

Short Pause...

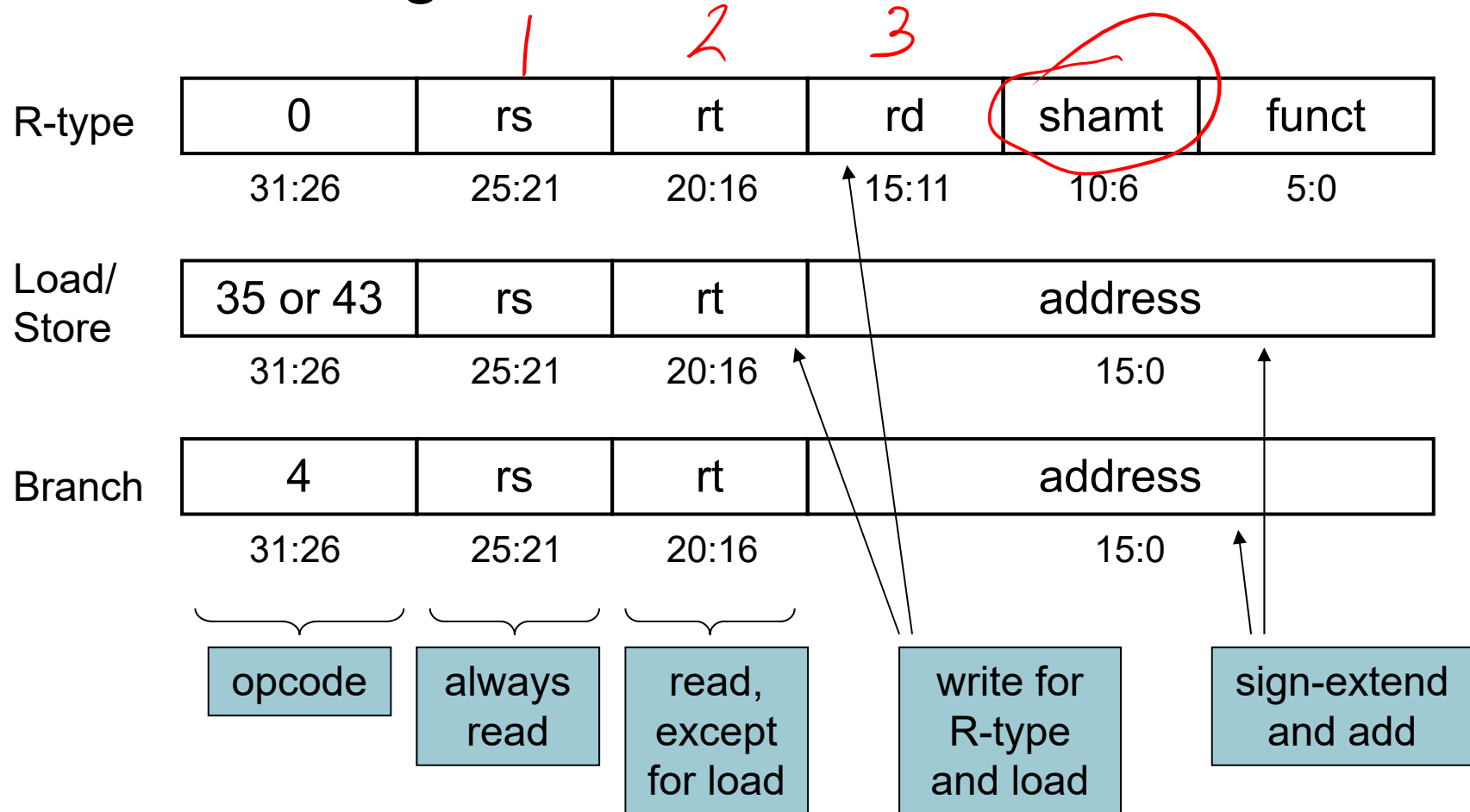
Homework / Quiz Questions

Midterm Tuesday!

