
CSE 502 Graduate Computer Architecture

Lec 3-5 – Performance + Instruction Pipelining Review

Larry Wittie

Computer Science, StonyBrook University

<http://www.cs.sunysb.edu/~cse502> and [~lw](http://www.cs.sunysb.edu/~lw)

Slides adapted from David Patterson, UC-Berkeley cs252-s06

Review from last lecture

- **Tracking and extrapolating technology part of architect's responsibility**
- **Expect Bandwidth in disks, DRAM, network, and processors to improve by at least as much as the square of the improvement in Latency**
- **Quantify Cost (vs. Price)**
 - $IC \approx f(\text{Area}^2) + \text{Learning curve, volume, commodity, margins}$
- **Quantify dynamic and static power**
 - $\text{Capacitance} \times \text{Voltage}^2 \times \text{frequency}$, Energy vs. power
- **Quantify dependability**
 - Reliability (MTTF vs. FIT), Availability ($\text{MTTF}/(\text{MTTF}+\text{MTTR})$)

Outline

- **Review**
- **F&P: Benchmarks age, disks fail, singlepoint fail danger**
- **502 Administrivia**
- **MIPS – An ISA for Pipelining**
- **5 stage pipelining**
- **Structural and Data Hazards**
- **Forwarding**
- **Branch Schemes**
- **Exceptions and Interrupts**
- **Conclusion**

Fallacies and Pitfalls (1/2)

- **Fallacies** - commonly held misconceptions
 - When discussing a fallacy, we try to give a counterexample.
- **Pitfalls** - easily made mistakes.
 - Often generalizations of principles true in limited context
 - Show Fallacies and Pitfalls to help you avoid these errors
- **Fallacy: Benchmarks remain valid indefinitely**
 - Once a benchmark becomes popular, there is tremendous pressure to improve performance by targeted optimizations or by aggressive interpretation of the rules for running the benchmark: “benchmarksmanship.”
 - 70 benchmarks in the 5 SPEC releases to 2000. 70% dropped from the next release because no longer useful
- **Pitfall: A single point of failure**
 - Rule of thumb for fault tolerant systems: make sure that every component was redundant so that no single component failure could bring down the whole system (e.g, power supply)

Lab rule of thumb: “Don’t buy one of anything.”

Fallacies and Pitfalls (2/2)

- **Fallacy - Rated MTTF of disks is 1,200,000 hours or ≈ 140 years, so disks practically never fail**
- **But disk lifetime is 5 years \Rightarrow replace a disk every 5 years; on average, 28 replacements, so wouldn't fail**
- **A better unit: % that fail (1.2M MTTF = 833 FIT)**
- **Fail over lifetime: if had 1000 disks for 5 years
 $= 1000 * (5 * 365 * 24) * 833 / 10^9 = 36,485,000 / 10^6 = 37$
 $= 3.7\%$ (37/1000) fail over 5 yr lifetime (1.2M hr MTTF)**
- **But this is under pristine conditions**
 - little vibration, narrow temperature range \Rightarrow no power failures
- **Real world: 3% to 6% of SCSI drives fail per year**
 - 3400 - 6800 FIT or 150,000 to 300,000 hour MTTF [Gray & van Ingen 05]
- **3% to 7% of ATA drives fail per year**
 - 3400 - 8000 FIT or 125,000 to 300,000 hour MTTF [Gray & van Ingen 05]

CSE502: Administrivia

Instructor: Prof Larry Wittie

Office/Lab: 1308 CompSci, lw AT icDOTsunysbDOTedu

Office Hours: 11:15AM - 1:45PM Tuesday; 11:15AM - 11:45AM Thursday or when 1308 door open, or by appt.

T. A.: To Be Determined

Class: Tu/Th, 2:20 - 3:40 pm 131 Earth & Space Sci

Text: *Computer Architecture: A Quantitative Approach, 4th Ed.* (Oct, 2006), ISBN 0123704901 or 978-0123704900, \$60 Amazon F09

Web page: <http://www.cs.sunysb.edu/~cse502/>

First reading assignment: Chapter 1 for today, Tuesday

Appendix A (at back of text) for Tuesday 9/8

CSE502: Administrivia

<http://www.cs.sunysb.edu/~lw/teaching/cse502/DoldF07/>

Last year's slides are in [~lw/teaching/cse502/DoldF08/](http://www.cs.sunysb.edu/~lw/teaching/cse502/DoldF08/)

DoldF07/lec01-intro.pdf

DoldF07/lec02-intro.pdf

DoldF07/lec03-pipe.pdf

DoldF07/lec04-cache.pdf

DoldF07/lec05-dynamic-sched.pdf

DoldF07/lec06-dynamic-schedB.pdf

DoldF07/lec07-ILP limits.pdf

DoldF07/lec07-limitsILP_SMT.pdf

DoldF07/lec08-SMT.pdf

DoldF07/lec09-Vector.pdf

DoldF07/lec10-Modern Vector.pdf

DoldF07/lec11-SMP.pdf

DoldF07/lec12-Snoop+MTreview.pdf

DoldF07/lec12-Snoop+MTreviewPreliminary.pdf

DoldF07/lec14-directory.pdf

DoldF07/lec16-T1 MP.pdf

DoldF07/lec17-memoryhier.pdf

DoldF07/lec18-VM memhier2.pdf

DoldF07/lec19-storage.pdf

DoldF07/lec20-review.pdf

Outline

- Review
- F&P: Benchmarks age, disks fail, single-points fail
- 502 Administrivia
- **MIPS – An ISA for Pipelining**
- **5 stage pipelining**
- **Structural and Data Hazards**
- **Forwarding**
- **Branch Schemes**
- **Exceptions and Interrupts**
- **Conclusion**

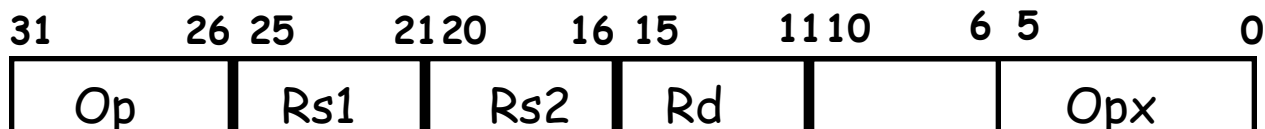
A "Typical" RISC ISA

- **32-bit fixed format instruction (3 formats)**
- **32 32-bit GPR (R0 contains zero, DP take pair)**
- **3-address, reg-reg arithmetic instruction**
- **Single address mode for load/store:
base + displacement**
 - no indirection (since it needs another memory access)
- **Simple branch conditions (e.g., single-bit: 0 or not?)**
- **(Delayed branch - ineffective in deep pipelines)**

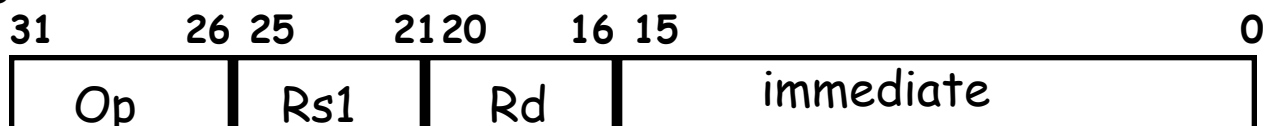
*see: SPARC, MIPS, HP PA-Risc, DEC Alpha, IBM PowerPC,
CDC 6600, CDC 7600, Cray-1, Cray-2, Cray-3*

Example: MIPS

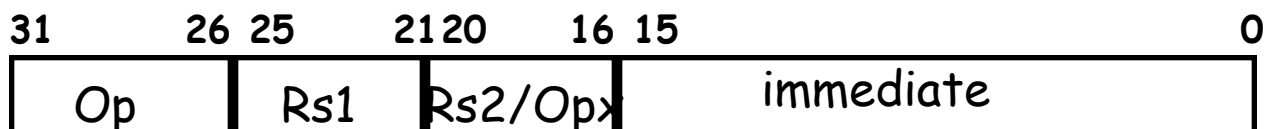
Register-Register - R Format - Arithmetic operations



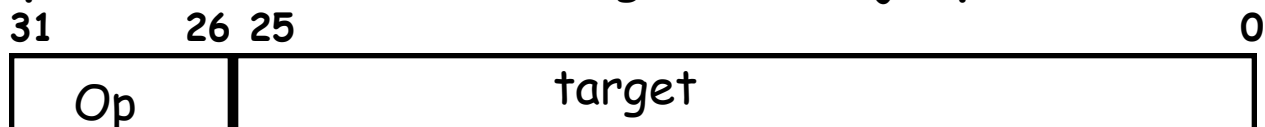
Register-Immediate - I Format - All immediate arithmetic ops



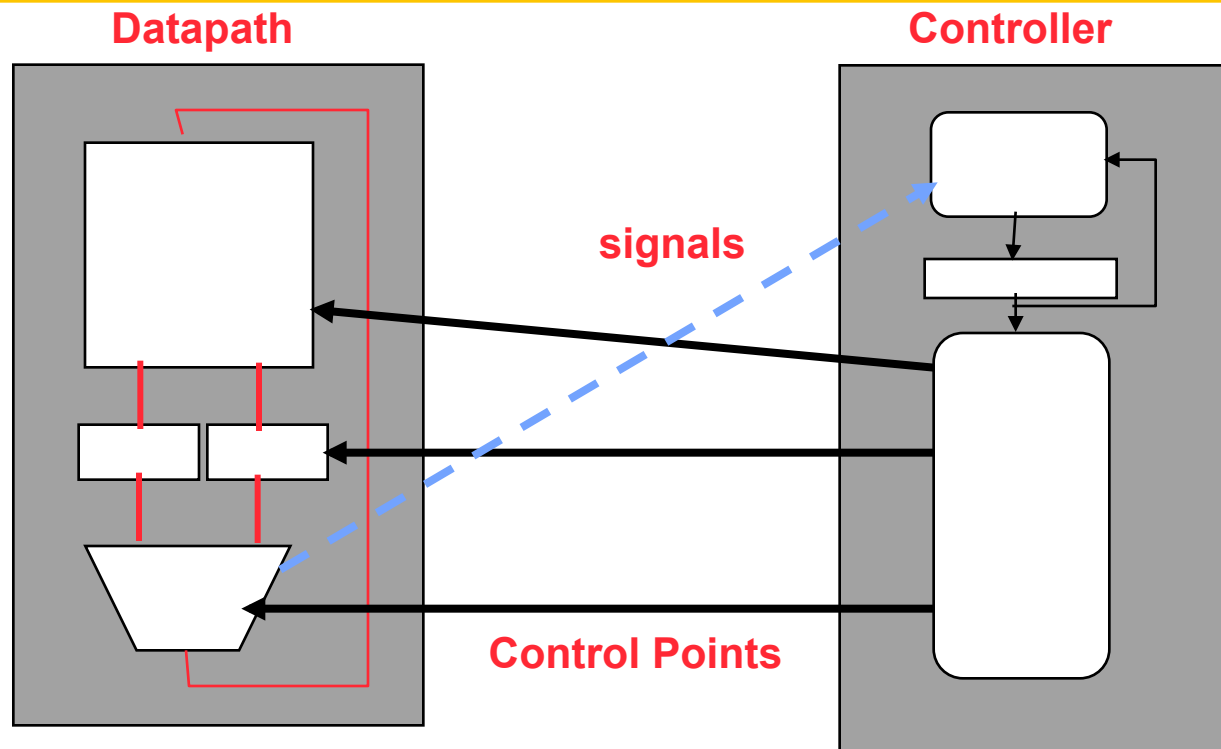
Branch - I Format - Moderate relative distance conditional branches



