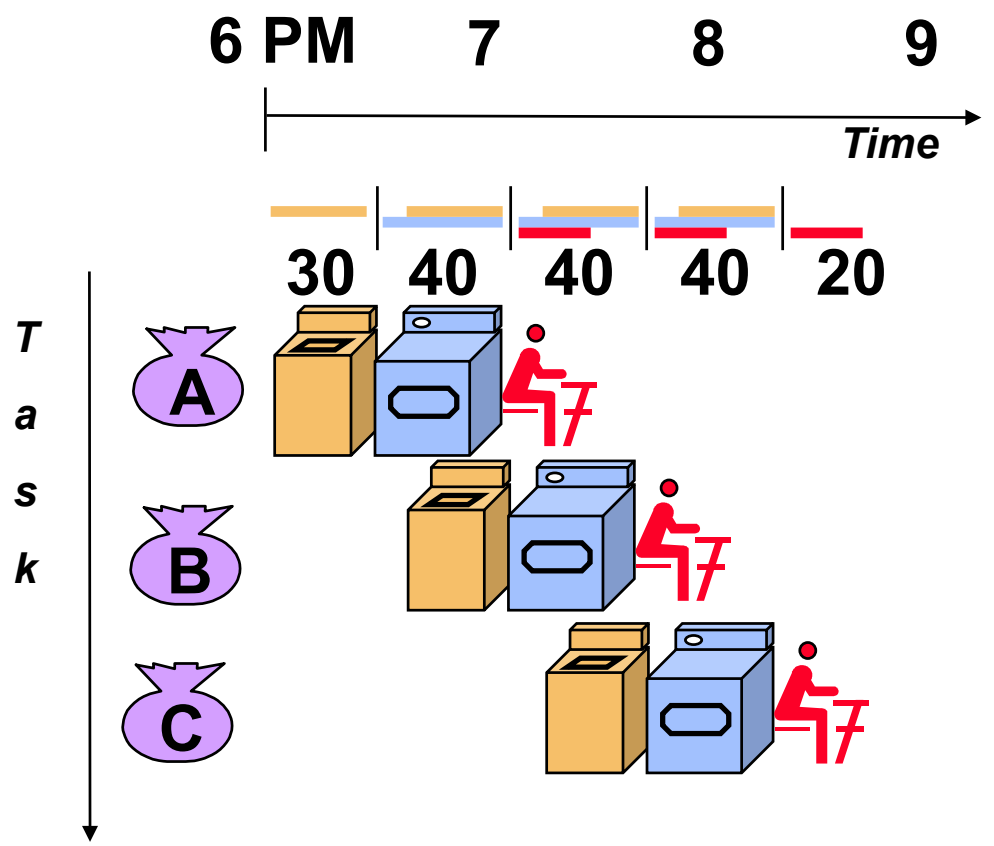
# CPT_S 260 Intro to Computer Architecture
# Lecture 33

