Course Info

- Lab 6 next week, prepare before lab sessions! Keep an eye on piazza.
- Project 1.2 ddl March 31st. Project 2.1 released next week!
- This week discussion on ALU & FSM. Next week discussion on datapath.
- Mid-term I solution & score released. If you have questions about the solution, feel free to ask on Piazza; If you have questions regarding your marks, email the instructors **BEFORE April 2nd**. We will get back to you ASAP.
- Any regrade request after April 2nd **WOULD NOT** be considered

Course Info

- HW4 released. Submit your paper homework to the box below (at SIST 3-322). DDL April 7th.
- Remember to add your name. You have only one chance to submit and cannot be withdrawn.

信息科学与技术学院
School of Information Science and Technology

CS 110
Computer Architecture
Pipeline

Instructors:

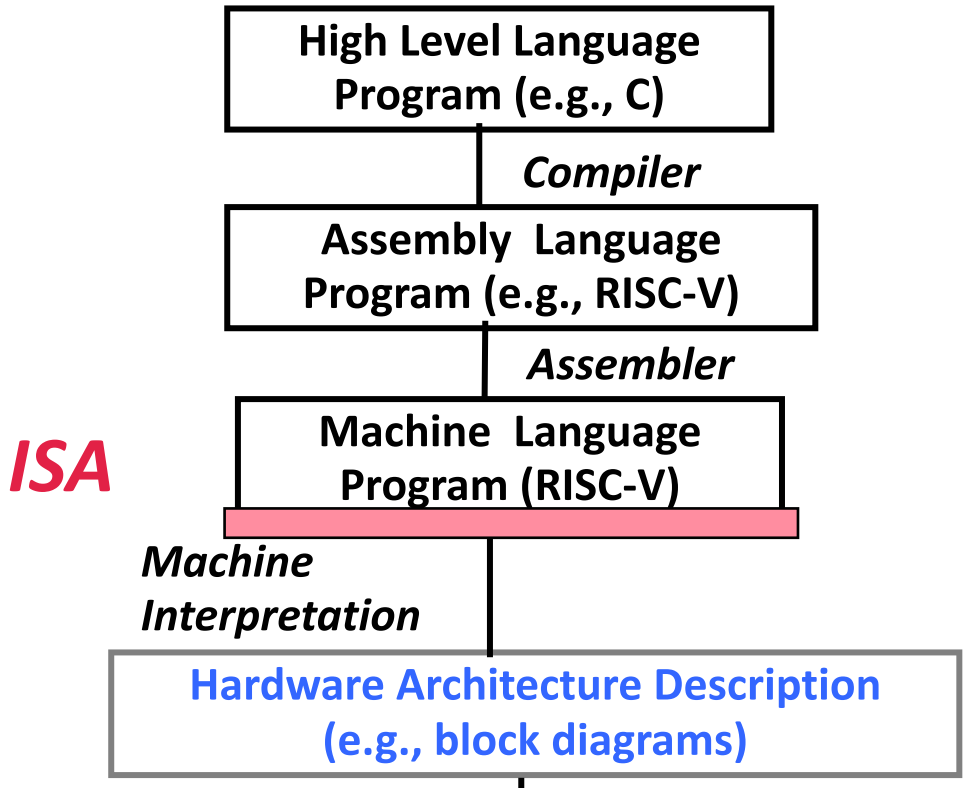
Siting Liu & Chundong Wang

Course website: https://toast-lab.sist.shanghaitech.edu.cn/courses/CS110@ShanghaiTech/Spring-2023/index.html

School of Information Science and Technology (SIST)

ShanghaiTech University

2023/3/23



High Level Language Program (e.g., C)
Compiler
Assembly Language Program (e.g., RISC-V)
Assembler
Machine Language Program (RISC-V)
ISA
Machine Interpretation
Hardware Architecture Description (e.g., block diagrams)

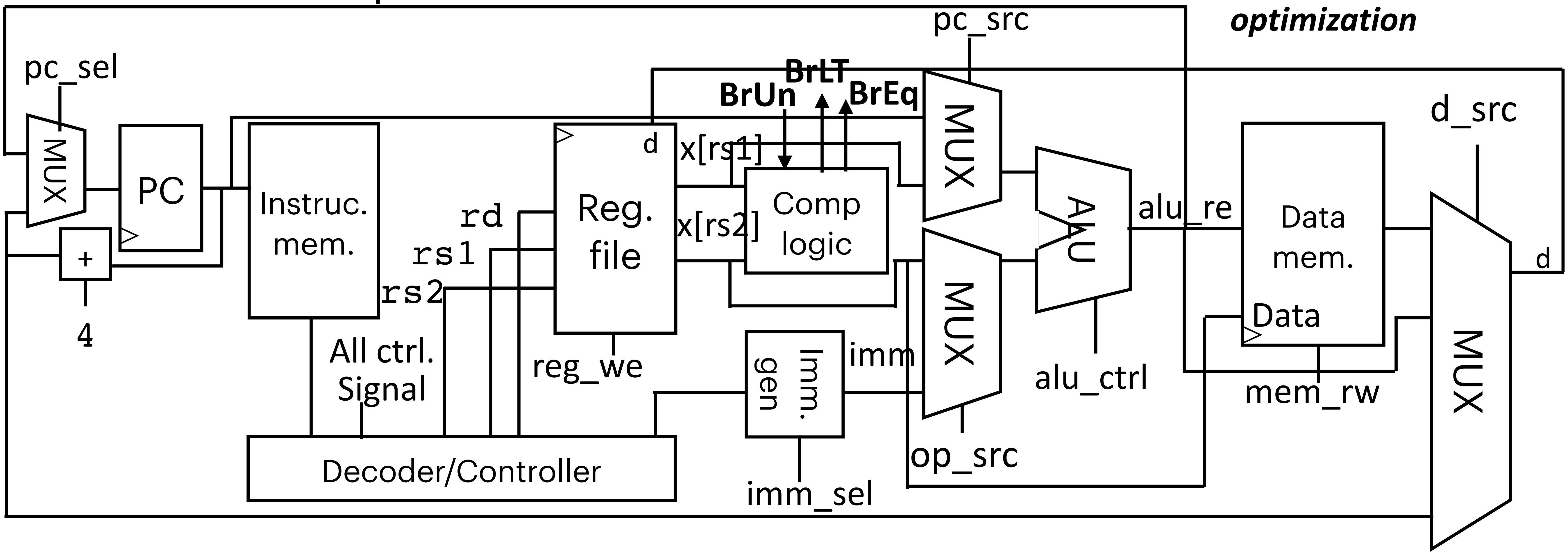
Architecture Implementation

temp = v[k];
v[k] = v[k+1];
v[k+1] = temp;

Anything can be represented
as a *number*,
i.e., data or instructions

```
0000 1001 1100 0110 1010 1111 0101 1000
1010 1111 0101 1000 0000 1001 1100 0110
1100 0110 1010 1111 0101 1000 0000 1001
0101 1000 0000 1001 1100 0110 1010 1111
```

Single-core simple CPU w/o any optimization

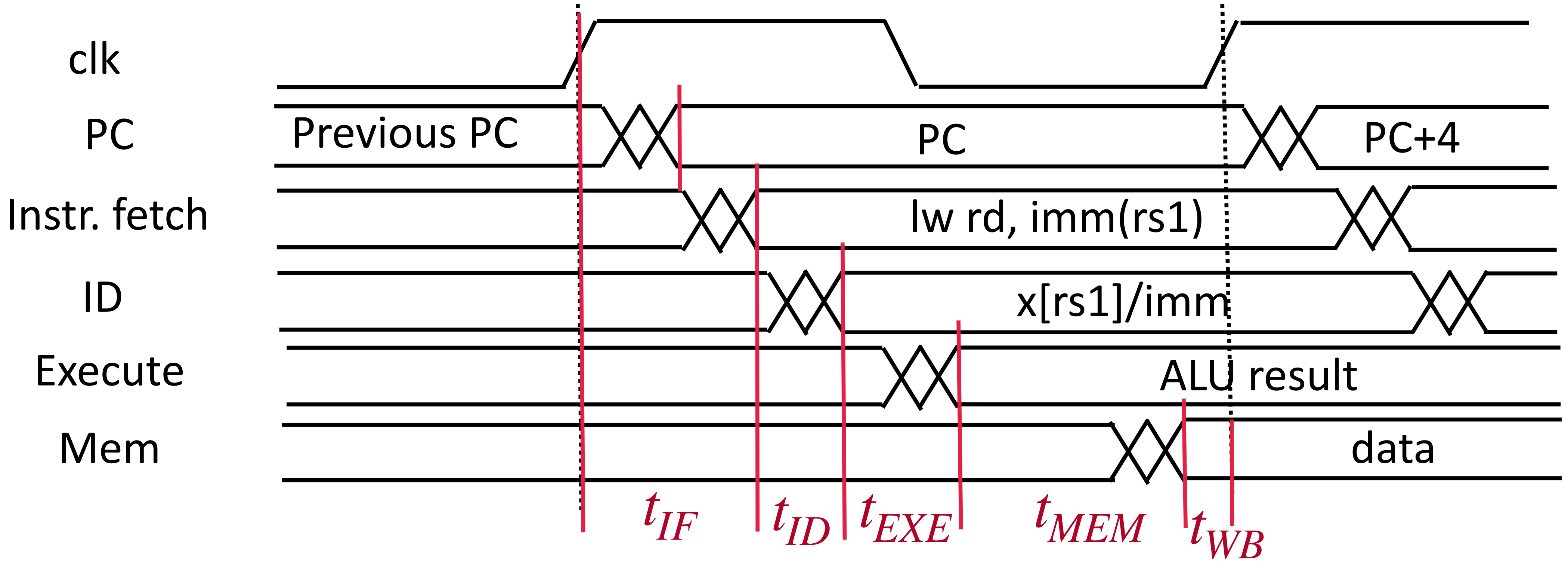


Summary

- We have built a processor!
  - Capable of executing all RISC-V instructions in one cycle each
  - Not all units (hardware) used by all instructions
  - Critical path changes for different instructions
  - $T_{clk} =$ Max delay, $t_{IF} + t_{ID} + t_{EX} + t_{MEM} + t_{WB}$
  - Max Frequency = 1/Max delay
- 5 Phases of execution
  - IF, ID, EX, MEM, WB
  - Not all instructions are active in all phases
- Controller specifies how to execute instructions
  - Implemented as ROM or logic

Bottleneck

Instru.	IF = 300 ps	ID = 100 ps	EX = 200 ps	MEM = 300 ps	WB = 100 ps	Total
`add`	X	X	X		X	700 ps
`beq`	X	X	X			600 ps
`jal`	X	X	X		X	700 ps
`lw`	X	X	X	X	X	1000 ps
`sw`	X	X	X	X		900 ps



Great Ideas in Computer Architecture

- Abstraction (Layers of Representation / Interpretation)
- Moore's Law
- Principle of Locality/Memory Hierarchy
- Parallelism
- Performance Measurement and Improvement
- Dependability via Redundancy

“Iron Law” of Processor Performance

$$\frac{\text{Time}}{\text{Program}} = \frac{\text{Instructions}}{\text{Program}} \cdot \frac{\text{Cycles}}{\text{Instruction}} \cdot \frac{\text{Time}}{\text{Cycle}}$$

$$\frac{\text{Instructions}}{\text{Program}}$$

RISC vs. CISC

- ISA-relevant
- Compiler
- What task, complexity
- Etc.

