



UC Berkeley
Teaching Professor
Dan Garcia

CS61C

Great Ideas in Computer Architecture (a.k.a. Machine Structures)



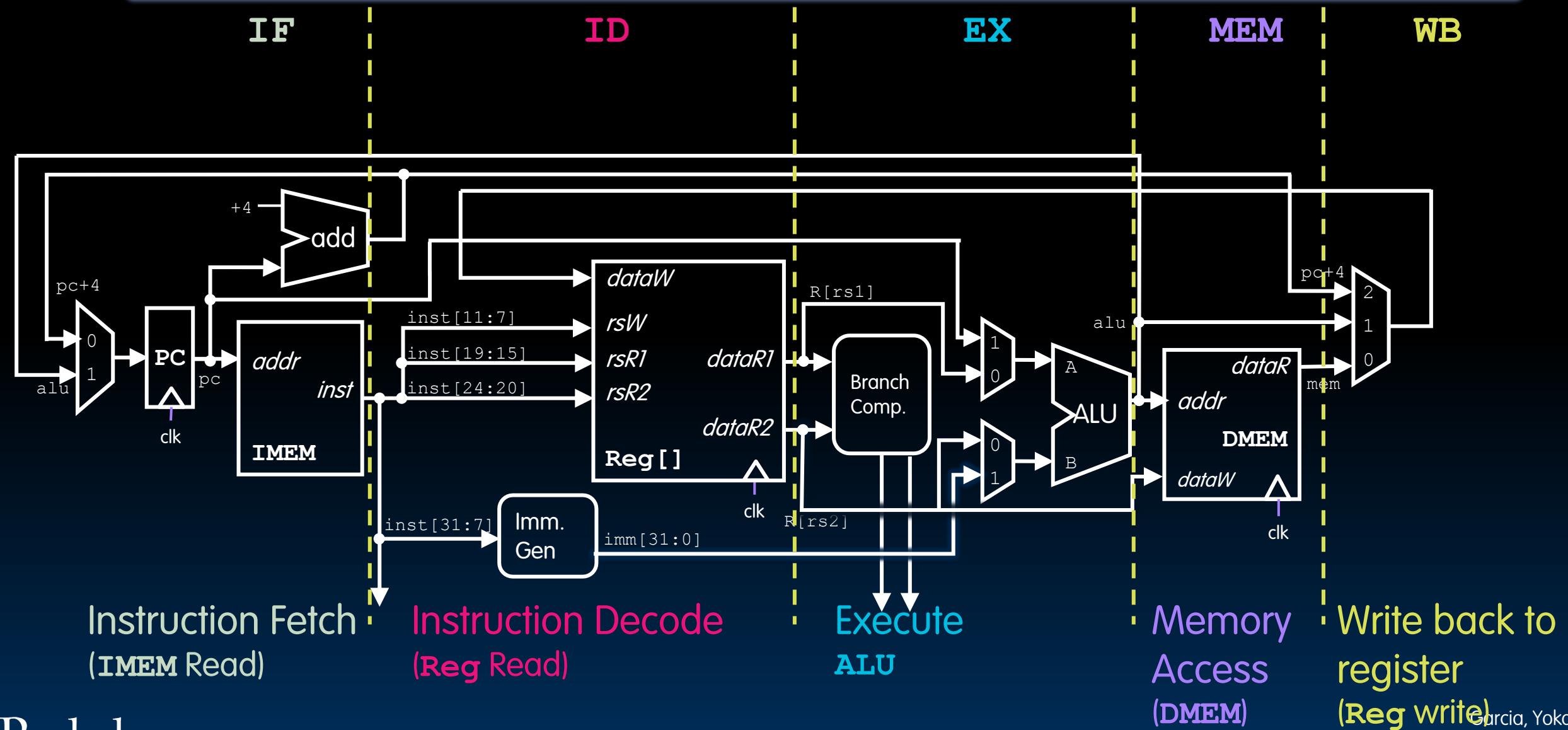
UC Berkeley
Lecturer
Justin Yokota

Pipeline II: Control Hazards, Data Hazards I

RISC-V's 5-Stage Pipeline

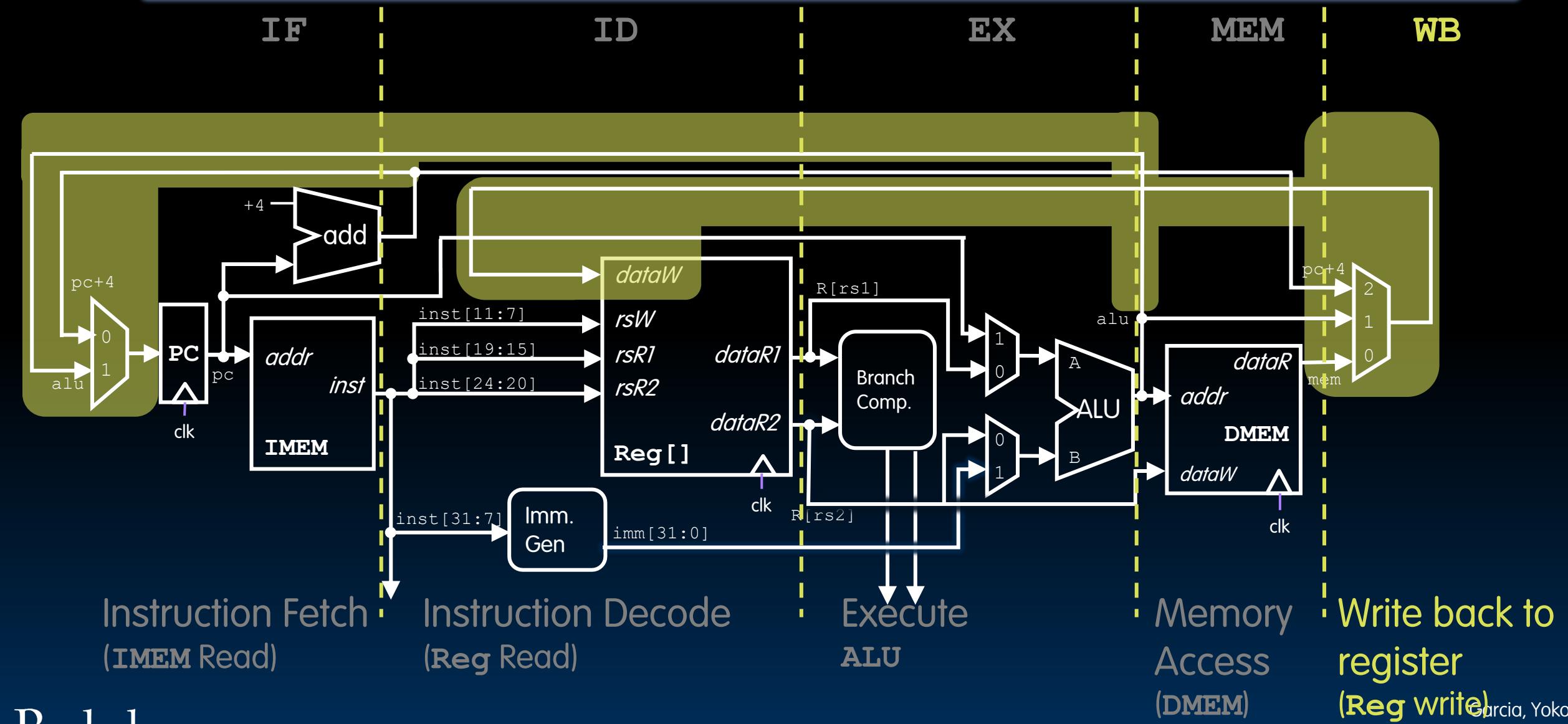
- RISC-V's 5-Stage Pipeline
- Pipelined Datapath and Control
- Structural Hazards
- Data Hazards I (Stalling, Forwarding)

