CS352H: Computer Systems Architecture

Topic 8: MIPS Pipelined Implementation

September 29, 2009

MIPS Pipeline

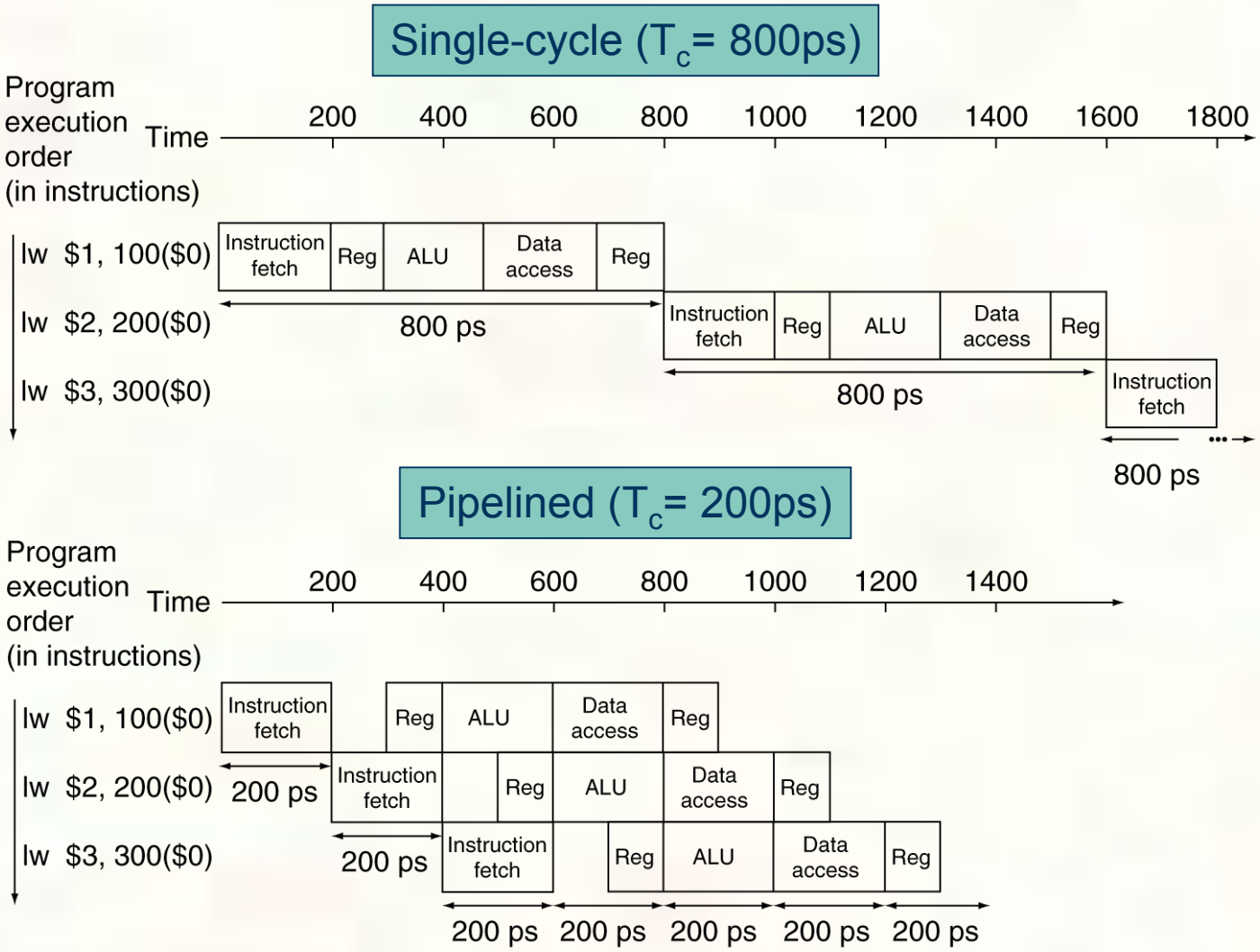
- Five stages, one step per stage
  - IF: Instruction fetch from memory
  - ID: Instruction decode & register read
  - EX: Execute operation or calculate address
  - MEM: Access memory operand
  - WB: Write result back to register

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
j	200ps					200ps

Pipeline Performance



Pipeline Speedup

- If all stages are balanced
  - i.e., all take the same time
  - Time between instructions$_{pipelined}$

$$= \frac{\text{Time between instructions}_{nonpipelined}}{\text{Number of stages}}$$

- 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

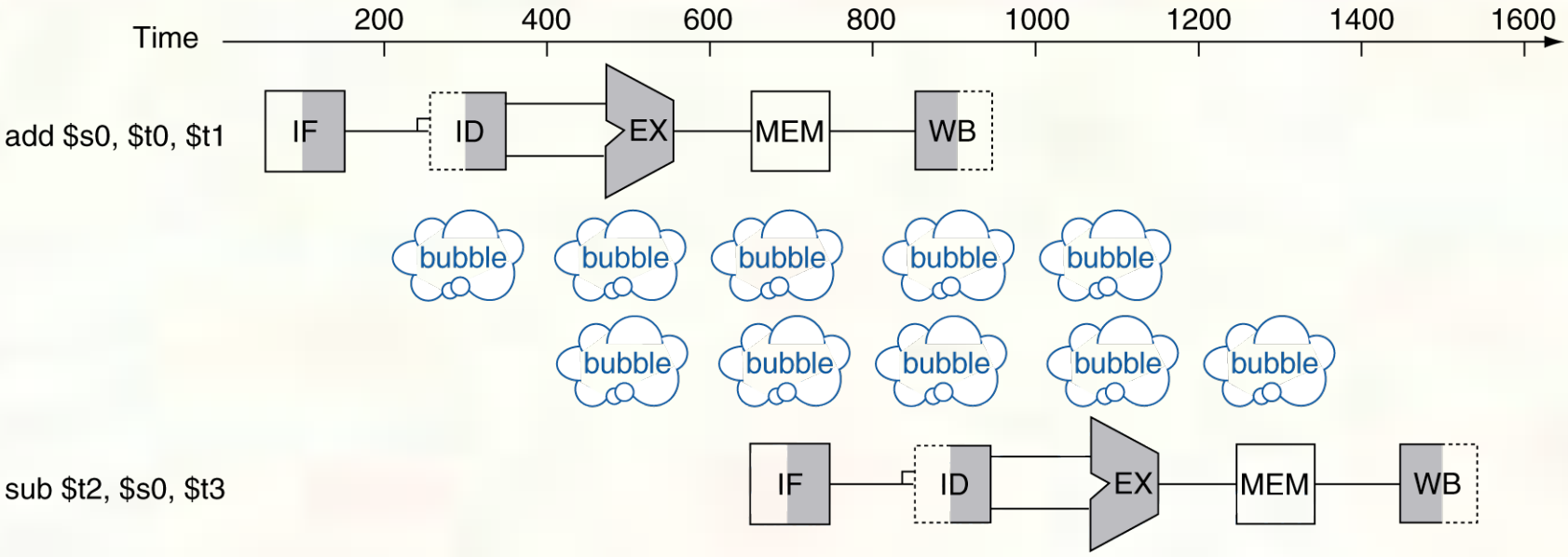
- Situations that prevent starting the next instruction in the next cycle
- Structure hazards
  - A required resource is busy
- Data hazard
  - Need to wait for previous instruction to complete its data read/write
- Control hazard
  - Deciding on control action depends on previous instruction

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

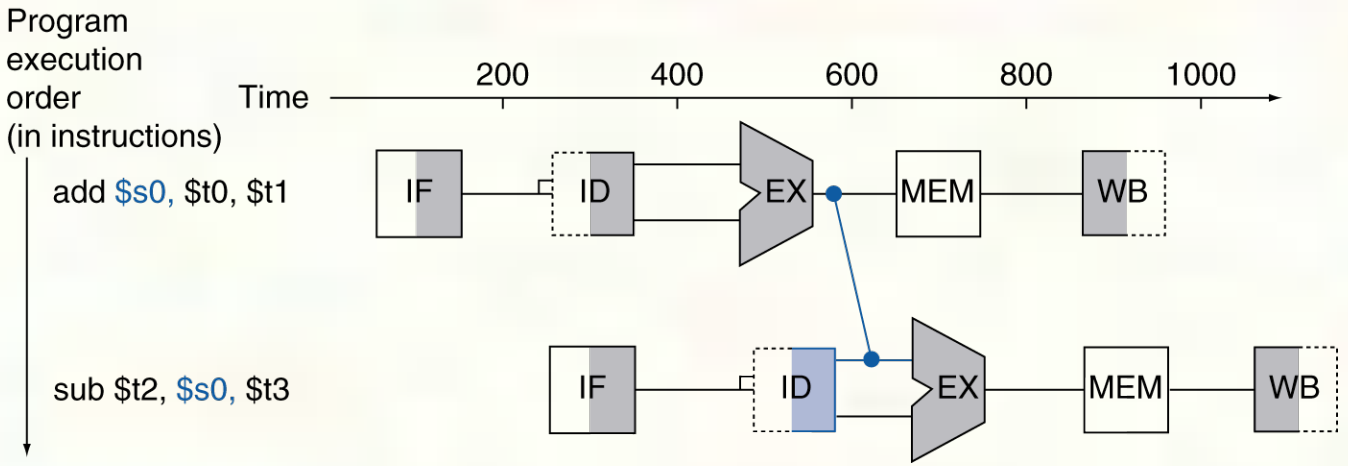
Data Hazards

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



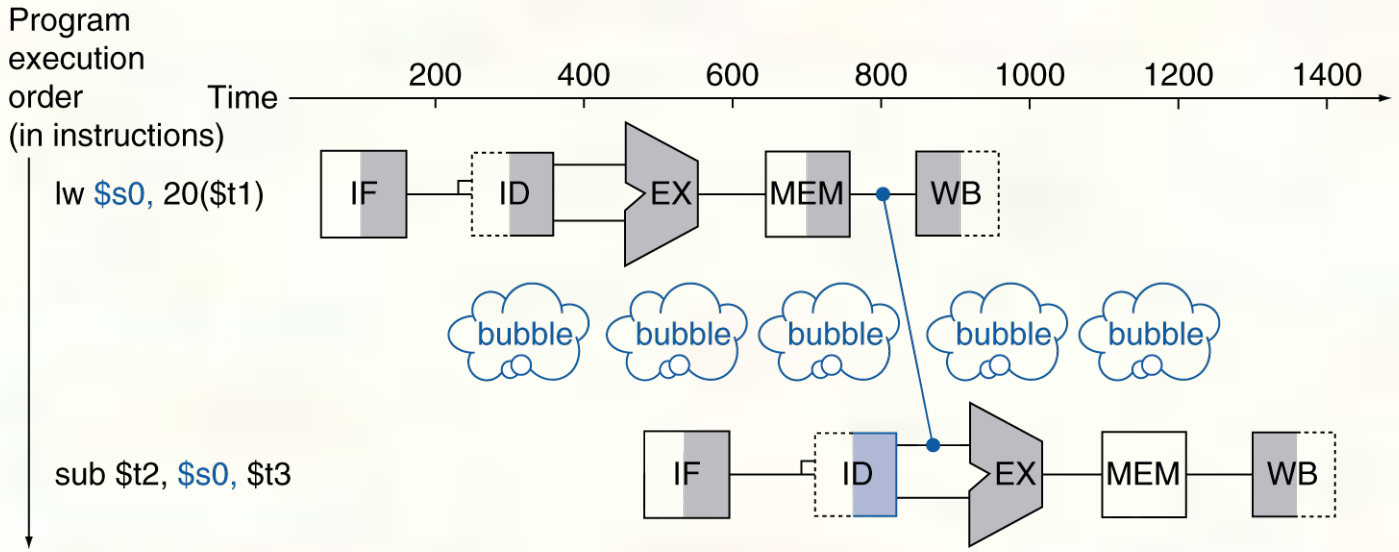
Forwarding (aka Bypassing)