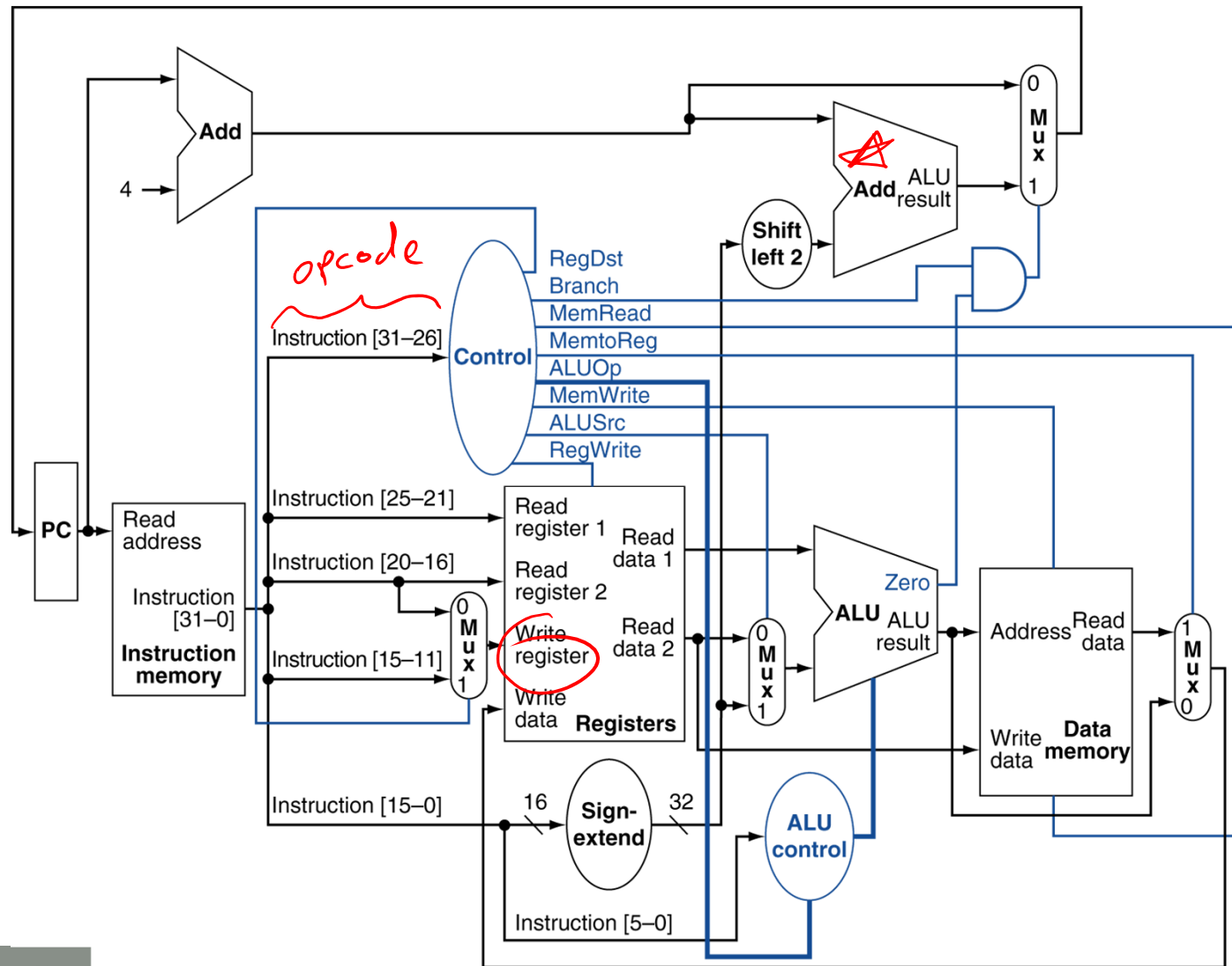
- Calculator*
- 8.5" x 11" notes (double-sided)*

The Main Control Unit

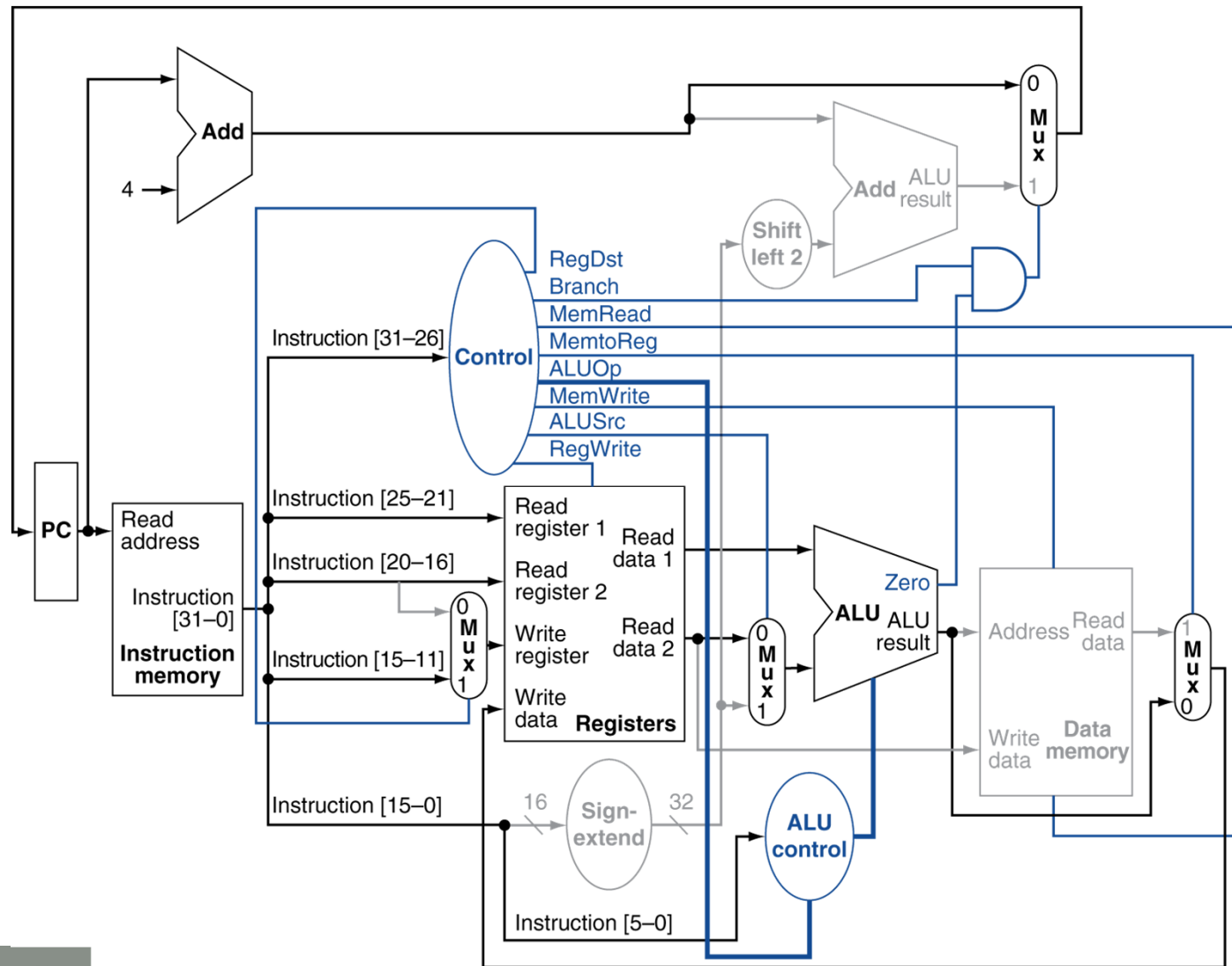
■ Control signals derived from instruction



