



NATIONAL RESEARCH
UNIVERSITY



Computer Architecture and Operating Systems

Lecture 10: Processor and Pipeline

Andrei Tatarnikov

atatarnikov@hse.ru

[@andrewt0301](https://twitter.com/andrewt0301)

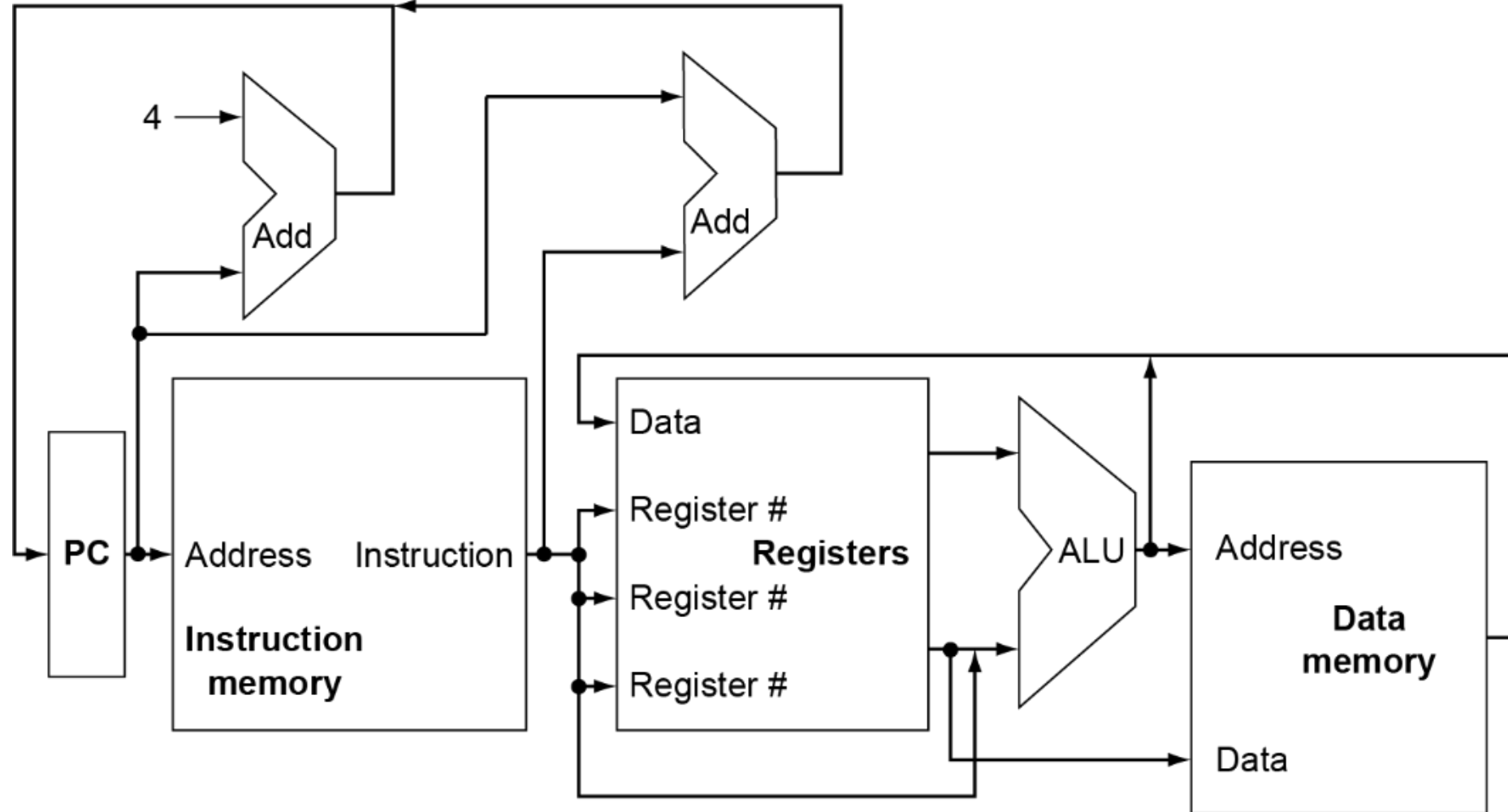
CPU Under The Hood

- CPU performance factors
 - Instruction count
 - Determined by ISA and compiler
 - CPI and Cycle time
 - Determined by CPU hardware
- We will examine two RISC-V implementations
 - A simplified version
 - A more realistic pipelined version
- Simple subset, shows most aspects
 - Memory reference: `ld`, `sd`
 - Arithmetic/logical: `add`, `sub`, `and`, `or`
 - Control transfer: `beq`

Instruction Execution

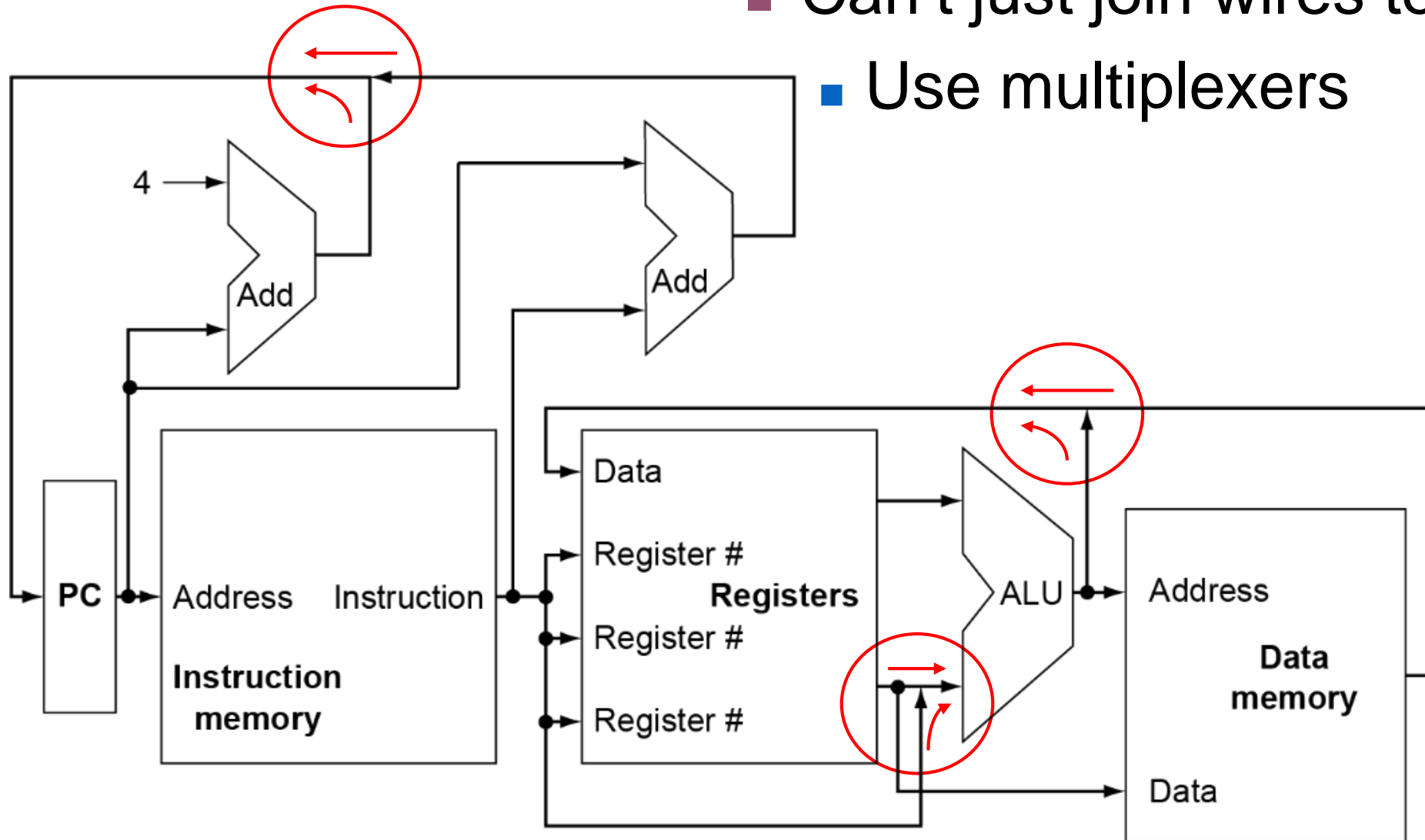
- PC → instruction memory, fetch instruction
- Register numbers → register file, read registers
- Depending on instruction class
 - Use ALU to calculate
 - Arithmetic result
 - Memory address for load/store
 - Branch comparison
 - Access data memory for load/store
 - PC ← target address or PC + 4

