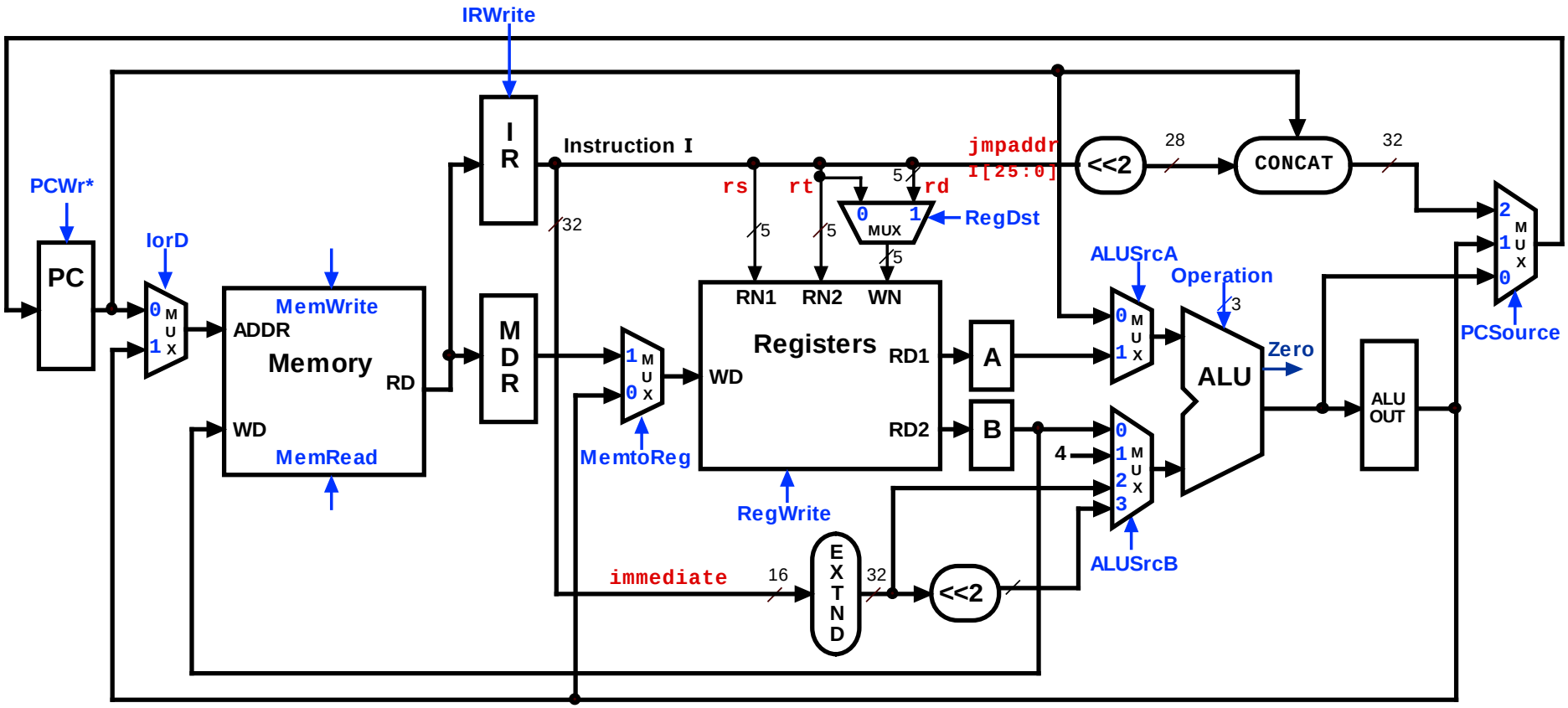
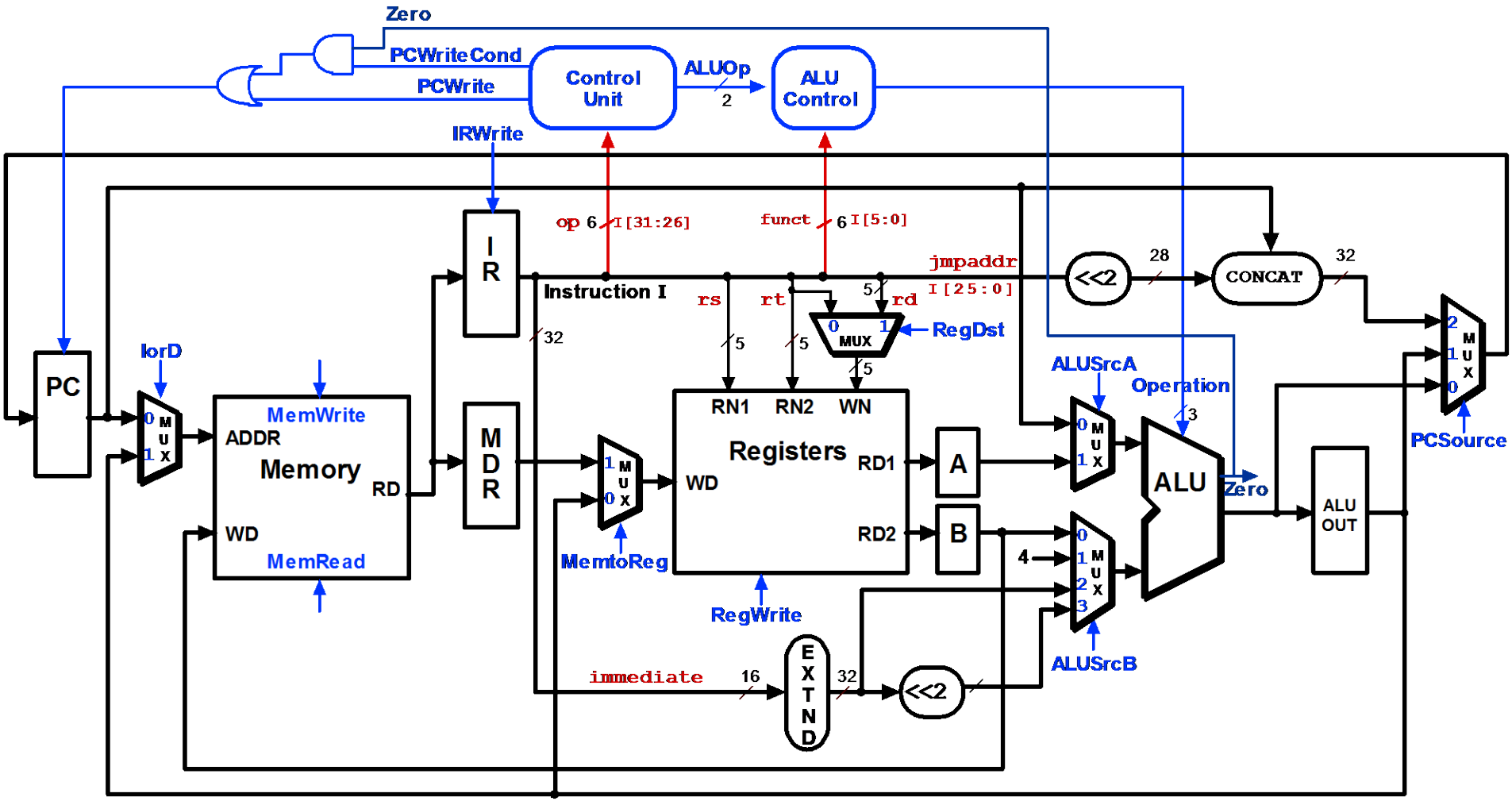
Full Multicycle Datapath



Our new datapath

- We eliminate *both* extra adders in a multicycle datapath, and instead use just one ALU, with multiplexers to select the proper inputs.
- A 2-to-1 mux ALUSrcA sets the first ALU input to be the PC or a register.
- A 4-to-1 mux ALUSrcB selects the second ALU input from among:
  - the register file (for arithmetic operations),
  - a constant 4 (to increment the PC),
  - a sign-extended constant (for effective addresses), and
  - a sign-extended and shifted constant (for branch targets).
- This permits a single ALU to perform all of the necessary functions.
  - Arithmetic operations on two register operands.
  - Incrementing the PC.
  - Computing effective addresses for lw and sw.
  - Adding a sign-extended, shifted offset to (PC + 4) for branches.