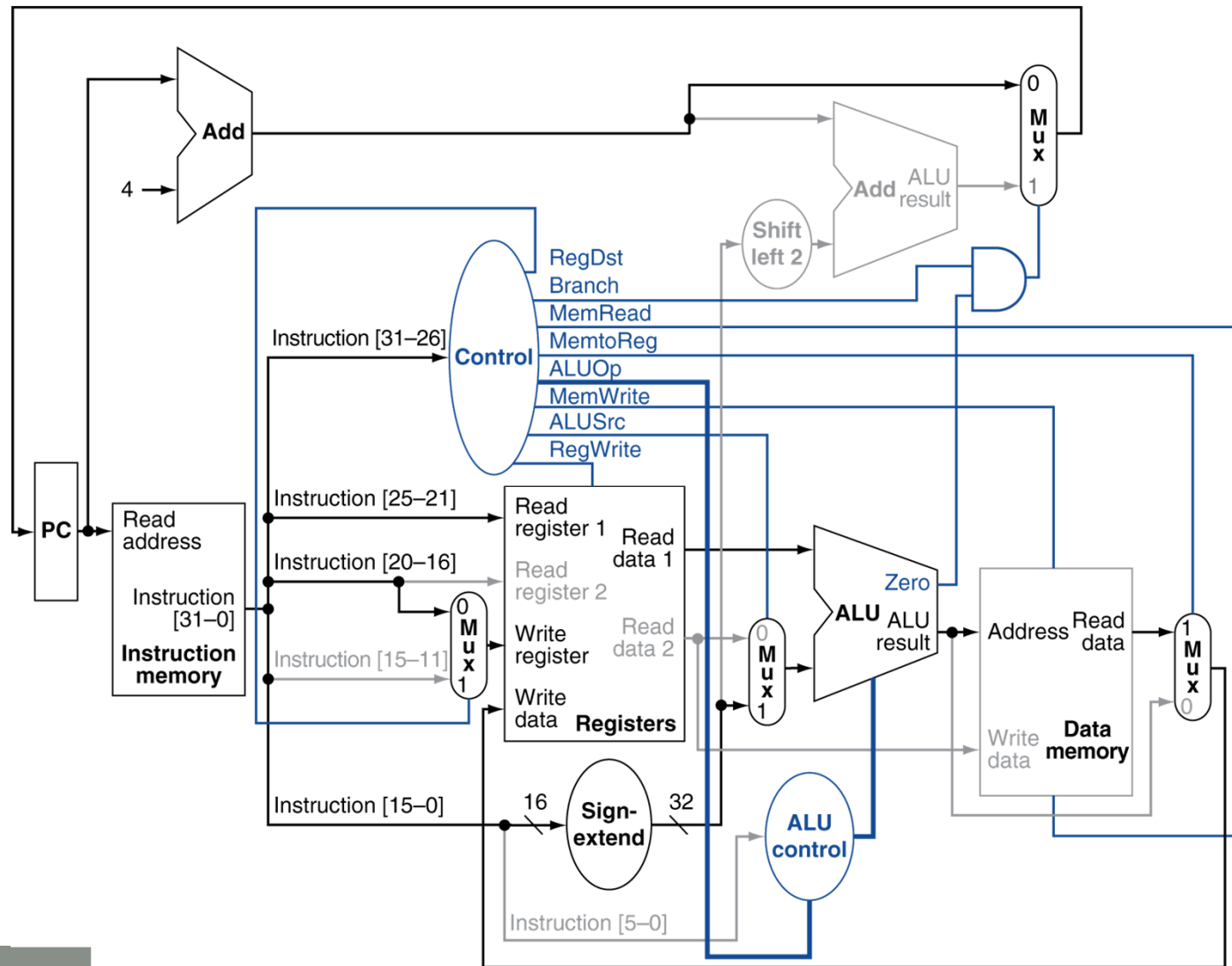
Datapath With Control



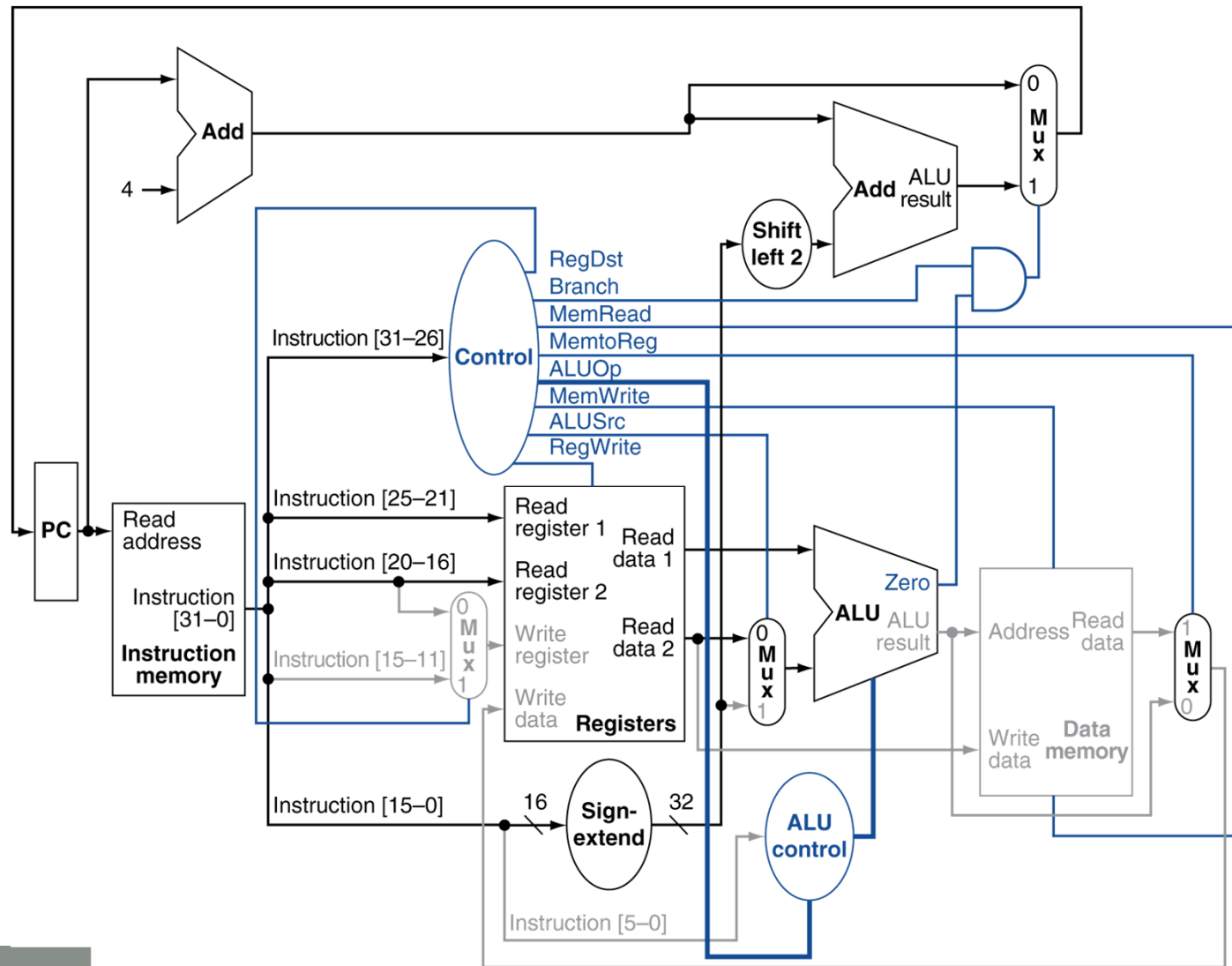
R-Type Instruction



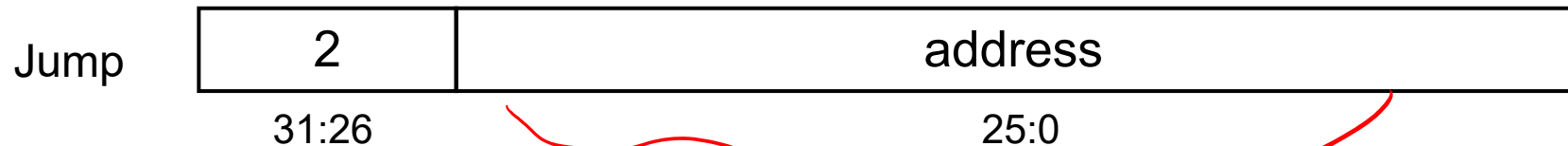
Load Instruction



Branch-on-Equal Instruction

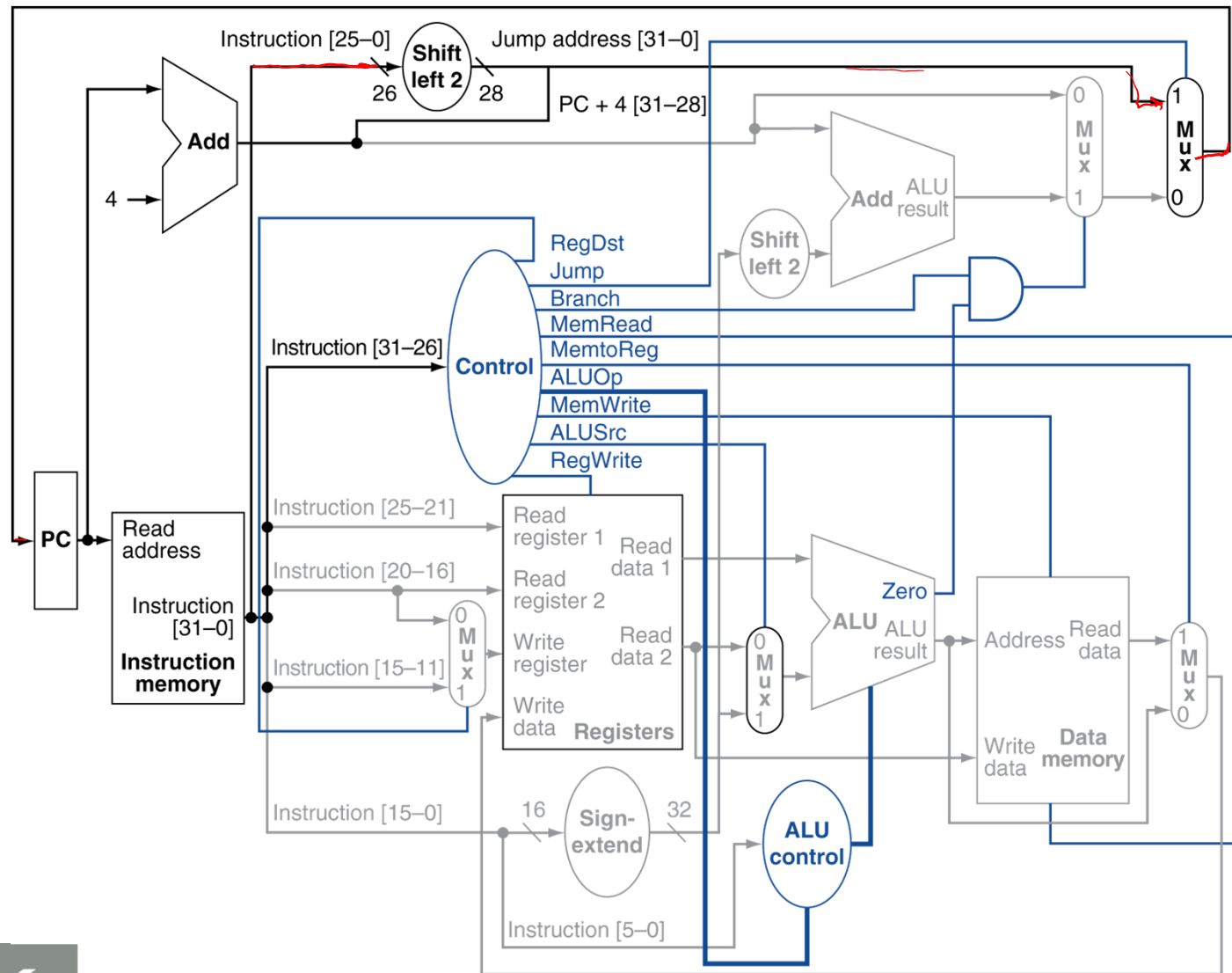


Implementing Jumps



