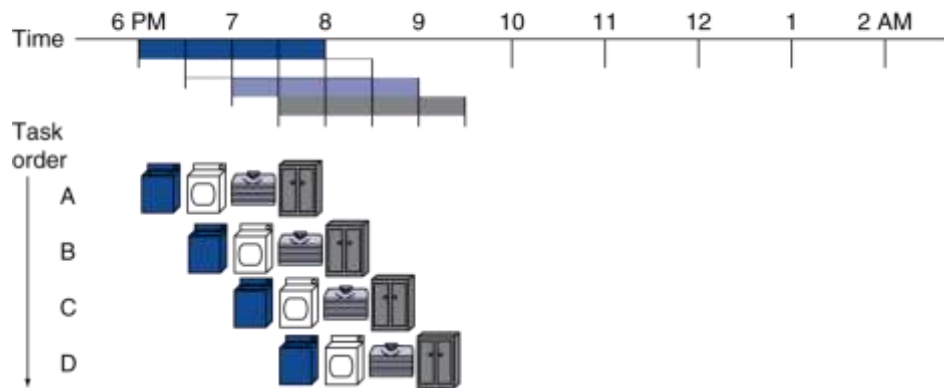
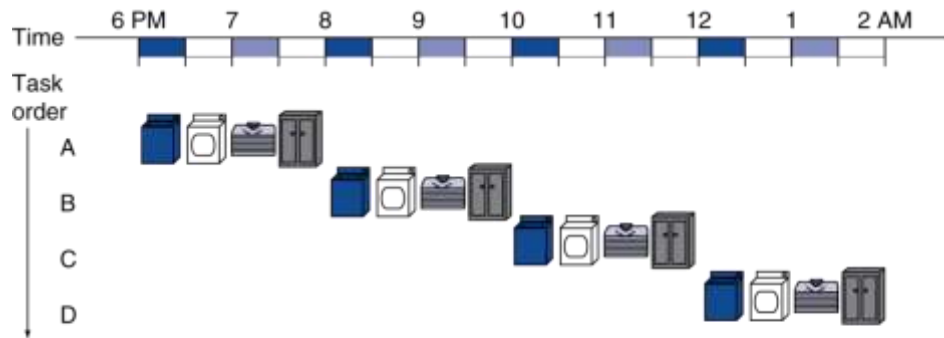


# **Processor Pipelining**

Adapted and Supplemented by,  
Dr. R. Shathanaa

# Pipelining Analogy

- Pipelined laundry: overlapping execution
  - Parallelism improves performance



- Four loads:

- Speedup  
 $= 8 / 3.5 = 2.3$

- Non-stop:

- Speedup  
 $= 2n / 0.5n + 1.5 \approx 4$   
 $= \text{number of stages}$

# MIPS Pipeline

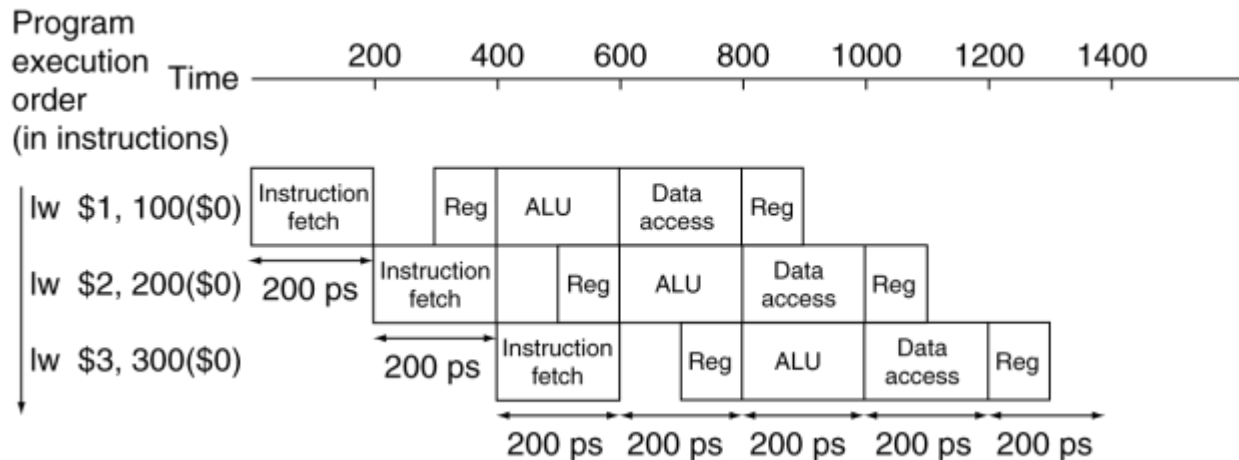
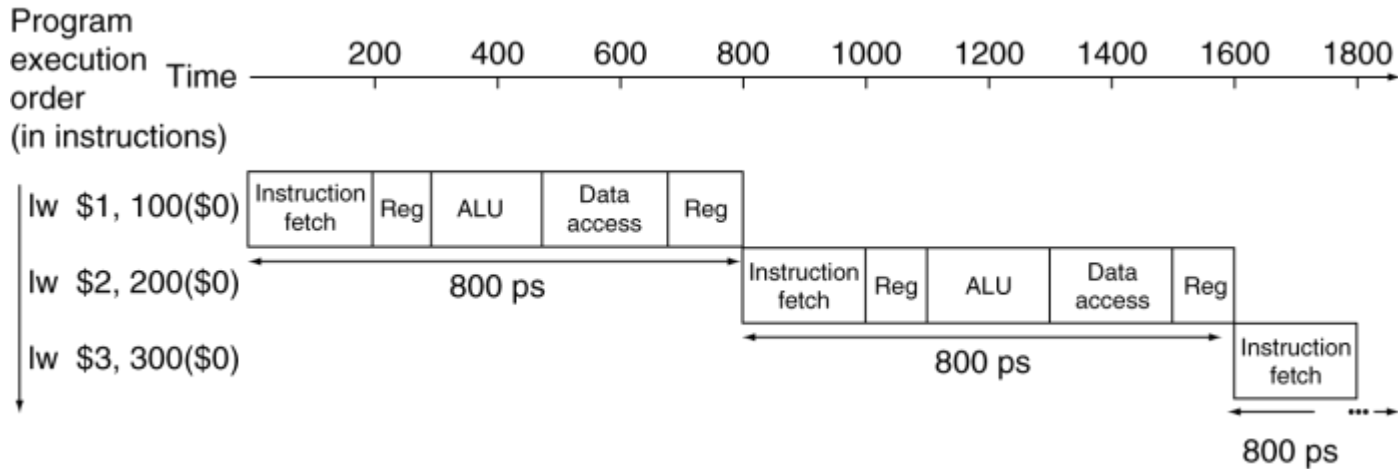
- Five stages, one step per stage
  1. IF: Instruction fetch from memory
  2. ID: Instruction decode & register read
  3. EX: Execute operation or calculate address
  4. MEM: Access memory operand
  5. WB: Write result back to register

