Chapter 4: The Processor

Part B - Basic Pipeline

Review : Performance and Measures

- Complex question
  - How fast is the processor?
  - How fast your application runs?
  - How quickly does it respond to you?
  - How fast can you process a big batch of jobs?
  - How much power does your machine use?
- Latency vs. Throughput
  - How long to finish my program
    - Response time, elapsed time, wall clock time
    - CPU time: user and system time
  - How much work finished per unit time
- Ideal : Want high throughput, low latency

  … also, low power, cheap ($$) etc.

How Can We Make It Faster?

- Pipelining

- Start fetching and executing the next instruction before the current one has completed
  - Pipelining – (all?) modern processors are pipelined for performance
  - Remember *the* performance equation:

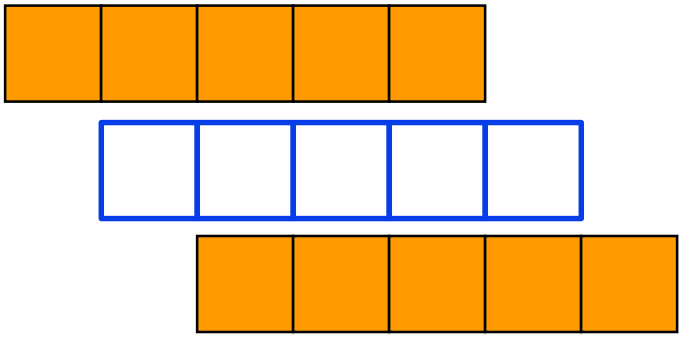
    CPU time = IC * CPI * CC

Pipelined Instruction Execution

- Sequential Execution

ADD $3, $1, $2 SUB $4, $5, $6 AND $7, $8, $9

- Pipelined Execution

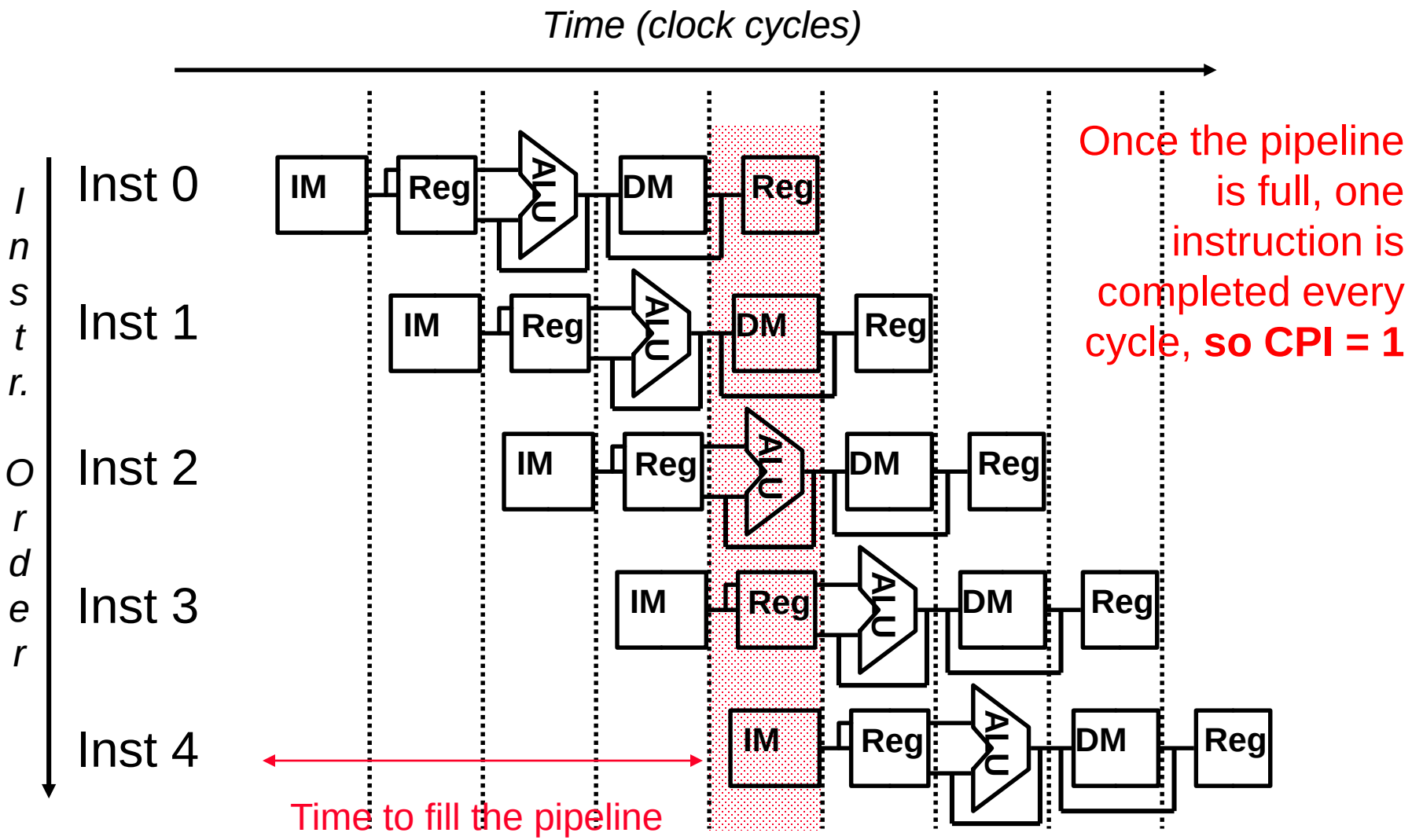


Pipelined Instruction Execution

- The speedup from pipelining

- Under *ideal* conditions and with a large number of instructions, the speedup from pipelining is approximately equal to the number of pipe stages
  - Latency vs. Throughput

Why Pipeline? For Performance!

