CprE 381: Computer Organization and Assembly Level Programming

Pipelining + Hazards

Henry Duwe
Electrical and Computer Engineering
Iowa State University

Administrative

- Term Project
  - Part 2a due BEFORE Spring Break
  - Anyone try the synthesis tool yet? (it will take an hour plus just to get a result – plan accordingly)

Questions

- What's your favorite color?
  - Red
  - The right color or colors at the right time/place/mood

- Are there any classes here that teach parallel processor design?
  - We will discuss a bit more than previous semesters
  - CprE 581
  - CprE 480 (not offered anymore ☹)

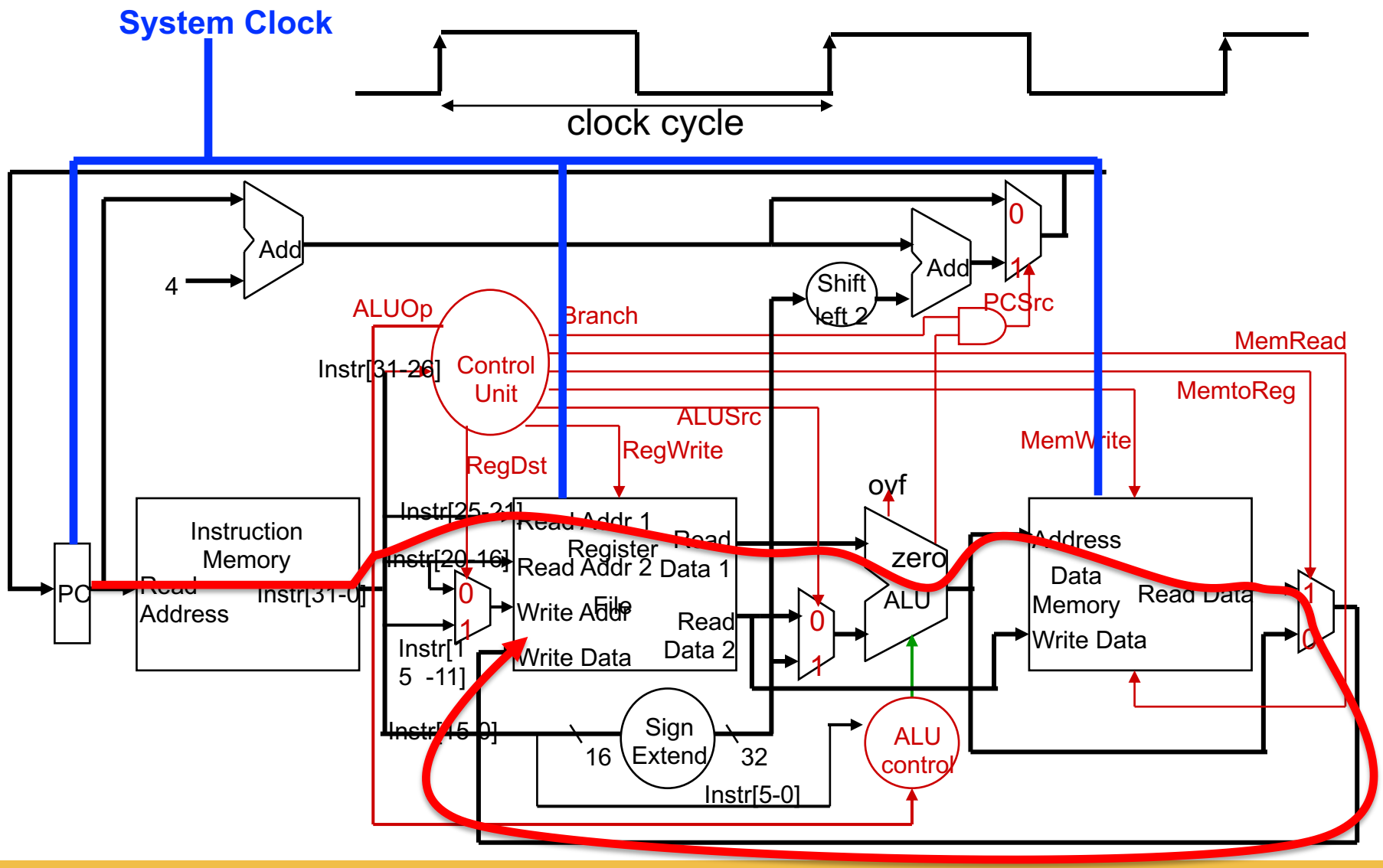
Questions

- **I've heard most people's processors dont work at the end of this class.** What have do you see as to the biggest reason for that?
  - Disagree with this assertion; in my experience, the majority of groups had a mostly-working processor by the last week in the semester
    - Most straight-line instructions work in all designs
    - Control flow works some of the time
    - Apps can be written to implement arbitrary (within memory size constraints) programs
  - Using the testing framework will more rigorously test your processors
    - 100% working processors from last semester actually had several errors discovered only with the testing framework and HW5 tests
  - Testing is critical! HW companies spend more money "testing" (verifying and validating) designs than actually "designing"
  - Silicon Errata
    - Intel Skylake has 177 known errors: https://www.intel.com/content/dam/www/public/us/en/documents/specification-updates/desktop-6th-gen-core-family-spec-update.pdf

Questions

- I've heard most people's processors dont work at the end of this class. **What have do you see as to the biggest reason for that?**
  - Not starting early in order to get help early
  - Poor understanding of MIPS
  - Jumping in to coding too quickly (doing before thinking)
  - Poor testing practices (sub components not working)
  - Poor debug skills (only time and experience will help hone, reflecting/recording may make this more rapid)
- Fixes (non-exhaustive list)
  - Already done (hopefully): started (some students already have mostly working processors)
  - Already done (hopefully): tested individual components you use in your processor
  - Already done (hopefully): learned basic MIPS and how to write/read MIPS
  - Use HW5 test suite to incrementally demonstrate that your processor is working
  - Use HW5 test suite + testing framework as regression tests (every time you update your processor, make sure you didn't break anything)

Review: Worst Case Timing (Load Instruction)



