

CS61C: Great Ideas in Computer Architecture (Machine Structure)

Instructors: Rosalie Fang, Charles Hong, Jero Wang

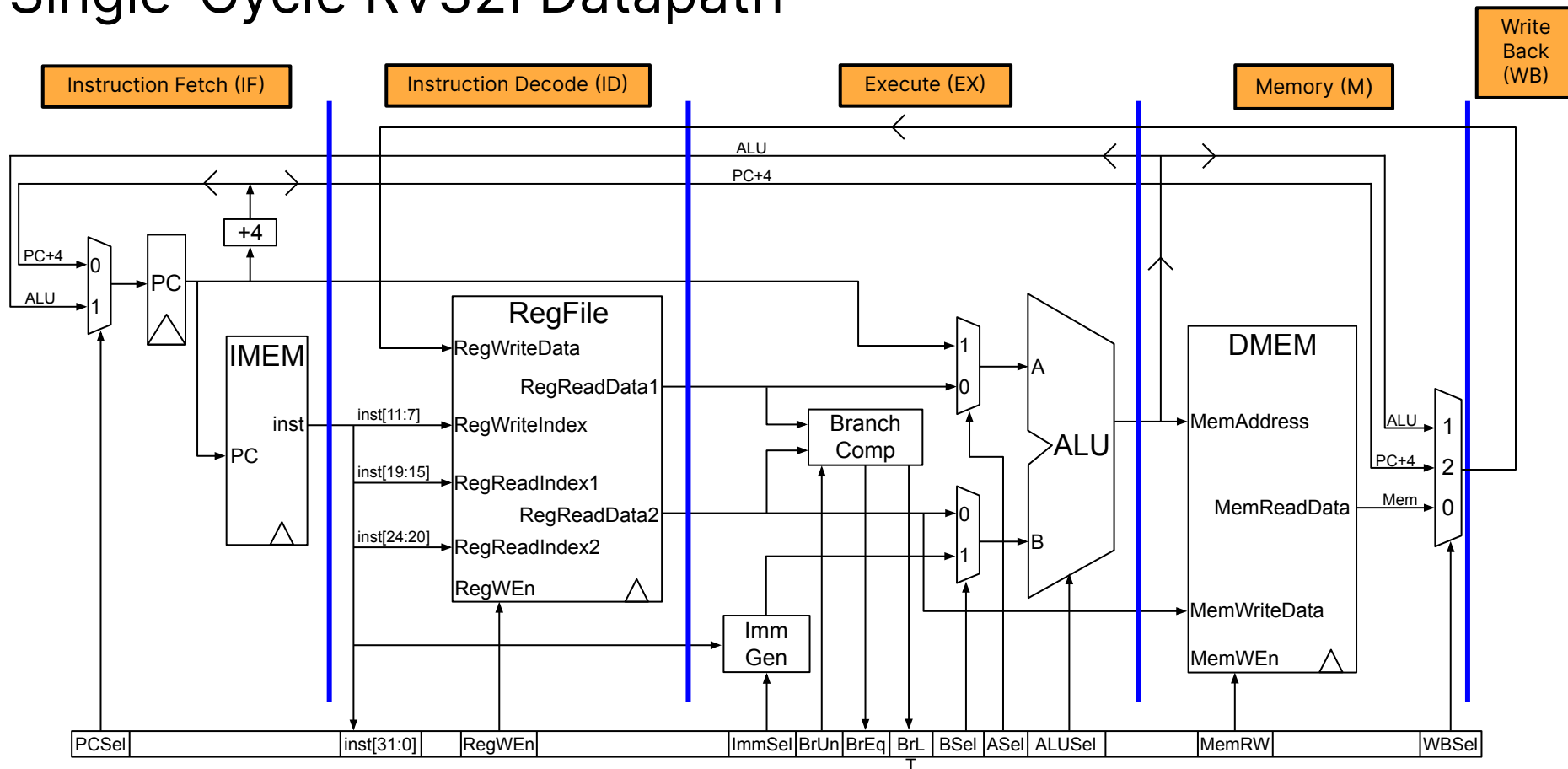
Lecture feedback:

<https://tinyurl.com/fyr-feedback>

Pipelining

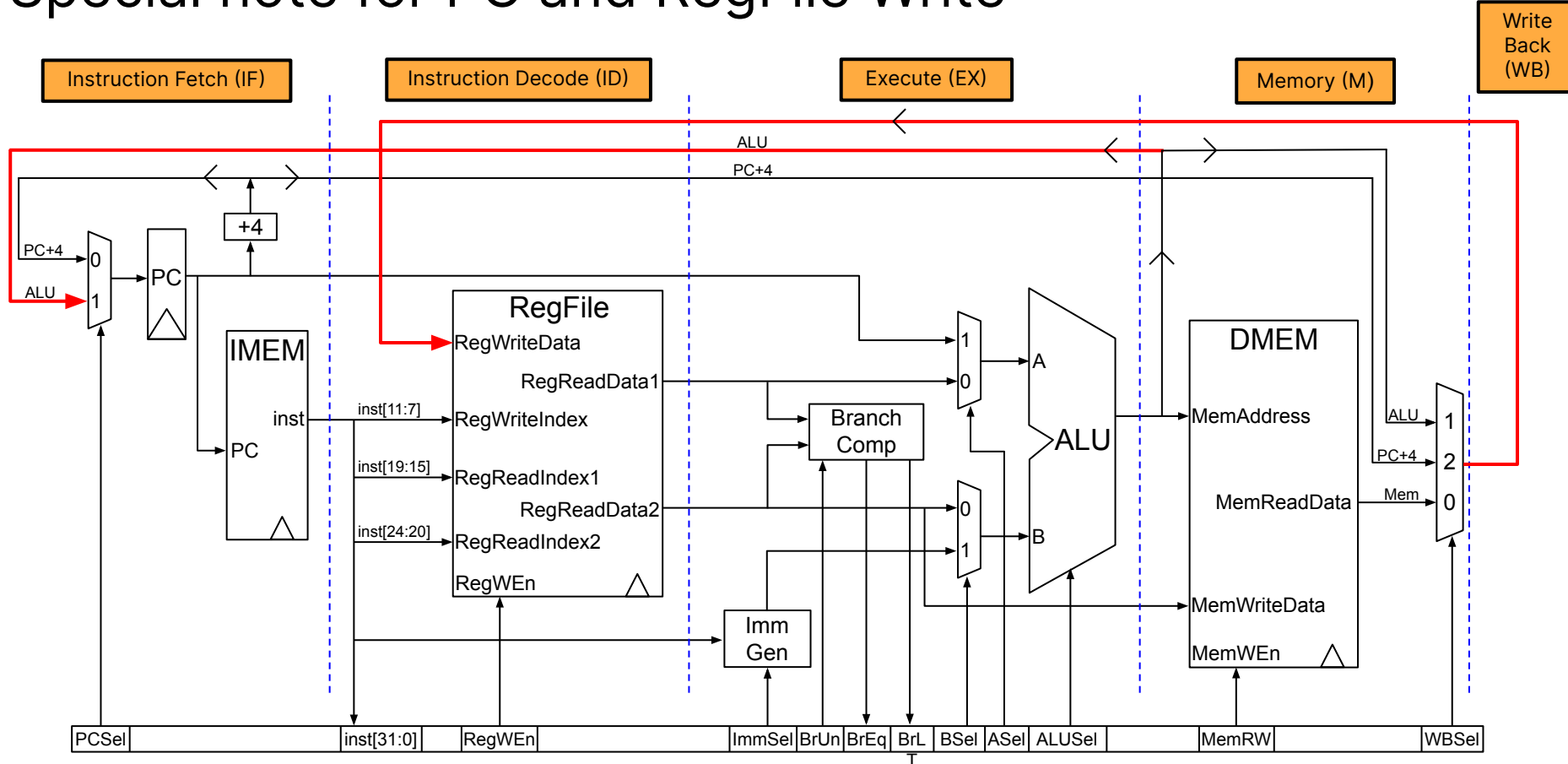
Single-Cycle RV32I Datapath

Review



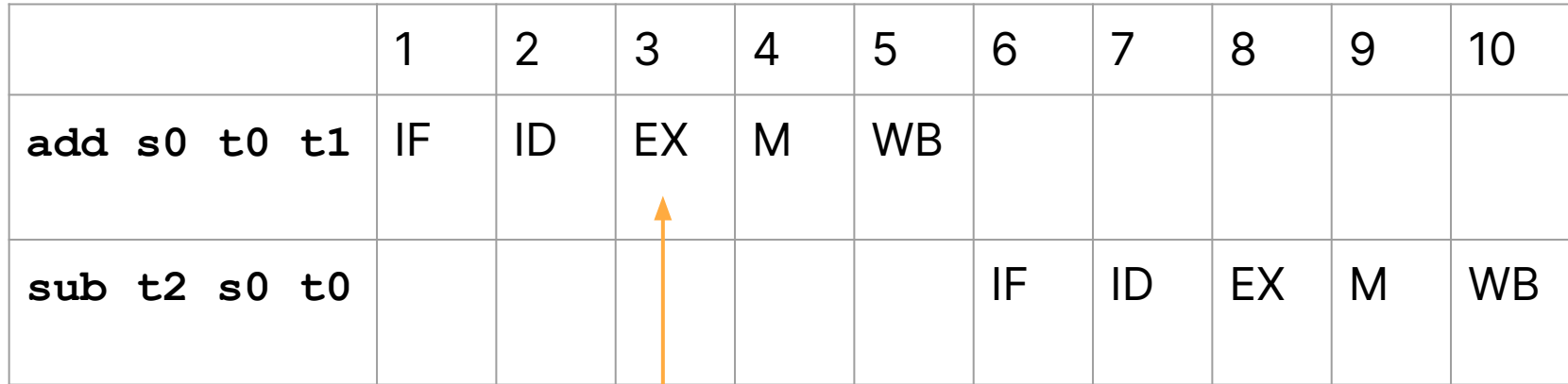
Special note for PC and RegFile Write

Review



Pipeline Timing Diagrams

Numbers represent time steps



	1	2	3	4	5	6	7	8	9	10
<code>add s0 t0 t1</code>	IF	ID	EX	M	WB					
<code>sub t2 s0 t0</code>						IF	ID	EX	M	WB

Sequence of instructions

Example: In cycle 3, the add instruction is in the execute stage of the pipeline

Sequential Instructions

	1	2	3	4	5	6	7	8	9	10
<code>add s0 t0 t1</code>	IF	ID	EX	M	WB					
<code>sub t2 s0 t0</code>						IF	ID	EX	M	WB

How long to run 2 instructions? **10 cycles**

How long to run each instruction? **5 cycles**

Pipelined Instructions

	1	2	3	4	5	6
<code>add s0 t0 t1</code>	IF	ID	EX	M	WB	
<code>sub t2 s0 t0</code>		IF	ID	EX	M	WB

How long to run 2 instructions? **6 cycles**

How long to run each instruction? **5 cycles** (same as before)

Pipelined Instructions

	1	2	3	4	5	6	7	8	9
<code>add t0 t1 t2</code>	IF	ID	EX	M	WB				
<code>sub t2 s0 t0</code>		IF	ID	EX	M	WB			
<code>or t6 s0 t3</code>			IF	ID	EX	M	WB		
<code>or t3 t4 t5</code>				IF	ID	EX	M	WB	
<code>slt t6 t0 t3</code>					IF	ID	EX	M	WB

Each instruction passes through all 5 stages (1 stage per cycle).

At a particular time, multiple instructions are using different resources.

Performance Analysis: Latency

	1	2	3	4	5	6	7	8	9	10
add	IF	ID	EX	M	WB					
sub						IF	ID	EX	M	WB

	1	2	3	4	5	6
add	IF	ID	EX	M	WB	
sub		IF	ID	EX	M	WB

	Single-Cycle	Pipelined
Time of each stage (clock period)	= 200 + 100 + 200 + 200 + 100 ps (Same as latency)	200 ps All stages same length
Instruction time (latency)	= 800 ps	= 1000 ps

What's the **clock period** and **latency** given the following delays per stage?
IF: 200ps; ID: 100ps; EX: 200ps; M: 200ps; WB: 100ps

Performance Analysis: Throughput

“Iron Law” of processor performance

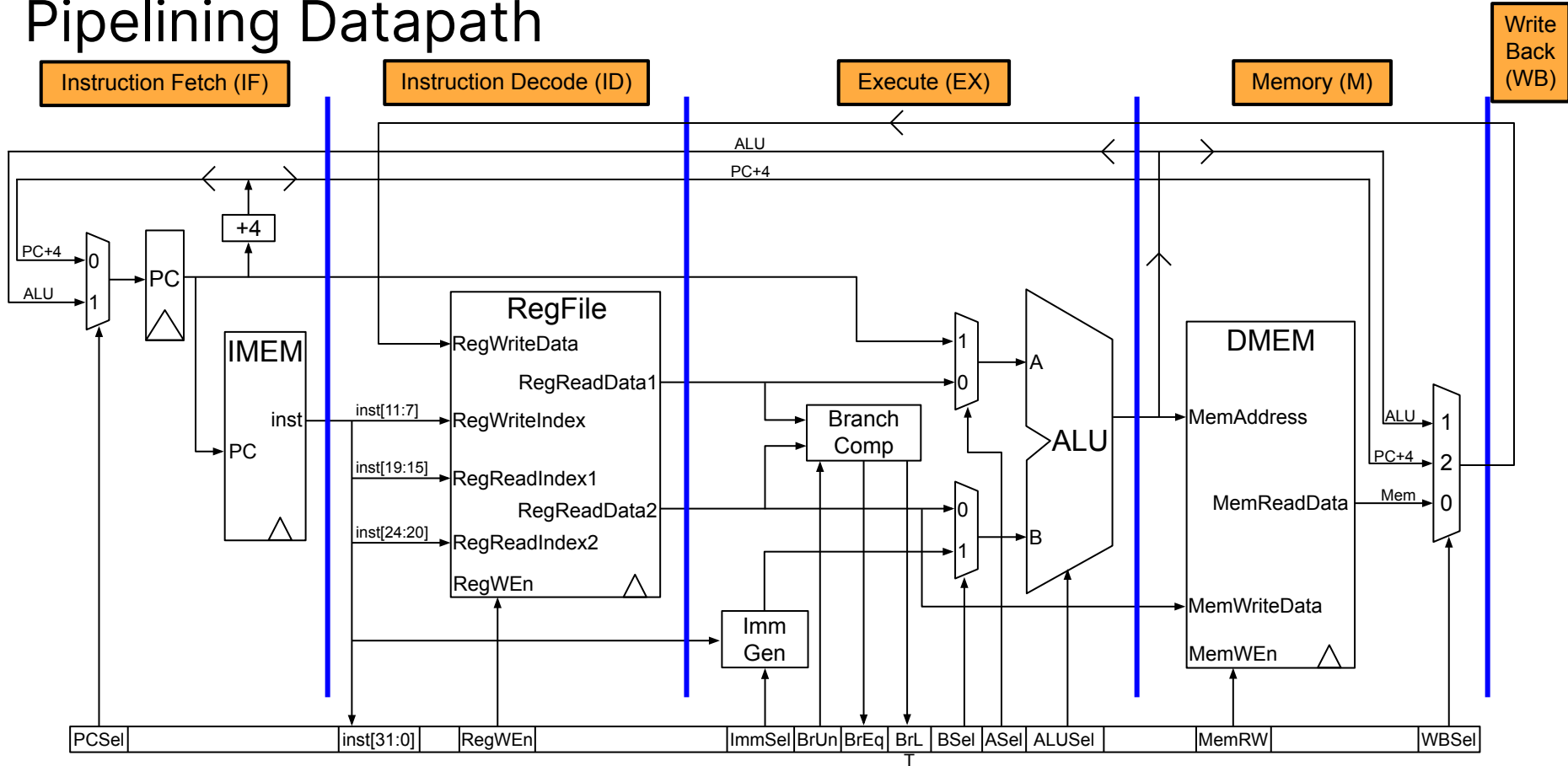
$$\left[\frac{\text{Time}}{\text{Program}} = \frac{\text{Instructions}}{\text{Program}} \times \frac{\text{Cycles}}{\text{Instruction}} \times \frac{\text{Time}}{\text{Cycle}} \right] = 1/\text{throughput}$$