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



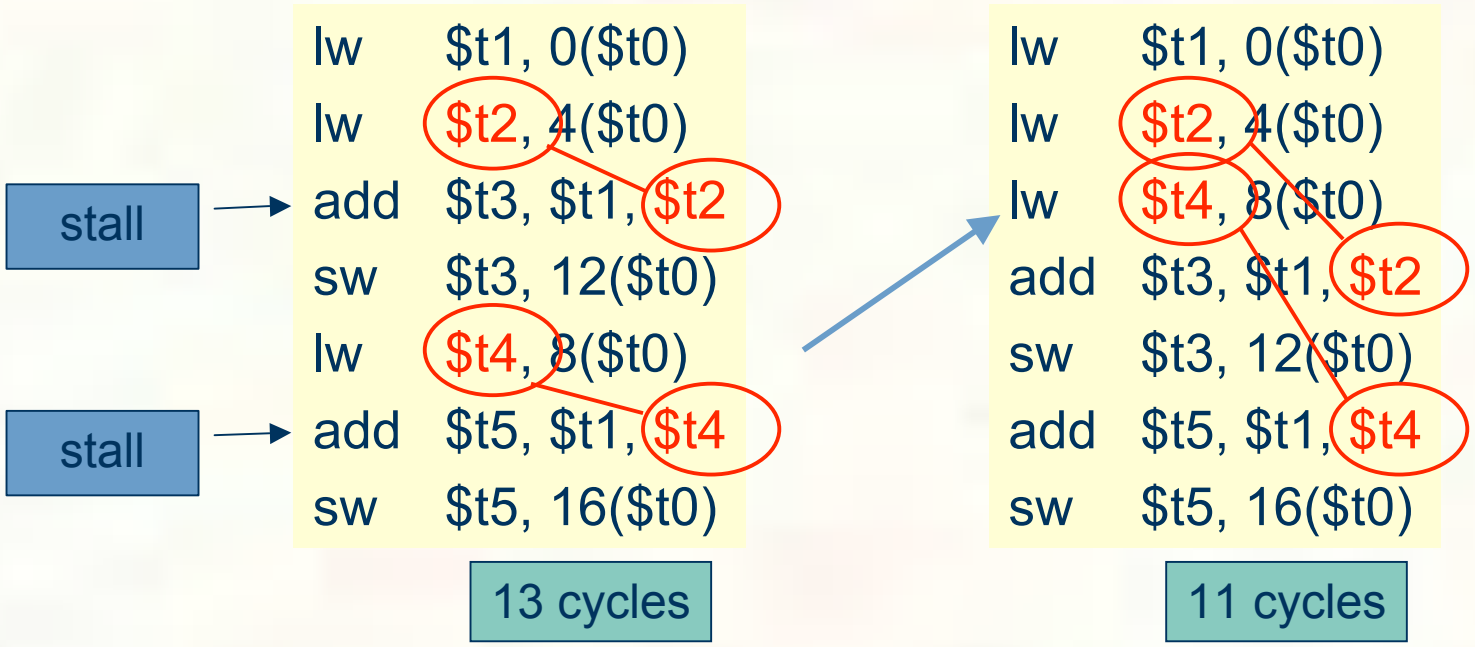
Load-Use Data Hazard

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



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;

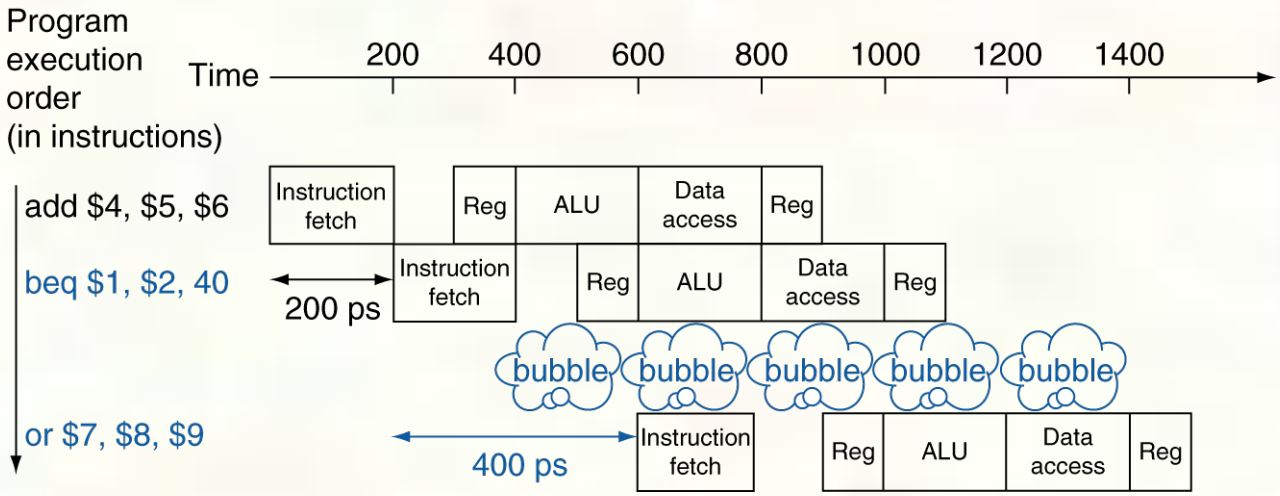


Control Hazards

- 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

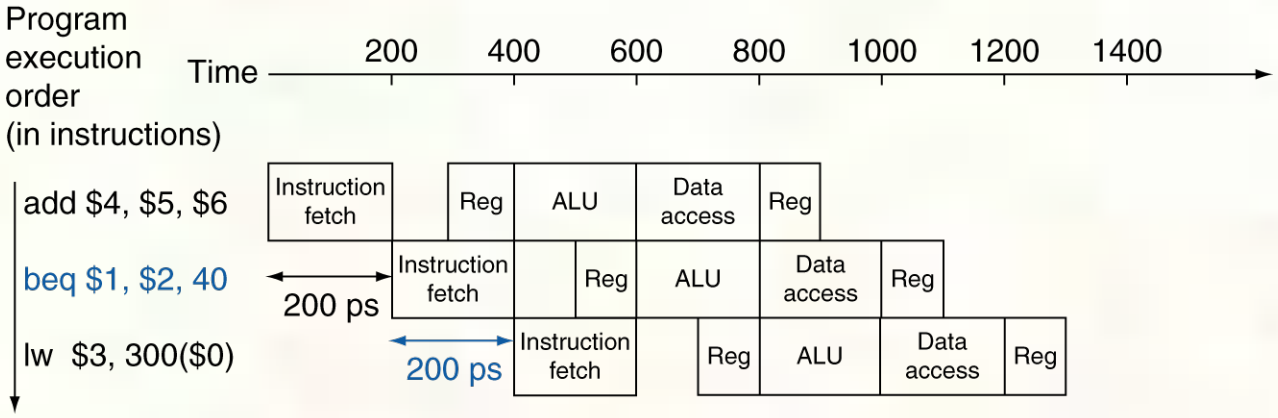


Branch Prediction

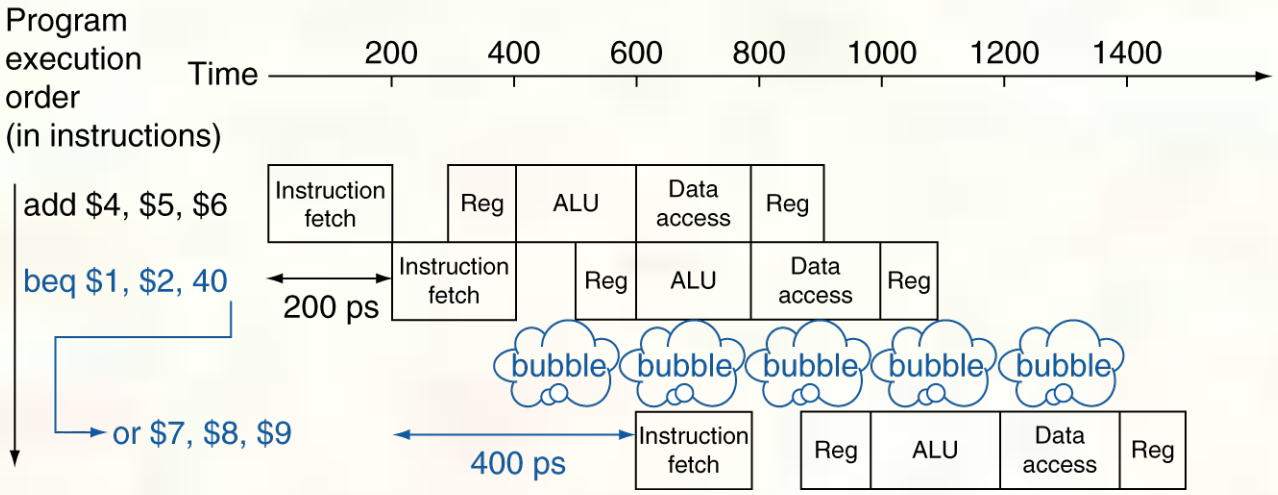
- Longer pipelines can't readily determine branch outcome early
  - Stall penalty becomes unacceptable
- Predict outcome of branch
  - Only stall if prediction is wrong
- In MIPS pipeline
  - Can predict branches not taken
  - Fetch instruction after branch, with no delay