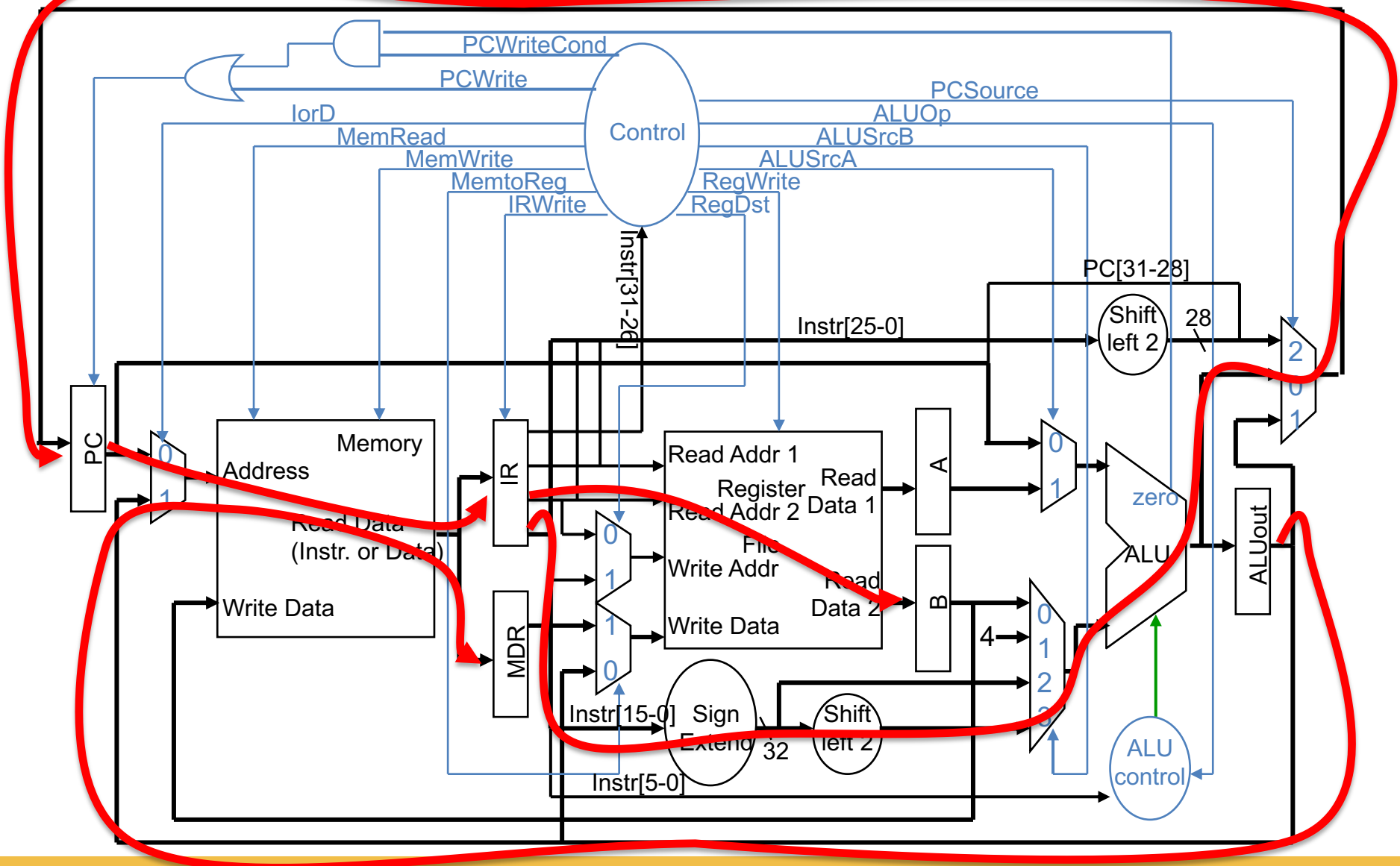
Review: Execution Time

- Drawing on the previous equation:

$$\textit{Execution Time} = \#\ \textit{Instructions} \times \frac{\textit{Cycles}}{\textit{Instruction}} \times \frac{\textit{Seconds}}{\textit{Cycle}}$$

- To improve performance (i.e., reduce execution time)
  - Increase clock rate (decrease clock cycle time) OR
  - Decrease CPI OR
  - Reduce the number of instructions
- Designers balance cycle time against the number of cycles required
  - Improving one factor may make the other one worse...

Review: Multicycle Processor



Review: Execution Time

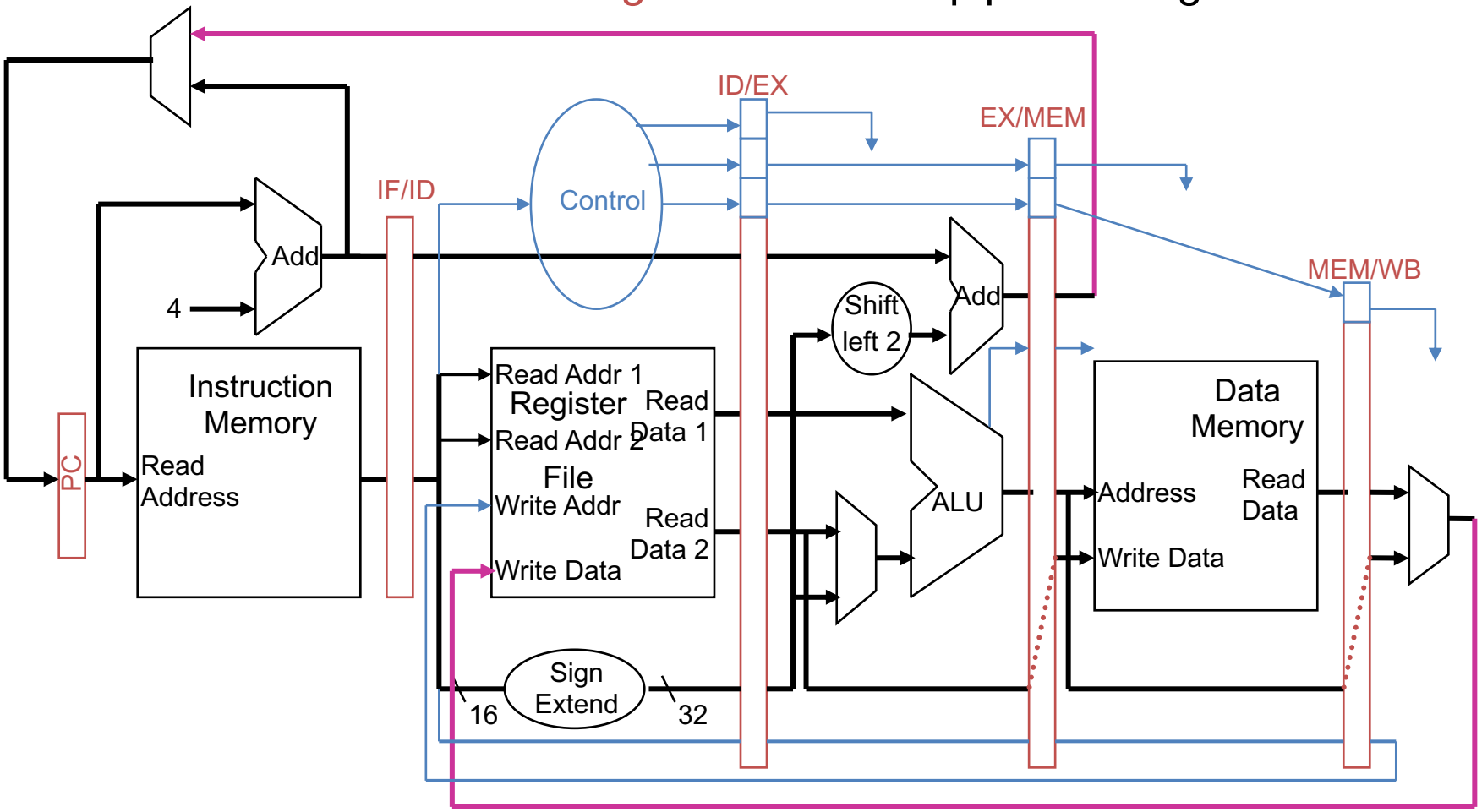
- Drawing on the previous equation:

$$\textit{Execution Time} = \#\ \textit{Instructions} \times \frac{\textit{Cycles}}{\textit{Instruction}} \times \frac{\textit{Seconds}}{\textit{Cycle}}$$

- To improve performance (i.e., reduce execution time)
  - Increase clock rate (decrease clock cycle time) OR
  - Decrease CPI OR
  - Reduce the number of instructions
- Designers balance cycle time against the number of cycles required
  - Improving one factor may make the other one worse...