CPU Overview

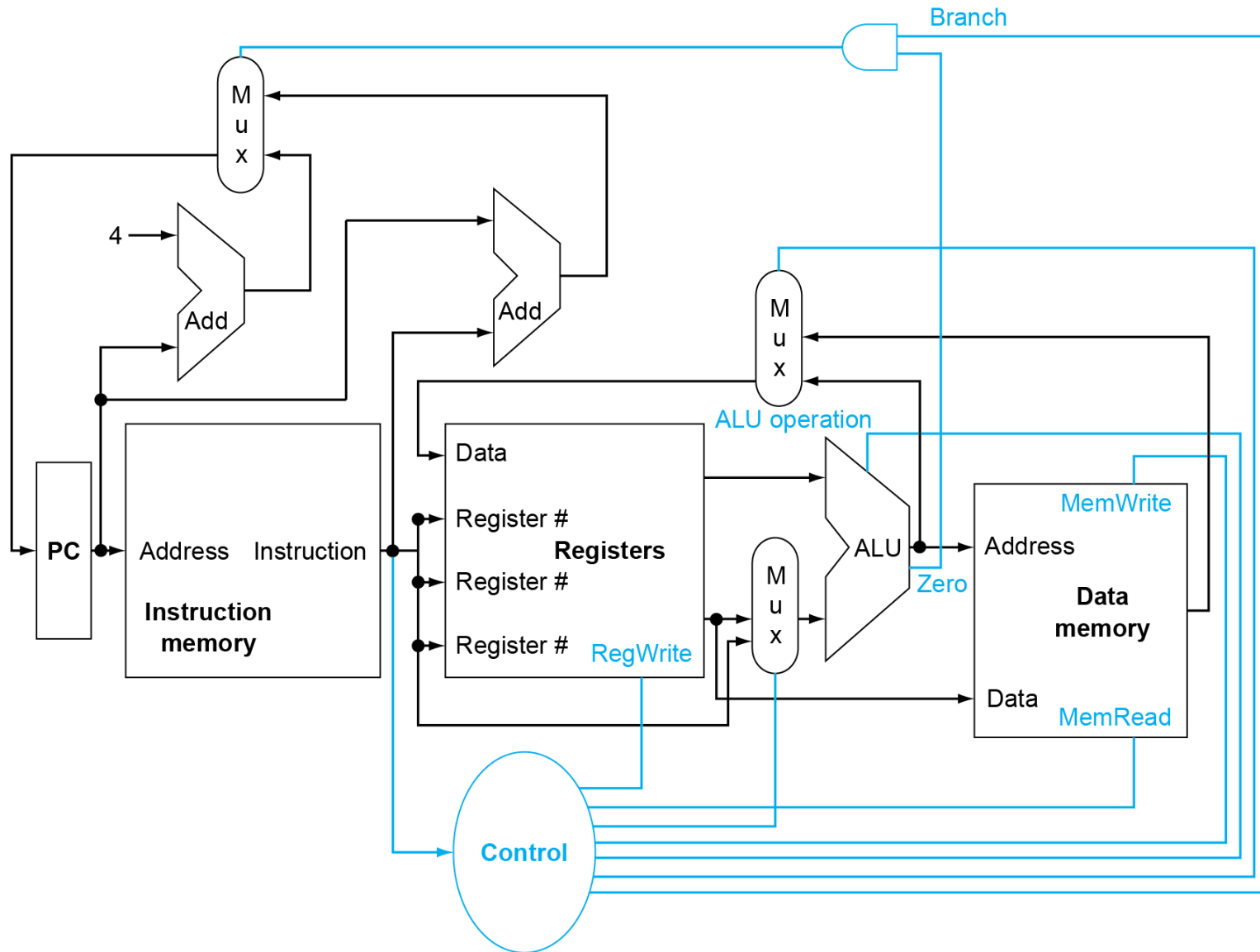


Multiplexers

- Can't just join wires together
- Use multiplexers



Control



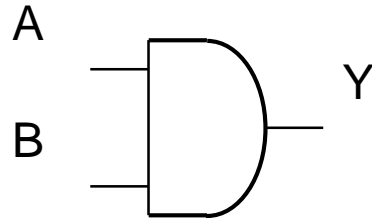
Logic Design Basics

- Information encoded in binary
 - Low voltage = 0, High voltage = 1
 - One wire per bit
 - Multi-bit data encoded on multi-wire buses
- Combinational element
 - Operate on data
 - Output is a function of input
- State (sequential) elements
 - Store information