Jump / Call - J Format - Long distance jumps



Datapath vs Control



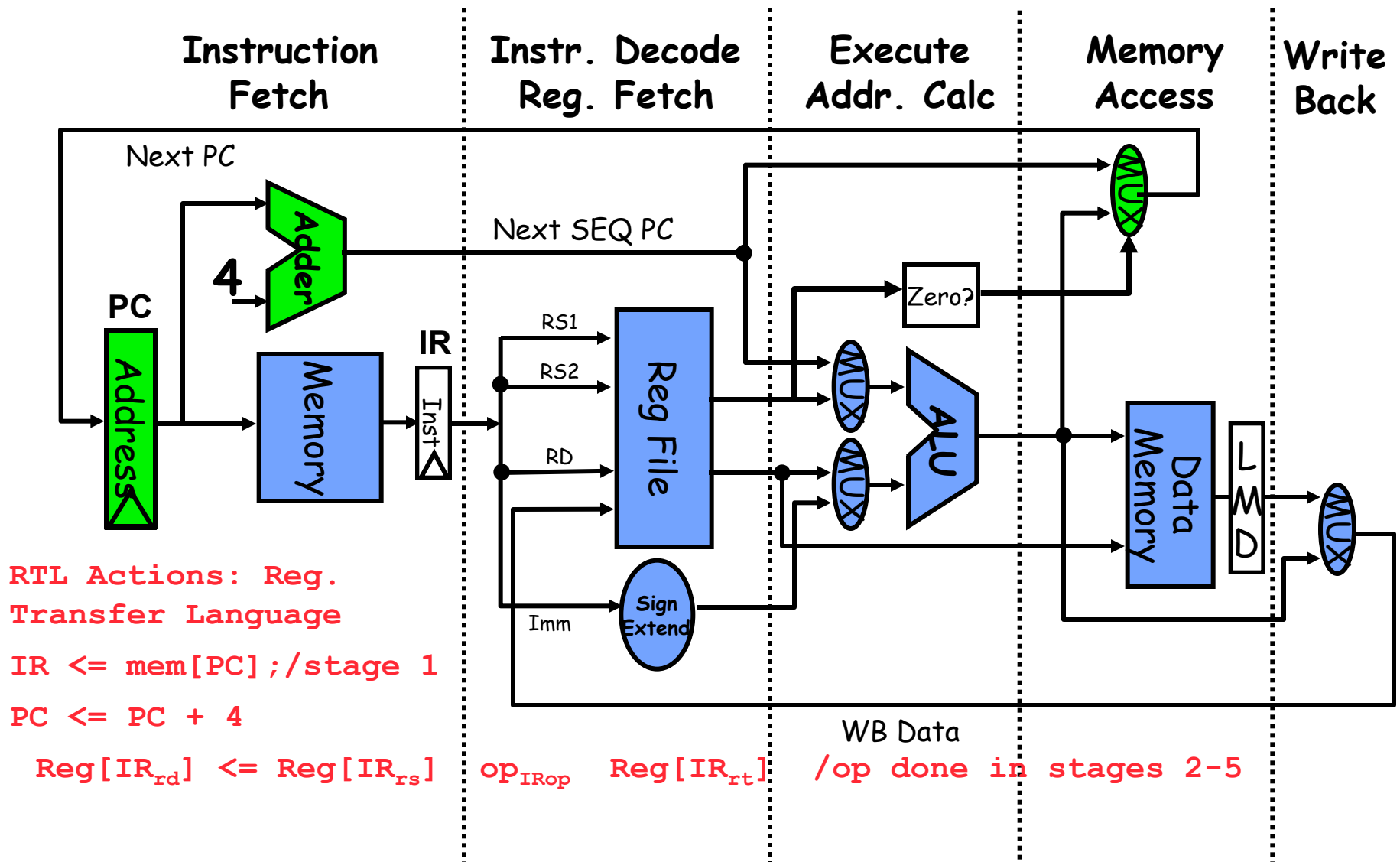
- **Datapath: Storage, Functional Units, Interconnections sufficient to perform the desired functions**
 - Inputs are Control Points
 - Outputs are signals
- **Controller: State machine to orchestrate operation on the data path**
 - Based on desired function and signals

Approaching an ISA

- **Instruction Set Architecture**
 - Defines set of operations, instruction format, hardware supported data types, named storage, addressing modes, sequencing
- **Meaning of each instruction is described by RTL (register transfer language) on *architected registers* and memory**
- **Given technology constraints, assemble adequate datapath**
 - Architected storage mapped to actual storage
 - Function Units (FUs) to do all the required operations
 - Possible additional storage (eg. Internal registers: **MAR**, **MDR**, **IR**, ... {**M**emory **A**ddress **R**egister, **M**emory **D**ata **R**egister, **I**nstruction **R**egister})
 - Interconnect to move information among registers and function units
- **Map each instruction to a sequence of RTL operations**
- **Collate sequences into symbolic controller state transition diagram (STD)**
- **Lower symbolic STD to control points**
- **Implement controller**

5 Steps of MIPS Datapath (non-pipelined)

Figure A.2, Page A-8



RTL Actions: Reg. Transfer Language

$IR \leftarrow mem[PC];$ /stage 1

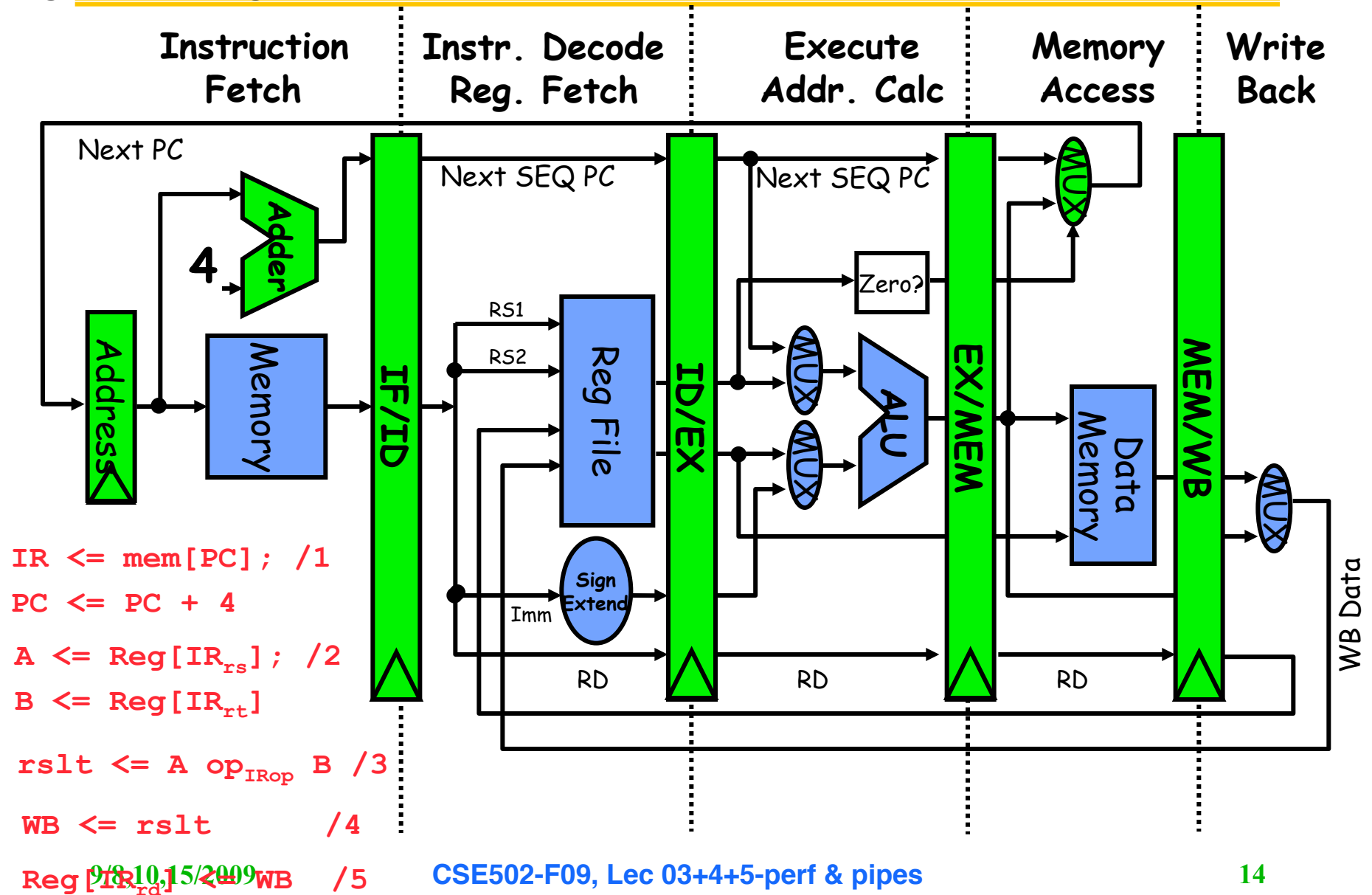
$PC \leftarrow PC + 4$

$Reg[IR_{rd}] \leftarrow Reg[IR_{rs}]$

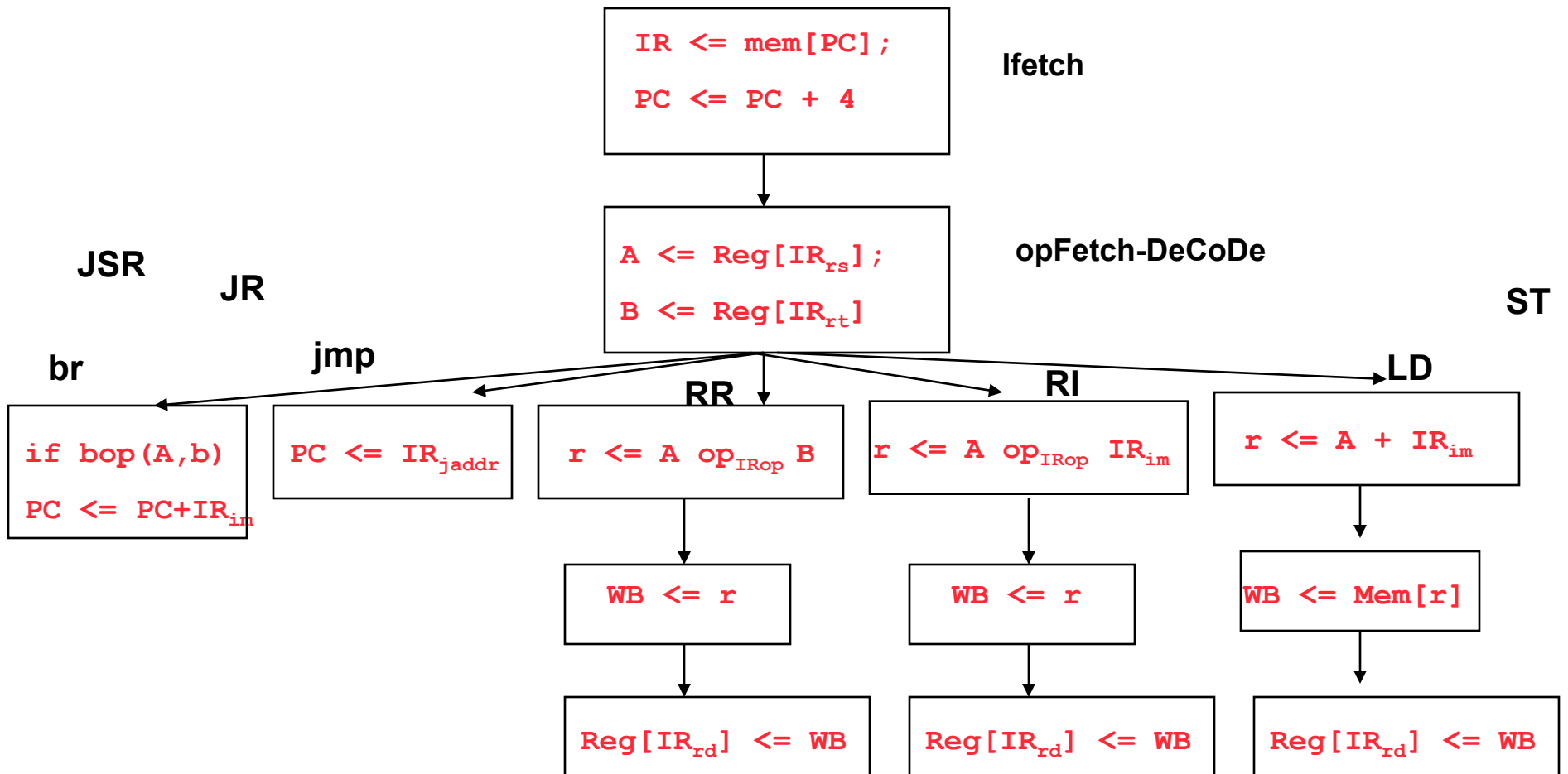
WB Data /op done in stages 2-5

5-Stage MIPS Datapath (has pipeline latches)