Review: A Simple MIPS Pipeline

- All control signals can be determined during Decode
  - And held in the state registers between pipeline stages

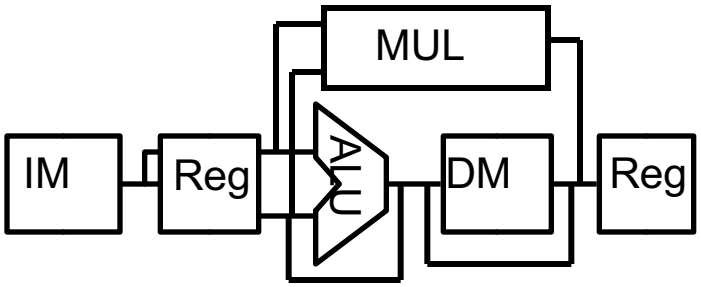


Performance Analyses

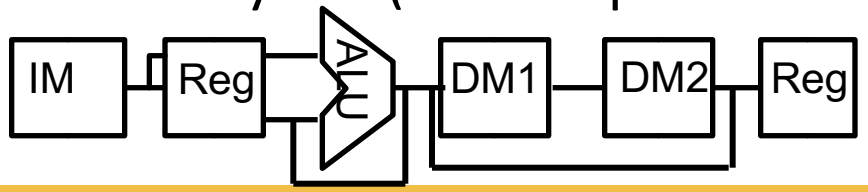
- What is the cycle time of each design?
  - Single-cycle: identify instruction that requires the longest/most serial propagation through components
  - Multicycle and pipeline: step/cycle/stage that requires serial propagation through the longest serial components
- What is the max performance of single-cycle, multi-cycle, pipelined design?
  - Partial performance measure: instruction throughput
- What is the performance of each design on a particular sequence of instructions? When the pipeline is full vs empty?
  - Instruction throughput (partial measure)
  - Execution time

Other Pipeline Structures Are Possible

- What about the (slow) multiply operation?
  - Make the clock twice as slow or … **Tricksy Professor warning**
  - Let it take two cycles (since it doesn't use the DM stage)



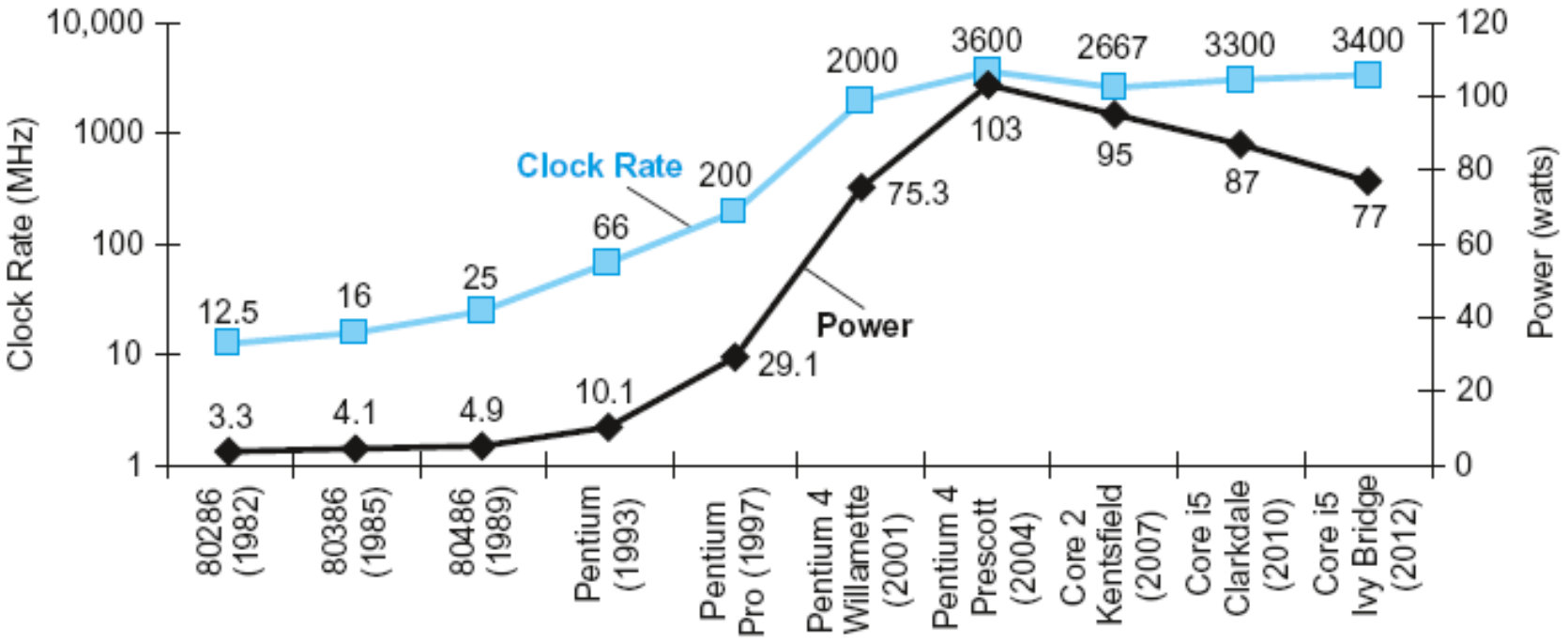
- What if the data memory access is twice as slow as the instruction memory?
  - Make the clock twice as slow or …
  - Let data memory access take two cycles (and keep the same clock rate)



If some is good, then more is…?

- If dividing it into 5 parts made the clock faster
  - And the effective CPI is still one
- Then dividing it into 10 parts would make the clock even faster
  - And wouldn't the CPI still be one?
- Then why not go to twenty cycles?
- Really two issues
  - Some things really have to complete in a cycle
    - Standard Cell Design: A single logic gate…
    - FPGA Design: A LUT access…
  - CPI is not really one
    - Sometimes you need the results from the previous instruction

~~Two~~Three Issues: Power Trends



- In CMOS IC technology

$$\text{Power} = \text{Capacitive load} \times \text{Voltage}^2 \times \text{Frequency}$$

×30

5V → 1.5V

×300

Reducing Dynamic Power

- Suppose a new CPU has
  - 85% of capacitive load of old CPU
  - 15% voltage and 15% frequency reduction

$$\frac{P_{new}}{P_{old}} = \frac{C_{old} \times 0.85 \times (V_{old} \times 0.85)^2 \times F_{old} \times 0.85}{C_{old} \times V_{old}^{\ 2} \times F_{old}} = 0.85^4 = 0.52$$

- The power wall
  - We can't reduce voltage further
  - We can't remove more heat
- How else can we improve performance?