Combinational Elements

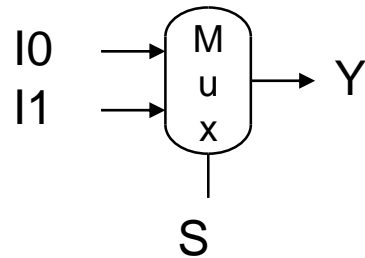
■ AND-gate

- $Y = A \& B$



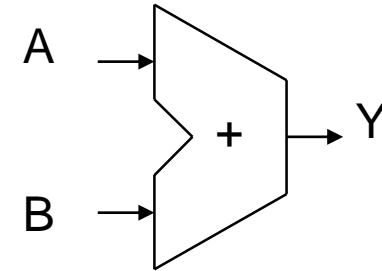
■ Multiplexer

- $Y = S ? I1 : I0$



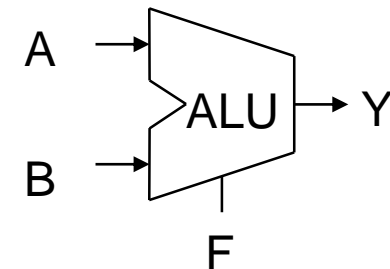
■ Adder

- $Y = A + B$



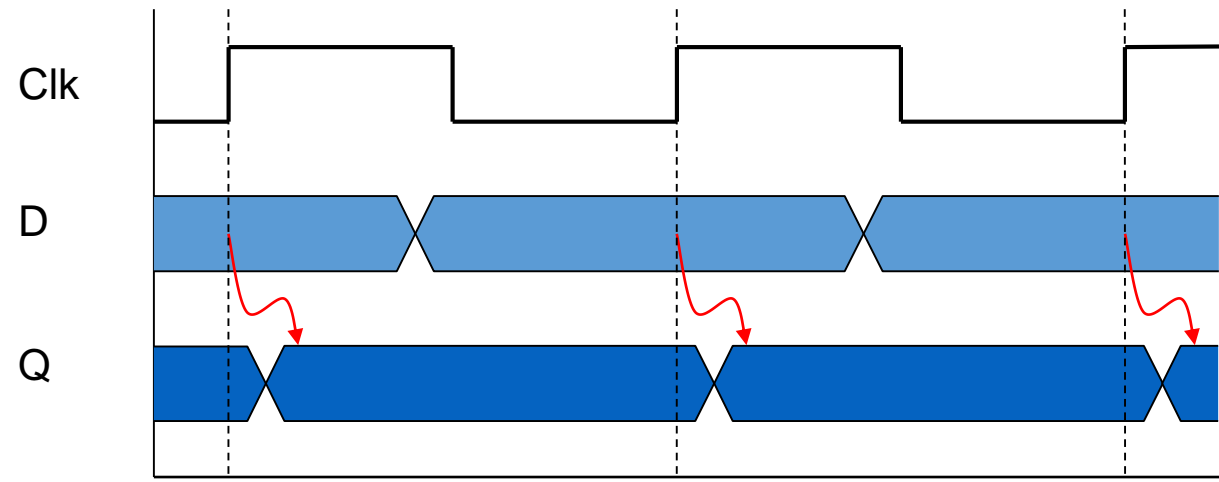
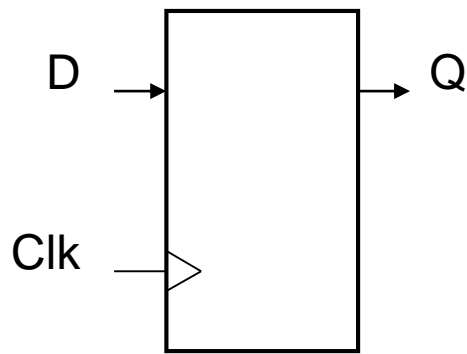
■ Arithmetic/Logic Unit

- $Y = F(A, B)$



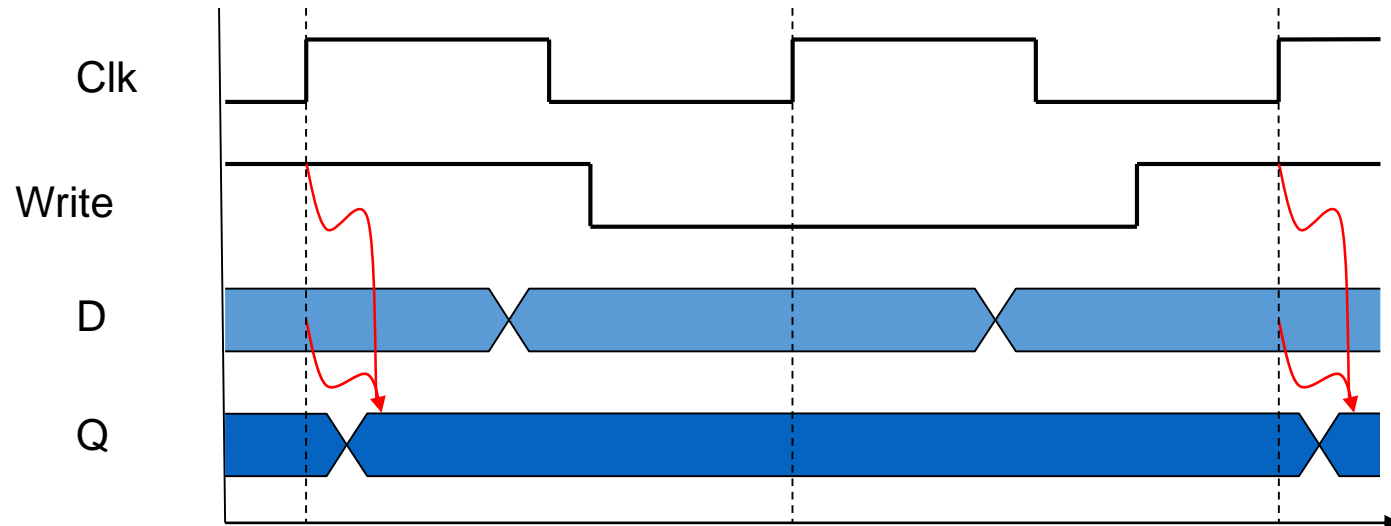
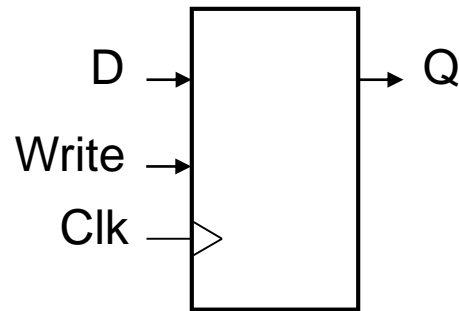
Sequential Elements

- Register: stores data in a circuit
 - Uses a clock signal to determine when to update the stored value
 - Edge-triggered: update when Clk changes from 0 to 1



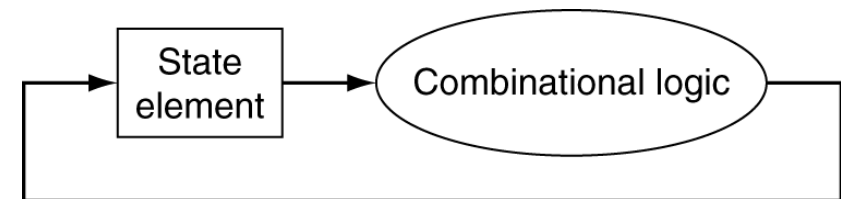
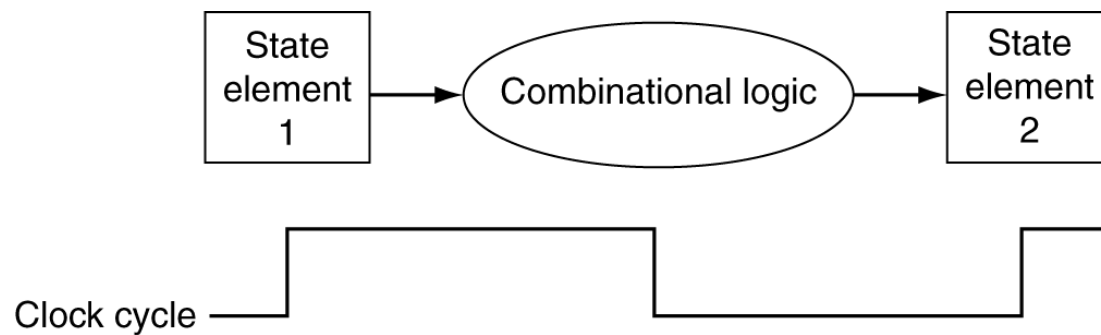
Sequential Elements

- Register with write control
 - Only updates on clock edge when write control input is 1
 - Used when stored value is required later



Clocking Methodology

- Combinational logic transforms data during clock cycles
 - Between clock edges
 - Input from state elements, output to state element
 - Longest delay determines clock period



