UC Berkeley
Teaching Professor
Dan Garcia

CS61C

Great Ideas in Computer Architecture

(a.k.a. Machine Structures)

UC Berkeley
Teaching Professor
Lisa Yan

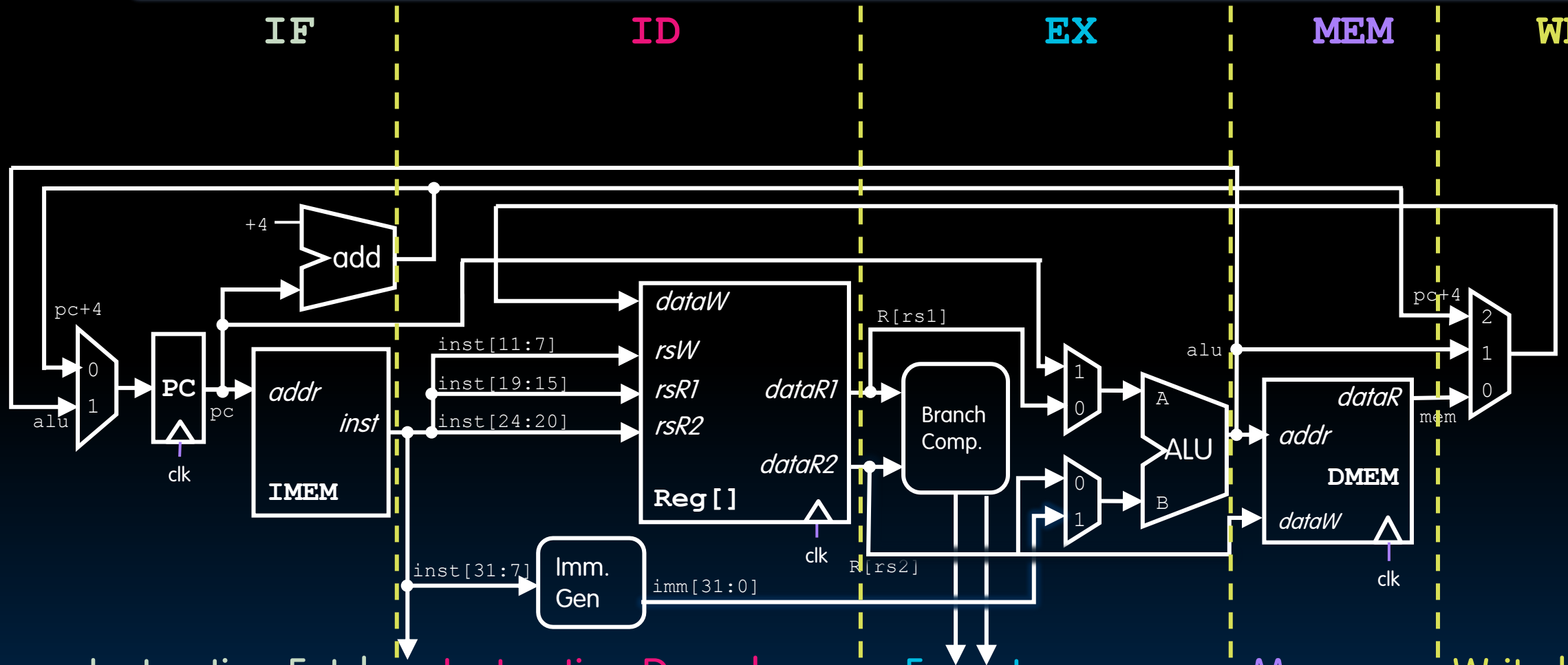
Pipeline II: Control Hazards, Data Hazards I

RISC-V's 5-Stage Pipeline

- RISC-V's 5-Stage Pipeline
- Pipelined Datapath and Control
- Structural Hazards
- Data Hazards I (Stalling, Forwarding)

Single-Cycle RV32I Datapath

Review



Instruction Fetch (**IMEM** Read)

Instruction Decode (**Reg** Read)

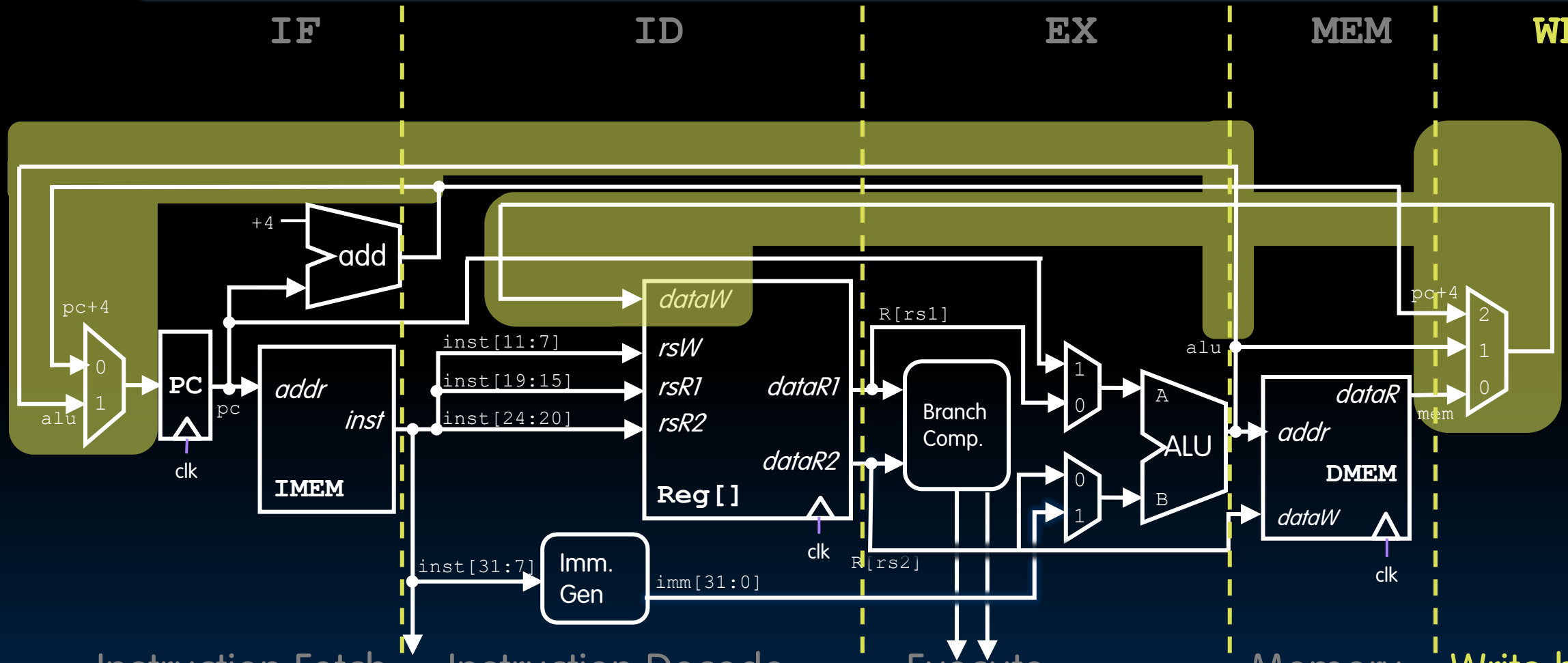
Execute **ALU**

Memory Access (**DMEM**)

Write back to register (**Reg** write)

Single-Cycle RV32I Datapath

Review



Instruction Fetch (**IMEM** Read)

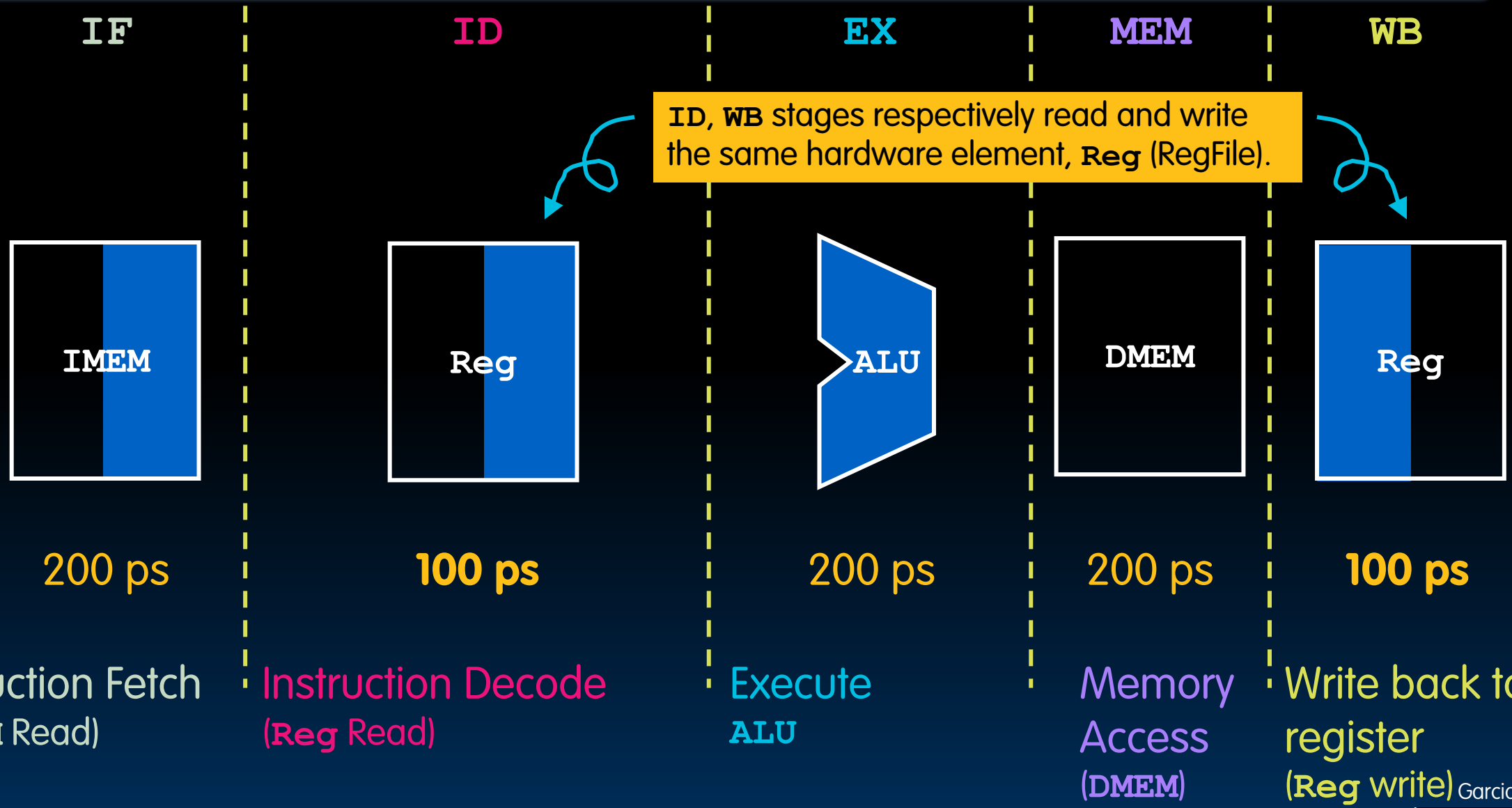
Instruction Decode (**Reg** Read)

Execute **ALU**

Memory Access (**DMEM**)

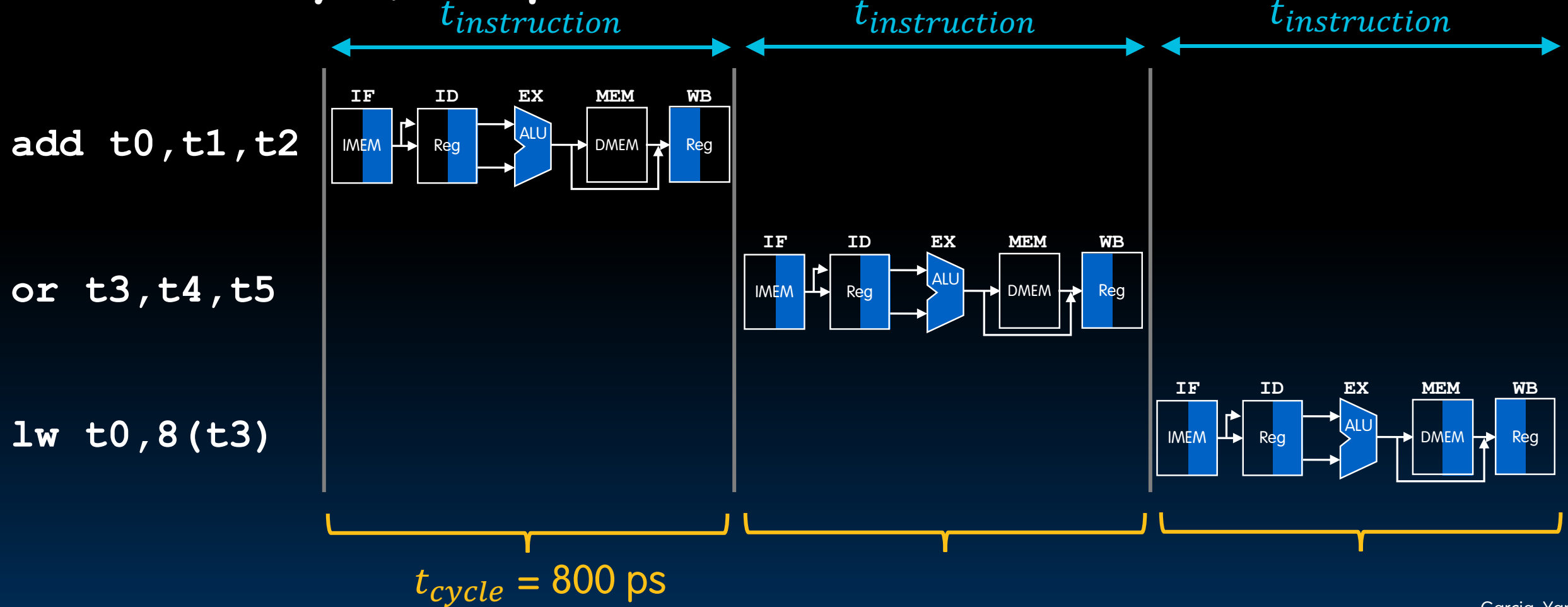
Write back to register (**Reg** write)