Making it Even Faster

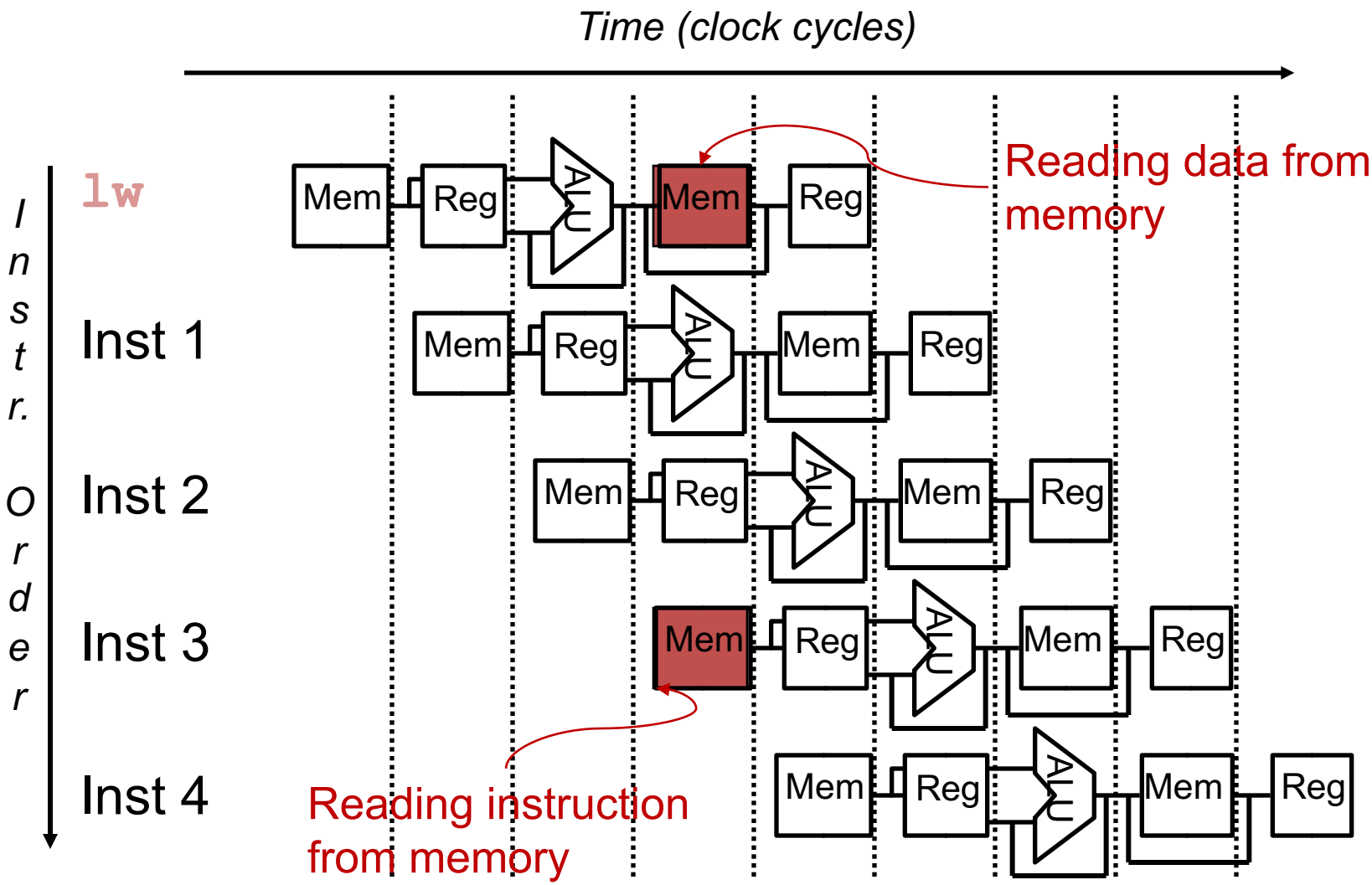
- In splitting the pipelined design into smaller and smaller steps, there is an optimal #/length!

- Other potential optimizations:
  - Fetch (and execute) more than one instruction at a time (out-of-order superscalar and VLIW (epic) – CprE 581)
  - Fetch (and execute) instructions from more than one instruction stream (multithreading (hyperthreading)) – CprE 581)

Oh, the *Hazards* I've Seen

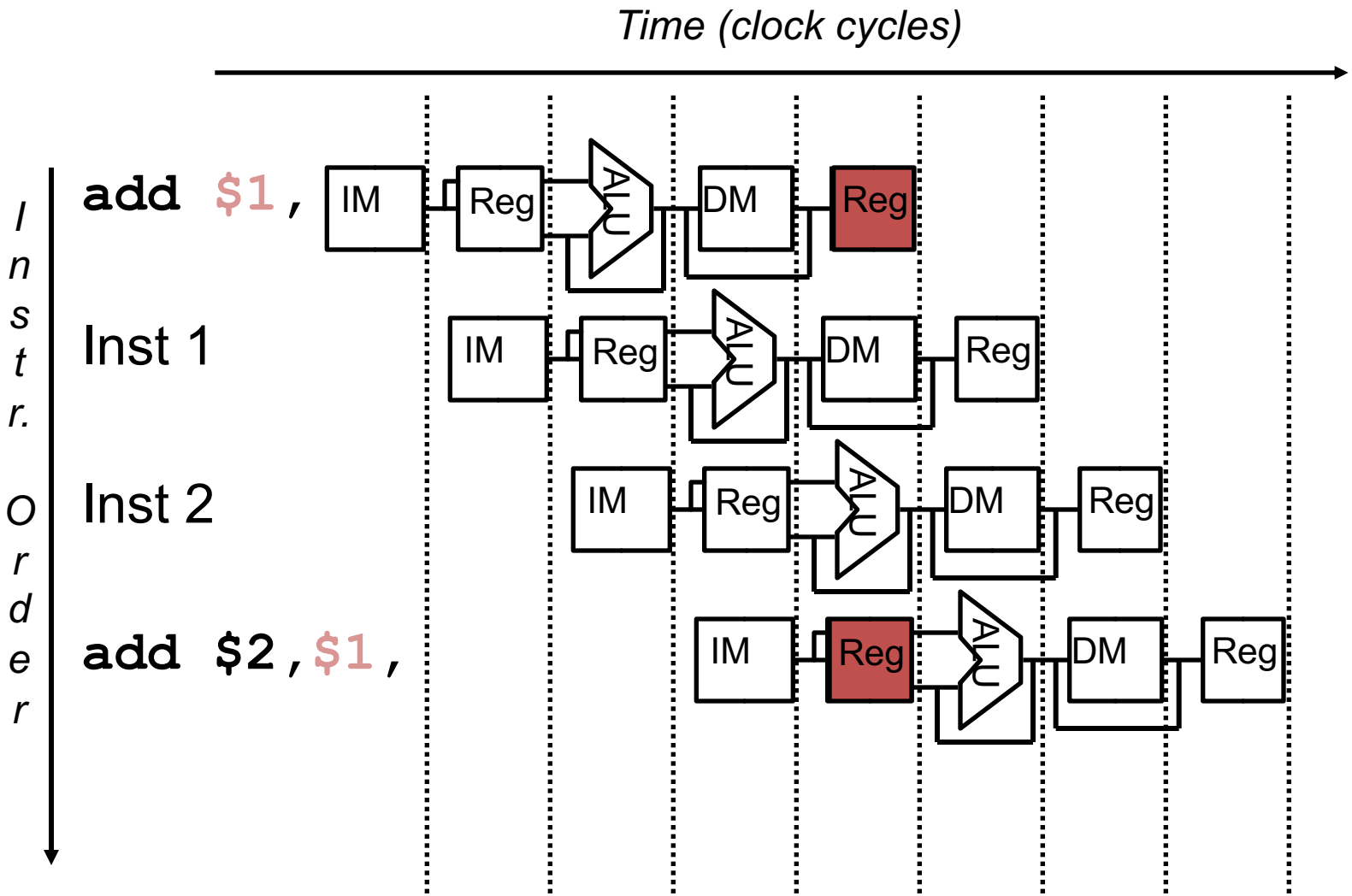
- Pipeline Hazards
  - Structural hazards: attempt to use the same resource by two different instructions at the same time
  - Data hazards: attempt to use data before it is ready
    - An instruction's source operand(s) are produced by a prior instruction still in the pipeline
  - Control hazards: attempt to make a decision about program control flow before the condition has been evaluated and the new PC target address calculated
    - Branch and jump instructions, exceptions
- Can always resolve hazards by **waiting**
  - Pipeline control must detect the hazard
  - And take action to resolve hazards