# Pipeline Performance

- Assume time for stages is
  - 100ps for register read or write
  - 200ps for other stages
- Compare pipelined datapath with single-cycle datapath

Instr	Instr fetch	Register read	ALU op	Memory access	Register write	Total time
lw	200ps	100 ps	200ps	200ps	100 ps	800ps
sw	200ps	100 ps	200ps	200ps		700ps
R-format	200ps	100 ps	200ps		100 ps	600ps
beq	200ps	100 ps	200ps			500ps

# Pipeline Performance



# Pipeline Speedup

- If all stages are balanced
  - i.e., all take the same time
  - Time between instructions<sub>pipelined</sub>  
= 
$$\frac{\text{Time between instructions}_{\text{nonpipelined}}}{\text{Number of stages}}$$
- If not balanced, speedup is less
- Speedup due to increased throughput
  - Latency (time for each instruction) does not decrease

# Hazards

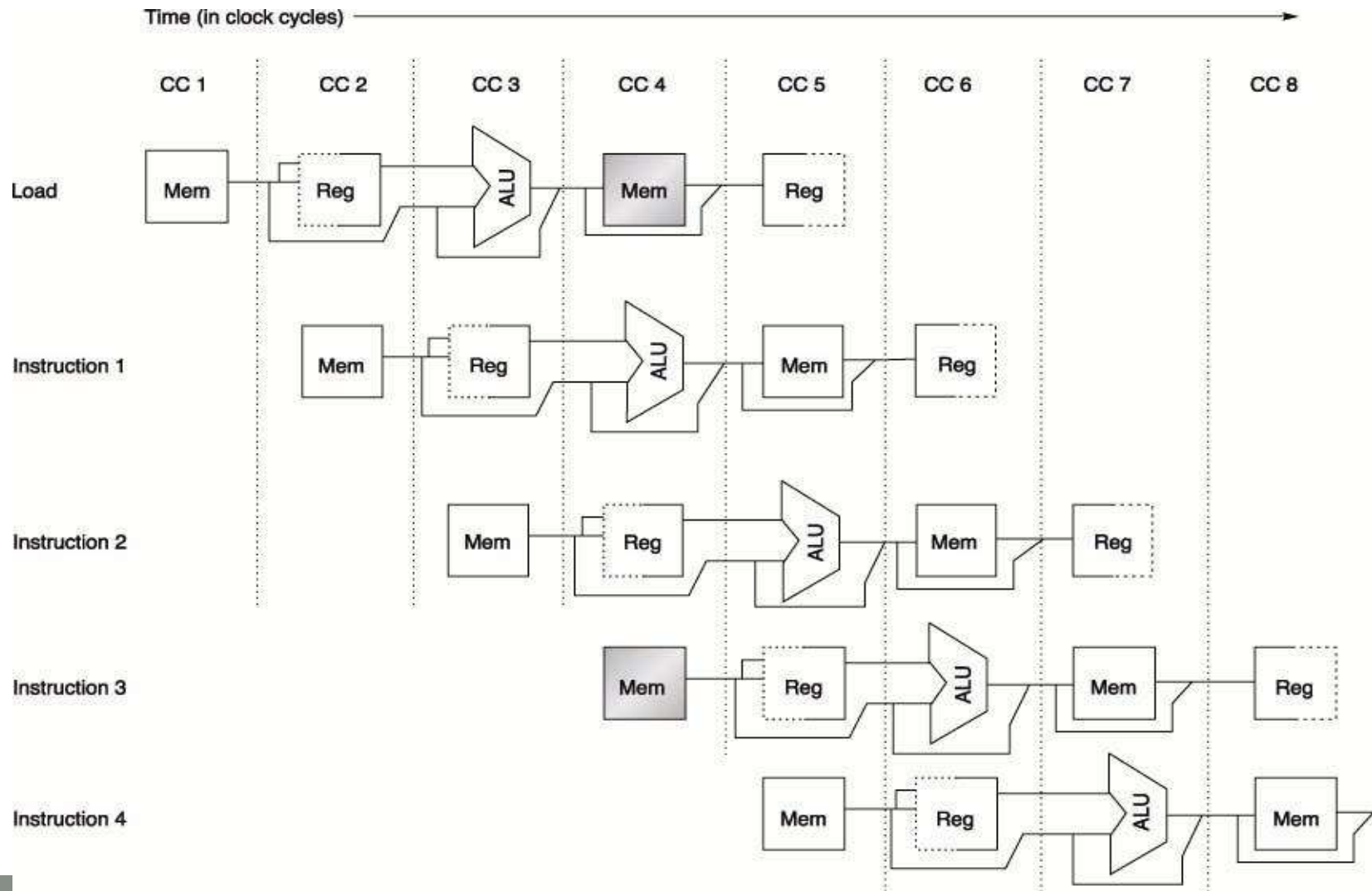
- Situations that prevent starting the next instruction in the next cycle
- Structural hazard
  - A required resource is busy
- Data hazard
  - Need to wait for previous instruction to complete its data read/write
- Control hazard
  - Deciding on control action depends on previous instruction