Figure A.3, Page A-9

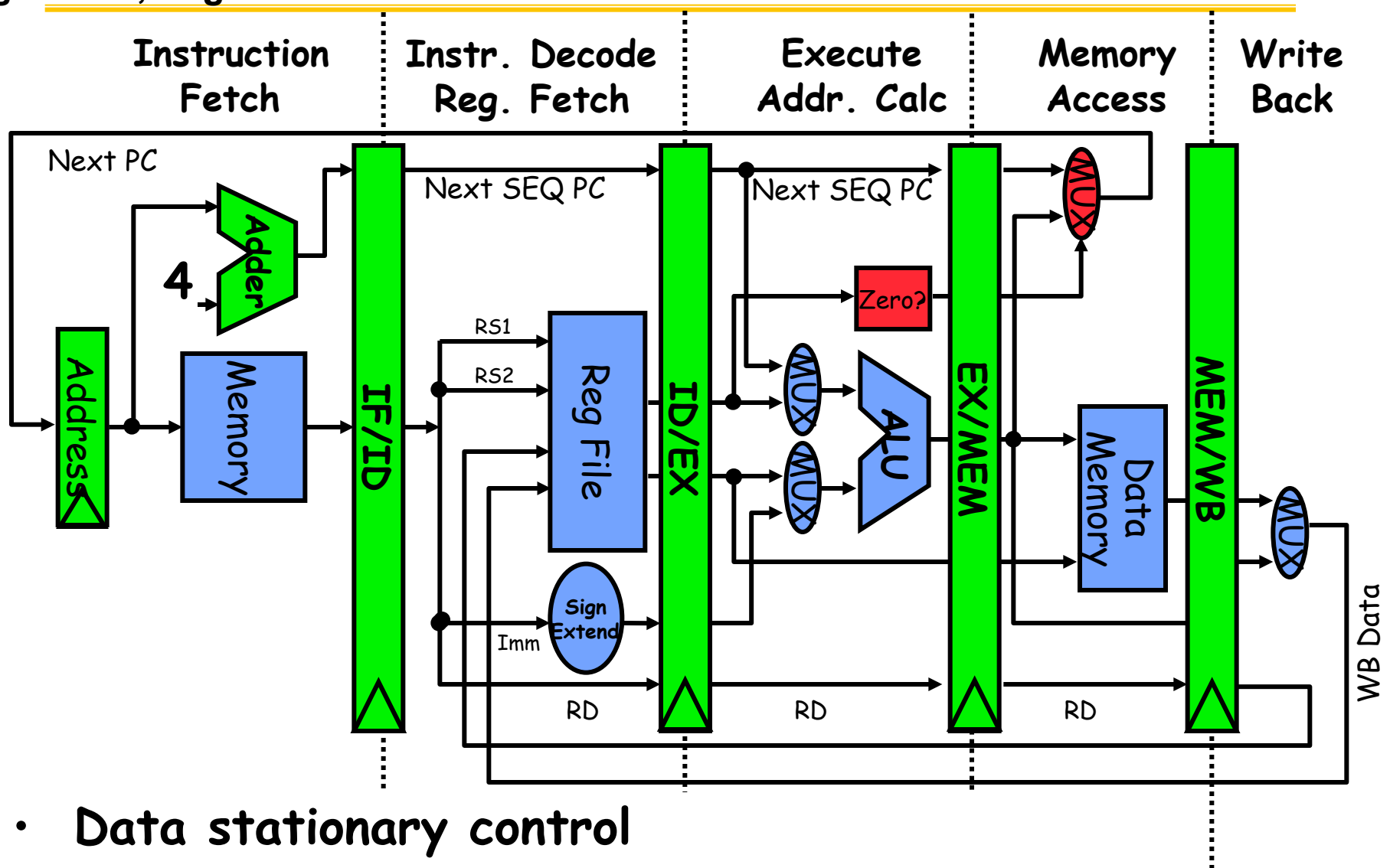


Inst. Set Processor Controller



5-Stage MIPS Datapath (has pipeline latches)

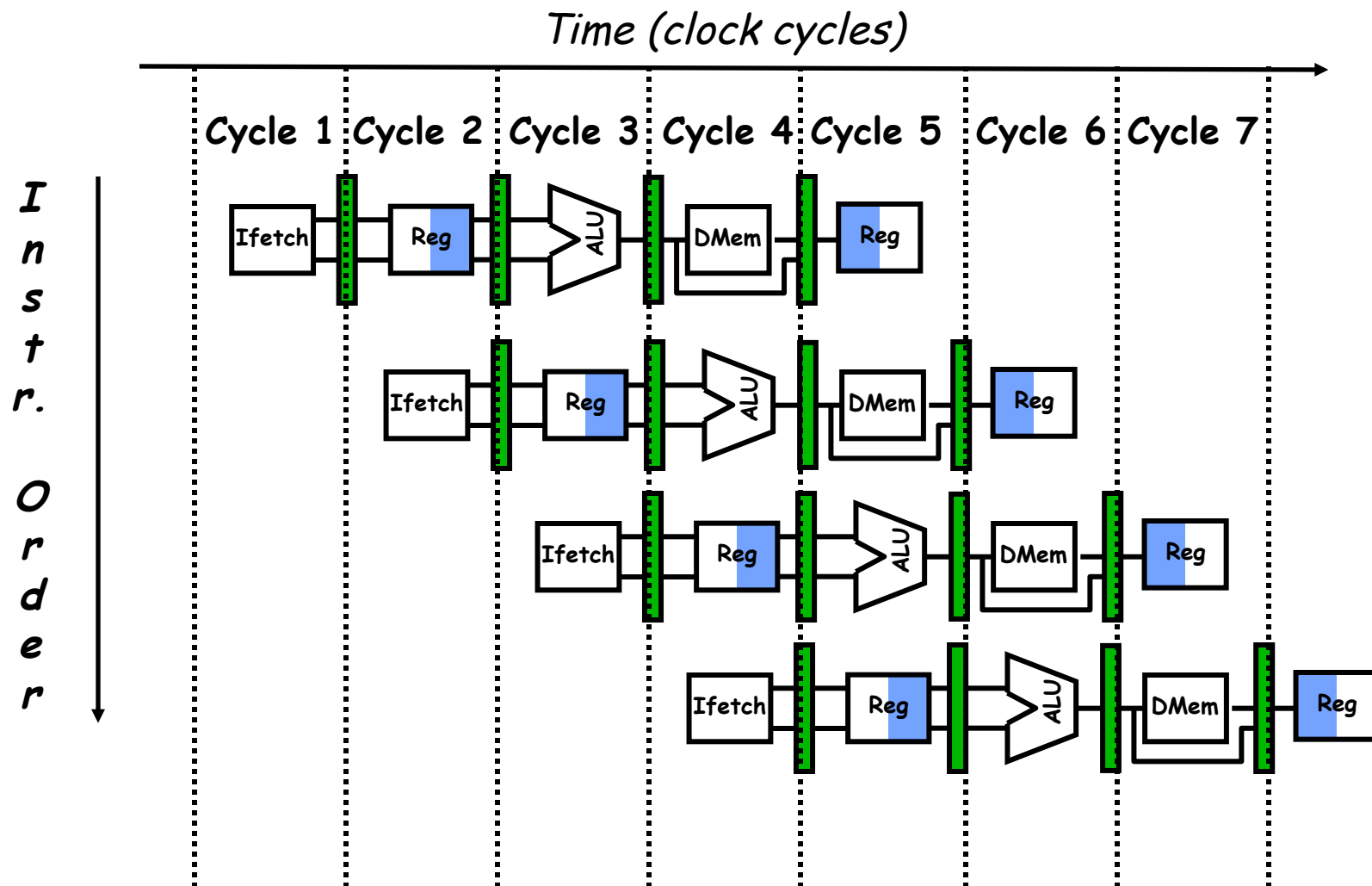
Figure A.3, Page A-9



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

Visualizing Pipelining

Figure A.2, Page A-8

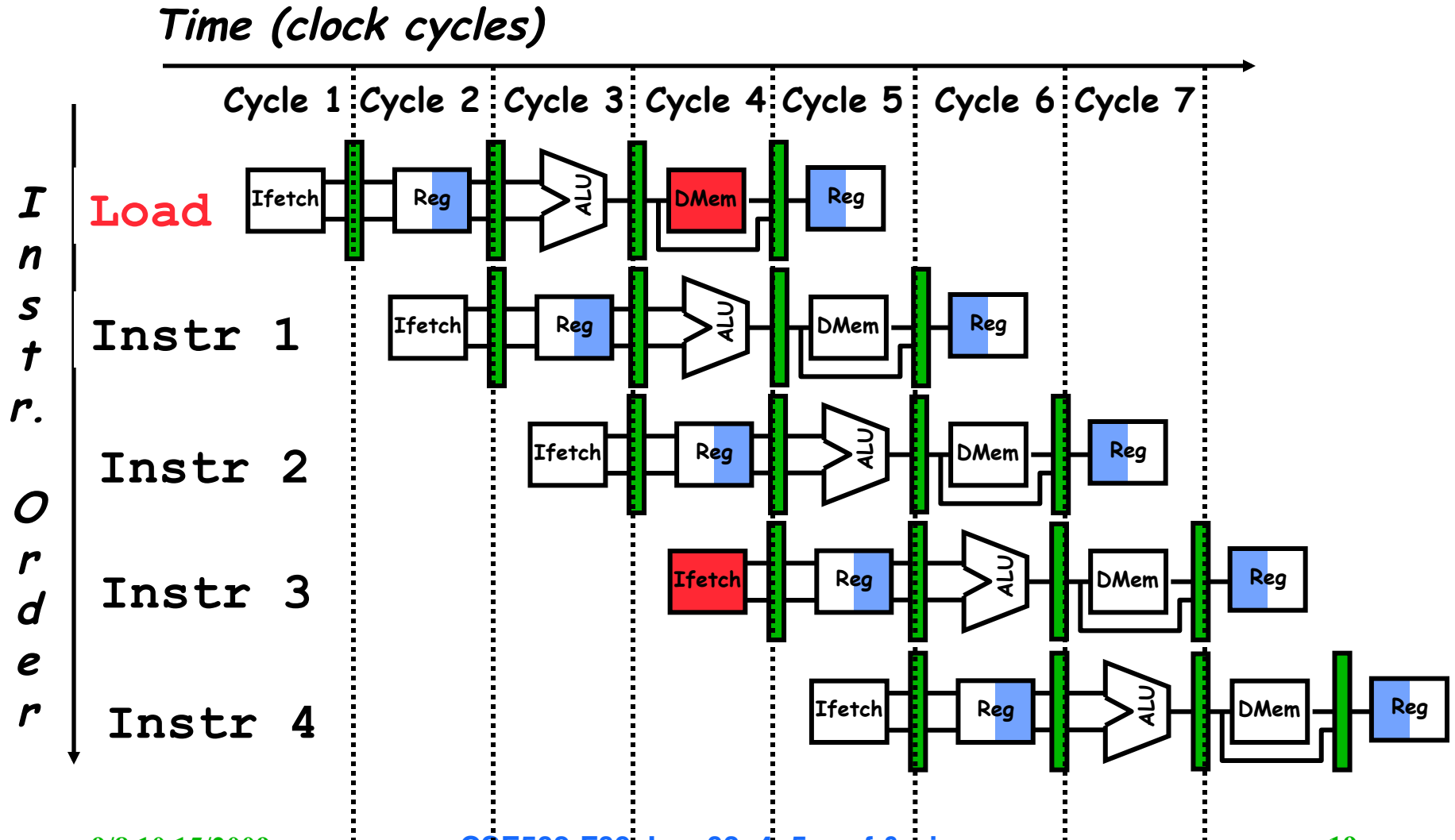


Pipelining is not quite that easy!