Structural Hazard: Single Memory

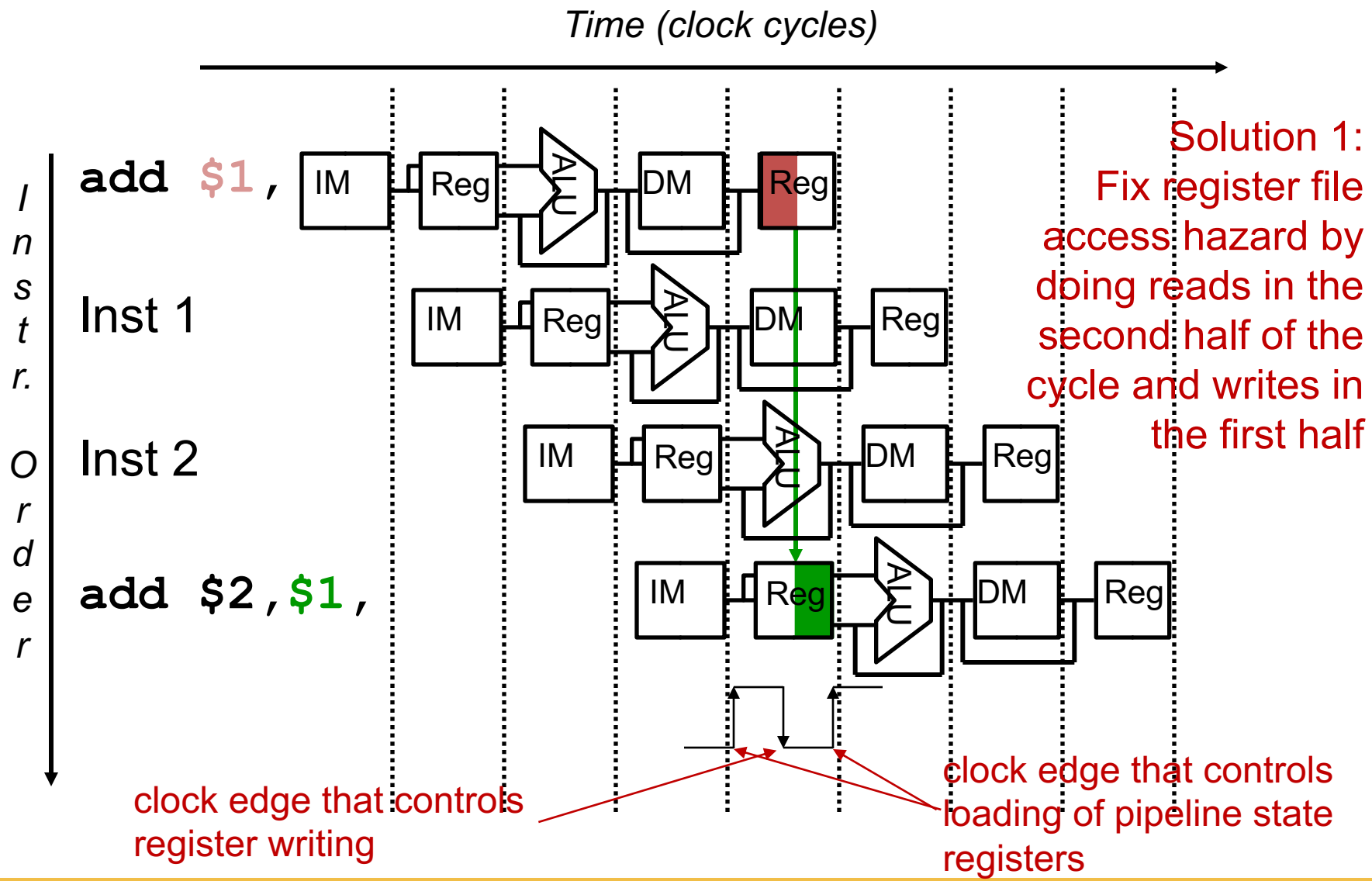


- Can fix with separate instr and data memories

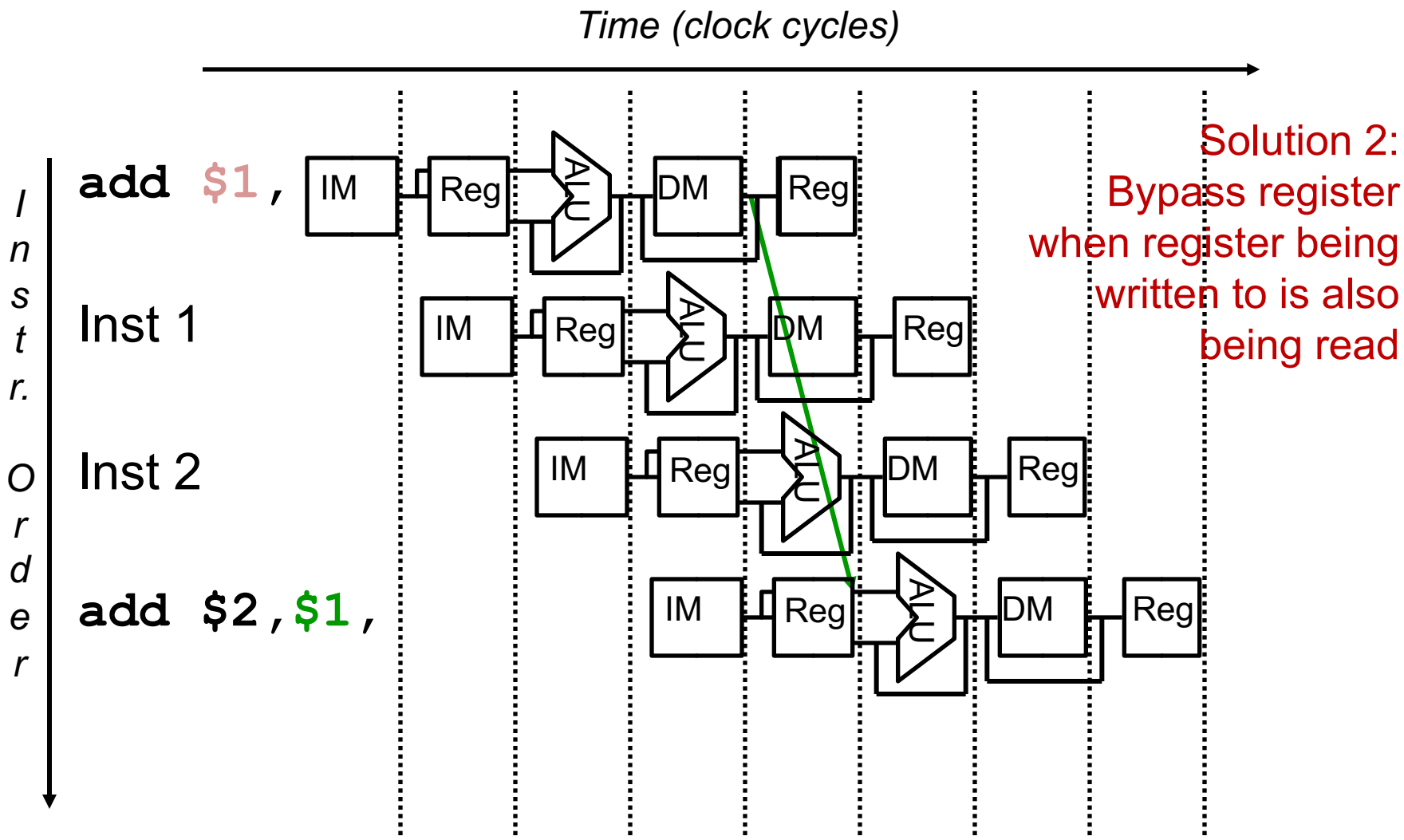
How About Register File Access?