Full Multicycle Implementation



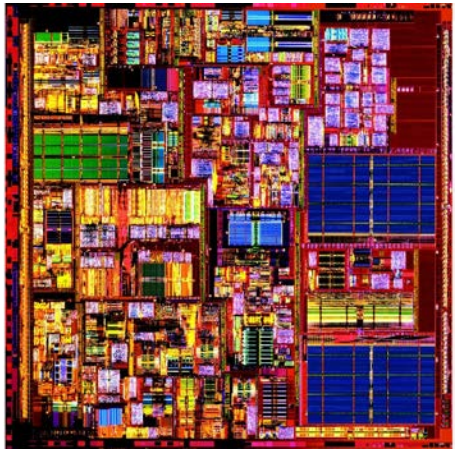
Historical Perspective

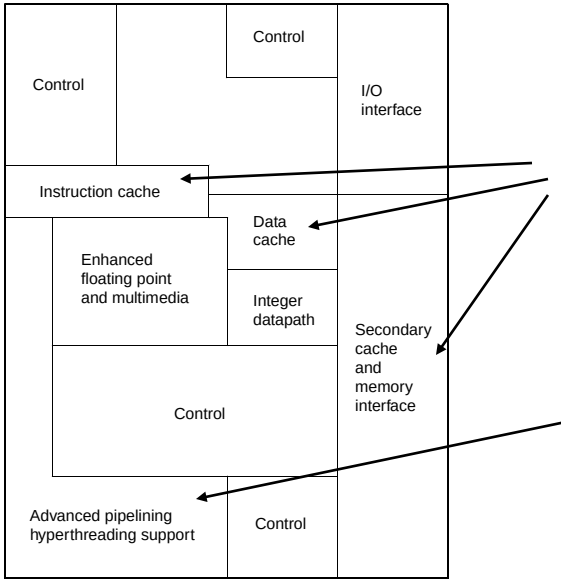
- In the ‘60s and ‘70s microprogramming was very important for implementing machines
- This led to more sophisticated ISAs and the VAX
- In the ‘80s RISC processors based on pipelining became popular
- Pipelining the microinstructions is also possible!
- Implementations of IA-32 architecture processors since 486 use:
  - “hardwired control” for simpler instructions
    (few cycles, FSM control implemented using PLA)
  - “microcoded control” for more complex instructions
    (large numbers of cycles, central control store)

- The IA-64 architecture uses a RISC-style ISA and can be implemented without a large central control store

Pentium 4

- Pipelining is important (last IA-32 without it was 80386 in 1985)





Chapter 5

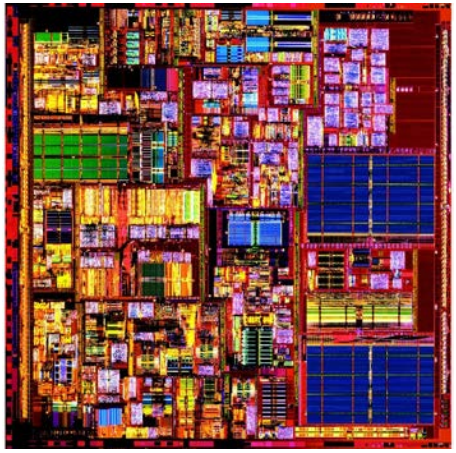
Chapters 3 and 4

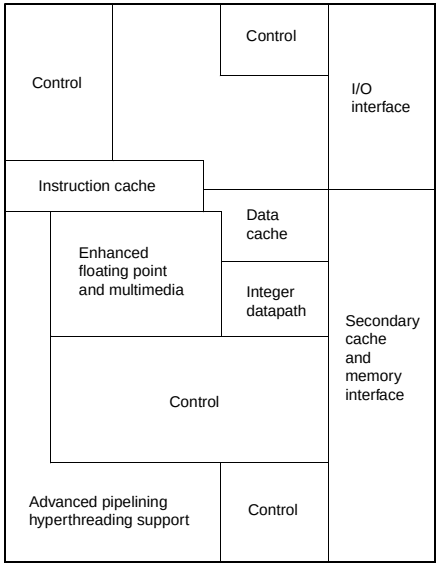
- Pipelining is used for the simple instructions favored by compilers

"Simply put, a high performance implementation needs to ensure that the simple instructions execute quickly, and that the burden of the complexities of the instruction set penalize the complex, less frequently used, instructions"

Pentium 4

- Somewhere in all that “control” we must handle complex instructions





- Processor executes simple microinstructions, 70 bits wide (hardwired)
- 120 control lines for integer datapath (400 for floating point)
- If an instruction requires more than 4 microinstructions to implement, control from microcode ROM (8000 microinstructions)
- Its complicated!

Summary (1)

- A single-cycle CPU has two main disadvantages.
  - The cycle time is limited by the worst case latency.
  - It requires more hardware than necessary.
- A multicycle processor splits instruction execution into several stages.
  - Instructions only execute as many stages as required.
  - Each stage is relatively simple, so the clock cycle time is reduced.
  - Functional units can be reused on different cycles.
- We made several modifications to the single-cycle datapath.
  - The two extra adders and one memory were removed.
  - Multiplexers were inserted so the ALU and memory can be used for different purposes in different execution stages.
  - New registers are needed to store intermediate results.

Summary (2)

- If we understand the instructions…

  We can build a simple processor!

- If instructions take different amounts of time, multi-cycle is better
- Datapath implemented using:
  - Combinational logic for arithmetic
  - State holding elements to remember bits
- Control implemented using:
  - Combinational logic for single-cycle implementation
  - Finite state machine for multi-cycle implementation

Pipeline: Introduction

These slides are adapted from notes by Dr. David Patterson (UCB)

What is Pipelining?

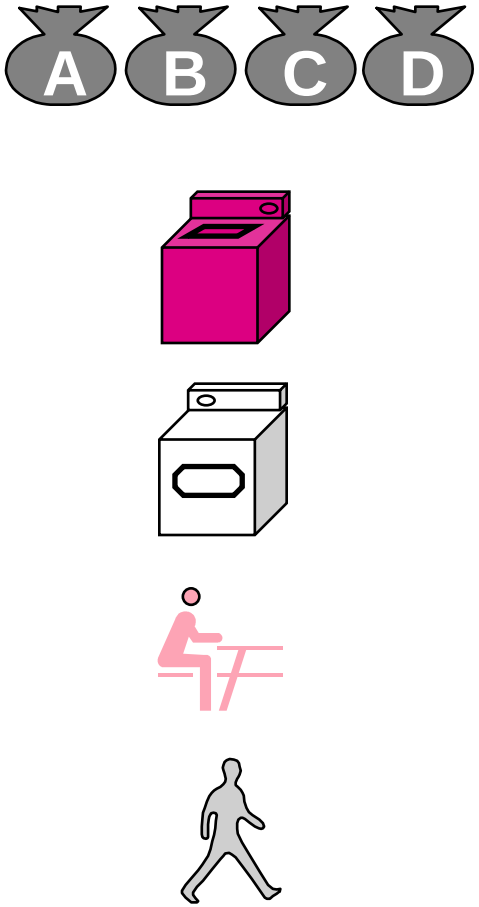
- A way of speeding up execution of instructions

- *Key idea*:

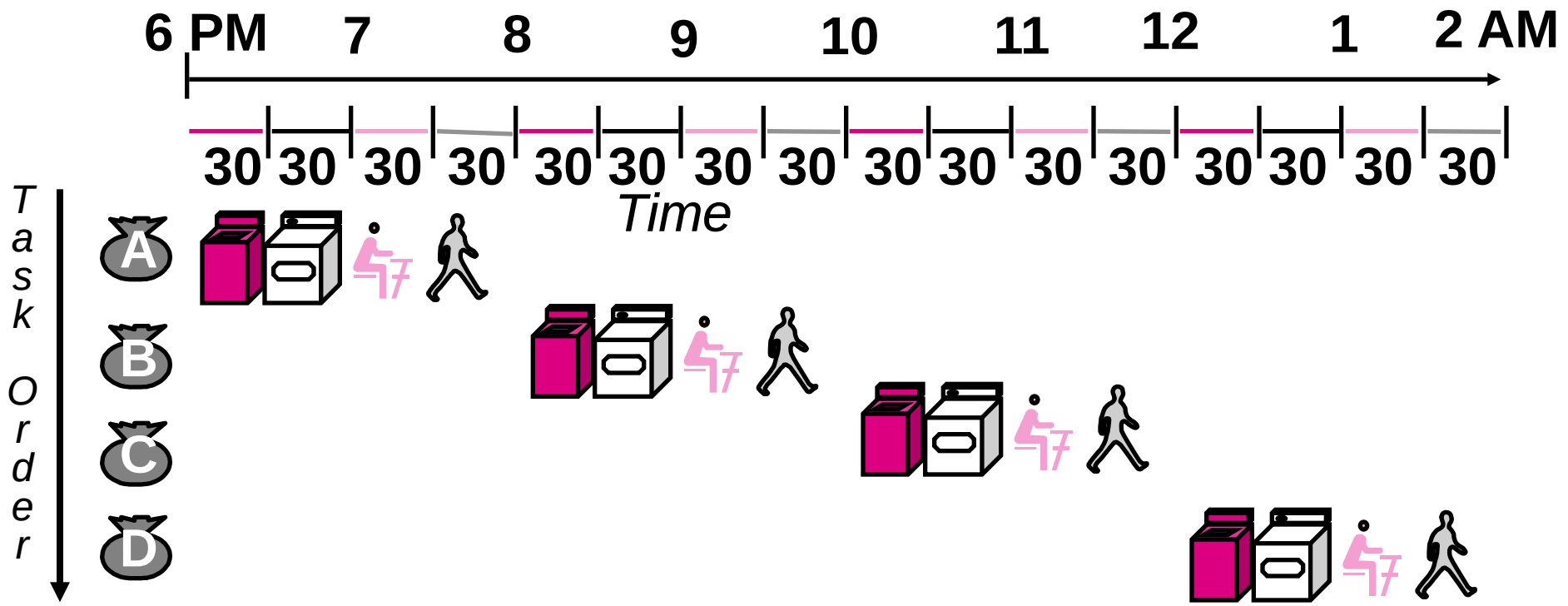
overlap execution of multiple instructions

