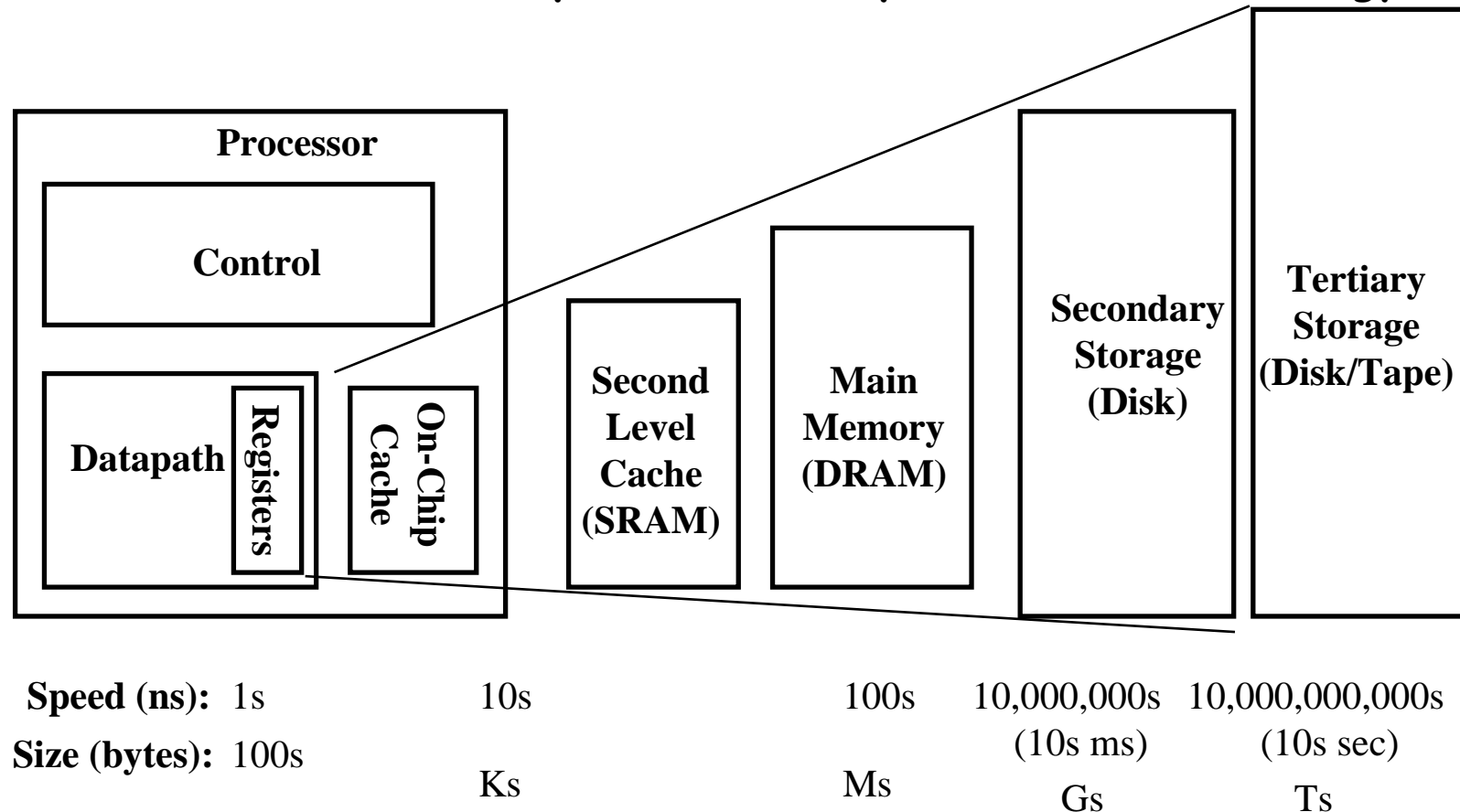
# Write Buffer for Write Through



- **A Write Buffer is needed between the Cache and Memory**
  - Processor: writes data into the cache and the write buffer
  - Memory controller: write contents of the buffer to memory
- **Write buffer is just a FIFO:**
  - Typical number of entries: 4
  - Works fine if: Store frequency (w.r.t. time)  $\ll 1 / \text{DRAM write cycle}$
- **Memory system designer's nightmare:**
  - Store frequency (w.r.t. time)  $\rightarrow 1 / \text{DRAM write cycle}$
  - Write buffer saturation

# A Modern Memory Hierarchy

- By taking advantage of the principle of locality:
  - Present the user with as much memory as is available in the cheapest technology.
  - Provide access at the speed offered by the fastest technology.