How About Register File Access?



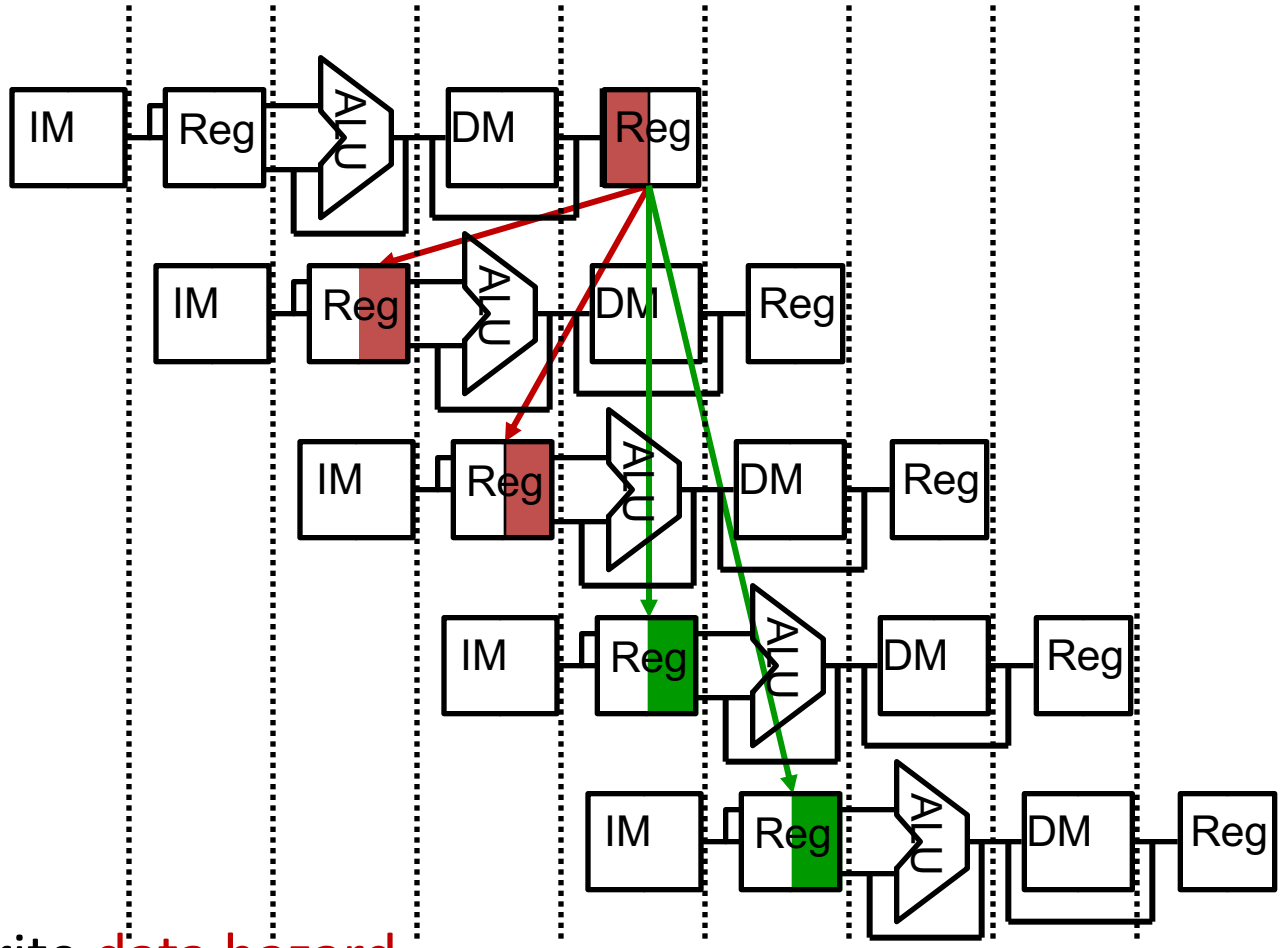
How About Register File Access?



