# CEng 3420 Computer Organization & Design Lecture 09: Pipeline

Textbook: Chapter 4.6-4.7

Zhengrong Wang

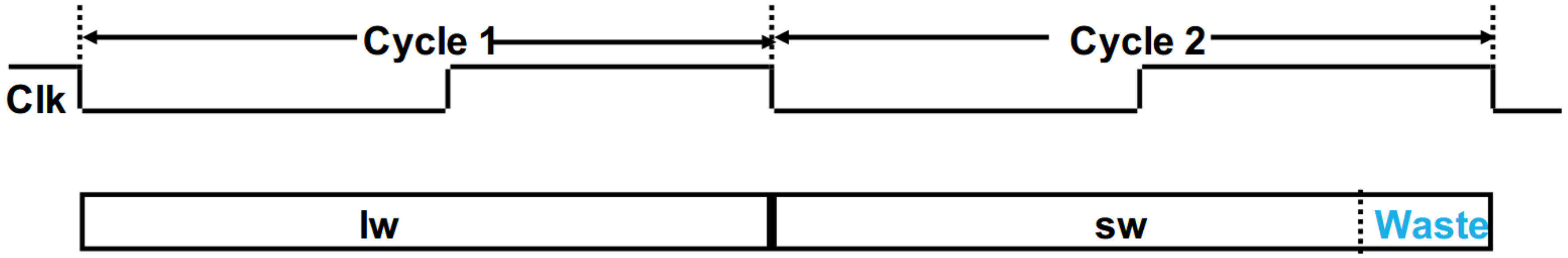
CSE Department, CUHK

zhengrongwang@cuhk.edu.hk

## Motivation

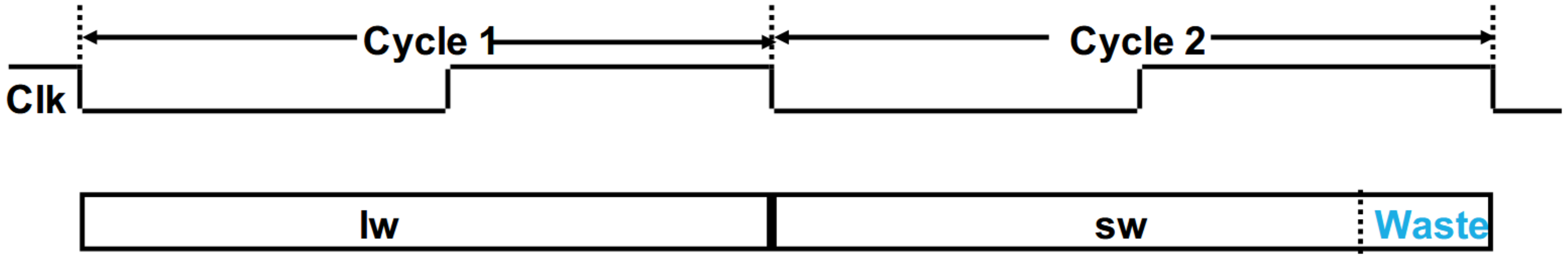
## Single Cycle Disadvantages

- Single cycle: the whole datapath is finished in one clock cycle.
- It is simple and easy to understand.
- Uses the clock cycle inefficiently – must be timed to accommodate the slowest instruction.
- Problematic for more complex instructions like floating point multiply.
- May be wasteful of area since some functional units (e.g., adders) must be duplicated since they can not be shared during a clock cycle.



# Single Cycle Disadvantages

- Though simple, the single cycle approach is not used because it is very slow.
- Clock cycle must have the same length for every instruction.
- What is the longest path (slowest instruction)? Load instruction!
- It is too long for the store so the last part of the cycle here is wasted.



