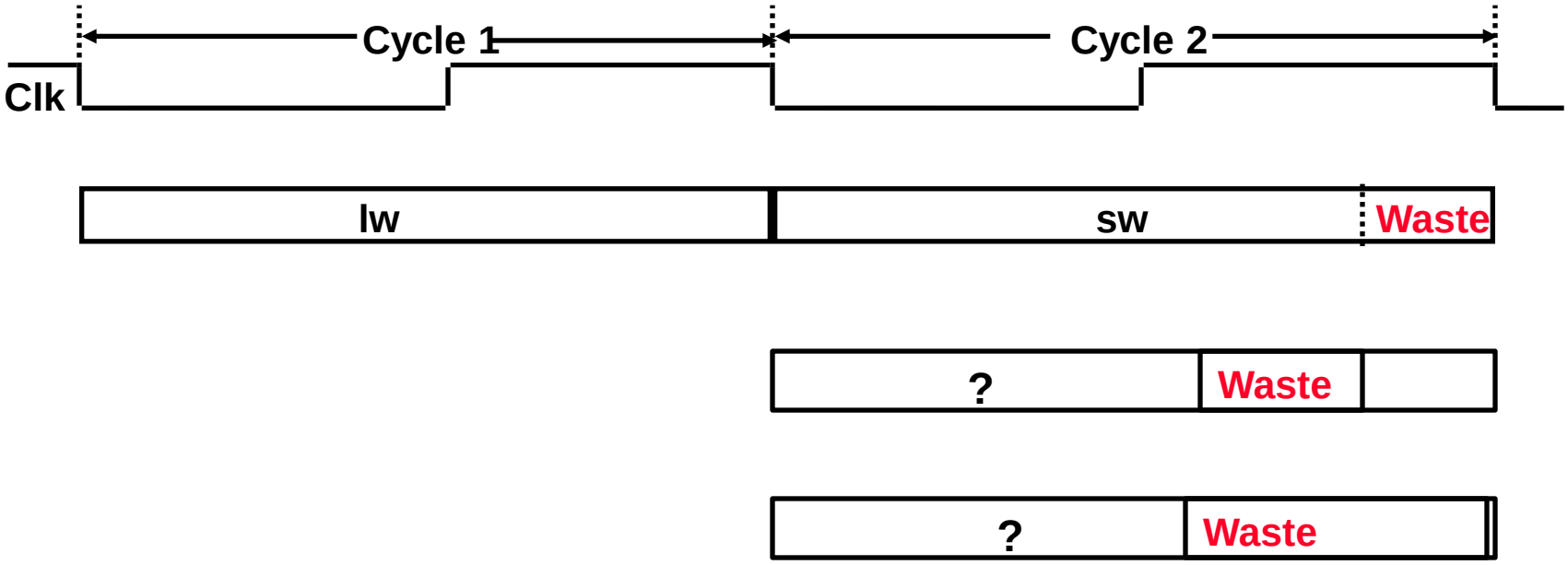
Instruction Critical Paths

- What is the clock cycle time?
  - Instruction and Data Memory (200 ps)
  - ALU and adders (200 ps)
  - Register File access (reads or writes) (100 ps)

Instr.	I Mem	Reg Rd	ALU Op	D Mem	Reg Wr	Total
R-type	200	100	200		100	600
load	200	100	200	200	100	800
store	200	100	200	200		700
beq	200	100	200			500
jump	200					200

Single Cycle Disadvantages & Advantages

- Uses the clock cycle inefficiently
  - the clock cycle with respect to the slowest instruction



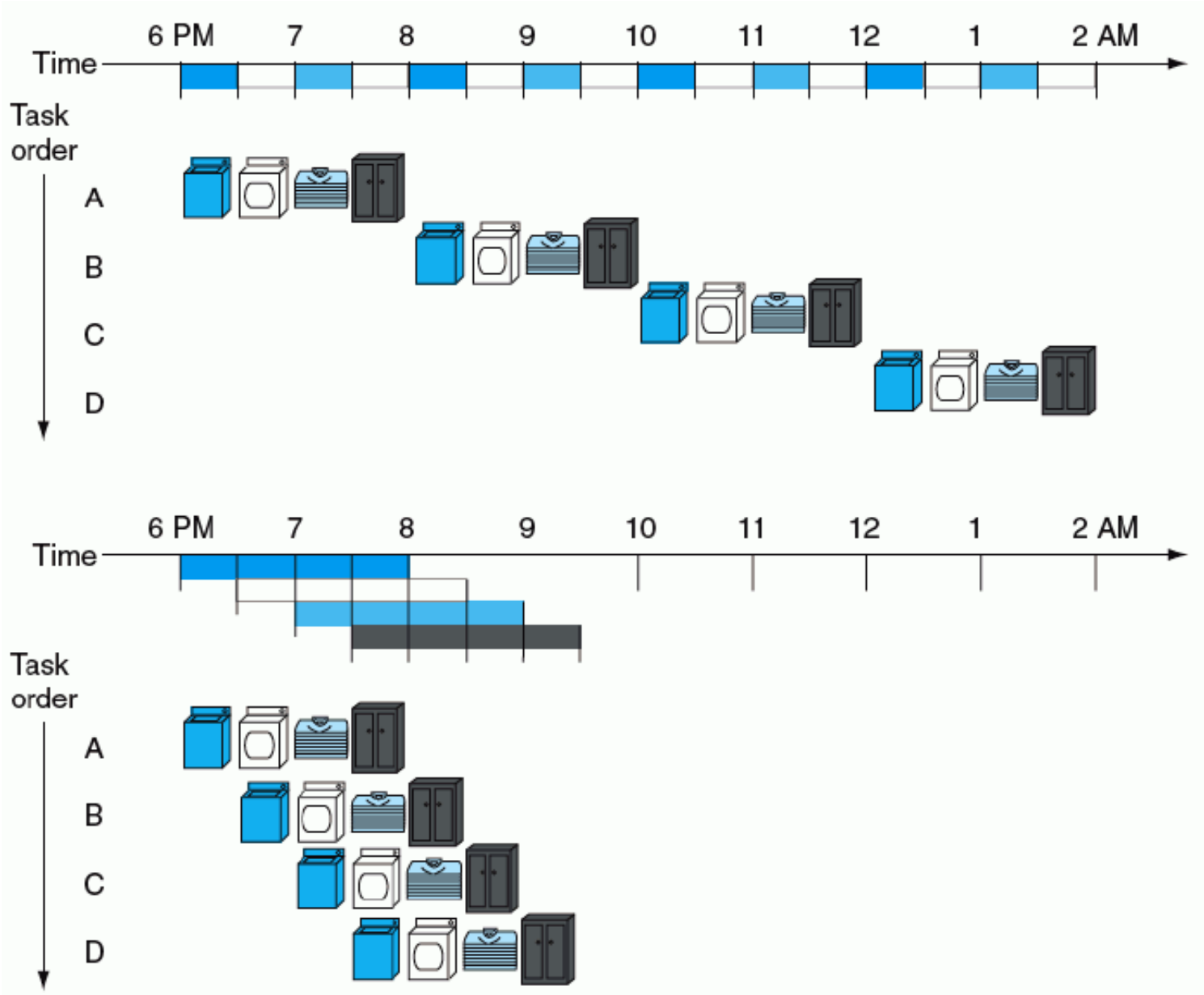
How Can We Make It Faster?