# Structural Hazard

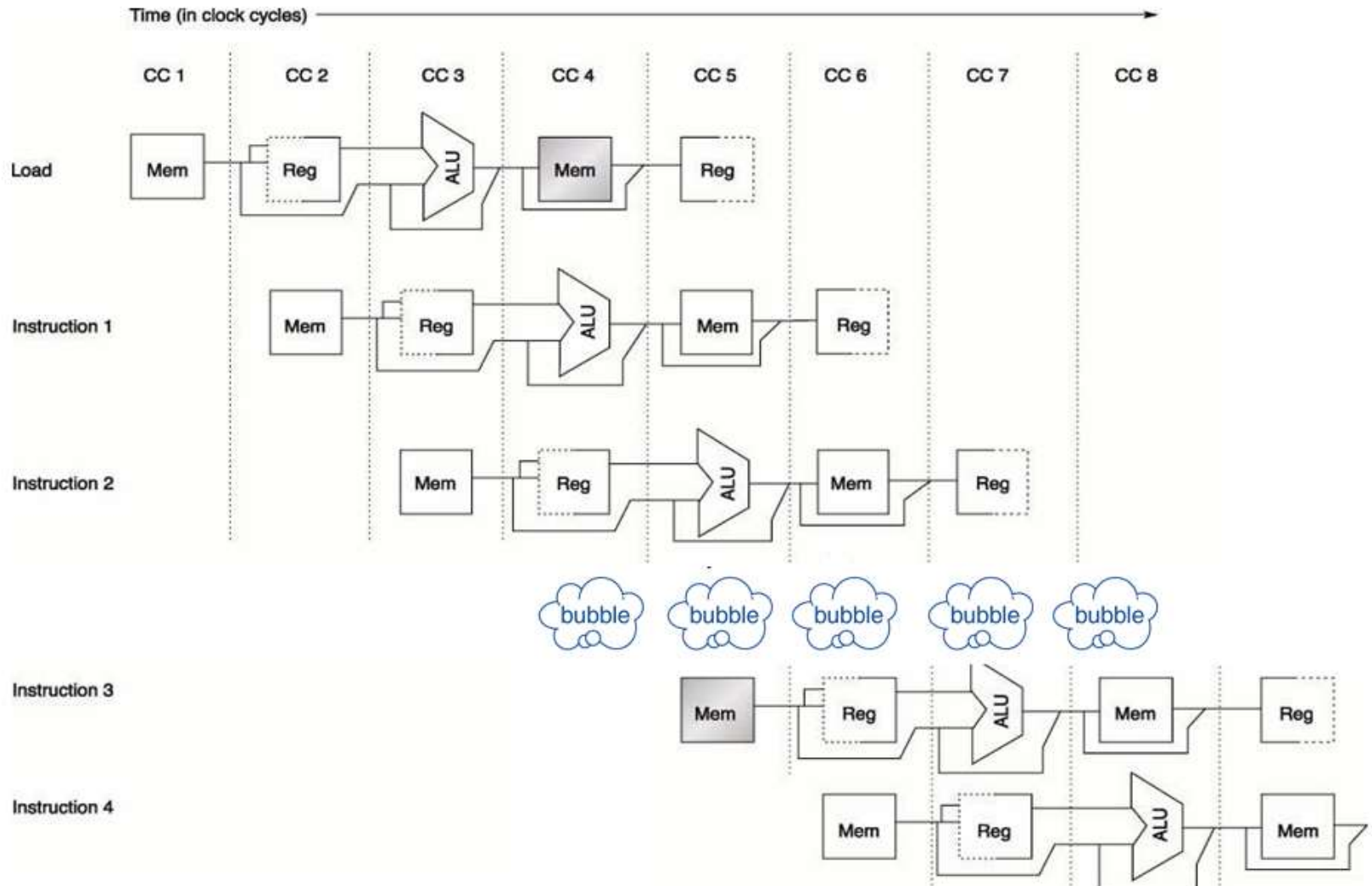
- Conflict for use of a resource
- In MIPS pipeline with a single memory
  - Load/store requires data access
  - Instruction fetch would have to *stall* for that cycle
    - Would cause a pipeline “bubble”
- Hence, pipelined datapaths require separate instruction/data memories
  - Or separate instruction/data caches
  - In general, resources need to be replicated to avoid structural hazard and stall