"Iron Law" of Processor Performance

$$\frac{\text{Time}}{\text{Program}} = \frac{\text{Instructions}}{\text{Program}} \cdot \frac{\text{Cycles}}{\text{Instruction}} \cdot \frac{\text{Time}}{\text{Cycle}}$$

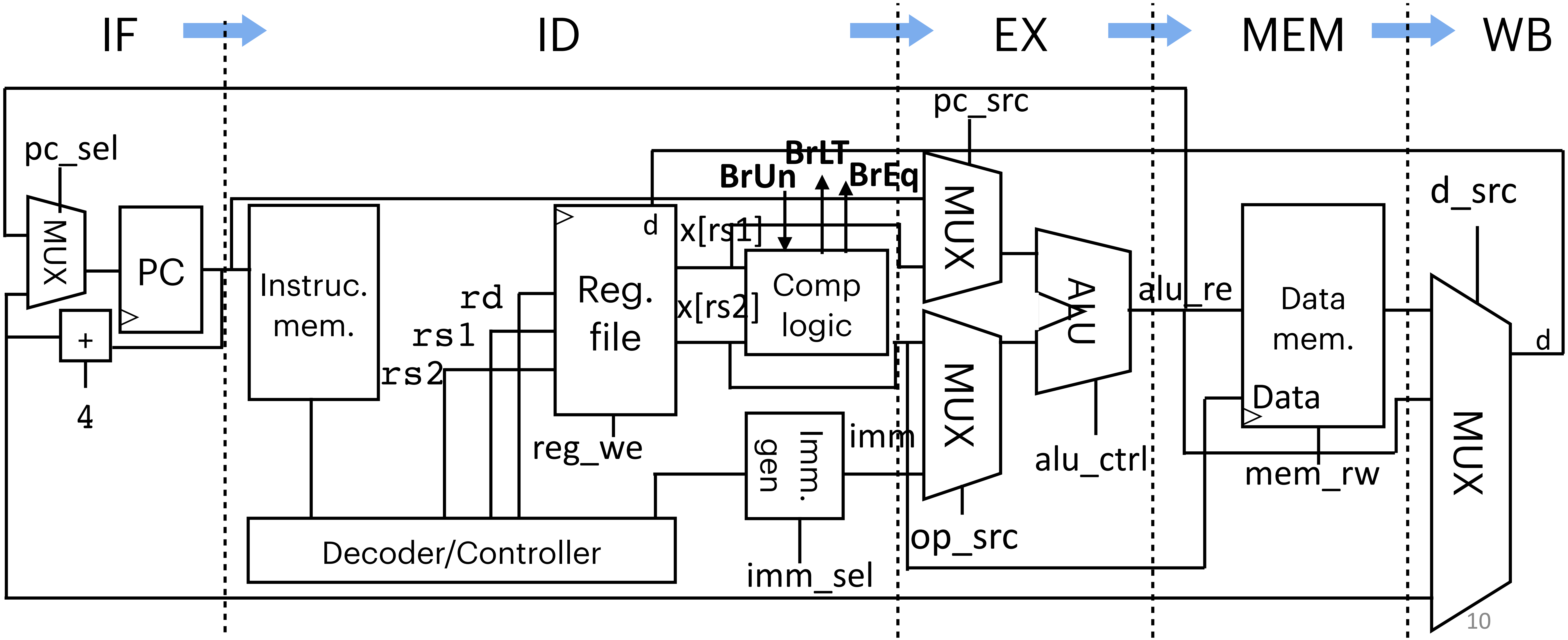
$$\frac{\text{Cycles}}{\text{Instruction}}$$

- Microarchitecture implementation or circuit design
- ISA

Timing Diagram (Consider delays)

Instru.	IF = 300 ps	ID = 100 ps	EX = 200 ps	MEM = 300 ps	WB = 100 ps	Total
`lw`	X	X	X	X	X	1000 ps

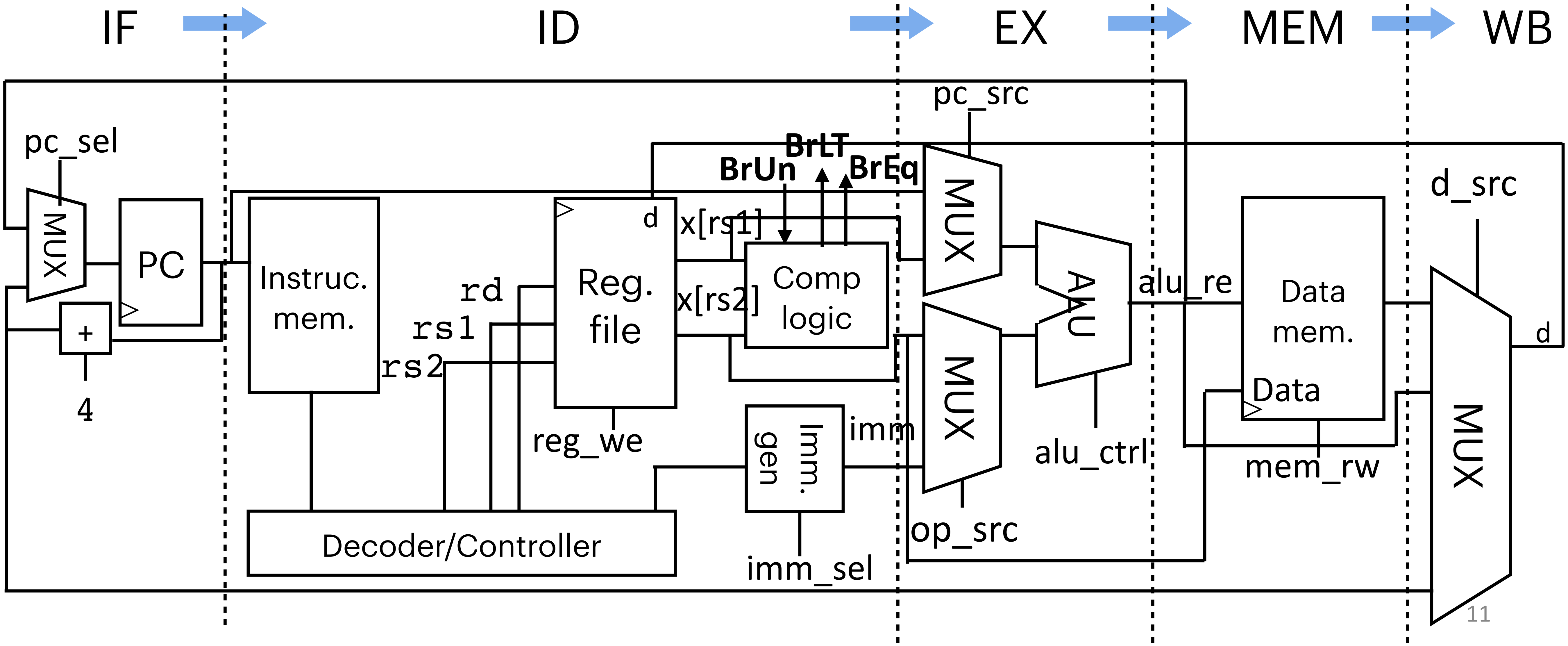
$\frac{\text{Cycles}}{\text{Instruction}}$ (CPI) $\frac{\text{Time}}{\text{Cycle}}$ (Critical path; technology; TPD etc.)



Timing Diagram (Consider delays)

Instru.	IF = 300 ps	ID = 100 ps	EX = 200 ps	MEM = 300 ps	WB = 100 ps	Total
`lw`	X	X	X	X	X	1000 ps

$$\frac{\text{Time}}{\text{Program}} = \frac{\text{Instructions}}{\text{Program}} \cdot \frac{\text{Cycles}}{\text{Instruction}} \cdot \frac{\text{Time}}{\text{Cycle}}$$



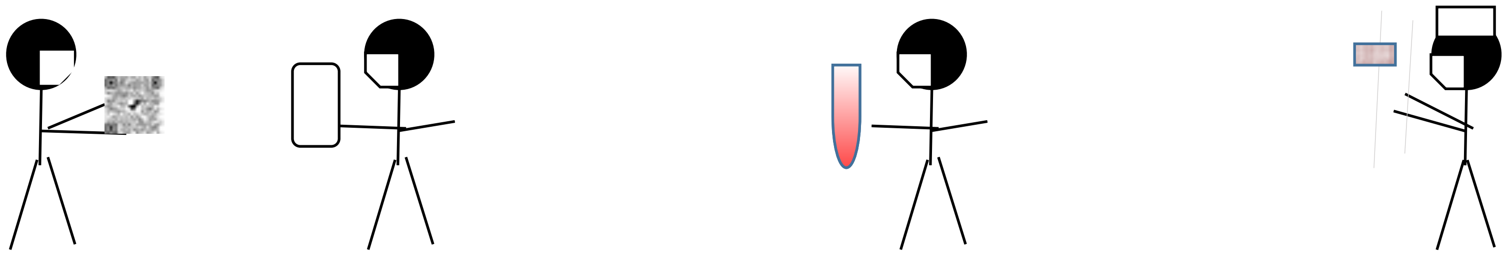
Pipeline: Improve Time/Program

- Instruction: PCR test: 1. scan QR code; 2. get the tube; 3. sample
- 5 seconds for each step:
- Single-cycle (once for all)

$$\frac{Time}{Program} = \frac{Instructions}{Program} \times \frac{Clock\ cycles}{Instruction} \times \frac{Time}{Clock\ cycle}.$$

Multi-cycle

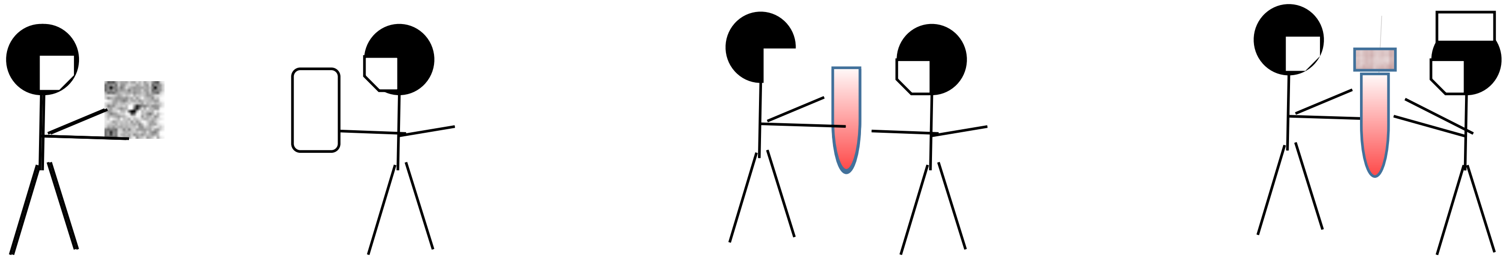
- Instruction: PCR test: 1. scan QR code; 2. get the tube; 3. sample
- 5 seconds for each step
- Multi-cycle or multi-step



$$\frac{Time}{Program} = \frac{Instructions}{Program} \times \frac{Clock\ cycles}{Instruction} \times \frac{Time}{Clock\ cycle}.$$