Single-Cycle RV32I Datapath

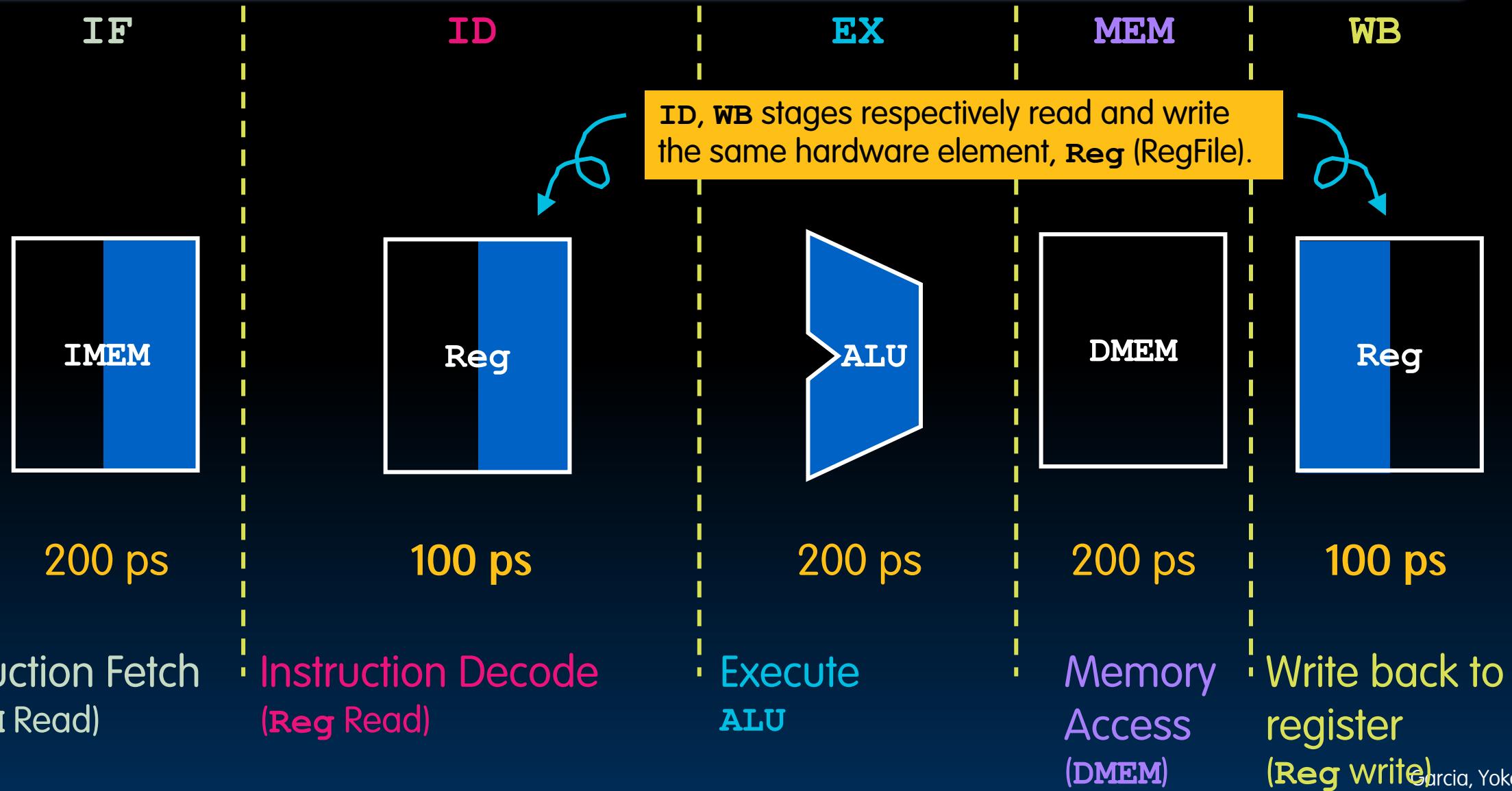


Single-Cycle RV32I Datapath

Review

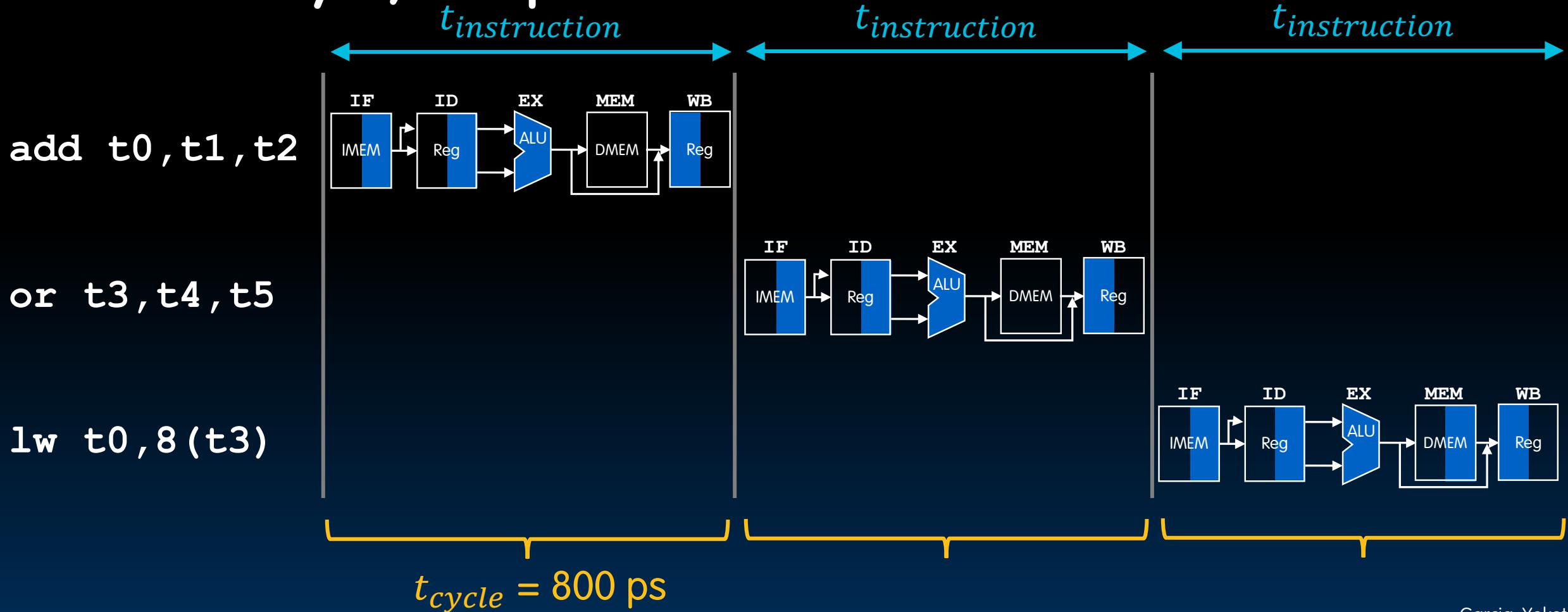


Symbolic Representation of 5 Stages



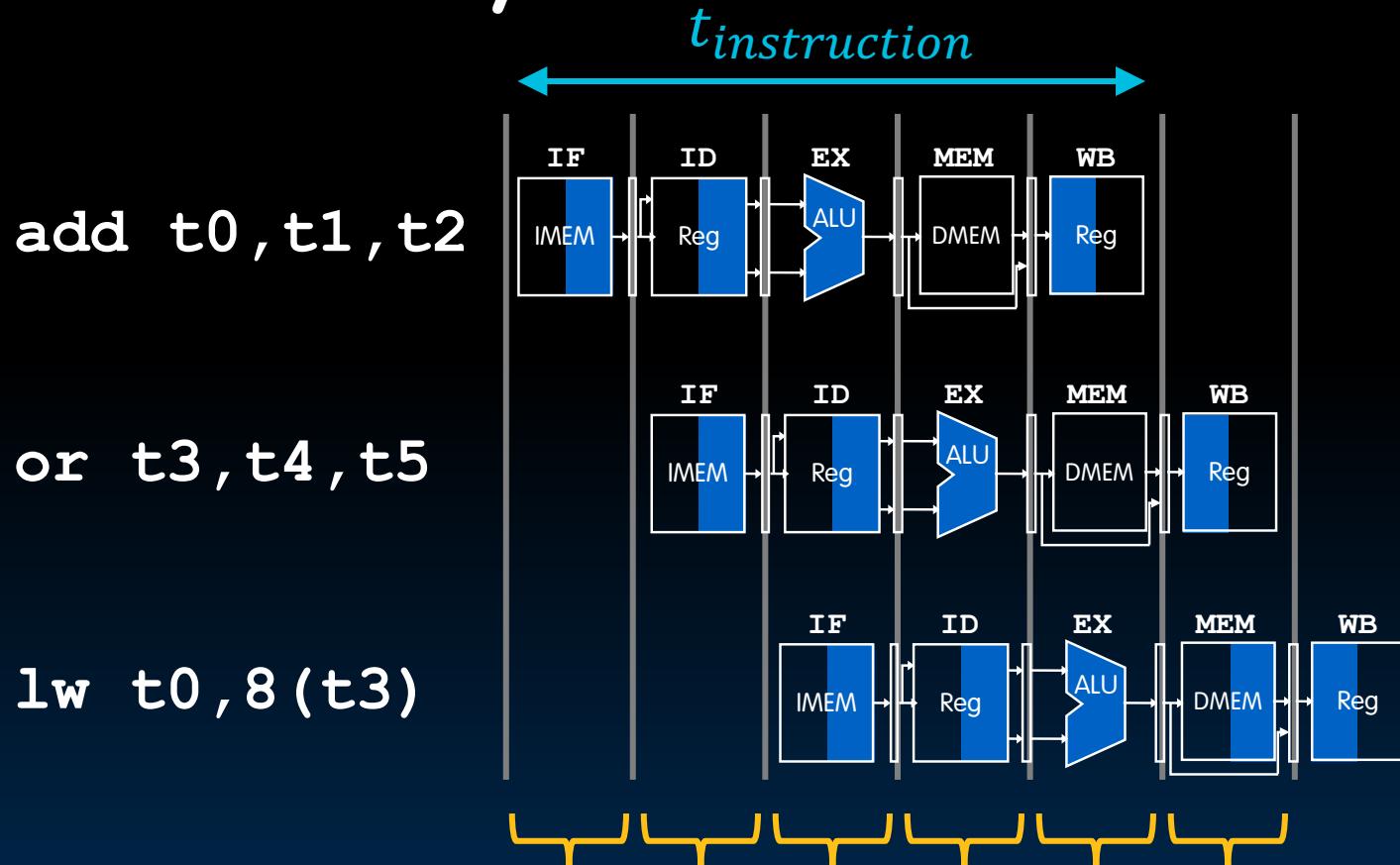
“Sequential” RISC-V Datapath

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

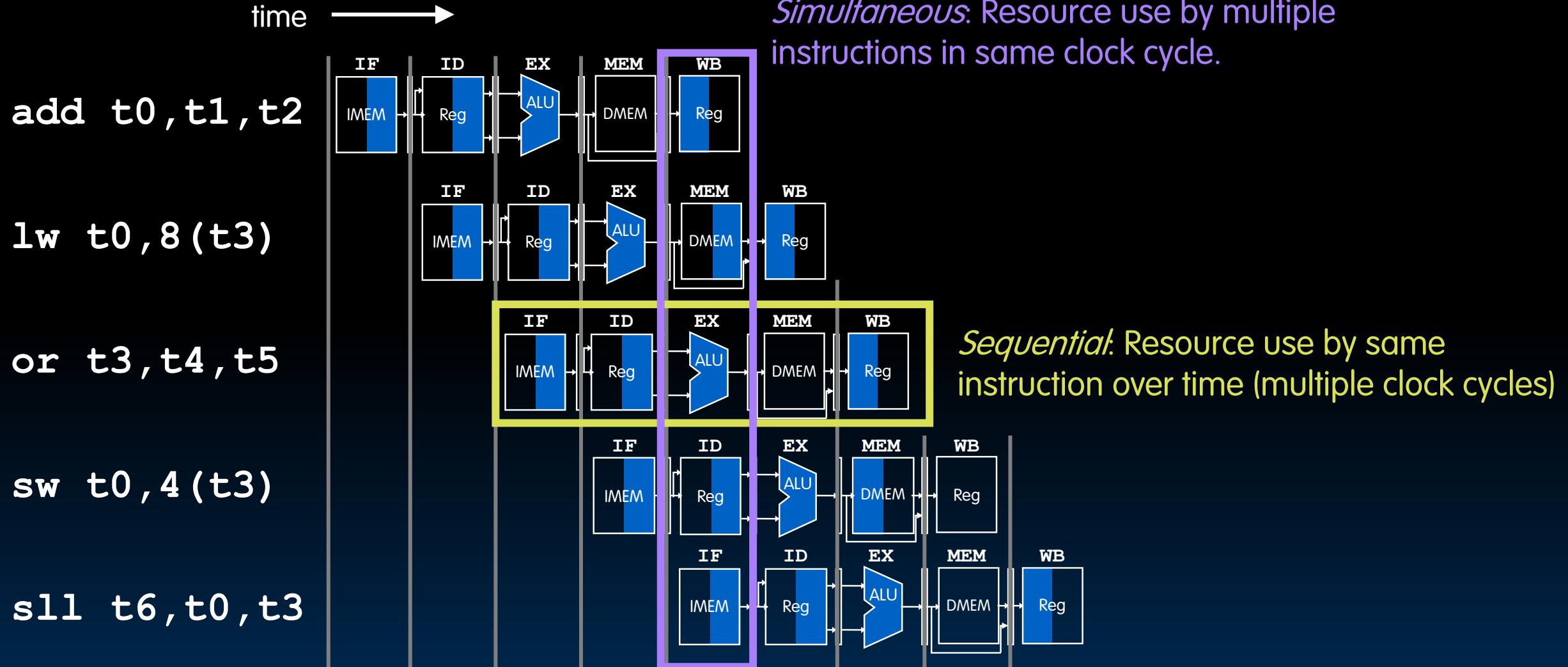


Pipelined RISC-V Datapath

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

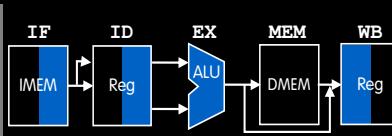


What Happens Sequentially? Simultaneously?

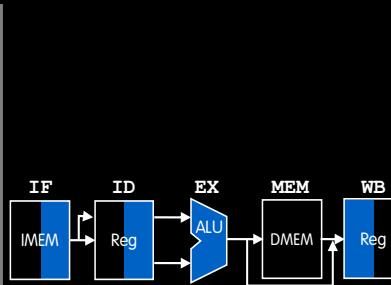


Performance (1/2): Latency

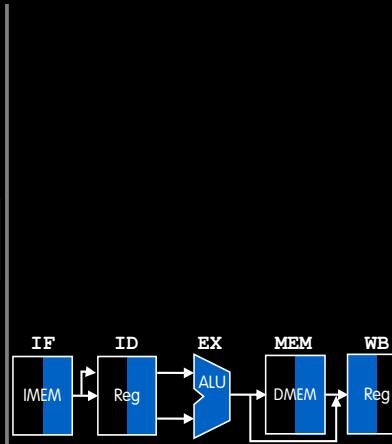
add t0, t1, t2



or t3, t4, t5



lw t0, 8(t3)



Single Cycle

Timing of each stage

 $t_{stage} = 200, 100, 200, 200, 100 \text{ ps}$
 (Reg access stages **ID**, **WB** only 100 ps)

Instruction time (Latency)

 $t_{instruction} = 800 \text{ ps}$

Pipelined

 $t_{stage} = 200 \text{ ps}$
 All stages same length
 $t_{instruction} = 5 \cdot t_{cycle} = 1000 \text{ ps}$

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:

$$\frac{\text{time}}{\text{program}} = \frac{\text{instructions}}{\text{program}} \times \frac{\text{cycles}}{\text{instruction}} \times \frac{\text{time}}{\text{cycle}}$$

CPI, inverse of (# instrs executed/cycle)

1/throughput

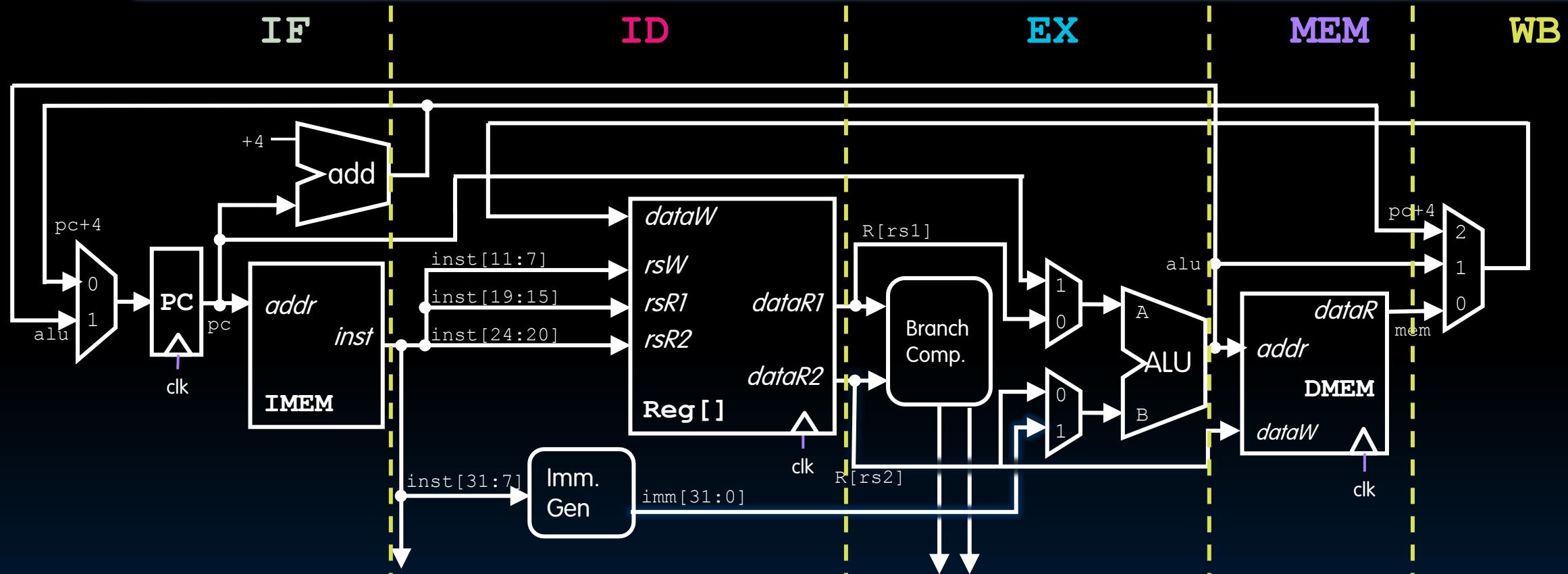
Processor Throughput: $\frac{\# \text{ instructions}}{\text{time}}$

	Single Cycle	Pipelined
Timing of each stage	$t_{stage} = 200, 100, 200, 200, 100 \text{ ps}$ (Reg access stages ID , WB only 100 ps)	$t_{stage} = 200 \text{ ps}$ All stages same length
Instruction time (Latency)	$t_{instruction} = 800 \text{ ps}$	$t_{instruction} = 5 \cdot t_{cycle} = 1000 \text{ ps}$
Clock cycle time, t_{cycle}	$t_{cycle} = t_{instruction} = 800 \text{ ps}$	$t_{cycle} = t_{stage} = 200 \text{ ps}$
Clock rate, $f_s = 1/t_{cycle}$	$f_s = 1/800 \text{ ps} = 1.25 \text{ GHz}$	$f_s = 1/200 \text{ ps} = 5 \text{ GHz}$
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

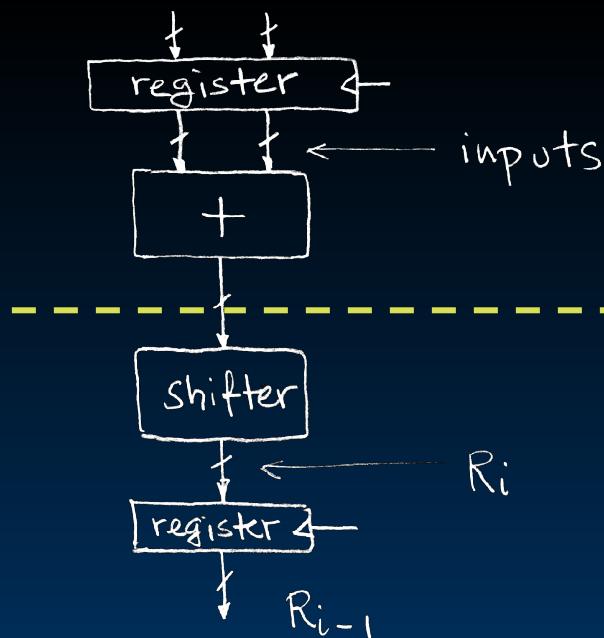


- A pipelined datapath needs to “separate” the five stages of the RV32I datapath.
 - Each stage needs to process data from a different instruction.