# Pipeline Hazards
# April 4, 2022

**Ganapati Bhat**
**School of Electrical Engineering and Computer Science**
**Washington State University**

# Pipelined Laundry

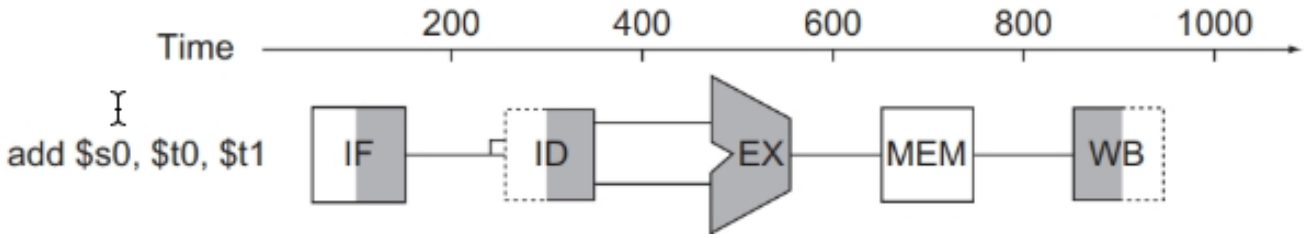



- Pipelined laundry takes 2 hours and 50 minutes

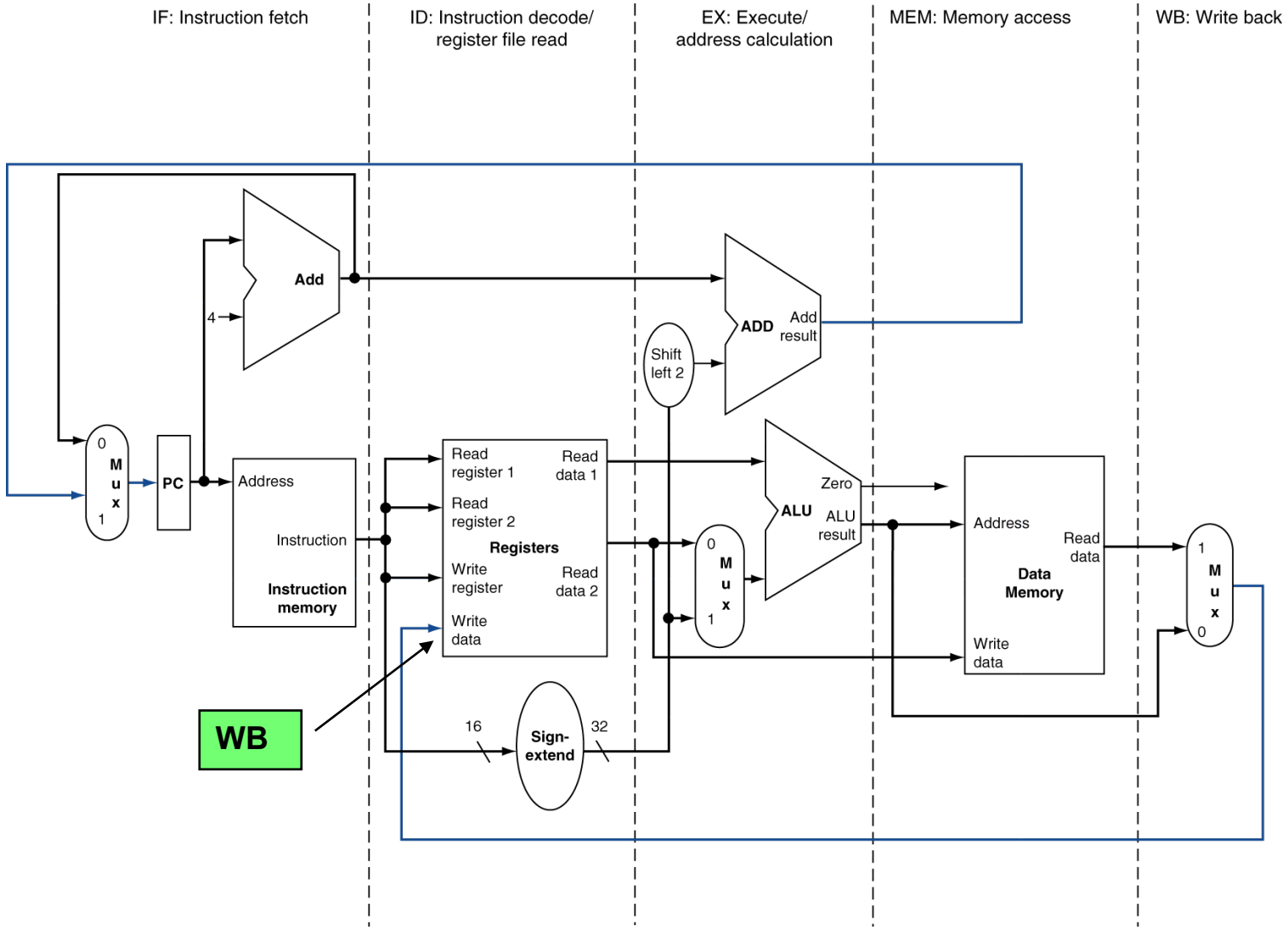
- Pipelining doesn't help latency of single task, it helps throughput of entire workload
- Pipeline rate limited by slowest pipeline stage
- Multiple tasks operating simultaneously using different resources
- Potential speedup = Number of pipeline stages
- Unbalanced lengths of pipe stages reduces speedup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

# Recap: MIPS Pipeline

- **Five stages, one step per stage**
  1. IF: Instruction fetch from memory
  2. ID: Instruction decode & register read
  3. EX: Execute operation or calculate address
  4. MEM: Access memory operand
  5. WB: Write result back to register