Register Usage Can Cause Data Hazards

- Dependencies backward in time cause hazards

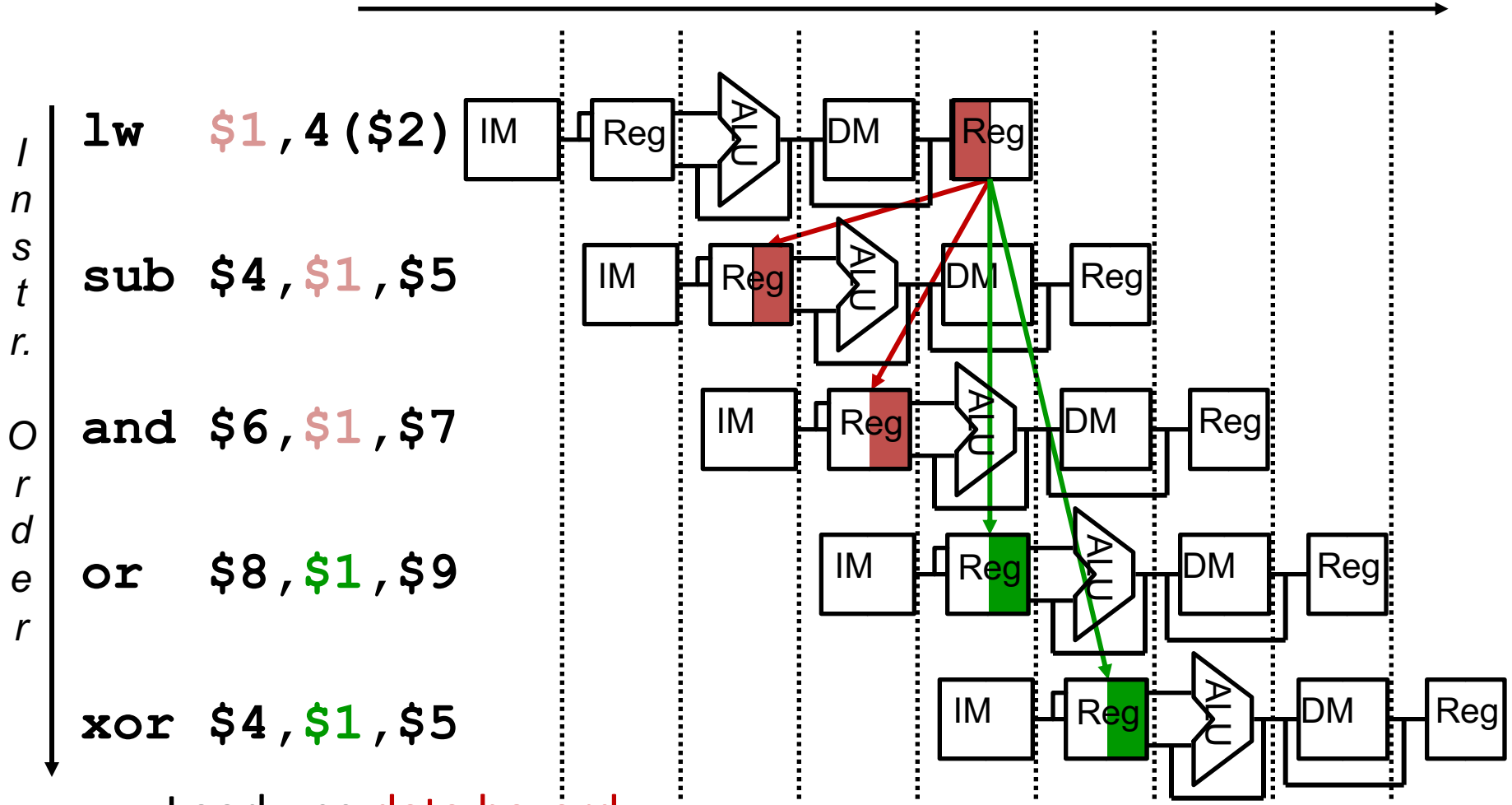
```
add $1,
sub $4,$1,$5
and $6,$1,$7
or  $8,$1,$9
xor $4,$1,$5
```



- Read before write data hazard

Loads Can Cause Data Hazards

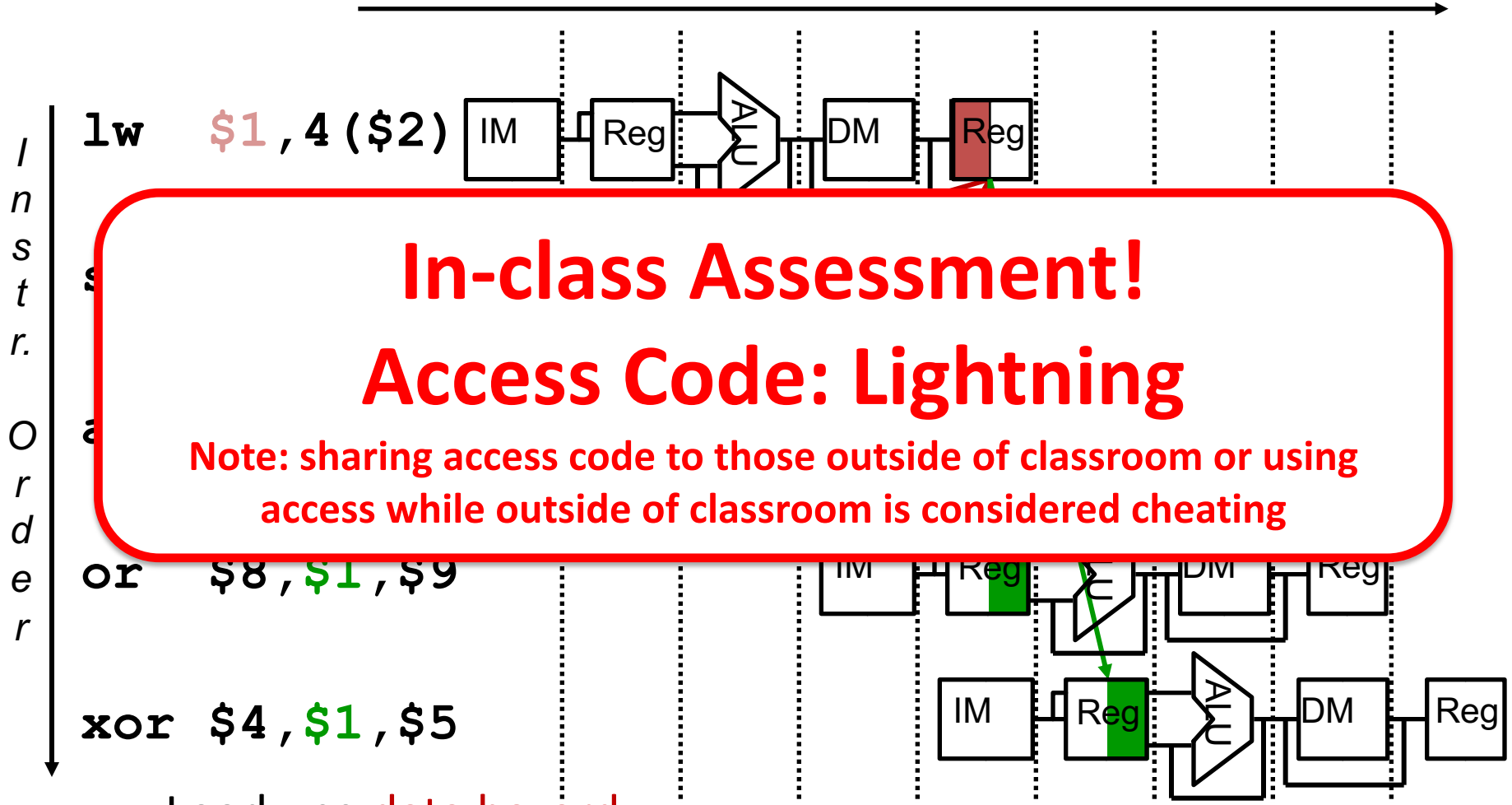
- Dependencies backward in time cause hazards



- Load-use data hazard

Loads Can Cause Data Hazards

- Dependencies backward in time cause hazards



In-class Assessment!