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

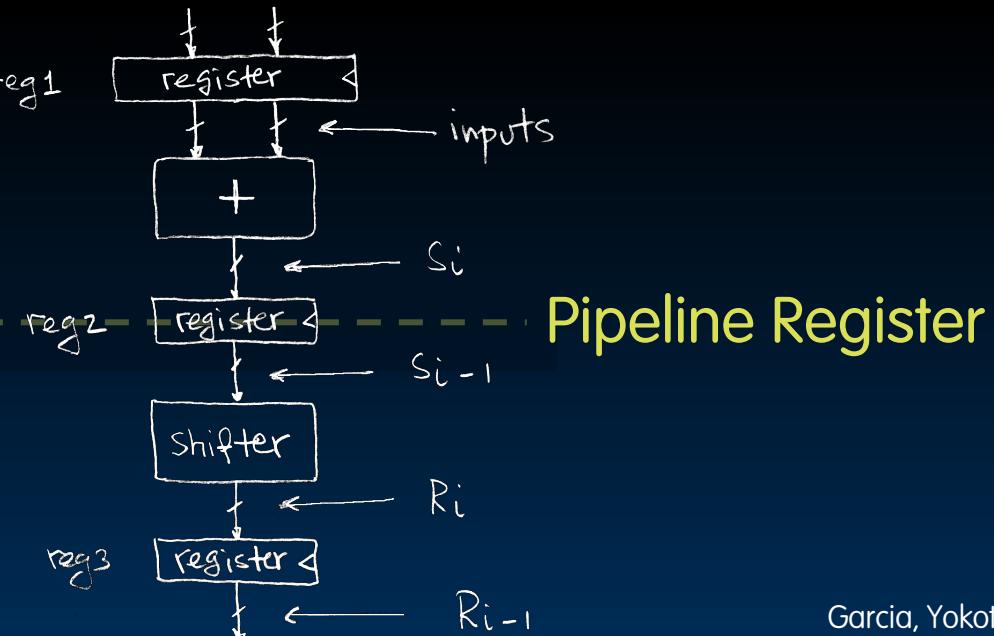
Single-cycle datapath means
1 clock cycle, from input to output.

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

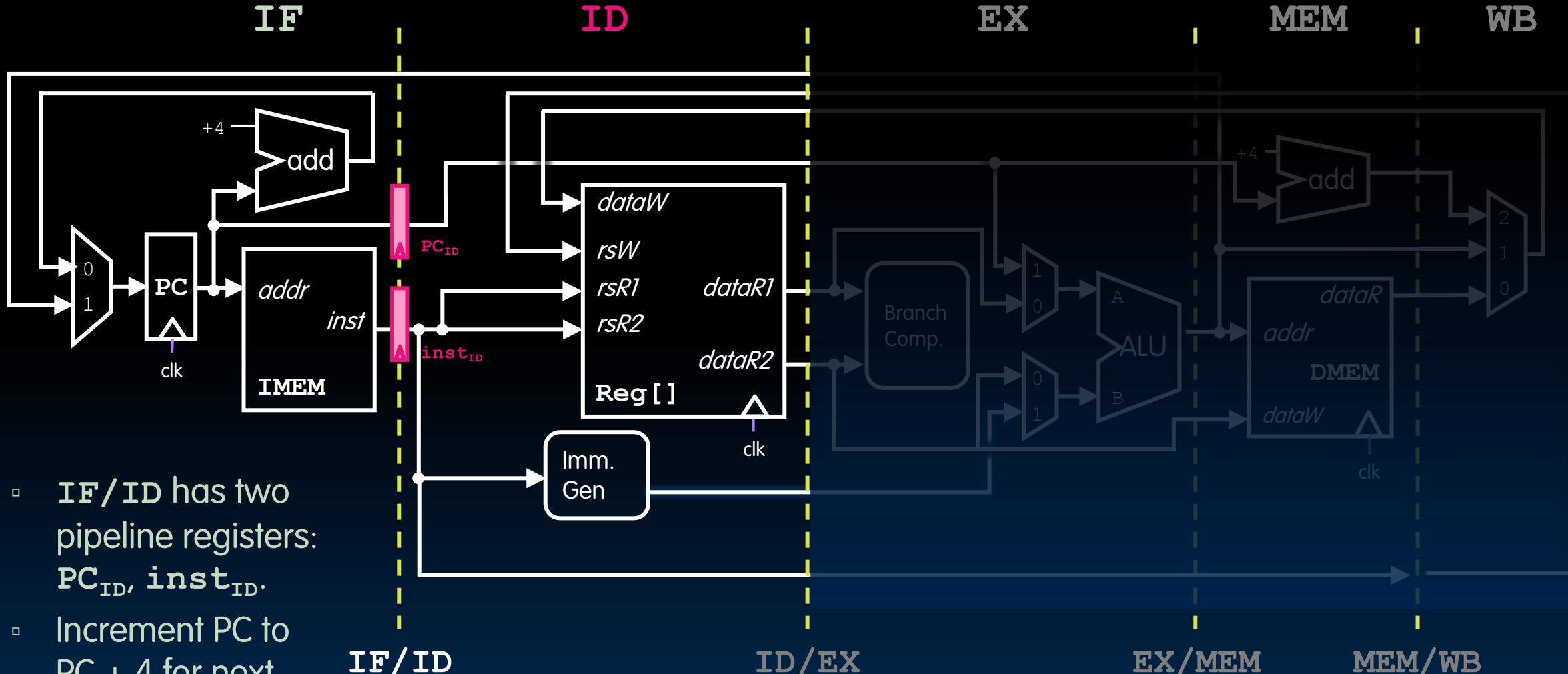


Insertion of pipeline register allows higher clock frequency.

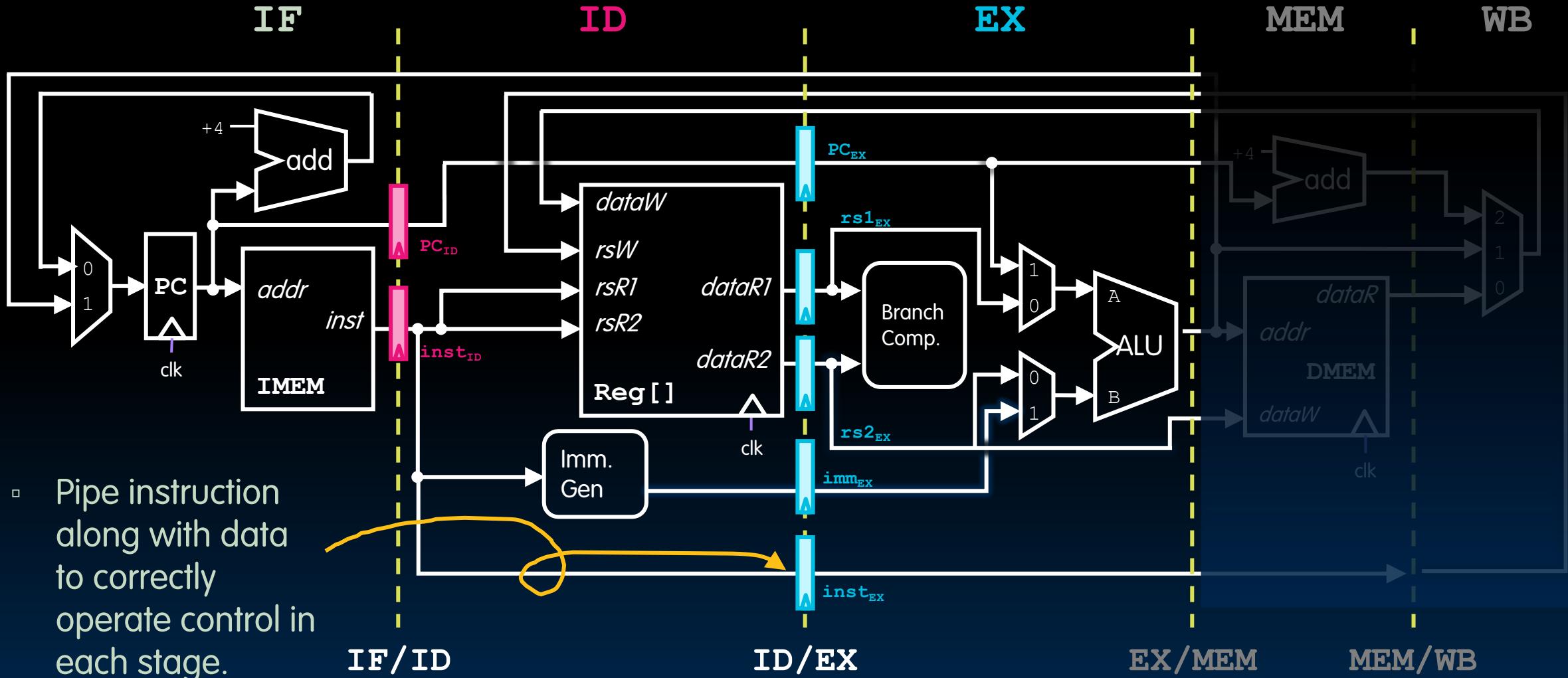
- Clock period now limited by $\max\{\text{adder/shifter prop. delays}\}$.
- Higher throughput (outputs/s).