# Partition Datapath into Stages

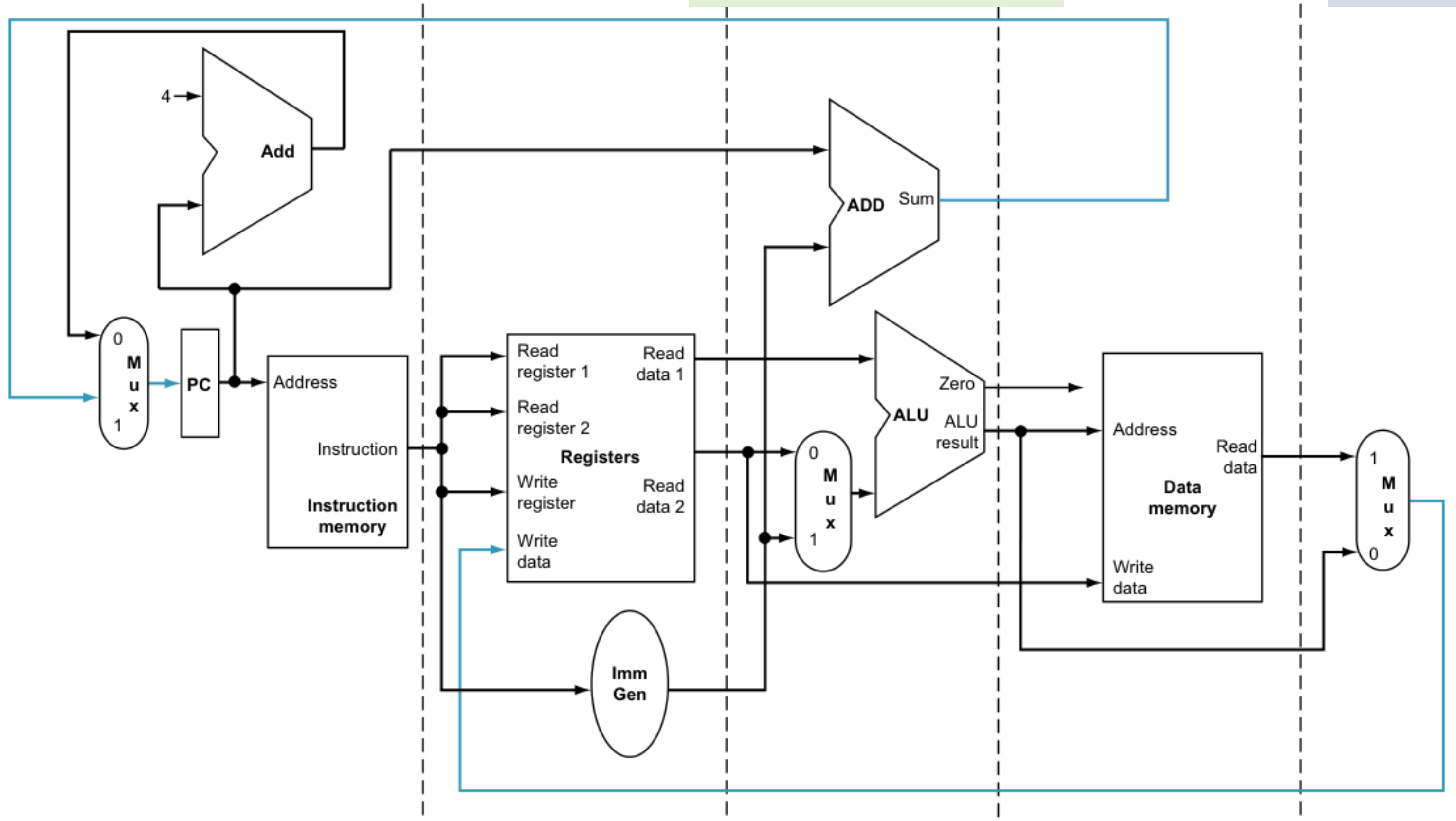
Stage 2 ID: Instruction decode/ register file read.

Stage 4 MEM: Access memory.

Stage 1 IF: Instruction fetch.

Stage 3 EX: Execute/address calculation.

Stage 5 WB: Write back to register file.