MIPS with Predict Not Taken

Prediction correct



Prediction incorrect



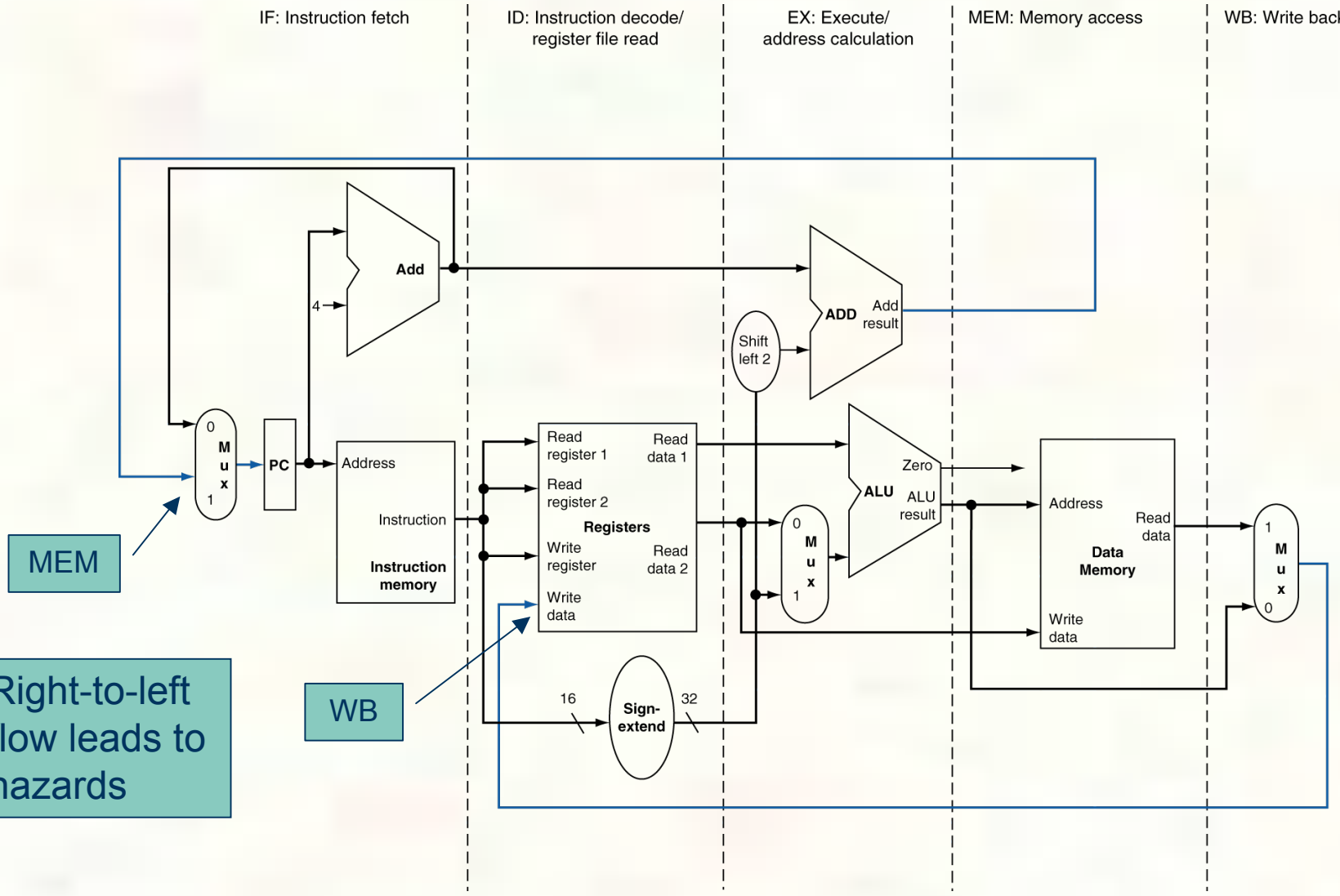
More-Realistic Branch Prediction

- 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

Pipeline Summary

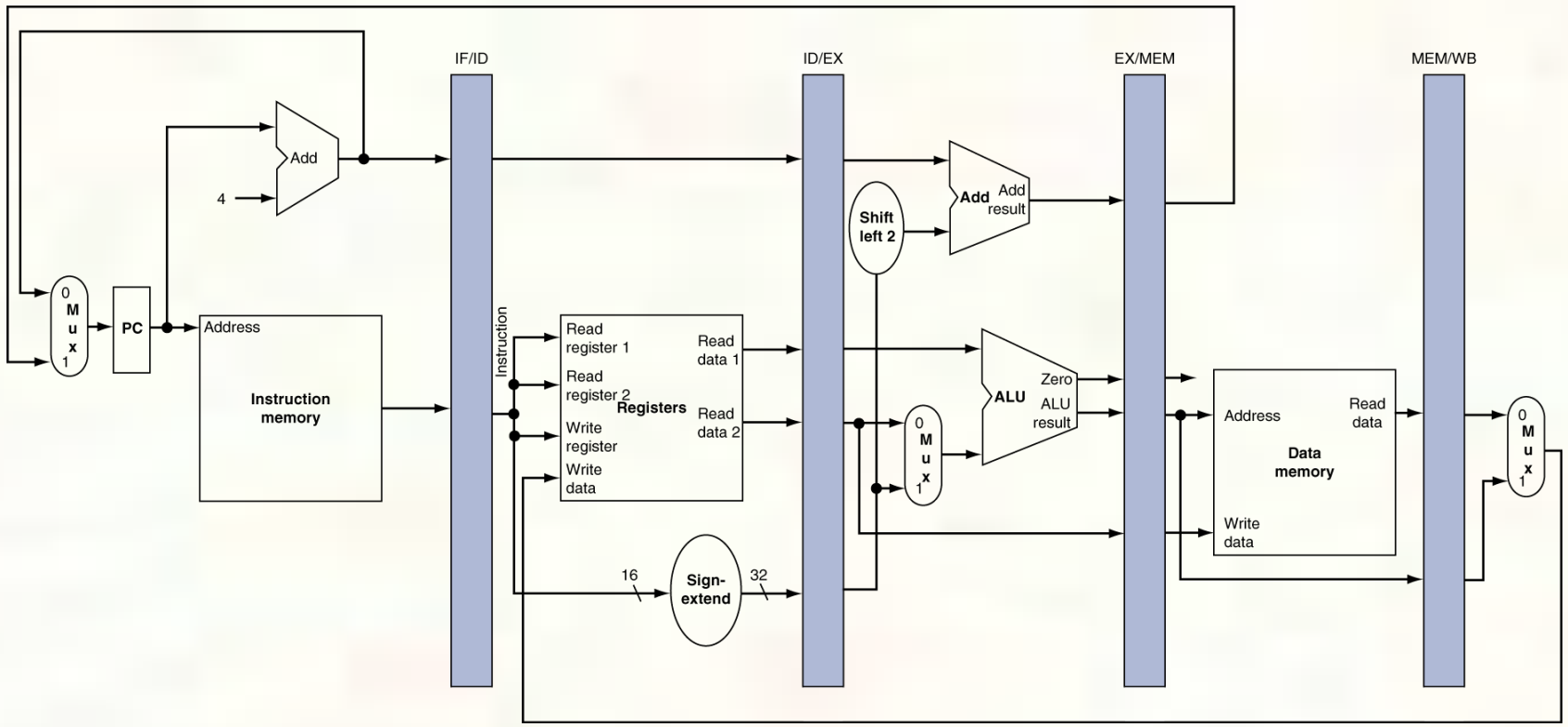
- Pipelining improves performance by increasing instruction throughput
  - Executes multiple instructions in parallel
  - Each instruction has the same latency
- Subject to hazards
  - Structure, data, control
- Instruction set design affects complexity of pipeline implementation

MIPS Pipelined Datapath



Pipeline registers

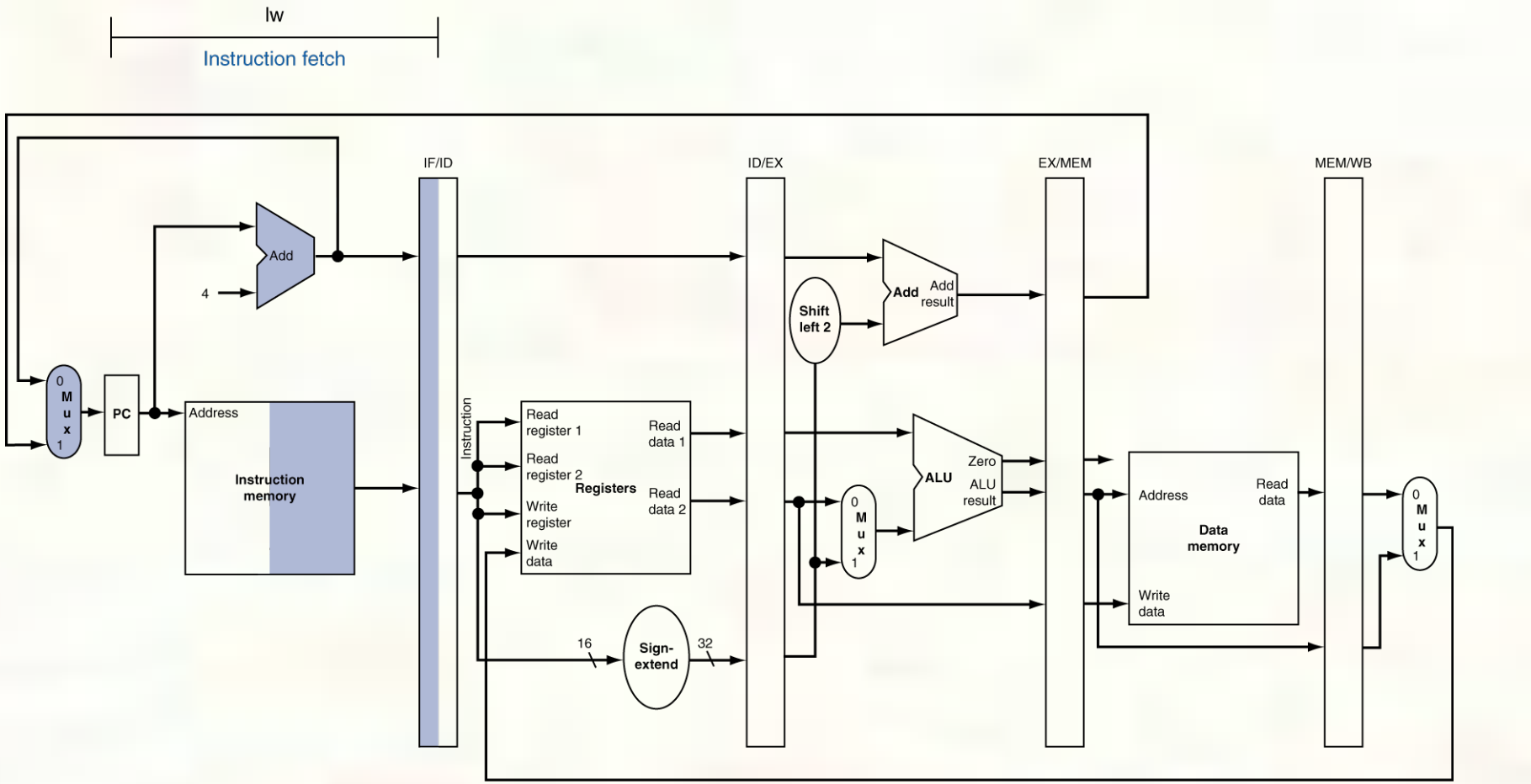
- Need registers between stages
  - To hold information produced in previous cycle