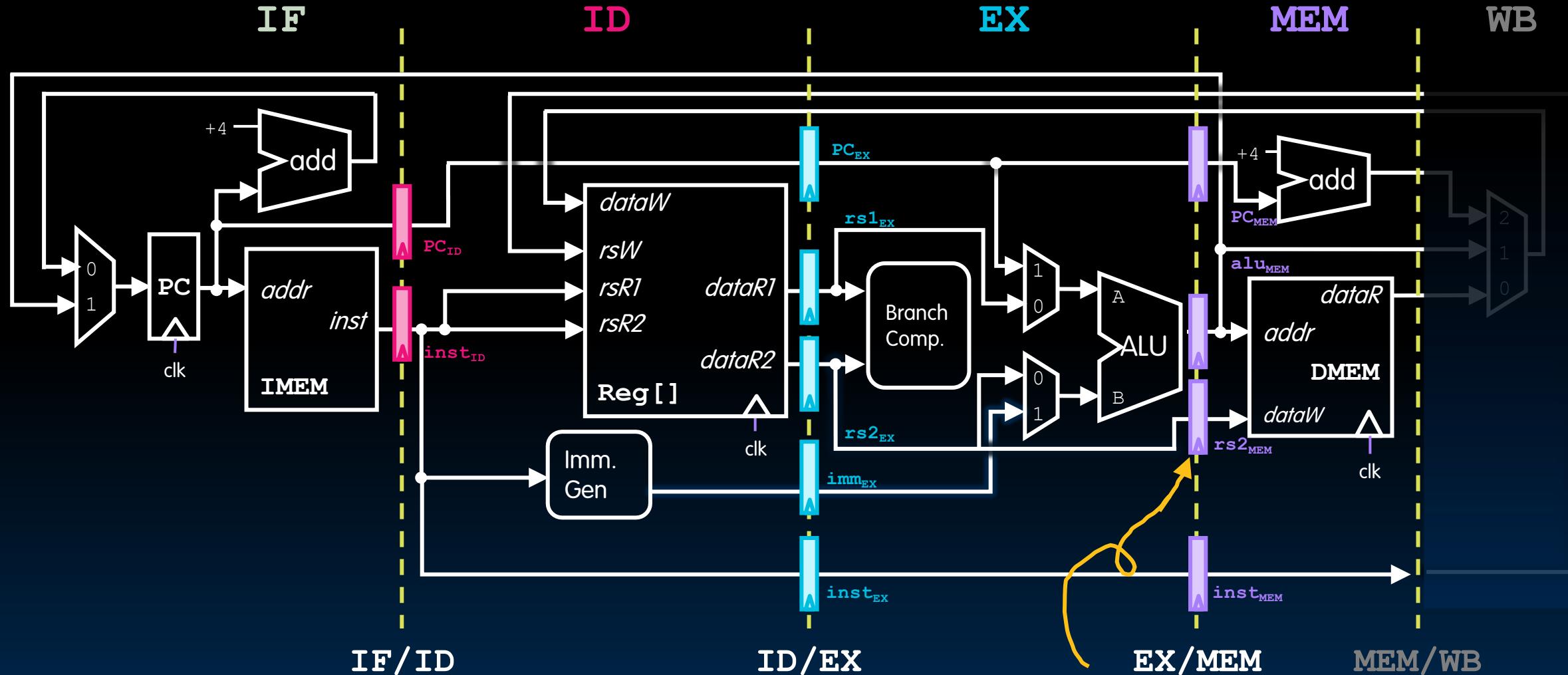
IF/ID Pipeline Registers



ID/EX Pipeline Registers



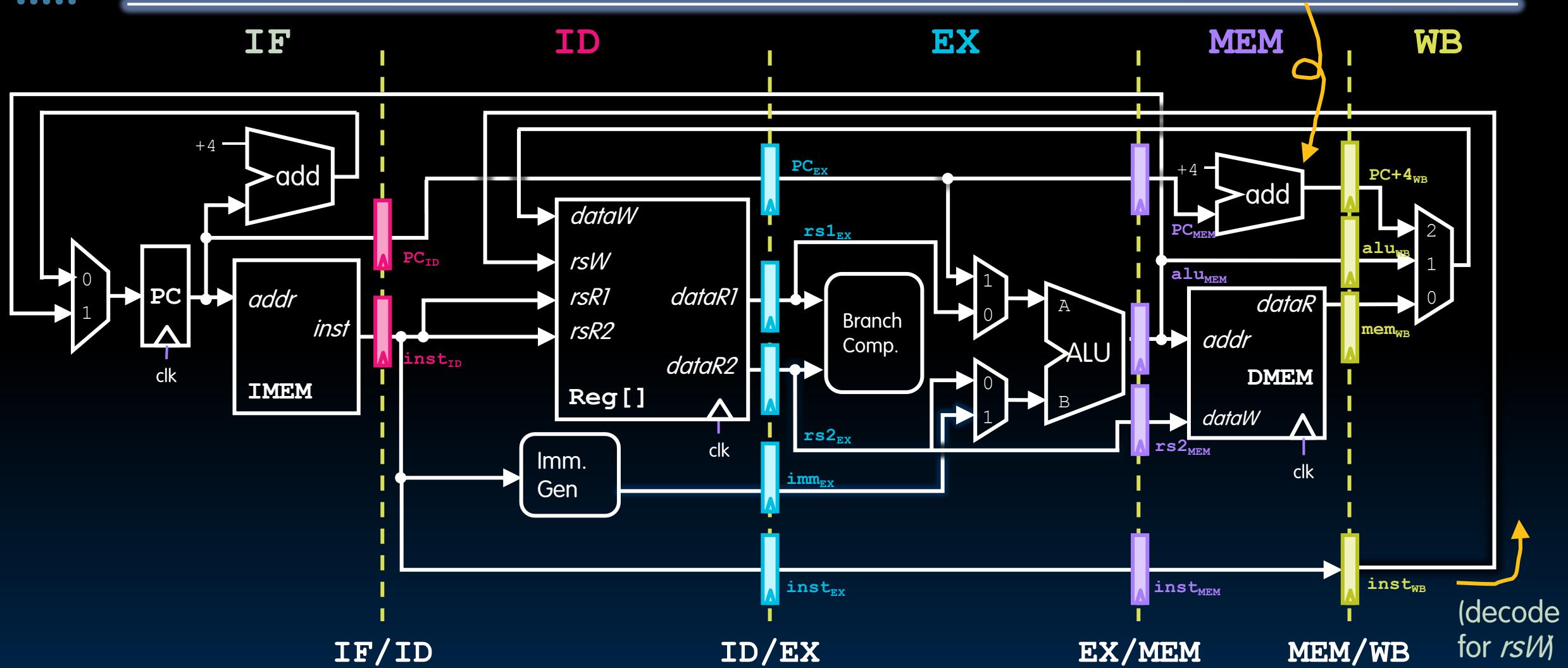
EX/MEM Pipeline Registers



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

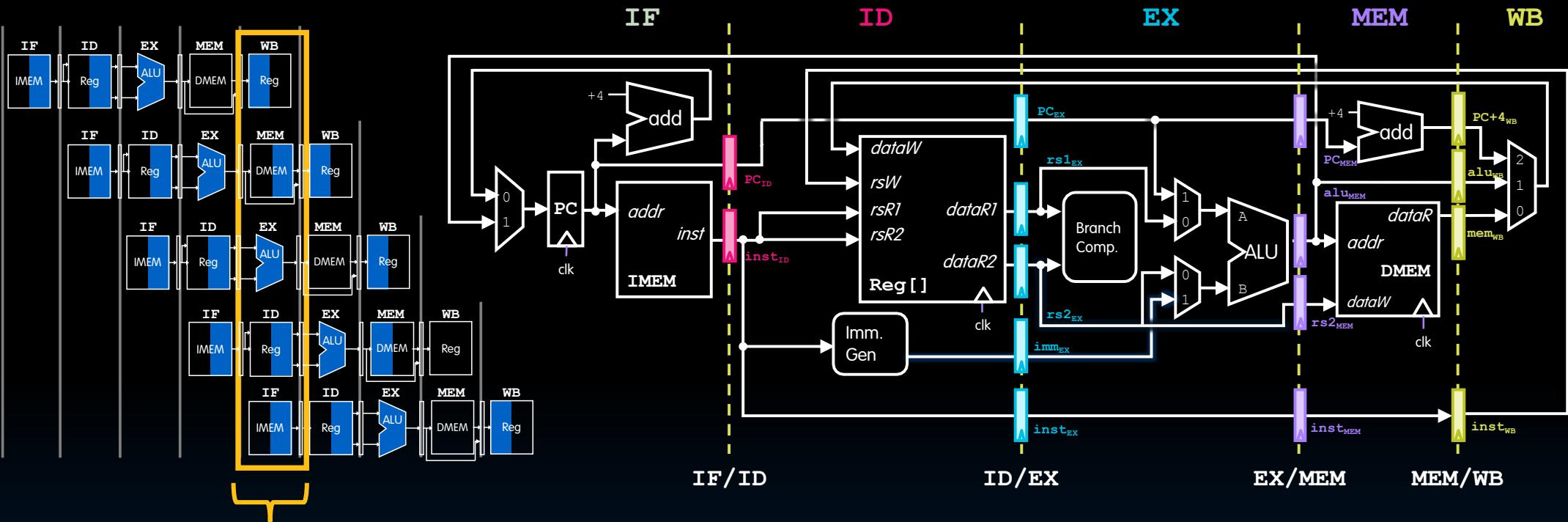
MEM/WB Pipeline Registers

Recalculate $\text{PC}+4$ to avoid sending both PC , $\text{PC}+4$ down pipeline.