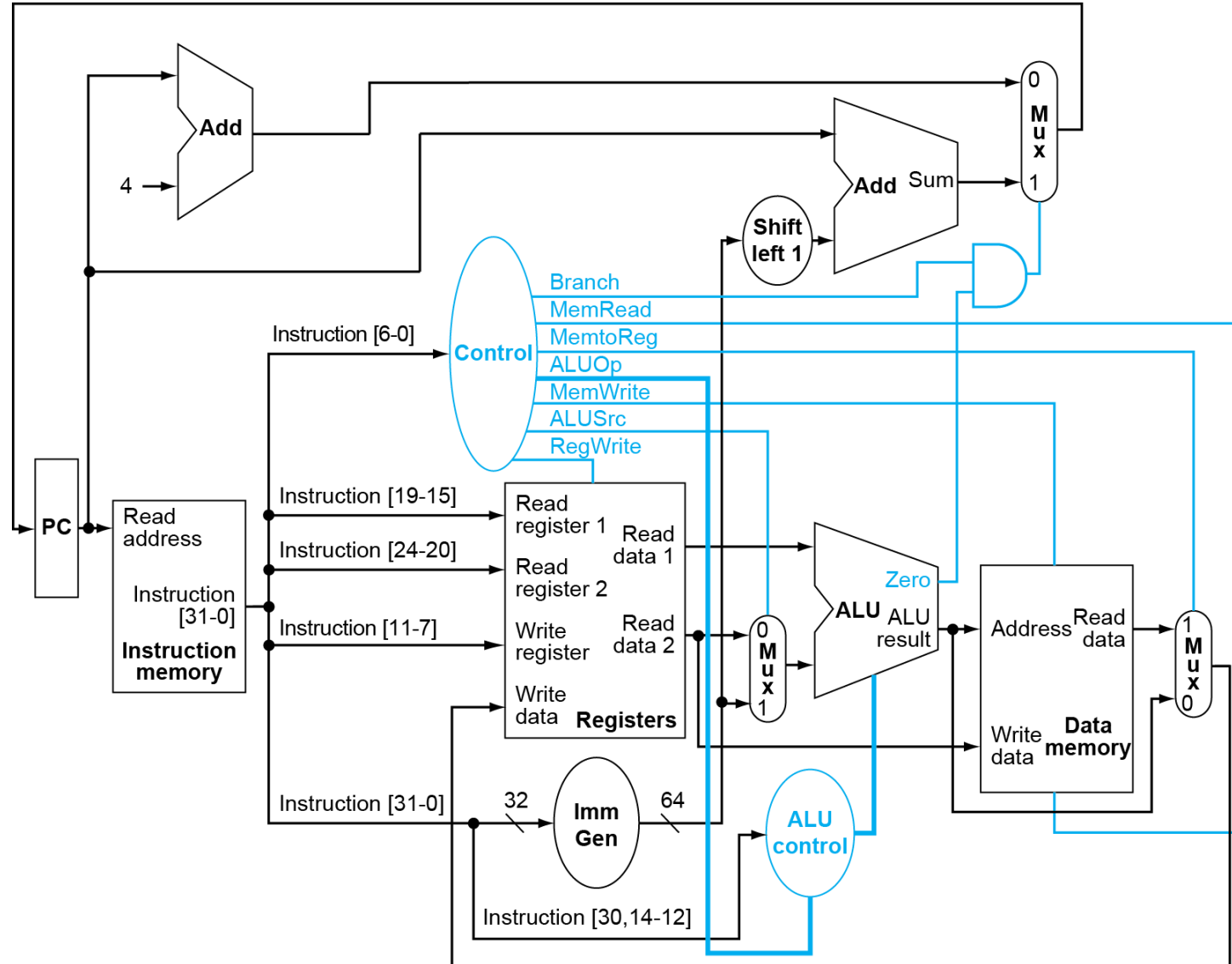
Main Control Unit

Control signals derived from instruction

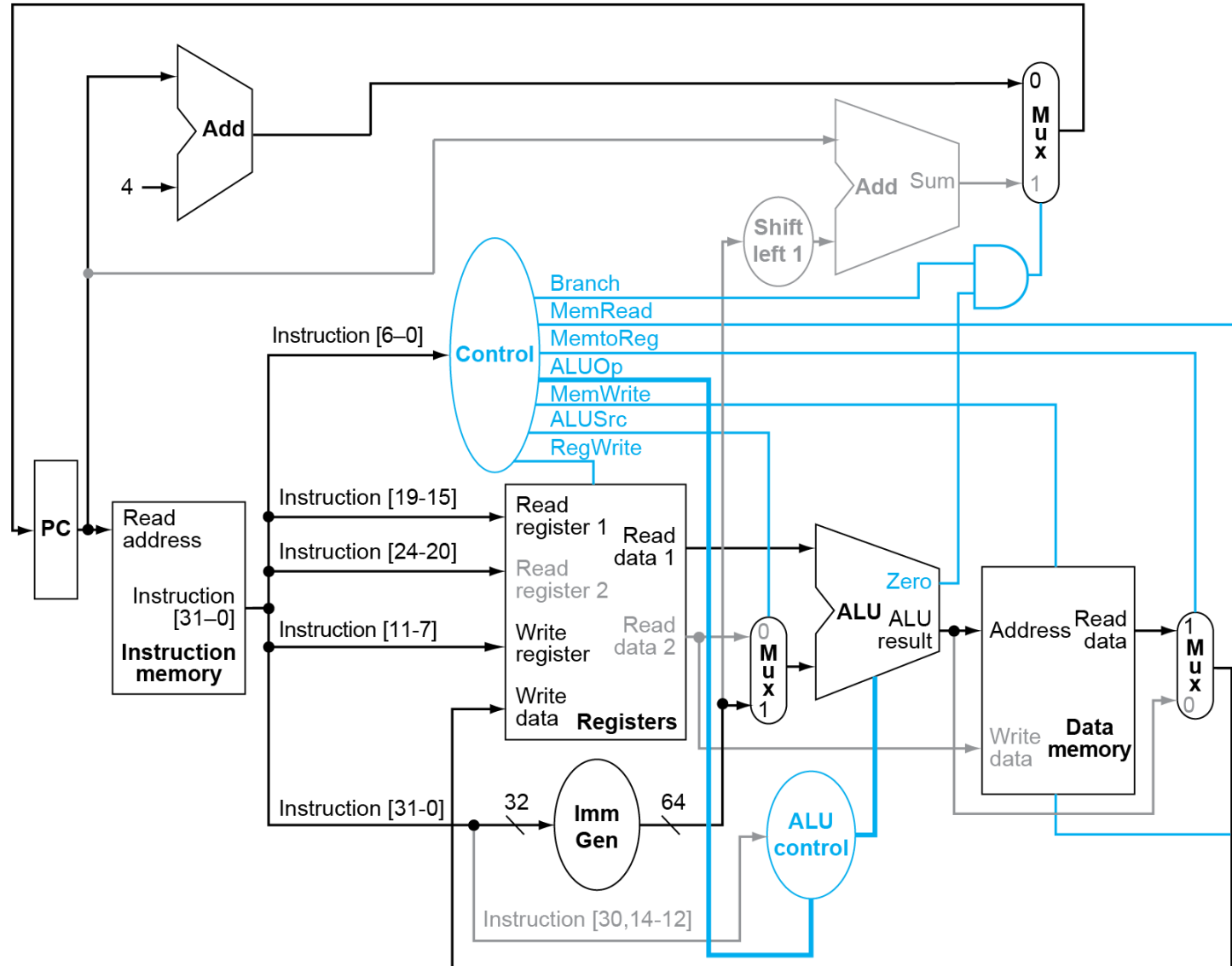
Name (Bit position)	Fields					
	31:25	24:20	19:15	14:12	11:7	6:0
(a) R-type	funct7	rs2	rs1	funct3	rd	opcode
(b) I-type	immediate[11:0]		rs1	funct3	rd	opcode
(c) S-type	immed[11:5]	rs2	rs1	funct3	immed[4:0]	opcode
(d) SB-type	immed[12,10:5]	rs2	rs1	funct3	immed[4:1,11]	opcode

ALUOp		Funct7 field						Funct3 field			Operation	
ALUOp1	ALUOp0	I[31]	I[30]	I[29]	I[28]	I[27]	I[26]	I[25]	I[14]	I[13]		I[12]
0	0	X	X	X	X	X	X	X	X	X	X	0010
X	1	X	X	X	X	X	X	X	X	X	X	0110
1	X	0	0	0	0	0	0	0	0	0	0	0010
1	X	0	1	0	0	0	0	0	0	0	0	0110
1	X	0	0	0	0	0	0	0	1	1	1	0000
1	X	0	0	0	0	0	0	0	1	1	0	0001