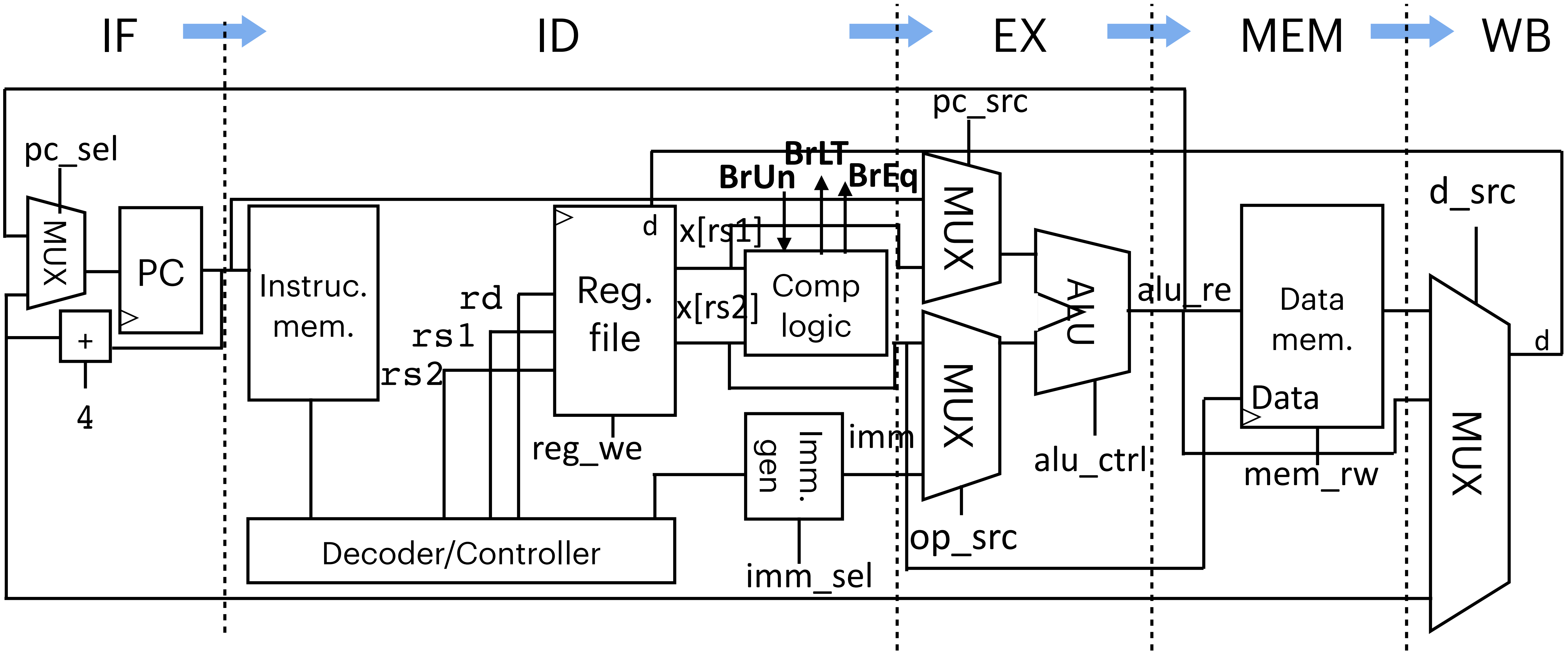
Pipeline

- PCR test in pipeline



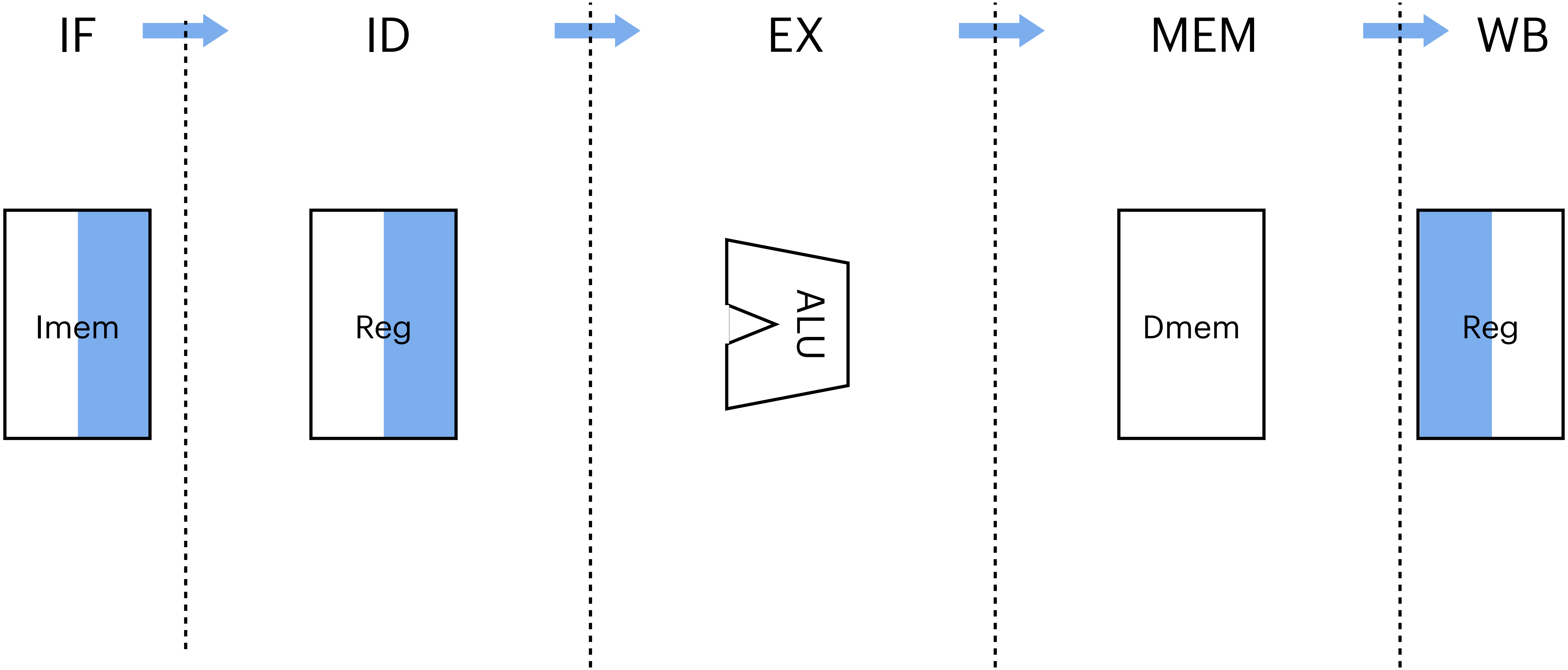
$$\frac{Time}{Program} = \frac{Instructions}{Program} \times \frac{Clock\ cycles}{Instruction} \times \frac{Time}{Clock\ cycle}.$$

Analogy in our CPU Design

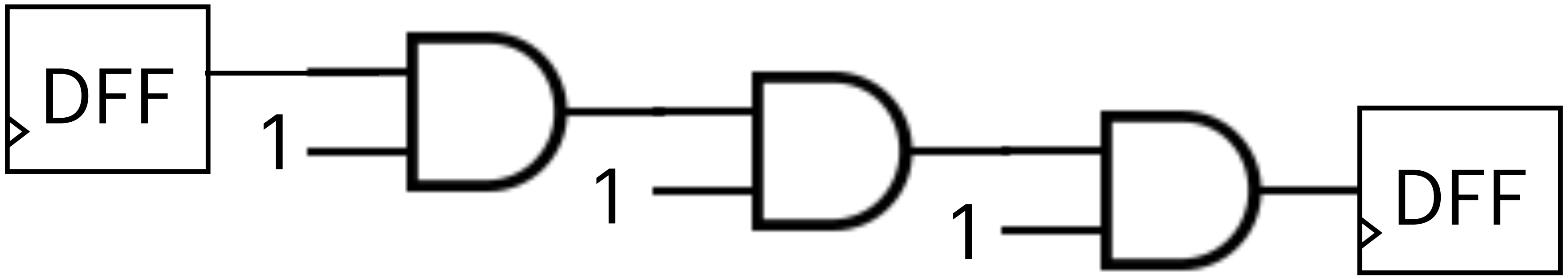


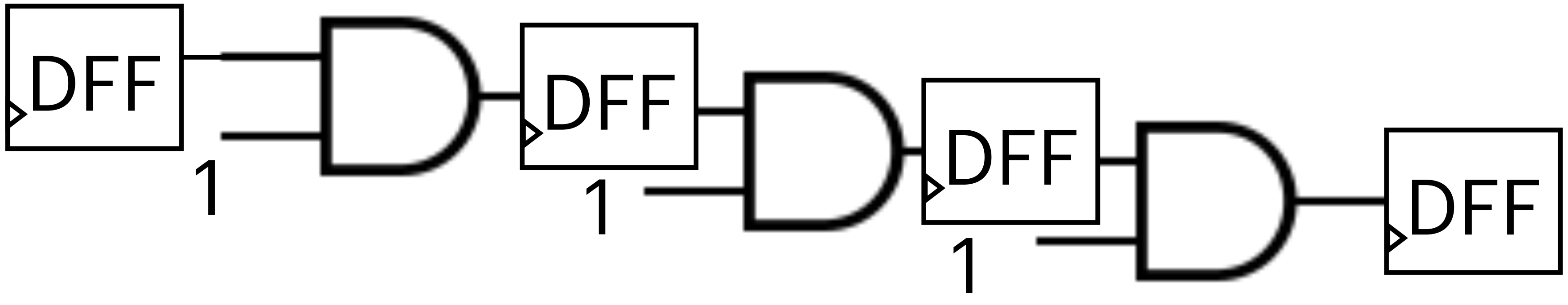
$$\frac{Time}{Program} = \frac{Instructions}{Program} \times \frac{Clock\ cycles}{Instruction} \times \frac{Time}{Clock\ cycle}.$$