# Instruction Critical Path

- Calculate cycle time assuming negligible delays (for muxes, control unit, sign extend, PC access, shift left 2, wires) except:
  - Instruction fetch and update PC (IF) (4ns).
  - Register fetch and instruction decode (ID) (1ns).
  - Execute R-type; calculate memory address (EXE) (2 ns).
  - Read/write data from/to data memory (MEM) (4 ns)
  - Write the result data into the register file (WB) (1 ns).

| Inst | IF | ID | EXE | MEM | WB | Total |
|---|---|---|---|---|---|---|
| R-Type | 4 | 1 | 2 |  | 1 | 8 |
| lw | 4 | 1 | 2 | 4 | 1 | 12 |
| sw | 4 | 1 | 2 | 4 |  | 11 |
| beq | 4 | 1 | 2 |  |  | 7 |

## How Can We Make It Faster?

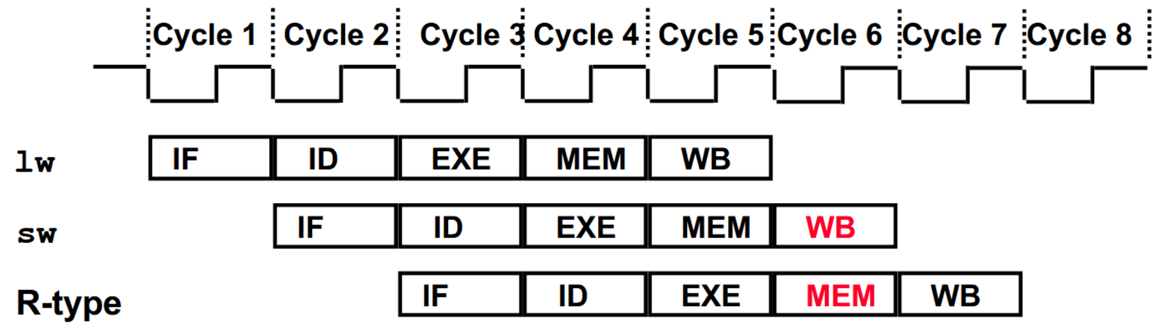
$$\text{CPU time} = \text{CPI} \times \text{CC} \times \text{IC}$$

- Start fetching and executing the next instruction before the current one has completed.
  - Pipelining – (all?) modern processors are pipelined for performance.
  - Under ideal conditions and with many instructions, the speedup from pipelining is approximately equal to the number of pipe stages.
  - A five-stage pipeline is nearly five times faster because the CC is "nearly" five times faster.
- Fetch (and execute) more than one instruction at a time.
  - Superscalar processing – stay tuned.

## Pipeline Basis

# A Pipelined RISC-V Processor

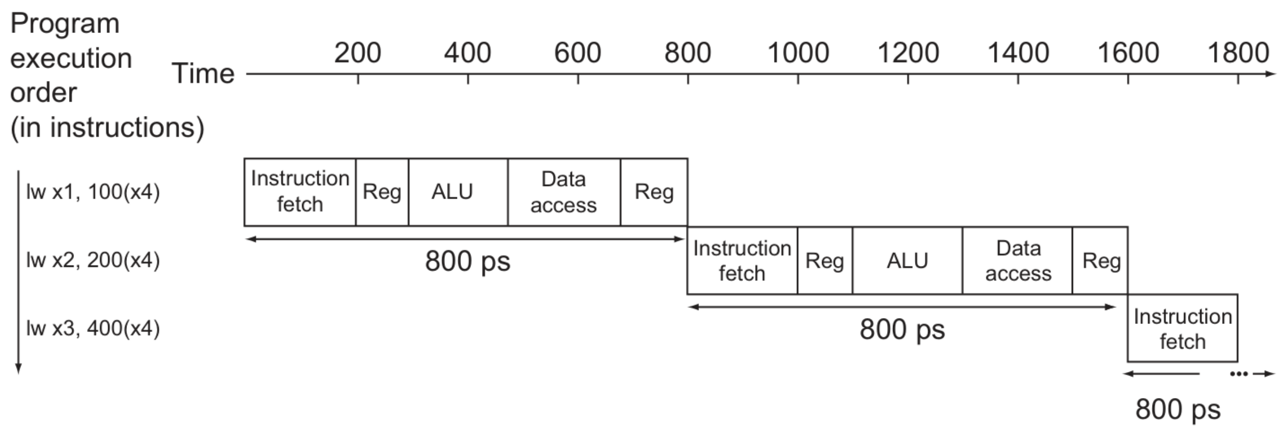
- Start the next instruction before the current one has completed.
  - Improves throughput - total amount of work done in a given time.
  - instruction latency (execution time, delay time, response time - time from the start of an instruction to its completion) is not reduced