Matching Question

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

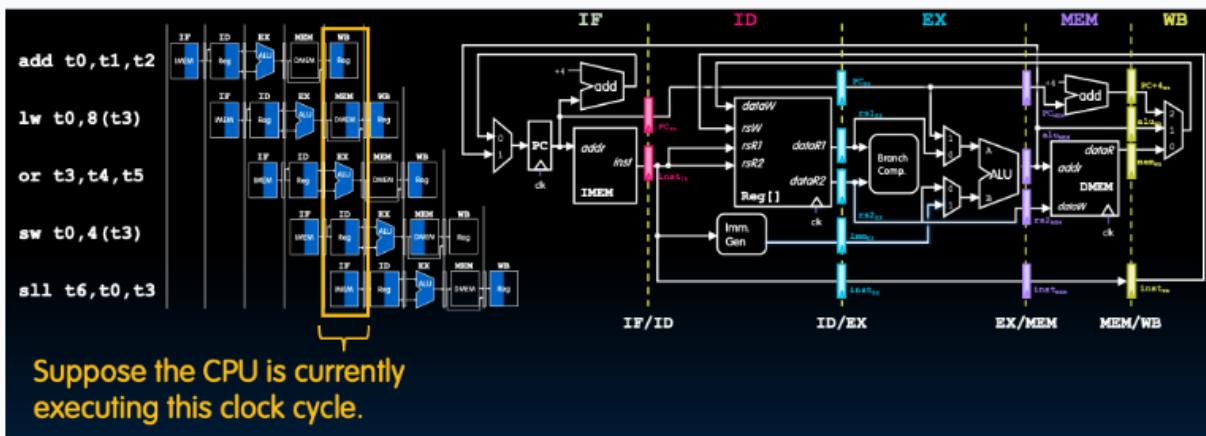


Suppose the CPU is currently executing this clock cycle.

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

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



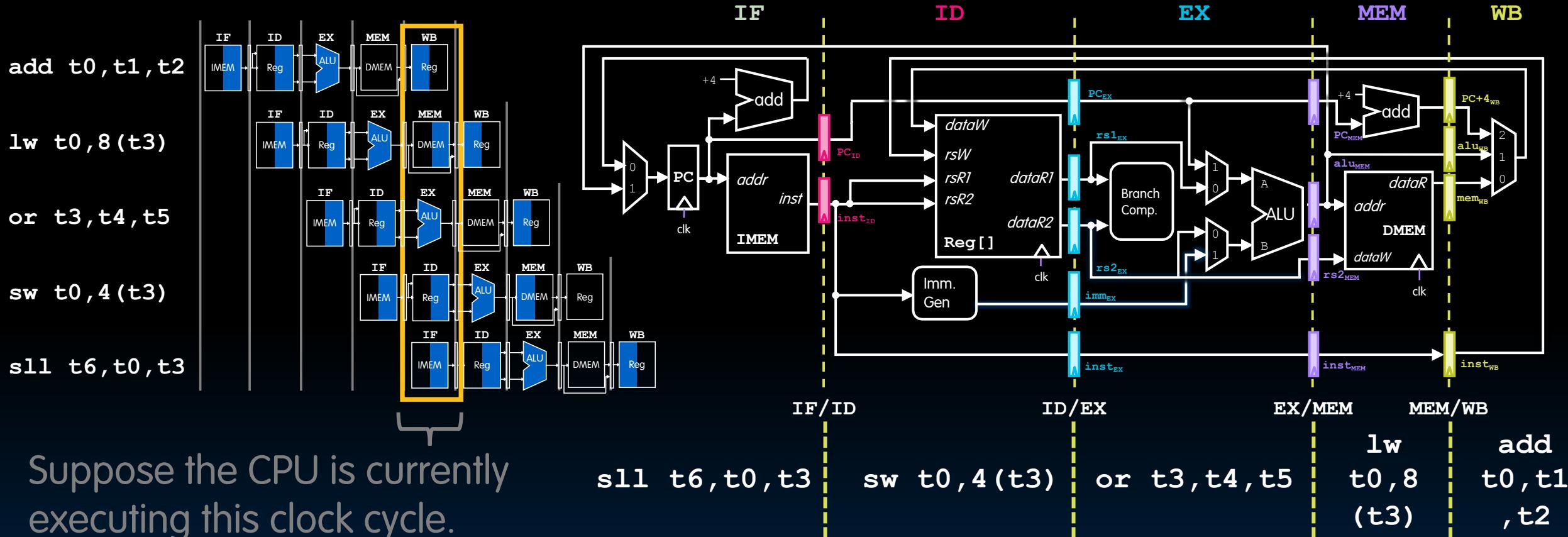
add, lw, or,
sw, sll

sll, sw, or,
lw, add

Something
else

Total Results: 0

Matching Solution

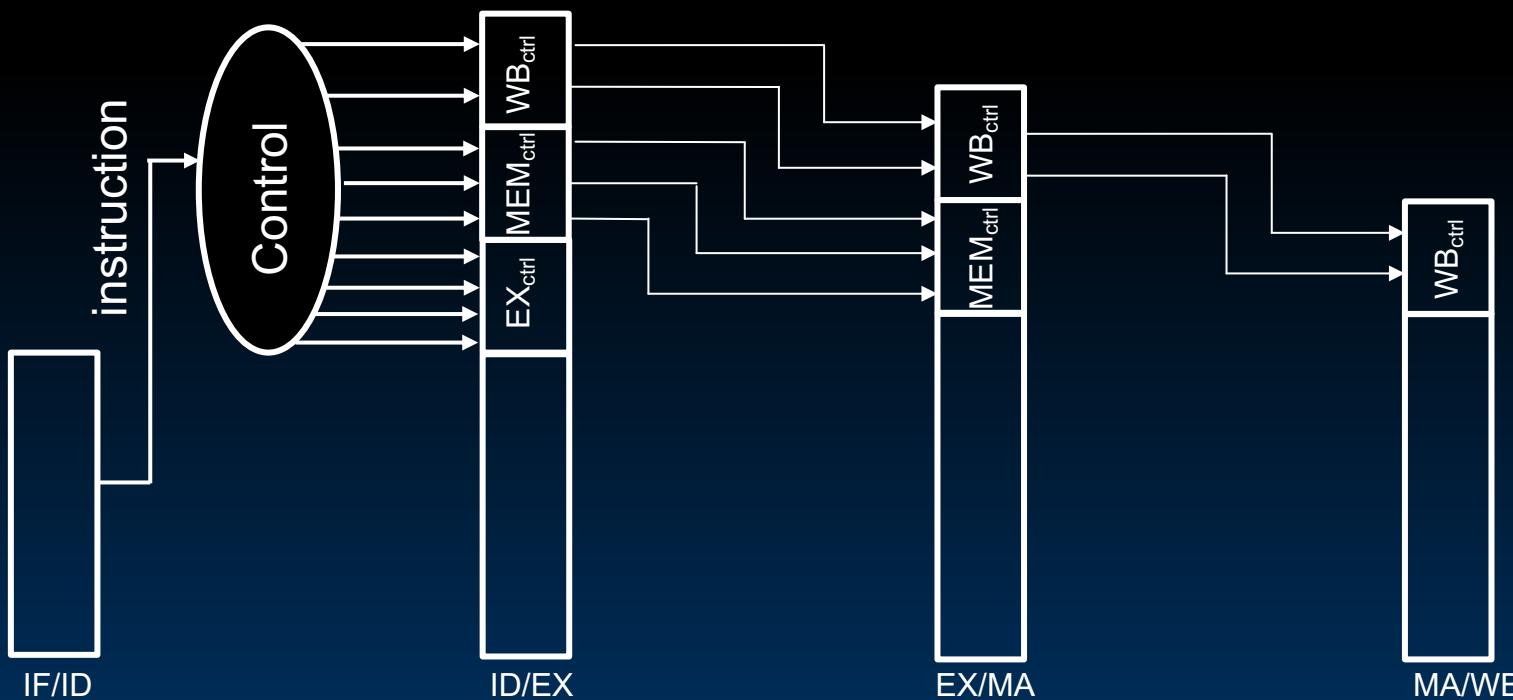


Suppose the CPU is currently executing this clock cycle.

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)

Garcia, Yokota



Structural Hazards

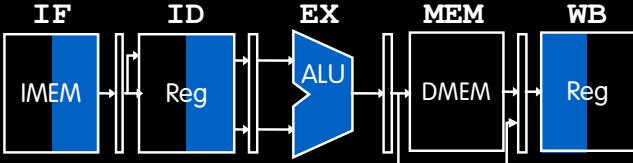
- RISC-V's 5-Stage Pipeline
- Pipelined Datapath and Control
- Structural Hazards
- Data Hazards I
(Stalling, Forwarding)

Pipeline Hazards Ahead!!!

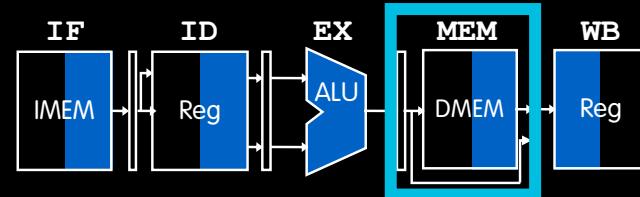
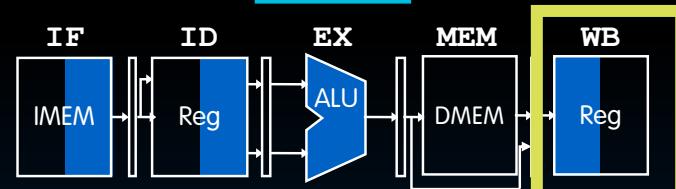
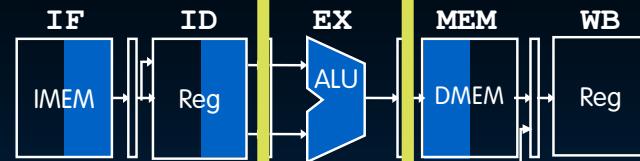
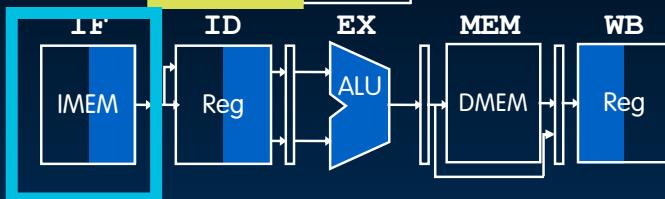


Pipeline Hazards Ahead!!!

time →

add t0, t1, t2

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

lw t0, 8(t3)**or t3, t4, t5****sw t0, 4 (t3)****sll t6, t0, t3**

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 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*.