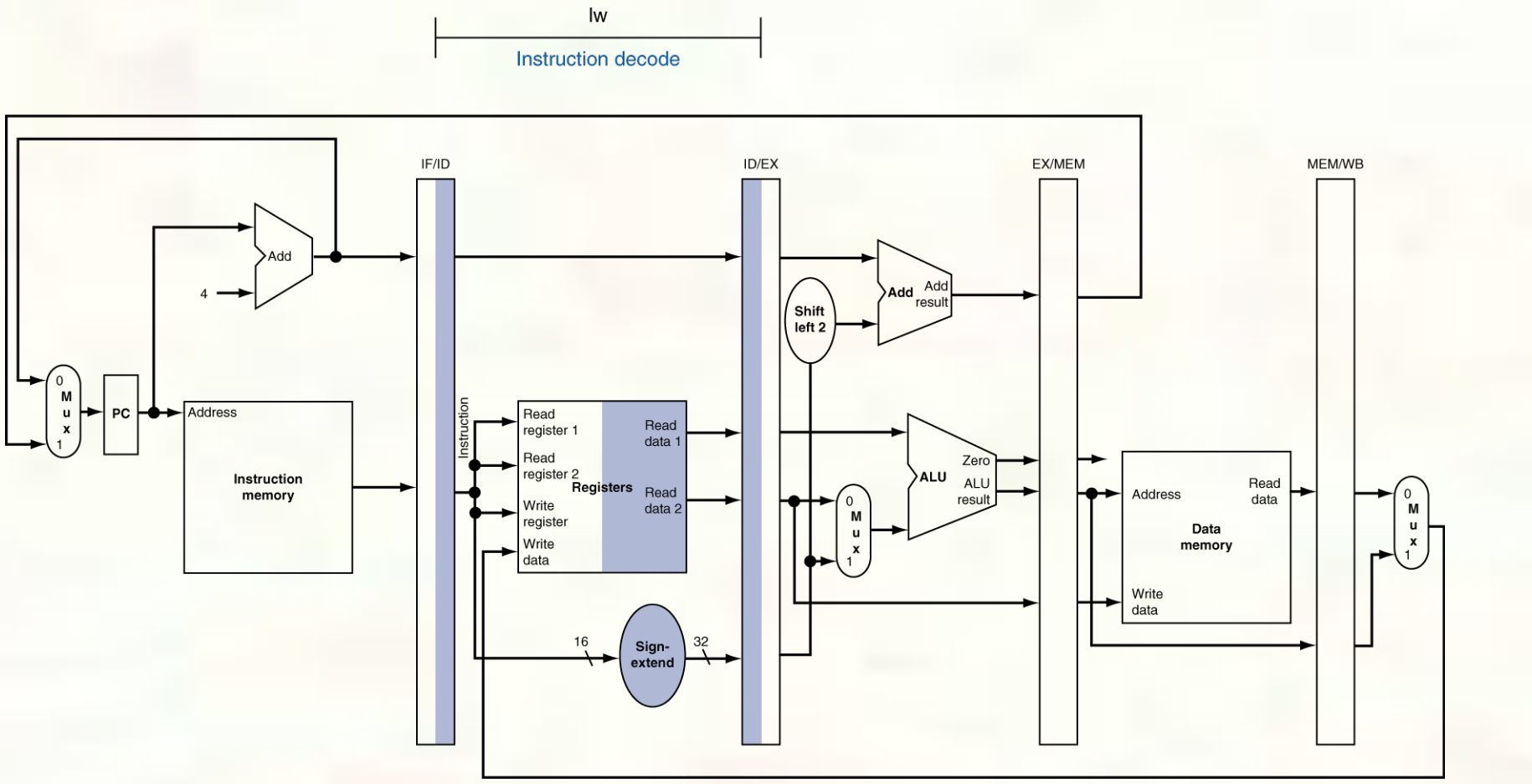
Pipeline Operation

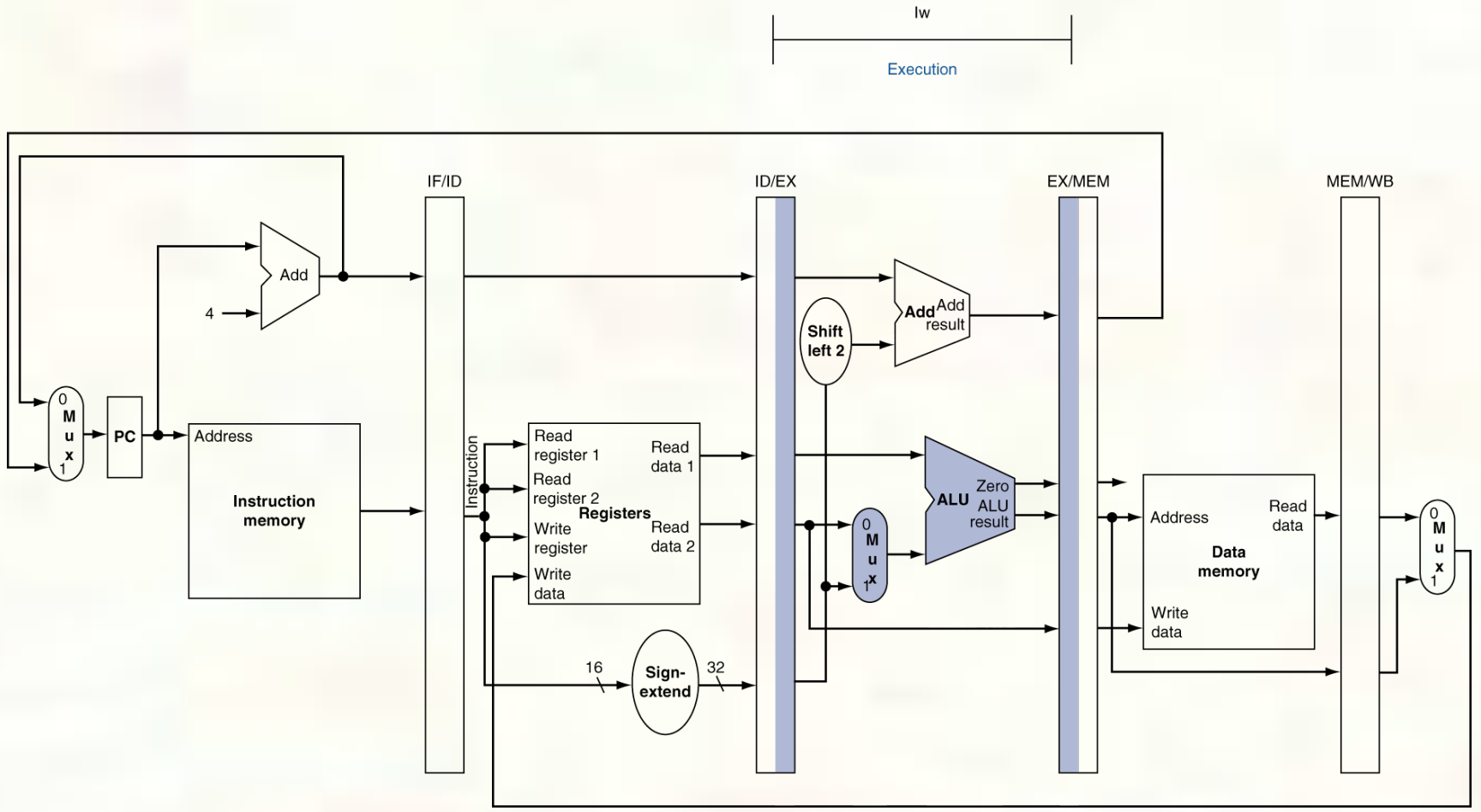
- Cycle-by-cycle flow of instructions through the pipelined datapath
  - "Single-clock-cycle" pipeline diagram
    - Shows pipeline usage in a single cycle
    - Highlight resources used
  - c.f. "multi-clock-cycle" diagram
    - Graph of operation over time
- We'll look at "single-clock-cycle" diagrams for load & store