Simplify the Model using Symbols



Recall DFFs

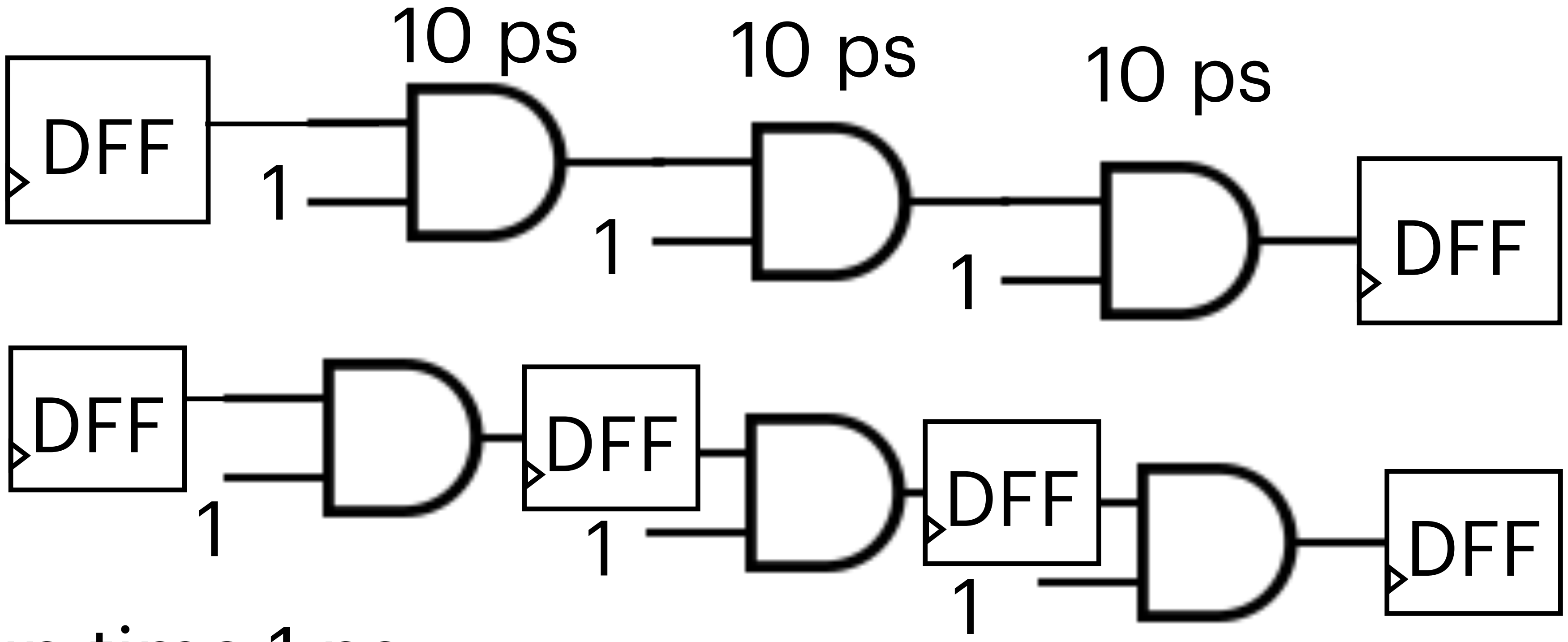




Recall DFFs

Single-cycle

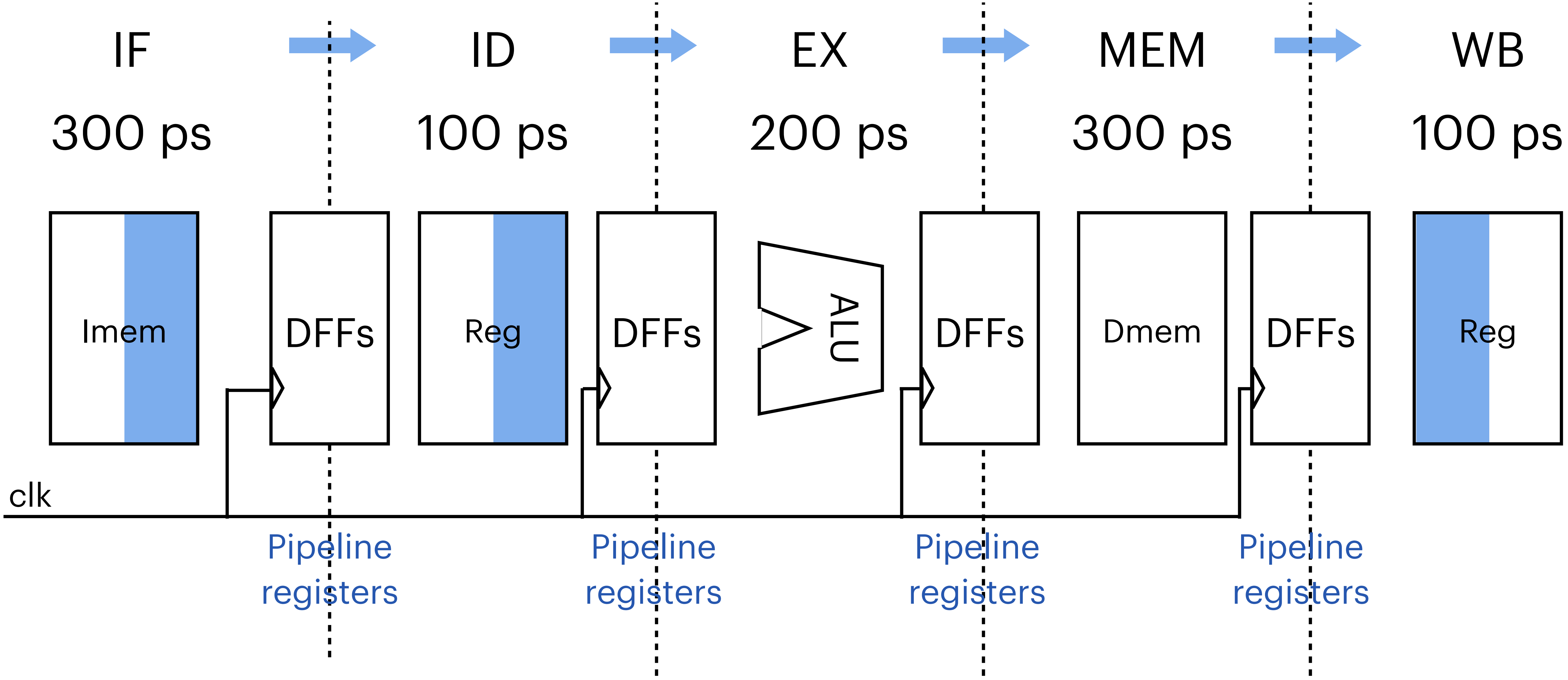
Pipelining



Clk-to-q & setup time 1 ps

	Single-cycle	Pipelining
T_{clk}		
Clock rate/frequency		
Computation time		
Clock cycle per output		
Relative speed		
Hardware		

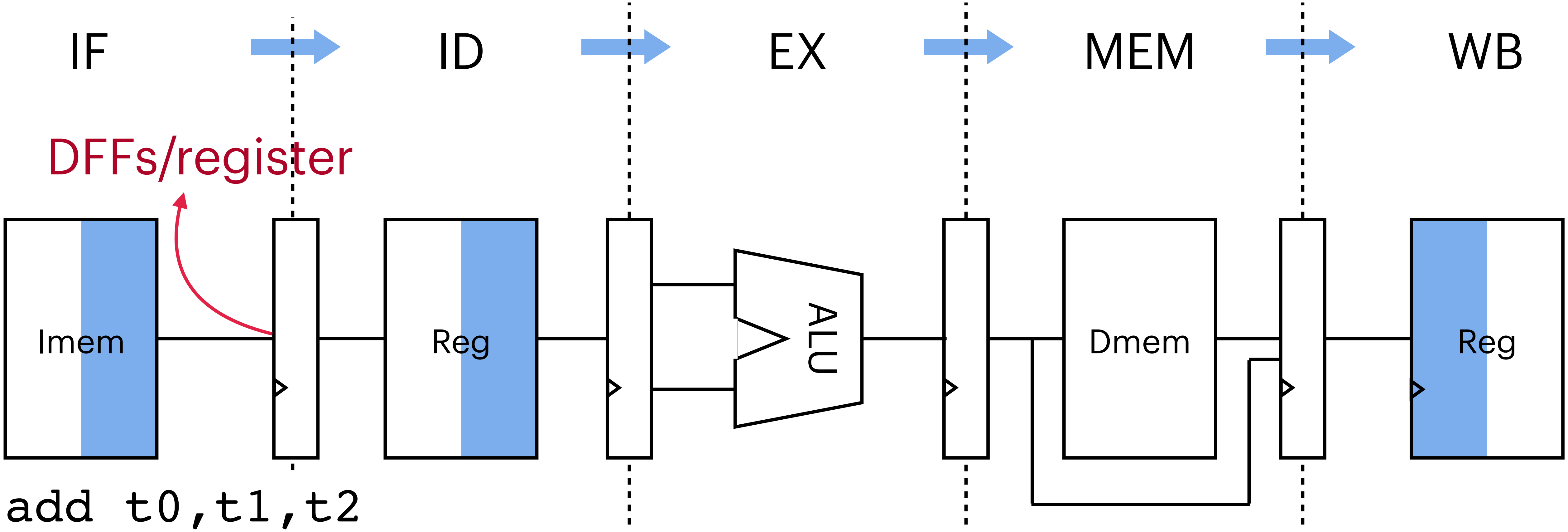
Insert Pipeline Registers



$$\frac{\text{Time}}{\text{Program}} = \frac{\text{Instructions}}{\text{Program}} \cdot \frac{\text{Cycles}}{\text{Instruction}} \cdot \frac{\text{Time}}{\text{Cycle}}$$

$$T_{clk} = \text{Max Delay} = \max(t_{IF}, t_{ID}, t_{EX}, t_{MEM}, t_{WB})$$

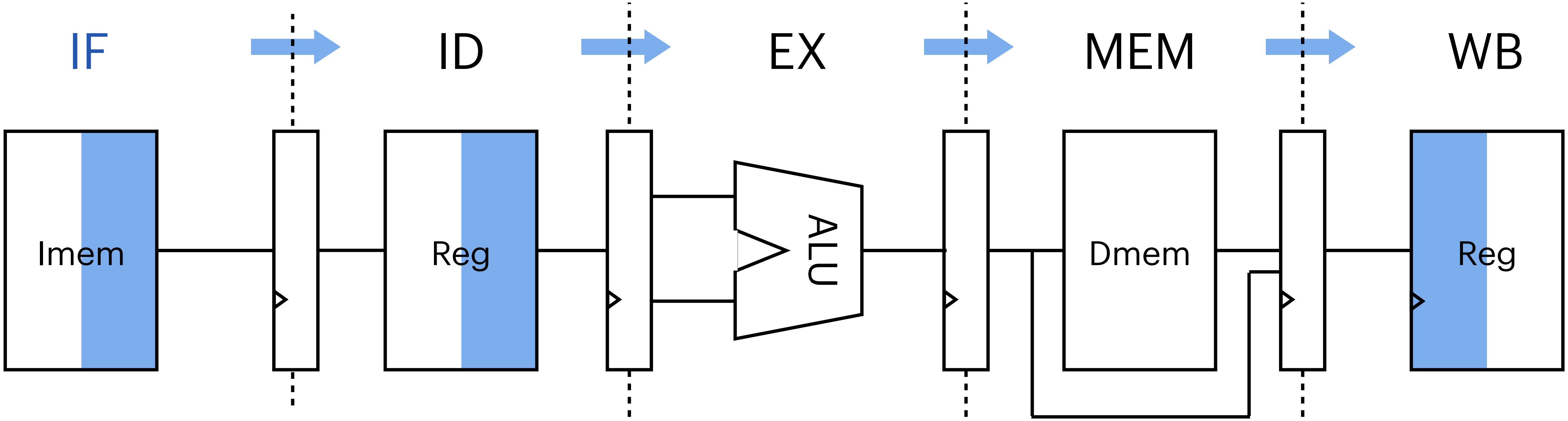
Insert Pipeline Registers



```
add t0,t1,t2
or t3,t4,t5
lw t0,8(t3)
addi t6,x0,1
xor t0,t0,t6
… …
```

$$\frac{\text{Time}}{\text{Program}} = \frac{\text{Instructions}}{\text{Program}} \cdot \frac{\text{Cycles}}{\text{Instruction}} \cdot \frac{\text{Time}}{\text{Cycle}}$$