# Recap: Overall Idea

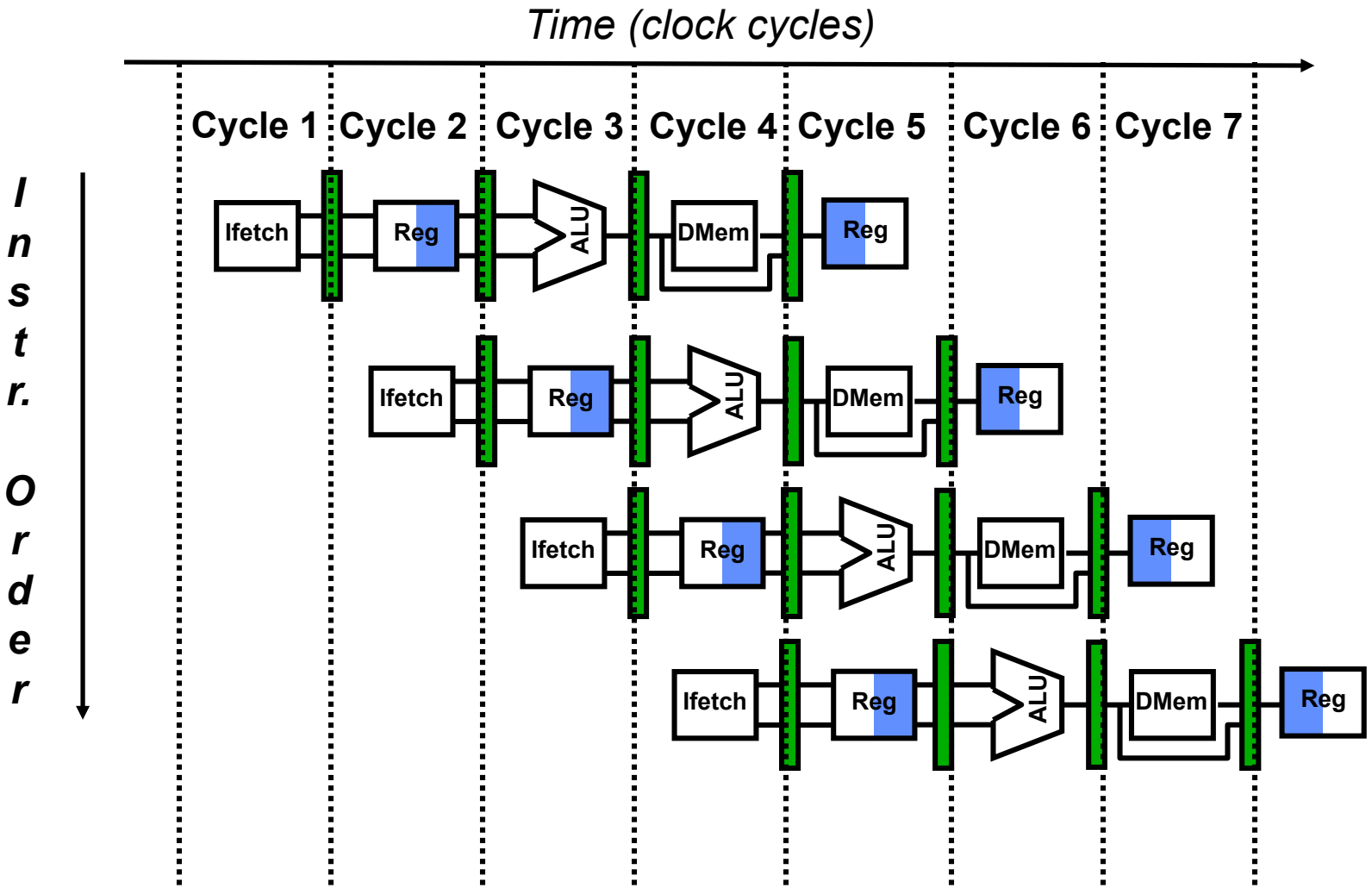


# Pipeline Speedup

- **Pipelining improves performance by increasing instruction throughput, as opposed to decreasing the execution time of an individual instruction**

- ***instruction throughput* is the important metric because real programs execute billions of instructions.**