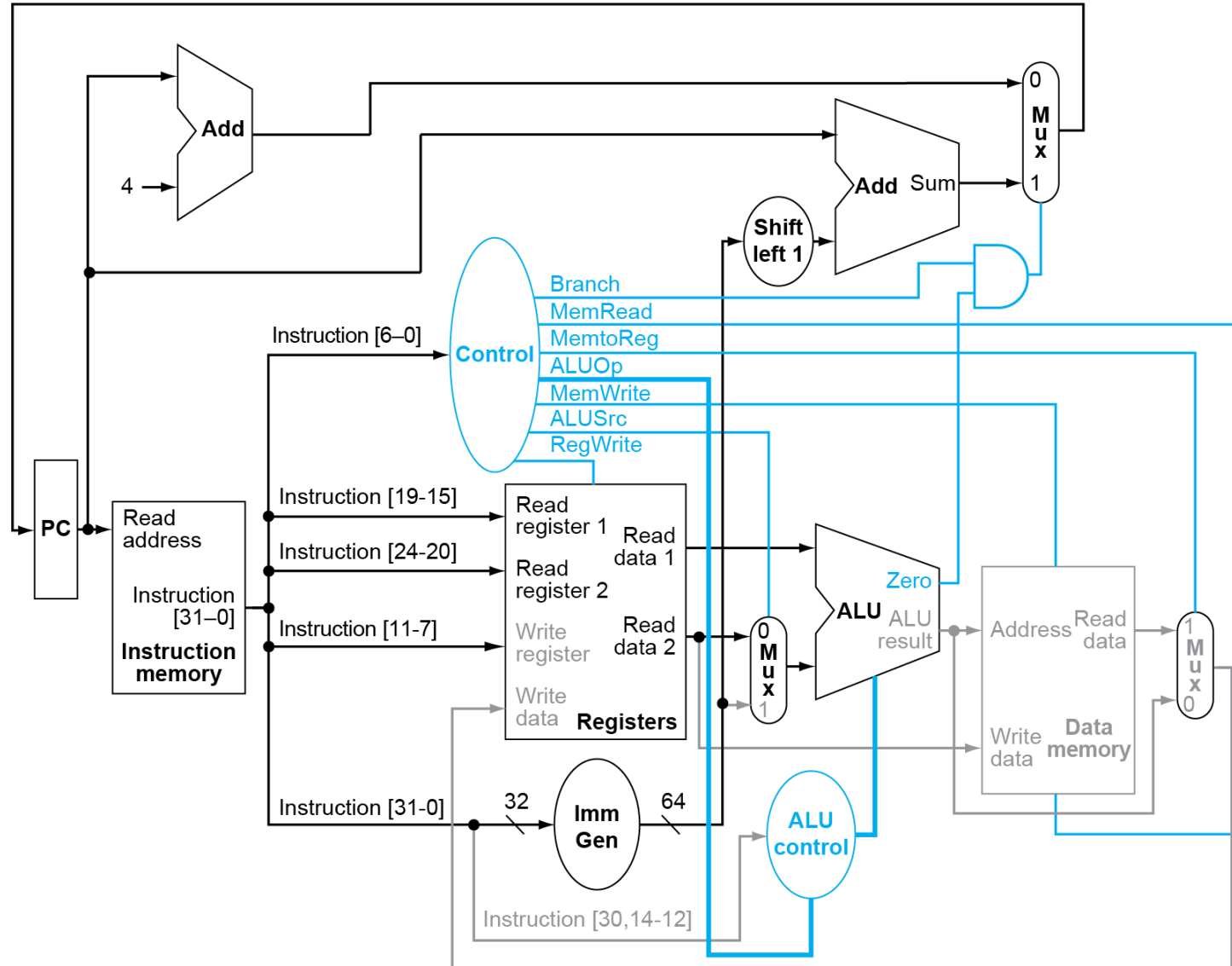
Datapath With Control



Load Instruction



BEQ Instruction



Performance Issues

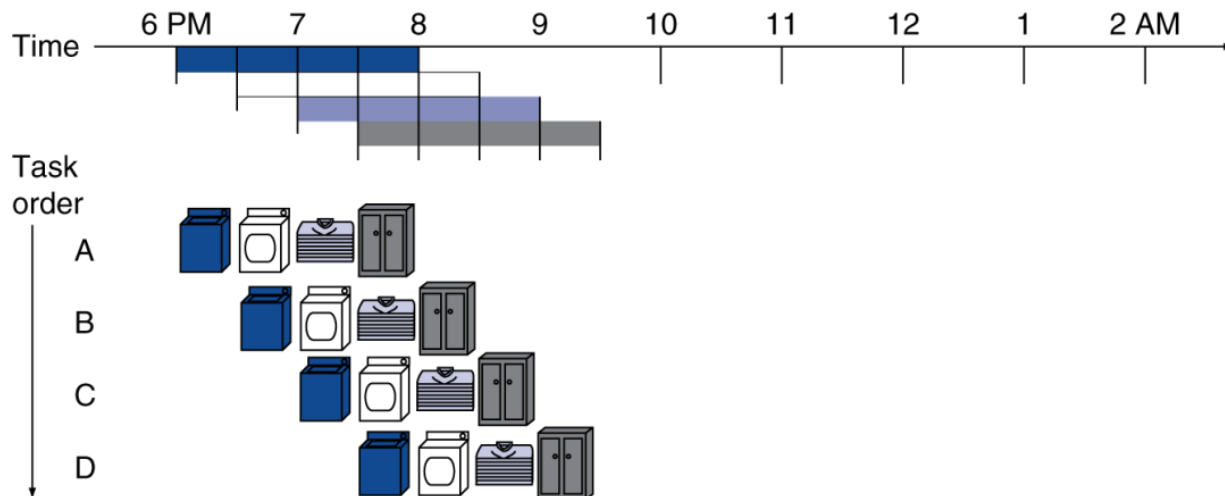
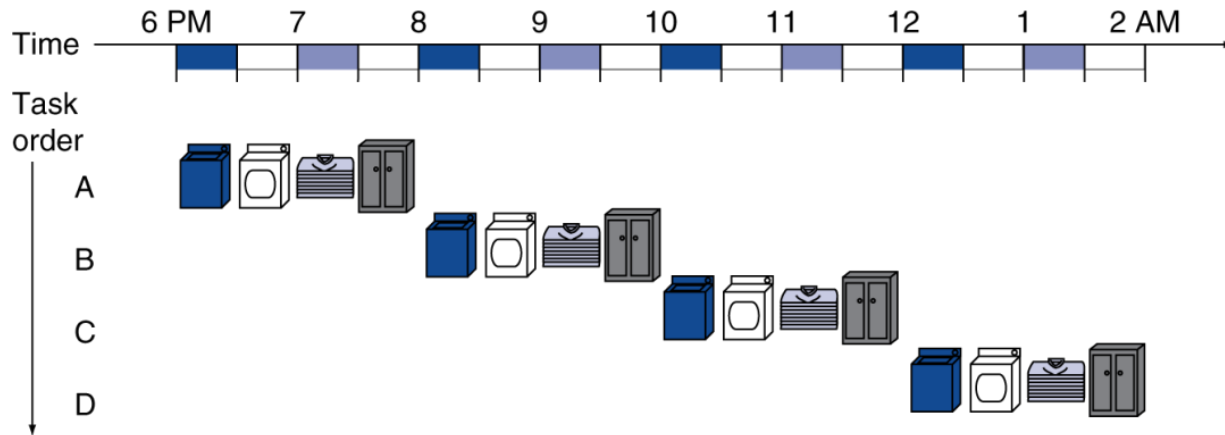
- Longest delay determines clock period
 - Critical path: load instruction
 - Instruction memory → register file → ALU → data memory → register file
- Not feasible to vary period for different instructions
- Violates design principle
 - Making the common case fast
- We will improve performance by pipelining