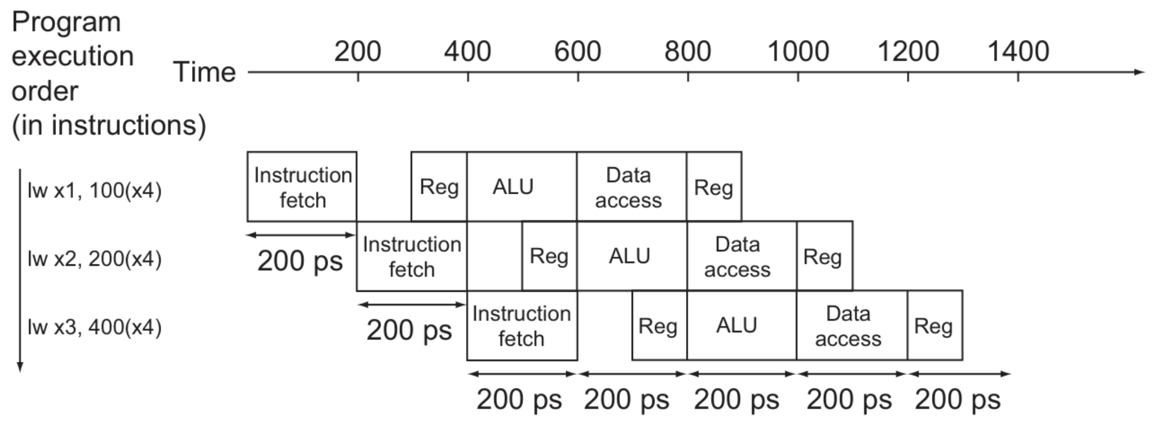


  - Clock cycle (pipeline stage time) is limited by the slowest stage.
  - For some stages don't need the whole clock cycle (e.g., WB is usually fast).
  - For some instructions, some stages are wasted cycles (i.e., nothing is done during that cycle for that instruction).

## Single Cycle vs. Pipelined Design

- To complete an entire instruction in the pipelined case takes 1000 ps (as compared to 800 ps for the single cycle case). Why ?
- How long does each take to complete 1,000,000 adds ?