IF for Load, Store, …

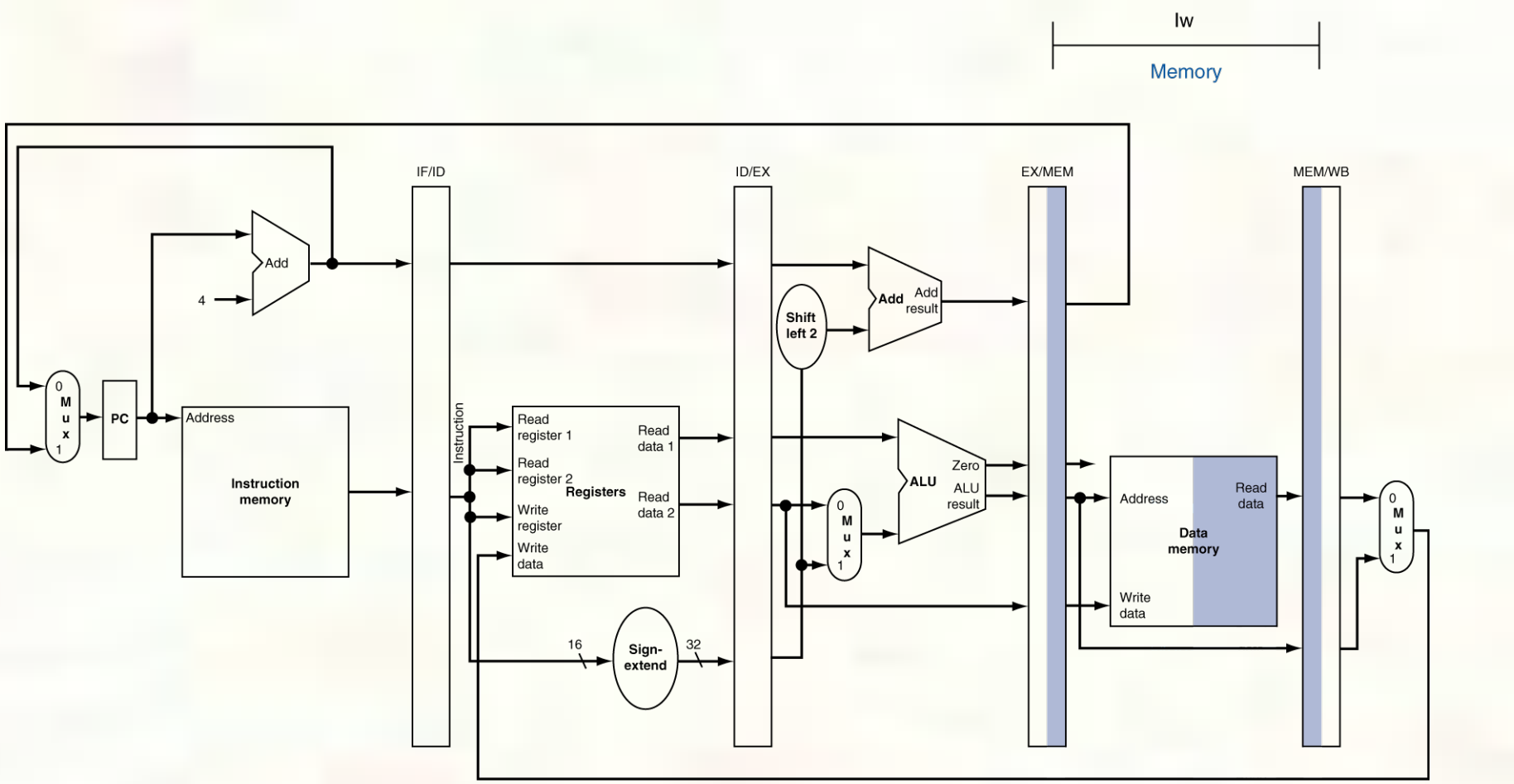


ID for Load, Store, …

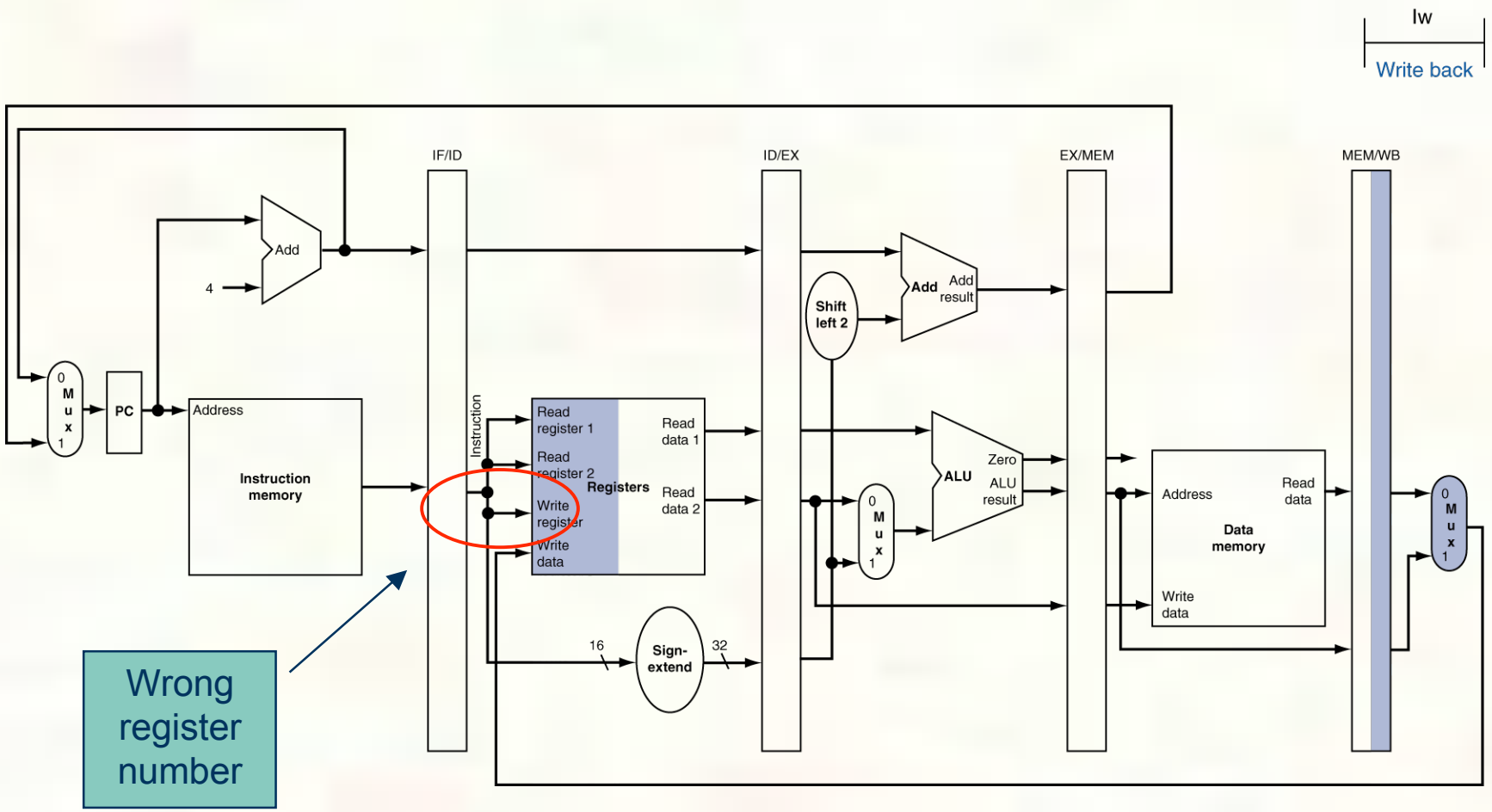



lw
Execution
IF/ID
ID/EX
EX/MEM
MEM/WB
Add
4
M
u
x
0
1
PC
Address
Instruction memory
Instruction
Read register 1
Read register 2
Registers
Write register
Write data
Read data 1
Read data 2
16
Sign-extend
32
Shift left 2
Add
Add result
ALU
Zero
ALU result
Address
Read data
Data memory
Write data