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

	Single-Cycle	Pipelined
Time of each stage (clock period)	= 200 + 100 + 200 + 200 + 100 ps (Same as latency)	200 ps All stages same length
Instruction time (latency)	= 800 ps	= 1000 ps
CPI (cycles per instruction)	~1 (ideal)	~1 (ideal) <1 (actual)
Relative Throughput Gain	1x	4x

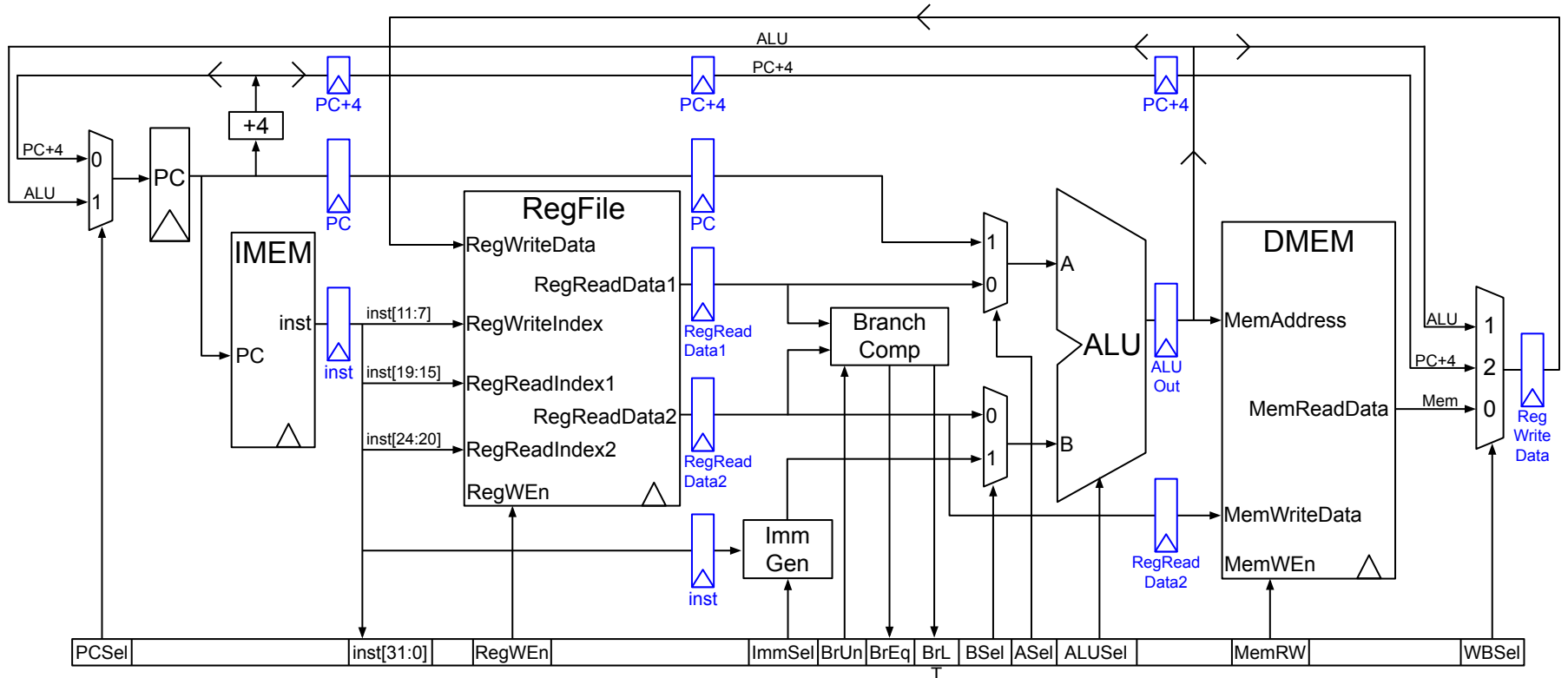
Pipelining Datapath

Pipelining Datapath



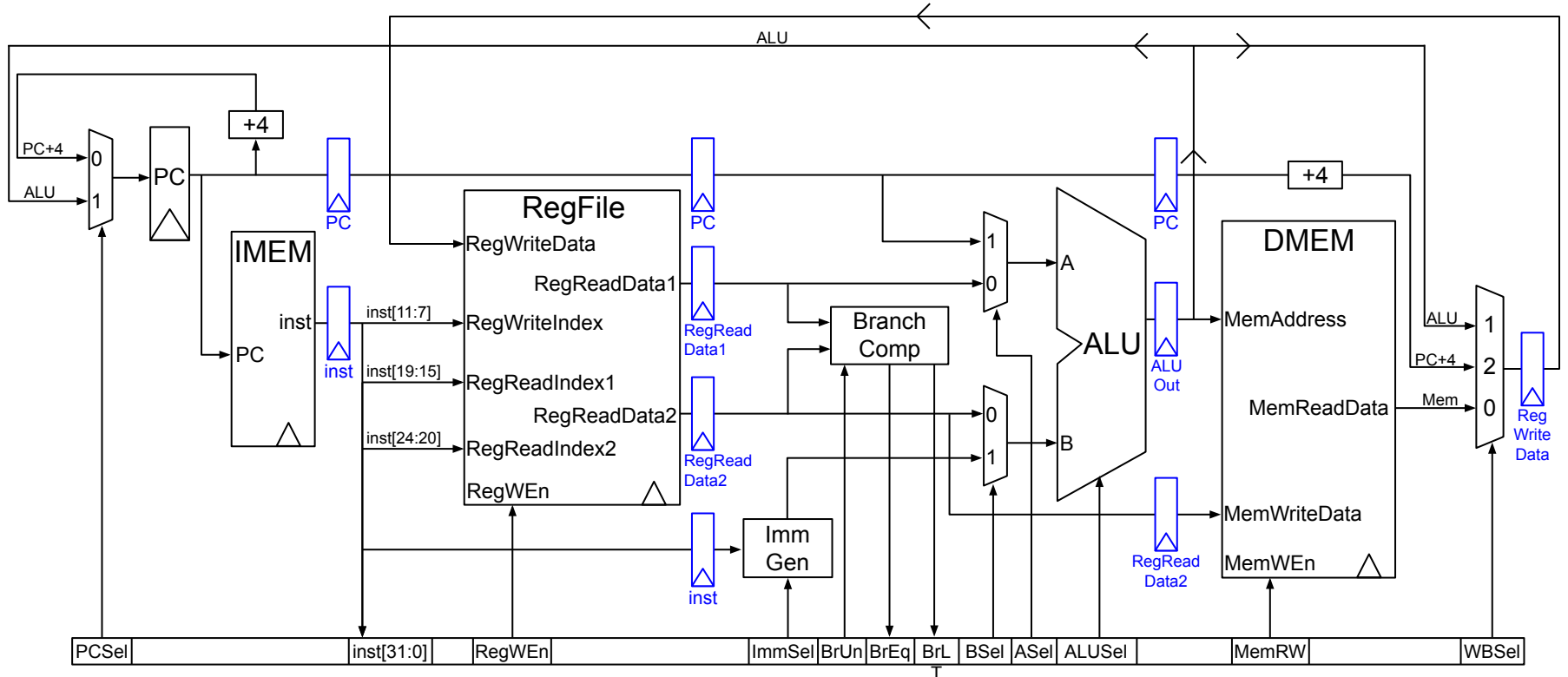
Pipelining Datapath

Add registers between each stage to "hold" a signal until the next rising edge.



Pipelining Datapath

Optimization: Recalculate PC+4 from PC in Memory stage to avoid storing/sending both PC and PC+4 down the pipeline. Is this pipeline functional?