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

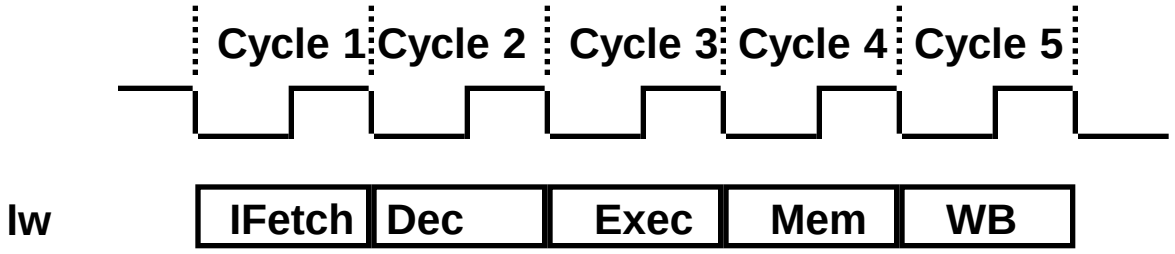
    CPU time = CPI * CC * IC

- Under *ideal* conditions and with a large number of instructions, the speedup from pipelining is approximately equal to the number of pipe stages
  - A five stage pipeline is nearly five times faster because the CC is nearly five times faster

Pipelining


6 PM
7
8
9
10
11
12
1
2 AM
Time
Task order
A
B
C
D
6 PM
7
8
9
10
11
12
1
2 AM
Time
Task order
A
B
C
D

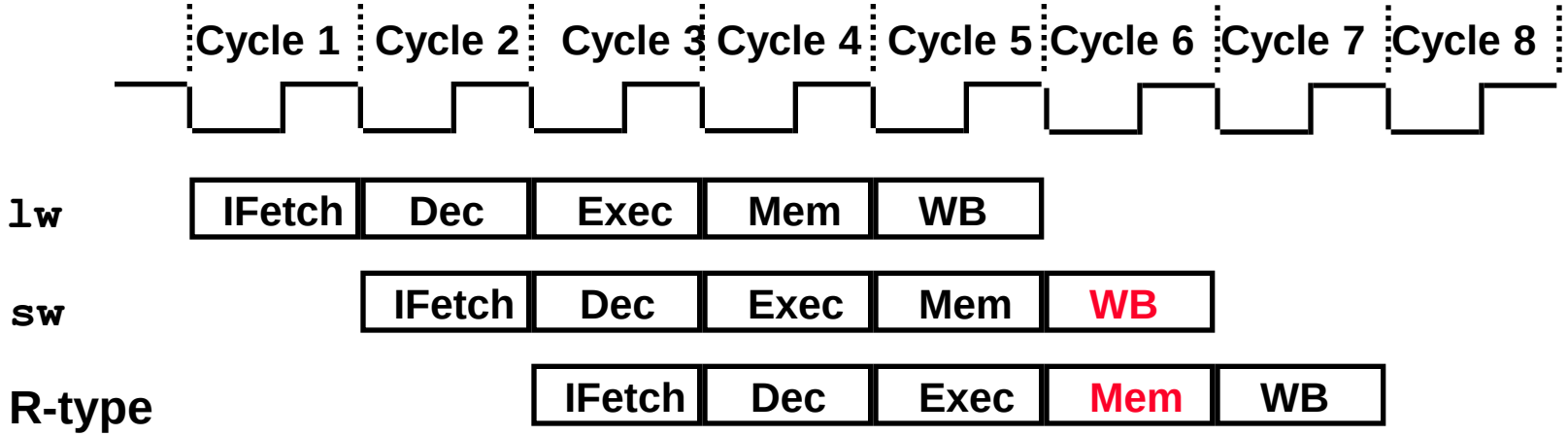
The Five Stages of Load Instruction



- IFetch: Instruction Fetch and Update PC
- Dec: Registers Fetch and Instruction Decode
- Exec: Execute R-type; calculate memory address
- Mem: Read/write the data from/to the Data Memory
- WB: Write the result data into the register file