# Visualizing Pipelining



# Pipeline Hazards

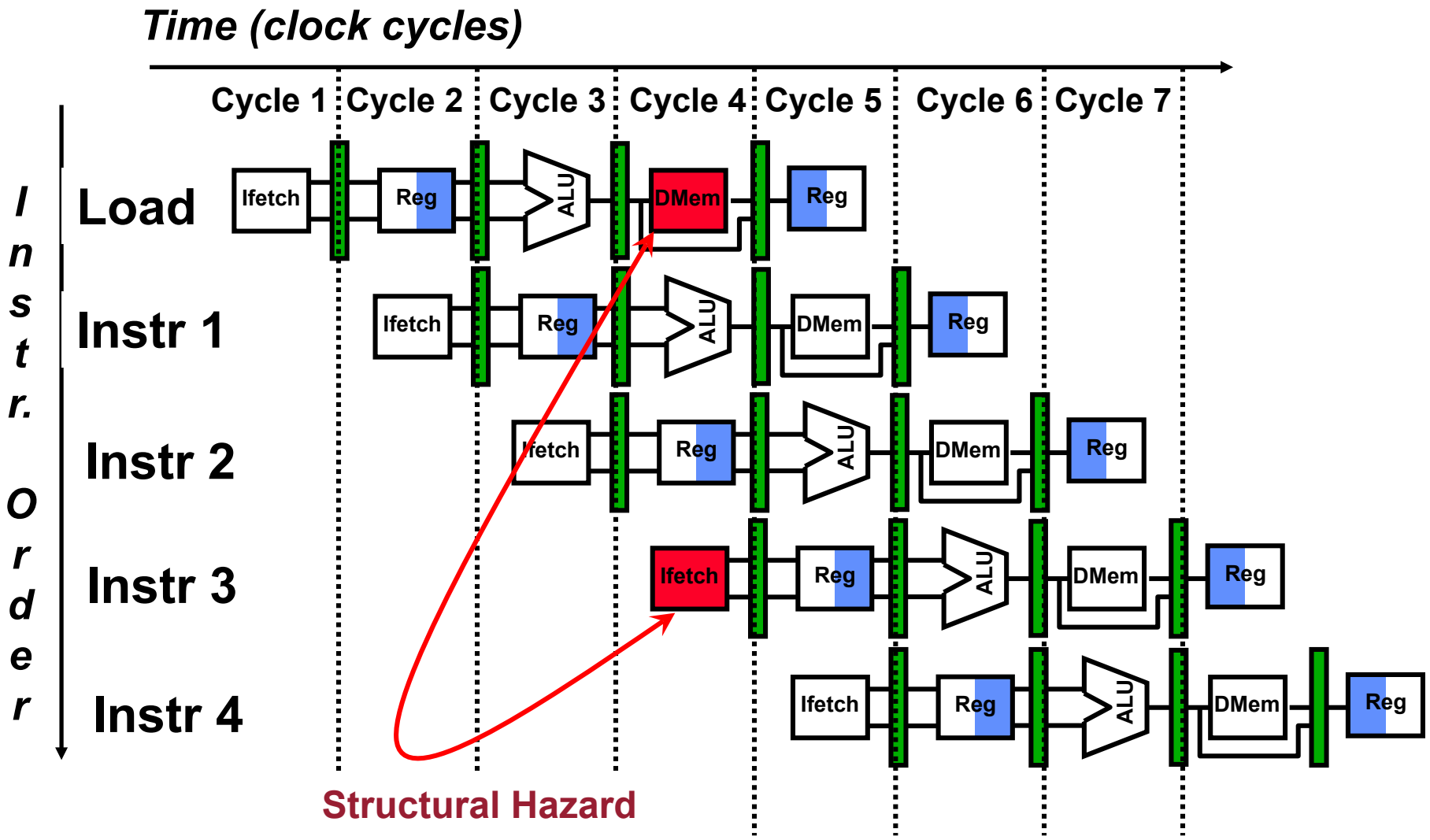
# Hazards

- **Situations that prevent starting the next instruction in the next cycle**
- **Structure hazards**
  - A required resource is busy
- **Data hazard**
  - Need to wait for previous instruction to complete its data read/write
- **Control hazard**
  - Deciding on control action depends on previous instruction

# Structure Hazards

- **Conflict for use of a resource**
- **In MIPS pipeline with a single memory**
  - Load/store requires data access
  - Instruction fetch would have to *stall* for that cycle
    - » Would cause a pipeline “bubble”
- **Hence, pipelined datapath require separate instruction/data memories**
  - Or separate instruction/data caches