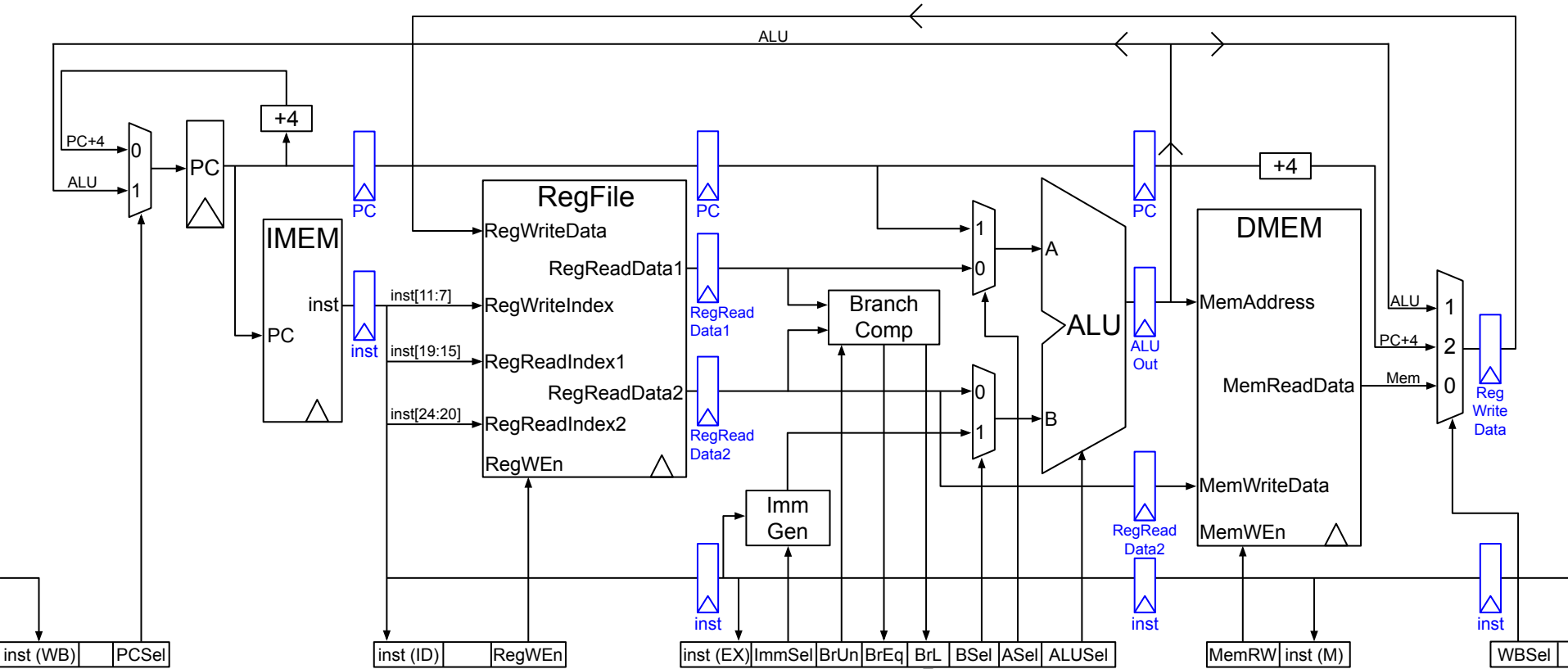


Pipelining Control Logic

- Approach #1: One control logic subcircuit per stage
 - Store and send the instruction through the pipeline stages
 - Calculate relevant signals in each stage
 - Seen on the previous slides
- Approach #2: One control logic subcircuit
 - Calculate all the control logic signals in the decode (ID) stage
 - Store and send the control logic signals through the pipeline stages

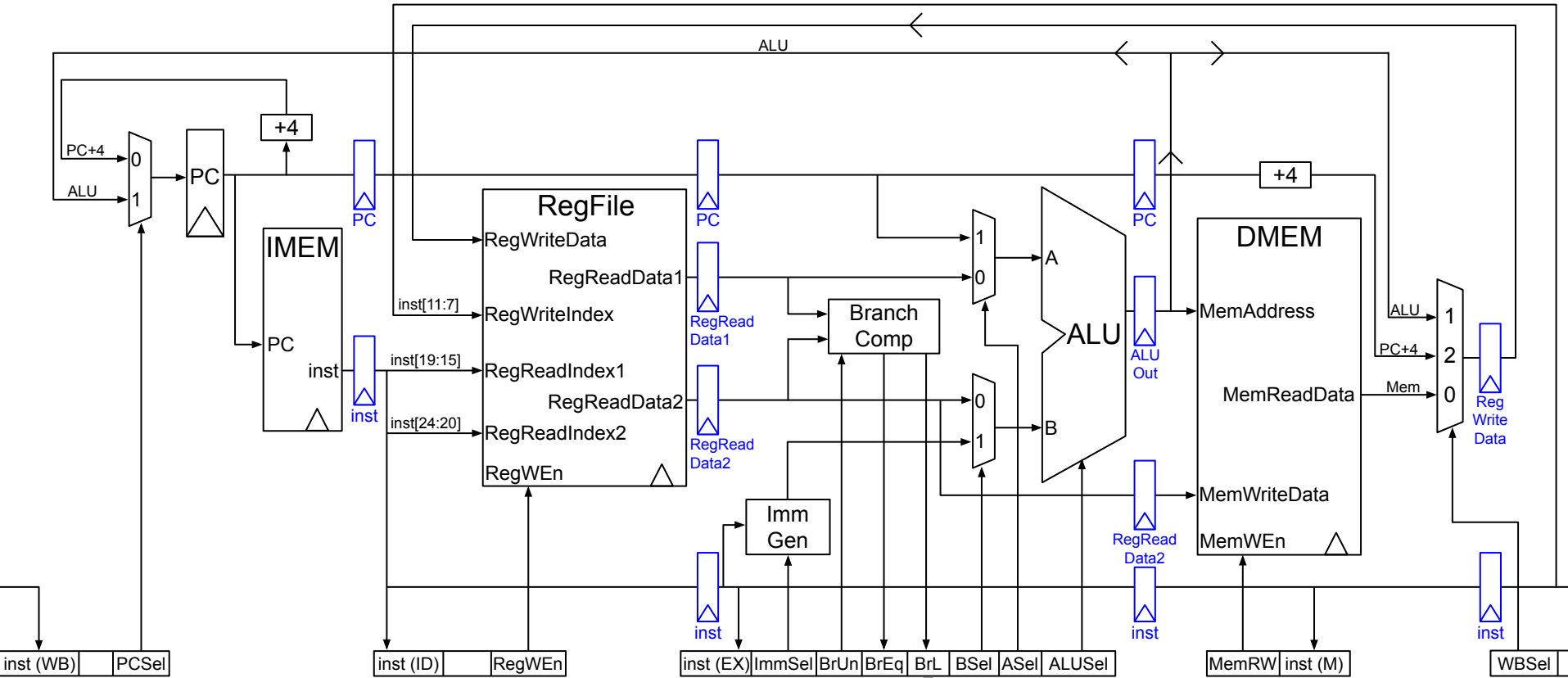
Pipelining Datapath

Store/send instructions down the pipeline so control logic operates correctly in each stage.
Is this pipeline functional?



Pipelining Datapath

Change **RegWriteIndex** to use the **rd** field from the WB stage instruction, not the ID stage instruction.
Is this pipeline functional?



Structural Hazards

Structural Hazards

- Problem: Two or more instructions in the pipeline both need to access a single physical resource
 - In other words, a physical resource is needed in multiple stages
- RegFile
 - What are the physical limits?
 - Can read two registers per cycle
 - Can write to one register per cycle
 - Avoid structural hazard by having separate “ports”
 - Two independent read ports and one independent write port
 - Three accesses per cycle can happen simultaneously
- Instruction memory (IMEM) and data memory (DMEM) used at the same time

Structural Hazard Solutions

- Solution 1: Instructions take turns to use the resources
 - Takes additional cycles per instruction
- Solution 2: Add more hardware to machine
 - Structural hazards can always be solved by adding more hardware
 - E.g. MEM
- Solution 3: Design instruction set architecture (ISA) to avoid structural hazards
 - RISC-V is designed like this
 - Example: RegFile hardware can read two values per cycle. No RISC-V instruction needs to read 3 or more registers at once

Data Hazards

Data Hazards

	1	2	3	4	5	6
add s0 t0 t1	IF	ID	EX	M	WB	
sub t2 s0 t0		IF	ID	EX	M	WB

When does **add** **write** to register s0?