# Structural Hazard

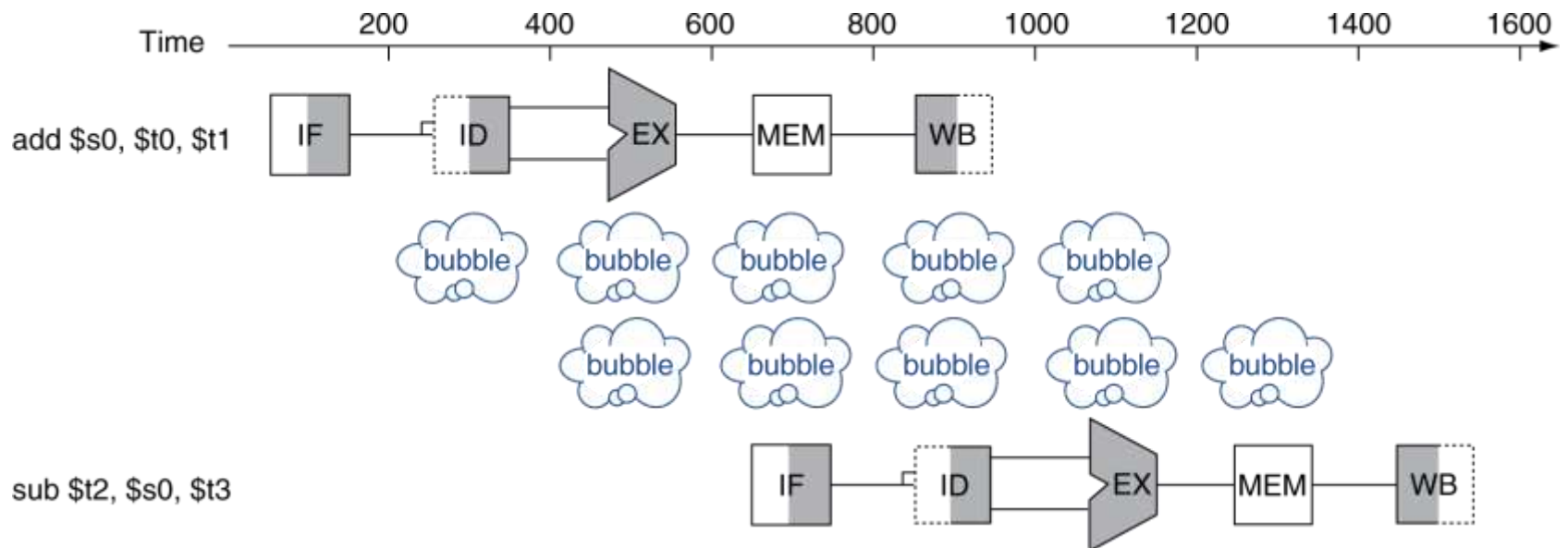


# Structural Hazard



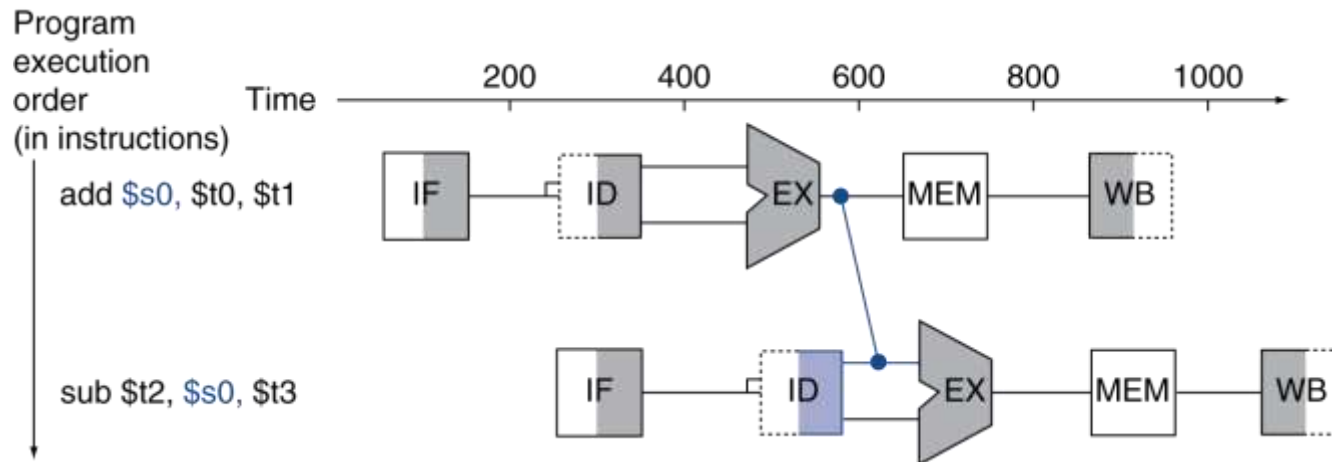