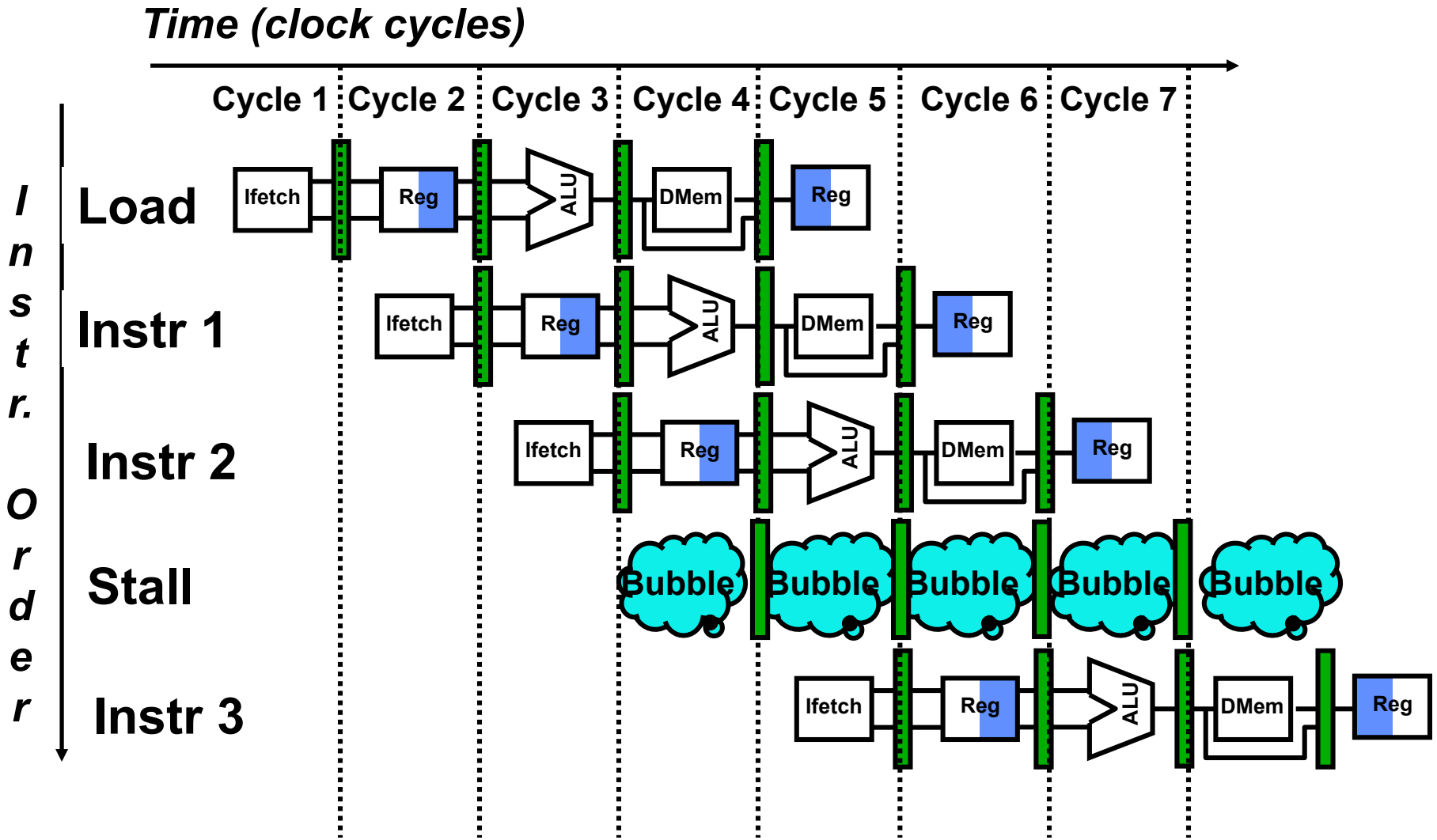
# One Memory Port/Structural Hazards



# Resolving Structural Hazards

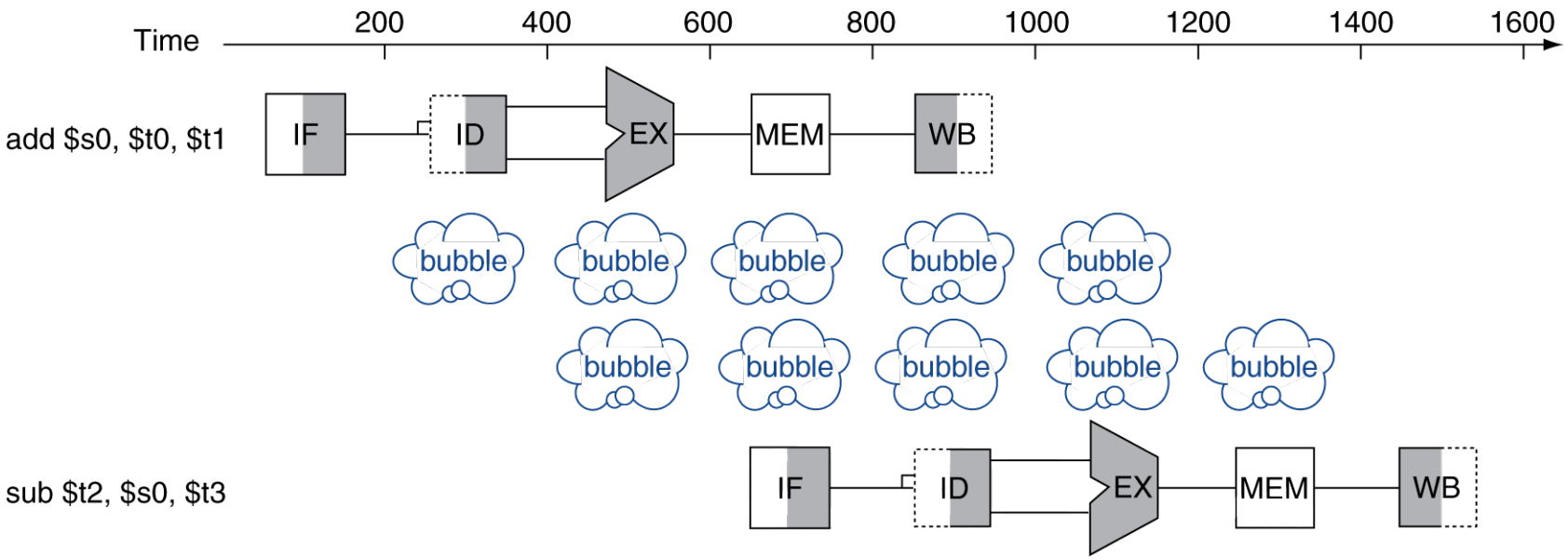
- **Structural hazard happens when two instructions try to use the same hardware for two different instructions**
- **What are the potential solutions?**
- **Solution 1: Wait for the resource to become available**
  - Must be able to detect the hazard
  - Mechanism to stall the pipeline
- **Solution 2: Add more resources to the pipeline**
  - Use separate data and instruction memories
  - Instruction cache
  - Data cache