Access Code: Lightning

Note: sharing access code to those outside of classroom or using access while outside of classroom is considered cheating

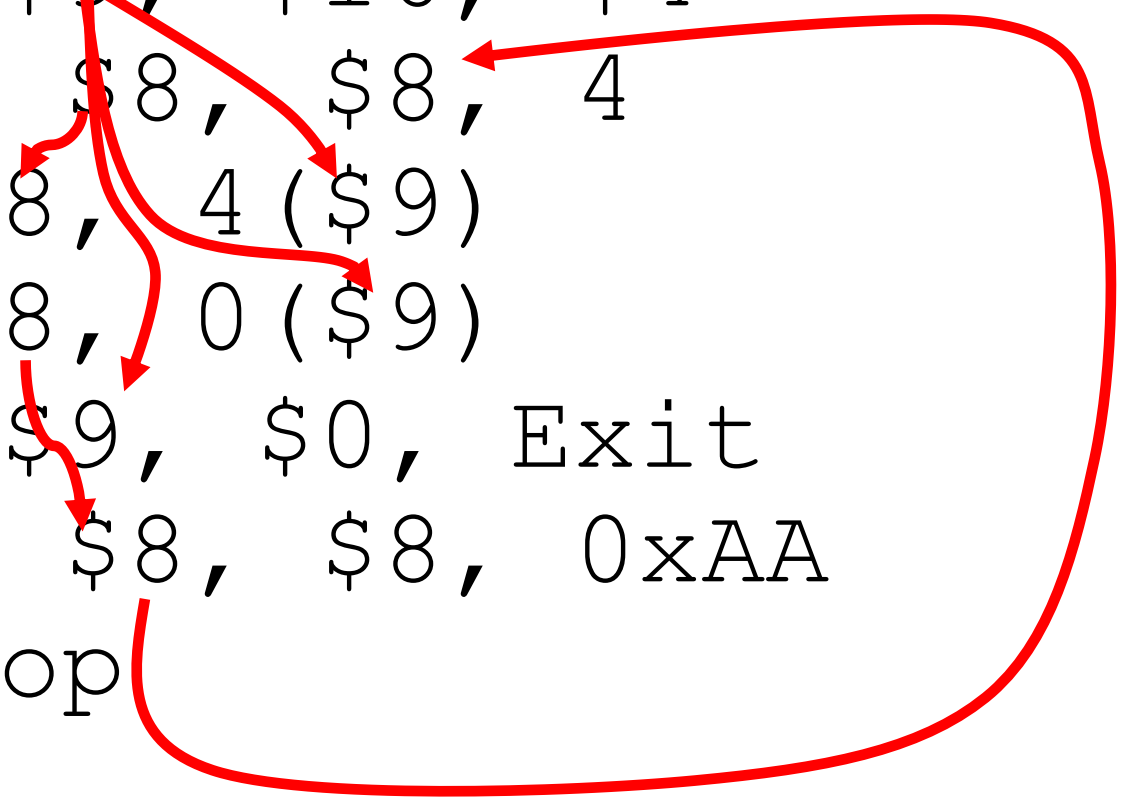
- Load-use data hazard

Data Dependencies

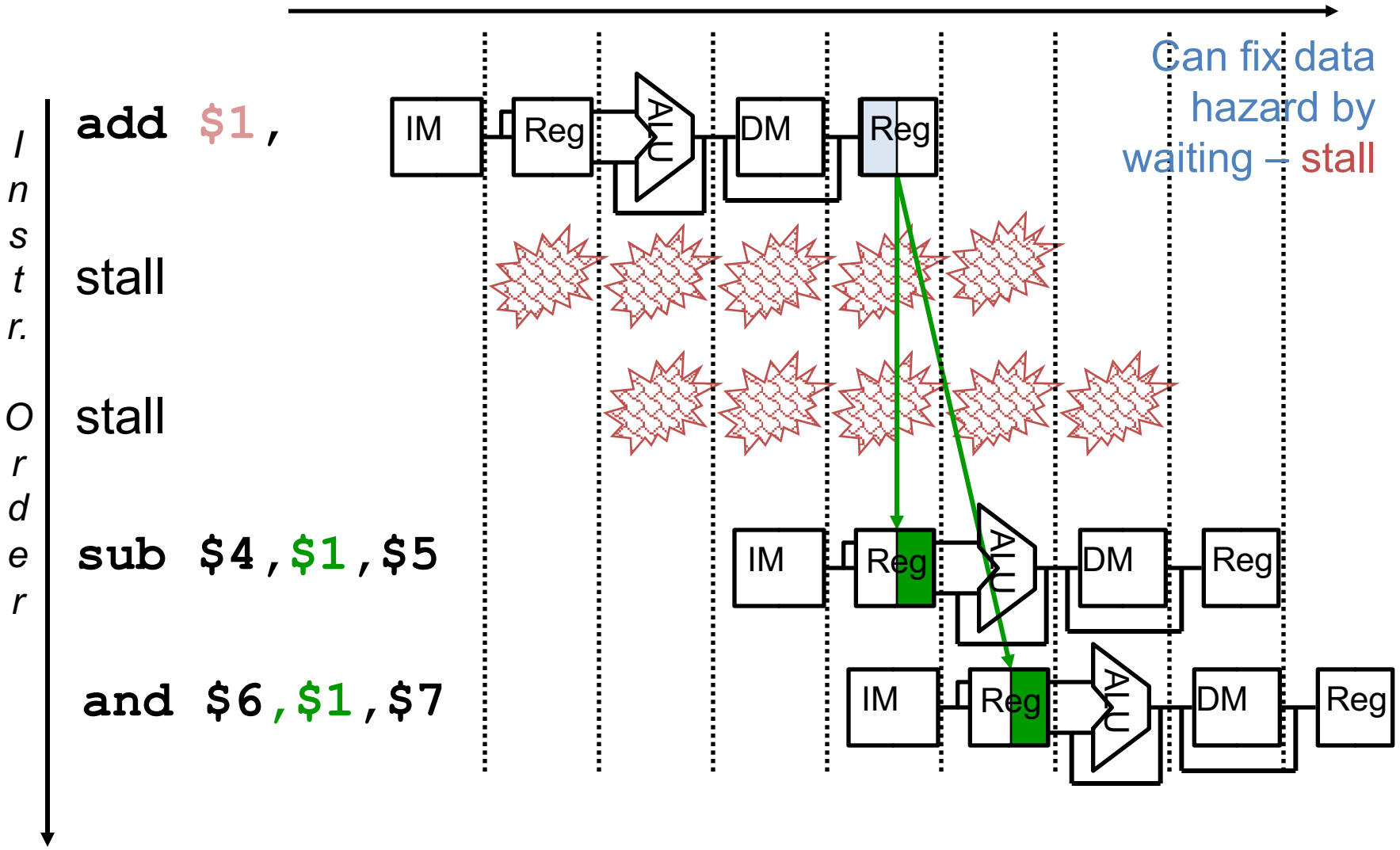
```
Loop:
1: add $9, $16, $4
2: addi $8, $8, 4
3: sw $8, 4($9)
4: lw $8, 0($9)
5: beq $9, $0, Exit
6: xori $8, $8, 0xAA
7: j Loop
Exit:
```

Data Dependencies

```
Loop:
1: add $9, $16, $4
2: addi $8, $8, 4
3: sw $8, 4($9)
4: lw $8, 0($9)
5: beq $9, $0, Exit
6: xori $8, $8, 0xAA
7: j Loop
Exit:
```



Preview: One Way to "Fix" a Data Hazard



Acknowledgments

- These slides contain material developed and copyright by:
  - Joe Zambreno (Iowa State)
  - David Patterson (UC Berkeley)
  - Mary Jane Irwin (Penn State)
  - Christos Kozyrakis (Stanford)
  - Onur Mutlu (Carnegie Mellon)
  - Krste Asanović (UC Berkeley)