The Laundry Analogy

- Anna, Brian, Cathy, Dave each have one load of clothes to wash, dry, and fold
- Washer takes 30 minutes
- Dryer takes 30 minutes
- “Folder” takes 30 minutes
- “Stasher” takes 30 minutes to put clothes into drawers

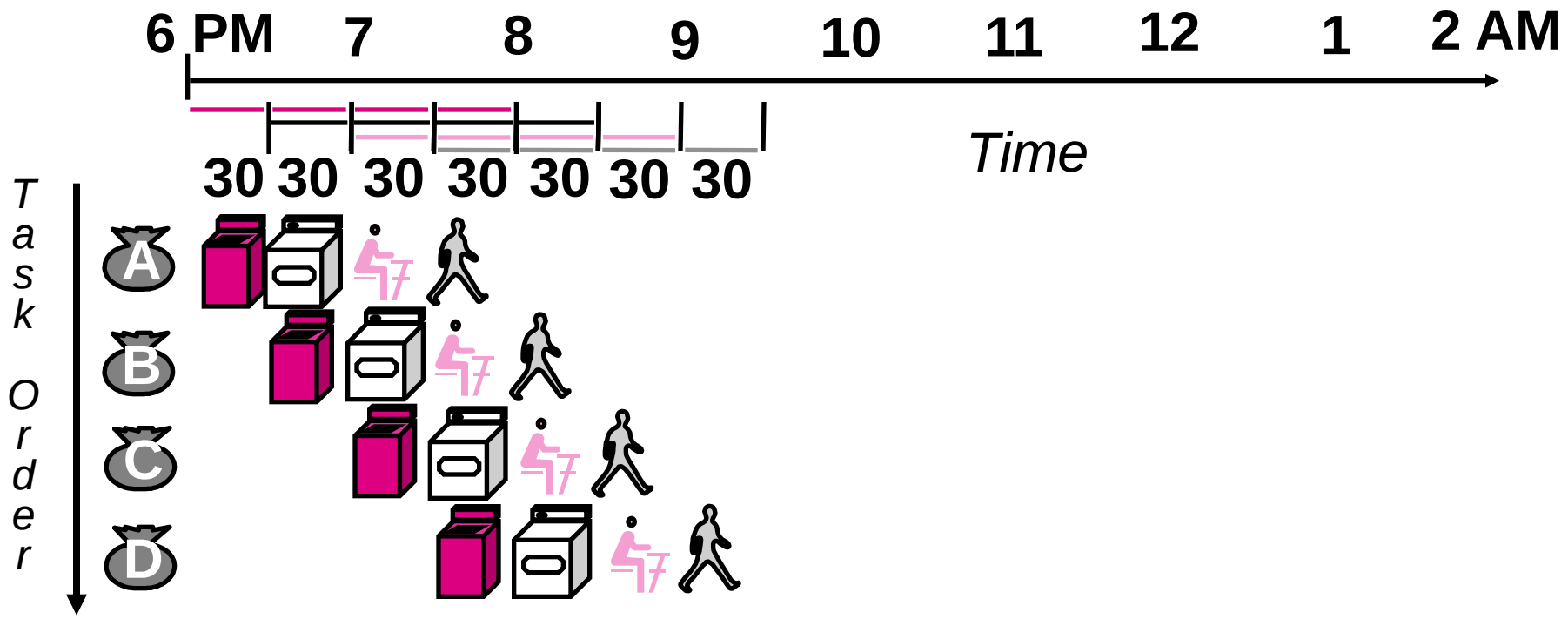


If we do laundry sequentially...



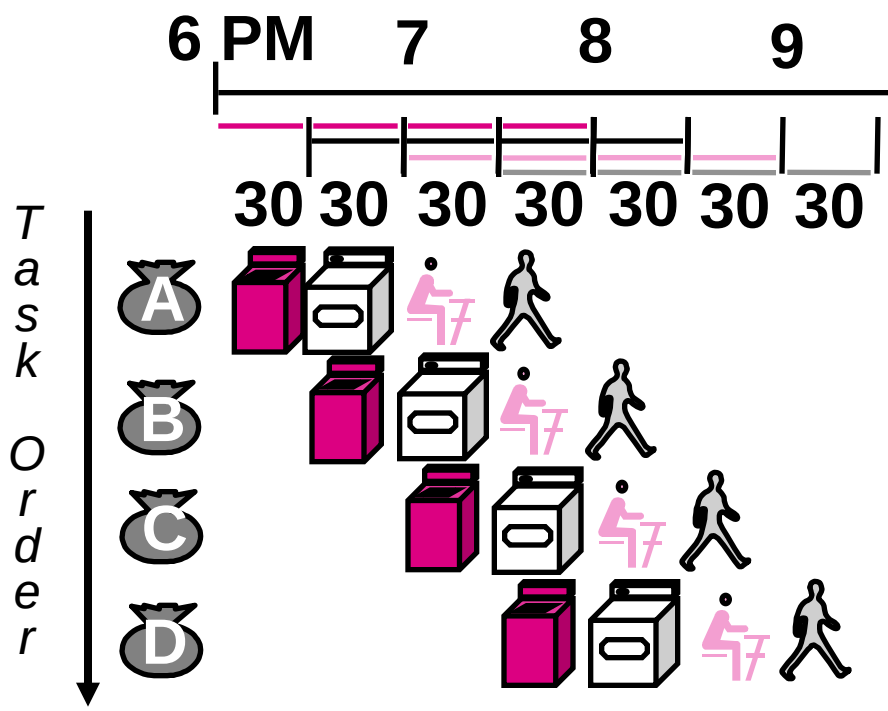
- Time Required: 8 hours for 4 loads

To Pipeline, We Overlap Tasks



- Time Required: 3.5 Hours for 4 Loads

To Pipeline, We Overlap Tasks

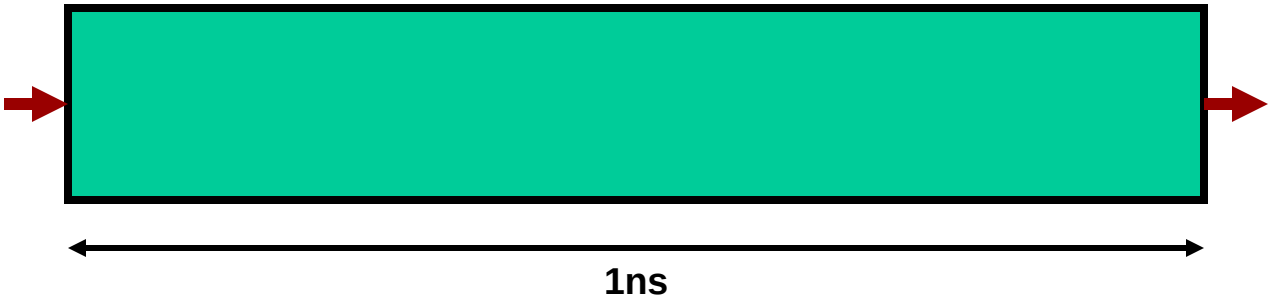


- Does Pipelining help latency of single task? No
- Does Pipelining help throughput of entire workload? Yes
- Pipeline rate limited by ___? the slowest pipeline stage
- Multiple tasks operating simultaneously
- Potential speedup = ? Number of pipe stage
- Unbalanced lengths of pipe stages will ___ reduces speedup
- Time to "fill" pipeline and time to "drain" it reduces speedup