Symbolic Representation of 5 Stages



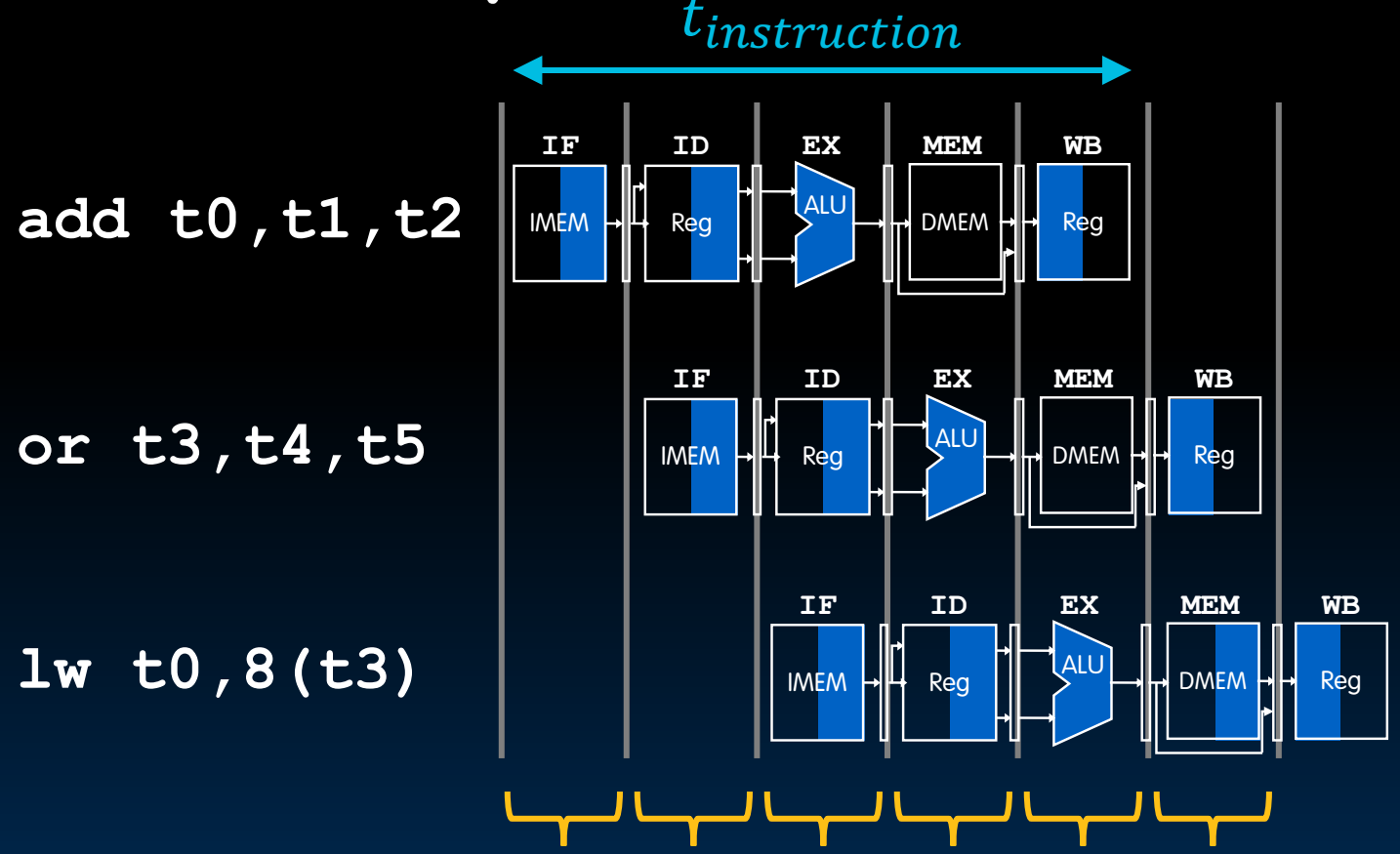
"Sequential" RISC-V Datapath

- In a single-cycle CPU, only one instruction can access any resources in one clock cycle, in sequence.



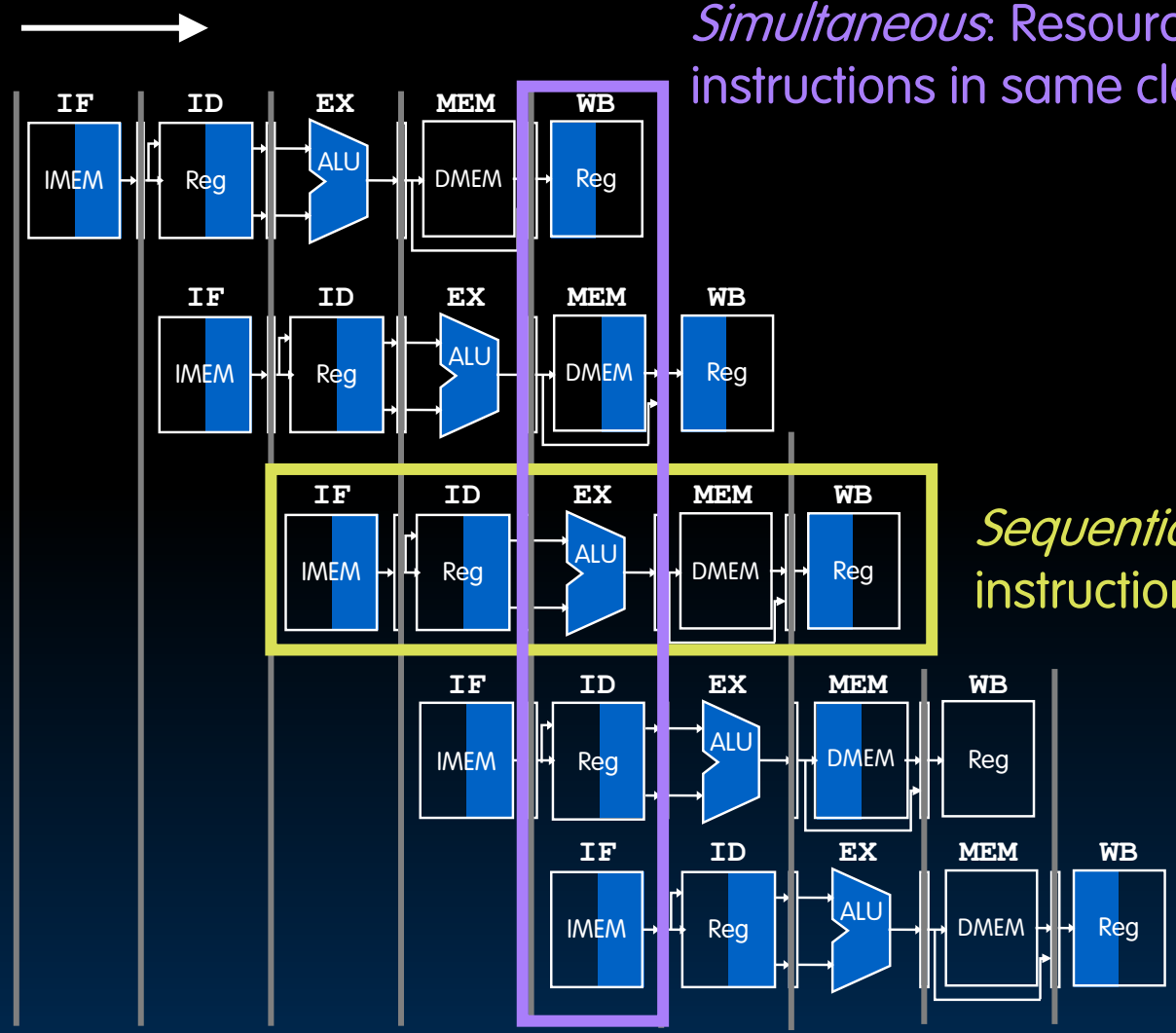
Pipelined RISC-V Datapath

- In a pipelined CPU, multiple instructions access resources in one clock cycle.



What Happens Sequentially? Simultaneously?

```
add t0,t1,t2
lw t0,8(t3)
or t3,t4,t5
sw t0,4(t3)
sll t6,t0,t3
```

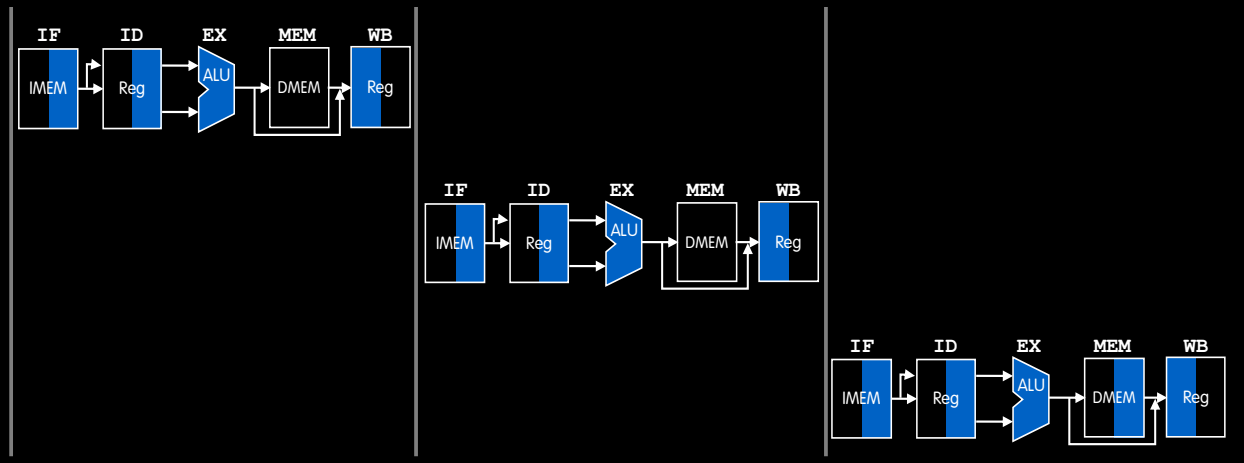


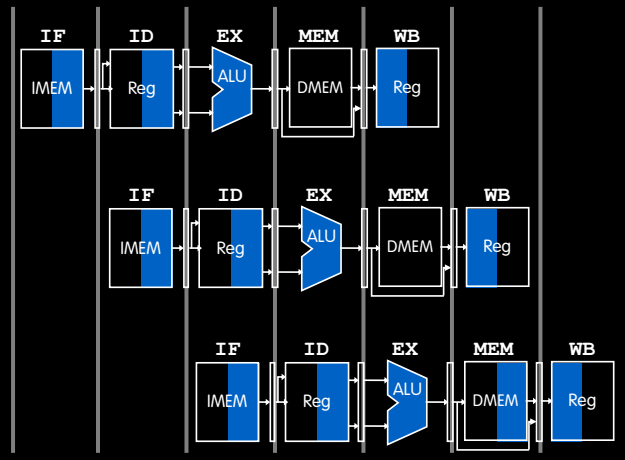
Simultaneous: Resource use by multiple instructions in same clock cycle.

Sequential: Resource use by same instruction over time (multiple clock cycles)

Performance (1/2): Latency

```
add t0,t1,t2
or t3,t4,t5
lw t0,8(t3)
```





	Single Cycle	Pipelined
Timing of each stage	t_{stage} = 200, 100, 200, 200, 100 ps (Reg access stages **ID**, **WB** only 100 ps)	t_{stage} = 200 ps All stages same length
Instruction time (Latency)	$t_{instruction}$ = 800 ps	$t_{instruction} = 5 \cdot t_{cycle}$ = *1000 ps*

The pipelined CPU uses one clock for all stages; clock cycle time is limited by the slower stages.