# Resolving Structural Hazards: Stall



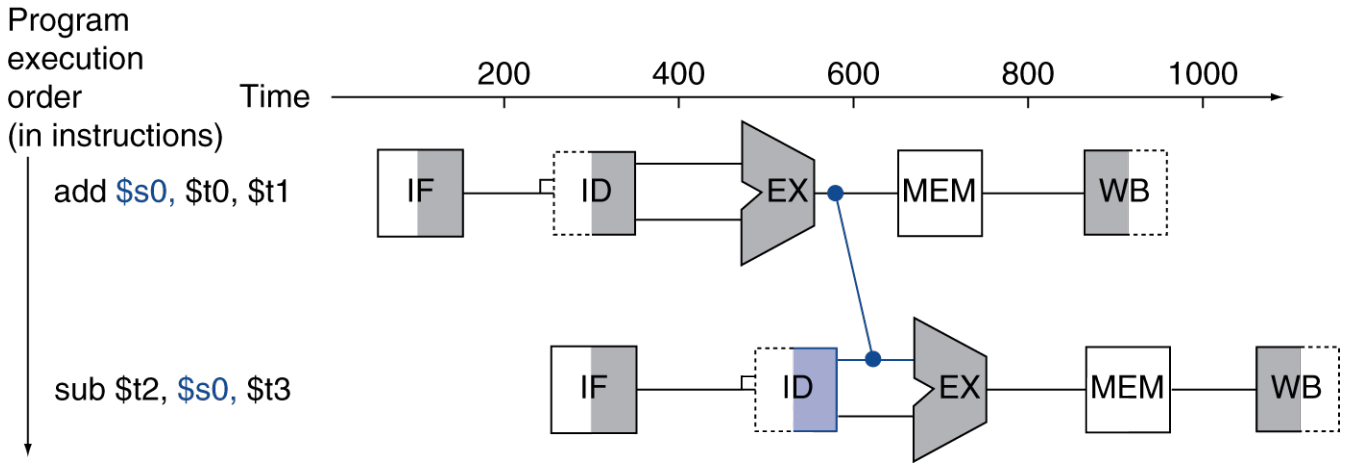
# Data Hazards

- **An instruction depends on completion of data access by a previous instruction**
  - `add $s0, $t0, $t1`
    `sub $t2, $s0, $t3`



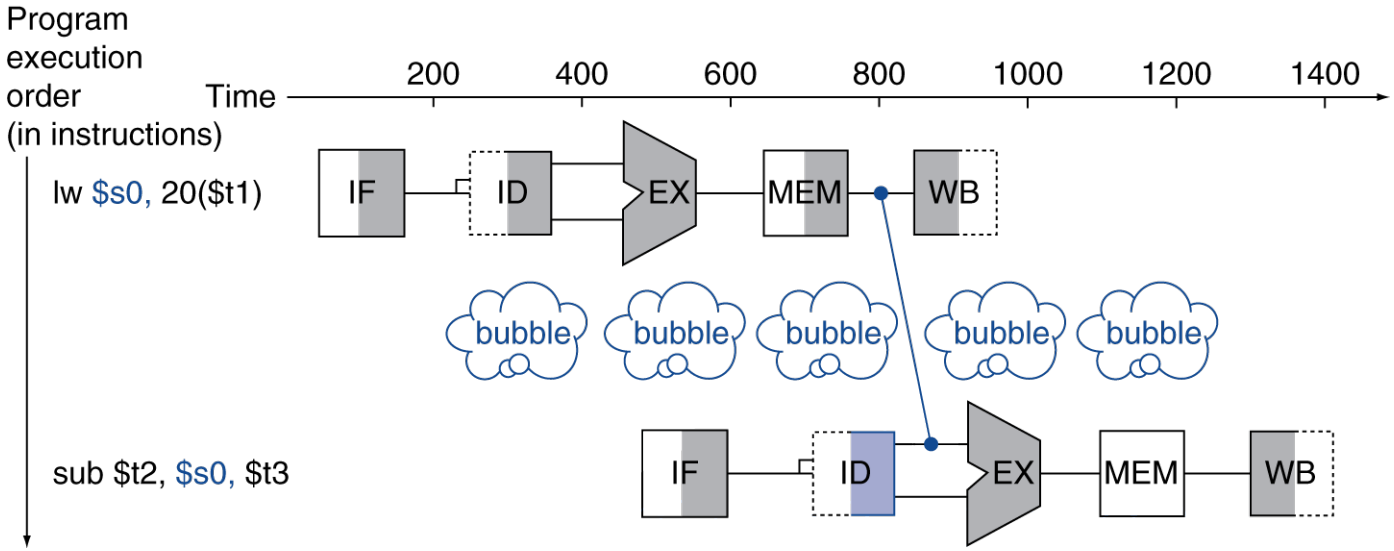
# Forwarding

- **Use result when it is computed**
  - Don't wait for it to be stored in a register
  - Requires extra connections in the datapath