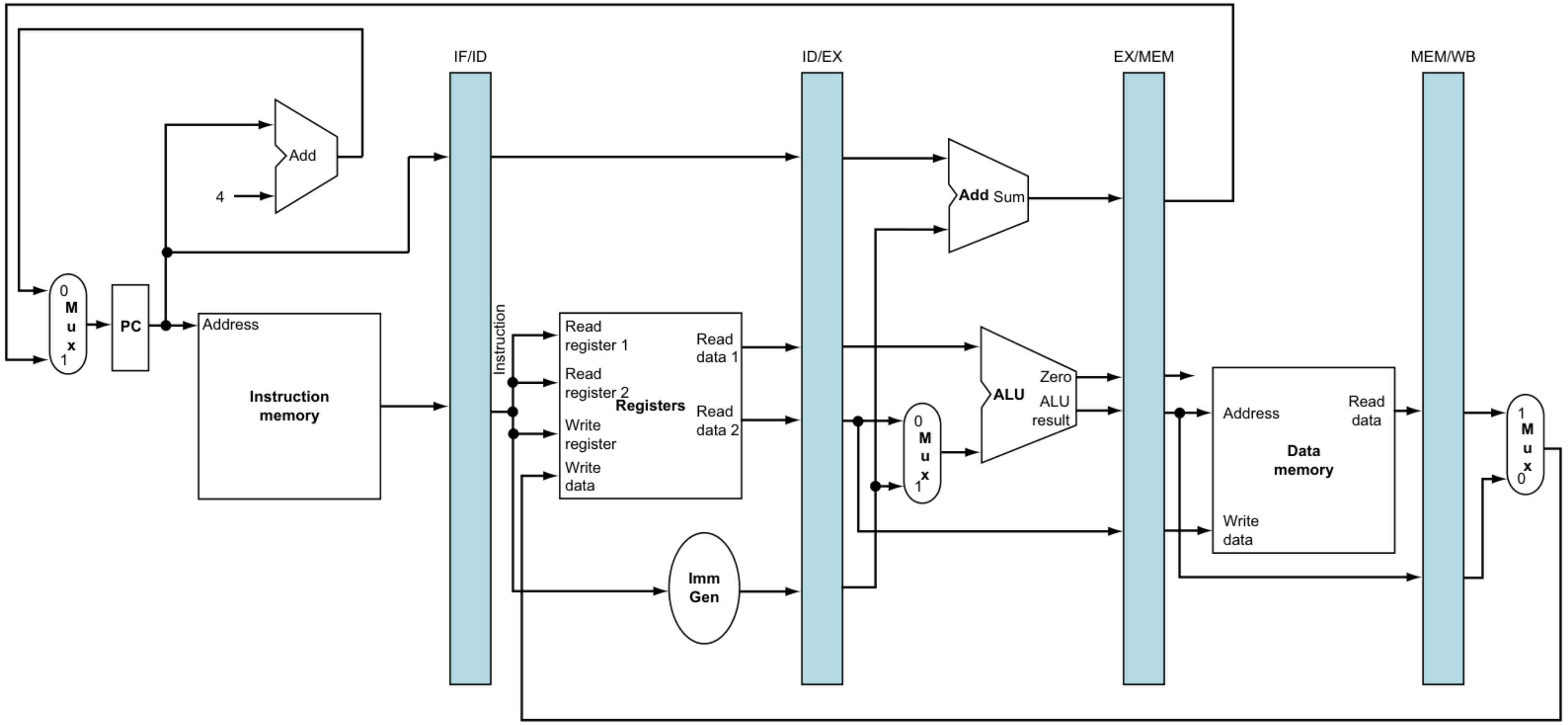


## Pipelining RISC-V

- What makes it easy?
  - All instructions are the same length (32 bits).
    - can fetch in the 1st stage and decode in the 2nd stage.
  - Few instruction formats (three) with symmetry across formats.
    - can begin reading register file in 2nd stage.
  - Memory operations occur only in loads and stores.
    - can use the execute stage to calculate memory addresses.
  - Each instruction writes at most one result (i.e., changes the machine state) and does it in the last few pipeline stages (MEM or WB).
  - Operands must be aligned in memory so a single data transfer takes only one data memory access.

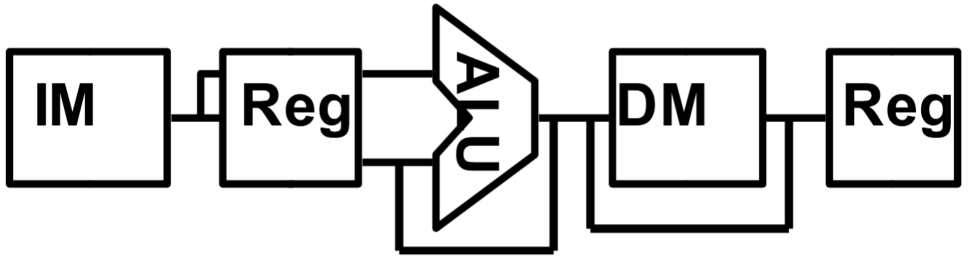
# Pipeline Datapath Addition

- Add state registers between each pipeline stages to isolate them.