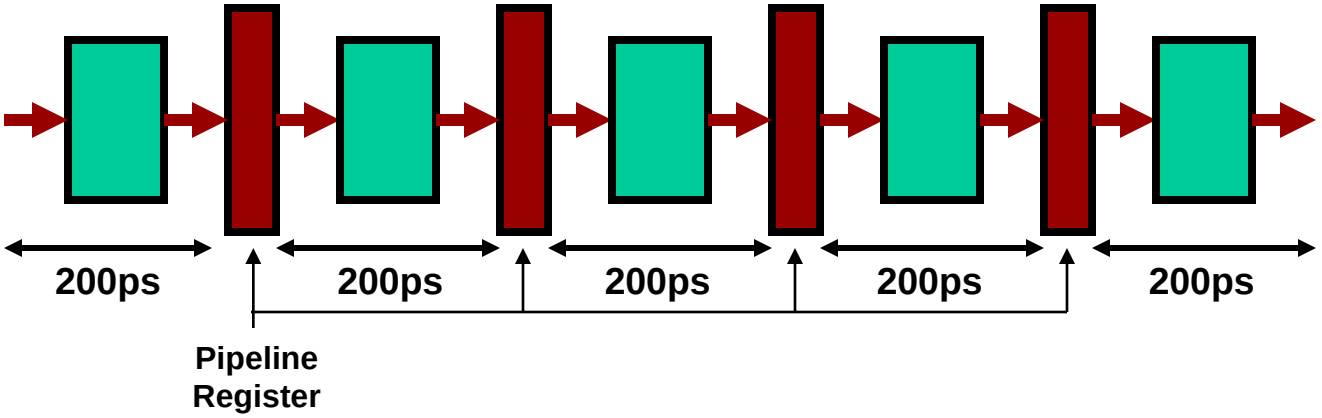
Pipelining a Digital System

1 nanosecond = 10^-9 second
1 picosecond = 10^-12 second

- Key idea: break big computation up into pieces

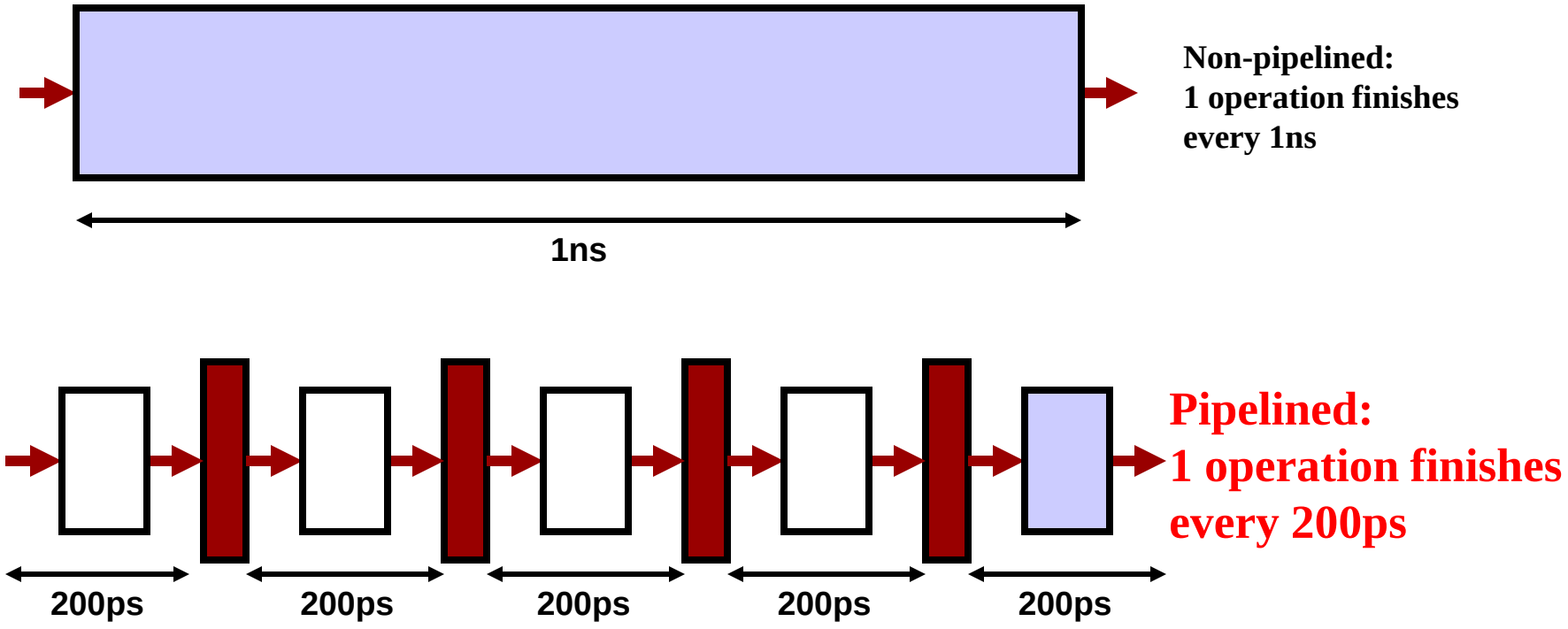


Separate each piece with a pipeline register



Pipelining a Digital System

- Why do this? Because it's faster for repeated computations



Comments about pipelining

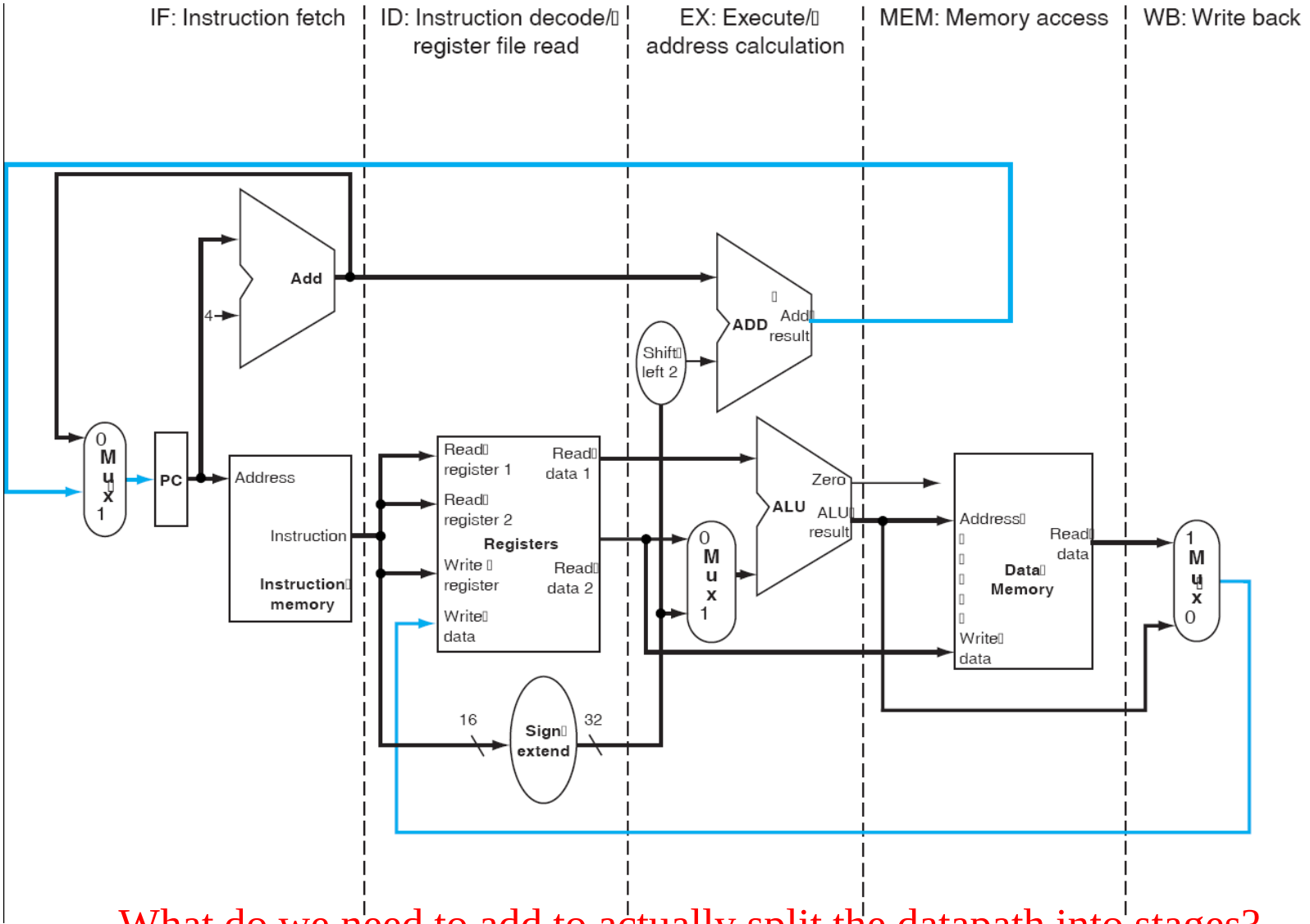
- Pipelining increases throughput, but not latency
  - Answer available every 200ps, BUT
  - A single computation still takes 1ns
- Limitations:
  - Computations must be divisible into stage size
  - ? Pipeline registers add overhead

Pipelining a Processor

- Recall the 5 steps in instruction execution:
  1. Instruction Fetch (IF)
  2. Instruction Decode and Register Read (ID)
  3. Execution operation or calculate address (EX)
  4. Memory access (MEM)
  5. Write result into register (WB)