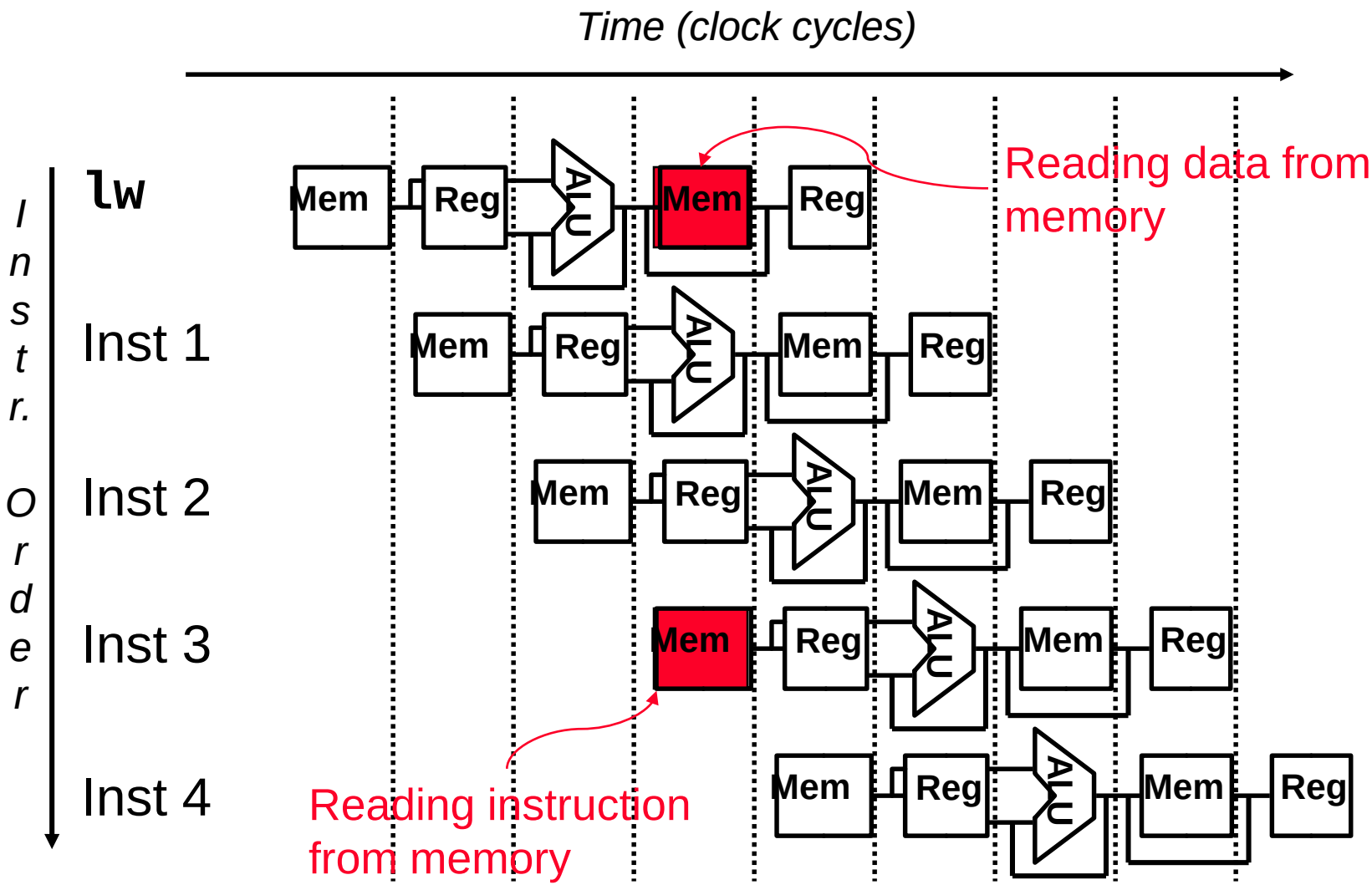


Can Pipelining Get Us Into Trouble?

- Yes: 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