- Jump uses word address
- Update PC with concatenation of
 - Top 4 bits of old PC
 - 26-bit jump address
 - 00
- Need an extra control signal decoded from opcode

Datapath With Jumps Added

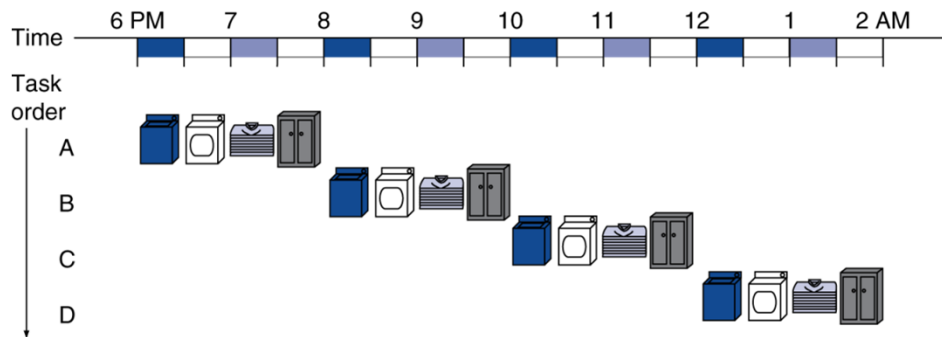


Performance Issues

- Longest delay determines clock period
 - Critical path: load instruction
 - Instruction memory → register file → ALU → data memory → register file
- Not feasible to vary period for different instructions
- Violates design principle
 - Making the common case fast
- We will improve performance by pipelining