Response Time and Throughput

- Response time
 - How long it takes to do a task
- Throughput
 - Total work done per unit time

Pipelining Analogy

- Pipelined laundry: overlapping execution
 - Parallelism improves performance



- Four loads:
 - Speedup
 $= 8 / 3.5 = 2.3$
- Non-stop:
 - Speedup
 $= 2n / 0.5n + 1.5 \approx 4$
 $= \text{number of stages}$

RISC-V 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

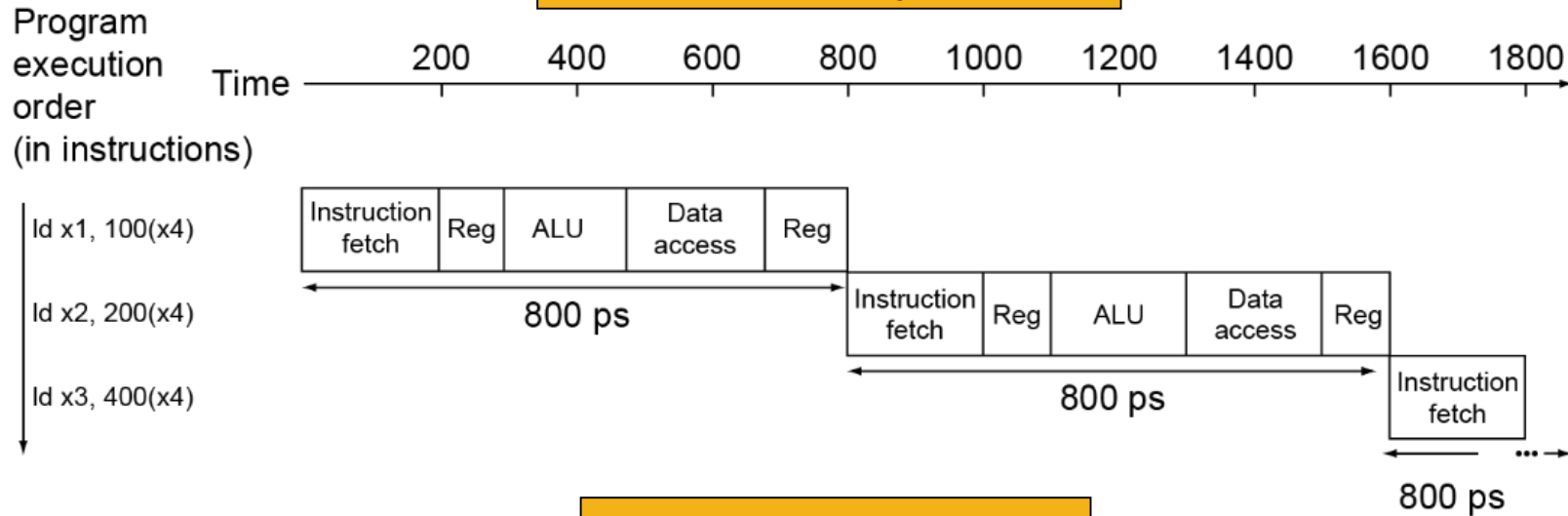
Pipeline Performance

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

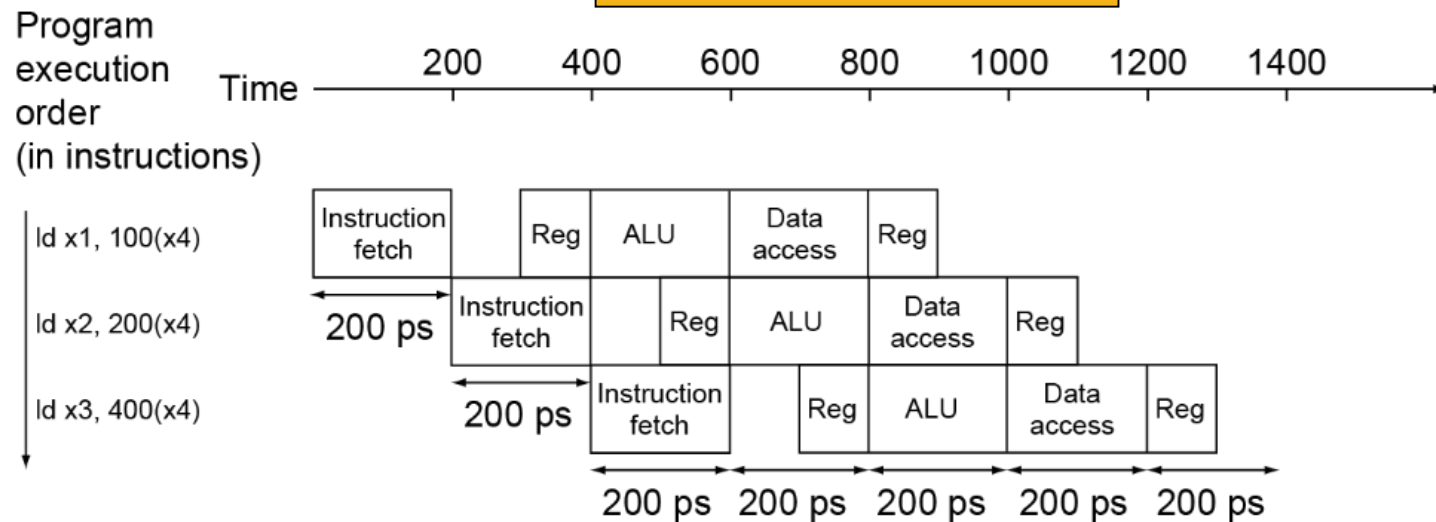
Instr	Instr fetch	Register read	ALU op	Memory access	Register write	Total time
ld	200ps	100 ps	200ps	200ps	100 ps	800ps
sd	200ps	100 ps	200ps	200ps		700ps
R-format	200ps	100 ps	200ps		100 ps	600ps
beq	200ps	100 ps	200ps			500ps

Pipeline Performance

Single-Cycle ($T_c = 800\text{ps}$)



Pipelined ($T_c = 200\text{ps}$)