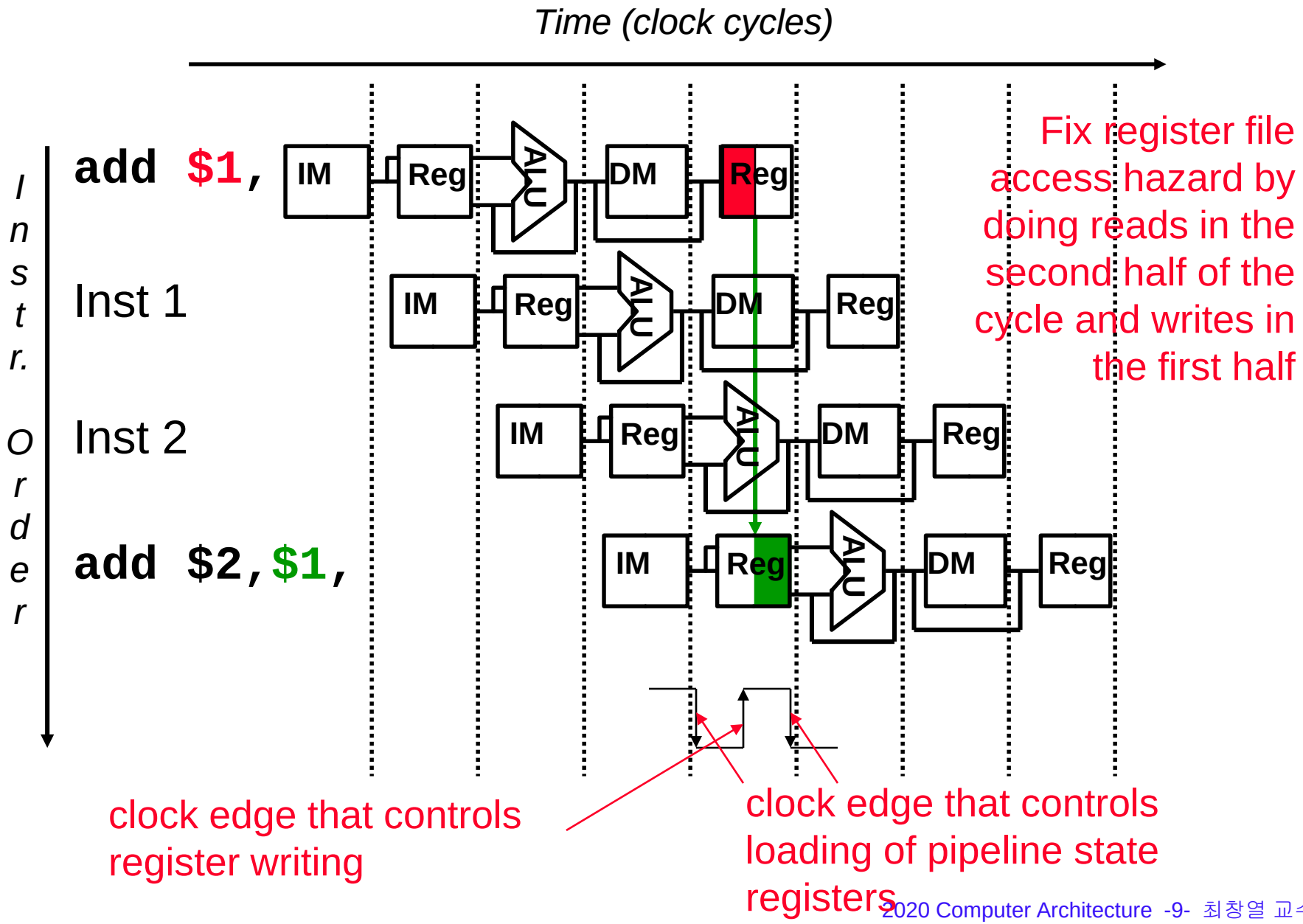
- Can usually resolve hazards by waiting
  - pipeline control must detect the hazard
  - and take action to resolve hazards