Performance (2/2): Throughput

"Iron Law" of Processor Performance:

$$\frac{time}{program} = \frac{instructions}{program} \times \underbrace{\frac{cycles}{instruction}}_{\text{CPI, inverse of (\# instrs executed/cycle)}} \times \underbrace{\frac{time}{cycle}}_{t_{cycle}}$$

1/throughput

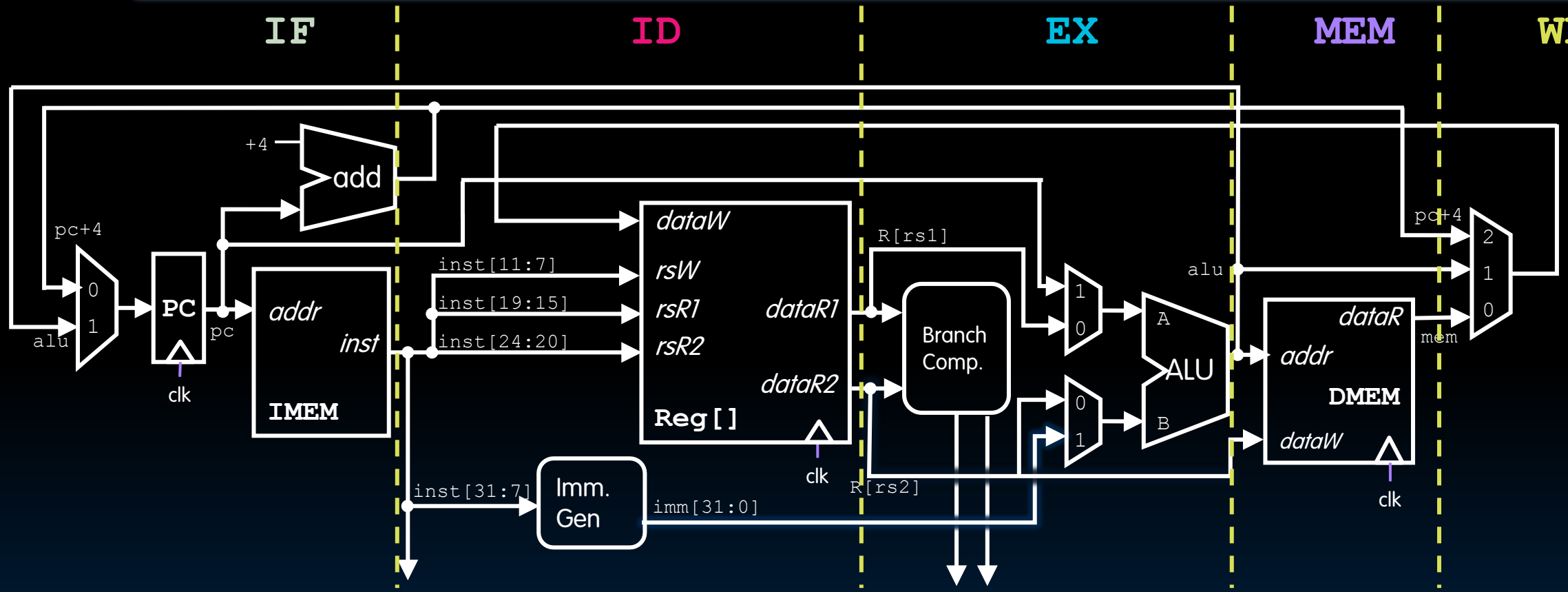
Processor Throughput: $\frac{\#\ instructions}{time}$

	Single Cycle	Pipelined
Timing of each stage	t_{stage} = 200, 100, 200, 200, 100 ps (Reg access stages `ID`, `WB` only 100 ps)	t_{stage} = 200 ps All stages same length
Instruction time (Latency)	$t_{instruction}$ = 800 ps	$t_{instruction} = 5 \cdot t_{cycle}$ = *1000 ps*
Clock cycle time, t_{cycle} Clock rate, $f_s = 1/t_{cycle}$	$t_{cycle} = t_{instruction}$ = 800 ps f_s = 1/800 ps = 1.25 GHz	$t_{cycle} = t_{stage}$ = 200 ps f_s = 1/200 ps = 5 GHz
CPI (Cycles Per Instruction)	~1 (ideal)	~1 (ideal) <1 (actual, more later/CS152)
Relative throughput gain	1 x	**4 x**

Pipelined Datapath and Control

- RISC-V's 5-Stage Pipeline
- Pipelined Datapath and Control
- Structural Hazards
- Data Hazards I (Stalling, Forwarding)

Constructing a Pipelined RV32I Datapath



- **A pipelined datapath needs to "separate" the five stages of the RV32I datapath.**
  - Each stage needs to process data from a different instruction.

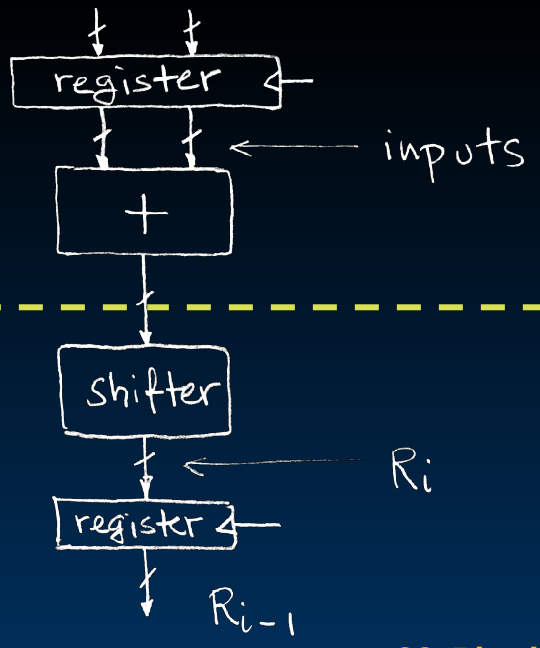
Use *pipeline registers* to carry instruction data between stages!

Pipelining to improve performance

Review

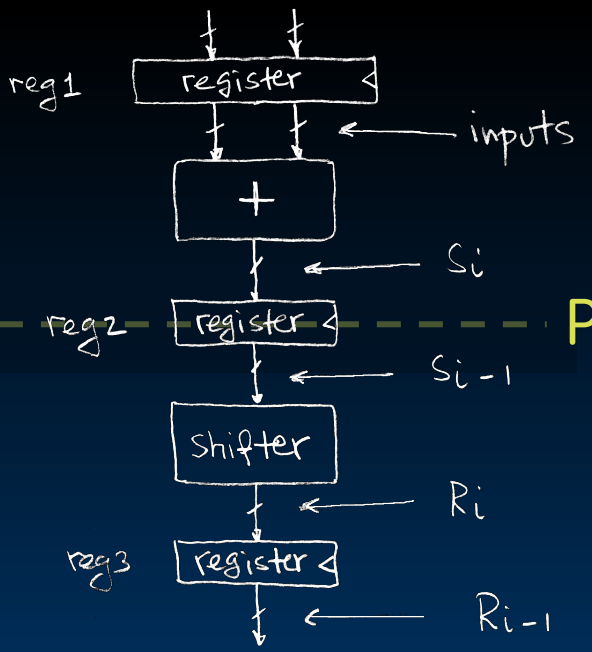
Single-cycle datapath means 1 clock cycle, from input to output.

- Clock period limited by propagation delays of adder and shifter.



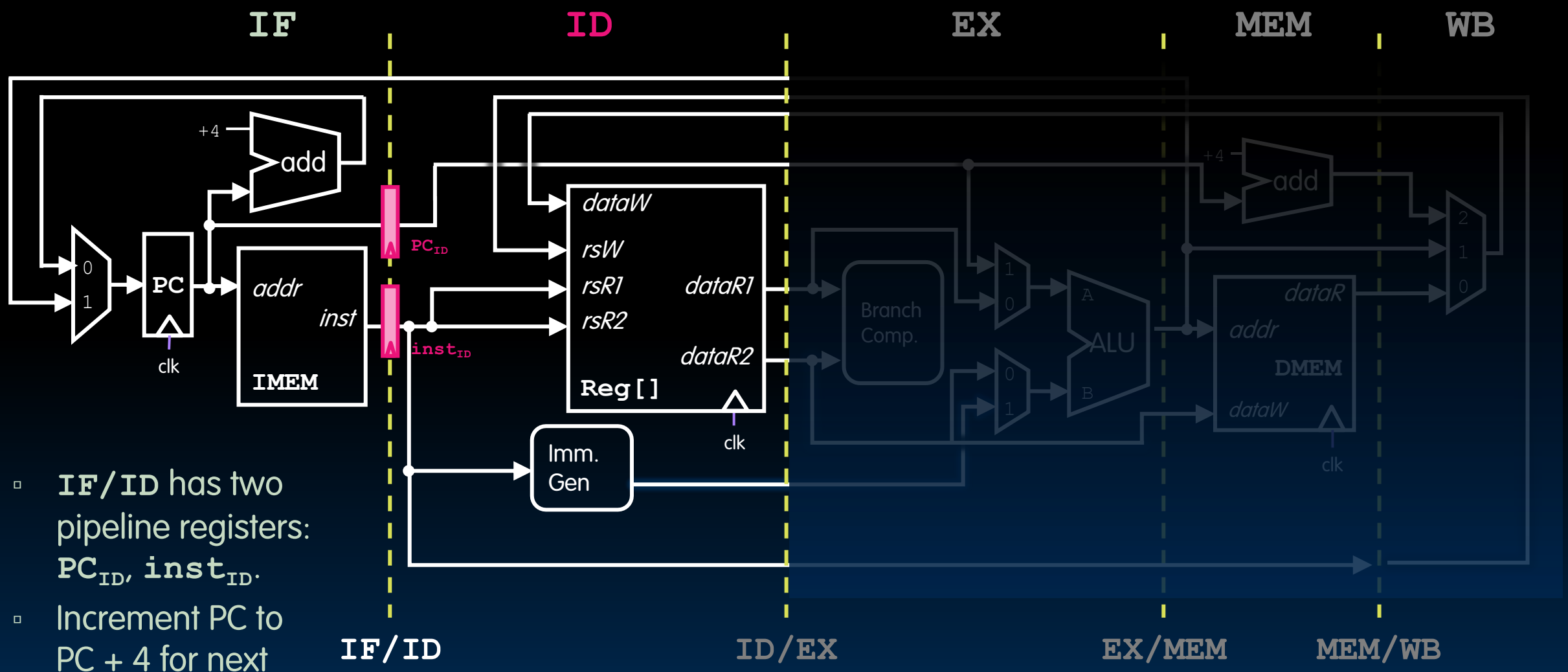
Insertion of pipeline register allows higher clock frequency.

- Clock period now limited by *max*{adder/shifter prop. delays}.
- Higher throughput (outputs/s).



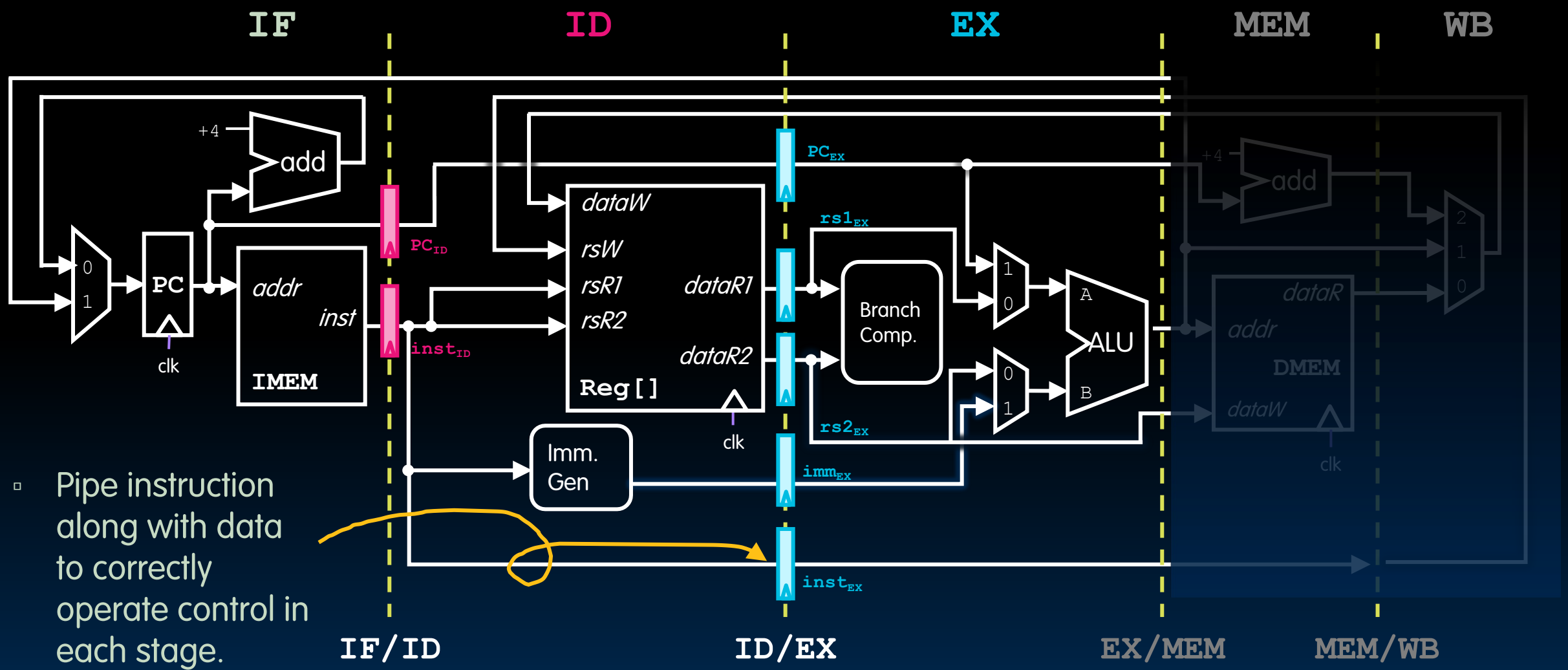
Pipeline Register

IF/ID Pipeline Registers



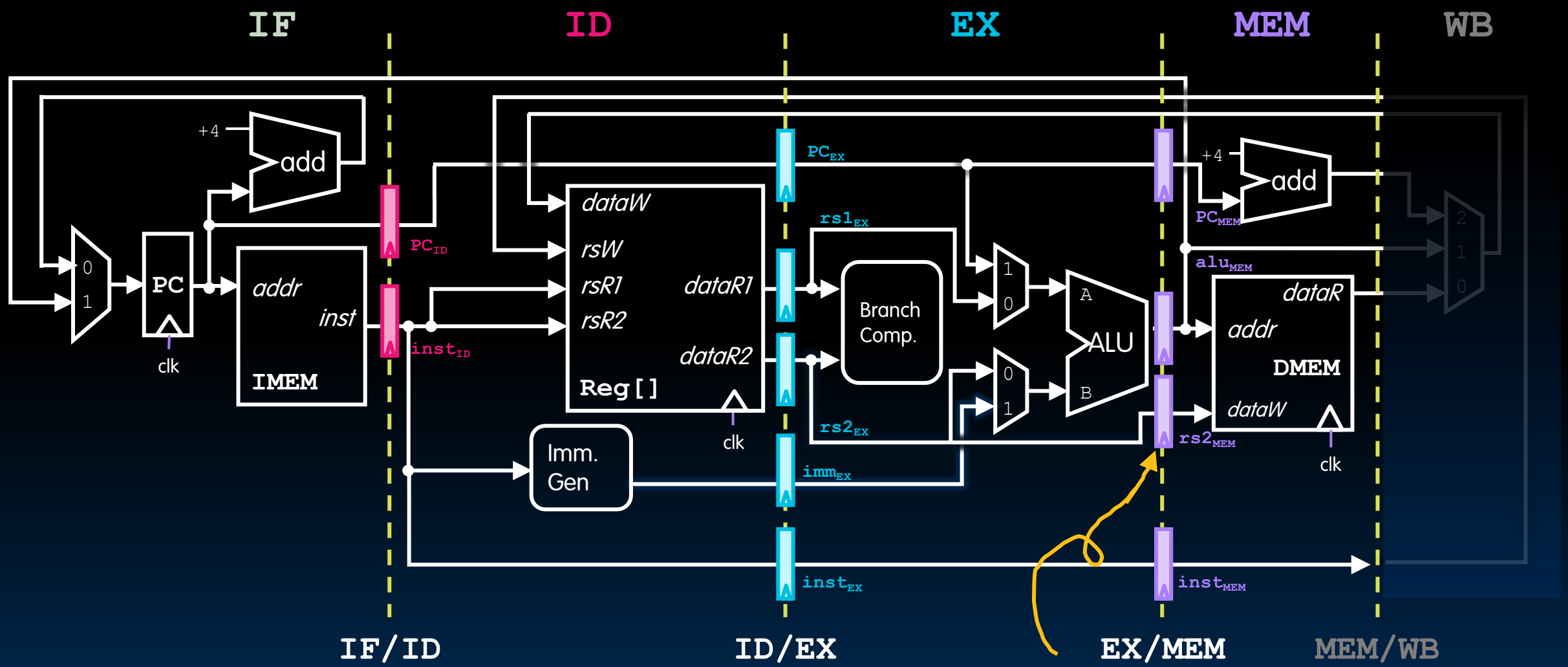
- IF/ID has two pipeline registers: PC_{ID}, $inst_{ID}$.
- Increment PC to PC + 4 for next cycle's IF stage.