Insert Pipeline Registers



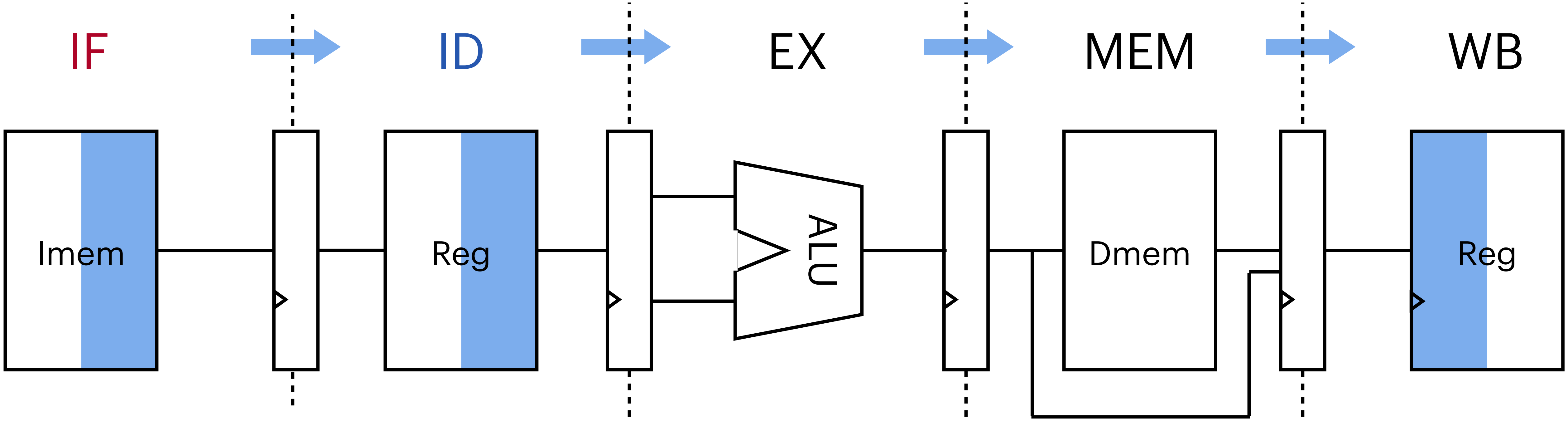
Clock cycle 1

`IF add`

Clock cycle 2 (PC + 4)

`IF or` **`ID add`**

Insert Pipeline Registers



Clock cycle 1

IF add

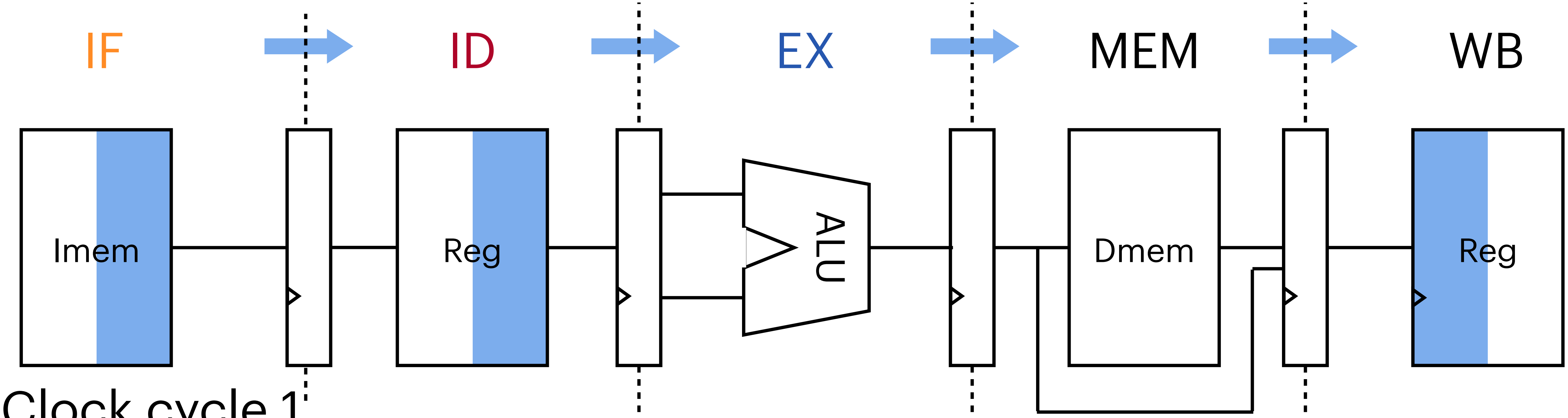
Clock cycle 2 (PC + 4)

IF or **ID add**

Clock cycle 3 (PC + 8)

IF lw **ID or** **EX add**

Insert Pipeline Registers



Clock cycle 1

`IF add`

Clock cycle 2

`IF or` **`ID add`**

Clock cycle 3

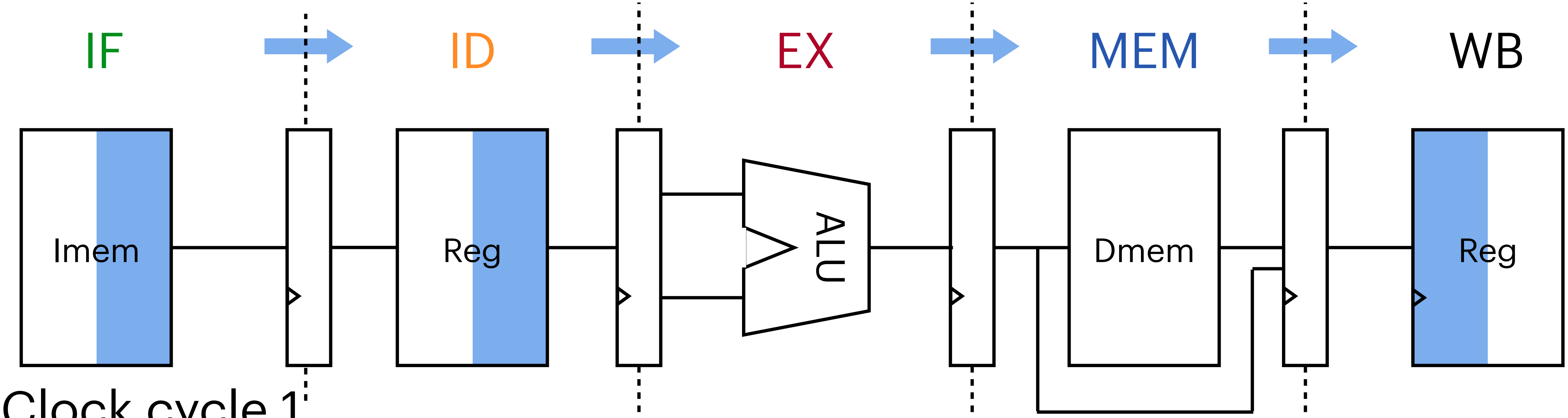
`IF lw` **`ID or`** **`EX add`**

Clock cycle 4

`IF addi` **`ID lw`** **`EX or`** **`MEM add`**

Though no MEM stage in **`add`**, the data still go through the register and consume one clock cycle

Insert Pipeline Registers



Clock cycle 1

IF add

Clock cycle 2

IF or **ID add**

Clock cycle 3

IF lw **ID or** **EX add**

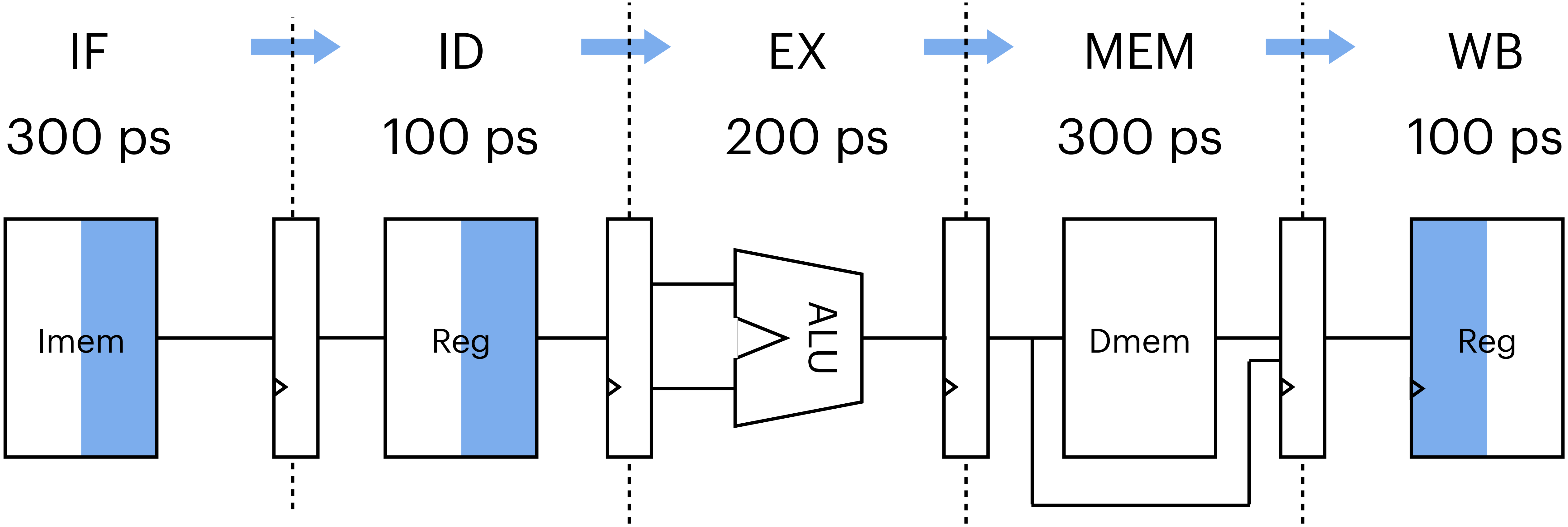
Clock cycle 4

IF addi **ID lw** **EX or** **MEM add**

Clock cycle 5

IF xor **ID addi** **ID lw** **MEM or** **WB add**

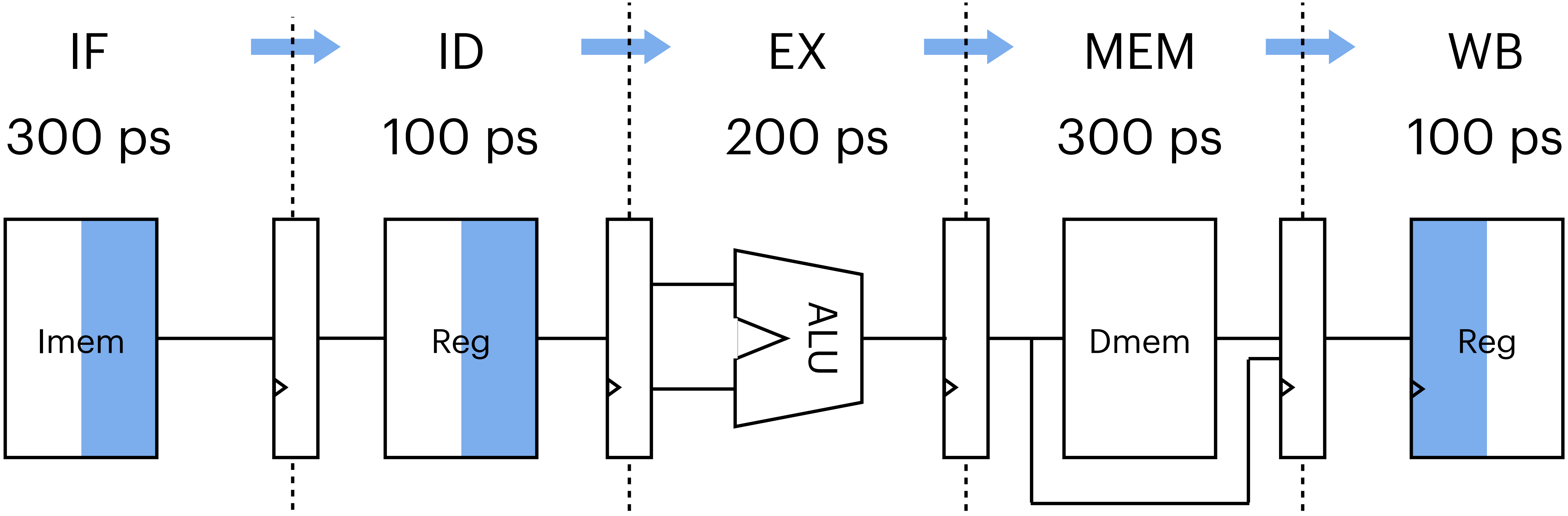
Insert Pipeline Registers



```
add t0,t1,t2
or t3,t4,t5
lw t0,8(t3)
addi t6,x0,1
xor t0,t0,t6
… …
```