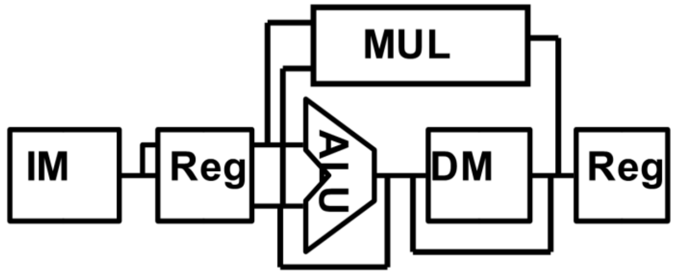
## Graphical Representation of the Pipeline

- Can help with answering questions like:
  - How many cycles does it take to execute this code?
  - What is the ALU doing during cycle 4?
  - Is there a hazard, why does it occur, and how can it be fixed?

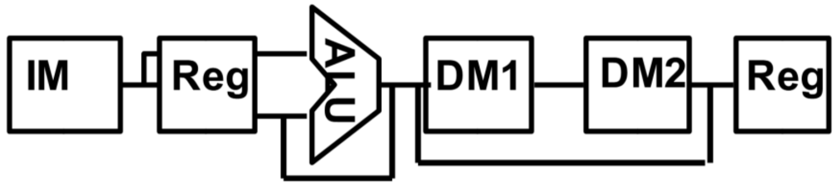


# Other Pipeline Structures are Possible

- What about the (slow) multiply operation?
  - Make the clock twice as slow or ...
  - Let it take two cycles (since it doesn't use the MEM stage)



- What if the data memory access is twice as slow as the instruction memory?
  - Make the clock twice as slow or ...
  - Let data memory access take two cycles (and keep the same clock rate)



# Modern Processor Pipeline

- Deep: usually more than 10 stages to reduce cycle time → higher frequency.
  - Fetch, decode, rename/dispatch, schedule/issue, execute, memory, retire.
  - Each stage can be further divided into multiple sub-stage and takes multiple cycles.
- Wide: every stage can process 2-8 instructions per cycle → higher throughput.
  - A balanced design requires every stage has roughly same throughput.
- Out-of-Order: execute instruction once resources are ready → less latency.
  - Avoids pipeline stall when prior instructions are not ready, e.g., waiting for load result.
  - Dispatch and retire are still in-order to maintain the sequential program order.
  - Issue, execute, and memory are out-of-order.
- E.g., Intel Core, Apple A-series, Apple M-series, ...
- Higher performance, but at the cost of more area and power.
  - Heterogenous design in cellphone SoC: Large core (gaming, etc.) + small core.

## Structural Hazards

## Can Pipelining Get Us Into Trouble?

- Yes! Pipeline Hazards.
  - Structural hazards: a required resource is busy.
  - Data hazards: a required data is not ready yet (from memory or prior instruction).
  - Control hazards: deciding on control action depends on prior instruction (e.g., branching).