When does the **sub** instruction **read** the value in s0?

Data Hazards

- Problem: Instruction depends on result from previous instruction
 - Example:
`add s0, t0, t1`
`sub t2, s0, t3`
 - When the sub instruction reads s0, the add instruction has not updated s0 yet

Data Hazard Example 1

	1	2	3	4	5	6	7
<code>add t0 t1 t2</code>	IF	ID	EX	M	WB		
<code>or t3 t4 t5</code>		IF	ID	EX	M	WB	
<code>slt t6 t0 t3</code>			IF	ID	EX	M	WB

Does this code have a data hazard?

Data Hazard Example 1

	1	2	3	4	5	6	7
add t0 t1 t2	IF	ID	EX	M	WB		
or t3 t4 t5		IF	ID	EX	M	WB	
slt t6 t0 t3			IF	ID	EX	M	WB

Check if a register being **written to** is **read from** later.

Data Hazard Example 1

	1	2	3	4	5	6	7
<code>add t0 t1 t2</code>	IF	ID	EX	M	WB		
<code>or t3 t4 t5</code>		IF	ID	EX	M	WB	
<code>slt t6 t0 t3</code>			IF	ID	EX	M	WB

`add` writes `t0` at time 5.

`slt` reads `t0` at time 4 (too early to get the updated value).

Data Hazard Example 2

	1	2	3	4	5	6	7	8	9
<code>add t0 t1 t2</code>	IF	ID	EX	M	WB				
<code>or t3 t4 t5</code>		IF	ID	EX	M	WB			
<code>slt t6 t2 t3</code>			IF	ID	EX	M	WB		
<code>lw t3 4(t0)</code>				IF	ID	EX	M	WB	
<code>add t0 t1 t2</code>					IF	ID	EX	M	WB

Does this code have a data hazard?

Data Hazard Example 2

	1	2	3	4	5	6	7	8	9
add t0 t1 t2	IF	ID	EX	M	WB				
or t3 t4 t5		IF	ID	EX	M	WB			
slt t6 t2 t3			IF	ID	EX	M	WB		
lw t3 4(t0)				IF	ID	EX	M	WB	
add t0 t1 t2					IF	ID	EX	M	WB

Check if a register being **written to** is **read from** later.

Data Hazard Example 2

	1	2	3	4	5	6	7	8	9
add t0 t1 t2	IF	ID	EX	M	WB				
or t3 t4 t5		IF	ID	EX	M	WB			
slt t6 t2 t3			IF	ID	EX	M	WB		
lw t3 4(t0)				IF	ID	EX	M	WB	
add t0 t1 t2					IF	ID	EX	M	WB

add updates t0 at time 5. lw reads t0 at time 5. Is that too early?

Data Hazard Example 2

	5
add t0 t1 t2	WB
or t3 t4 t5	M
slt t6 t2 t3	EX
lw t3 4(t0)	ID
add t0 t1 t2	IF

Some regfiles support **writing** a new value to a register, then **reading** the new value, in the same cycle.

“Double pumping”

Not all regfiles support this. Check assumptions in any question.

Fixing Data Hazards

- Solution 1: Stalling
 - Wait for the first instruction to write its result before the second instruction reads the value
- Solution 2: Forwarding
 - Add hardware to send the result back to earlier stages before the result is written
- Solution 3: Code Scheduling
 - Rearrange instructions to avoid data hazards

Stalling Example 1

	1	2	3	4	5	6	7
add t0 t1 t2	IF	ID	EX	M	WB		
or t3 t4 t5		IF	ID	EX	M	WB	
slt t6 t0 t3			IF	ID	EX	M	WB

Without stalling, we have a data hazard:

add writes t0 at time 5.

lw reads t0 at time 4 (too early to get the updated value).

We need **slt** to read from the register 2 time steps later.

Stalling

- Idea: Insert **nops** into the code to delay future instructions
 - Make sure that value is written before future instructions read it
 - Recall: **nop** is an instruction that does nothing. Example?
 - Sometimes called “bubbles”
- Stalls reduce performance
 - But stalls are required to get correct results
- Compiler can insert **nops** to avoid data hazards
 - Requires knowledge of the pipeline structure

Stalling Example 1

	1	2	3	4	5	6	7	8	9
add t0 t1 t2	IF	ID	EX	M	WB				
nop		IF	ID	EX	M	WB			
nop			IF	ID	EX	M	WB		
or t3 t4 t5				IF	ID	EX	M	WB	
slt t6 t0 t3					IF	ID	EX	M	WB

Now the read happens after the write. No more hazard!

Forwarding

Problem: **nops** are inefficient!

- Idea: Use the result as soon as it's computed
 - Read the value directly from a wire in the datapath
 - Don't wait for the result to be stored in a register
- Requires extra connections in the datapath
 - Need to compare destination registers (writes) of current instruction to source registers (reads) of future instructions
 - Need to ignore writes to x0 (you can't write to x0, so writes to x0 won't cause a data hazard)

Forwarding Example 1

	1	2	3	4	5	6
<code>add t0 t1 t2</code>	IF	ID	EX	M	WB	
<code>slt t6 t0 t3</code>		IF	ID	EX	M	WB

Problem: `add` writes `t0` at time 5, which is too late.

When do we first learn the result of the `add` computation?

Forwarding Example 1

	1	2	3	4	5	6
add t0 t1 t2	IF	ID	EX	M	WB	
slt t6 t0 t3		IF	ID	EX	M	WB



Idea: At time 3, we already know the value that will be written to t0.

Add hardware to *forward* this value to the EX stage.