$$\frac{\text{Time}}{\text{Program}} = \frac{\text{Instructions}}{\text{Program}} \cdot \frac{\text{Cycles}}{\text{Instruction}} \cdot \frac{\text{Time}}{\text{Cycle}}$$

Insert Pipeline Registers



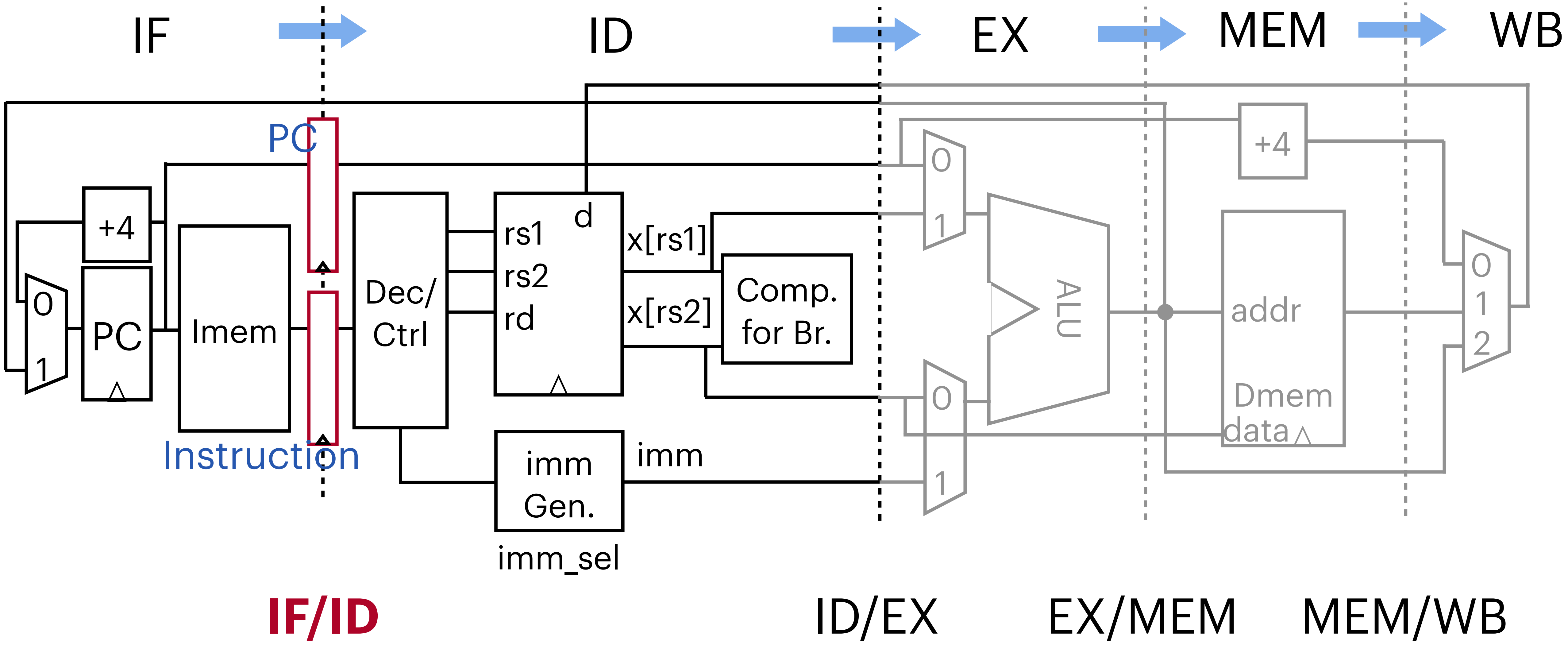
	Single-cycle	Pipelining
T_{clk}		
Clock rate/frequency		
Instruction time		
Clock per instruction		
Relative speed		
Hardware		

Question

Combinational logic in some stages takes 200 ps and in some 100 ps. Clk-Q delay is 30 ps, and setup-time is 20 ps. What is the maximum clock frequency at which a pipelined design with 10 stages can operate?

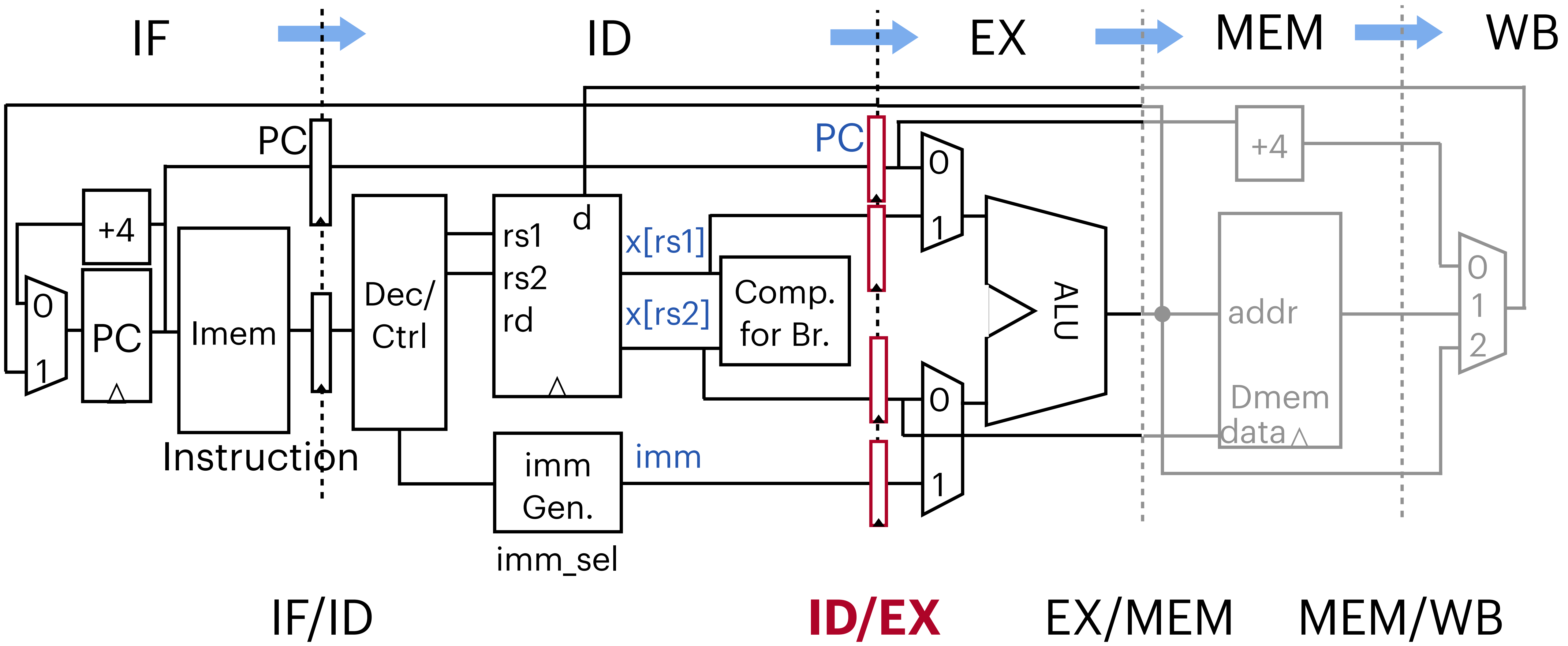
A: 10GHz
B: 5GHz
C: 6.7GHz
D: 4.35GHz
E: 4GHz

Detailed Considerations—Datapath



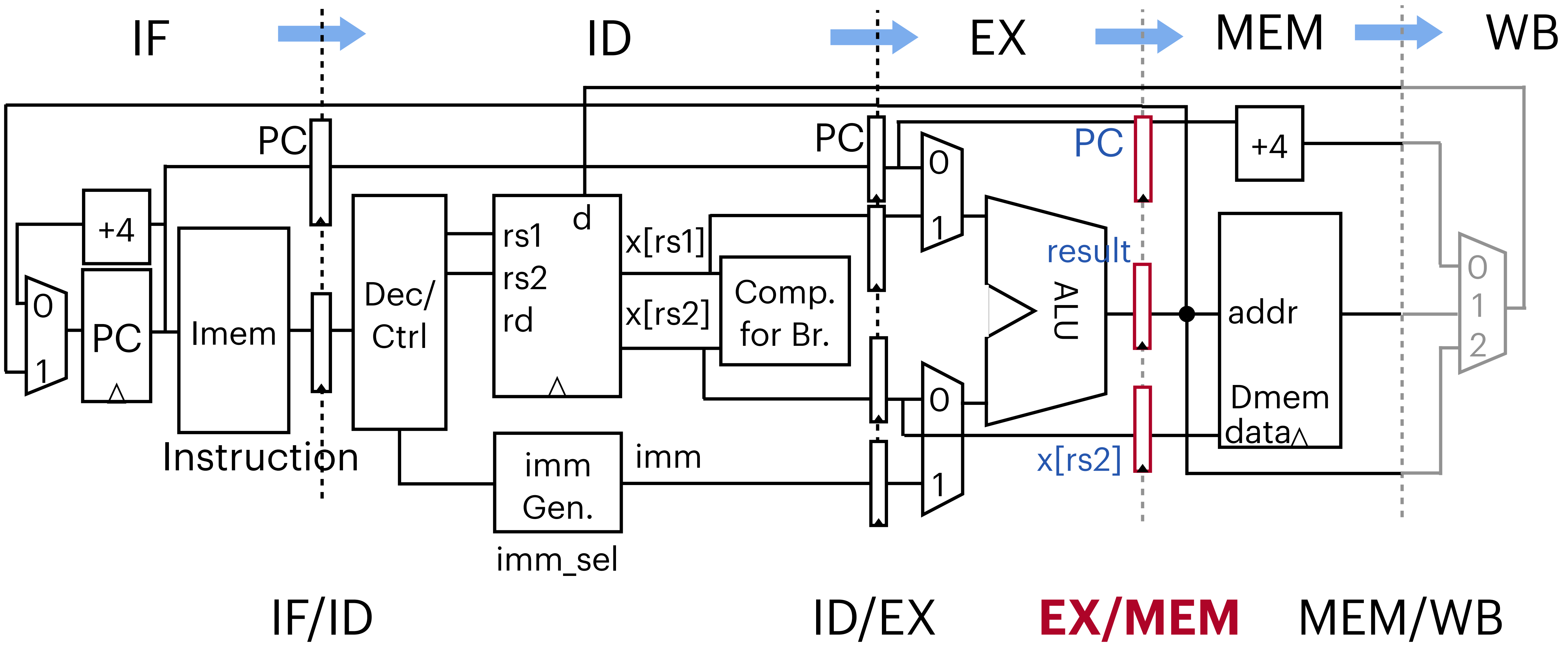
Use pipeline registers to carry instructions/ data between stages