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

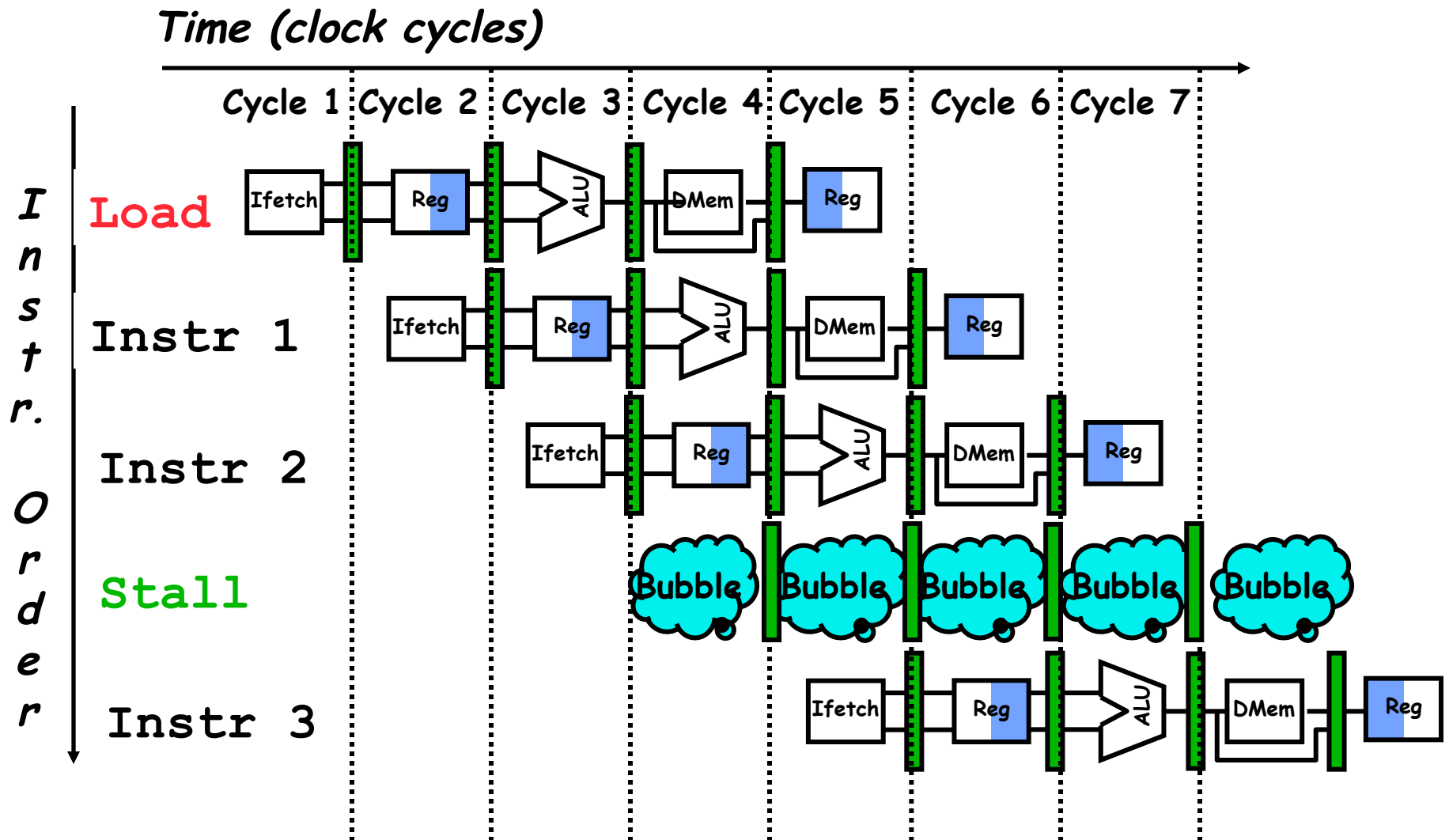
One Memory Port/Structural Hazards

Figure A.4, Page A-14



One Memory Port/Structural Hazards

(Similar to Figure A.5, Page A-15)



How do you "bubble" the pipe?

Code SpeedUp 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, Ideal 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: 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
- Assume loads are 20% of instructions executed

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

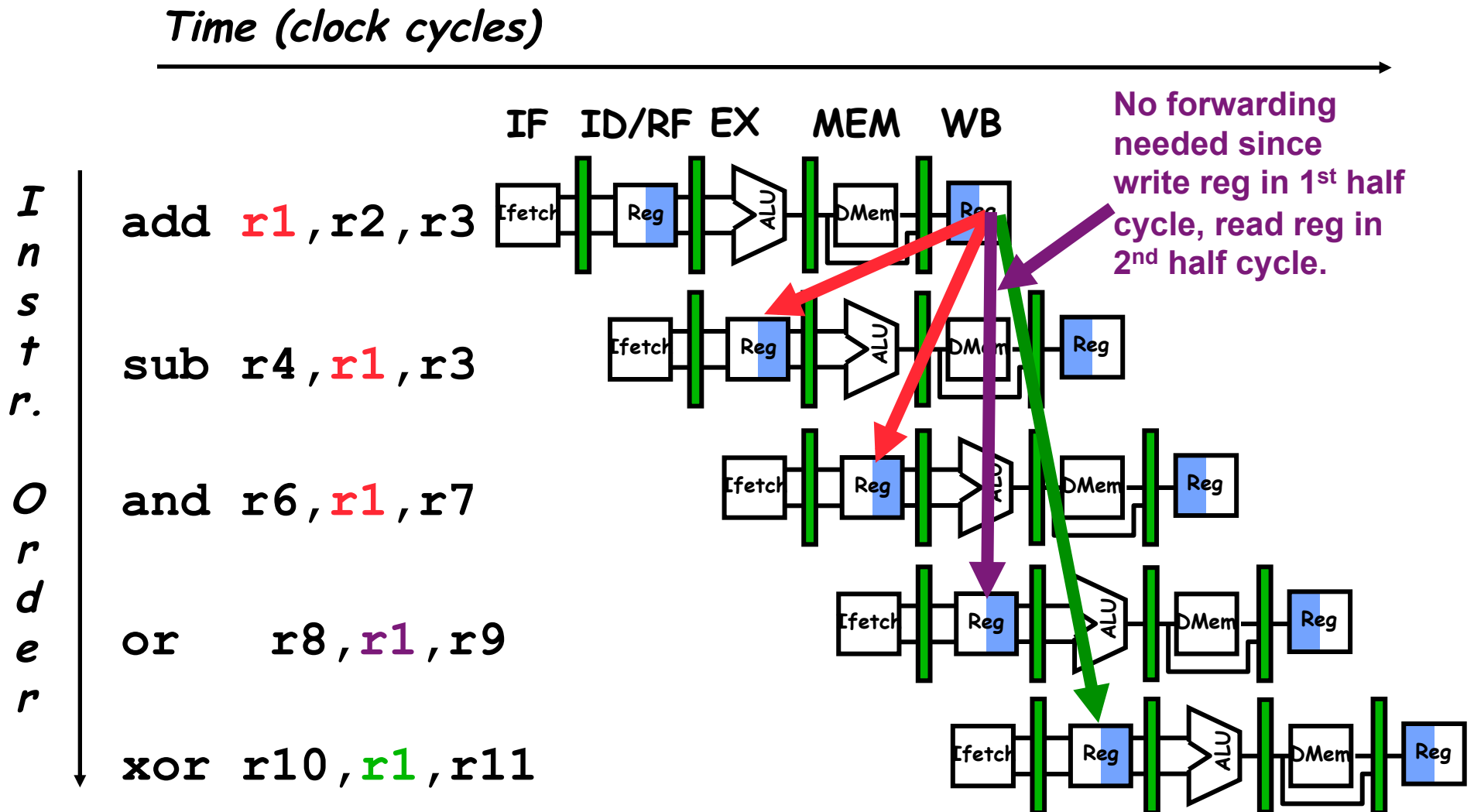
$$\begin{aligned}\text{SpeedUp}_B &= \text{Pipeline Depth} / (1 + 0.2 \times 1) \times (\text{clock}_{\text{unpipe}} / (\text{clock}_{\text{unpipe}} / 1.05)) \\ &= (\text{Pipeline Depth} / 1.2) \times 1.05 \\ &= 0.875 \times \text{Pipeline Depth}\end{aligned}$$

$$\text{SpeedUp}_A / \text{SpeedUp}_B = \text{Pipeline Depth} / (0.875 \times \text{Pipeline Depth}) = 1.14$$

- Machine A is 1.14 times faster

Data Hazard on Register R1 (If No Forwarding)

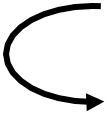
Figure A.6, Page A-17



Three Generic Data Hazards

- **Read After Write (RAW)**

Instr_j tries to read operand before Instr_i 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 communicating a new data value.