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

- RISC-V 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

Hazards

Situations that prevent starting the next instruction in the next cycle

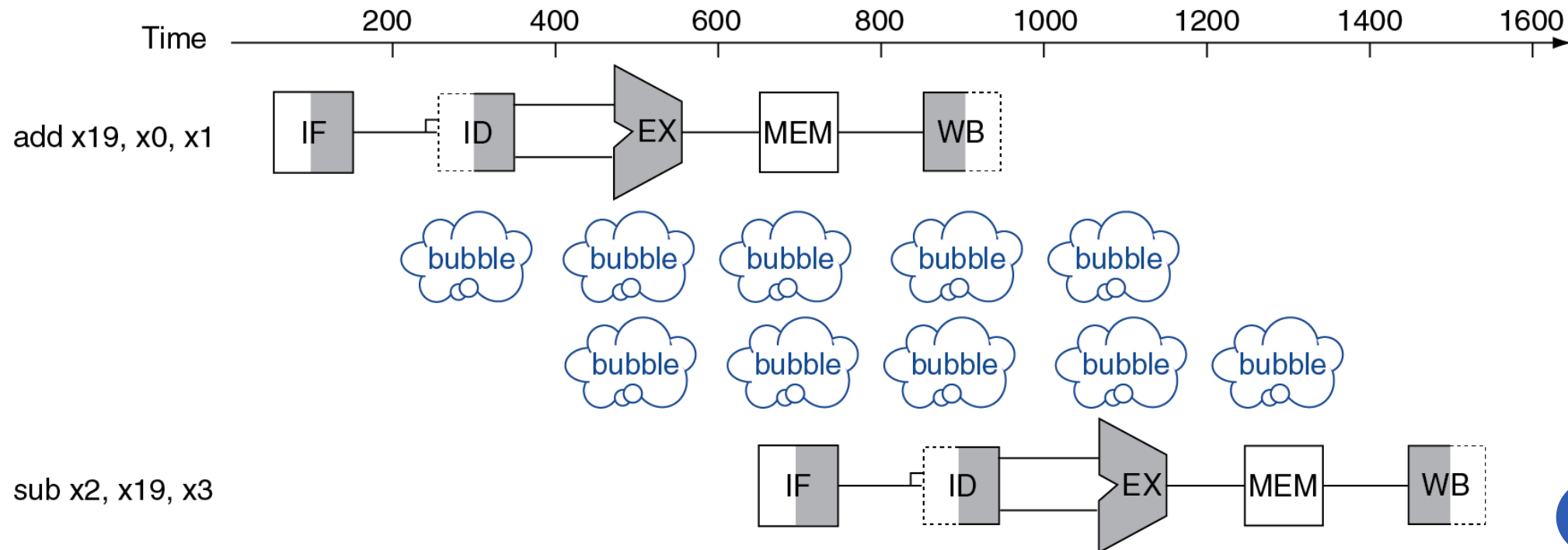
- Structure hazard
 - A required resource is busy
- Data hazard
 - Need to wait for previous instruction to complete its data read/write
- Control hazard
 - Deciding on control action depends on previous instruction

Structure Hazards

- Conflict for use of a resource
- In RISC-V 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

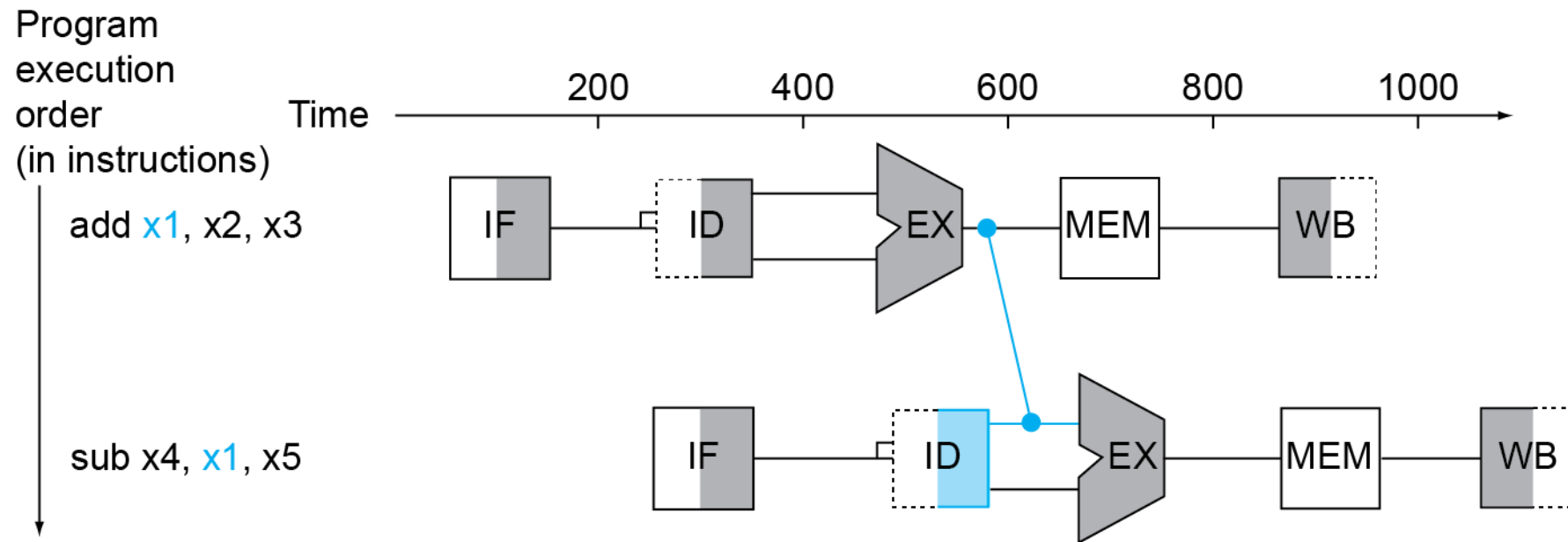
Data Hazards

- An instruction depends on completion of data access by a previous instruction
 - add x19, x0, x1
 - sub x2, x19, x3



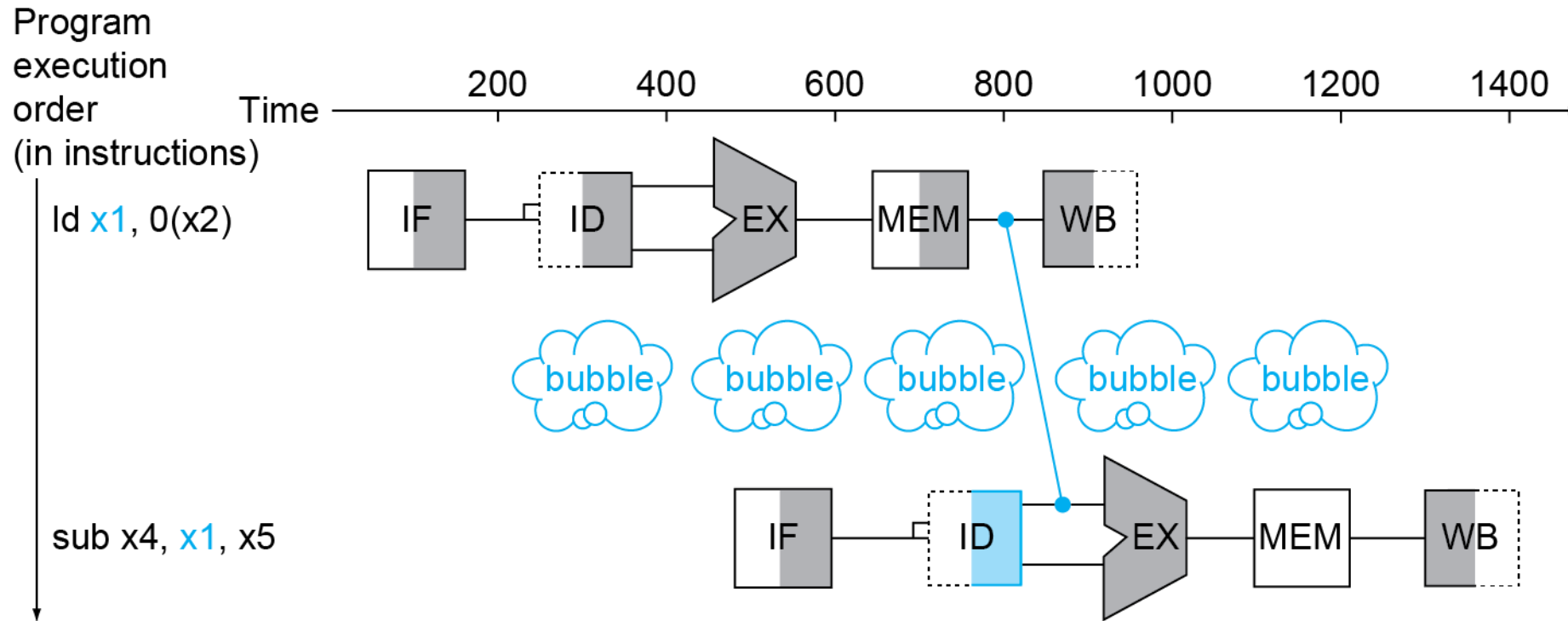
Forwarding (aka Bypassing)