A Single Memory Would Be a Structural Hazard



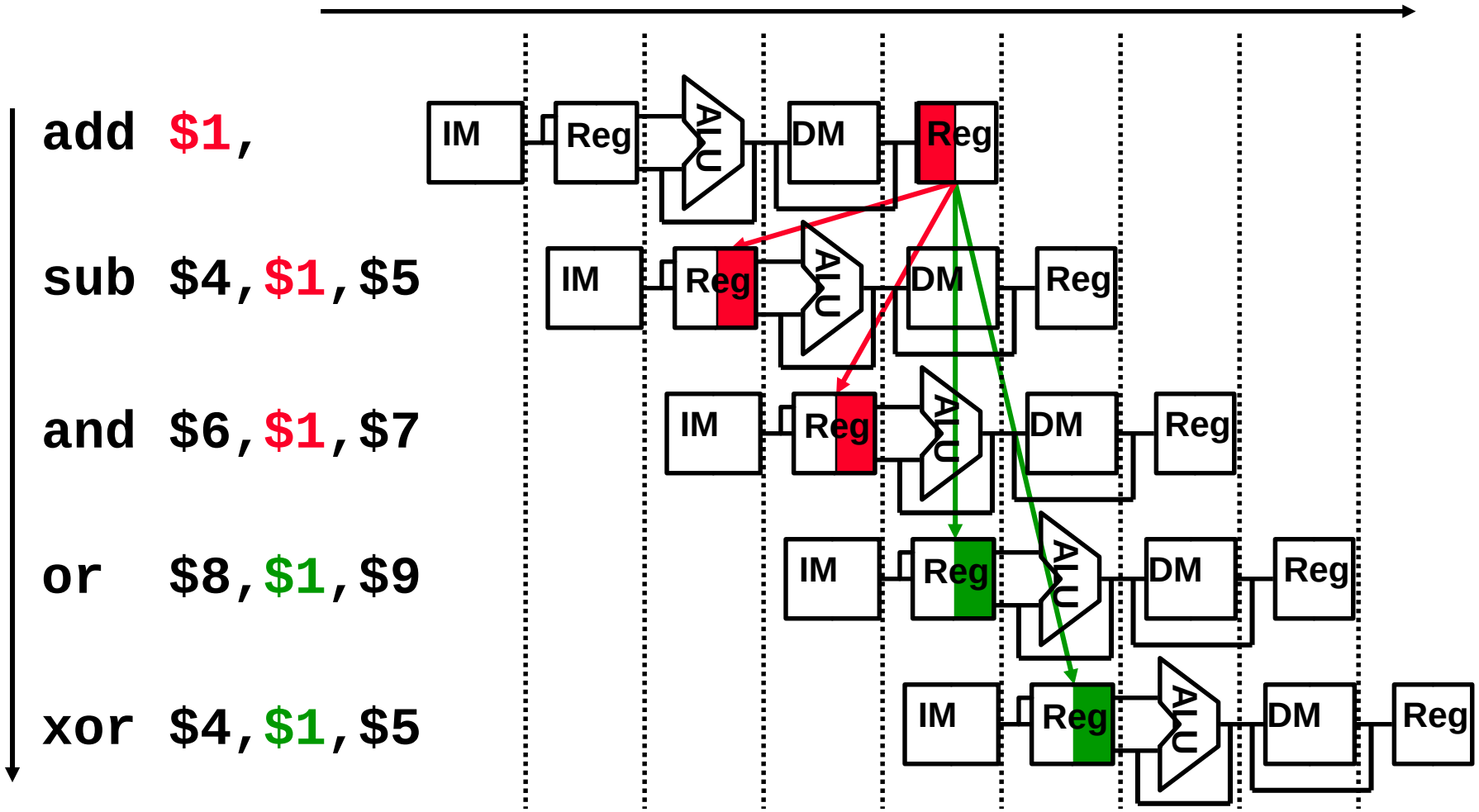
- Fix with separate instr and data memories (I$ and D$)

How About Register File Access?