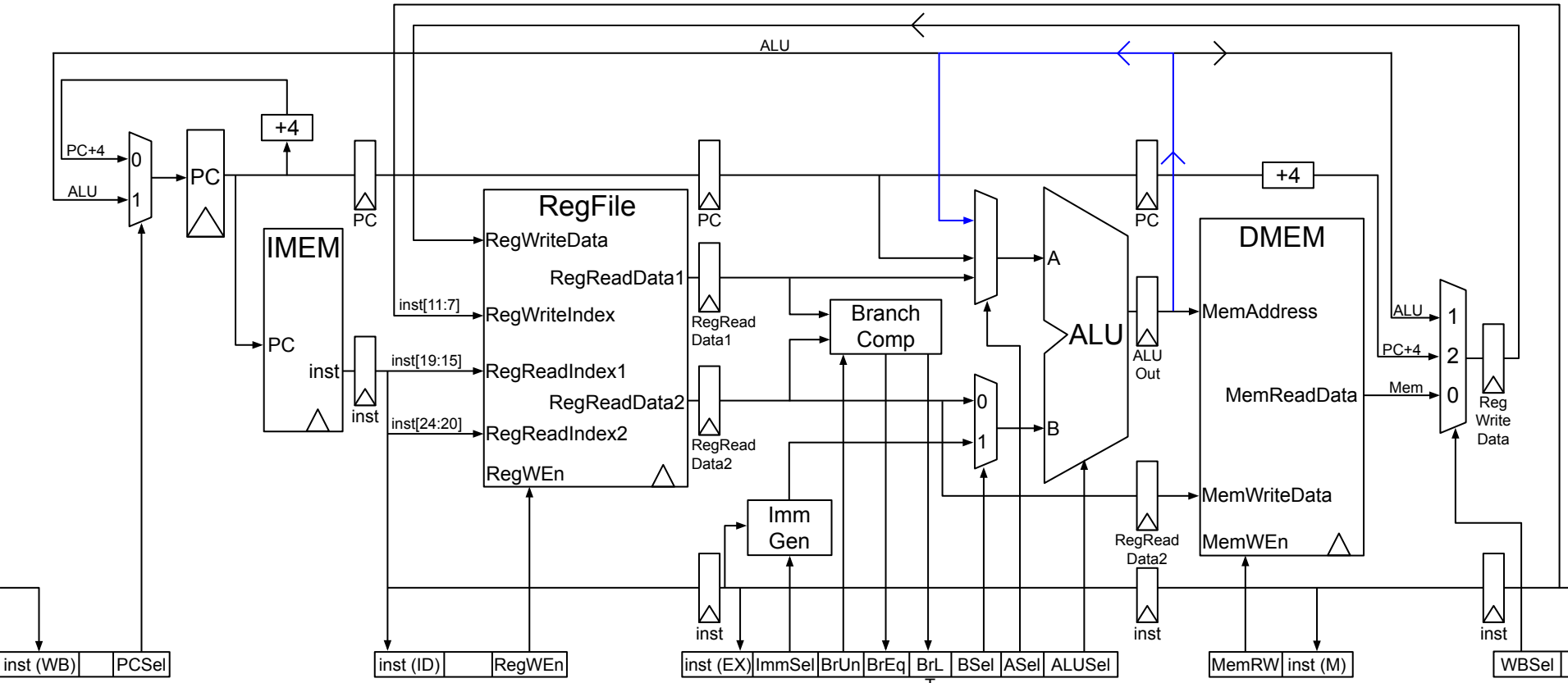
Forwarding Example 1

	1	2	3	4	5	6
add t0 t1 t2	IF	ID	EX	M	WB	
slt t6 t0 t3		IF	ID	EX	M	WB



How do we add a wire to send the output of the EX stage (ALU result) back to the input of the EX stage (ALU input)?

Adding Forwarding to Datapath



Forwarding Example 2

	1	2	3	4	5	6
<code>lw s2 20(s1)</code>	IF	ID	EX	M	WB	
<code>and s4 s2 s5</code>		IF	ID	EX	M	WB

Problem: `lw` writes `s2` at time 5, which is too late.

When do we first learn the result of the `lw` computation?

Forwarding Example 2

	1	2	3	4	5	6
<code>lw s2 20(s1)</code>	IF	ID	EX	M	WB	
<code>and s4 s2 s5</code>		IF	ID	EX	M	WB

The value from memory shows up at time 4.
That's too late to send the value into the ALU.

Load instructions require a one-cycle stall!

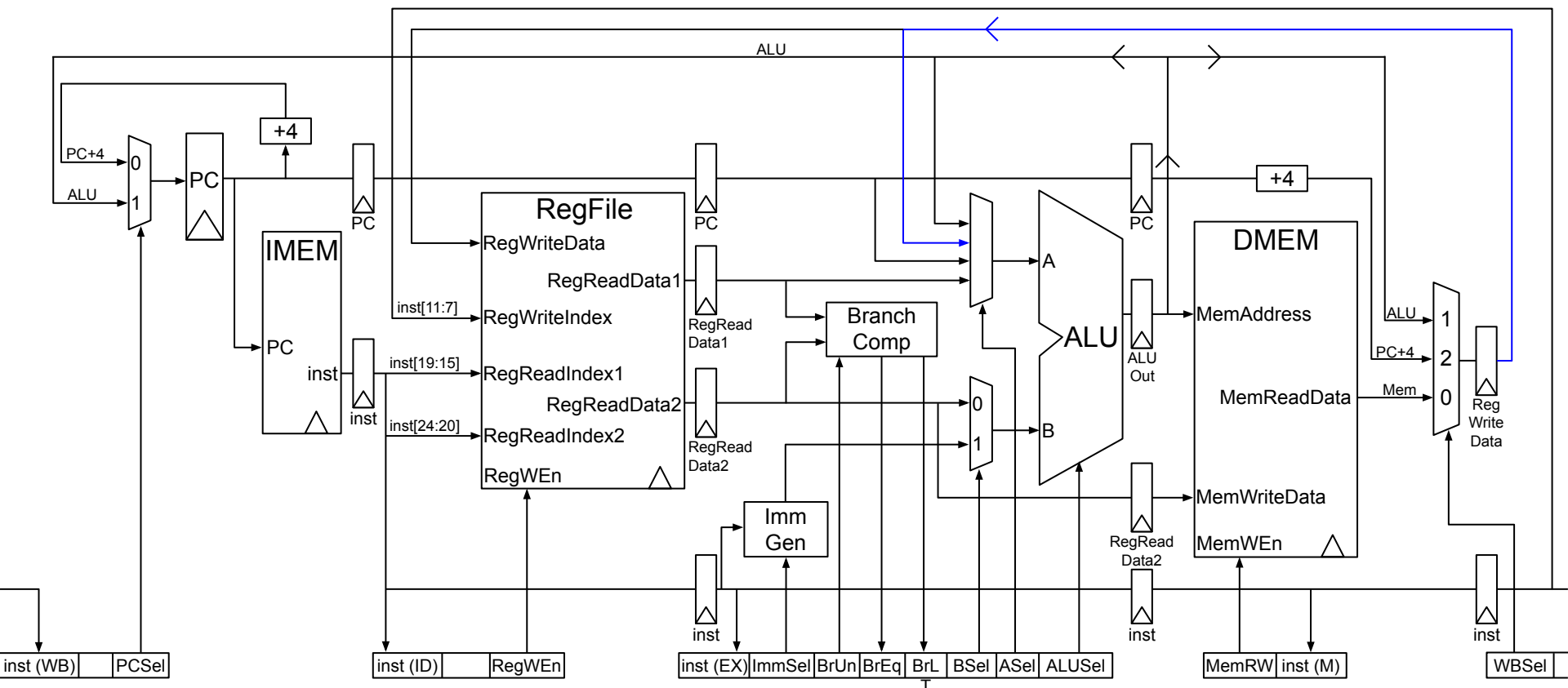
Forwarding Example 1

	1	2	3	4	5	6	7
lw s2 20(s1)	IF	ID	EX	M	WB		
nop		IF	ID	EX	M	WB	
and s4 s2 s5			IF	ID	EX	M	WB



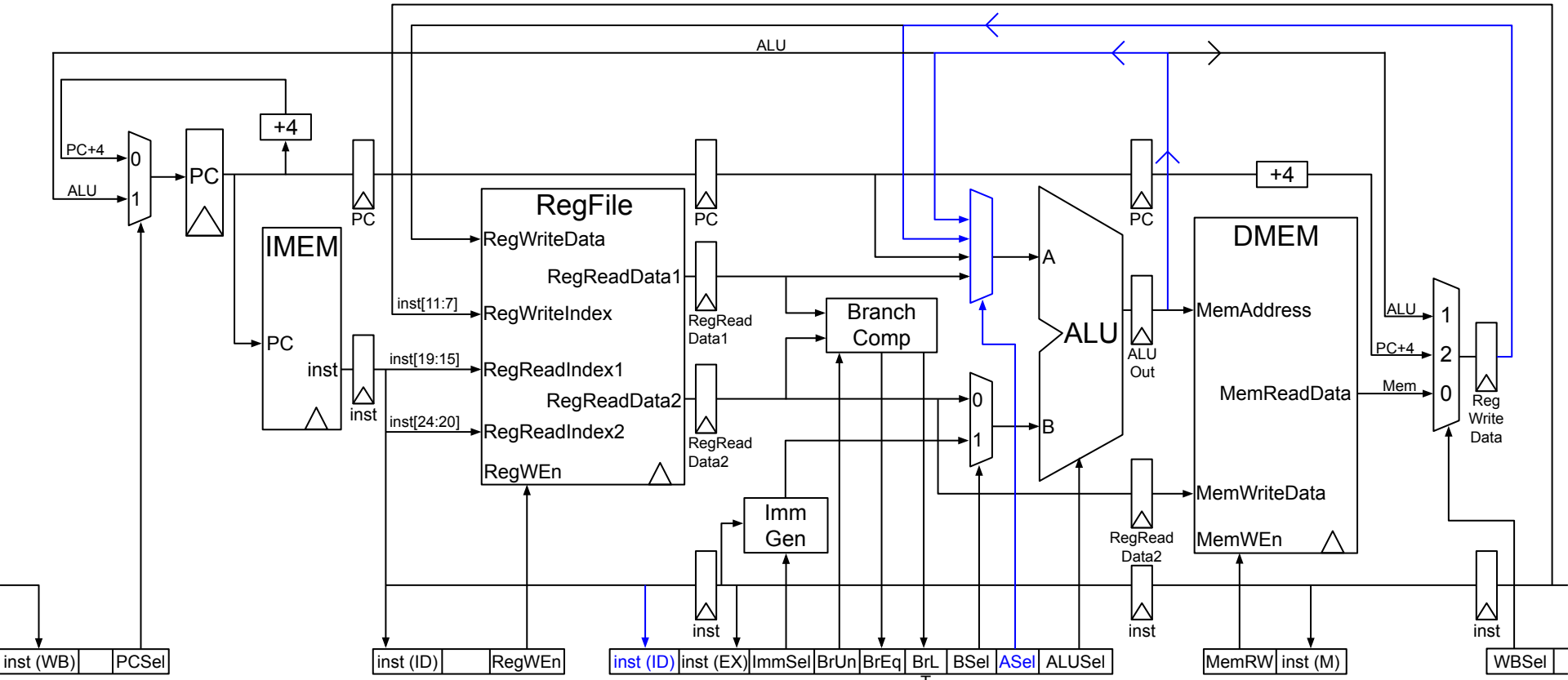
Now we can add hardware to forward the value from memory into the EX stage (ALU) input.