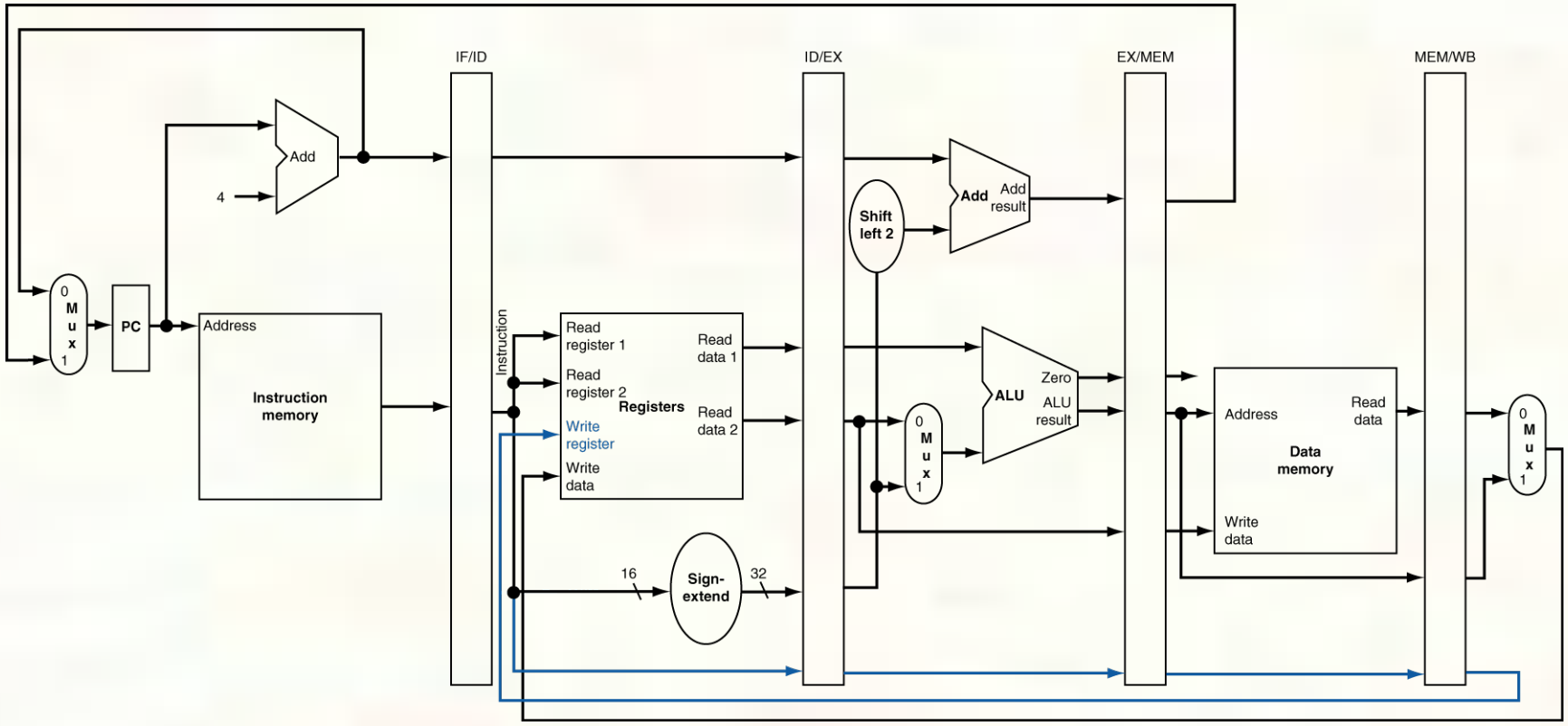
MEM for Load

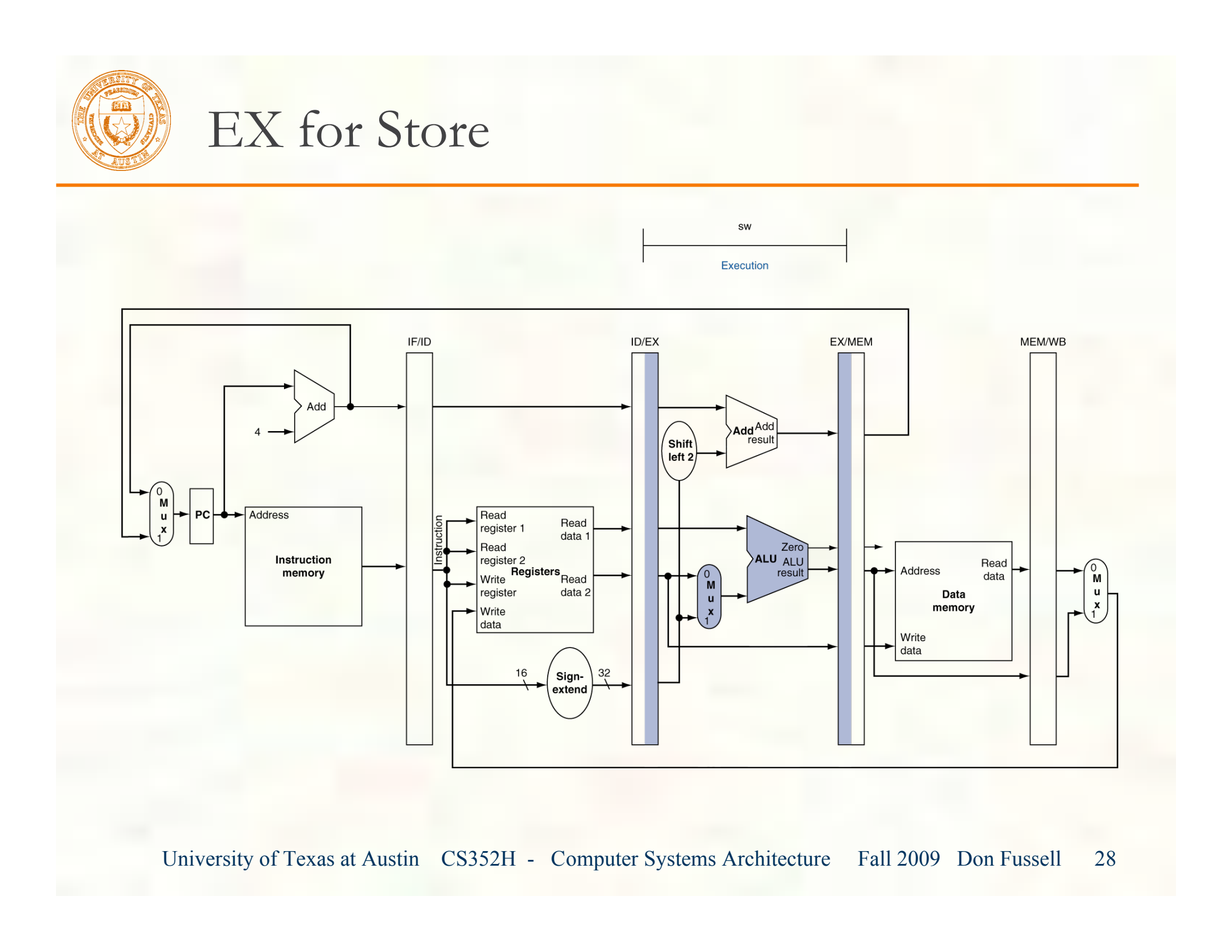

lw
Memory
IF/ID
ID/EX
EX/MEM
MEM/WB
Add
4
M u x
0
1
PC
Address
Instruction memory
Instruction
Read register 1
Read register 2
Registers
Write register
Write data
Read data 1
Read data 2
Shift left 2
Add
Add result
ALU
Zero
ALU result
16
Sign-extend
32
Address
Read data
Data memory
Write data

WB for Load


lw
Write back
IF/ID
ID/EX
EX/MEM
MEM/WB
Add
4
M u x
0
1
PC
Address
Instruction memory
Instruction
Read register 1
Read register 2
Registers
Write register
Write data
Read data 1
Read data 2
Shift left 2
Add
Add result
ALU
Zero
ALU result
Address
Read data
Data memory
Write data
16
Sign-extend
32
Wrong register number

Corrected Datapath for Load

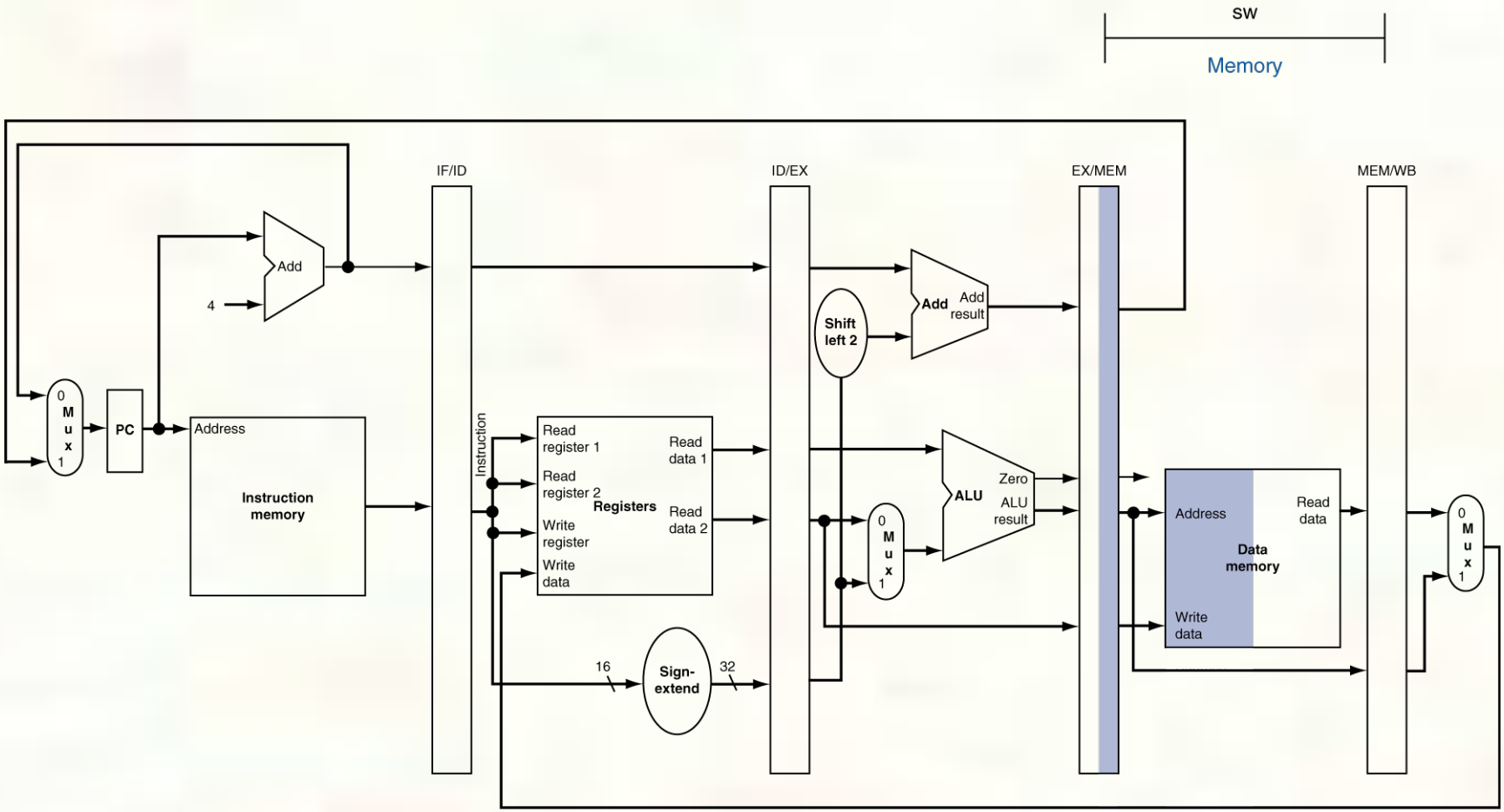




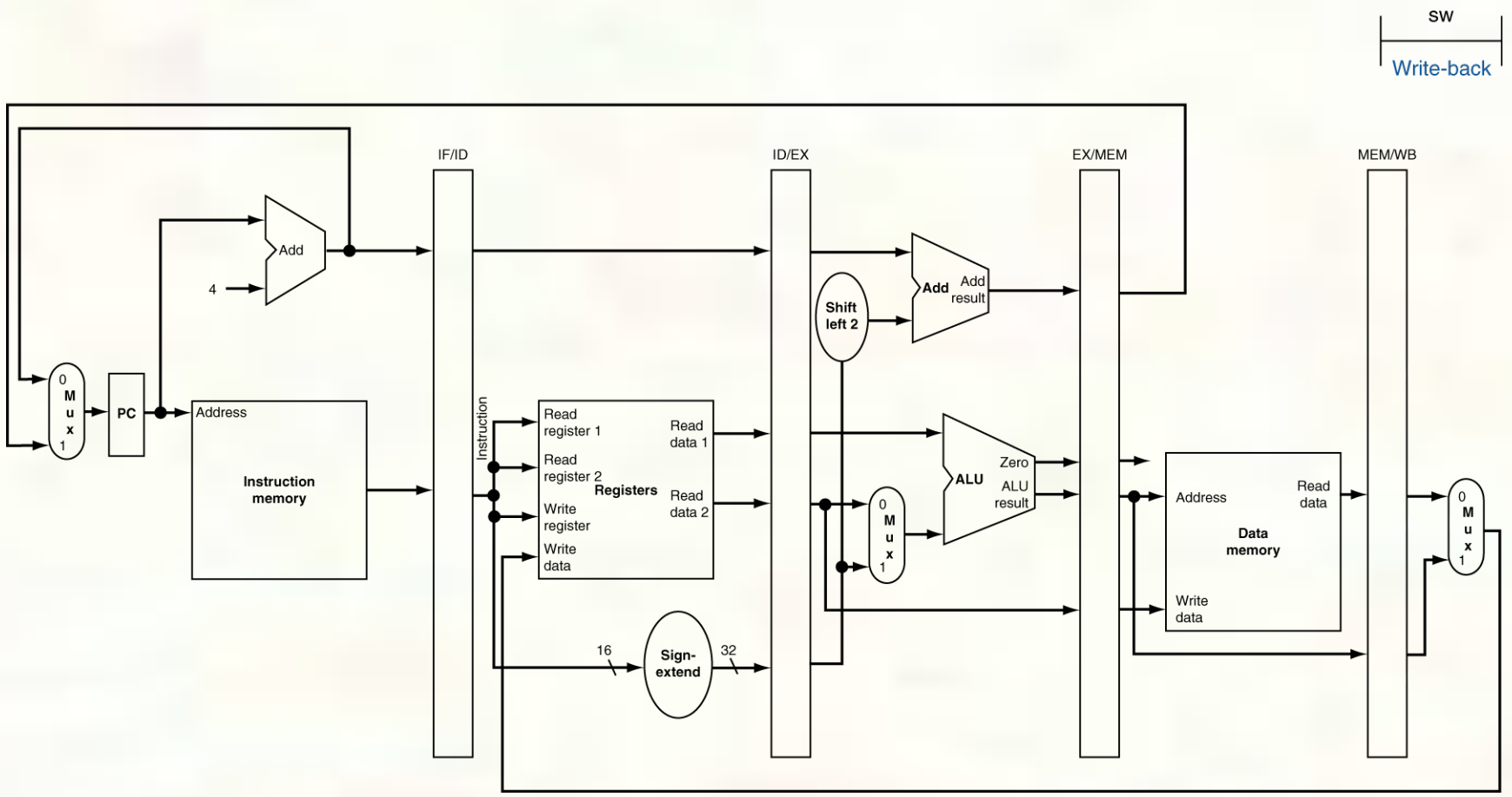
EX for Store

sw
Execution
IF/ID
ID/EX
EX/MEM
MEM/WB
Add
4
M u x
PC
Address
Instruction memory
Instruction
Read register 1
Read register 2
Write register
Write data
Registers
Read data 1
Read data 2
Shift left 2
Add
Add result
ALU
Zero
ALU result
16
Sign-extend
32
Address
Data memory
Read data
Write data