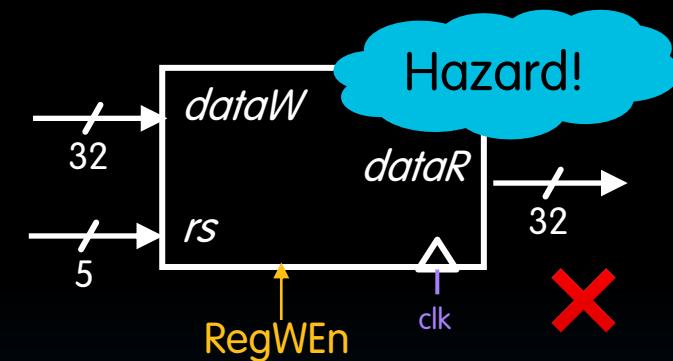


The RV32I ISA datapath avoids structural hazards via its hardware requirements on RegFile and Memory.

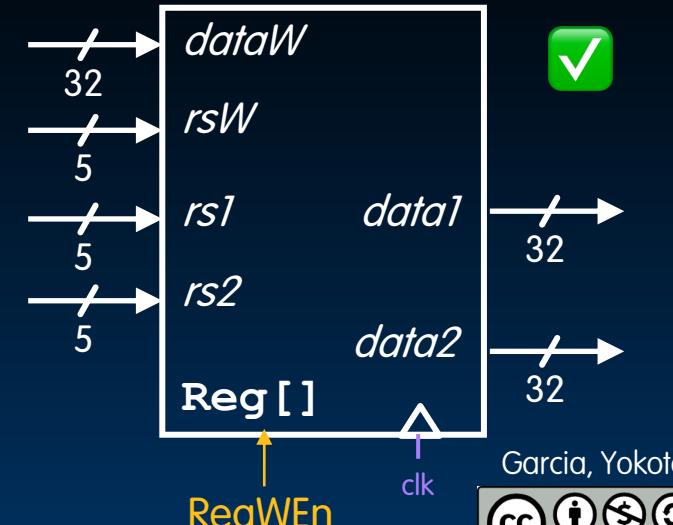
Required RegFile Avoids Structural Hazards

- **Each RV32I instruction:**
 - Reads up to 2 operands in ID (decode) stage; and
 - Writes up to 1 operand in WB (writeback) stage.
- **Structural hazard can occur if RegFile HW does not support simultaneous read/write!**

add t0 , t1 , t2



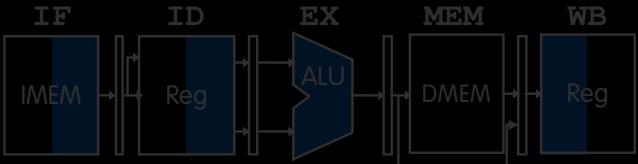
- **RV32I's required RegFile design works:**
 - Two independent read ports, one independent write port.
 - *Three accesses (2 read, 1 write)* can happen in the same cycle.



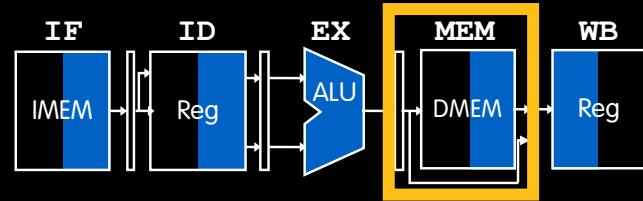
Separate IMEM, DMEM Avoids Structural Hazards

time →

add t0, t1, t2



lw t0, 8(t3)



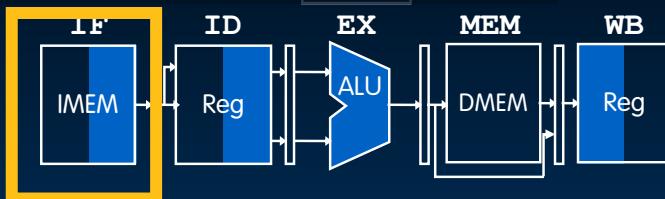
or t3, t4, t5



sw t0, 4(t3)



sll t6, t0, t3



CPU can read memory twice in the same cycle:

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

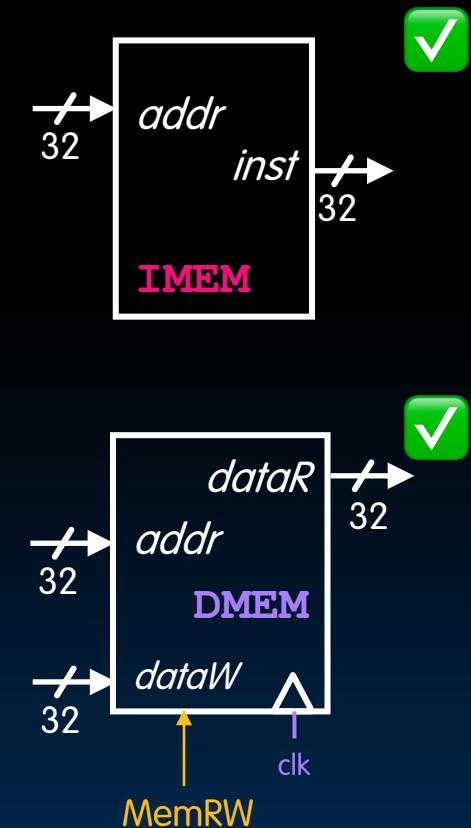
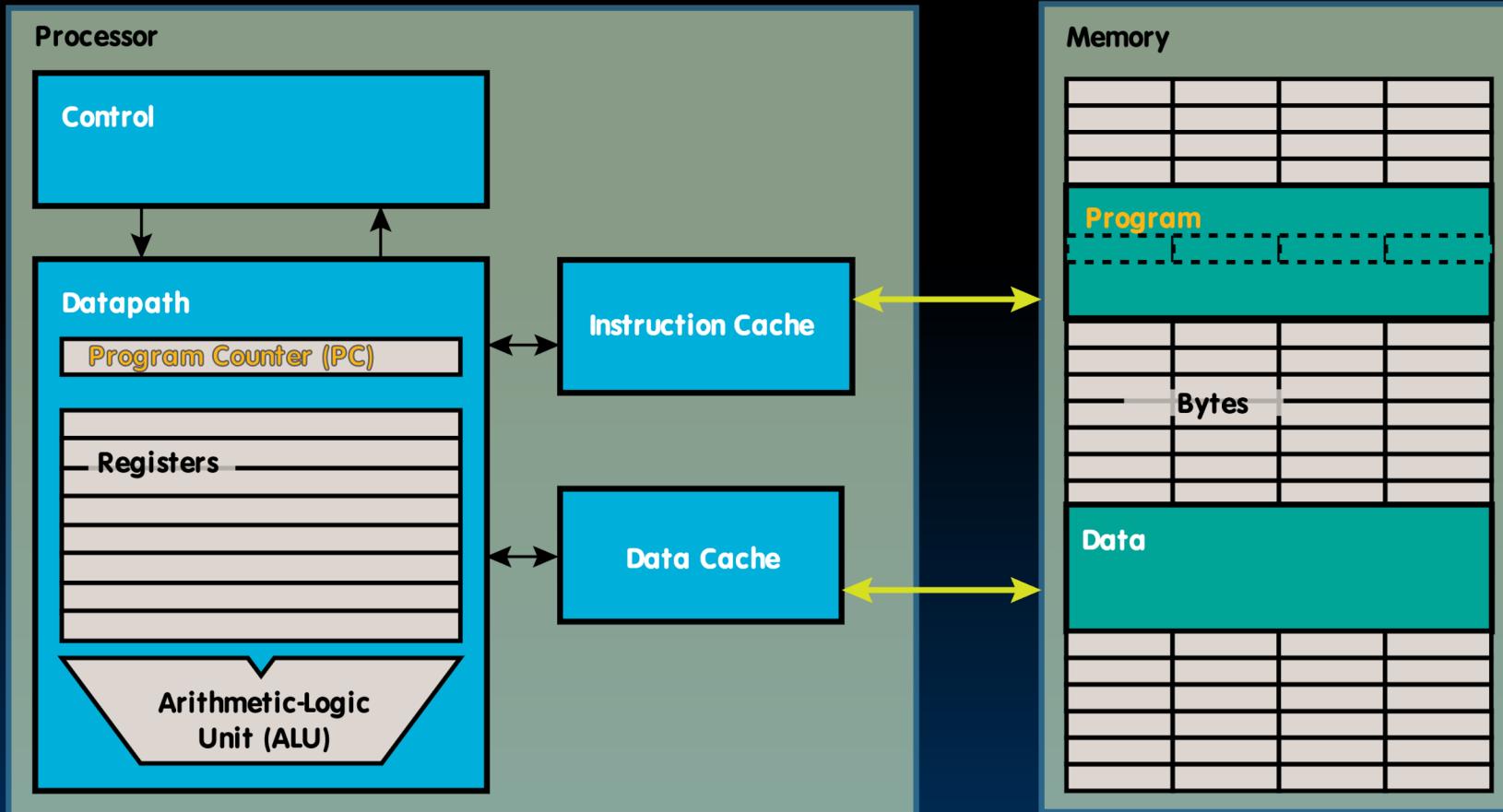
Structural hazard if IMEM, DMEM were same hardware:

- 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)