# Data Hazards

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



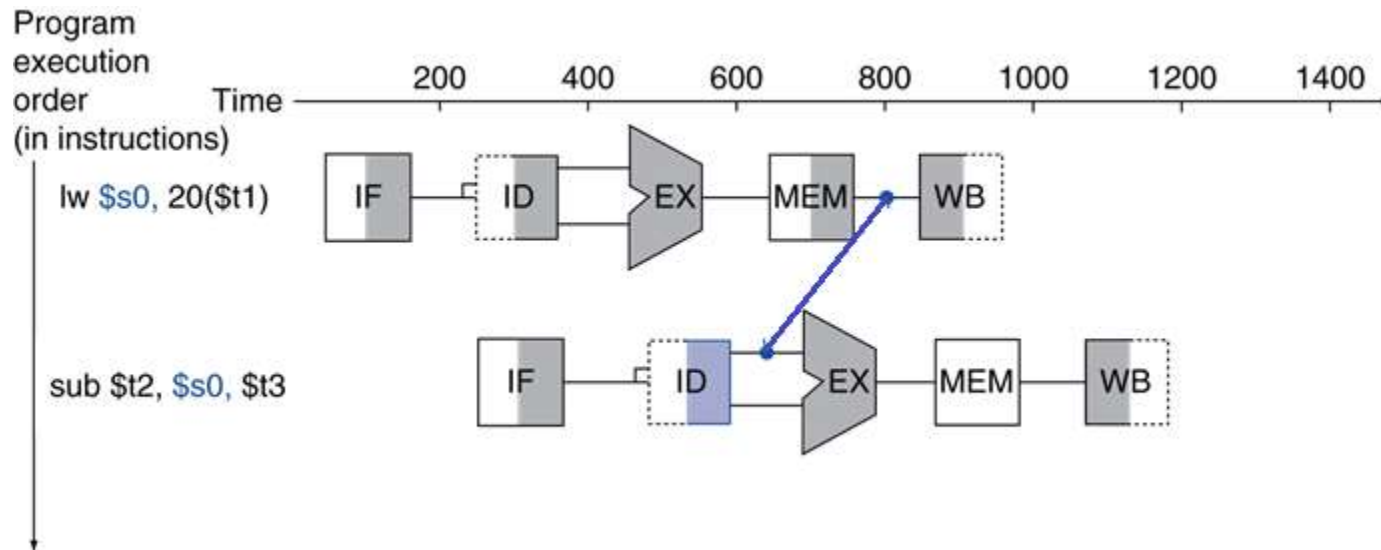
# Forwarding (aka Bypassing)