Adding Forwarding to Datapath



Adding Forwarding to Datapath

We need extra control logic to check whether we need to apply forwarding.



Code Scheduling

- Stalling inserts **nops** and causes performance loss
- Idea: Compiler can put an unrelated instruction in the **nop** slot
 - In other words, reorder code to avoid data hazards
 - Requires knowledge of the pipeline structure

Code Scheduling Example

Consider the code snippet

```
A[3] = A[0] + A[1];  
A[4] = A[0] + A[2];
```

Load A[0]	lw t0 0(a0)
Load A[1]	lw t1 4(a0)
A[0]+A[1]	add t3 t0 t1
Store A[3]	sw t3 12(a0)
Load A[2]	lw t2 8(a0)
A[0]+A[2]	add t4 t0 t2
Store A[4]	sw t4 16(a0)

Original order: 2 data hazards

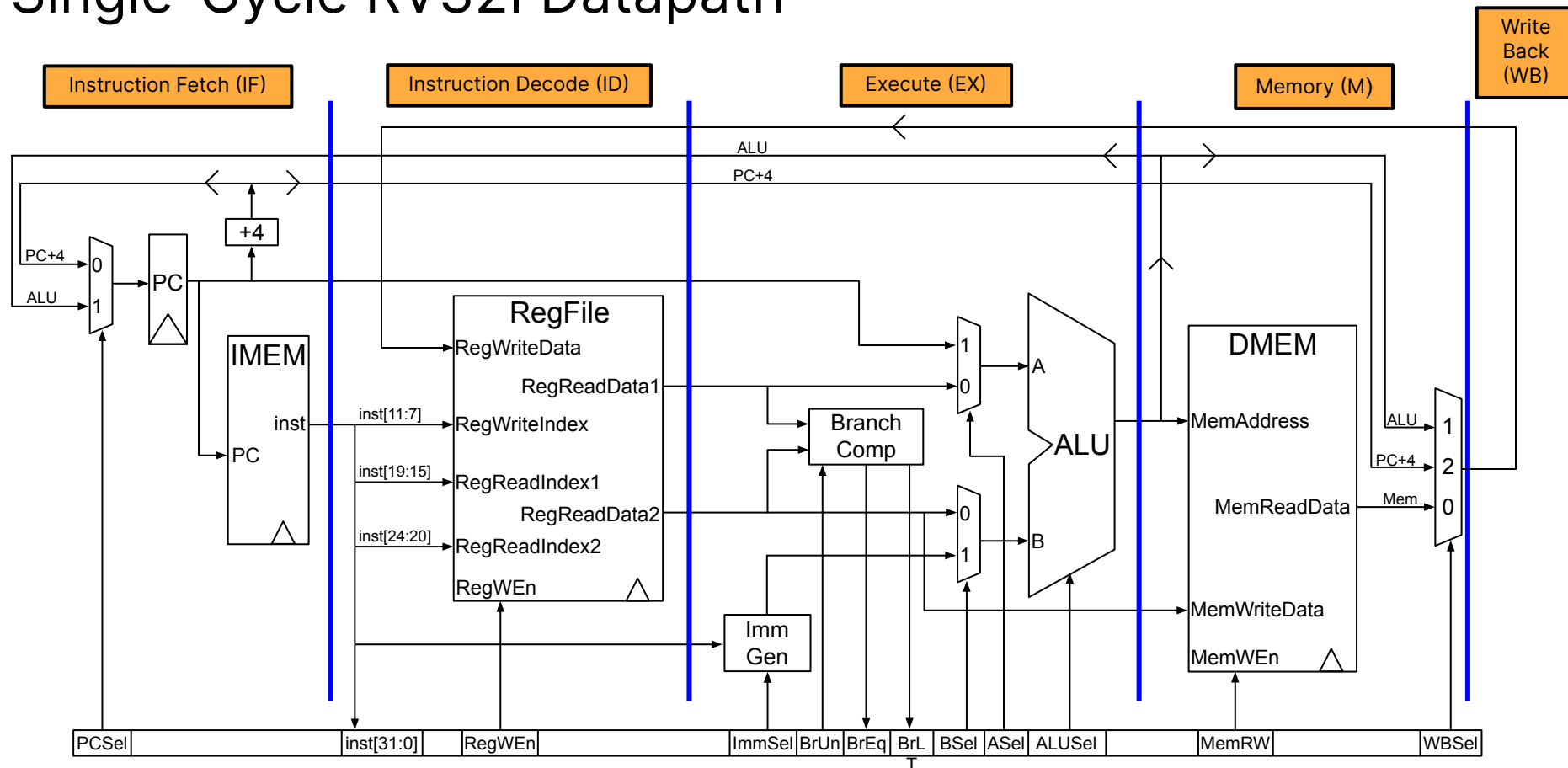
Load A[0]	lw t0 0(a0)
Load A[1]	lw t1 4(a0)
Load A[2]	lw t2 8(a0)
A[0]+A[1]	add t3 t0 t1
Store A[3]	sw t3 12(a0)
A[0]+A[2]	add t4 t0 t2
Store A[4]	sw t4 16(a0)