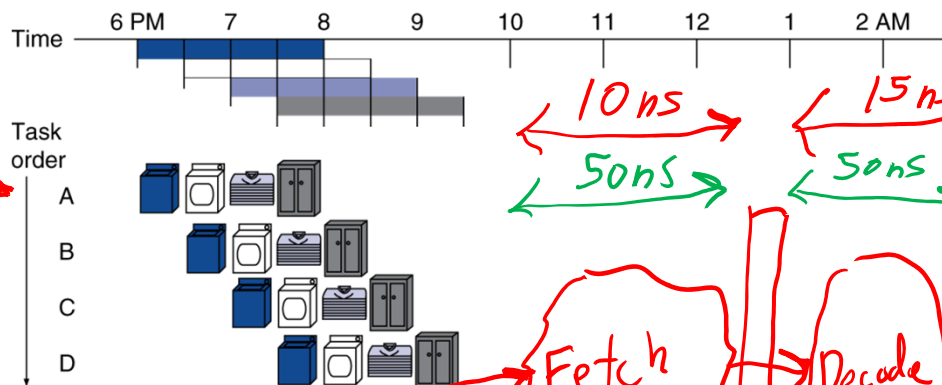
Pipelining Analogy

- Pipelined laundry: overlapping execution
 - Parallelism improves performance



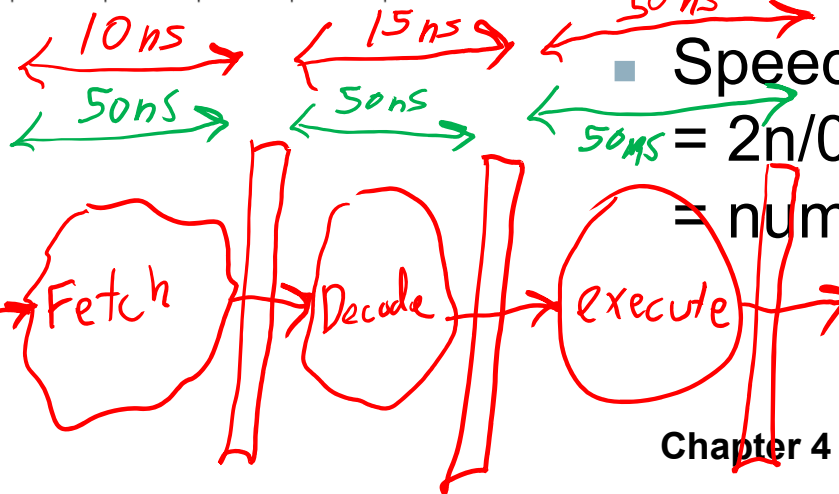
- Four loads:

- Speedup
= $8/3.5 = 2.3$



- Non-stop:

- Speedup
= $2n/0.5n + 1.5 \approx 4$
= 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

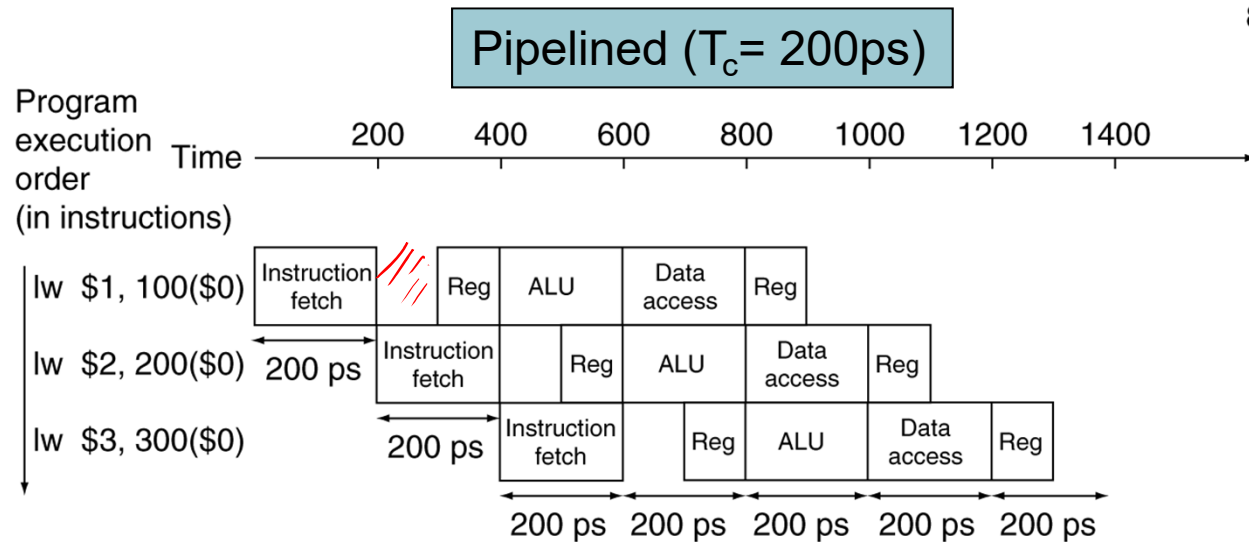
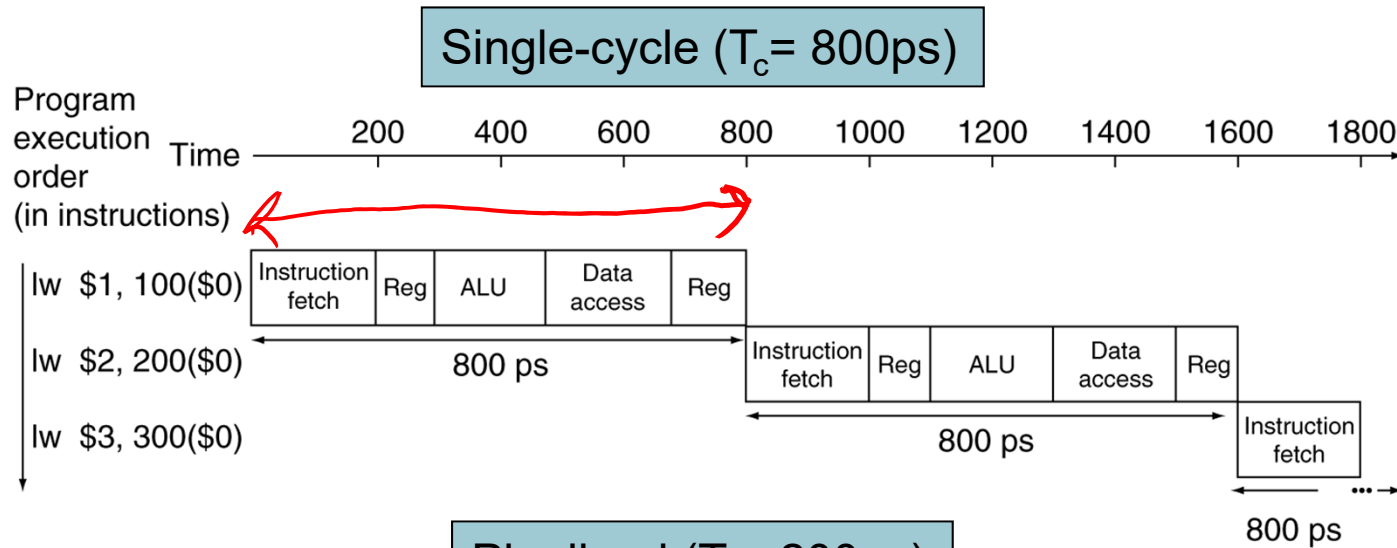
Pipeline Performance