ID/EX Pipeline Registers



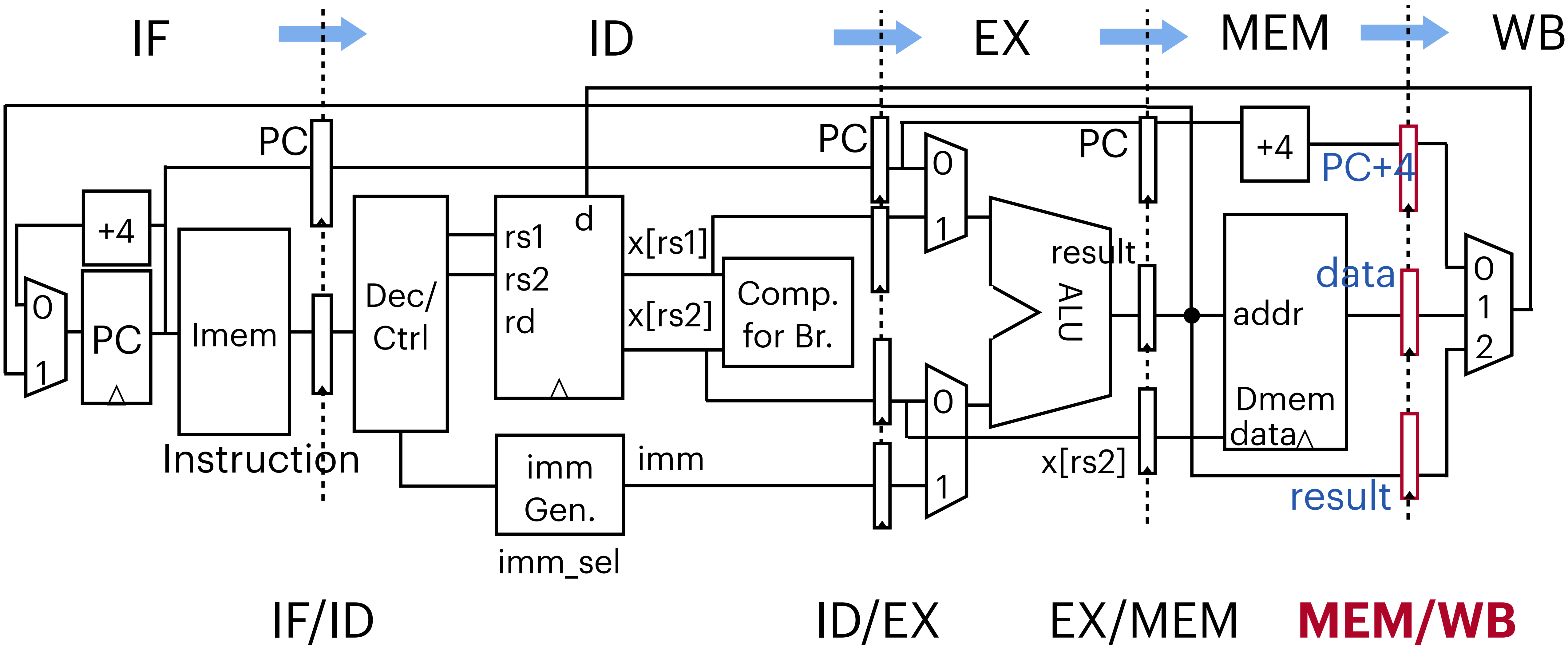
Use pipeline registers to carry instructions/ data between stages

EX/MEM Pipeline Registers



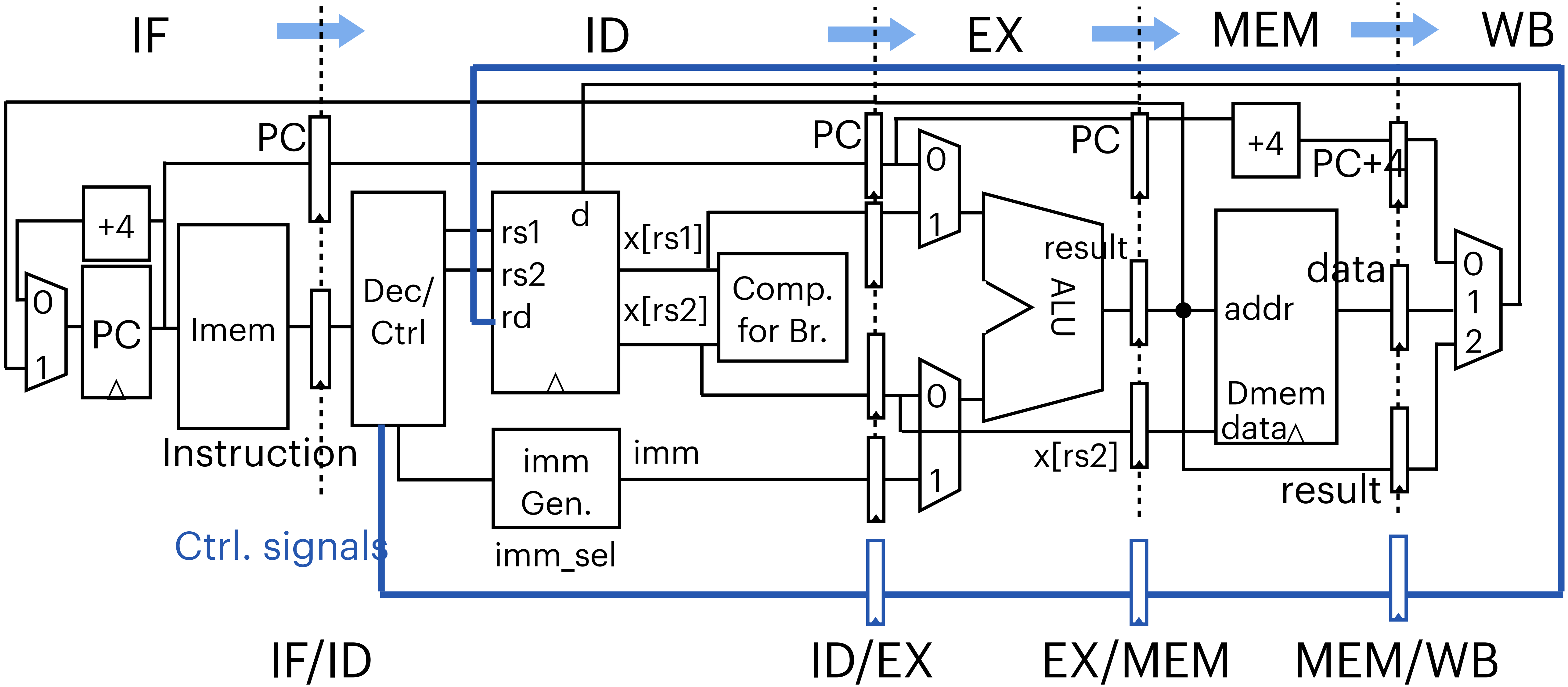
Use pipeline registers to carry instructions/ data between stages

MEM/WB Pipeline Registers



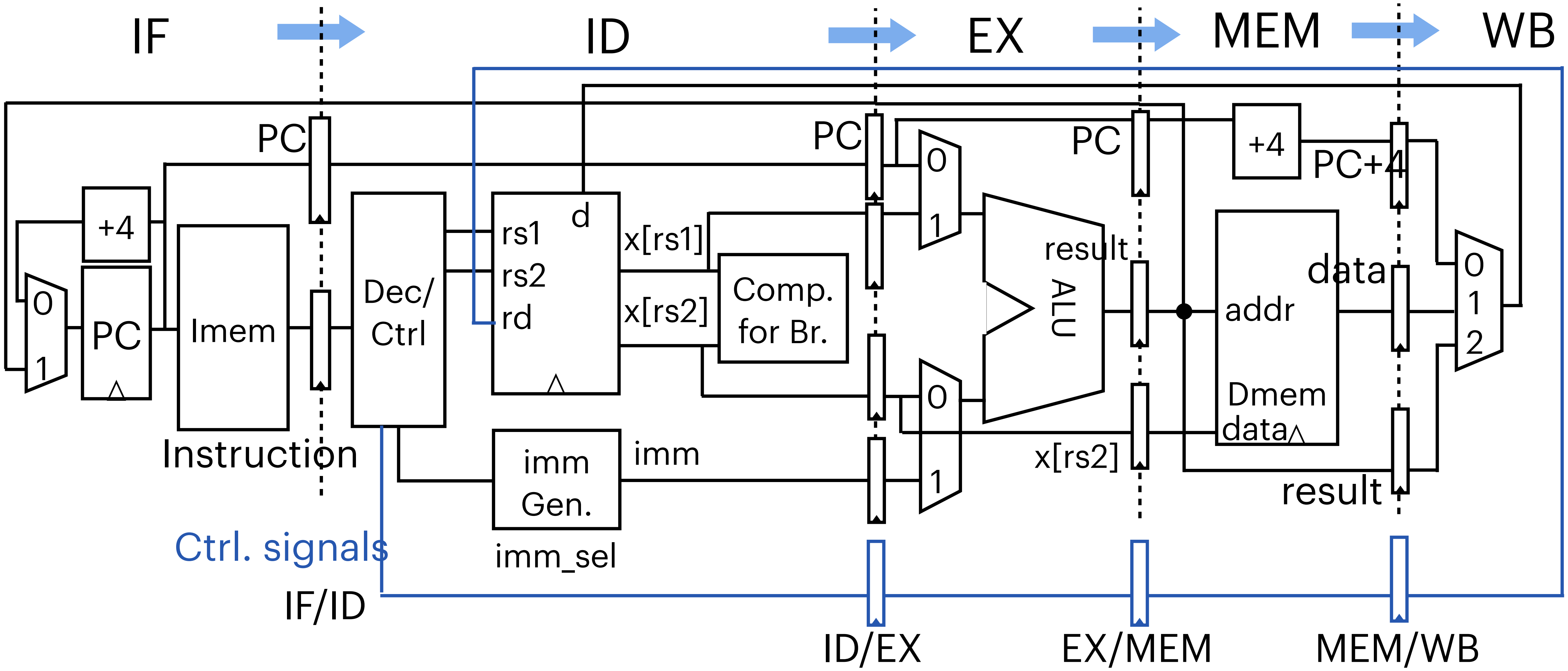
Use pipeline registers to carry instructions/ data between stages

Control Signal Pipeline Registers



Use pipeline registers to carry instructions/ data between stages

Question



```
add t0,t1,t2
or t3,t4,t5
lw t0,8(t3)
addi t6,x0,1
xor t0,t0,t6
```

Q: When `add` instruction is "*WB*-ing", which operands are selected for ALU simultaneously?