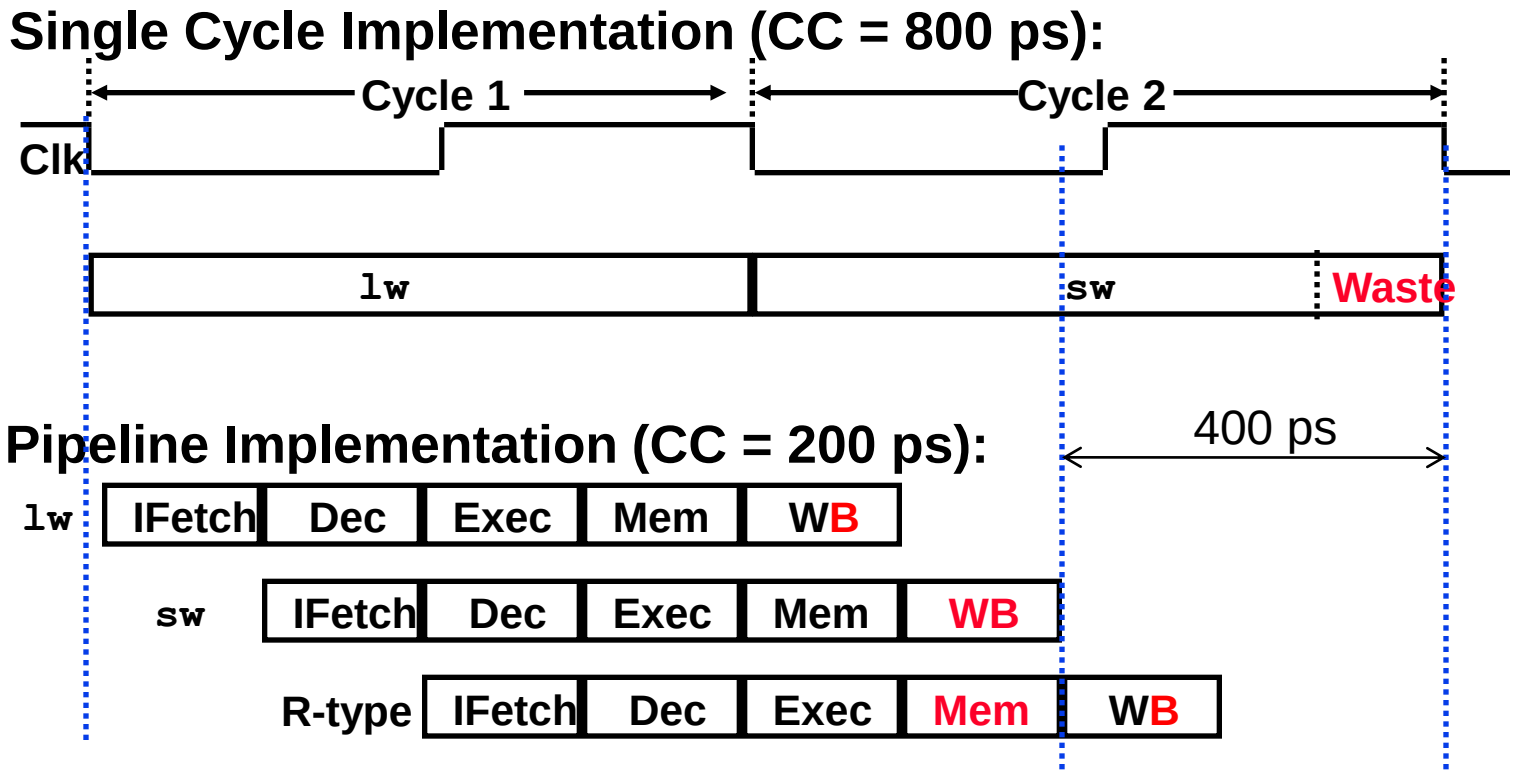
A Pipelined MIPS Processor

- Start the next instruction before the current one has completed
  - improves throughput - total amount of work done in a given time