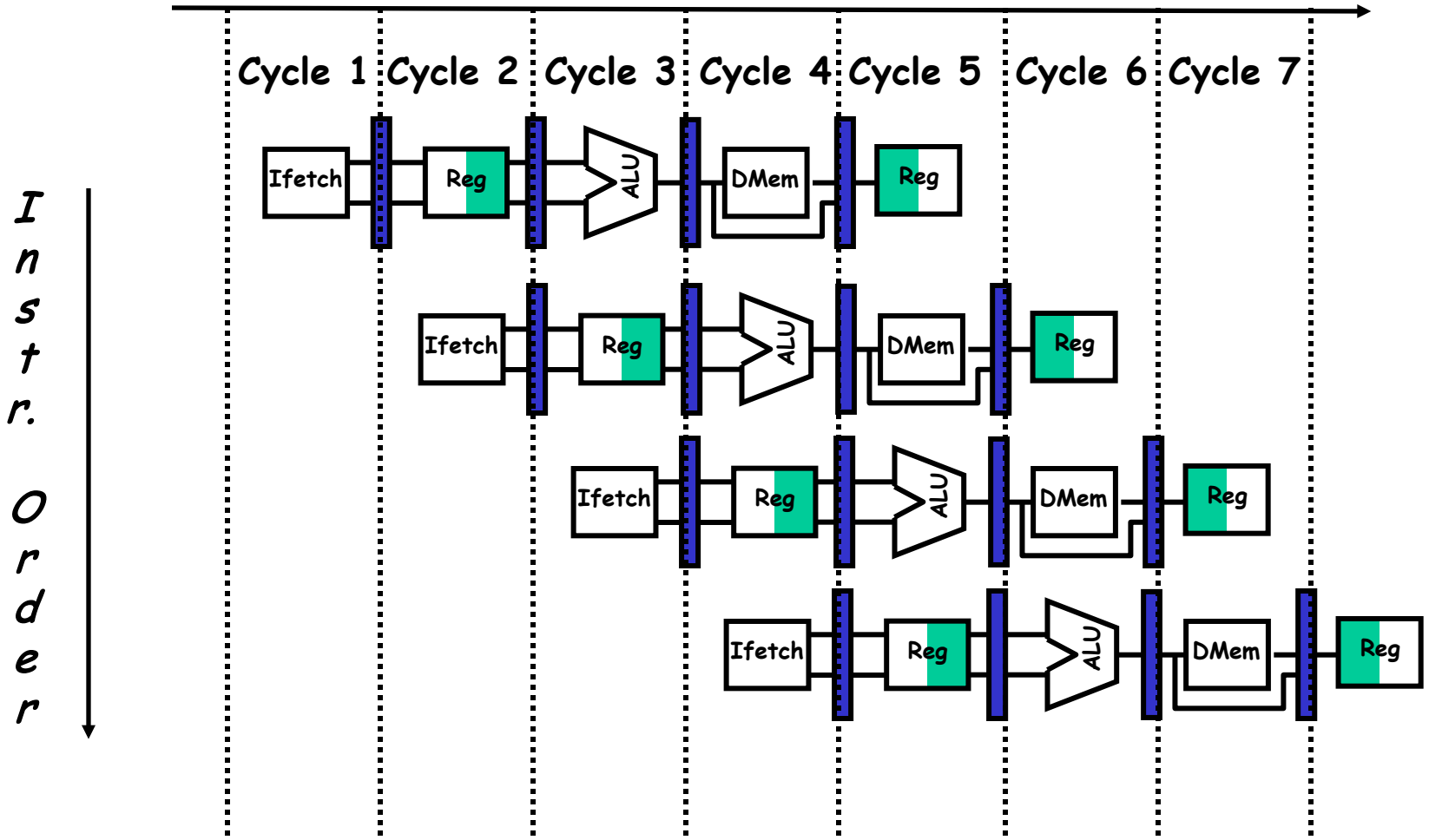
- Review: Single-Cycle Processor
  - All 5 steps done in a single clock cycle
  - Dedicated hardware required for each step

Review - Single-Cycle Processor

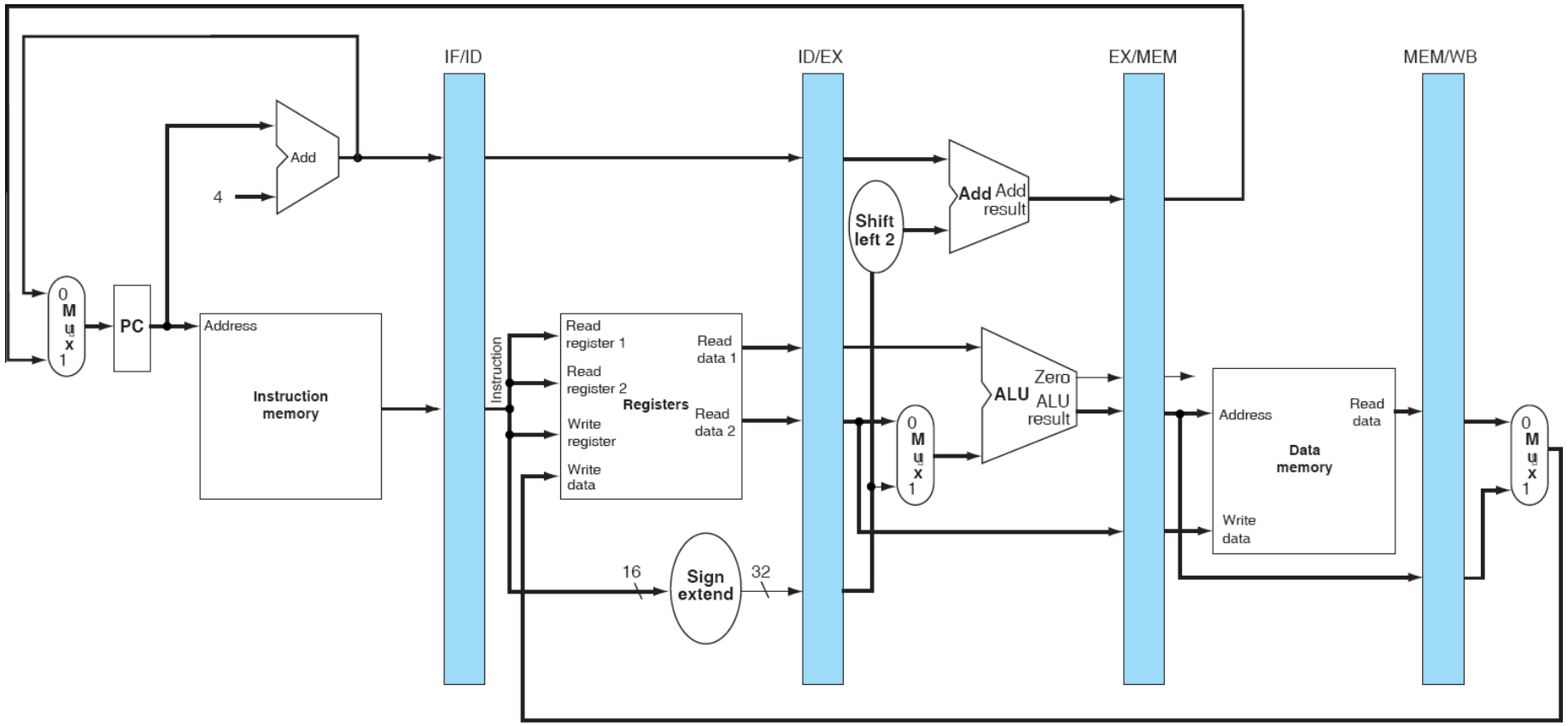


What do we need to add to actually split the datapath into stages?