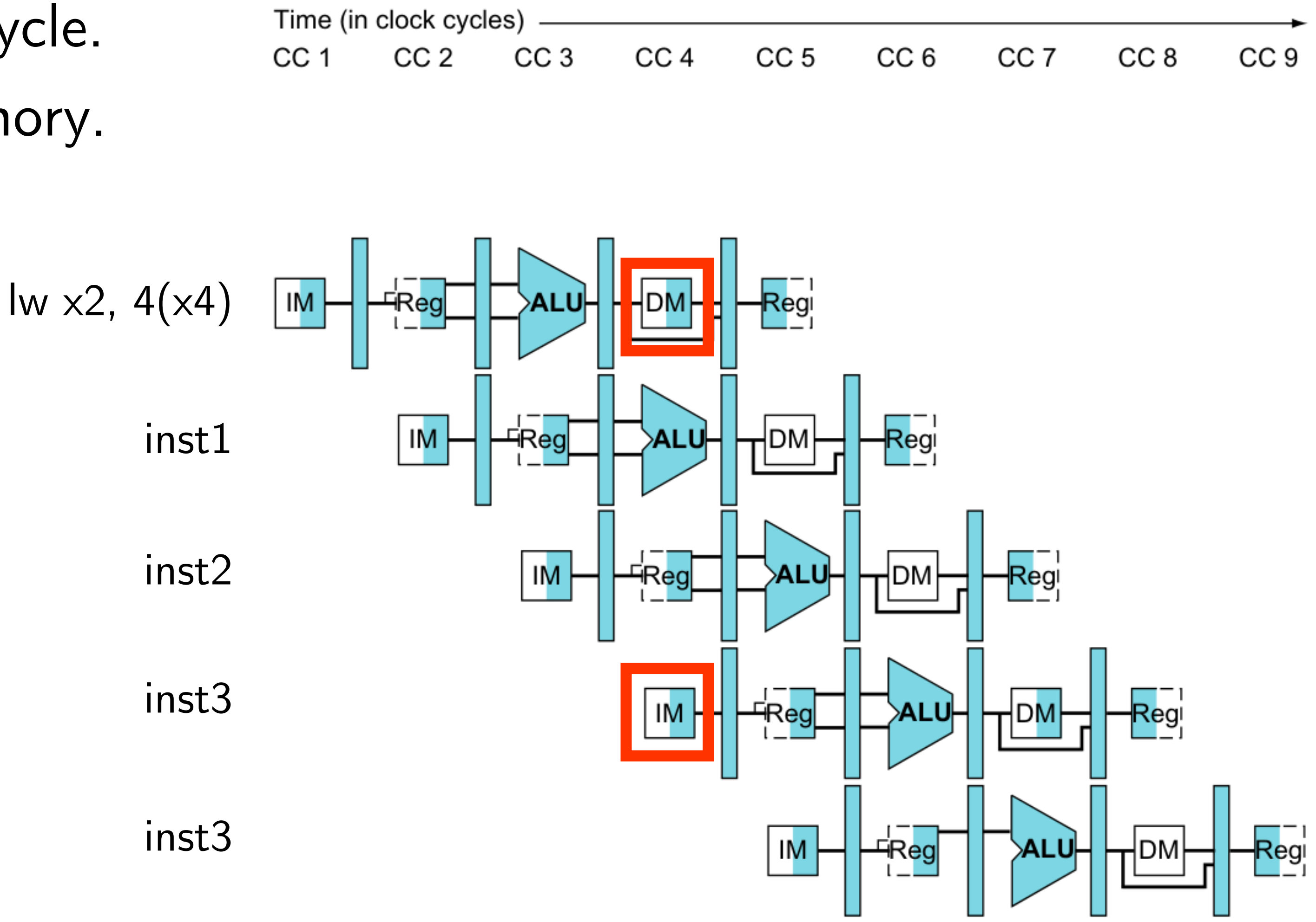
- Can usually be resolved by waiting:
  - Pipeline control must detect the hazard.
  - And take action to resolve hazards.

## Structure Hazards

- Conflict for use of a resource.
- In RISC-V pipeline with a single memory.
  - Load/store requires data access.
  - Instruction fetch requires instruction access.
- Hence, pipeline datapaths require separate instruction/data memories.
  - Or separate instruction/data caches.