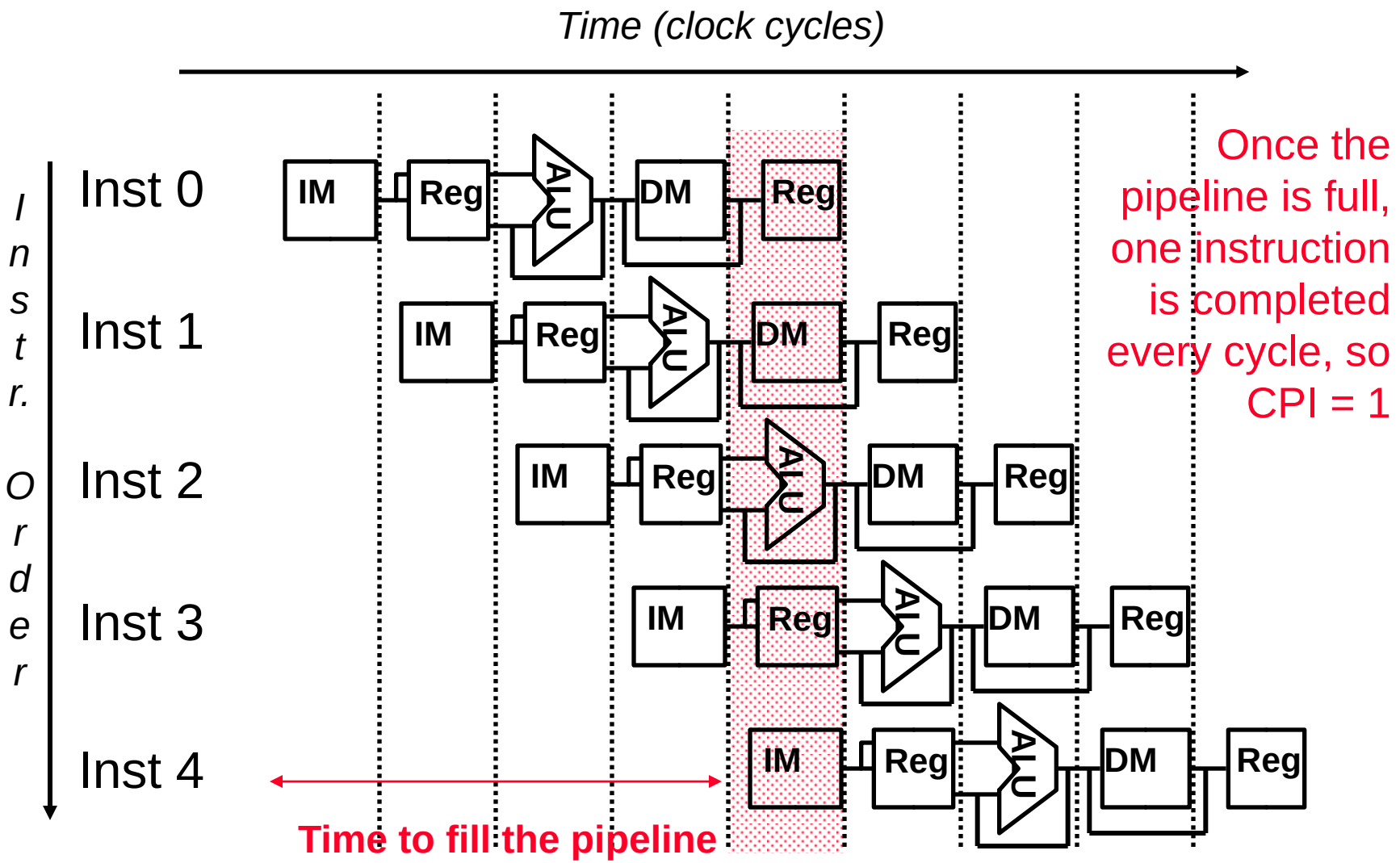
- clock cycle (pipeline stage time) is limited by the **slowest** stage
  - for some stages don't need the whole clock cycle (e.g., WB)
  - for some instructions, some stages are wasted cycles (i.e., nothing is done during that cycle for that instruction)

Single Cycle versus Pipeline

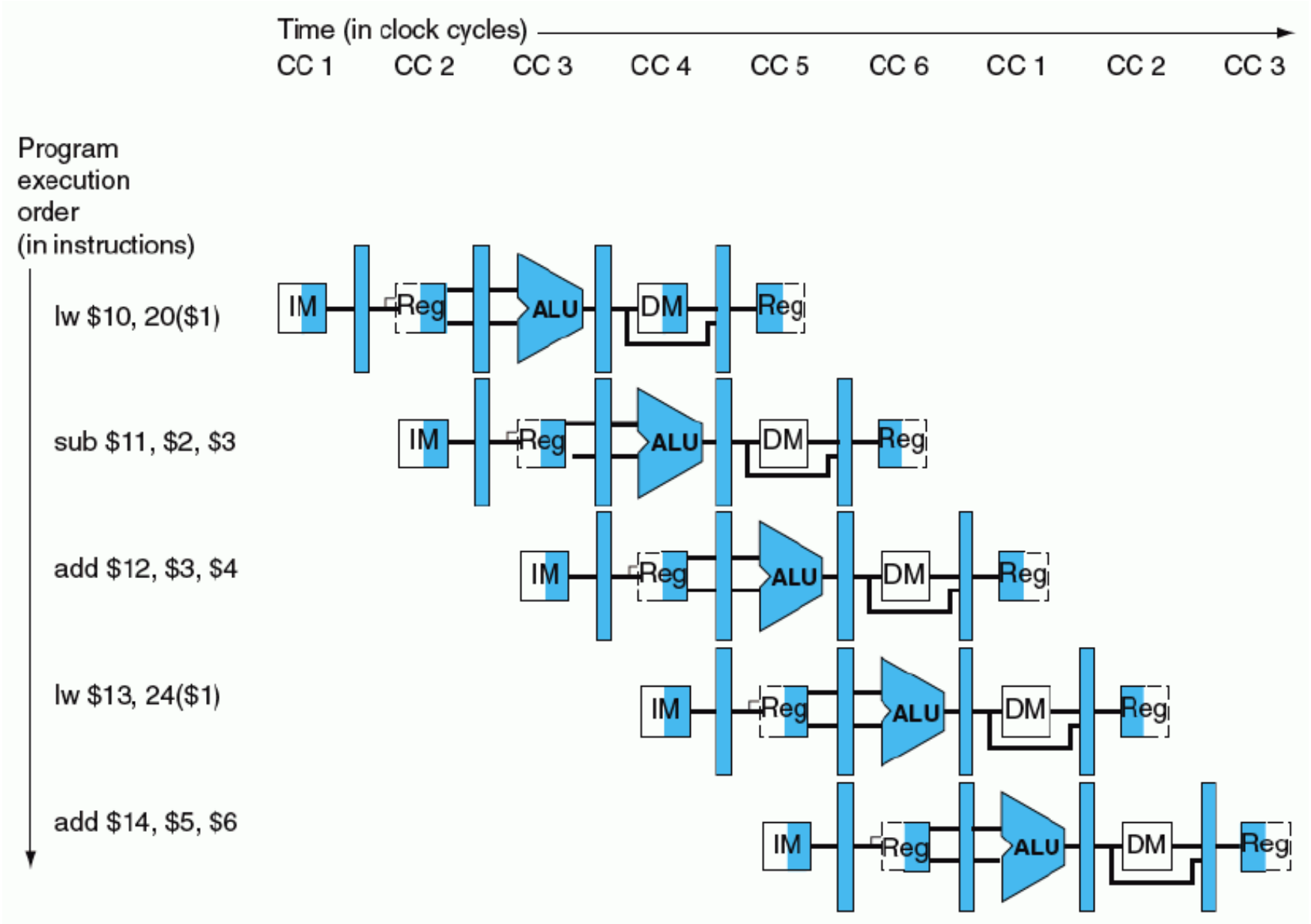


- To complete an entire instruction
  - Pipelined case takes 1000 ps
  - Single cycle takes 800 ps

Why Pipeline? For Performance!