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



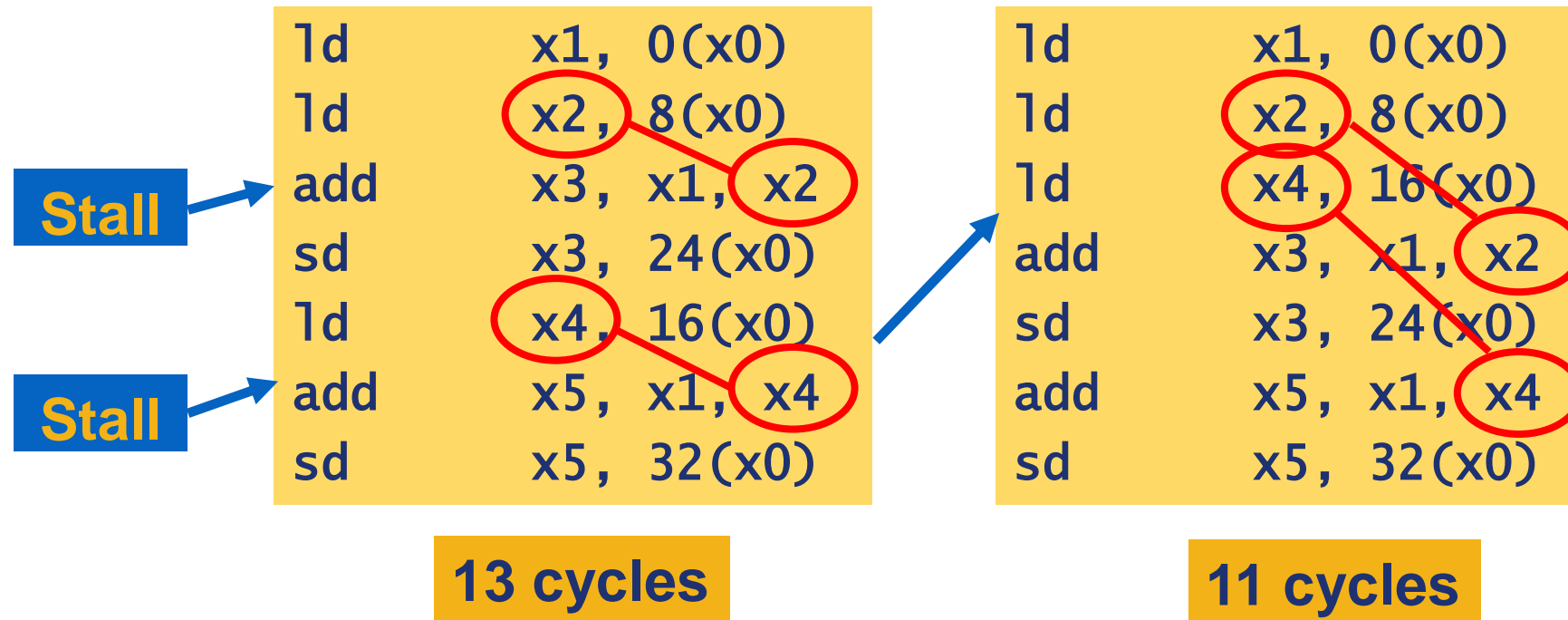
Load-Use Data Hazard

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



Code Scheduling to Avoid Stalls

- Reorder code to avoid use of load result in the next instruction
- C code for $a = b + e; c = b + f;$

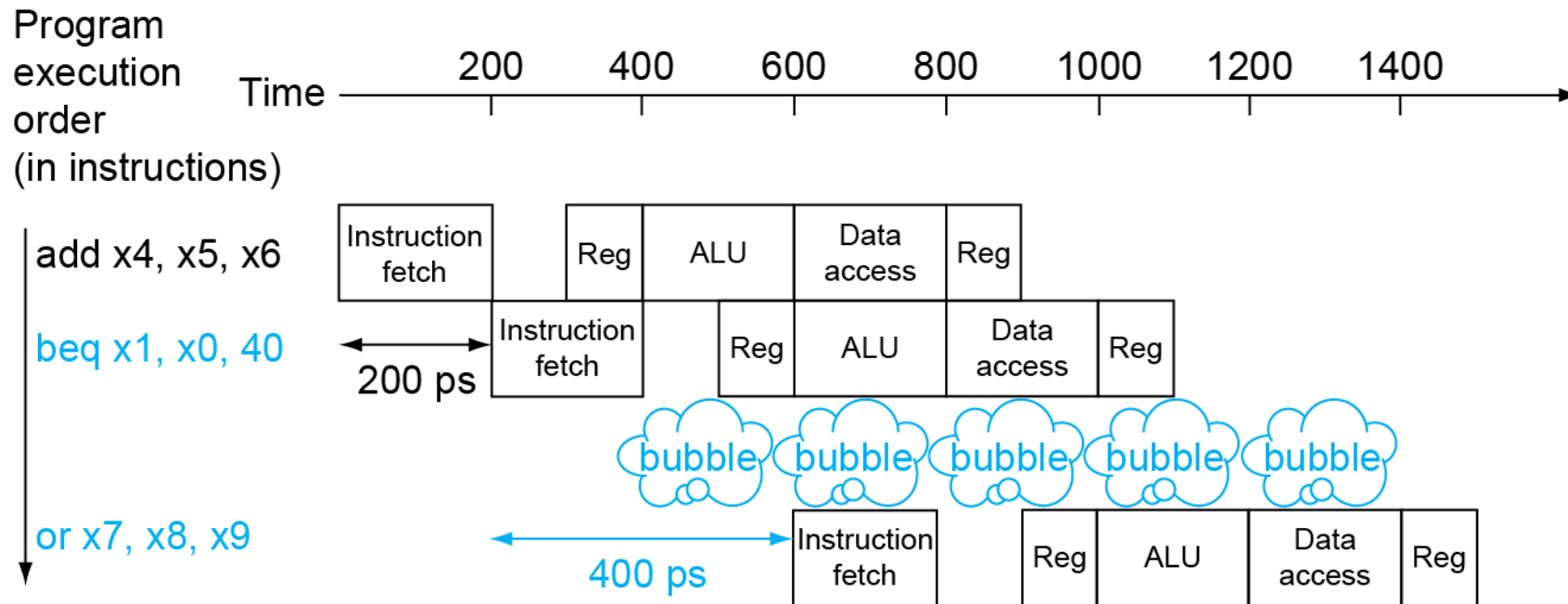


Control Hazards

- 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 RISC-V 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



Branch Prediction

- Longer pipelines cannot readily determine branch outcome early
 - Stall penalty becomes unacceptable
- Predict outcome of branch
 - Only stall if prediction is wrong
- In RISC-V pipeline
 - Can predict branches not taken
 - Fetch instruction after branch, with no delay

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

Pipeline Summary

- Pipelining improves performance by increasing instruction throughput
 - Executes multiple instructions in parallel
 - Each instruction has the same latency
- Subject to hazards
 - Structure, data, control
- Instruction set design affects complexity of pipeline implementation

Conclusion

- ISA influences design of datapath and control
- Datapath and control influence design of ISA
- Pipelining improves instruction throughput using parallelism
 - More instructions completed per second
 - Latency for each instruction not reduced
- Hazards: structural, data, control

Any Questions?

```
                .text
__start:      addi t1, zero, 0x18
              addi t2, zero, 0x21
cycle:       beg t1, t2, done
              slt t0, t1, t2
              bne t0, zero, if_less
              nop
              sub t1, t1, t2
              j cycle
              nop
if_less:     sub t2, t2, t1
              j cycle
done:       add t3, t1, zero
```