- 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_{pipelined}
=
$$\frac{\text{Time between instructions}_{\text{nonpipelined}}}{\text{Number of stages}}$$
- If not balanced, speedup is less
- Speedup due to increased throughput
 - Latency (time for each instruction) does not decrease

Pipelining and ISA Design

- MIPS ISA designed for pipelining
 - All instructions are 32-bits
 - Easier to fetch and decode in one cycle
 - c.f. x86: 1- to 17-byte instructions
 - Few and regular instruction formats
 - Can decode and read registers in one step
 - Load/store addressing
 - Can calculate address in 3rd stage, access memory in 4th stage
 - Alignment of memory operands
 - Memory access takes only one cycle

Hazards

- Situations that prevent starting the next instruction in the next cycle

- Structure hazards

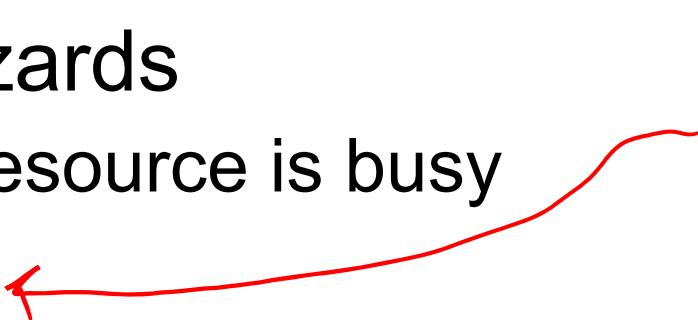
- A required resource is busy

- Data hazard

- Need to wait for previous instruction to complete its data read/write

- Control hazard

- Deciding on control action depends on previous instruction

$$\begin{cases} A = B + C \\ D = A + E \end{cases}$$


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 datapaths require separate instruction/data memories
 - Or separate instruction/data caches

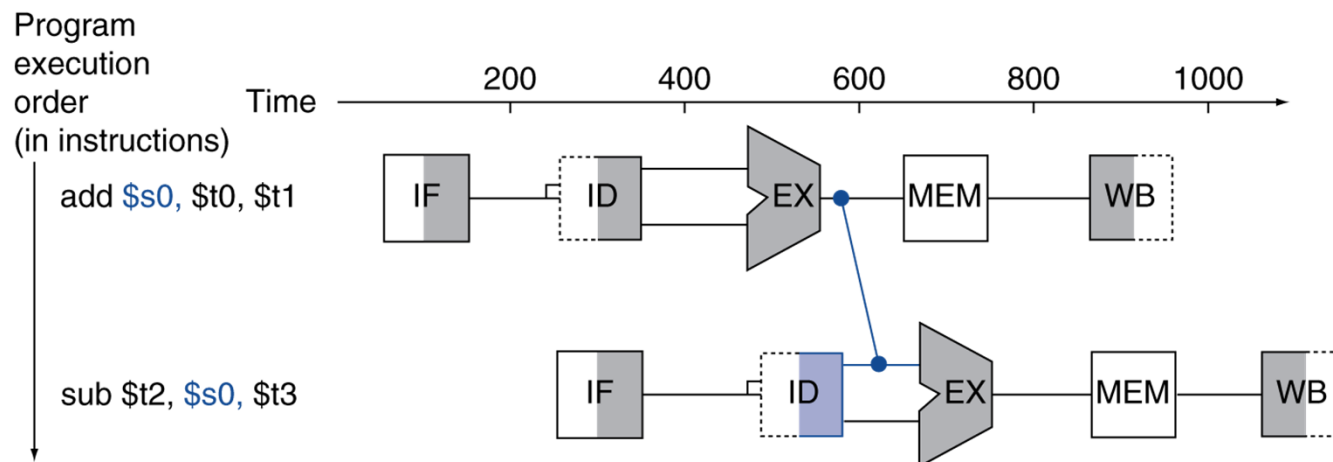
- An instruction depends on completion of data access by a previous instruction