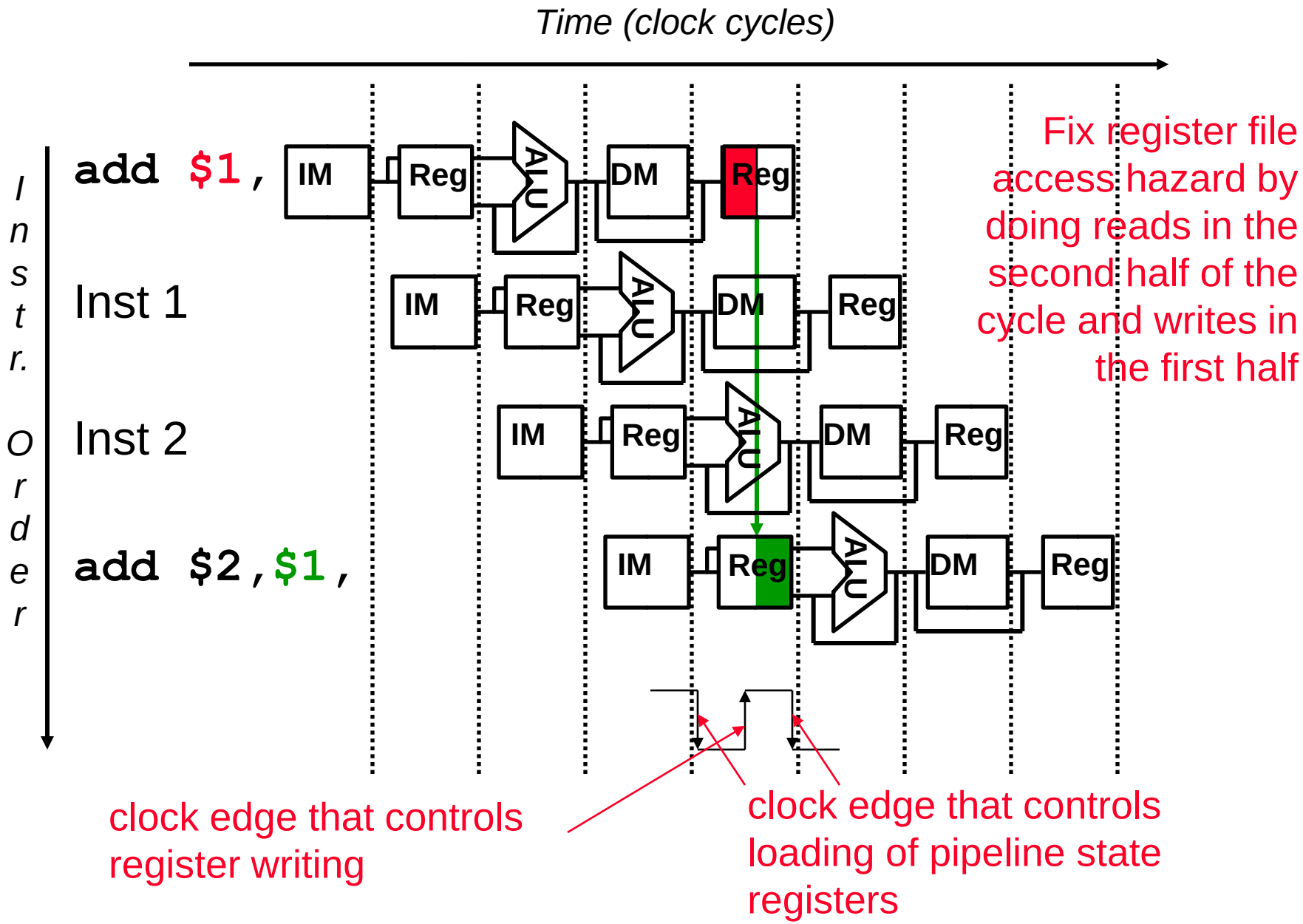
Pipelined multi-clock-cycle



Can Pipelining Get Us Into Trouble?