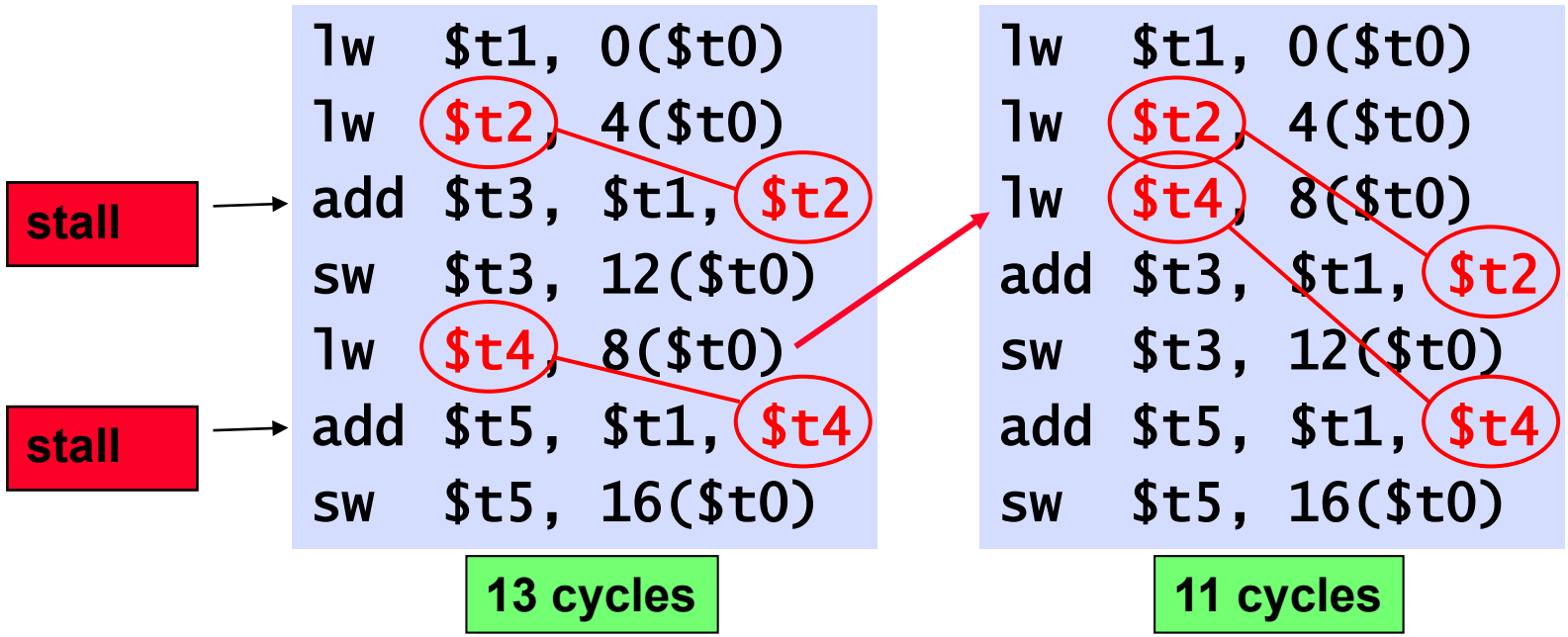
# Load-Use Data Hazard

- **Can’t always avoid stalls by forwarding**
  - If value not computed when needed
  - Can’t forward backward in time!



# Code Scheduling to Avoid Stalls

- Reorder code to avoid use of load result in the next instruction
- C code for `A = B + E; C = B + F;`

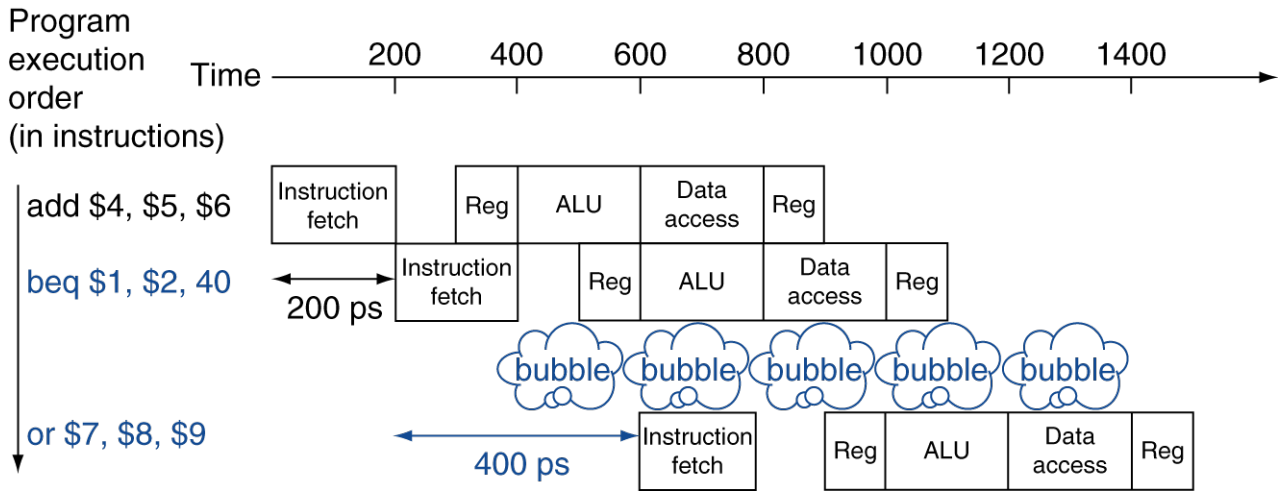


# Control Hazards

- **Branch determines flow of control**
  - Fetching next instruction depends on branch outcome
  - Pipeline can't always fetch correct instruction
    - » Still working on ID stage of branch
- **In MIPS pipeline**
  - Need to compare registers and compute target early in the pipeline
  - Add hardware to do it in ID stage