ID/EX Pipeline Registers



- Pipe instruction along with data to correctly operate control in each stage.

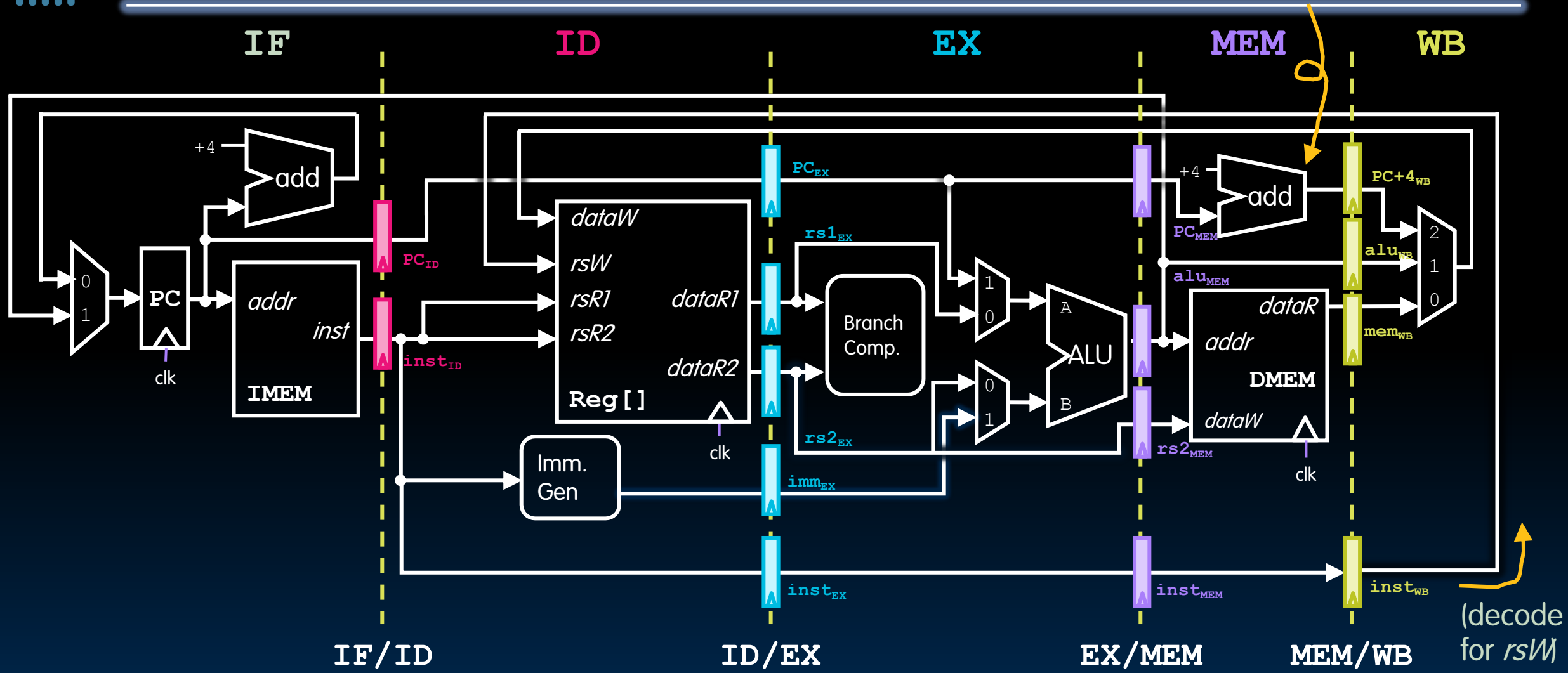
EX/MEM Pipeline Registers



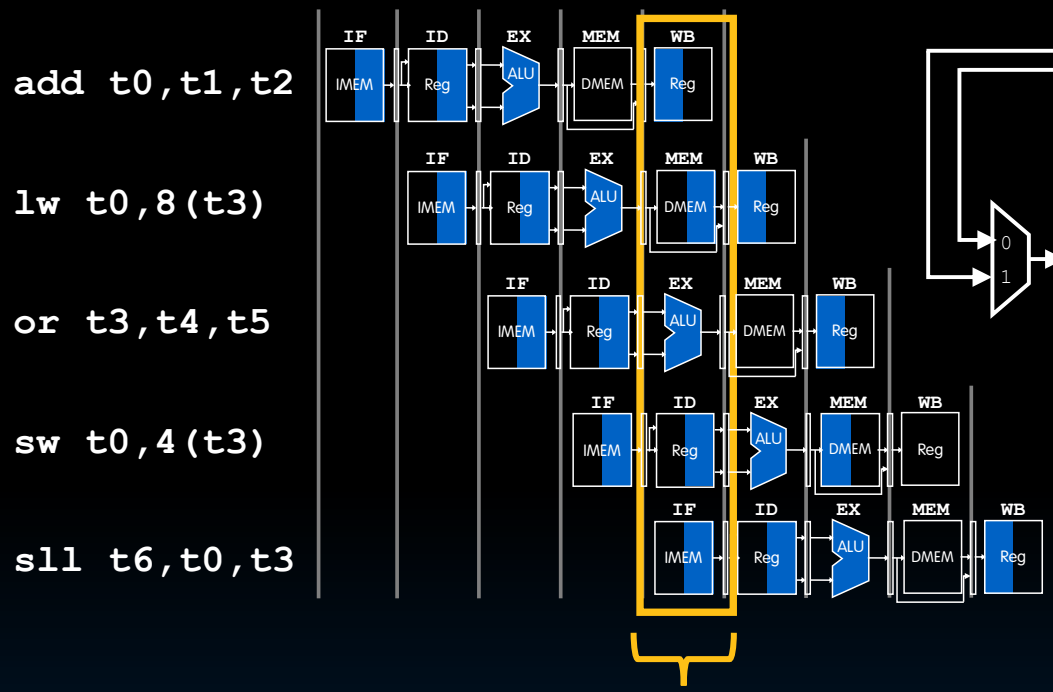
- `rs2` (data to store) needs to be piped through to **MEM**.

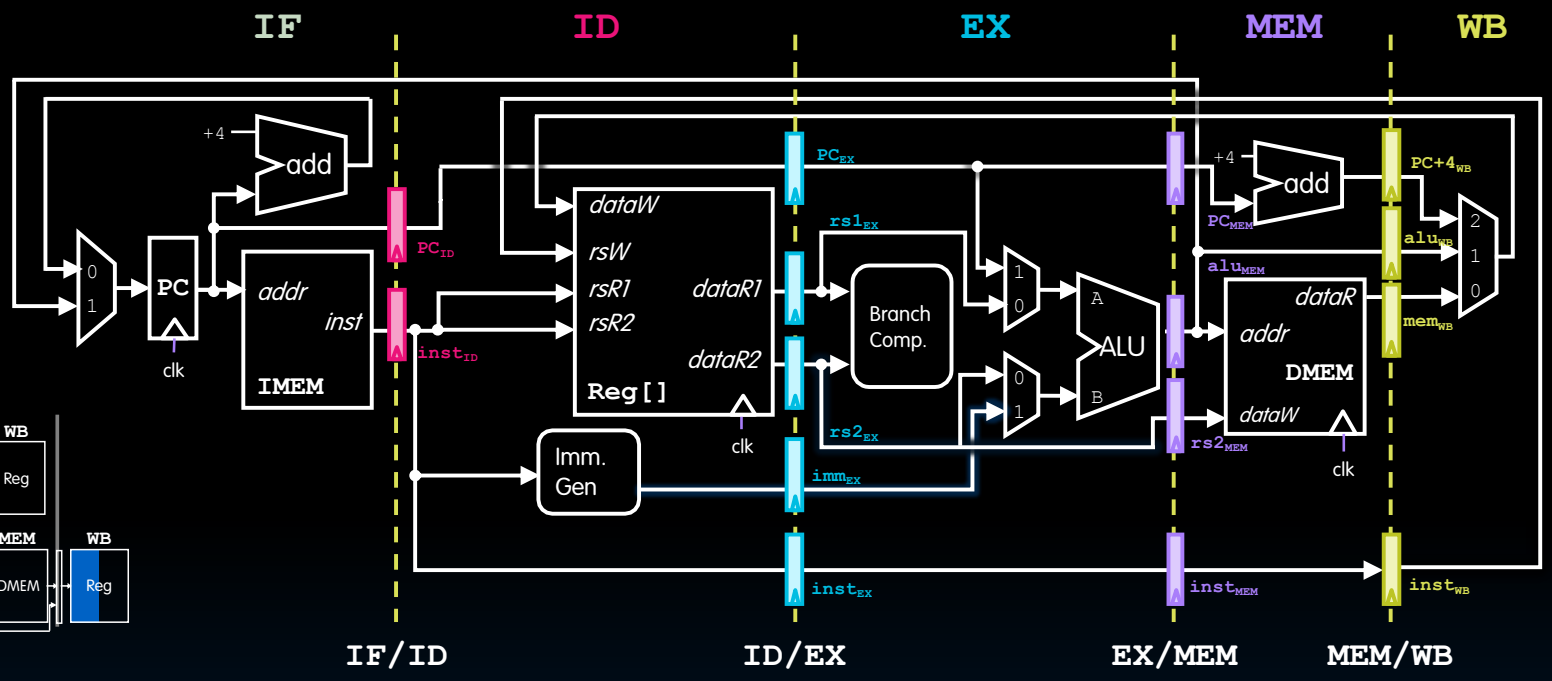
MEM/WB Pipeline Registers

Recalculate PC+4 to avoid sending both PC, PC+4 down pipeline.



Matching Question





Suppose the CPU is currently executing this clock cycle.

How are all simultaneously executing instructions using the pipelined stages of the datapath?

A. `add` `lw` `or` `sw` `sll`

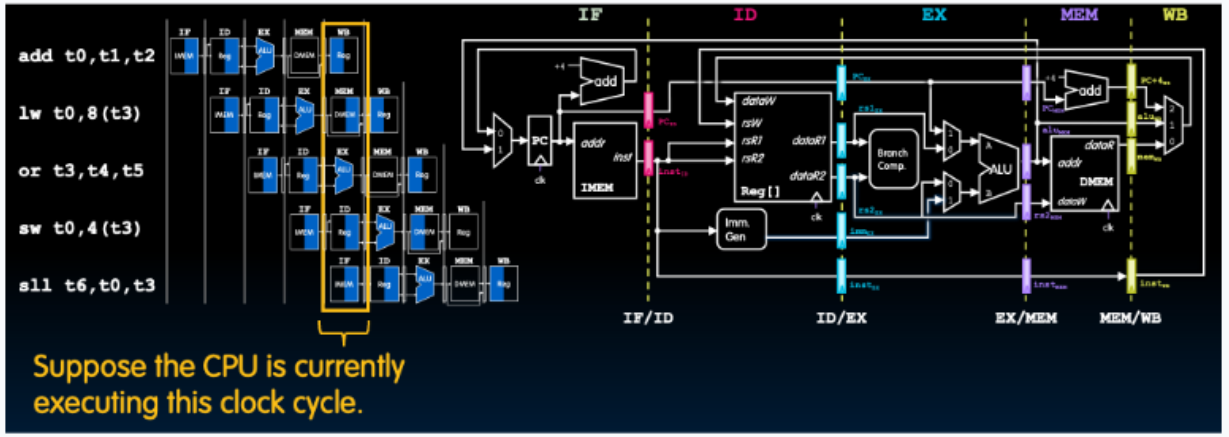
B. `sll` `sw` `or` `lw` `add`

C. Something else

When poll is active, respond at **pollev.com/yanl**

Text **YANL** to **22333** once to join

How are all simultaneously executing instructions using the pipelined stages of the datapath?