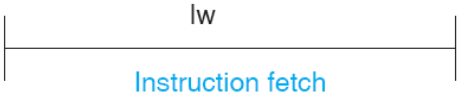
The Basic Pipeline For MIPS



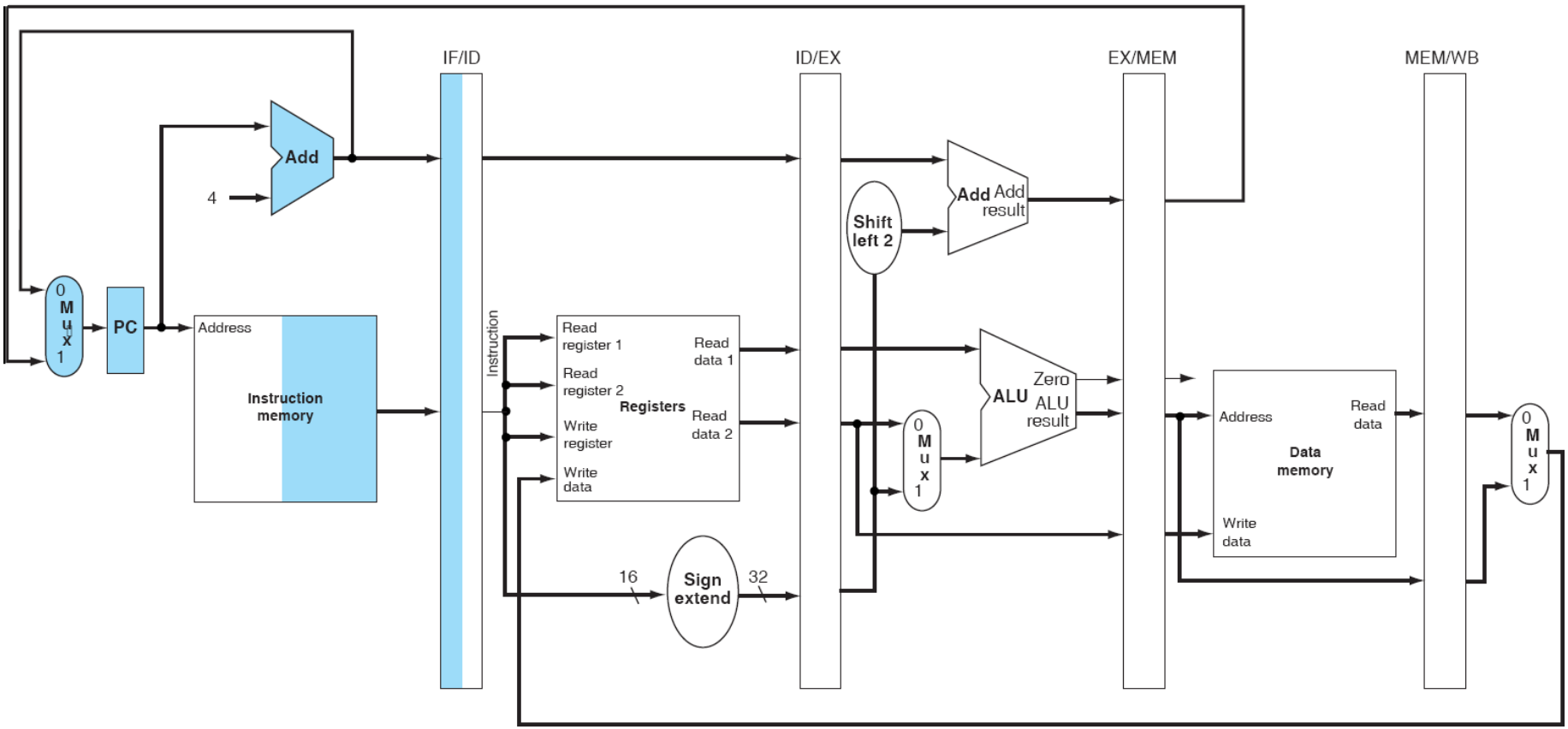
Basic Pipelined Processor



Pipeline example: lw

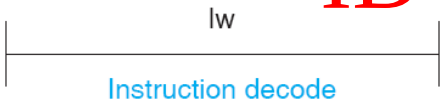


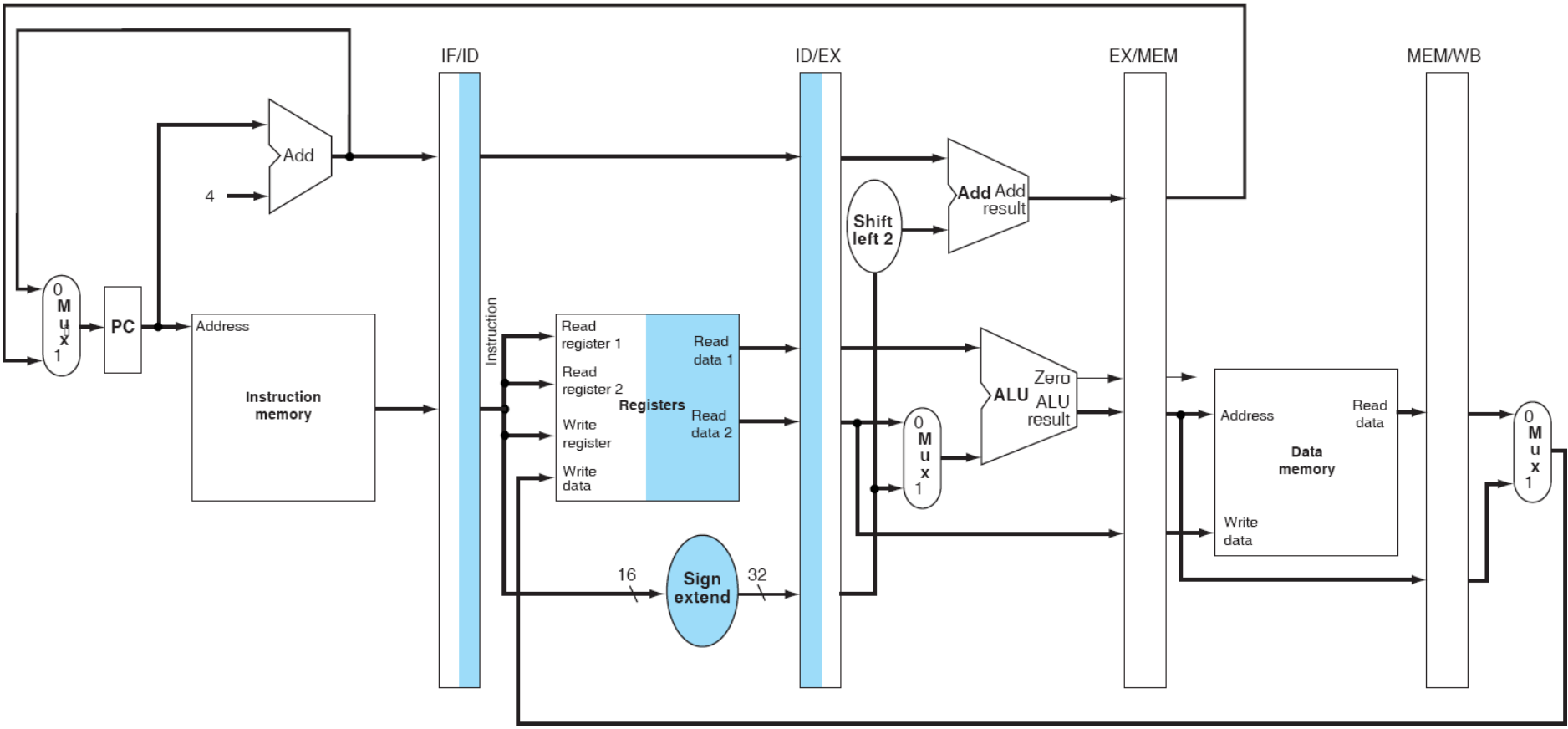
IF