Register Usage Can Cause Data Hazards

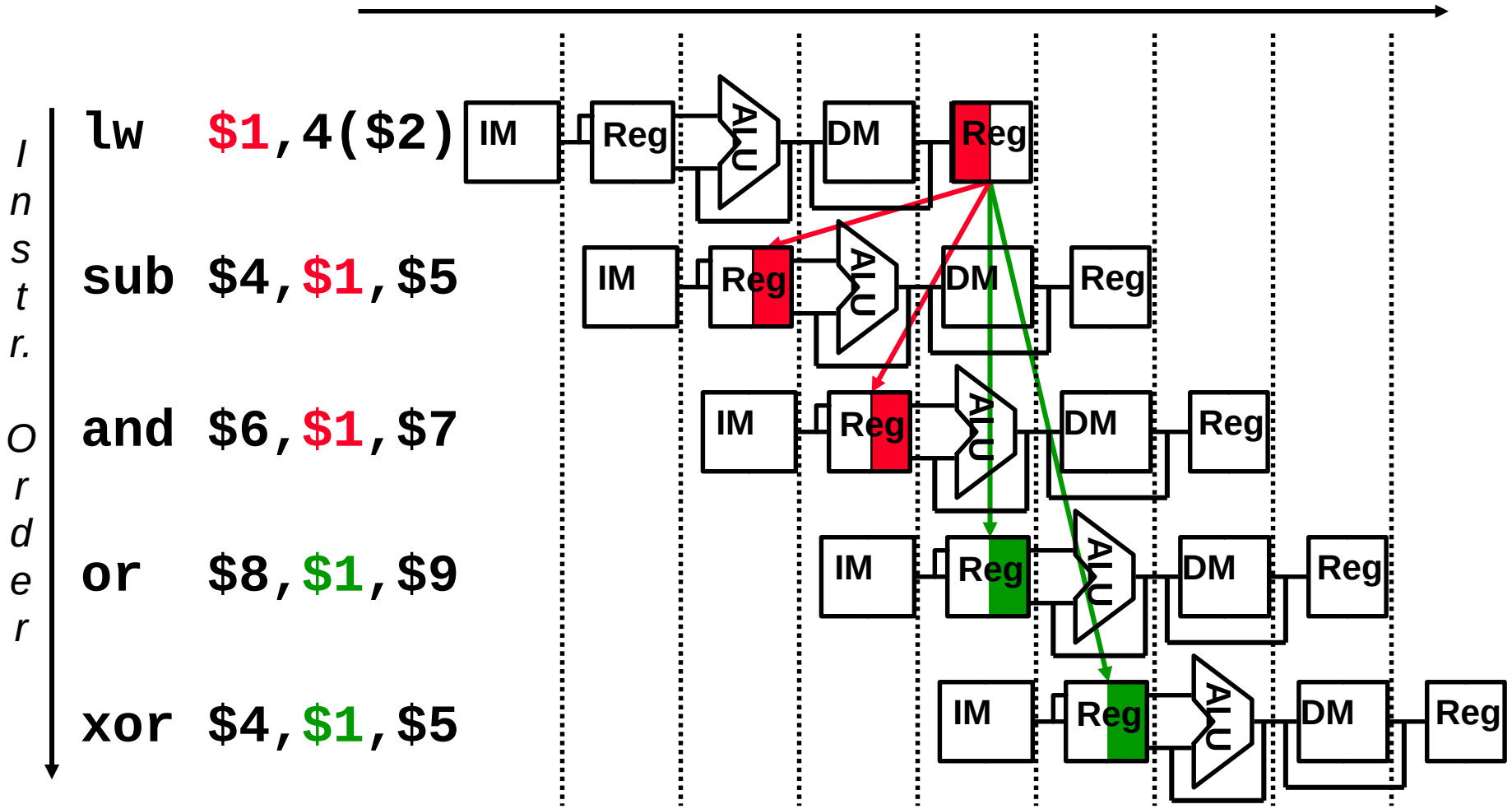
- Dependencies backward in time cause hazards