add, lw, or, sw, sll

sll, sw, or, lw, add

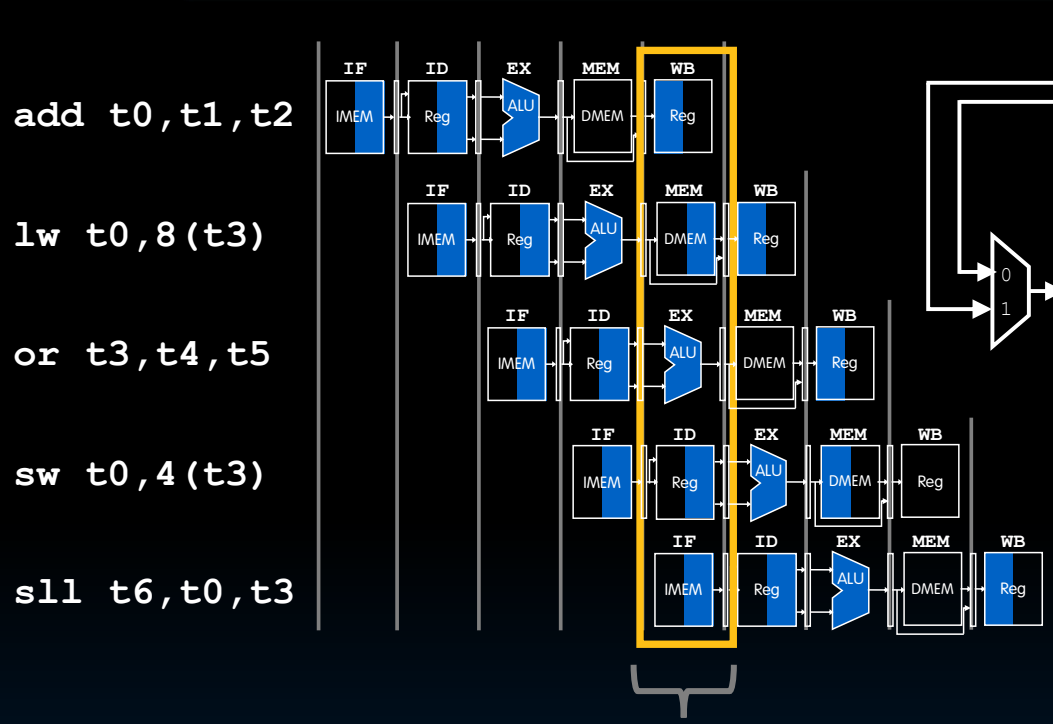
Something else

Total Results: 0

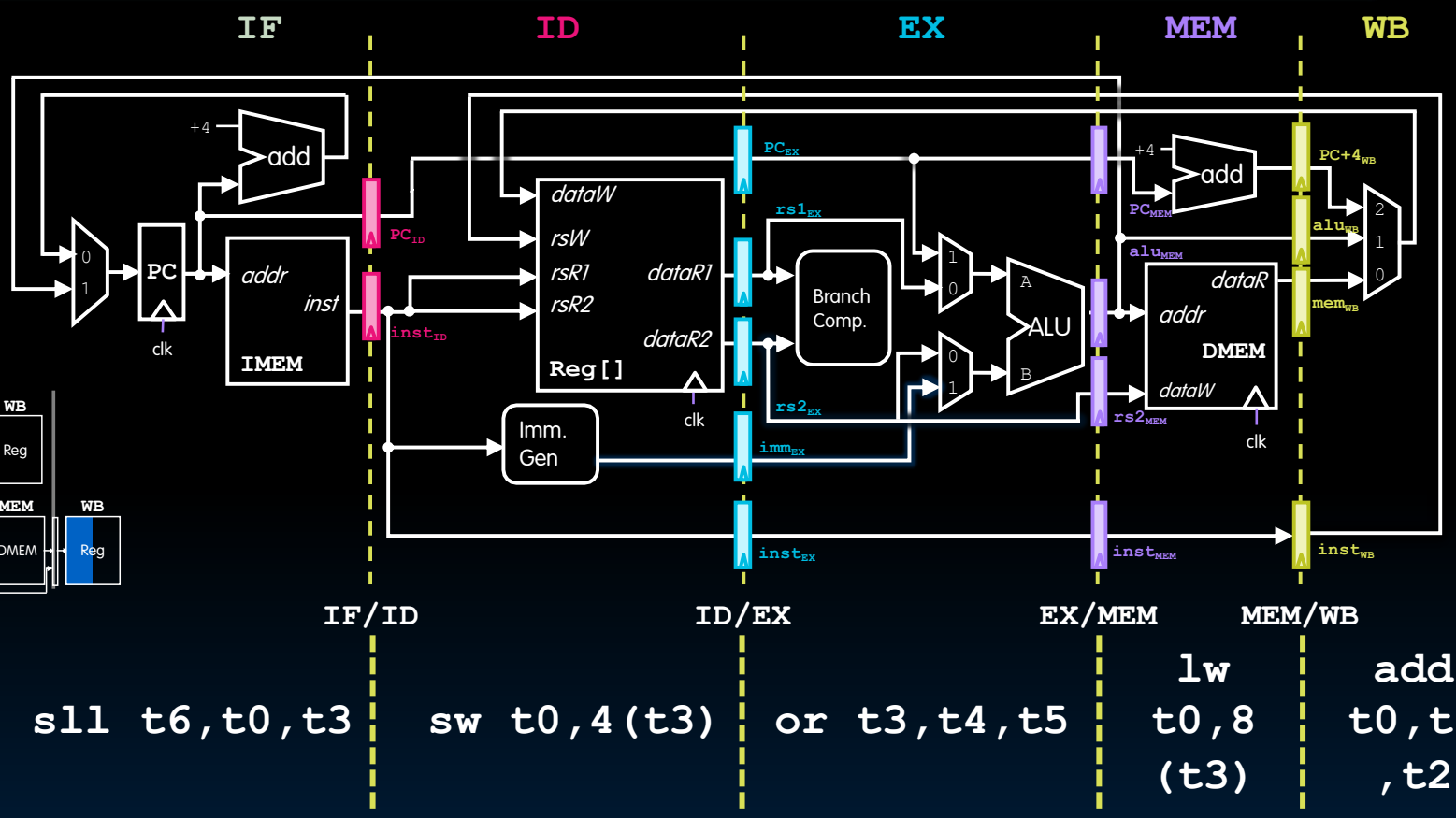
Powered by Poll Everywhere

Start the presentation to see live content. For screen share software, share the entire screen. Get help at **pollev.com/app**

Matching Solution



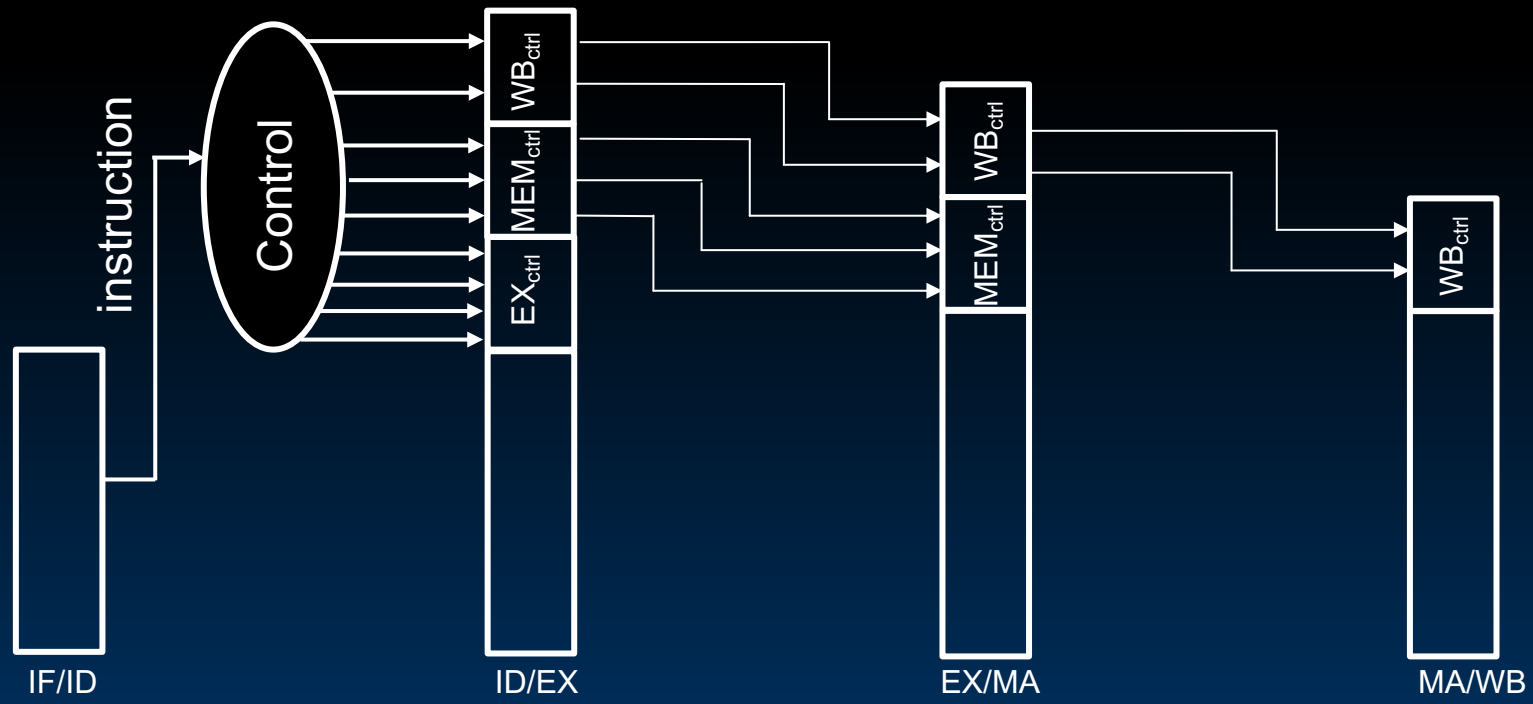
Suppose the CPU is currently executing this clock cycle.



The leftmost stage (**IF**) contains the most recent instruction. On the next clock cycle, pipeline registers carry the instruction/data to the next stage (**ID**).

Control Is Also Pipelined

- **Control signals are derived from the instruction.**
  - Like in the single-cycle CPU, control is usually computed during instruction decode (`ID`).
- **Control information for later stages is stored in pipeline registers.**
  - Example:

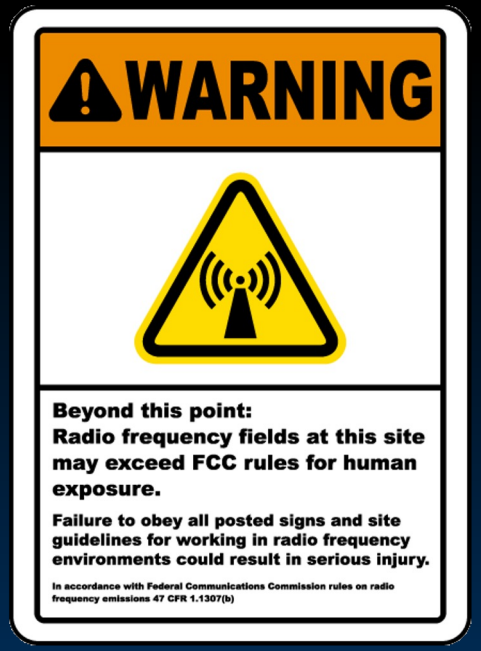


(more later on branch computation)

Structural Hazards

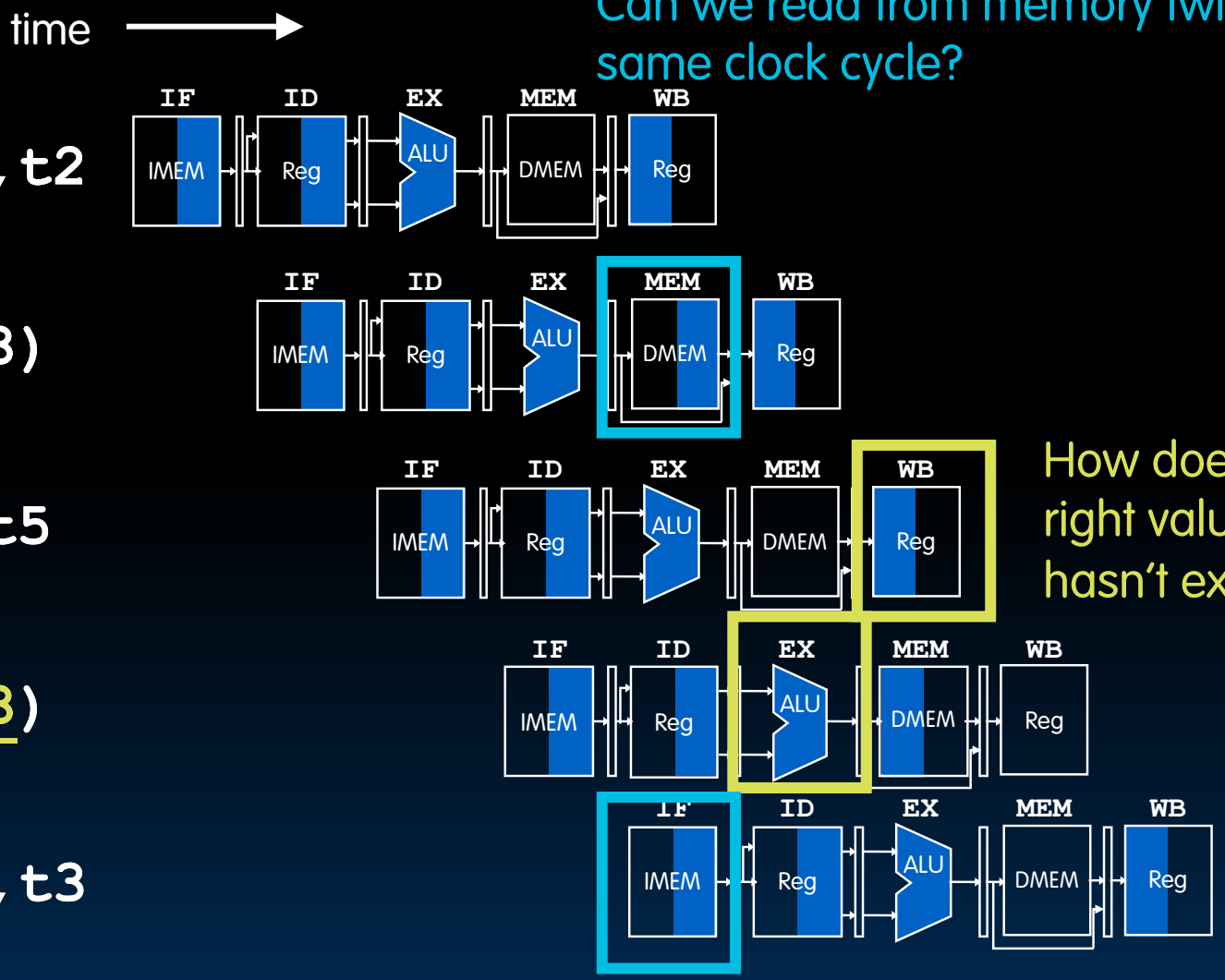
- RISC-V's 5-Stage Pipeline
- Pipelined Datapath and Control
- Structural Hazards
- Data Hazards I (Stalling, Forwarding)

Pipeline Hazards Ahead!!!