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



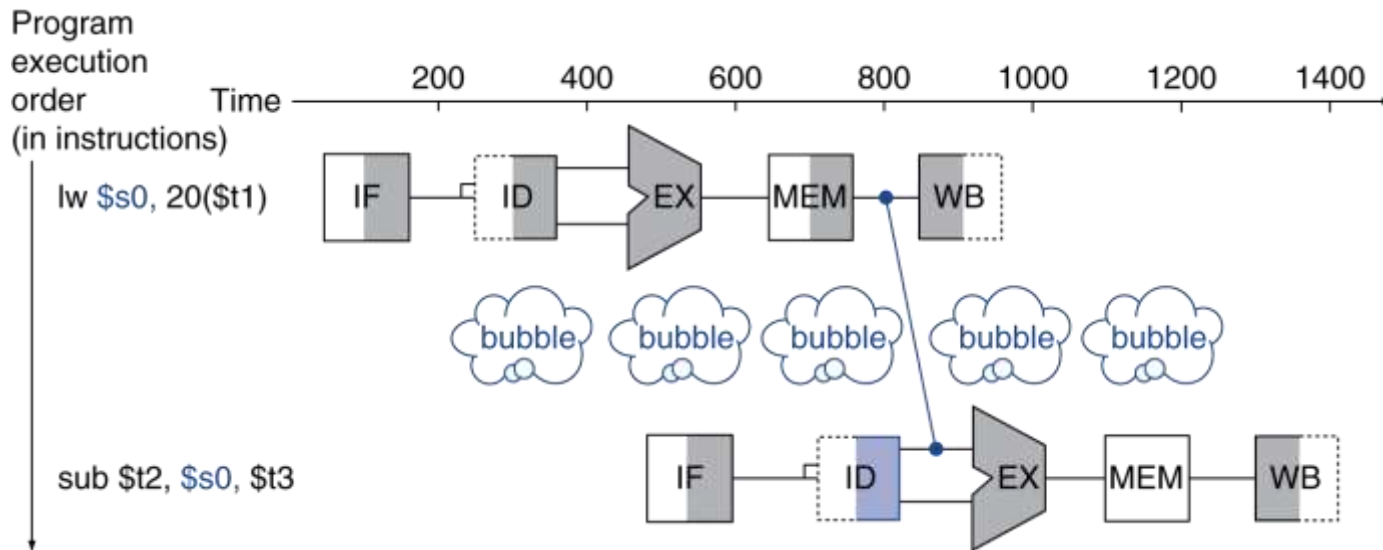
# Load-Use Data Hazard

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



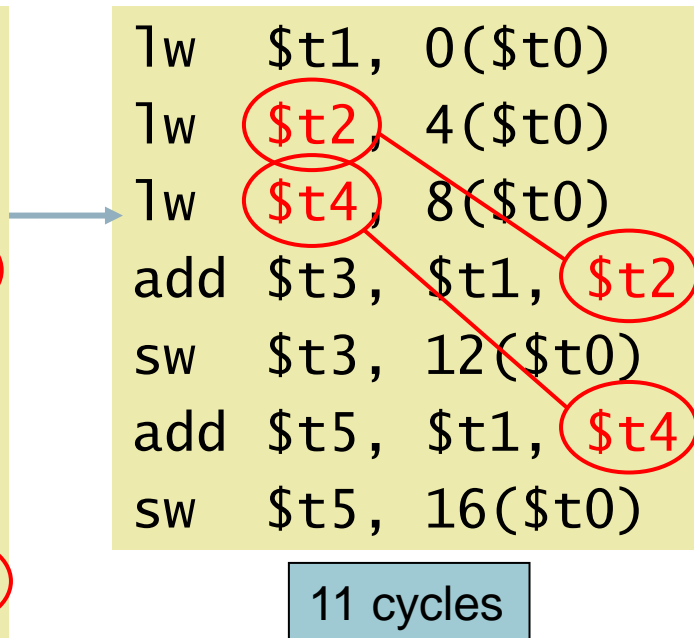
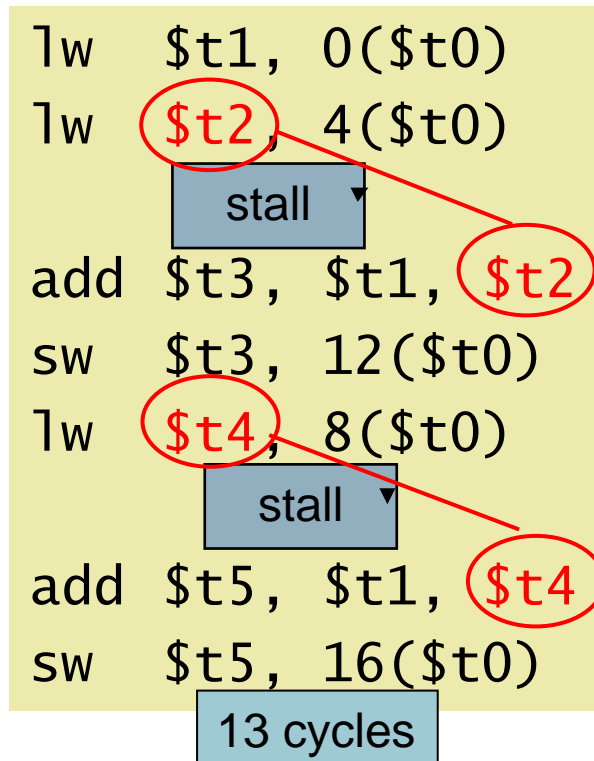
# Load-Use Data Hazard