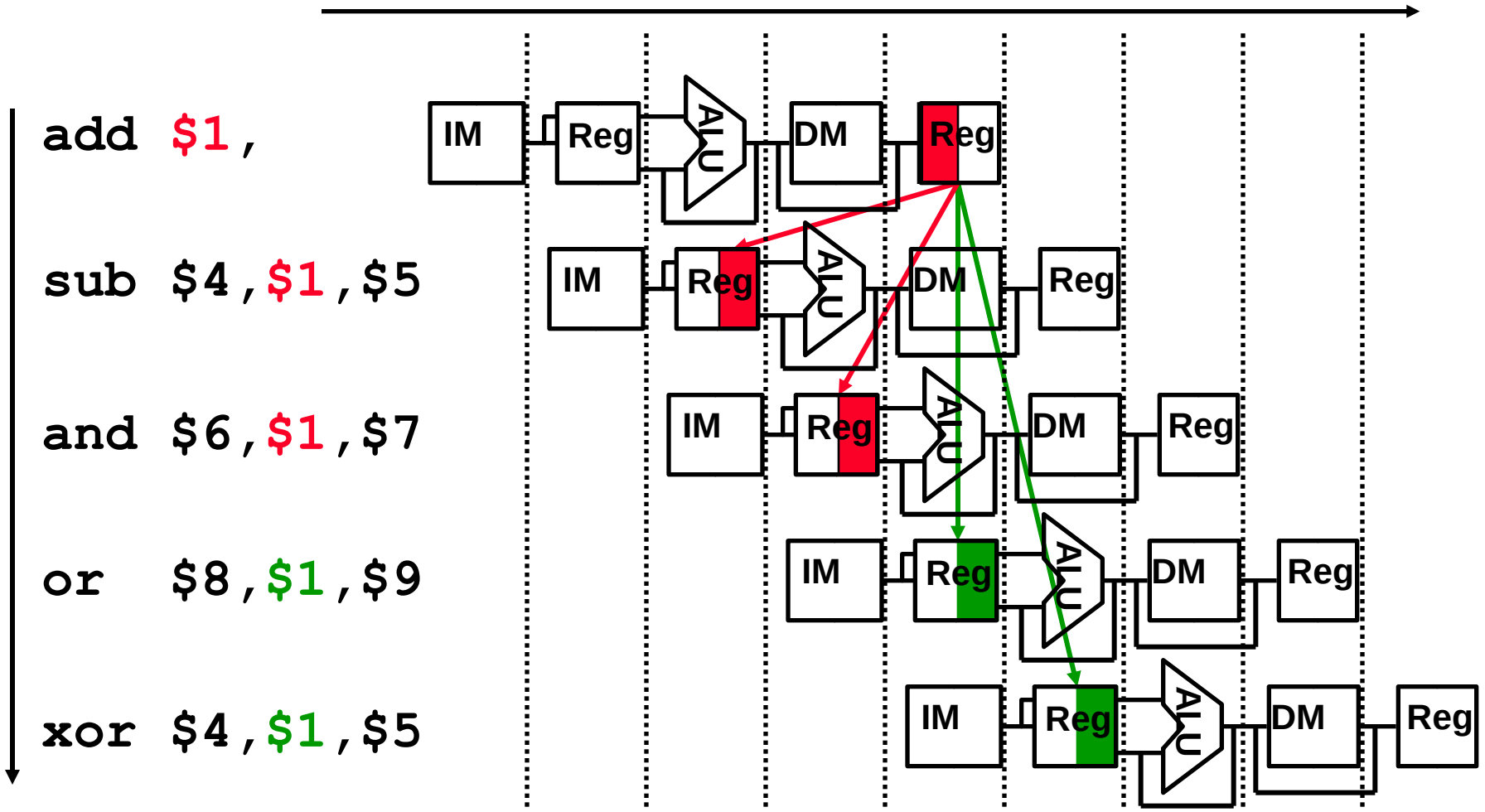
- Yes: Pipeline Hazards
  - data hazards: attempt to use data before it is ready
    - An instruction's source operand(s) are produced by a prior instruction still in the pipeline
  - control hazards: attempt to make a decision about program control flow before the condition has been evaluated and the new PC target address calculated
    - branch and jump instructions

Register File Access?



Register Usage Can Cause Data Hazards

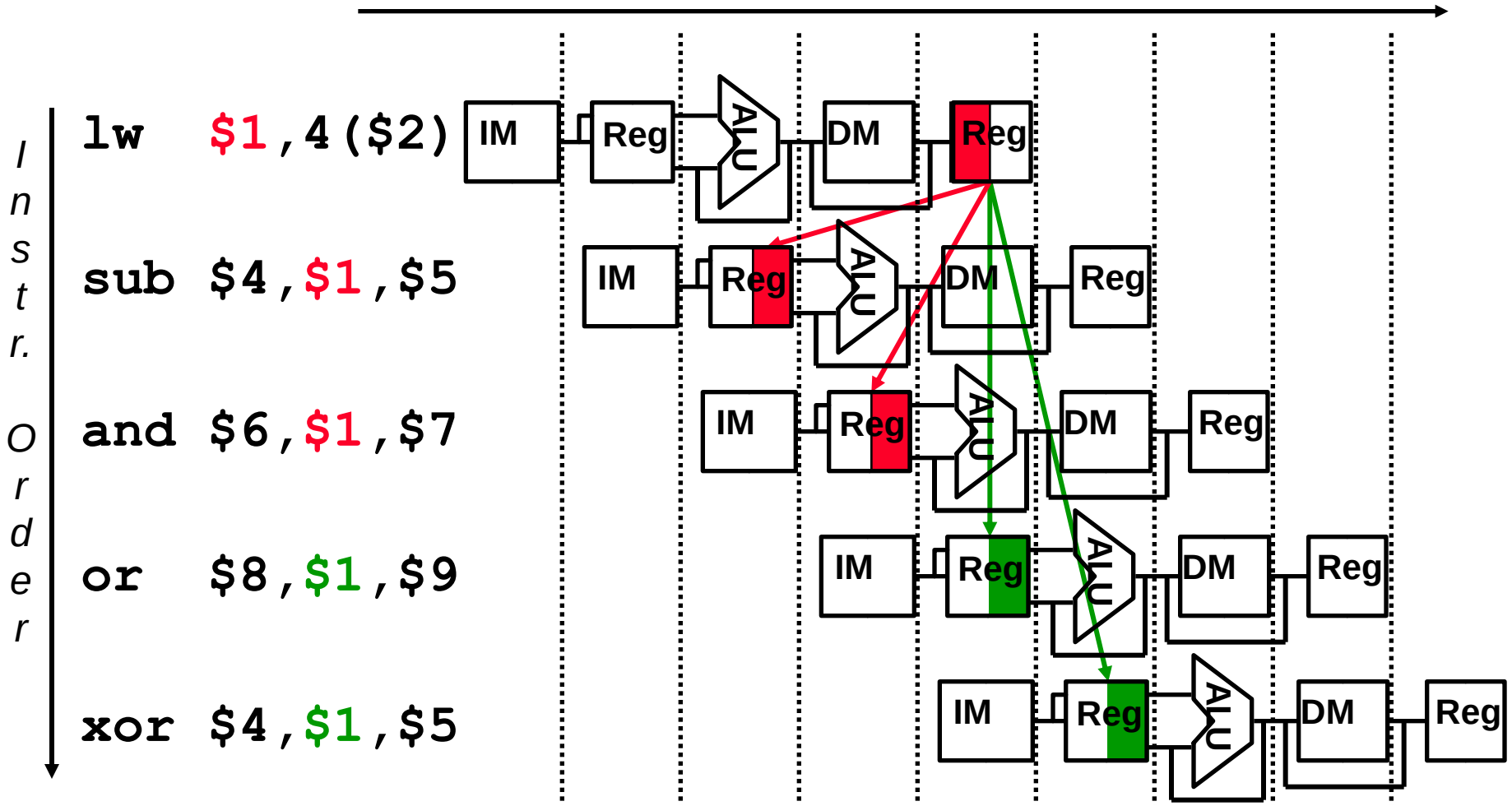
- Dependencies backward in time cause hazards