Three Generic Data Hazards

- **Write After Read (WAR)**

Instr_j writes operand before Instr_i 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**”.
- Cannot happen in MIPS 5 stage pipeline because:
 - All instructions take 5 stages, and
 - Register reads are always in stage 2, and
 - Register writes are always in stage 5

Three Generic Data Hazards

- **Write After Write (WAW)**

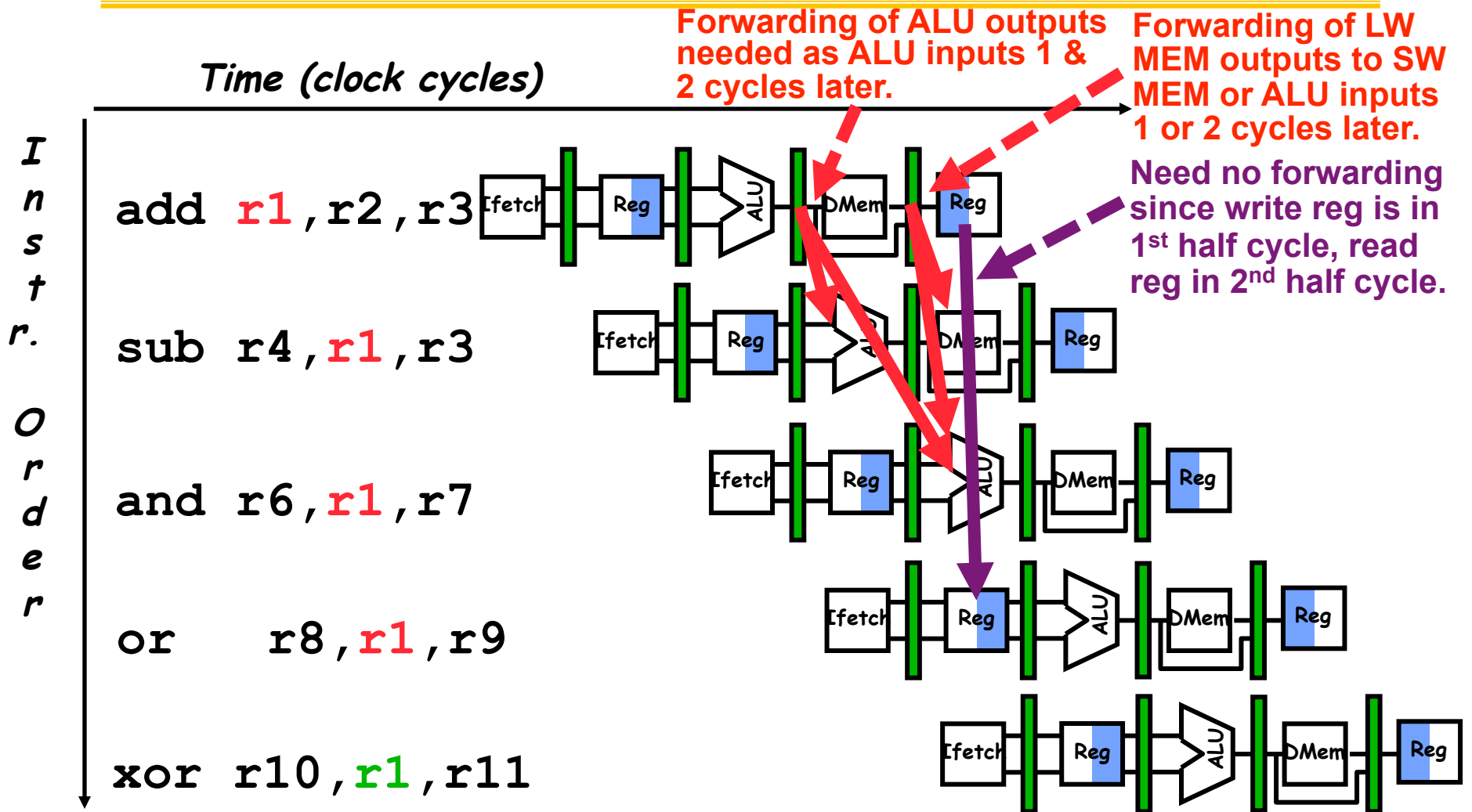
Instr_j writes operand before Instr_i writes it.

 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**”.
- Cannot happen in MIPS 5 stage pipeline because:
 - All instructions take 5 stages, and
 - Register writes are always in stage 5
- Will see WAR and WAW in more complicated pipes

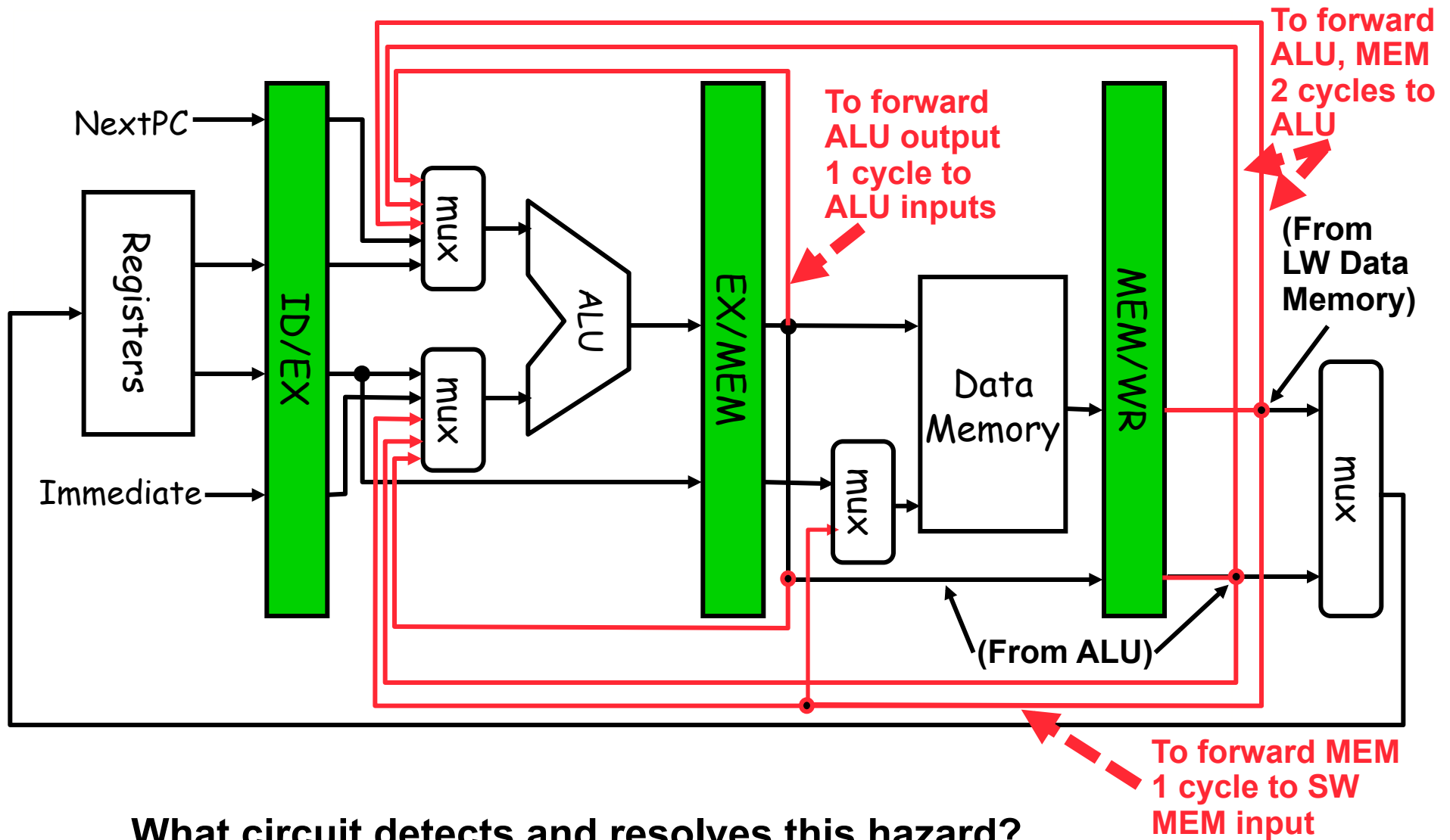
Forwarding to Avoid Data Hazard

Figure A.7, Page A-19



HW Datapath Changes (in red) for Forwarding

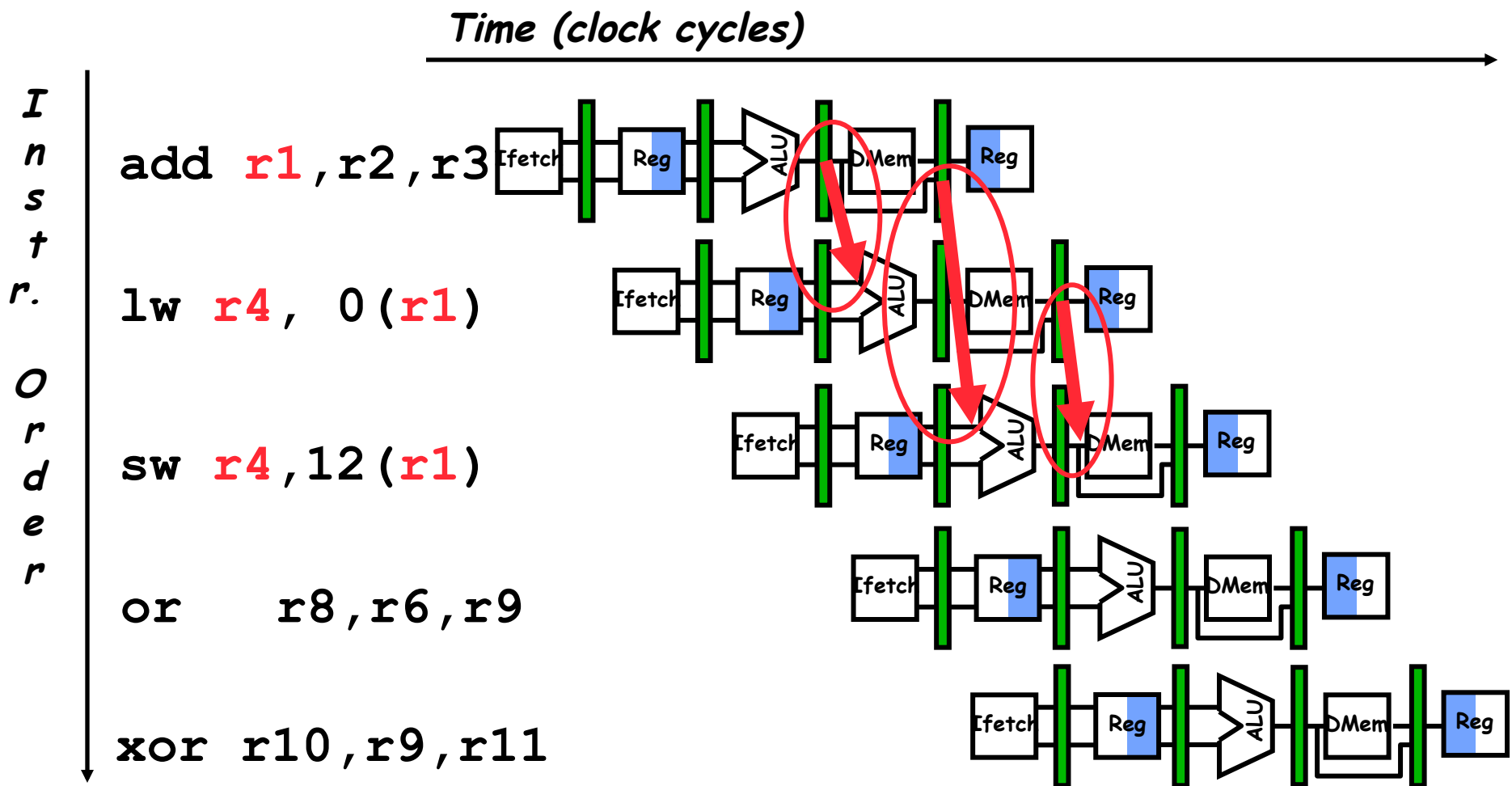
Figure A.23, Page A-37



What circuit detects and resolves this hazard?