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

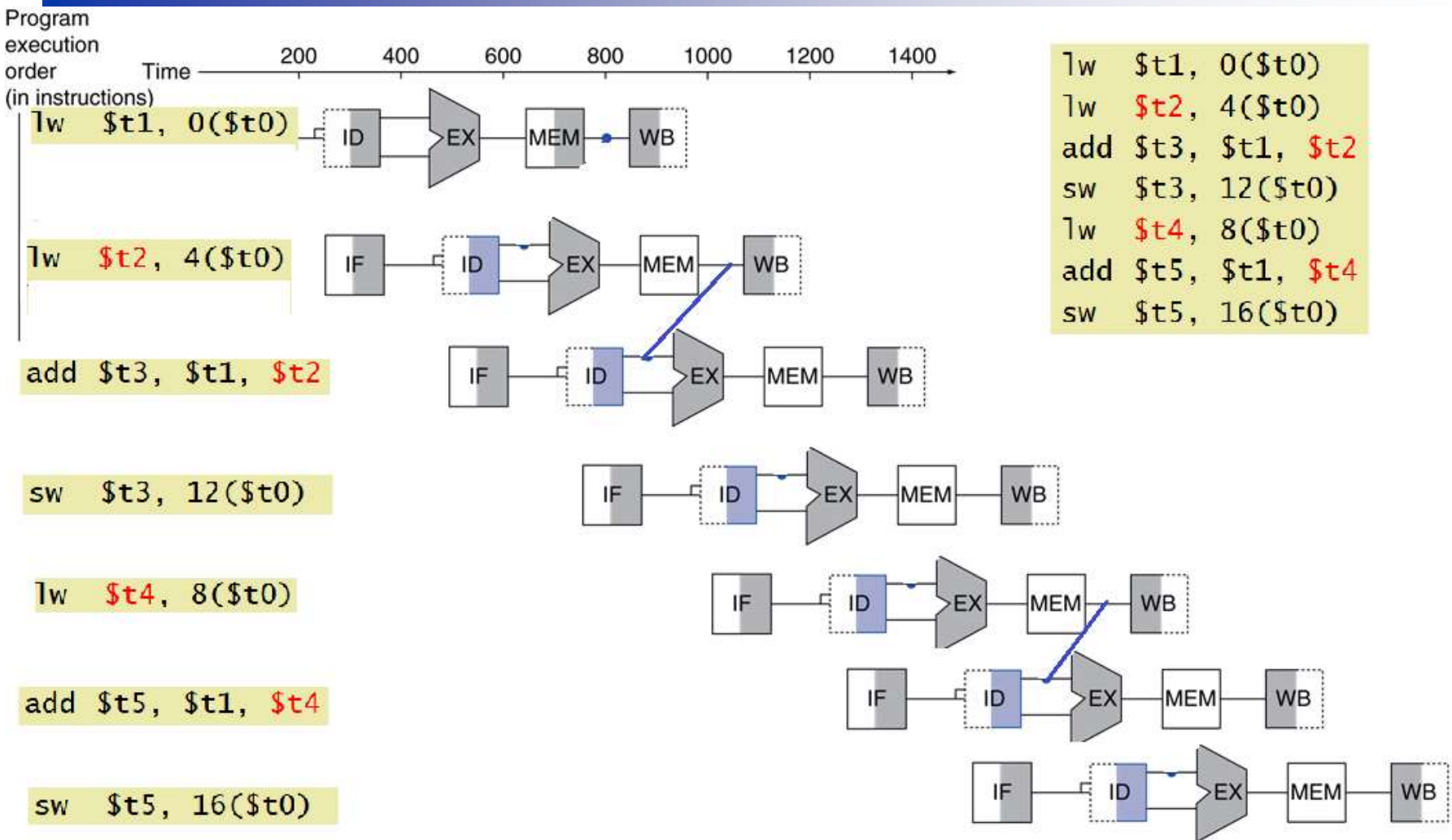


# Code Scheduling to Avoid Stalls

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

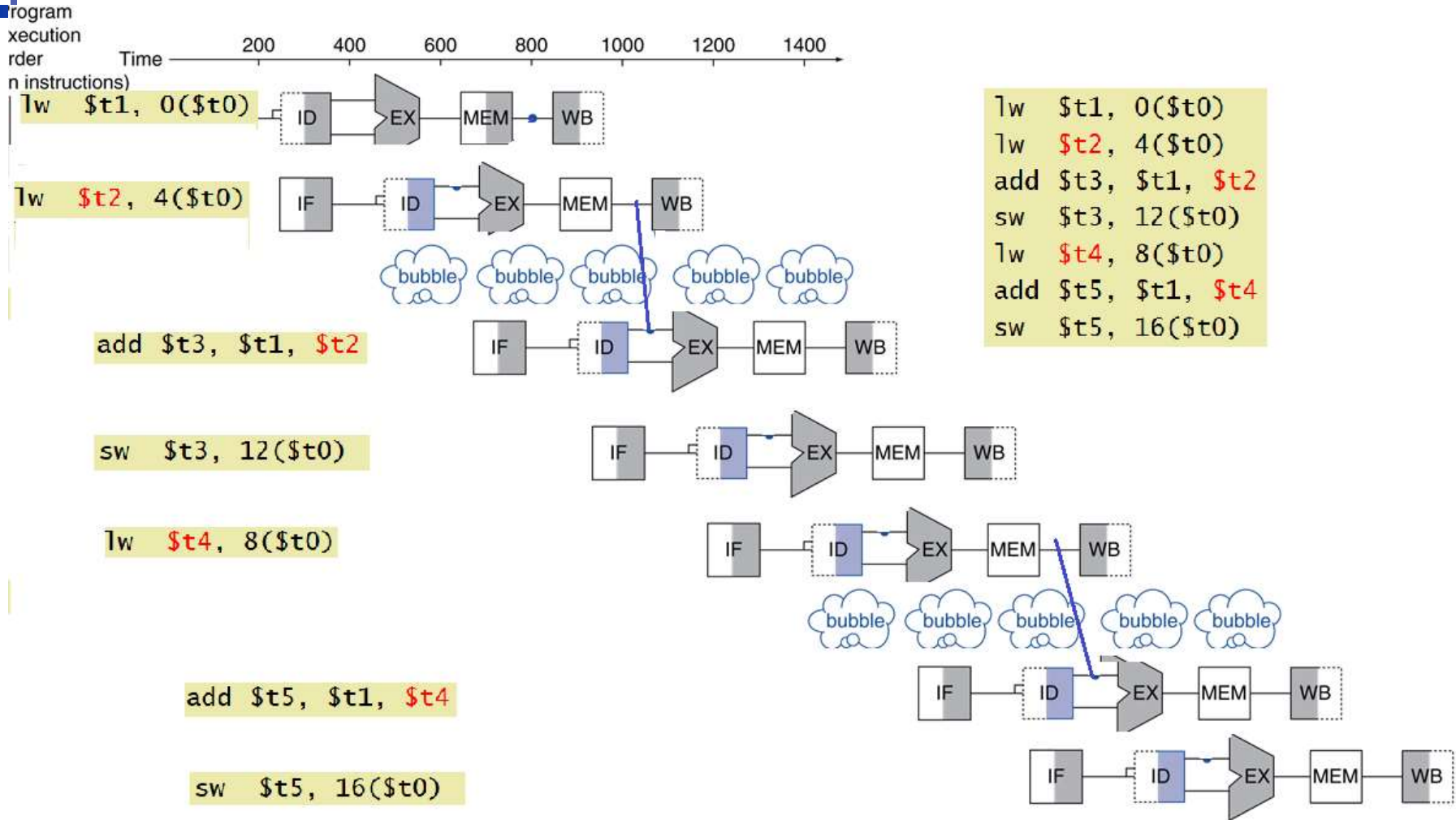