- Read before write data hazard

Loads Can Cause Data Hazards

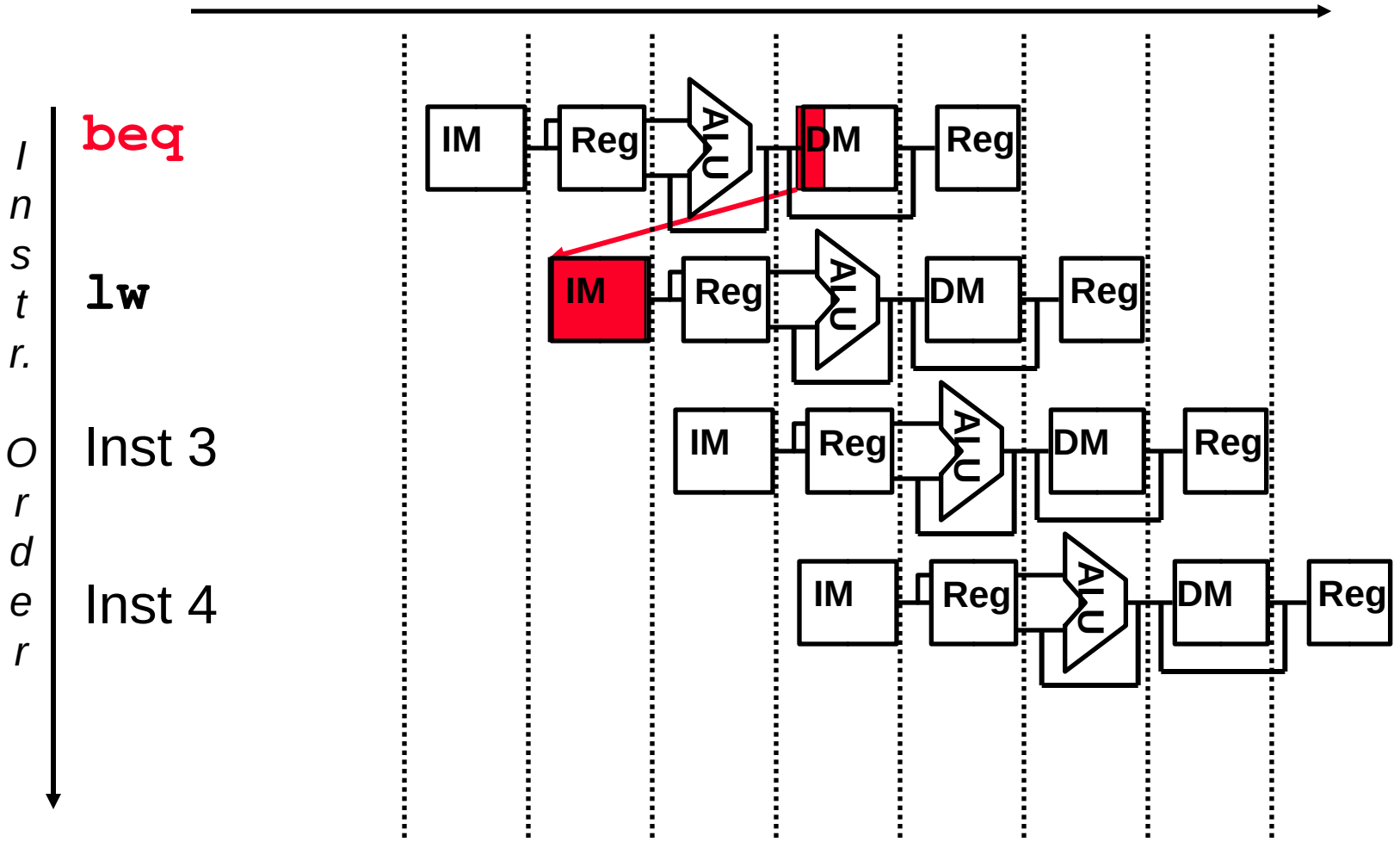
- Dependencies backward in time cause hazards