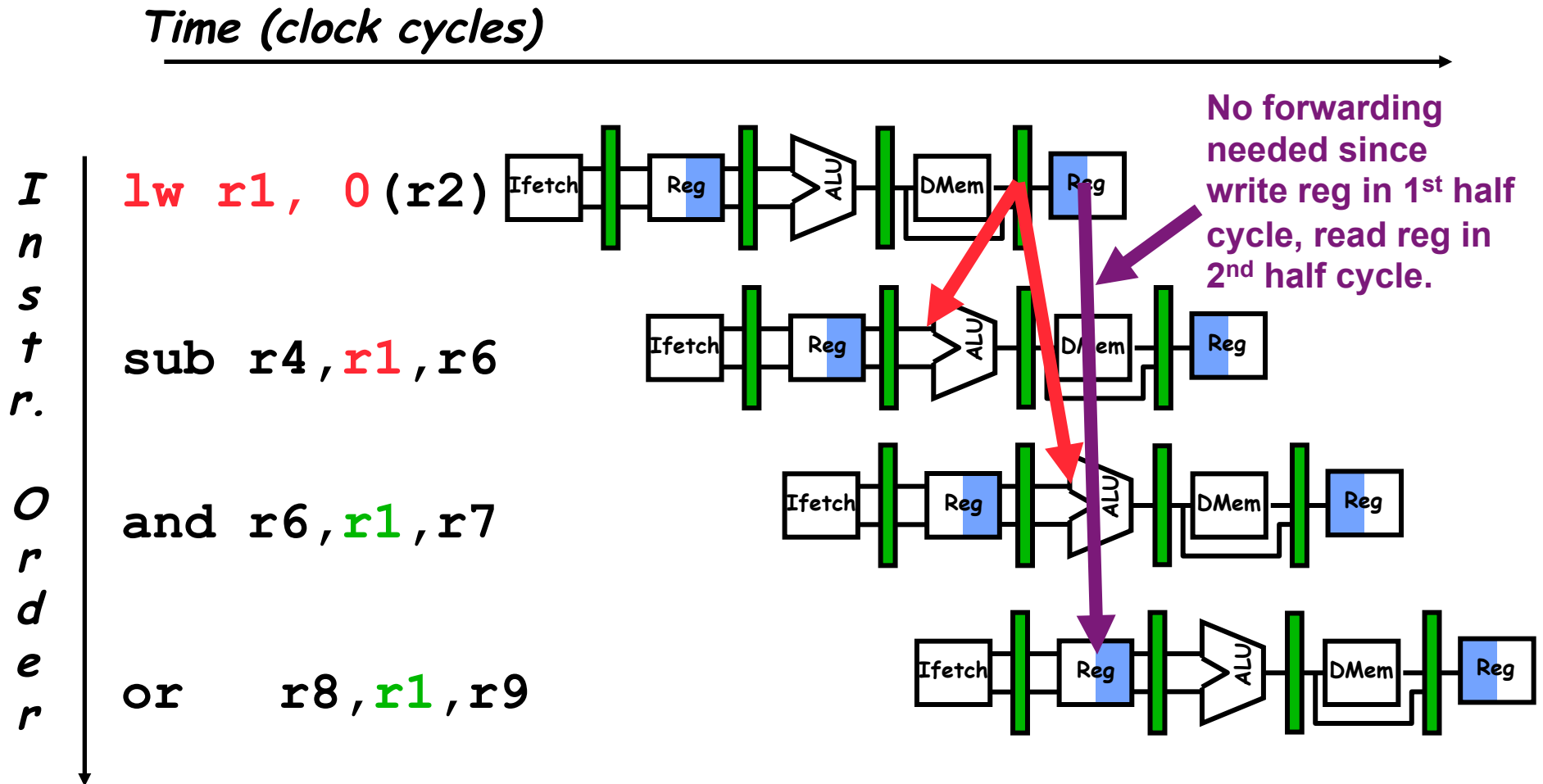
Forwarding Avoids ALU-ALU & LW-SW Data Hazards

Figure A.8, Page A-20



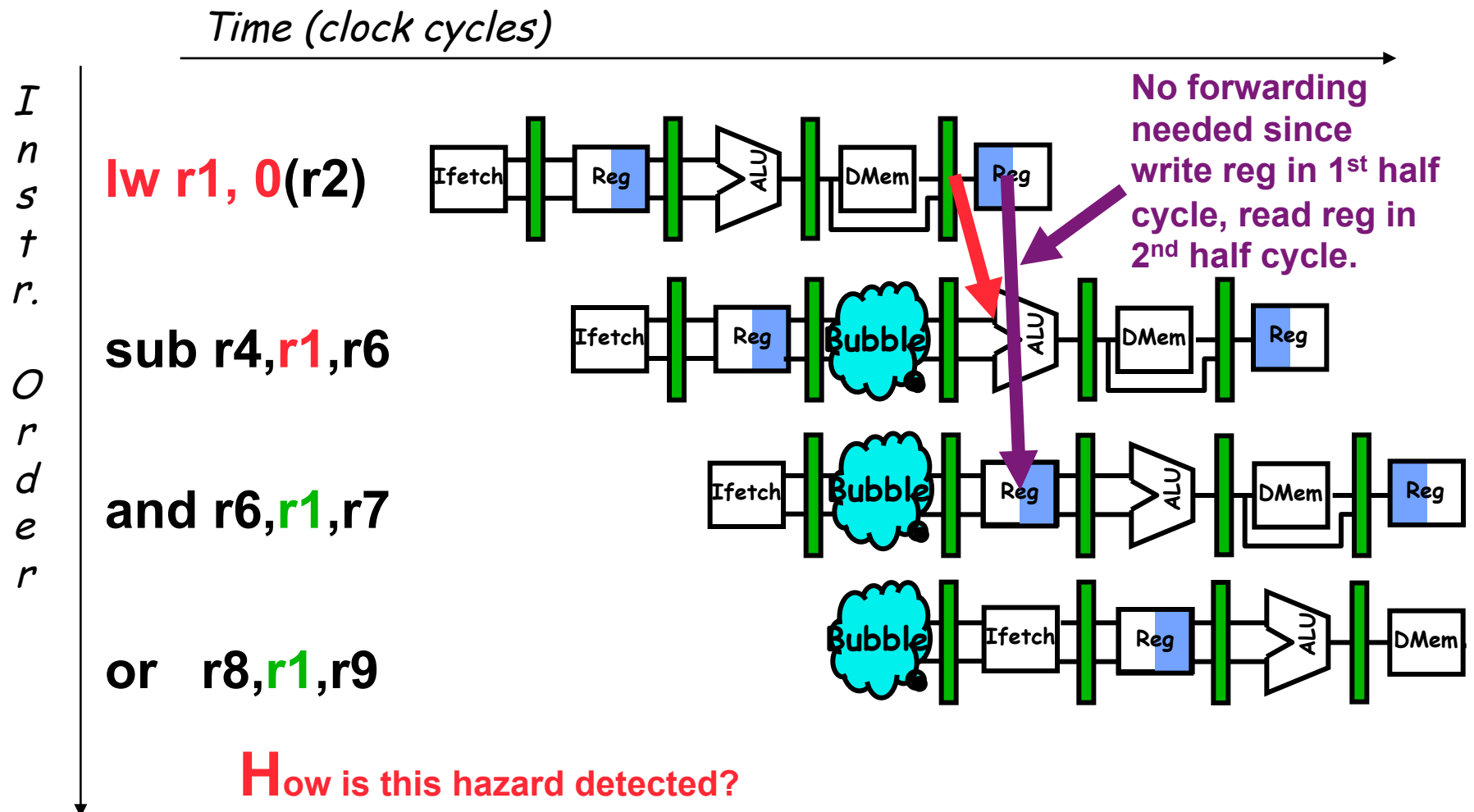
LW-ALU Data Hazard Even with Forwarding

Figure A.9, Page A-21



Data Hazard Even with Forwarding

(Similar to Figure A.10, Page A-21)



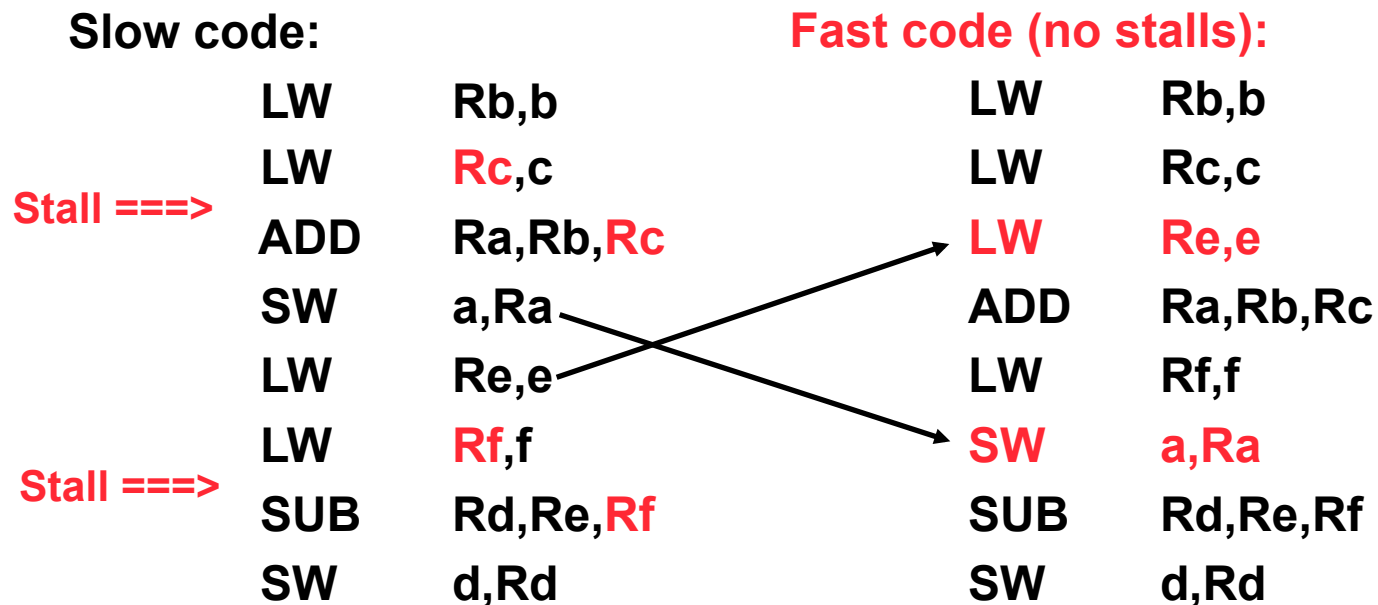
Software Scheduling to Avoid Load Hazards