# Code Scheduling to Avoid Stalls

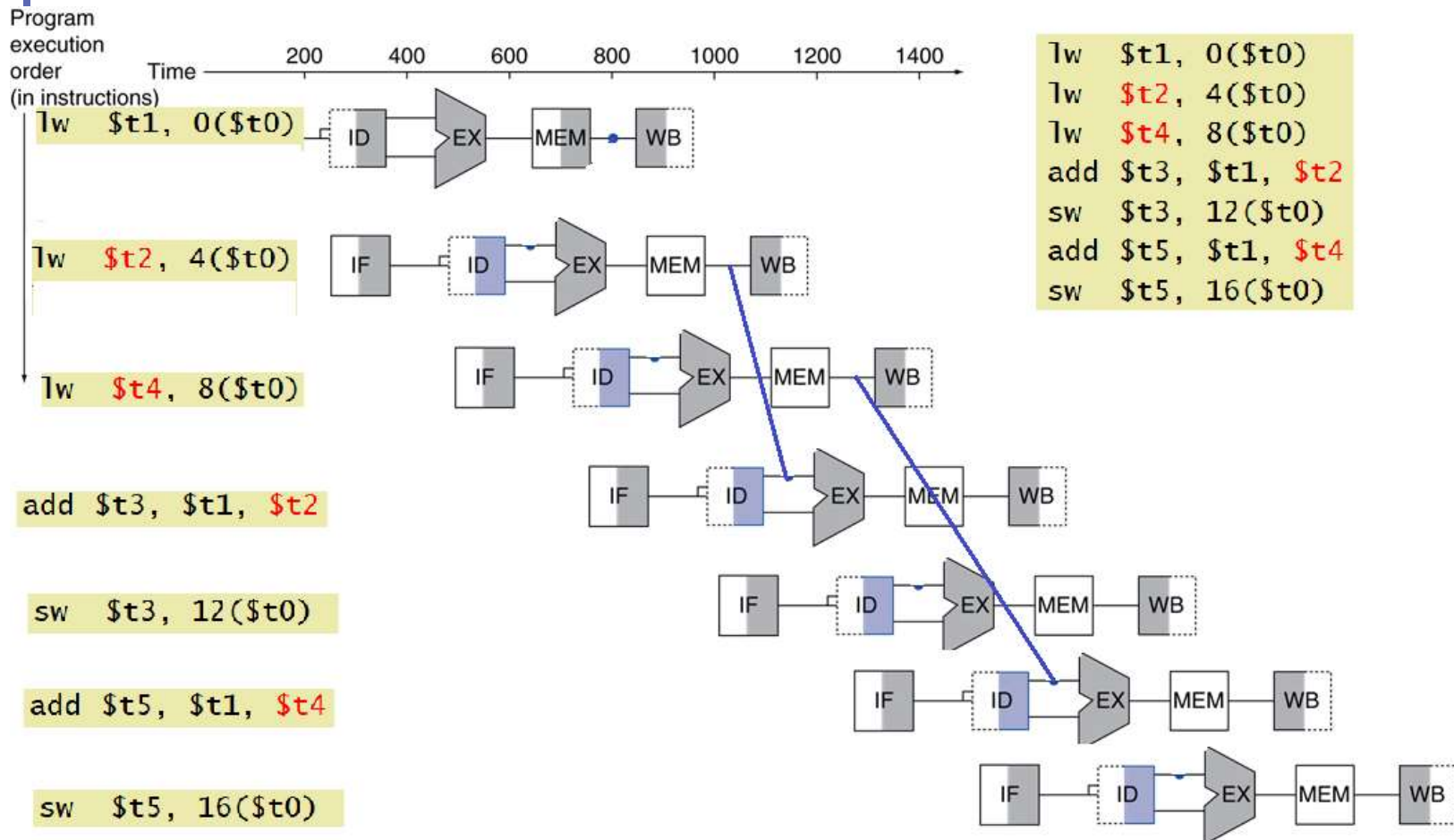




# Code Scheduling to Avoid Stalls



# Code Scheduling to Avoid Stalls



# Next Topic ...

---

- Control Hazards