## Summary #1/4: Pipelining & Performance

- Just overlap tasks; easy if tasks are independent
- Speed Up  $\leq$  Pipeline Depth; if ideal CPI is 1, then:

$$\text{Speedup} = \frac{\text{Pipeline depth}}{1 + \text{Pipeline stall CPI}} \times \frac{\text{Cycle Time}_{\text{unpipelined}}}{\text{Cycle Time}_{\text{pipelined}}}$$

- Hazards limit performance on computers:
  - Structural: need more HW resources
  - Data (RAW, WAR, WAW): need forwarding, compiler scheduling
  - Control: delayed branch, prediction
- Time is measure of performance: latency or throughput
- CPI Law:

CPU time	=	$\frac{\text{Seconds}}{\text{Program}}$	=	$\frac{\text{Instructions}}{\text{Program}}$	x	$\frac{\text{Cycles}}{\text{Instruction}}$	x	$\frac{\text{Seconds}}{\text{Cycle}}$
----------	---	---	---	--	---	--	---	---------------------------------------

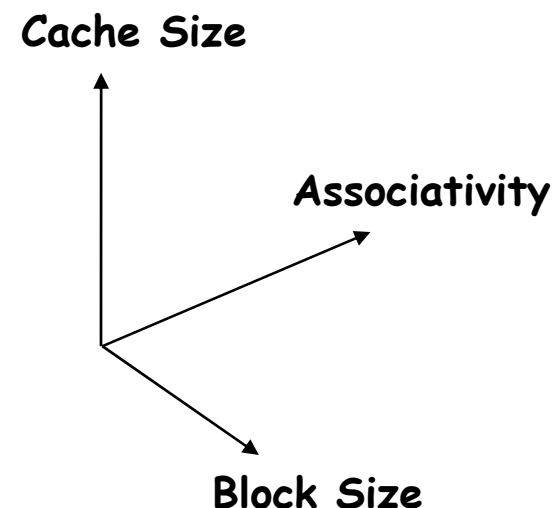
## Summary #2/4: Caches

- The Principle of Locality:
  - Program access a relatively small portion of the address space at any instant of time.
    - » Temporal Locality: Locality in Time
    - » Spatial Locality: Locality in Space
- Three Major Categories of Cache Misses:
  - Compulsory Misses: sad facts of life. Example: cold start misses.
  - Capacity Misses: increase cache size
  - Conflict Misses: increase cache size and/or associativity.
- Write Policy:
  - Write Through: needs a write buffer.
  - Write Back: control can be complex
- Today CPU time is a function of (ops, cache misses) vs. just  $f(\text{ops})$ : What does this mean to Compilers, Data structures, Algorithms?

## Summary #3/4: The Cache Design Space

- Several interacting dimensions

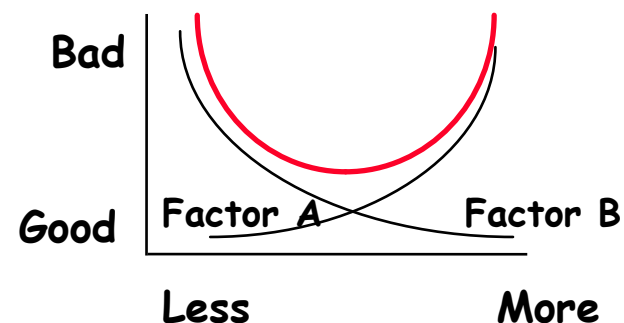
- cache size
- block size
- associativity
- replacement policy
- write-through vs write-back



- The optimal choice is a compromise

- depends on access characteristics
  - » workload
  - » use (I-cache, D-cache, TLB)
- depends on technology / cost

- Simplicity often wins



## Review #4/4: TLB, Virtual Memory

- Caches, TLBs, Virtual Memory all understood by examining how they deal with 4 questions: 1) Where can block be placed? 2) How is block found? 3) What block is replaced on miss? 4) How are writes handled?
- Page tables map virtual address to physical address
- TLBs make virtual memory practical
  - Locality in data => locality in addresses of data, temporal and spatial
- TLB misses are significant in processor performance
  - funny times, as most systems can't access all of 2nd level cache without TLB misses!
- Today VM allows many processes to share single memory without having to swap all processes to disk; today VM protection is more important than memory hierarchy