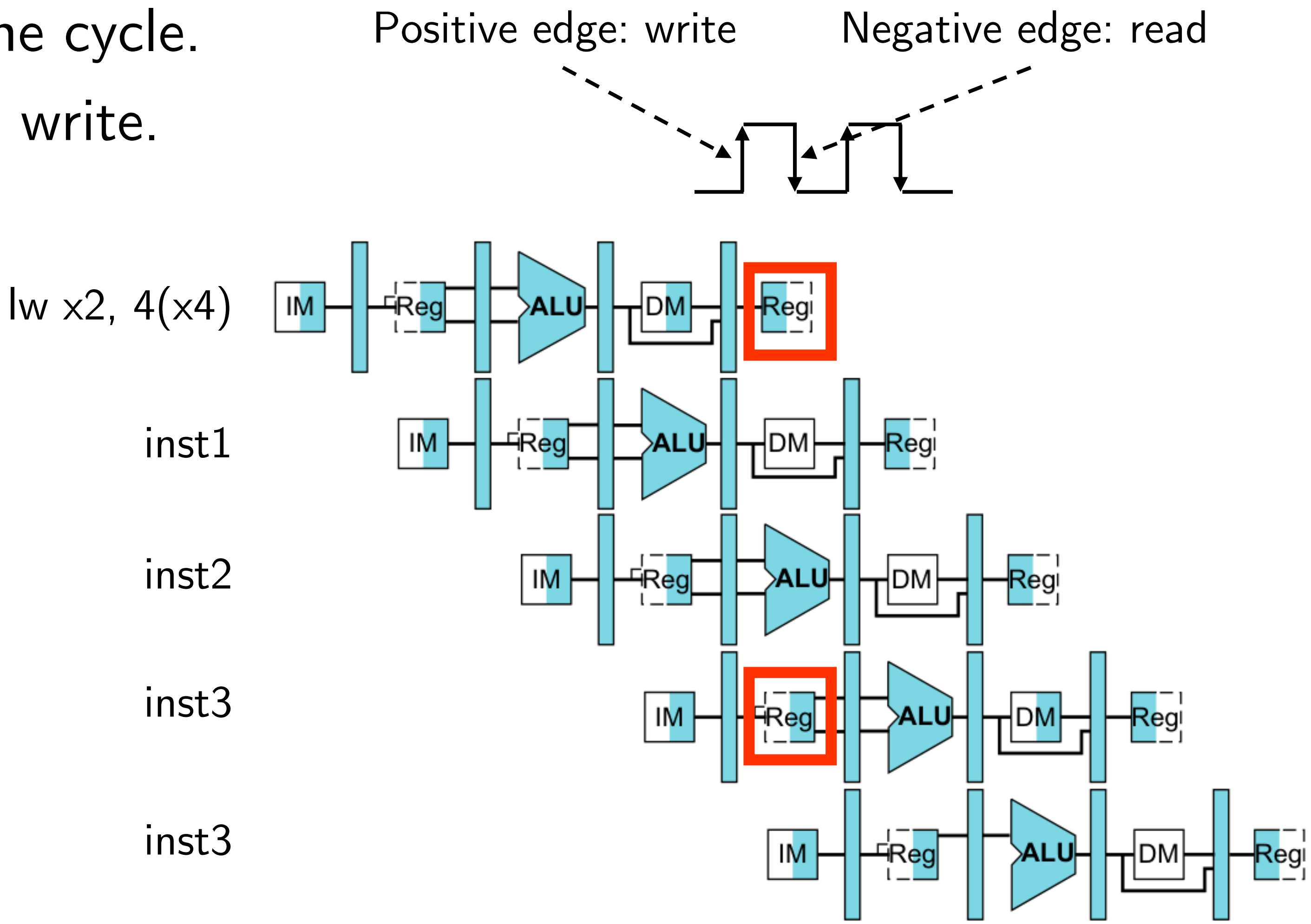
## Resolve Memory Structural Hazard

- Access memory at the same cycle.
- Split int data/instruction memory.



# Resolve Register File Structural Hazard

- Read and write register at same cycle.
- Resolved by splitting read and write.
  - First half of cycle, write.
  - Second half of cycle, read.



Positive edge: write
Negative edge: read
lw x2, 4(x4)
inst1
inst2
inst3
inst3
IM
Reg
ALU
DM
Reg