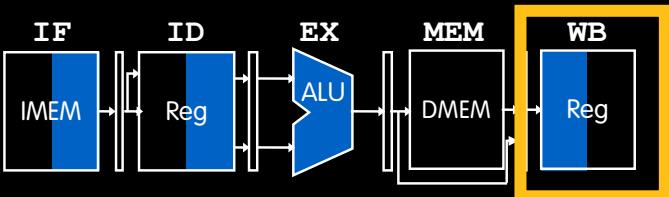
- Data hazard:
 - Instructions have data dependency.
 - Need to wait for previous instruction to complete its data read/write.
- **Occurs when an instruction reads a register before a previous instruction has finished writing to that register.**

Three cases to consider:

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

Data Hazard 1: Register Access

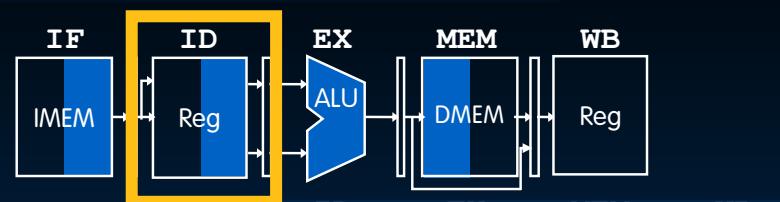
time →

add t0, t1, t2

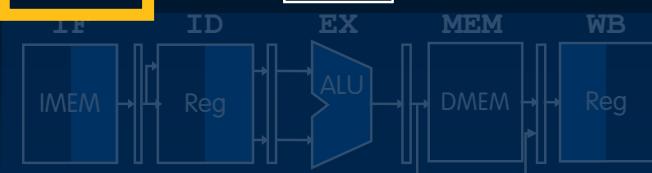
lw t0, 8(t3)



or t3, t4, t5

sw t0, 4(t3)

sll t6, t0, t3



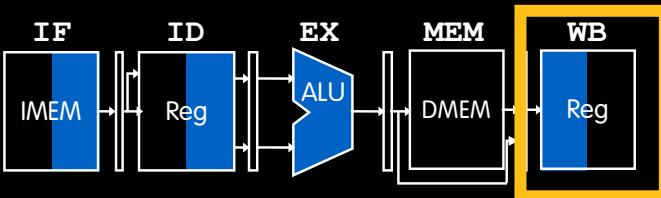
If the same register is written and read in one cycle:

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

Register Access, Fixed with HW Requirement

time →

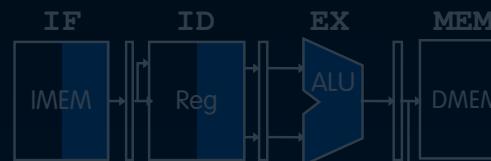
add t0, t1, t2



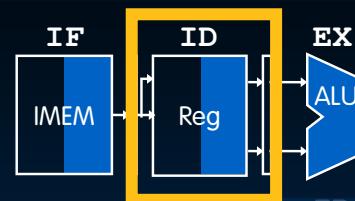
lw t0, 8(t3)



or t3, t4, t5



sw t0, 4(t3)



sll t6, t0, t3



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.

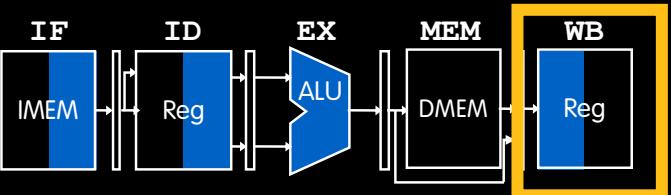
Garcia, Yokota



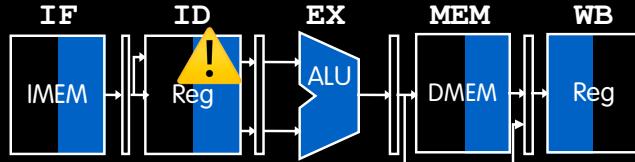
Data Hazard 2: ALU Result

time →

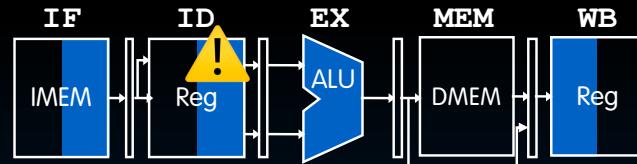
add s0, t0, t1



sub t2, s0, t0



or t6, s0, t3



xor t5, t1, s0



sw s0, 4(t4)



s0 value

5	5	5	5	5/9	9	9	9	9
---	---	---	---	-----	---	---	---	---

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

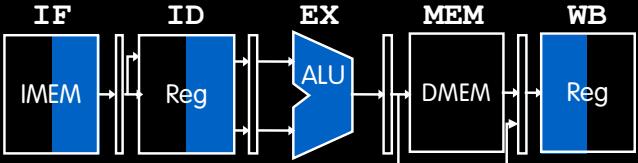
- sub, or's ID reads old value of s0 and calculates wrong result.

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

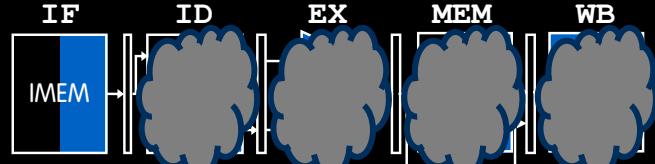
ALU Solution 1: Stalling

time →

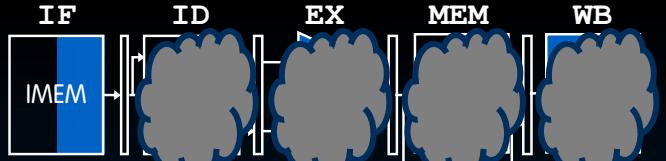
add s0, t0, t1



sub → nop



sub → nop



sub t2, s0, t0



or t6, s0, t3



s0 value

5	5	5	5	5/9	9	9	9	9
---	---	---	---	-----	---	---	---	---

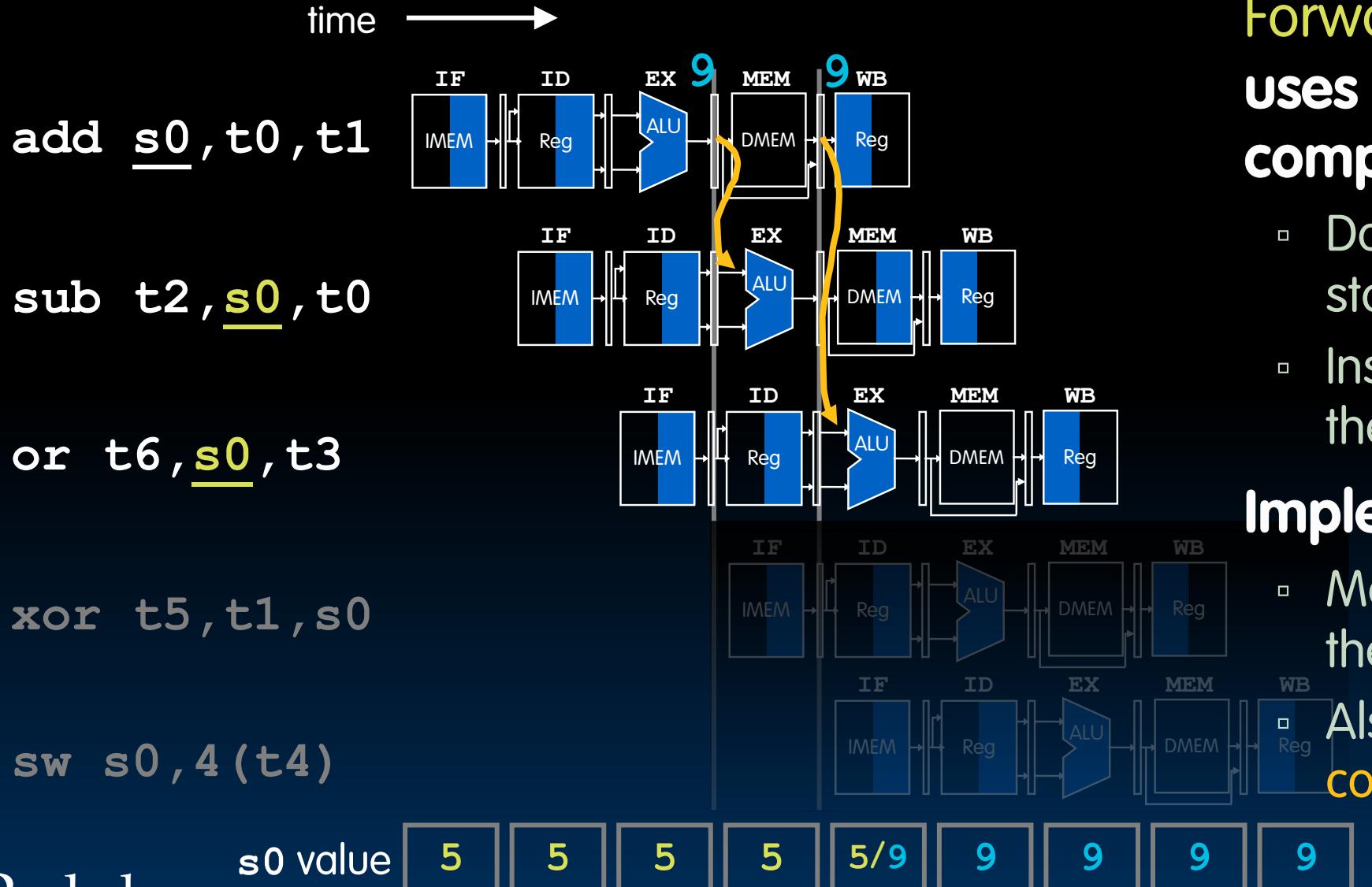
“Bubble” to effectively nop:

- 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**.

Forwarding EX output

MEM Forwarding omitted for simplicity (see book for more)