After reordering the third lw instruction: 0 data hazards

Control Hazards

Single-Cycle RV32I Datapath

Review



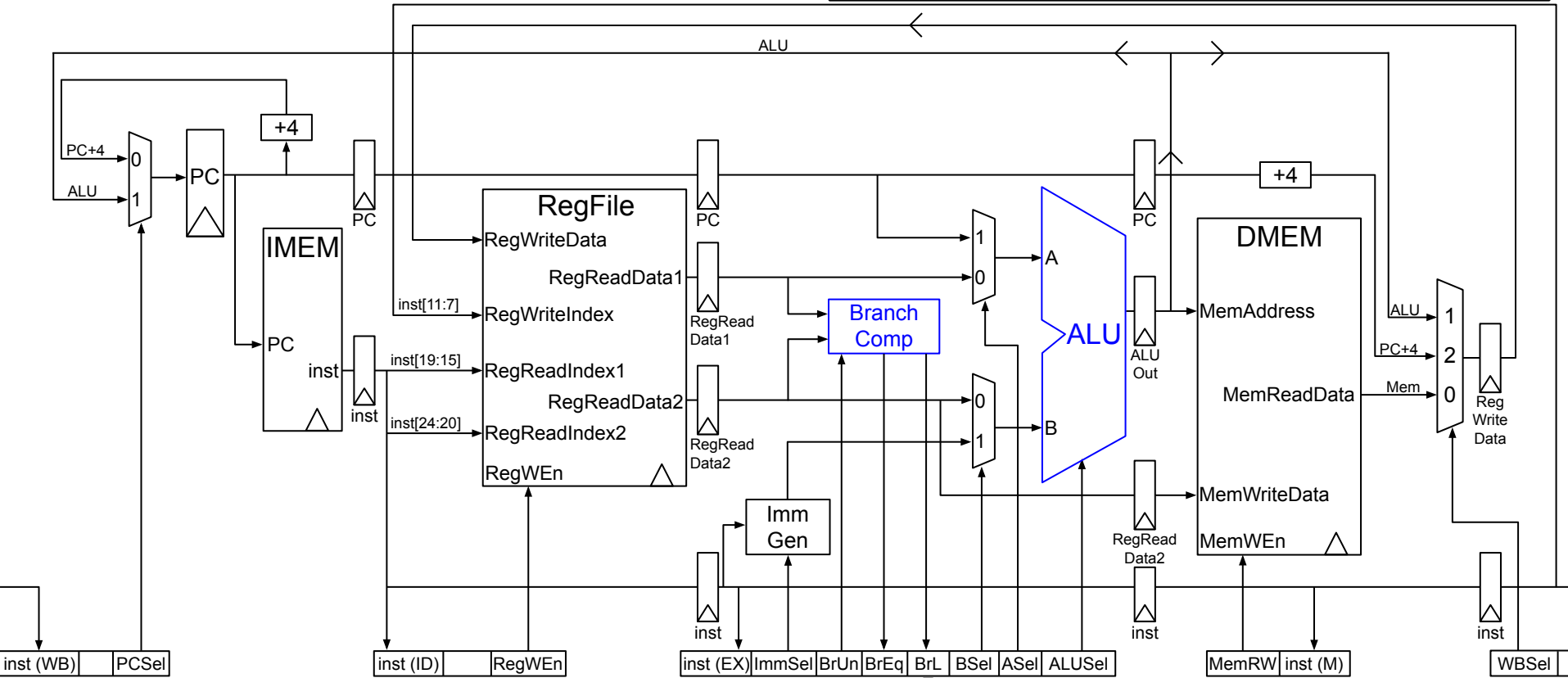
Control Hazard Example 1

	1	2	3	4	5	6	7	8	9
<code>beq t0 t1 Label</code>	IF	ID	EX	M	WB				
<code>sub t2 s0 t0</code>		IF	ID	EX	M	WB			
<code>or t6 s0 t3</code>			IF	ID	EX	M	WB		
<code>xor t5 t1 s0</code>				IF	ID	EX	M	WB	
<code>sw s0 8(t3)</code>					IF	ID	EX	M	WB

If the branch is taken, when do we figure out where to branch to?

Pipelining Datapath

By the time we figure out the next instruction after the branch (EX stage), there are already two more instructions in the pipeline (ID and IF stages).



Control Hazard Example 1

	1	2	3	4	5	6	7	8	9
<code>beq t0 t1 Label</code>	IF	ID	EX	M	WB				
<code>sub t2 s0 t0</code>		IF	ID	EX	M	WB			
<code>or t6 s0 t3</code>			IF	ID	EX	M	WB		
<code>xor t5 t1 s0</code>				IF	ID	EX	M	WB	
<code>sw s0 8(t3)</code>					IF	ID	EX	M	WB

If the branch is taken, convert the next two instructions to **nops**.

Control Hazards

- If branch is not taken:
 - Instructions fetched sequentially after branch are correct
 - No control hazard
- If branch is taken, or there's a jump:
 - The next two instructions still in the pipeline are incorrect
 - Need to convert incorrect instructions in the pipeline to **nops**
 - Called “flushing” the pipeline

Branch Prediction to Reduce Penalties

- Every taken branch in the simple RV32I pipeline costs 2 clock cycles.
 - Note if branch is not taken, then pipeline is not stalled; the correct instructions are correctly fetched sequentially after the branch instruction.
- We can improve the CPU performance on average through branch prediction.
 - Early in the pipeline, guess which way branches will go.
 - Flush pipeline if the guess was incorrect.
- Naive branch prediction: just predict branch “not taken”, which always fetches PC+4

Superscalar Processors

Details of which are not in scope :)
Take 152 if you want to learn more!

Pipelining and ISA Design

- RISC-V ISA (instruction set architecture) is designed for pipelining
- All instructions are 32 bits long
 - Easy to fetch and decode in one stage
 - (Compare to x86: instructions are 1 byte to 15 bytes long)
- Few and regular instruction formats
 - Decode and read registers in one stage
- Load/store addressing
 - Calculate address in stage 3 (EX), access memory in stage 4 (memory)
- Alignment of memory operands
 - Memory access takes only one cycle

Further Increasing Processor Performance?

1. Increase clock rate.

- Limited by technology and power dissipation

2. Increase pipeline depth.

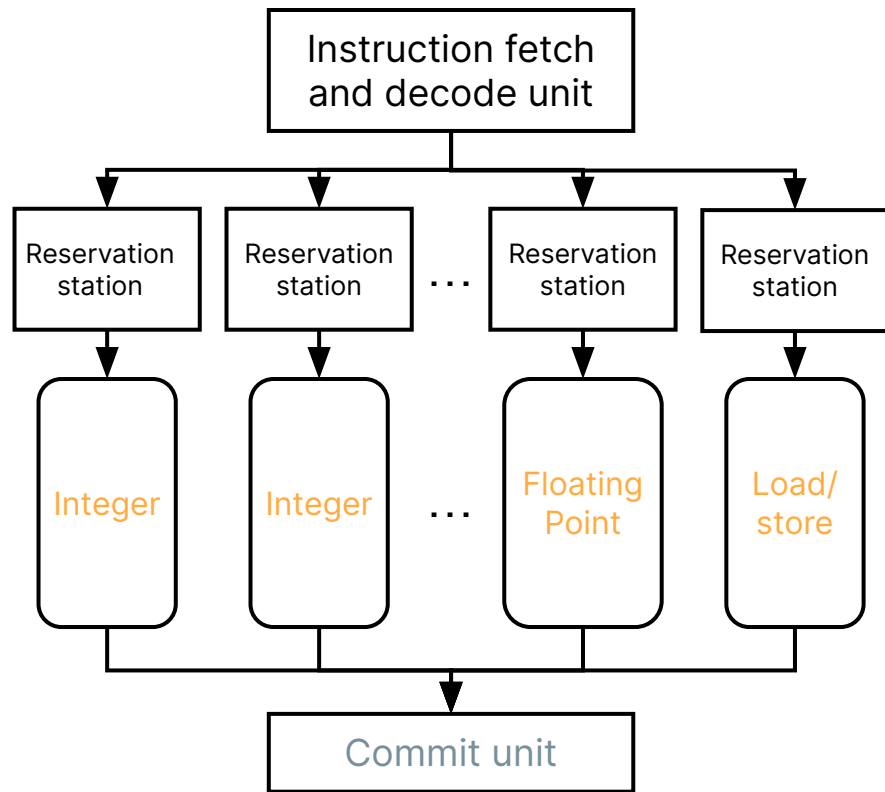
- “Overlap” instruction execution through deeper pipeline, e.g., 10 or 15 stages.
 - Less work per stage □ shorter clock cycle/lower power
 - But more potential for all three types of hazards! (more stalling □ $CPI > 1$)

3. Design a “superscalar” processor.

- Desktops, laptops, cell phones, etc. often have a few of these, combined with simpler 5-stage pipeline processors.

Superscalar Processors

- Multiple-issue: Start multiple instructions per clock cycle.
 - Multiple execution units execute instructions in parallel.
 - Each execution unit has its own pipeline.
 - $CPI < 1$: multiple instructions completed per clock cycle.
- Dynamic “out-of-order” execution:
 - Reorder instructions dynamically in HW to reduce impact of hazards.



Summary, What's Next

- We learned how to build a processor!
 - Datapath
 - Control
- And how to make it perform better!
 - Pipelining decreases time per cycle, increases throughput
 - Be careful about hazards :)
 - Superscalar processing
- Next, we go back to the software level and try to increase performance there!