- Read before write(RBW) data hazard
- Forwarding

Loads Can Cause Data Hazards

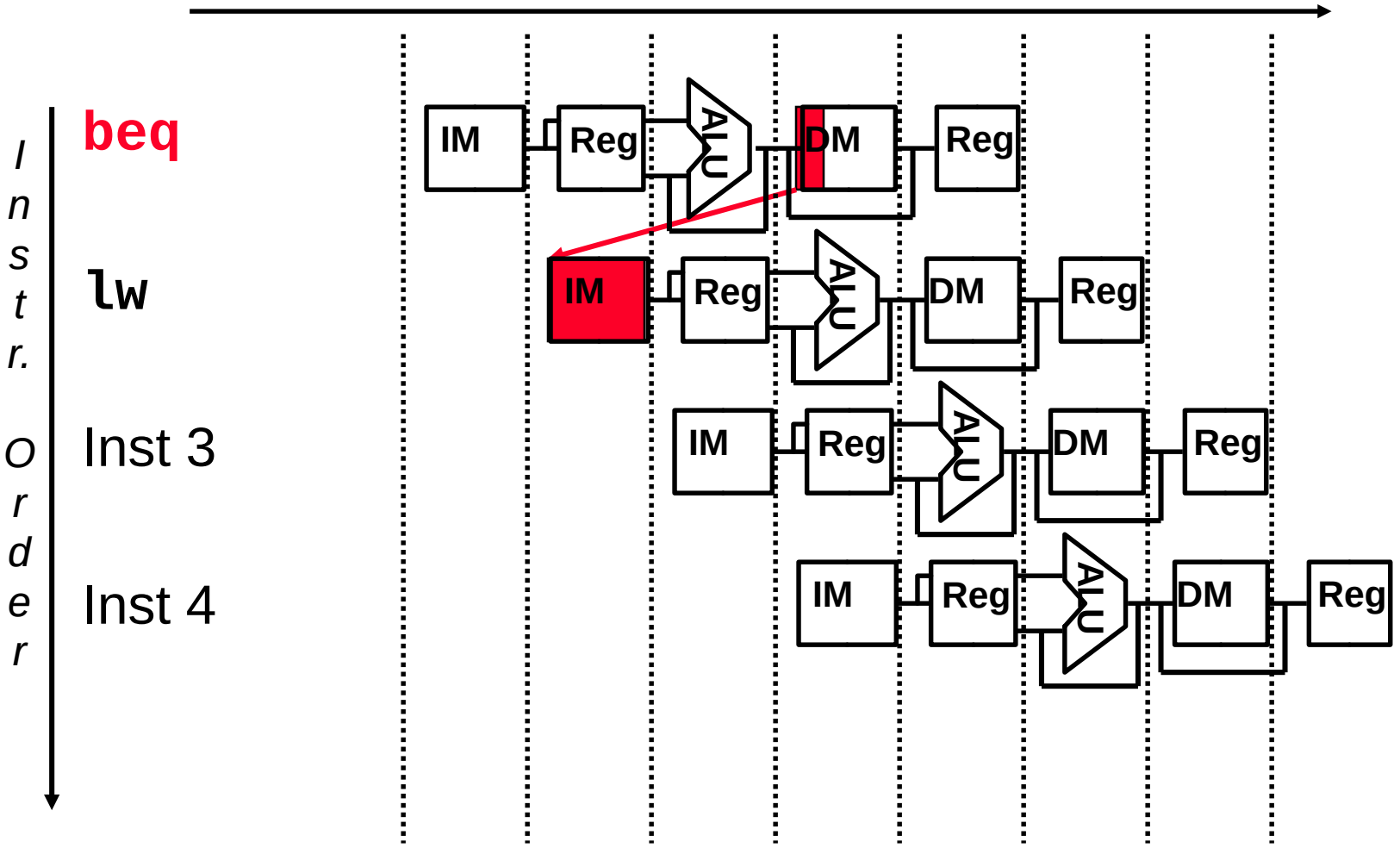
- Dependencies backward in time cause hazards