MEM for Store

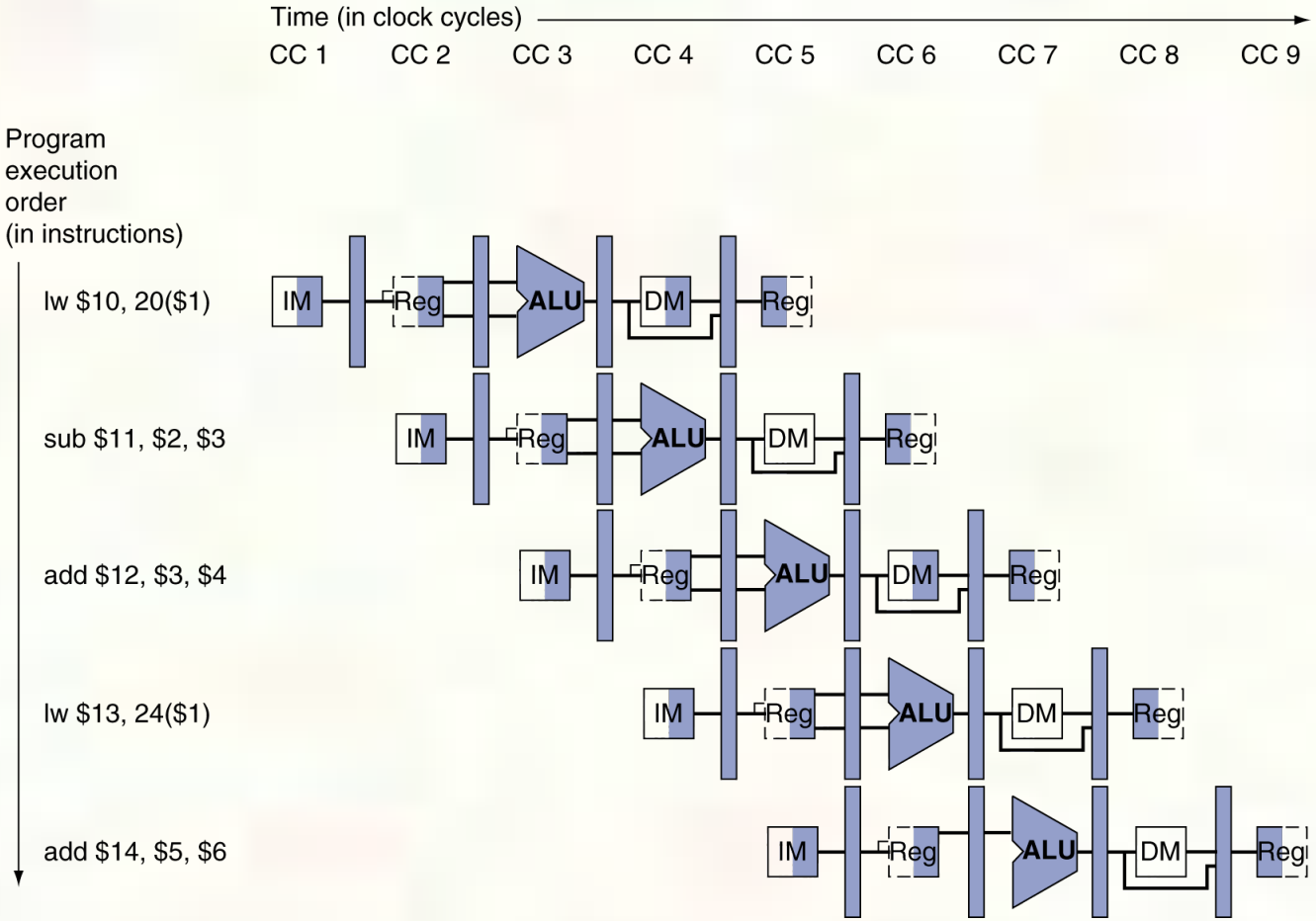


WB for Store


SW
Write-back
IF/ID
ID/EX
EX/MEM
MEM/WB
Add
4
M u x
0
1
PC
Address
Instruction memory
Instruction
Read register 1
Read register 2
Registers
Write register
Write data
Read data 1
Read data 2
16
Sign-extend
32
Shift left 2
Add
Add result
ALU
Zero
ALU result
Address
Read data
Data memory
Write data