Try producing fast code with no stalls for

$a = b + c;$

$d = e - f;$

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



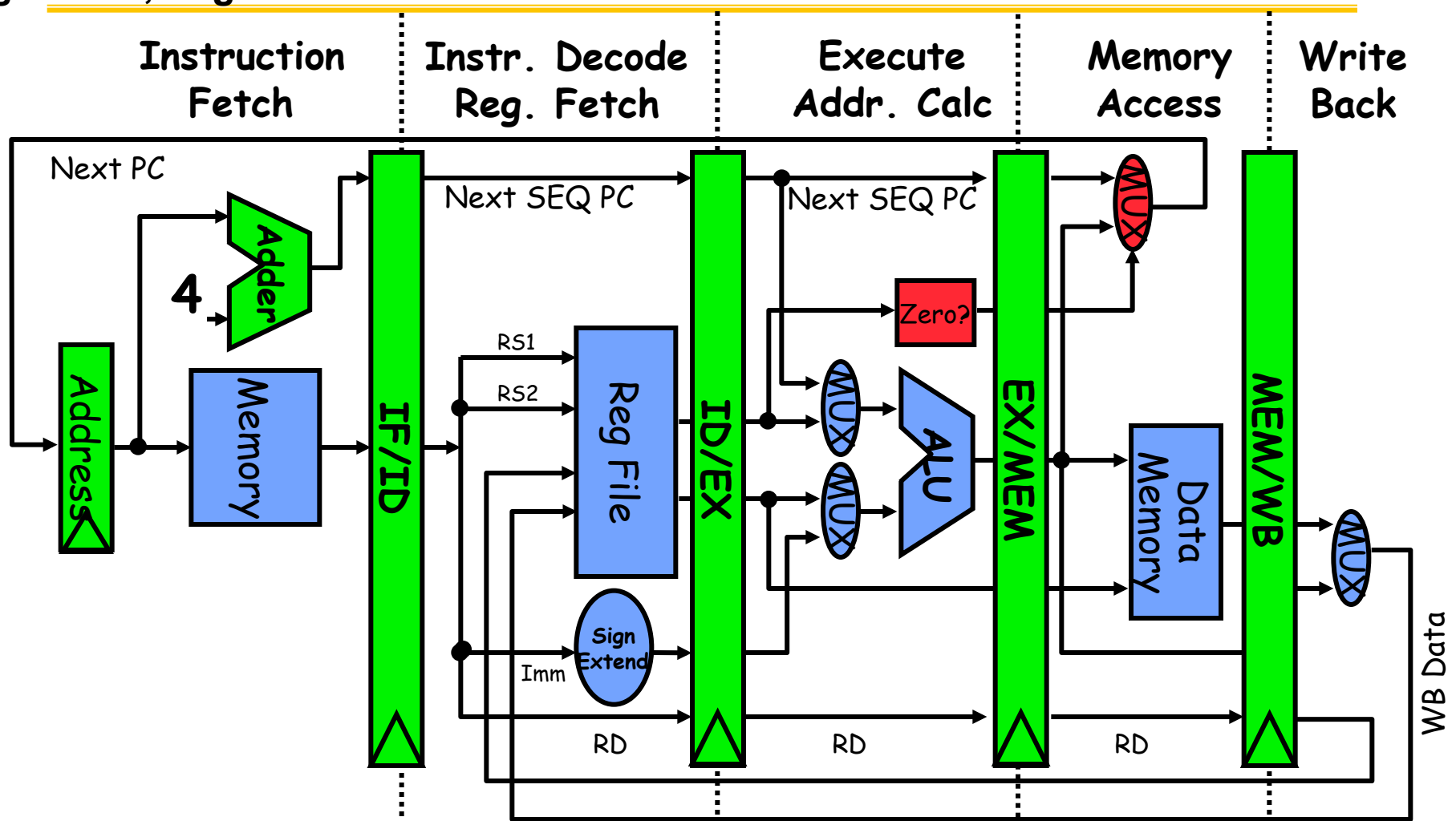
Compiler optimizes for performance. Hardware checks for safety.

Outline

- Review
- F&P: Benchmarks age, disks fail, single-points fail
- 502 Administrivia
- MIPS – An ISA for Pipelining
- 5 stage pipelining
- Structural and Data Hazards
- Forwarding
- **Branch Schemes**
- **Exceptions and Interrupts**
- **Conclusion**

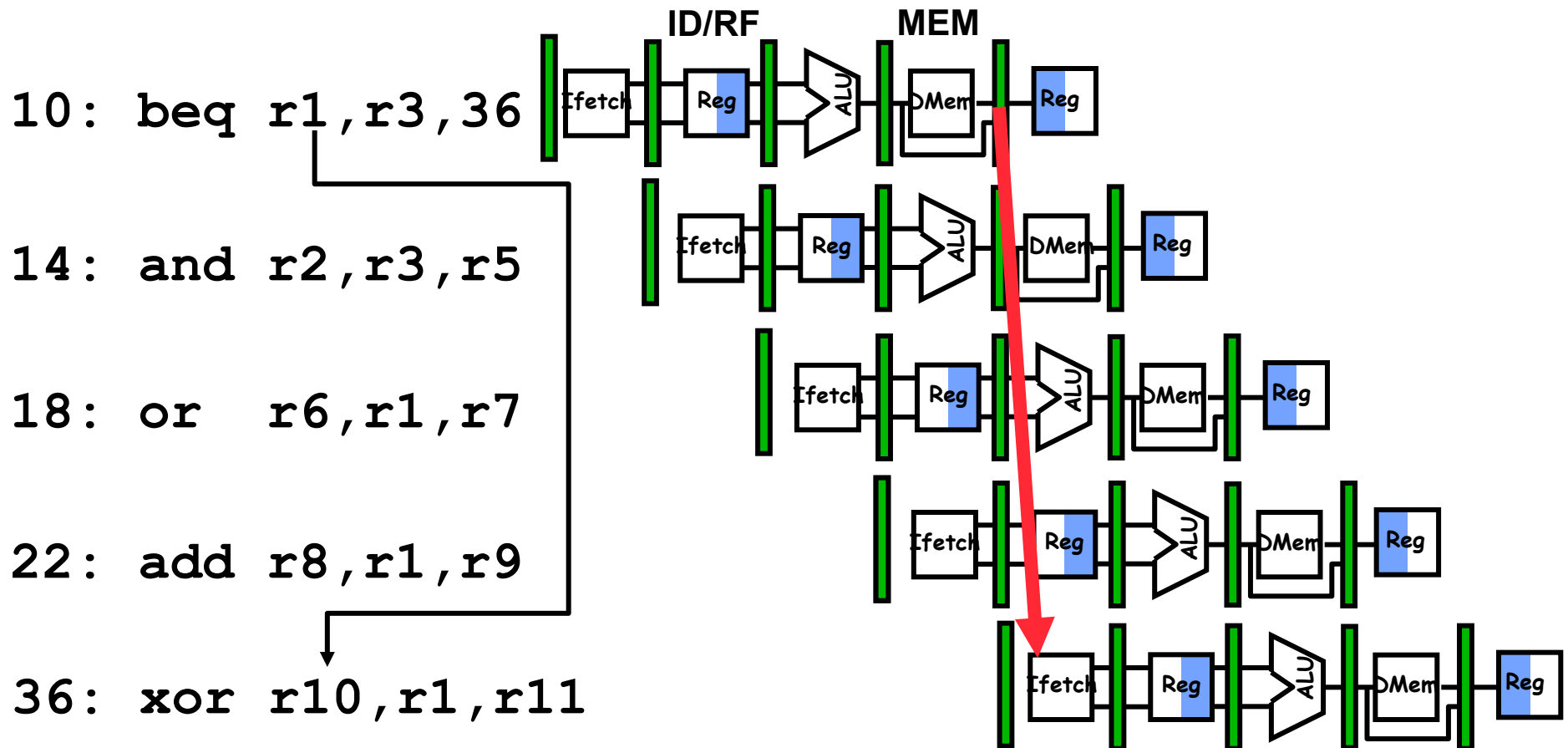
5-Stage MIPS Datapath (has pipeline latches)

Figure A.3, Page A-9



- Simple design put **branch completion** in stage 4 (Mem)

Control Hazard on Branch - Three Cycle Stall



What do you do with the 3 instructions in between?

How do you do it?

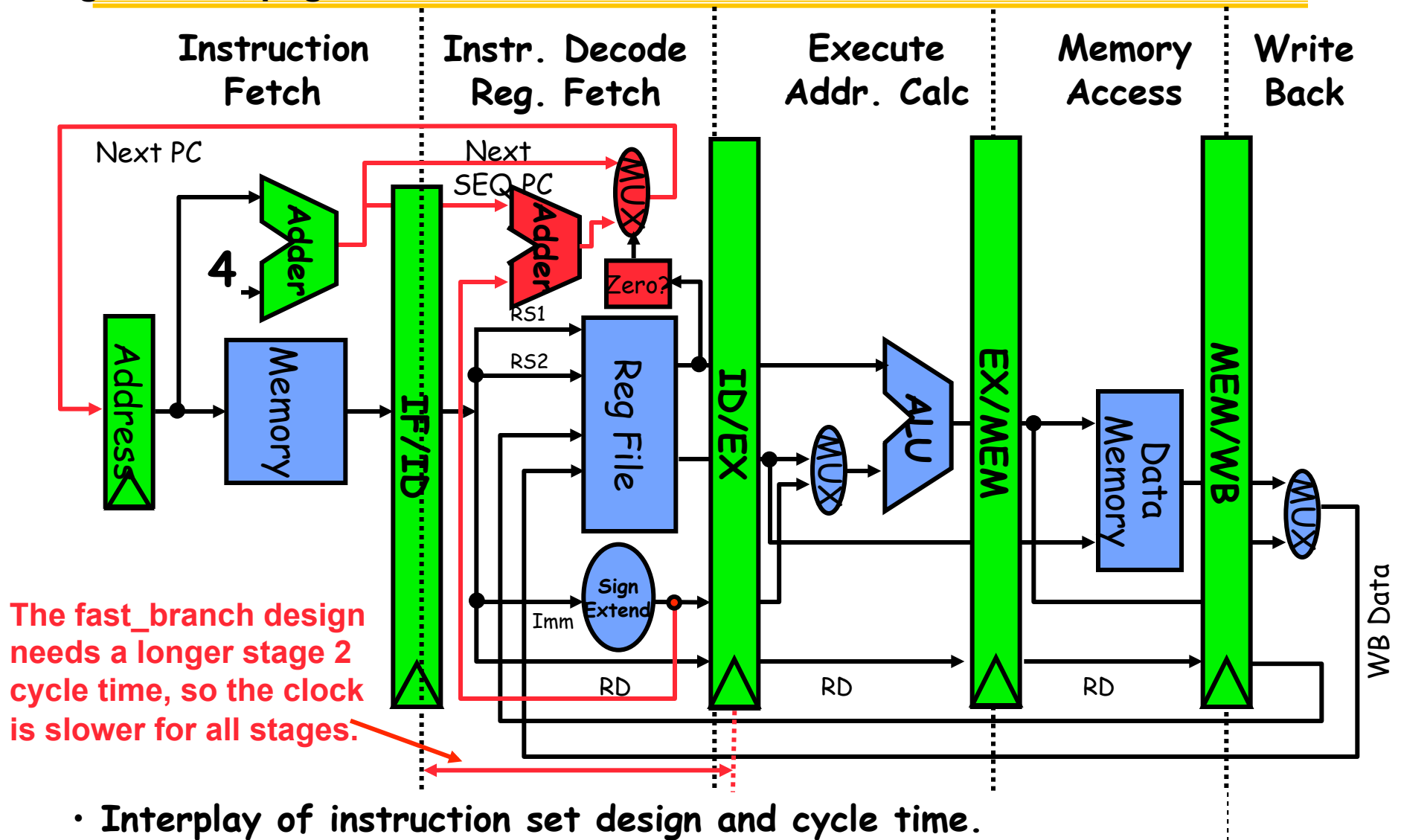
Where is the “commit”?

Branch Stall Impact if Commit in Stage 4

- **If CPI = 1 and 15% of instructions are branches, Stall 3 cycles => new CPI = 1.45!**
- **Two-part solution:**
 - Determine sooner whether branch taken or not, AND
 - Compute taken branch address earlier
- **MIPS branch tests if register = 0 or \neq 0**
- **MIPS Solution:**
 - Move zero_test to ID/RF (Instr Decode & Register Fetch) stage (2, 4=MEM)
 - Add extra adder to calculate new PC (Program Counter) in ID/RF stage
 - Result is 1 clock cycle penalty for branch versus 3 when decided in MEM

Pipelined MIPS Datapath

Figure A.24, page A-38



- Interplay of instruction set design and cycle time.