Pipeline Hazards Ahead!!!

Can we read from memory twice in the same clock cycle?



How does **sw**'s **EX** stage get the right value of **t3** if **or**'s **WB** stage hasn't executed yet??

How do branches work???

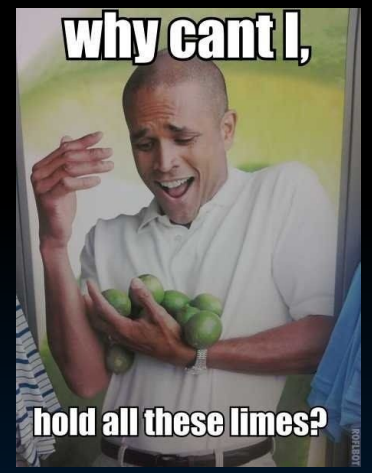
Three Types of Pipeline Hazards

A hazard is a situation in which a planned instruction cannot execute in the "proper" clock cycle.

1. Structural hazard:
   - Hardware does not support access across multiple instructions in the same cycle.
2. Data hazard:
   - Instructions have data dependency.
   - Need to wait for previous instruction to complete its data read/write.
3. Control hazard:
   - Flow of execution depends on previous instruction.

Structural Hazards

- Structural hazard:
  - Hardware does not support access across multiple instructions in the same cycle.
- **Occurs when multiple instructions compete for access to a single physical resource.**
- **Solution 1 (inefficient):**
  - Instructions take turns using the resource.
  - Some instructions *stall* while the resource is busy.
- **Solution 2: Add more hardware!**
  - Can always solve structural hazards by adding more HW.
  - In our current CPU, *structural hazards are not an issue*.

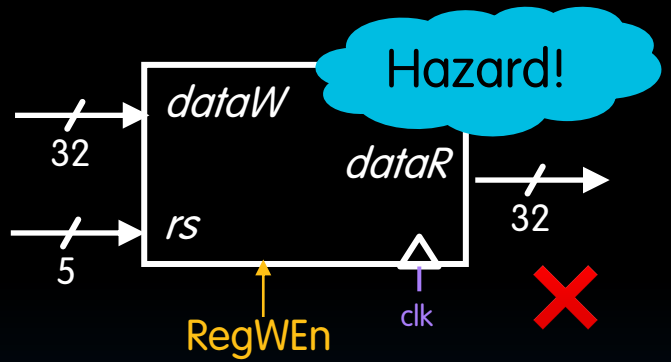


The RV32I ISA datapath avoids structural hazards via its hardware requirements on RegFile and Memory.

Required RegFile Avoids Structural Hazards

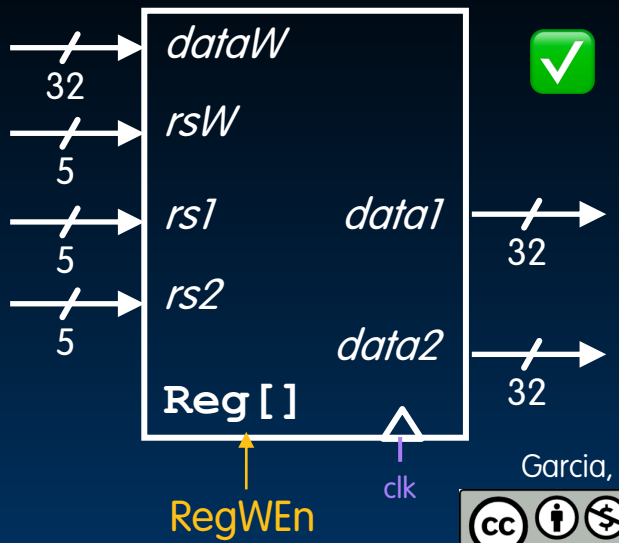
- **Each RV32I instruction:**
  - Reads up to 2 operands in ID (decode) stage; and
  - Writes up to 1 operand in WB (writeback) stage.

```
add t0,t1,t2
```

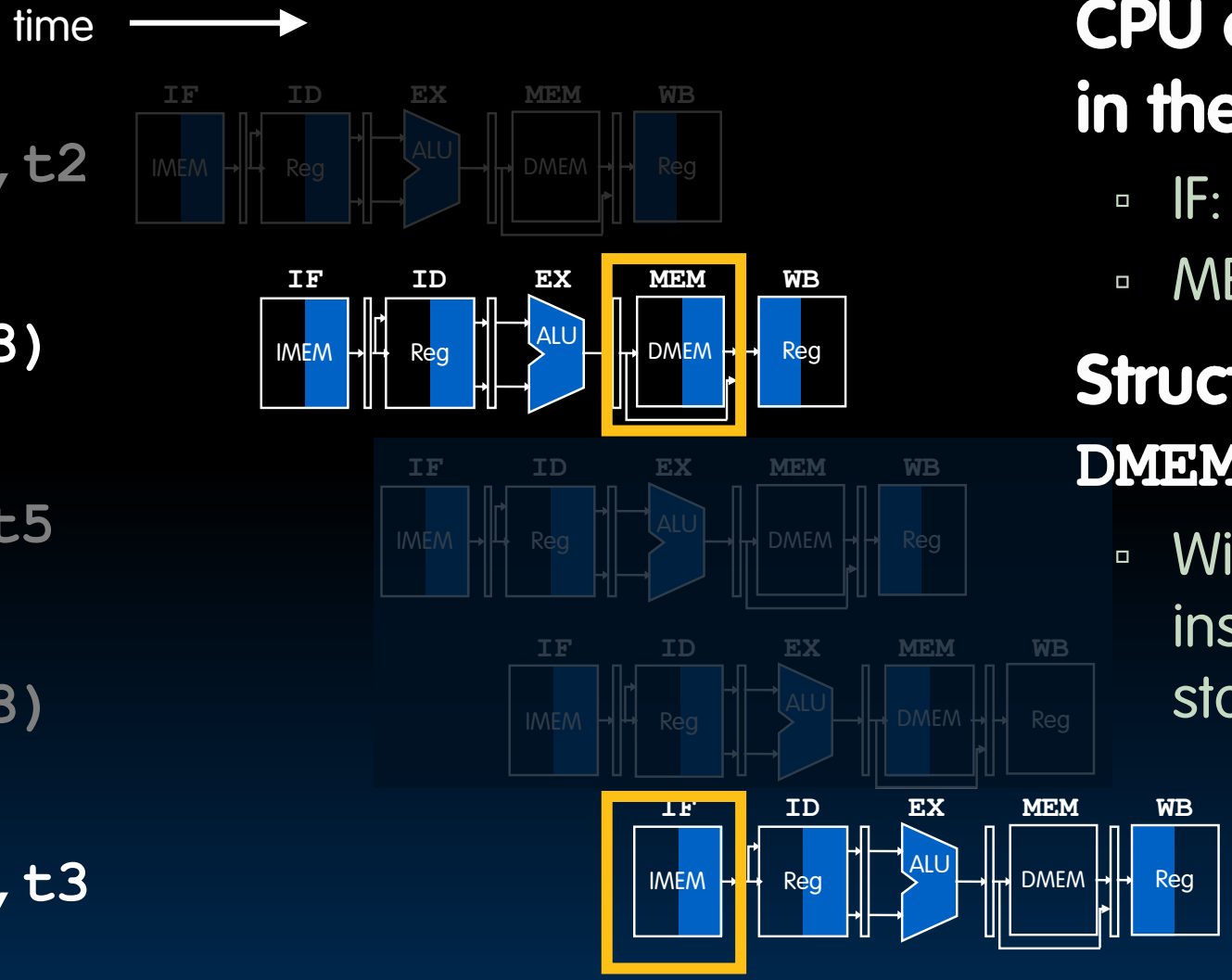


- **Structural hazard can occur if RegFile HW does not support simultaneous read/write!**

- **RV32I's required RegFile design works:**
  - Two independent read ports, one independent write port.
  - *Three accesses (2 read, 1 write)* can happen in the same cycle.



Separate IMEM, DMEM Avoids Structural Hazards



CPU can read memory twice in the same cycle:

- IF: Instruction memory (IMEM)
- MEM: data memory (DMEM)

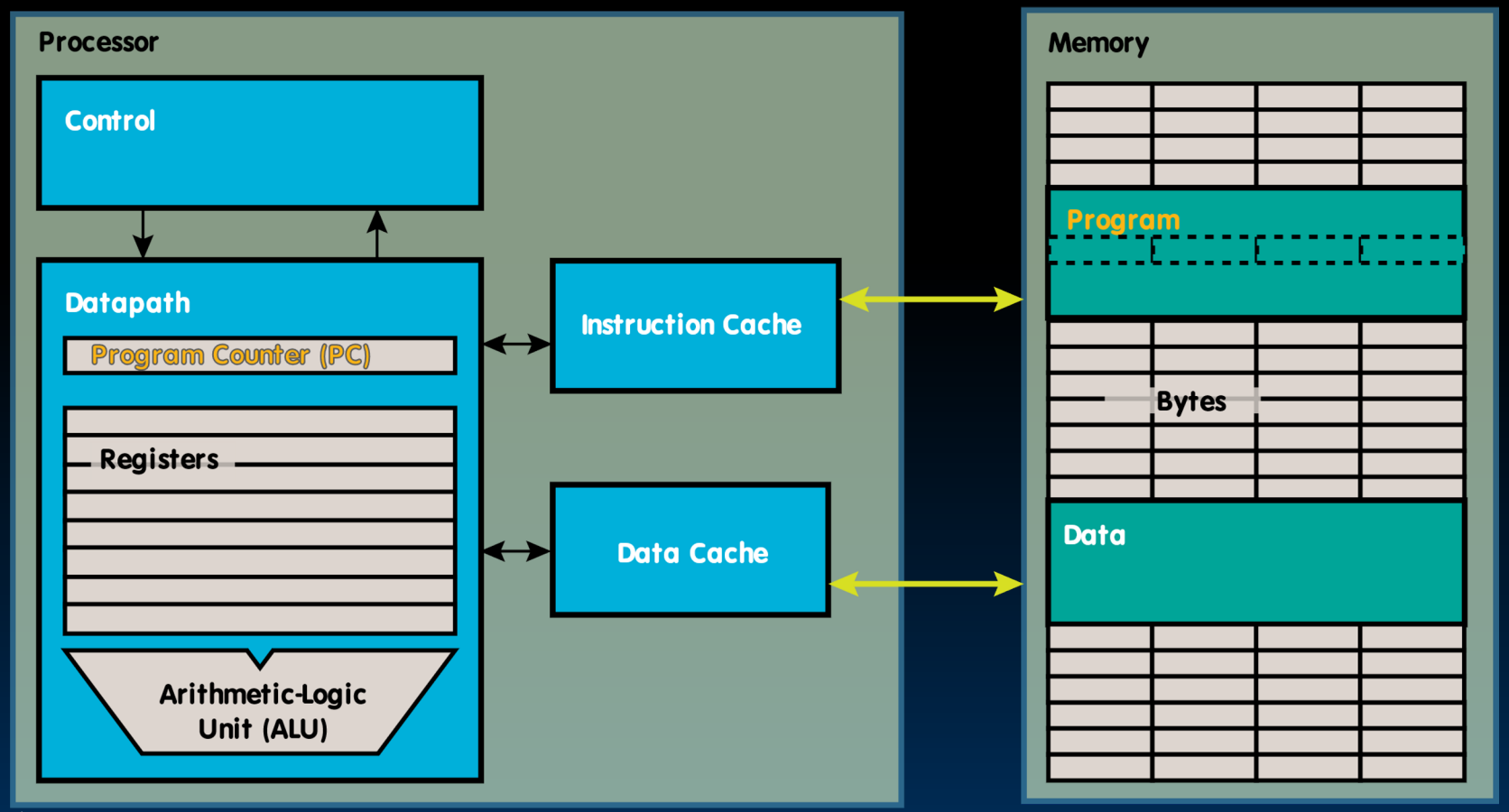
Structural hazard if IMEM, DMEM were same hardware:

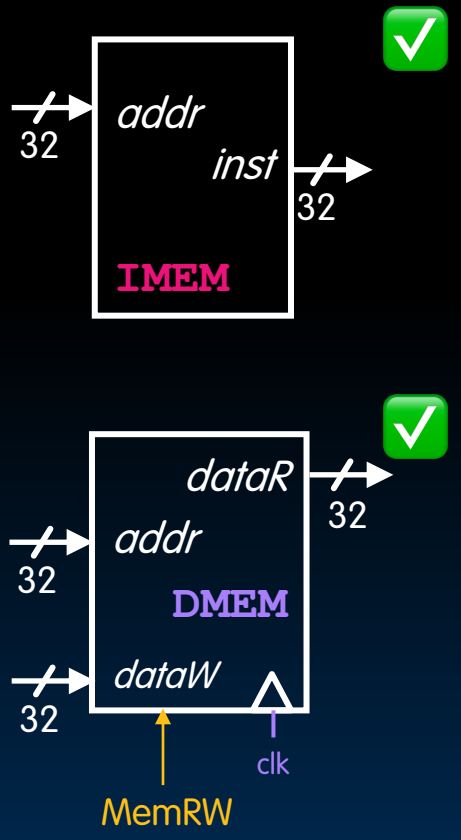
- Without separate memories, instruction fetch would have to stall for a cycle.