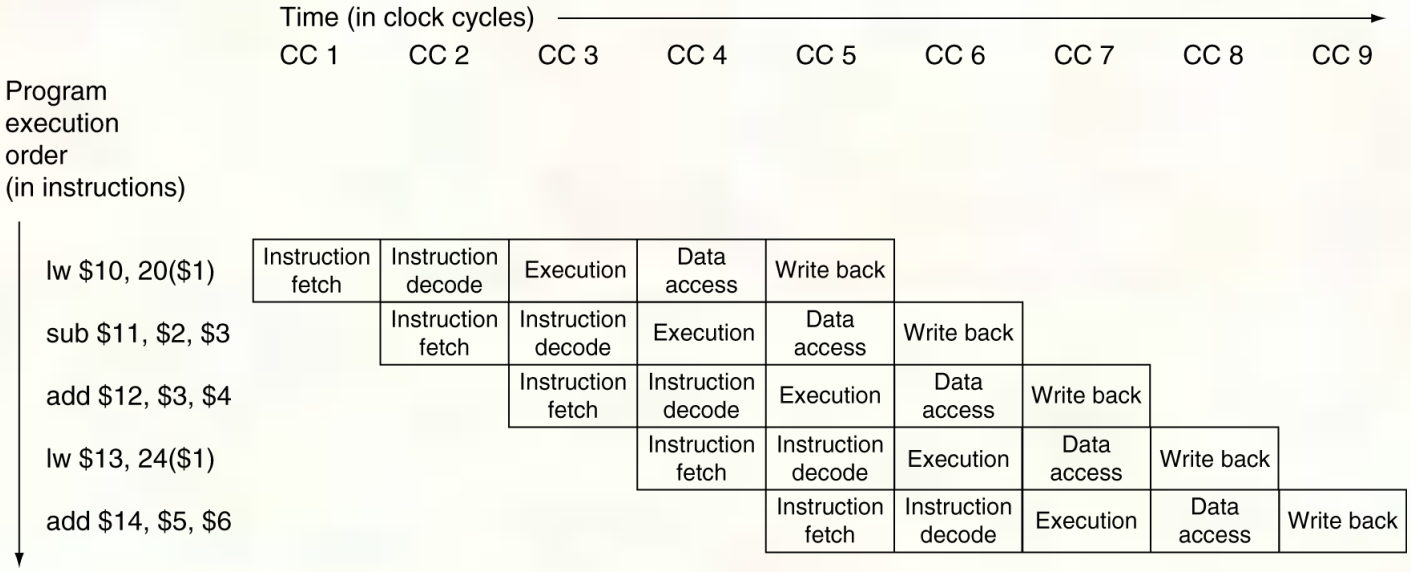
Multi-Cycle Pipeline Diagram

- Form showing resource usage



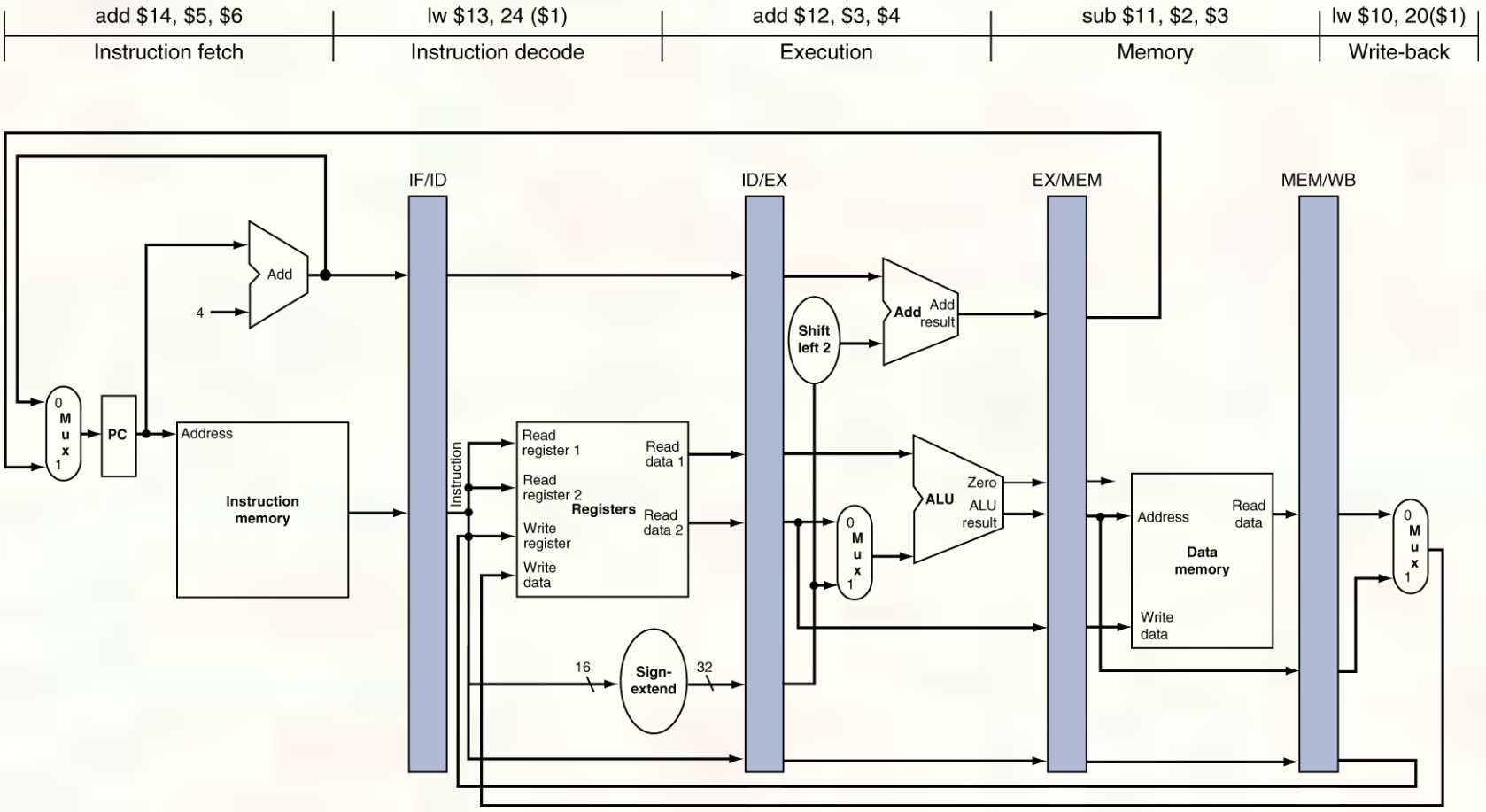
Multi-Cycle Pipeline Diagram

- Traditional form

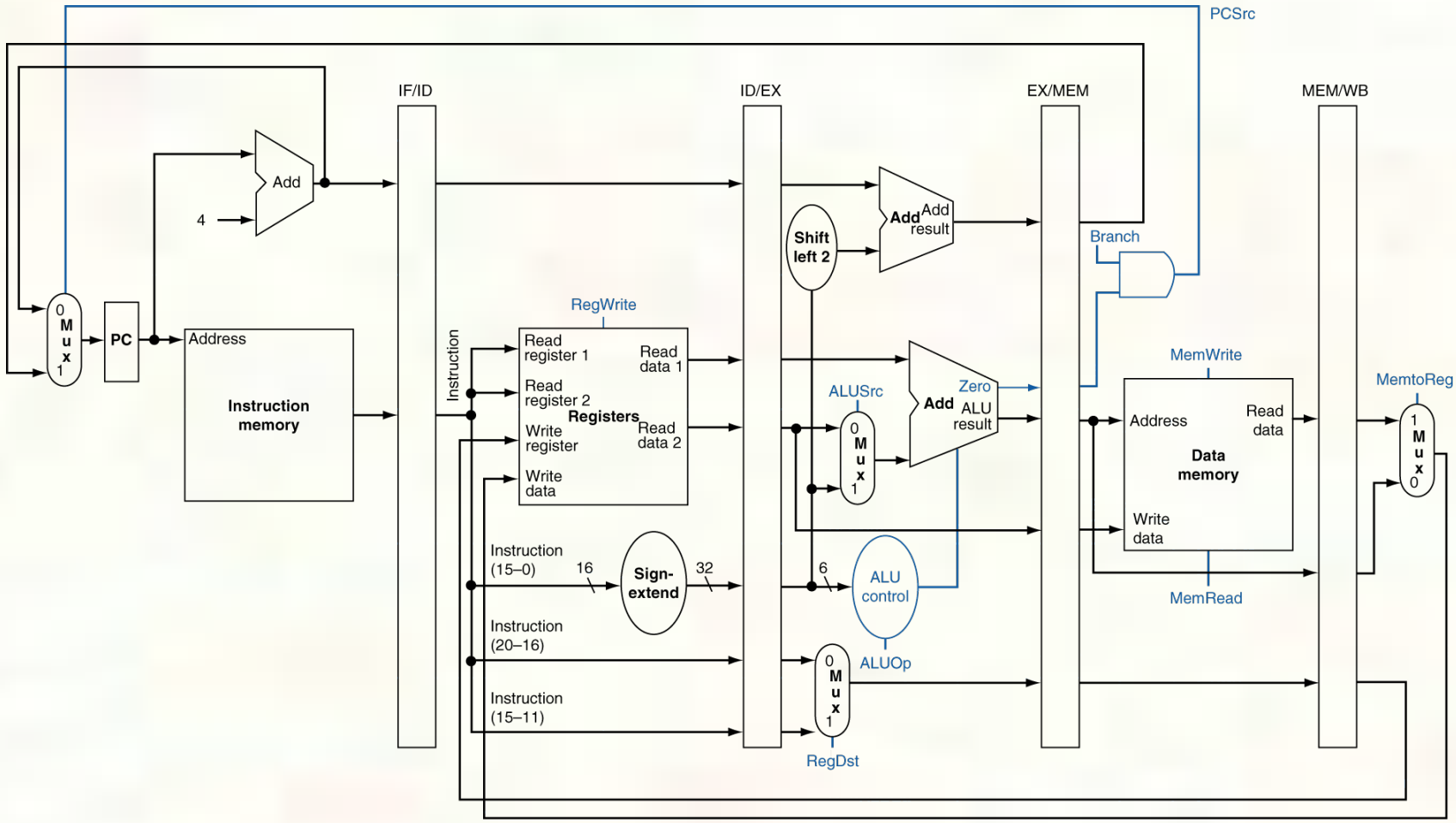


Single-Cycle Pipeline Diagram

- State of pipeline in a given cycle

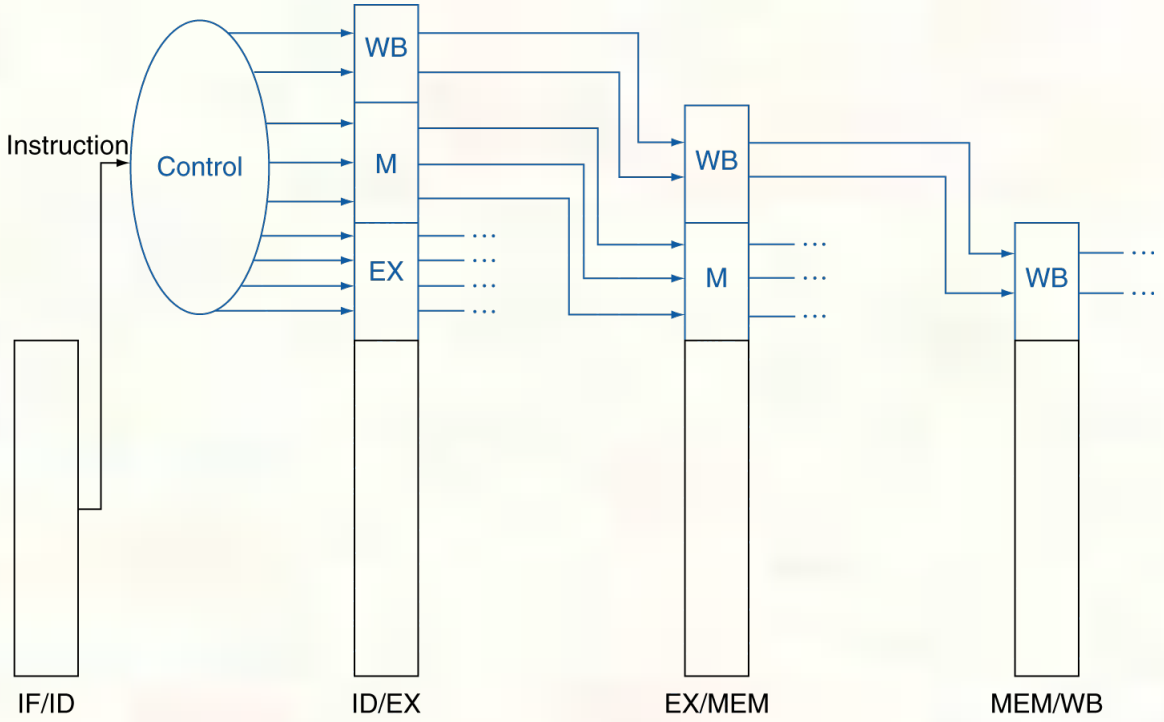


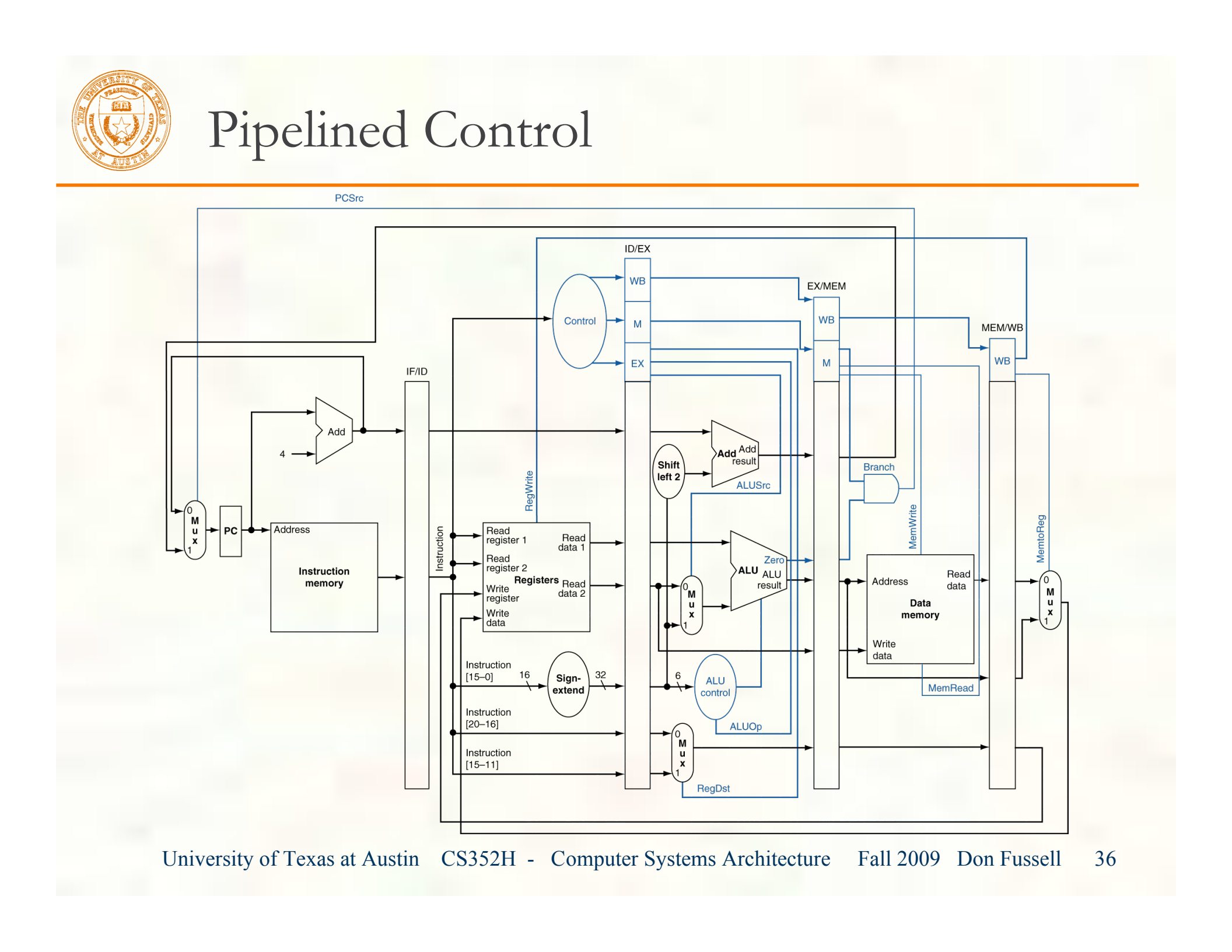
Pipelined Control (Simplified)



Pipelined Control

- Control signals derived from instruction
  - As in single-cycle implementation





Pipelined Control

Concluding Remarks

- 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