- Load-use data hazard

Branch Instructions Cause Control Hazards

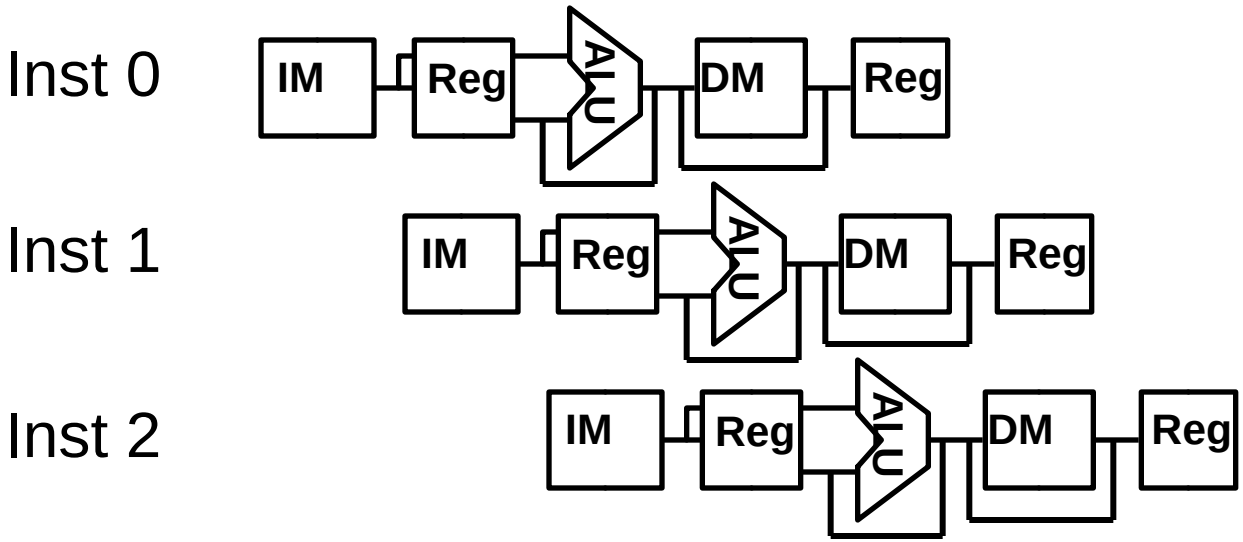
- Dependencies backward in time cause hazards



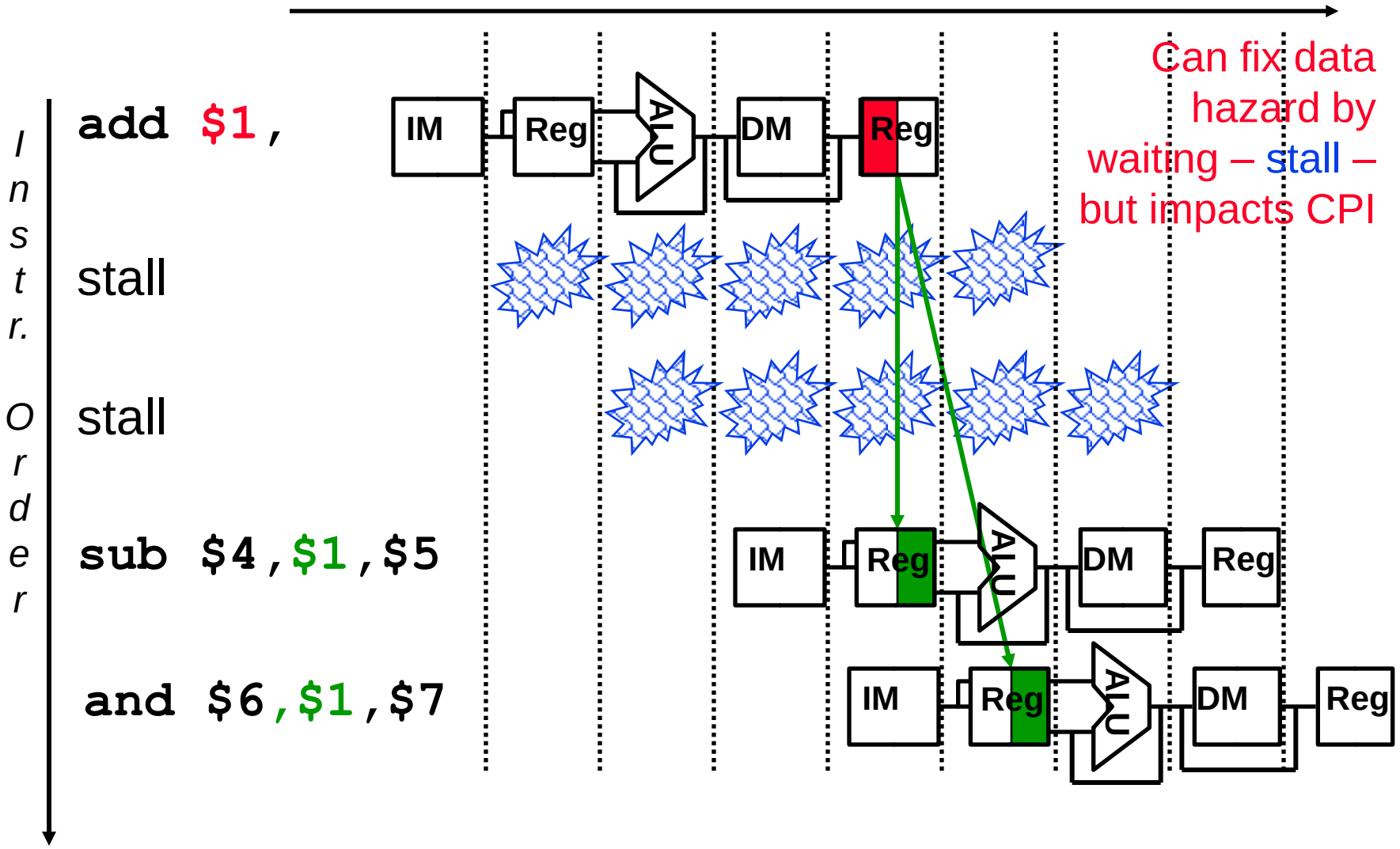
Resolving Hazards

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

Waiting / Stalling negatively affects CPI (makes CPI less than the ideal of 1)



One Way to "Fix" a Data Hazard



Summary

- All modern day processors use pipelining for performance (a CPI of 1 and a fast CC)
- Pipeline clock rate limited by slowest pipeline stage – so designing a balanced pipeline is important
- Must detect and resolve hazards