RV32I's required separation of IMEM and DMEM **works.**

Instruction and Data Caches

- Two fast, separate on-chip memories, one for instructions and one for data:





Data Hazards I: (Stalling, Forwarding)

- RISC-V's 5-Stage Pipeline
- Pipelined Datapath and Control
- Structural Hazards
- Data Hazards I (Stalling, Forwarding)

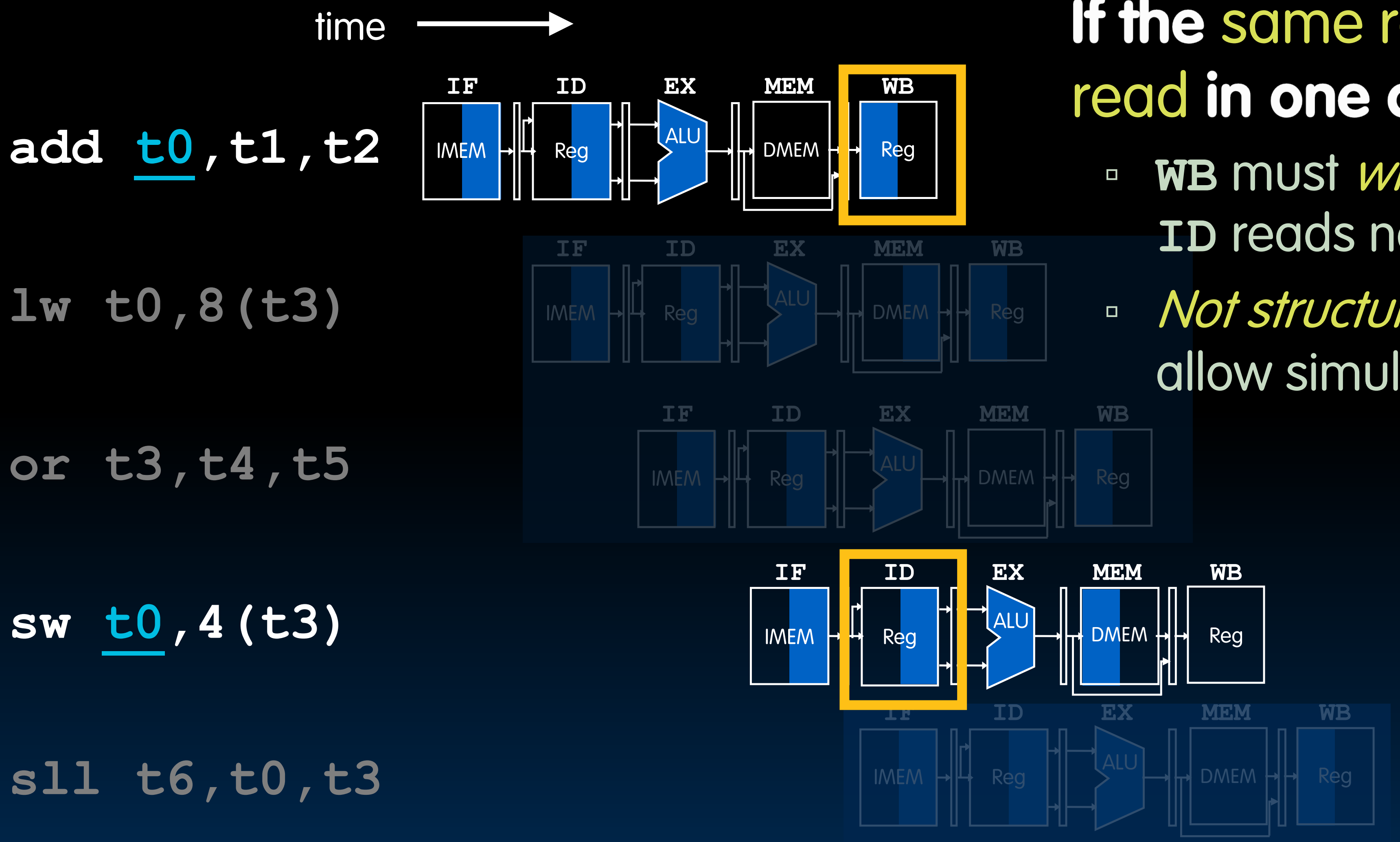
Data Hazards

- Data hazard:
  - Instructions have data dependency.
  - Need to wait for previous instruction to complete its data read/write.
- **Occurs when an instruction reads a register before a previous instruction has finished writing to that register.**

Three cases to consider:

1. Register access
2. ALU Result
3. Load data hazard (next time)

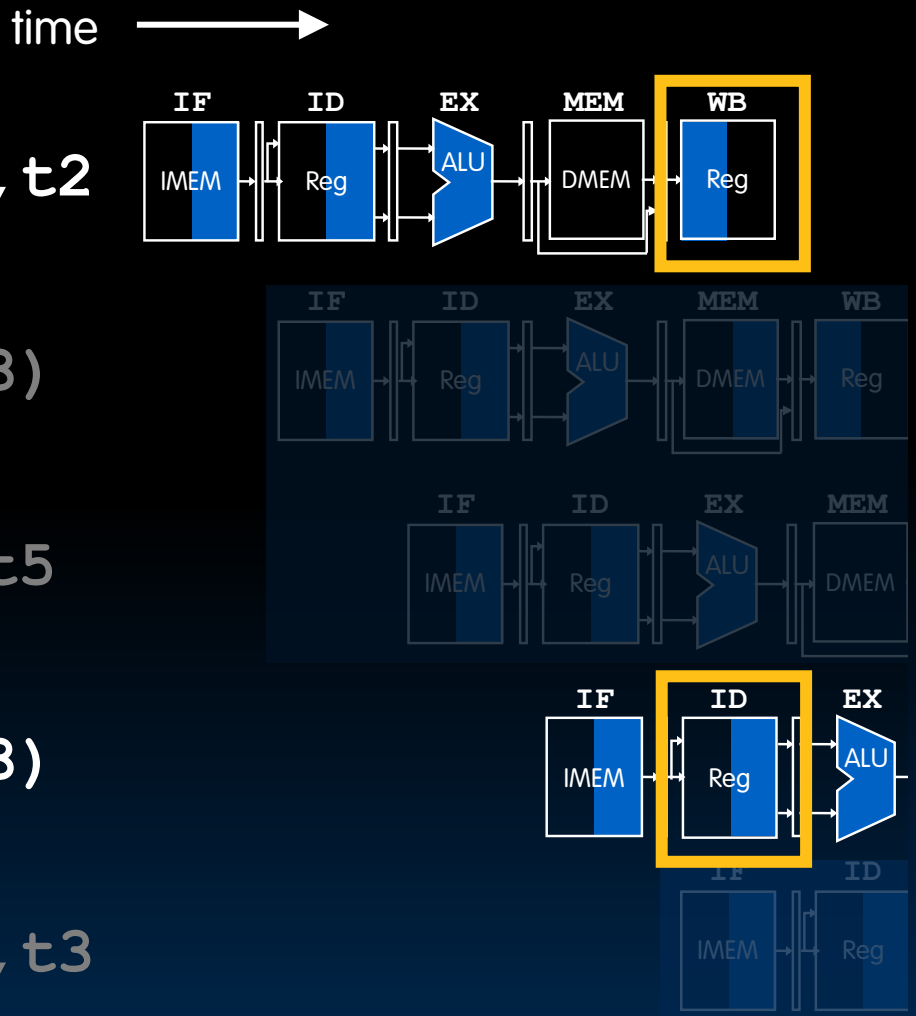
Data Hazard 1: Register Access



If the same register is written and read in one cycle:

- `WB` must *write value before* `ID` reads new value.
- *Not structural hazard*! Separate ports allow simultaneous R/W.

Register Access, Fixed with HW Requirement



If the same register is written and read **in one cycle:**

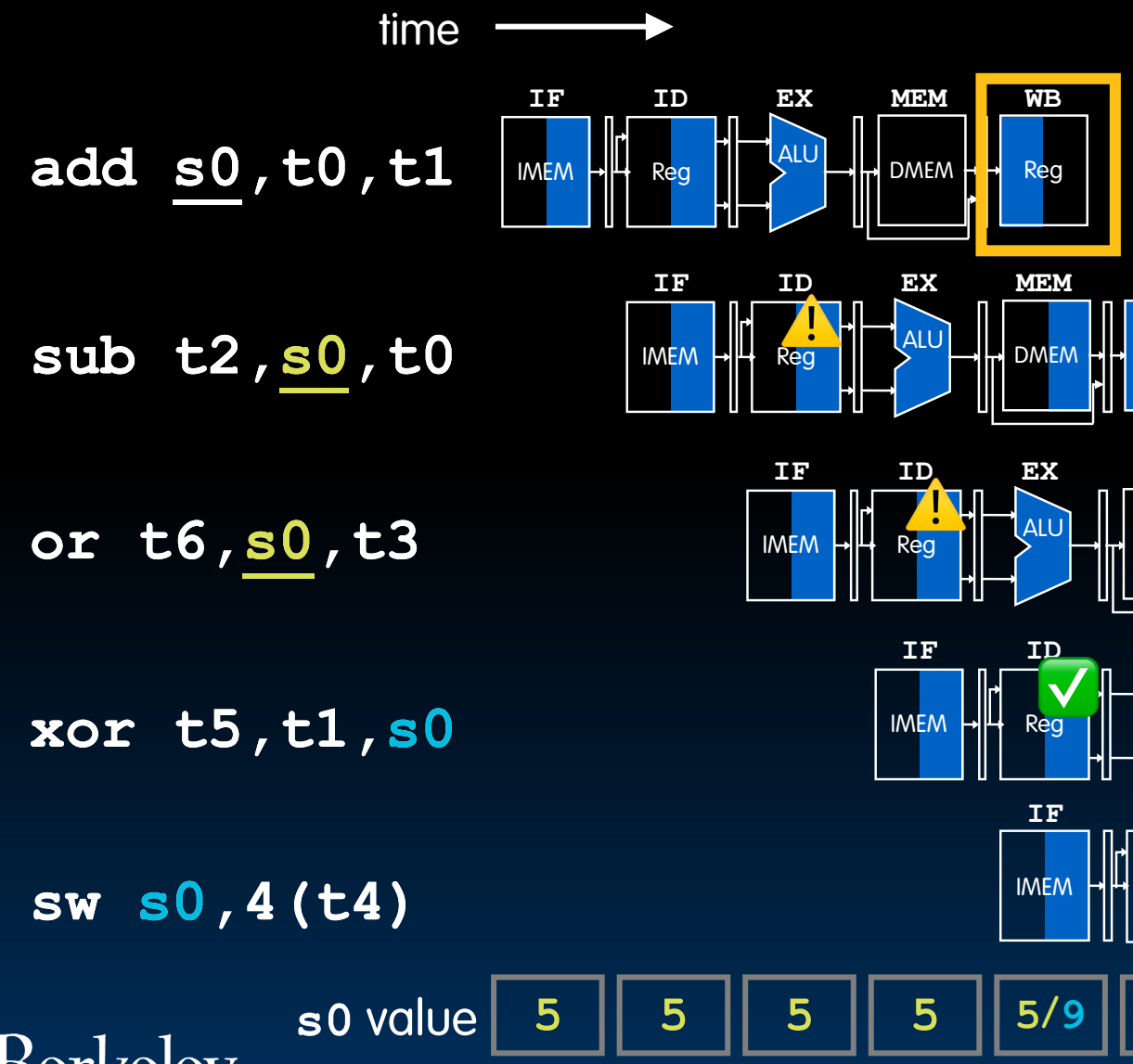
- `WB` must *write value before* `ID` reads new value.
- *Not structural hazard*! Separate ports allow simultaneous R/W.

Solution: RegFile HW should write-then-read in same cycle.

- Exploits high speed of RegFile (100 ps + 100 ps)
- Indicated by shading in diagram

Might not always be possible to write-then-read in same cycle, e.g., in high-frequency designs.

Data Hazard 2: ALU Result

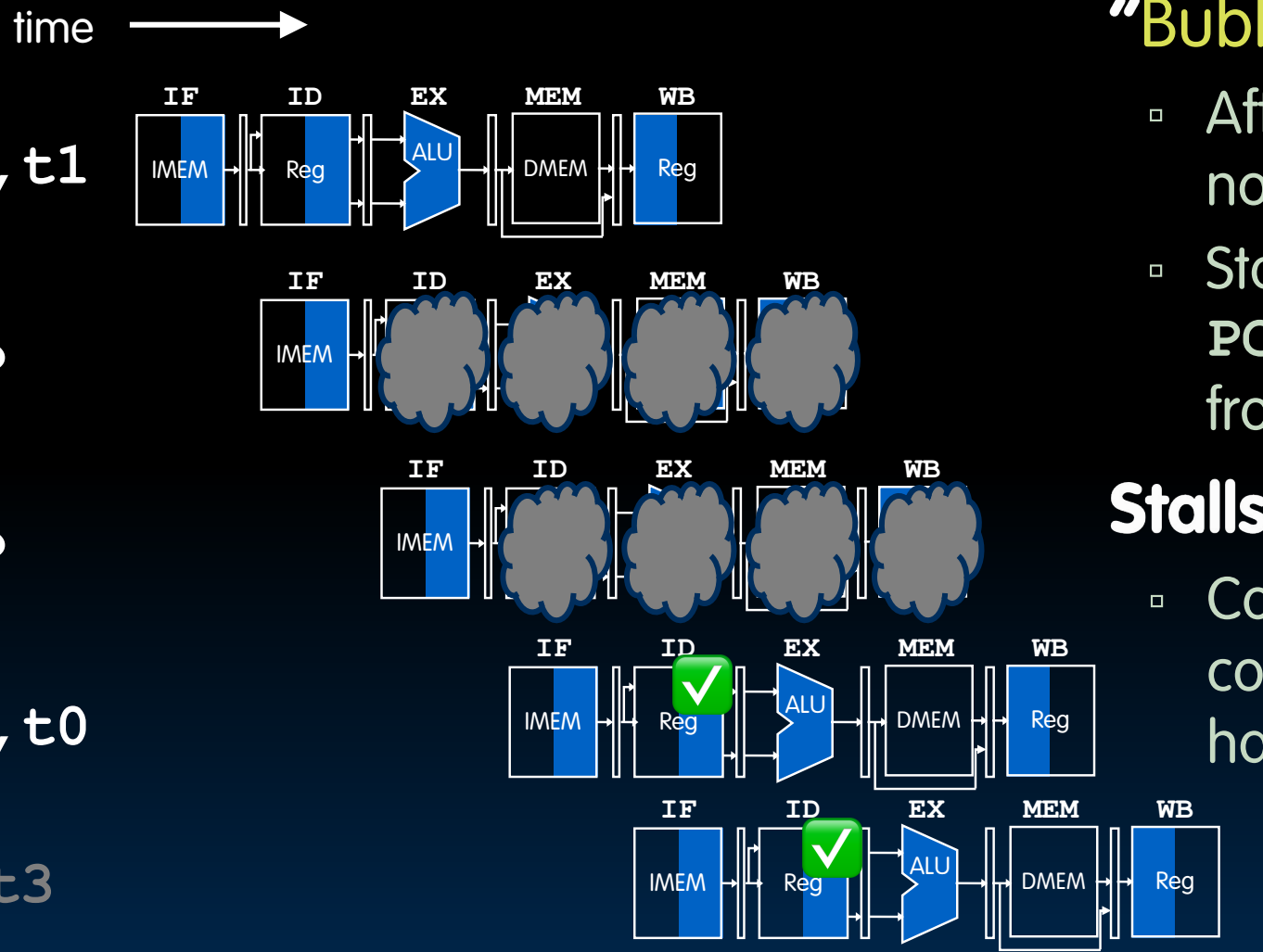


Problem: Instruction depends on WB's RegFile write from previous instruction.