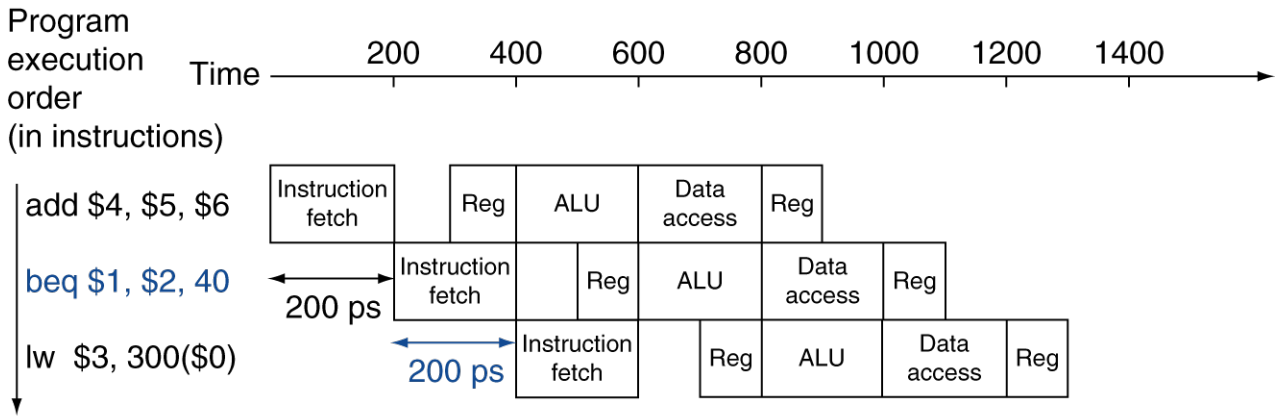
# Stall on Branch

- **Wait until branch outcome determined before fetching next instruction**

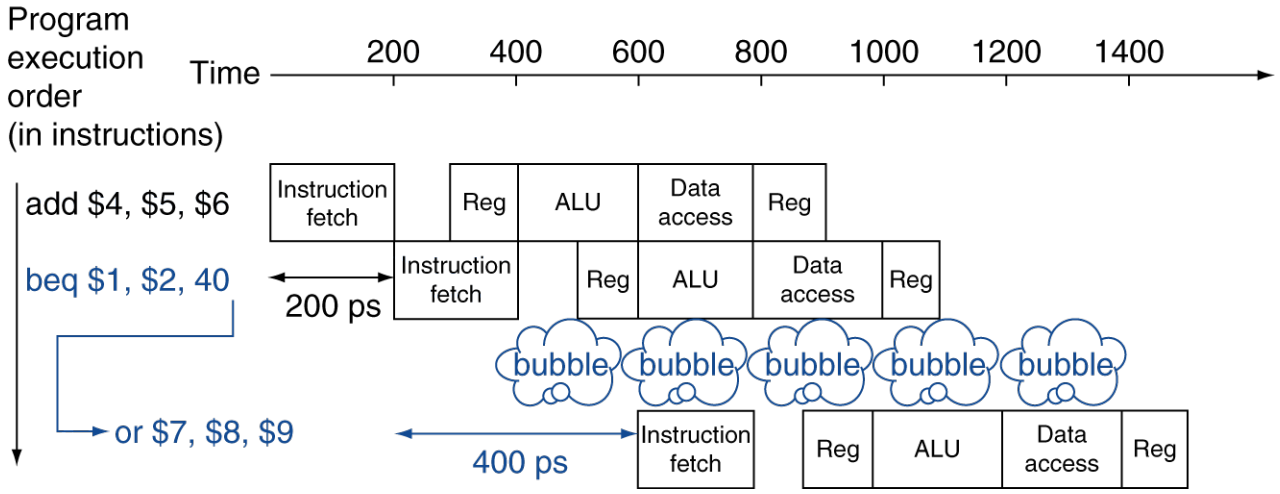


# MIPS with Predict Not Taken

**Prediction correct**




**Prediction incorrect**



# More-Realistic Branch Prediction

- **Static branch prediction**
  - Based on typical branch behavior
  - Example: loop and if-statement branches
    - » Predict backward branches taken
    - » Predict forward branches not taken
- **Dynamic branch prediction**
  - Hardware measures actual branch behavior
    - » e.g., record recent history of each branch
  - Assume future behavior will continue the trend
    - » When wrong, stall while re-fetching, and update history

# Branch Prediction

- **Longer pipelines can't readily determine branch outcome early**
  - Stall penalty becomes unacceptable
- **Predict outcome of branch**
  - Only stall if prediction is wrong
- **In MIPS pipeline**
  - Can predict branches not taken
  - Fetch instruction after branch, *delayed branch*