- ## Writing registers



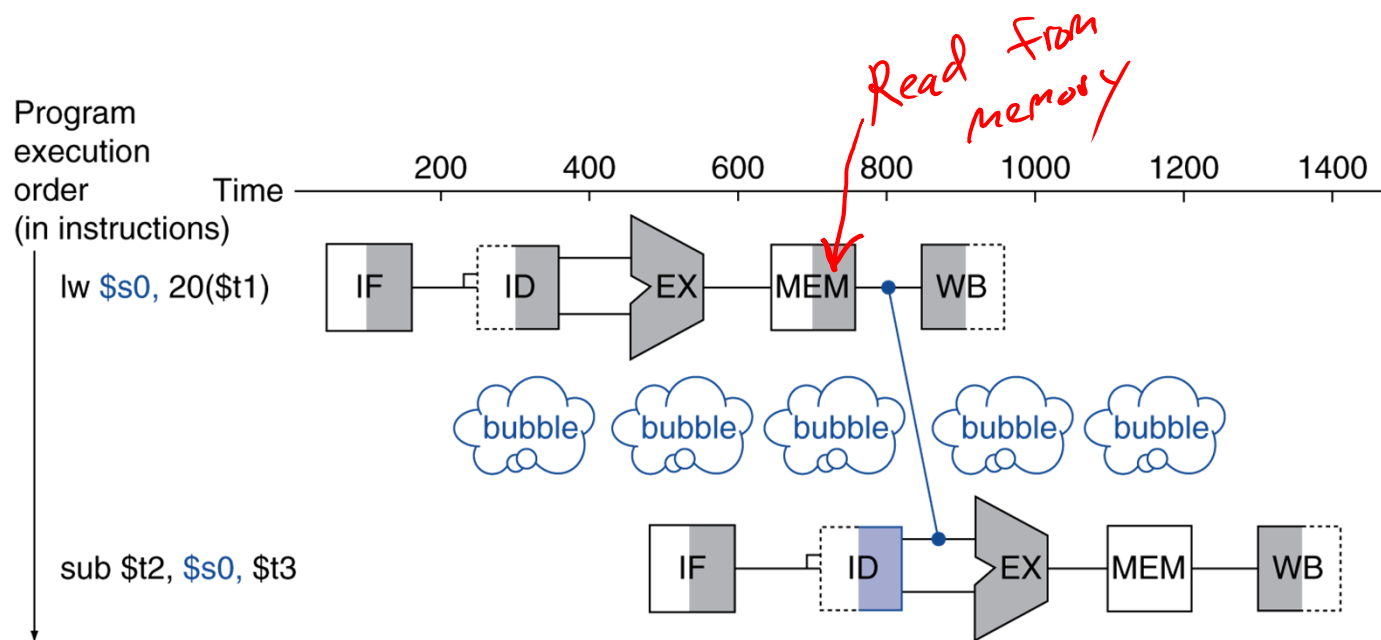
Forwarding (aka Bypassing)

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



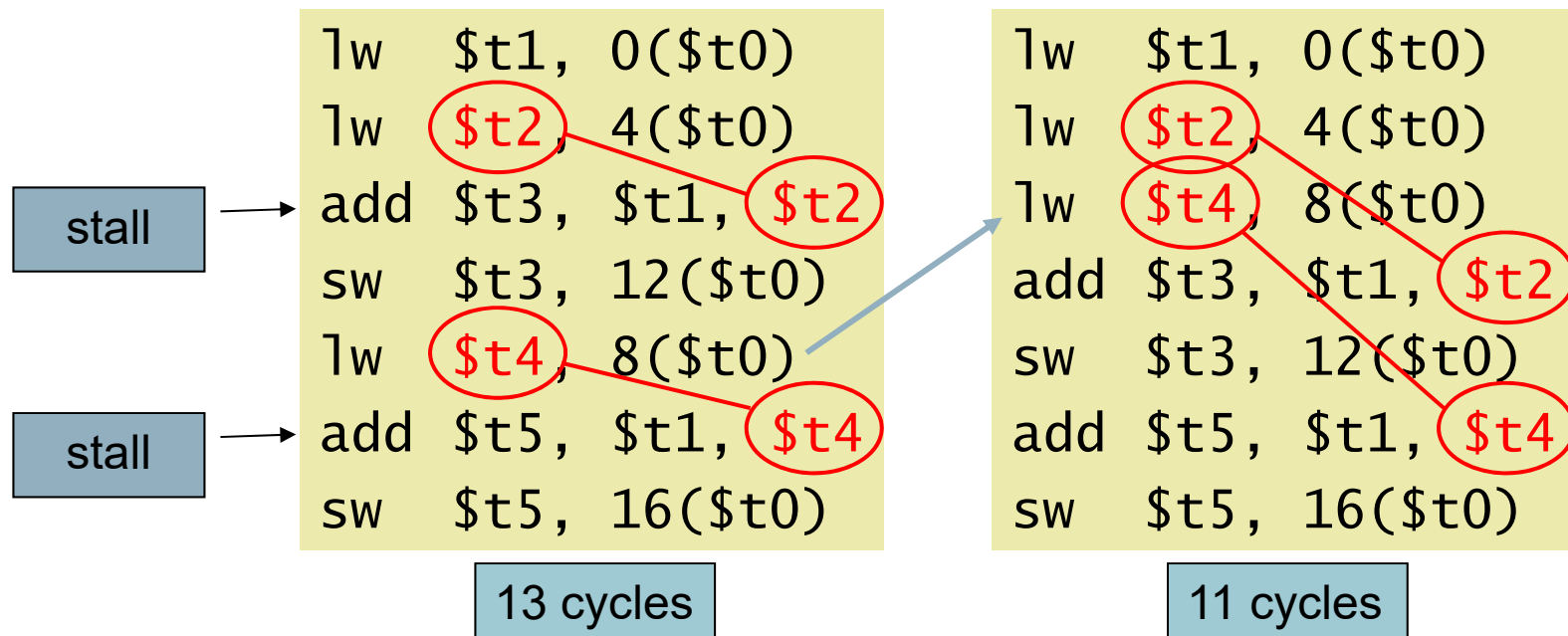
Load-Use Data Hazard

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



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