- `sub`, `or`'s `ID` *reads old value* of `s0` and calculates wrong result.

Note: `xor` gets right value; RegFile is write-then-read.

ALU Solution 1: Stalling



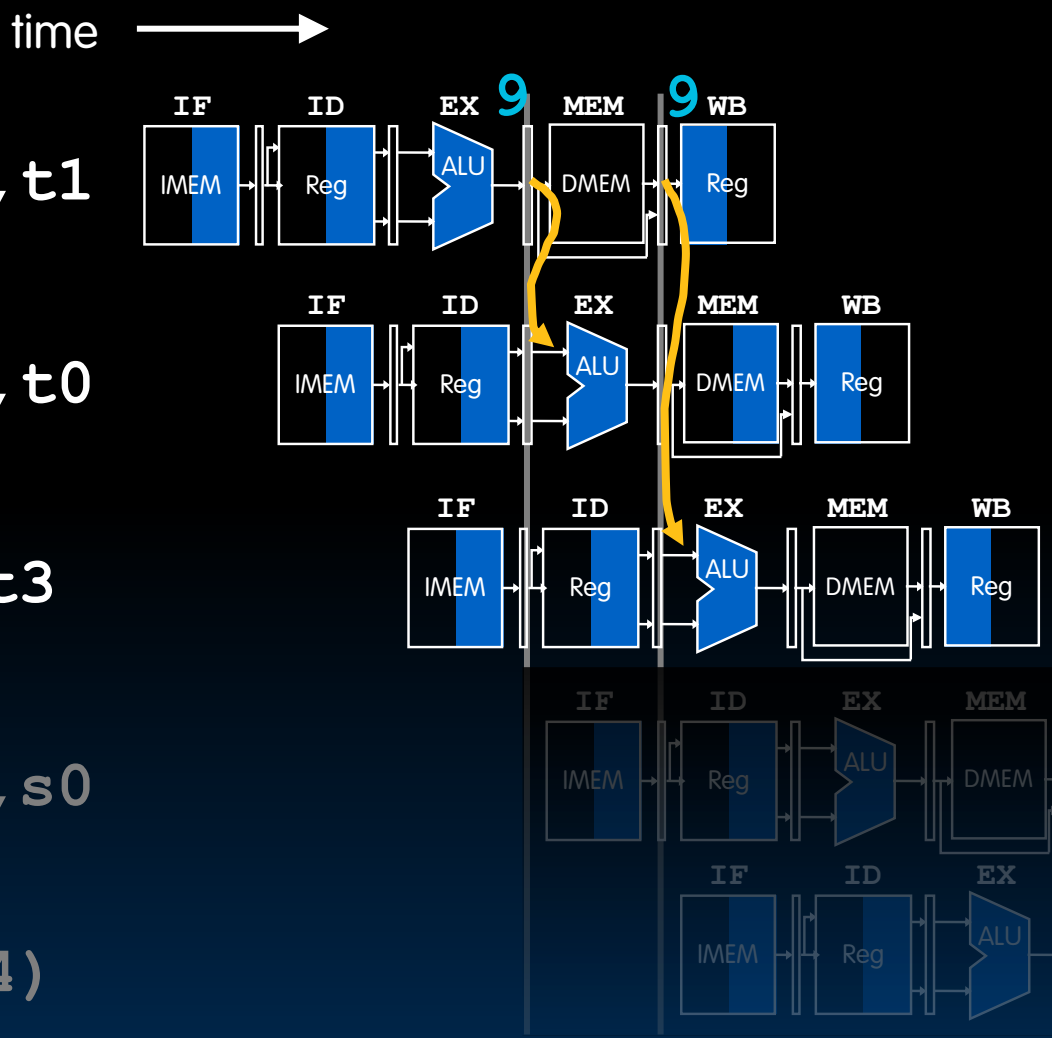
"Bubble" to effectively nop:

- Affected pipeline stages do nothing during clock cycles.
- Stall all stages by preventing `PC`, `IF/ID` pipeline register from writing (more in textbook).

Stalls reduce performance.

- Compiler could rearrange code/insert nops to avoid hazards (and therefore stalls), but this requires knowledge of the pipeline structure.

ALU Solution 2: Forwarding



Forwarding, **aka bypassing, uses the result when it is computed.**

- Don't wait for value to be stored into RegFile.
- Instead, grab operand from the pipeline stage.

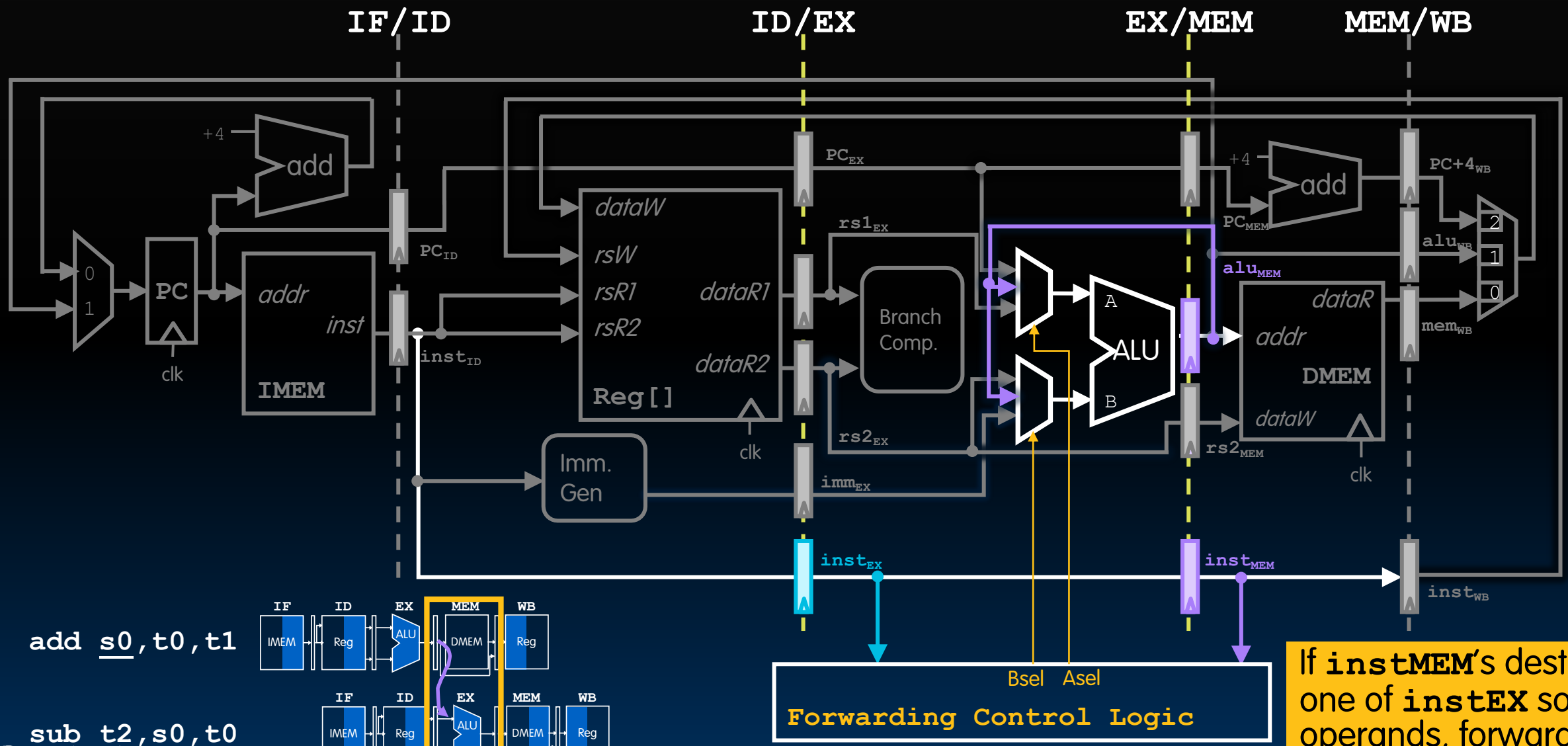
Implementation:

- Make extra connections in the datapath.
- Also add forwarding control logic.

s0 value	5	5	5	5	5/9	9	9	9	9

Forwarding EX output

MEM Forwarding omitted for simplicity; see slide at end.

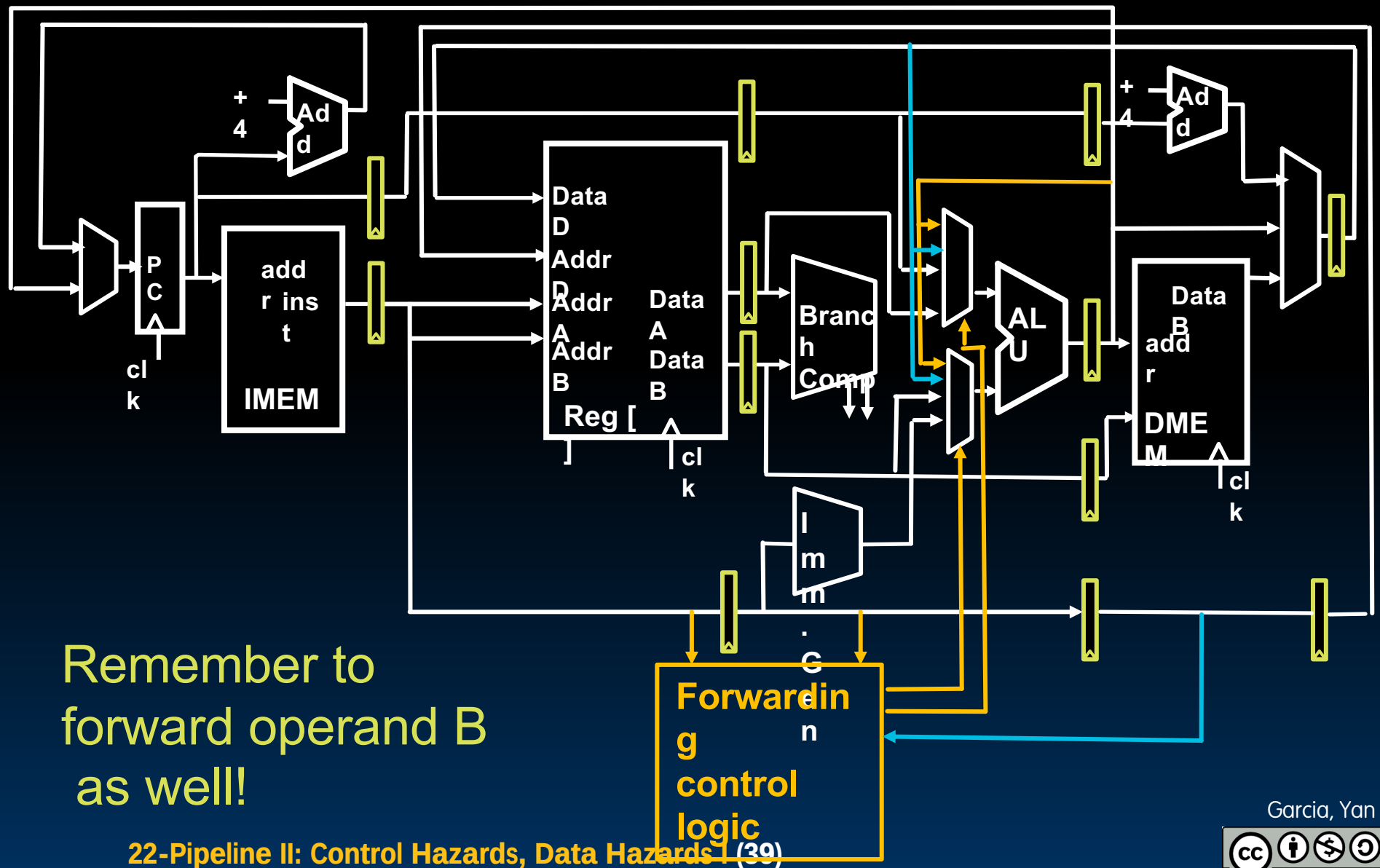



```
add s0,t0,t1
sub t2,s0,t0
```

If **instMEM**'s dest. is one of **instEX** source operands, forward it.

Good luck on the midterm, take-home part!

Pipelined RV32I Datapath



Remember to forward operand B as well!