Pipeline example: lw

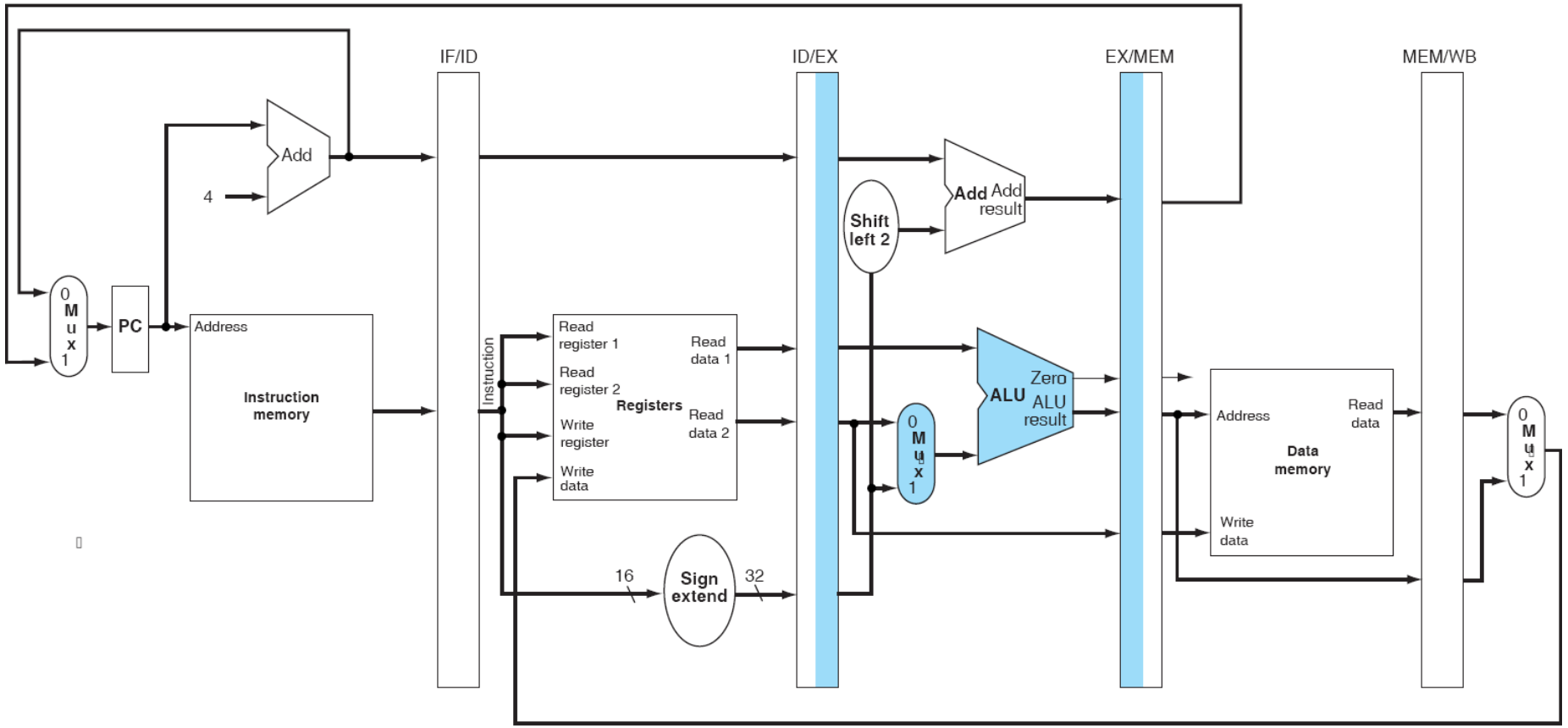
ID





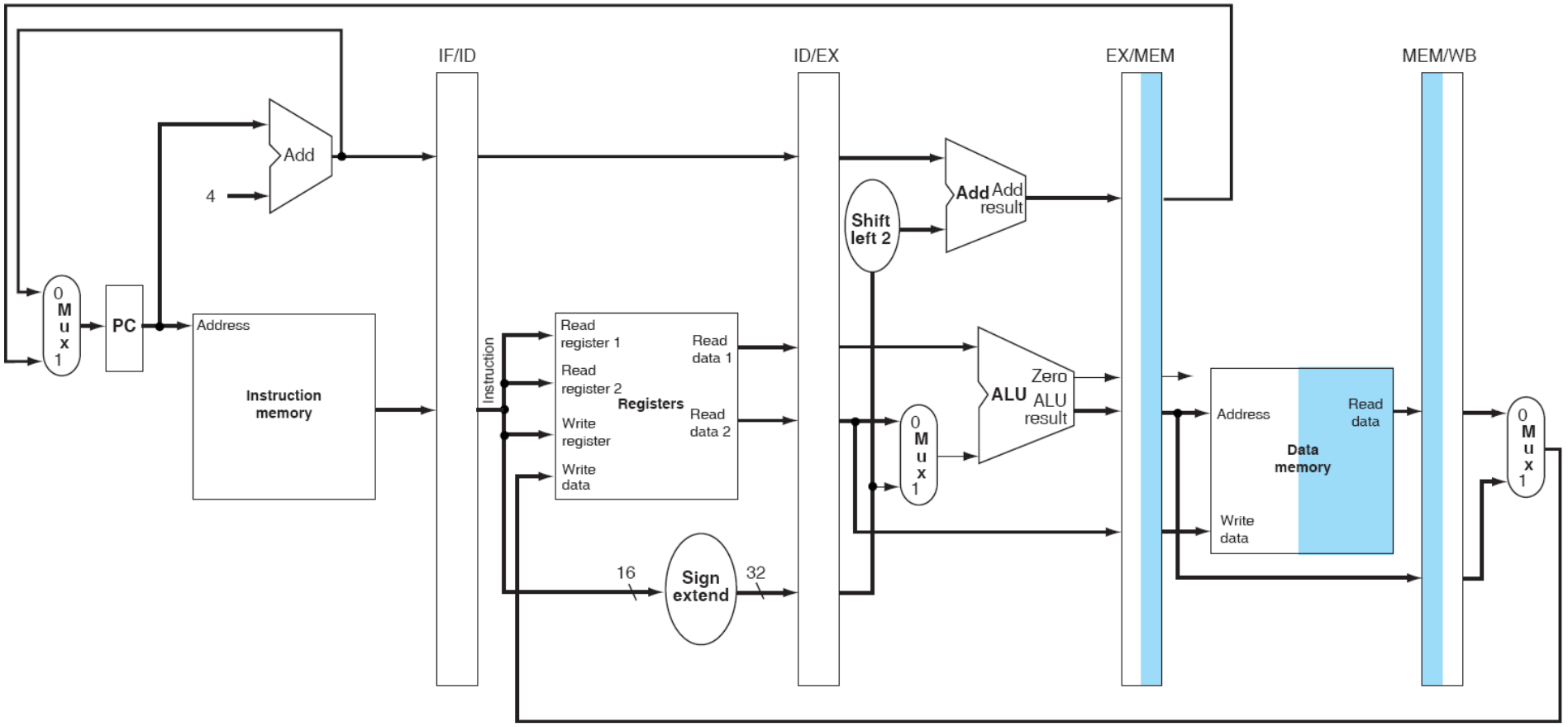
Pipeline example: lw

EX



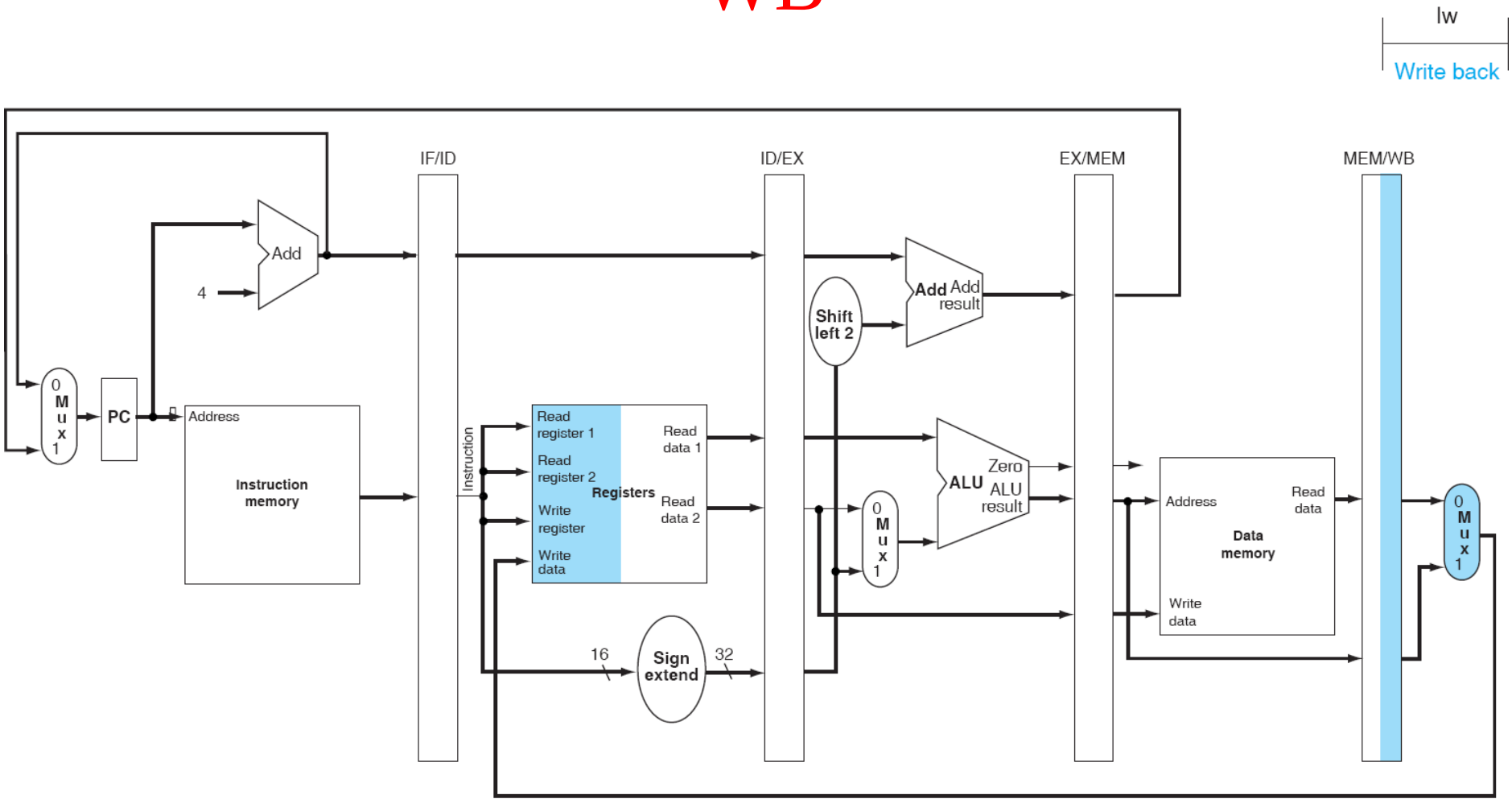
Pipeline example: lw

MEM



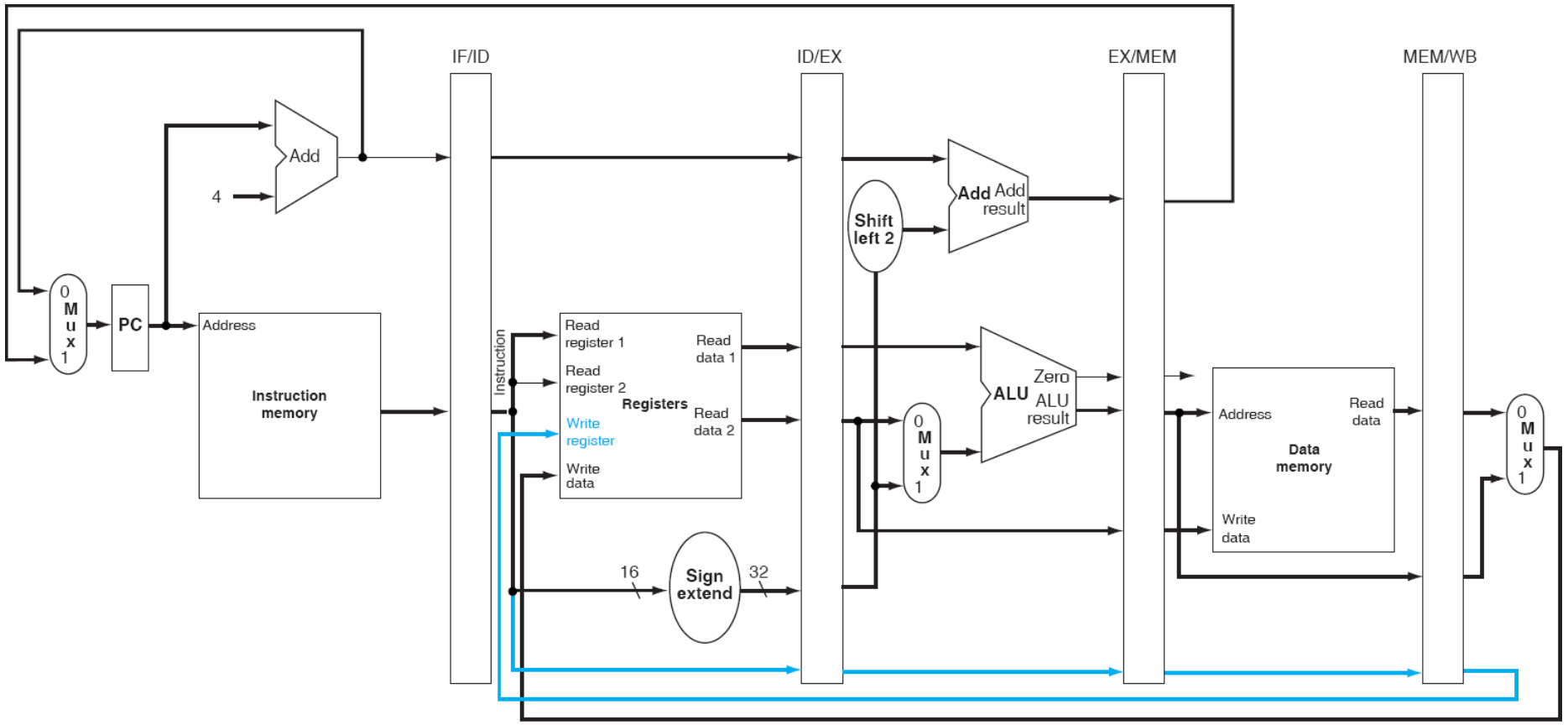
Pipeline example: lw