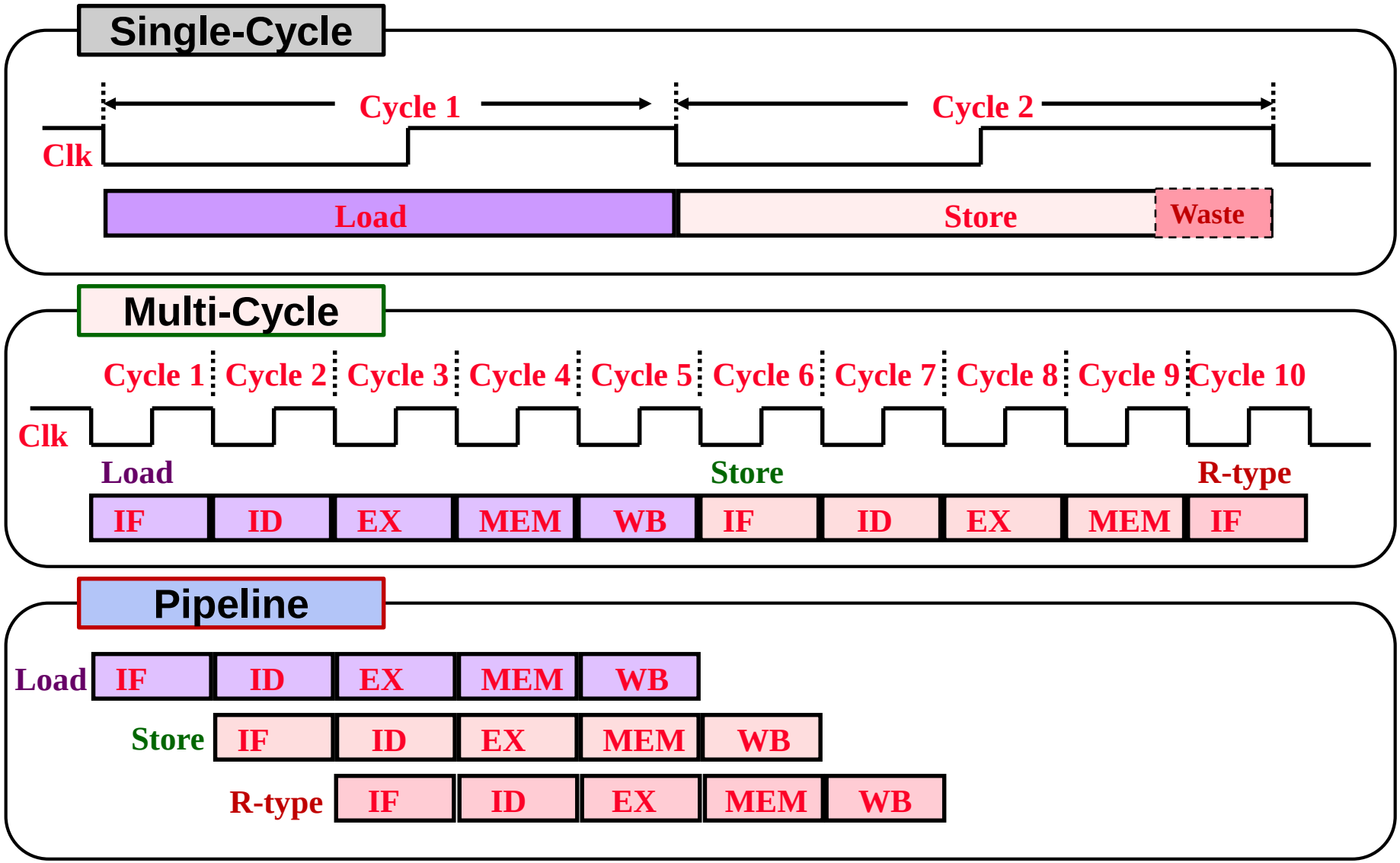
- Load-use data hazard
- Reordering, Stall

Branch Instructions Cause Control Hazards

- Dependencies backward in time cause hazards



Review : Different Implementations



- **Check Yourself**

- Understanding Program Performance

- The Big Picture

Summary

- All modern day processors use pipelining
- Pipelining doesn't help latency of single task, it helps throughput of entire workload
- Potential speedup: a CPI of 1 and fast a CC
- Pipeline rate limited by slowest pipeline stage
  - Unbalanced pipe stages makes for inefficiencies
  - The time to "fill" pipeline and time to "drain" it can impact speedup for deep pipelines and short code runs
- Must detect and resolve hazards