Four Branch Hazard Alternatives

#1: Stall until branch direction is clear

#2: Predict Branch Not Taken

- Execute the next instructions in sequence
- PC+4 already calculated, so use it to get next instruction
- Nullify bad instructions in pipeline if branch is actually taken
- Nullify easier since pipeline state updates are late (MEM, WB)
- 47% MIPS branches not taken on average

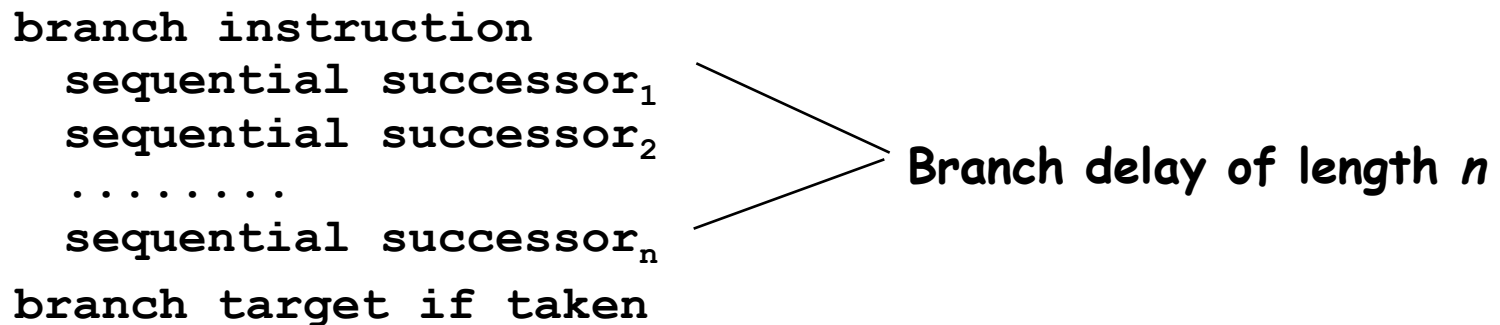
#3: Predict Branch Taken

- 53% MIPS branches taken on average
- But have not 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

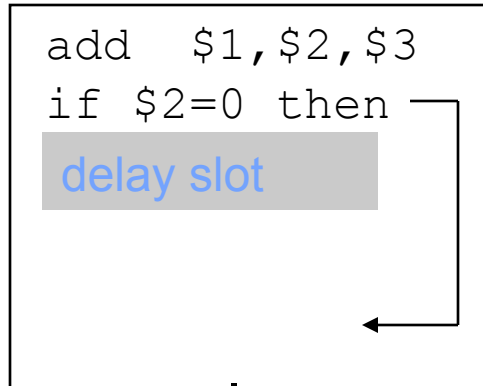
- Define branch to take place **AFTER** a following instruction



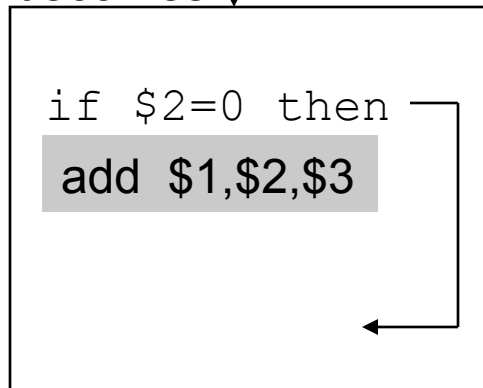
- 1 slot delay allows proper decision and branch target address in 5 stage pipeline
- MIPS 1st used this (Later versions of MIPS did not; pipeline deeper)

Scheduling Branch Delay Slots (Fig A.14)

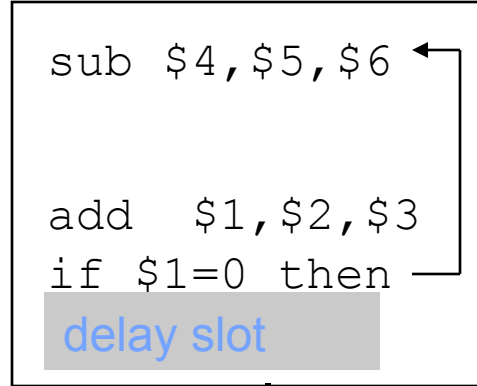
A. From before branch



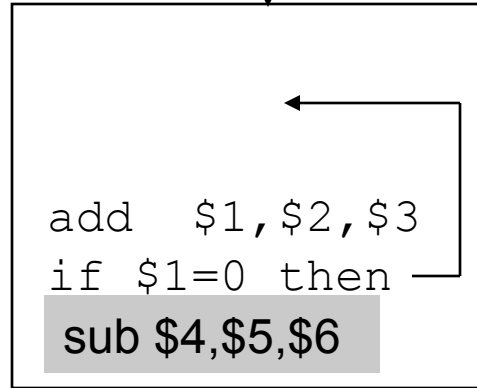
becomes



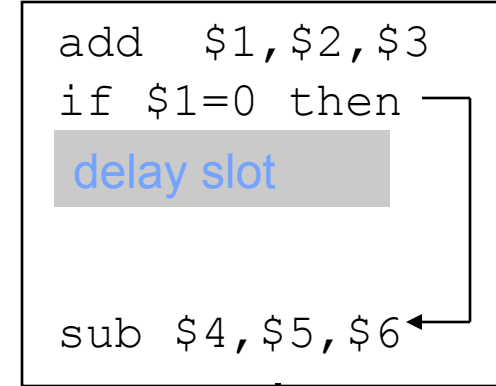
B. From branch target



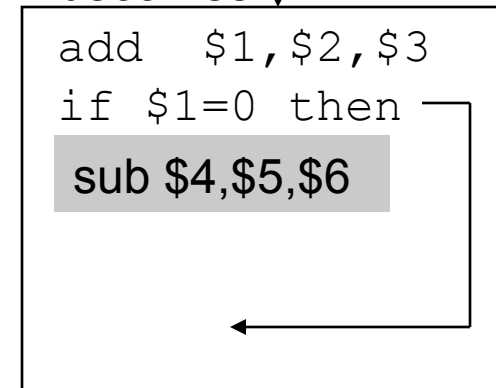
becomes



C. From fall through



becomes



- **A is the best choice, fills delay slot & reduces instruction count (IC)**
- **In B, the sub instruction may need to be copied, increasing IC**
- **In B and C, must be okay to execute sub when branch fails**

Delayed Branch Not Used in New CPUs

- **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
 - Only about 50% (60% x 80%) of slots usefully filled; cannot fill more
- **Delayed Branch downside: As processor designs use deeper pipelines and multiple issue, the branch delay grows and needs many more delay slots**
 - Delayed branching soon lost effectiveness and popularity compared to more expensive but more flexible dynamic approaches
 - Growth in available transistors soon permitted dynamic approaches that keep records of branch locations, taken/not-taken decisions, and target addresses
 - Multi-issue 2 => 3 delay slots needed, 4 => 7 slots, 8 => 15 slots

Evaluating Branch Alternatives

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

Assume 4% unconditional jump, 6% conditional branch-untaken, 10% conditional branch-taken, base CPI = 1.

<i>Scheduling scheme</i>	<i>Branch penalty</i>	<i>CPI</i>	<i>speedup vs. no-pipe 5 cycles</i>	<i>speedup vs. stall_pipeline</i>
Stall pipeline	3	1.60	3.1	1.00
Predict taken	1	1.20	4.2	1.33
Predict not taken	1	1.14	4.4	1.40
Delayed branch	0.5	1.10	4.5	1.45

(Sample calculation)

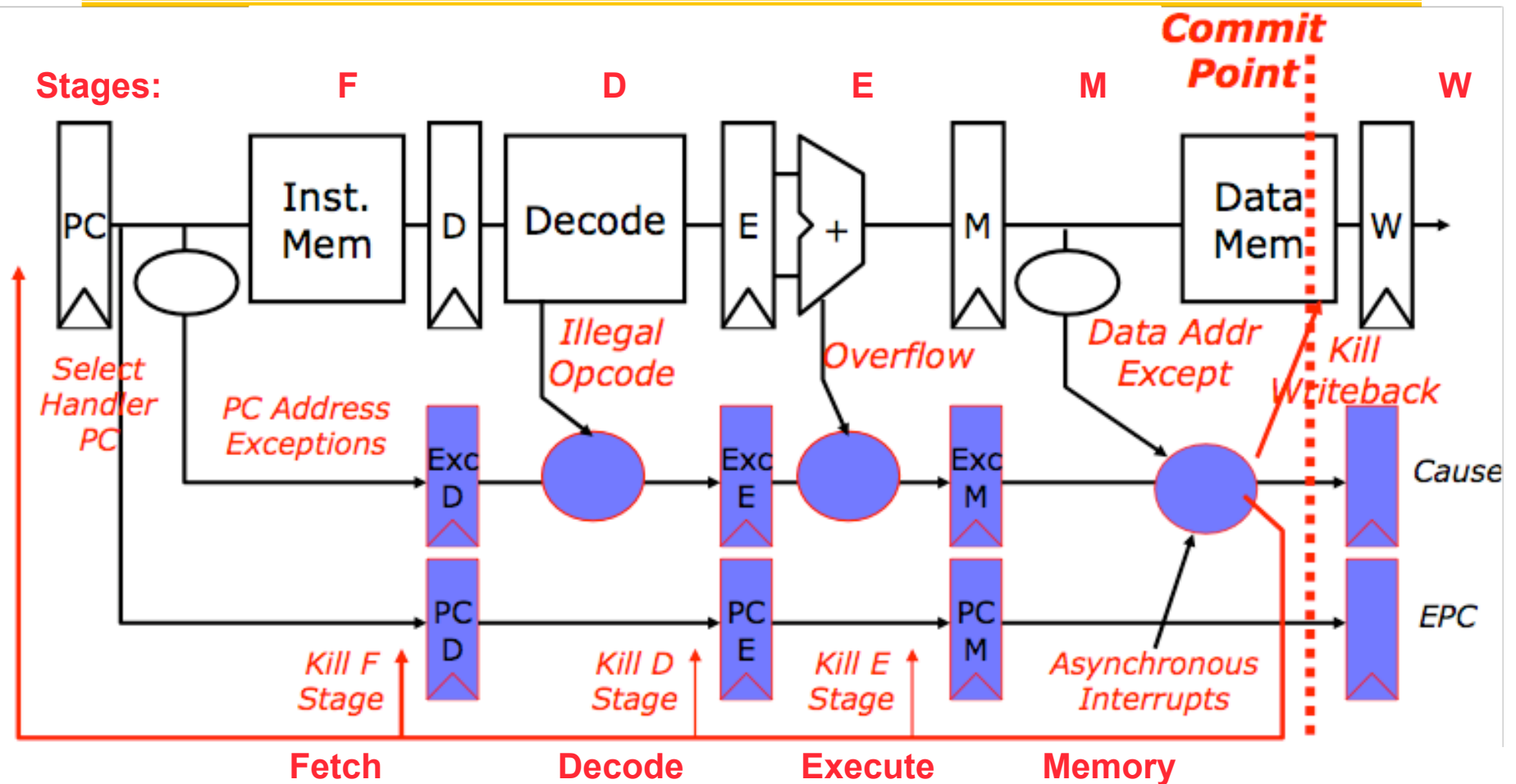
(=5/1.10)

(=1.6/1.1)

Problems with Pipelining

- **Exception:** An unusual event happens to an instruction during its execution
 - Examples: divide by zero, undefined opcode
- **Interrupt:** Hardware signal to switch the processor to a new instruction stream
 - Example: a sound card interrupts when it needs more audio output samples (an audio “click” happens if it is left waiting)
- **Precise Interrupt Problem:** Must seem that the exception or interrupt appeared between 2 instructions (I_i and I_{i+1}) although several instructions were executing at the time
 - The effects of all instructions up to and including I_i are totally complete
 - No effect of any instruction after I_i are allowed to be saved
- After a precise interrupt, the interrupt (exception) handler either aborts the program or restarts at instruction I_{i+1}

Precise Exceptions in Static Pipelines



Key observation: “Architected” states change only in memory (M) and register write (W) stages.

And In Conclusion: Control and Pipelining

- Quantify and summarize performance
 - Ratios, Geometric Mean, Multiplicative Standard Deviation
- F&P: Benchmarks age, disks fail, single-point failure
- Control via **State Machines** and **Microprogramming**
- 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 or branch (taken/not-taken) prediction
- Exceptions and interrupts add complexity
- Next time: Read Appendix C
- No class Tuesday 9/29/09, when Monday classes will run.