WB

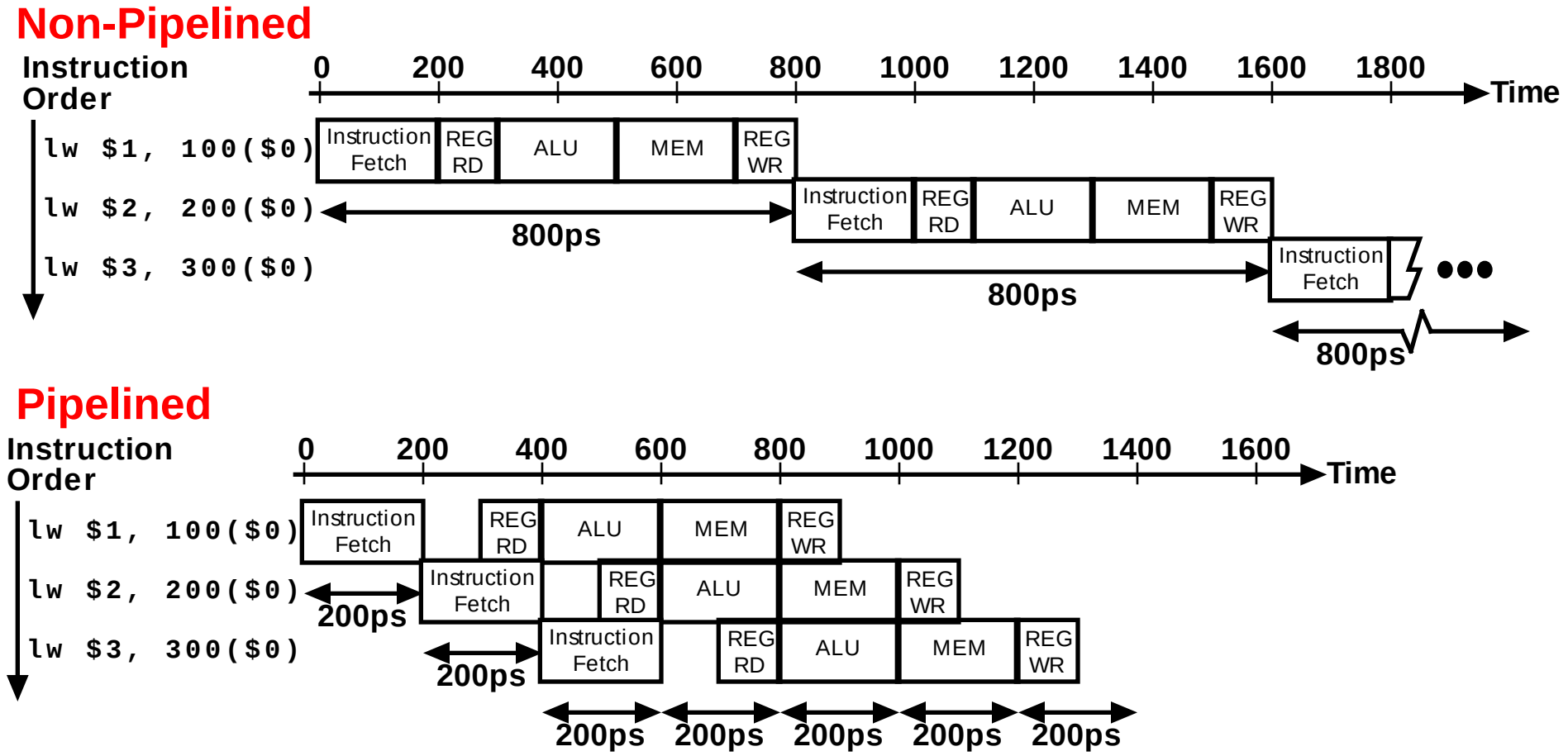


Can you find a problem?

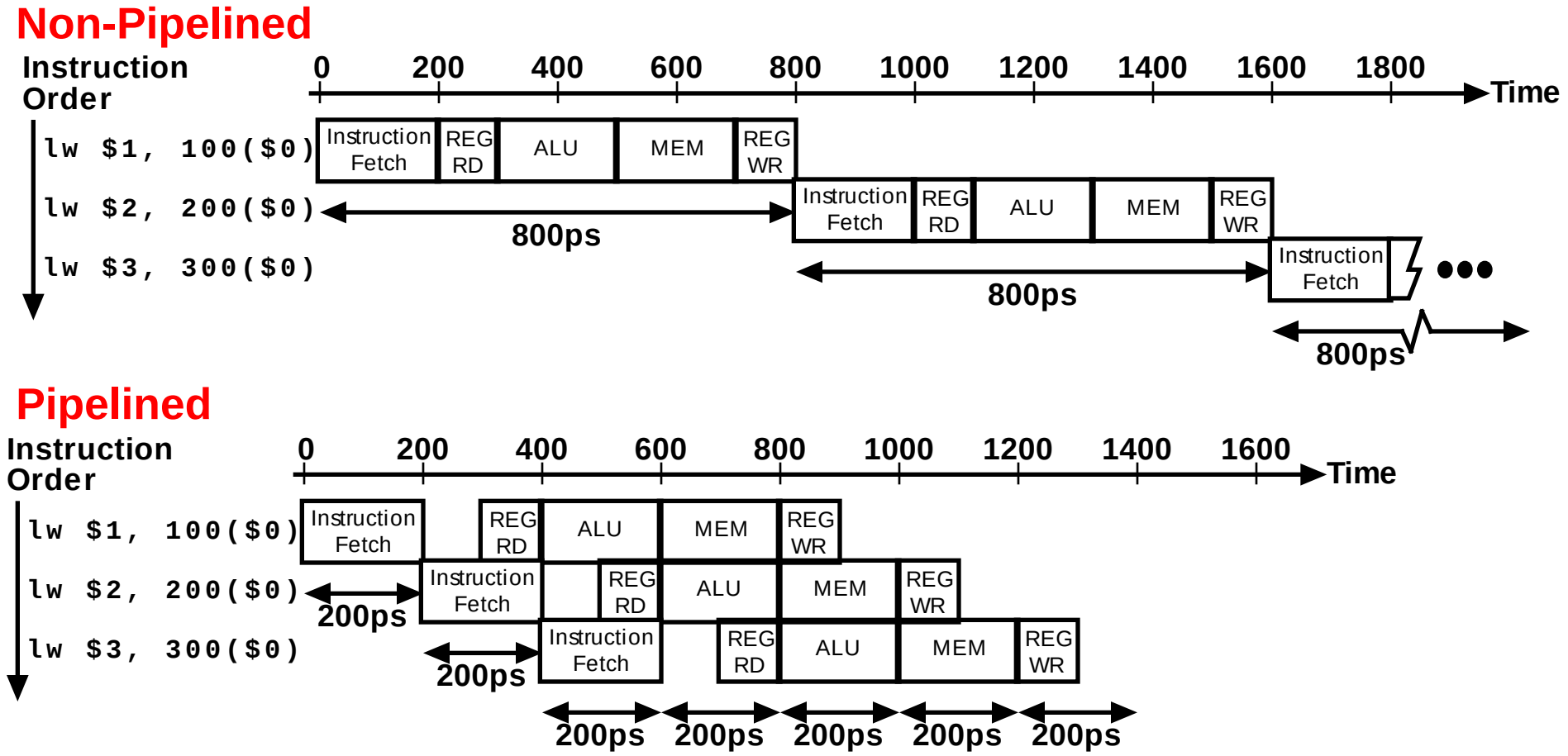
Basic Pipelined Processor (Corrected)



Single-Cycle vs. Pipelined Execution



Single-Cycle vs. Pipelined Execution



Pipelined datapath