A: PC; immediate.

B: PC; t4

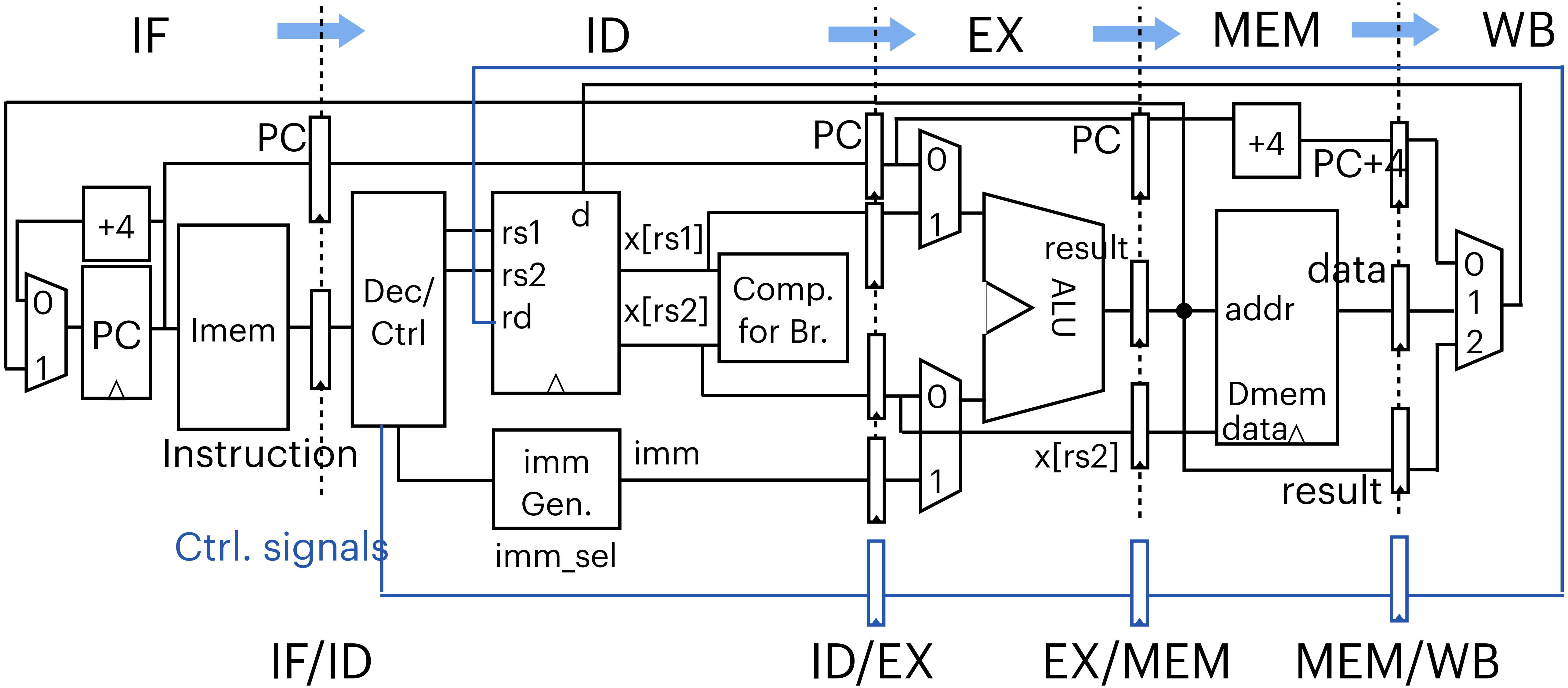
C: t3; immediate.

D: t4; t5

E: Something else.

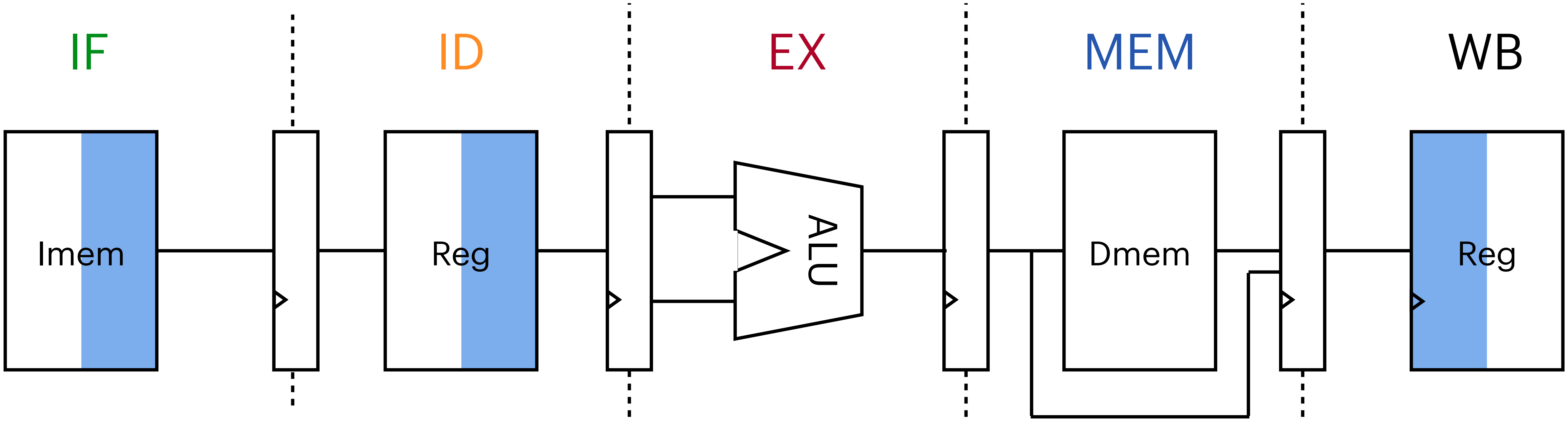
Without considering correct execution

Up to Now



Pipeline Hazards Ahead!!!

Pipeline Hazards Ahead!!!



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

Clock cycle 1

IF add

Clock cycle 2

IF lw **ID add**

Clock cycle 3

IF or **ID lw** **EX add**

Clock cycle 4

IF sw **ID** **MEM add**

Simultaneous memory read

Clock cycle 5

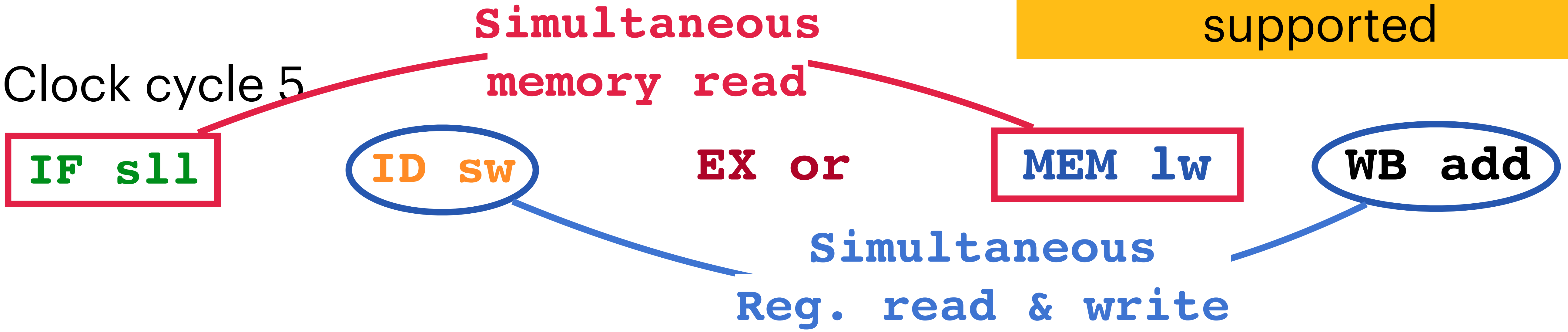
IF sll **ID sw** **EX or** **MEM lw** **WB add**

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

Structural hazard occurs if these operations not supported



Clock cycle 5

`IF sll` `ID sw` `EX or` `MEM lw` `WB add`

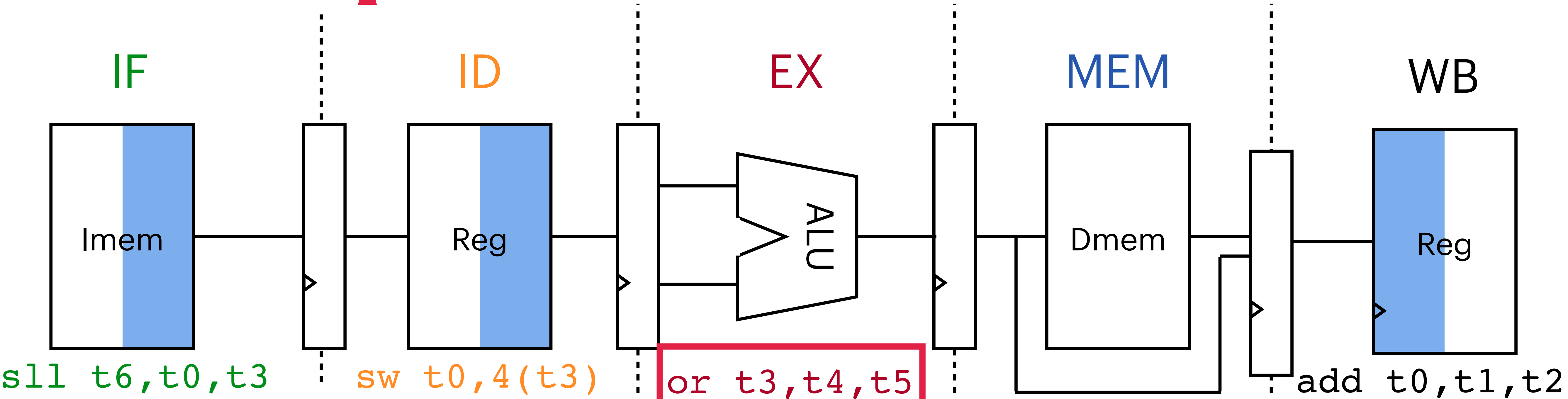
`Simultaneous memory read`

`Simultaneous Reg. read & write`

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:
  - Instructions take turns using the resource
  - Stall while the resource is busy
- Solution 2:
  - Can always avoid structural hazards by adding more HW
  - In our current design, *structural hazards are not an issue*
    - Separated instruction and data memory (cache)
    - Synchronized write & unsynchronized read, support simultaneous read/write; Reg. has two read ports

Pipeline Hazards Ahead!!!



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

t3 has not been calculated/updated

Clock cycle 1				
IF add				
Clock cycle 2				
IF lw	**ID add**			
Clock cycle 3				
IF or	**ID lw**	**EX add**		
Clock cycle 4				
IF sw	**ID or**	**EX lw**	**MEM add**	
Clock cycle 5				
IF sll	**ID sw**	**EX or**	**MEM lw**	**WB add**

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

Three Types of Pipeline Hazards

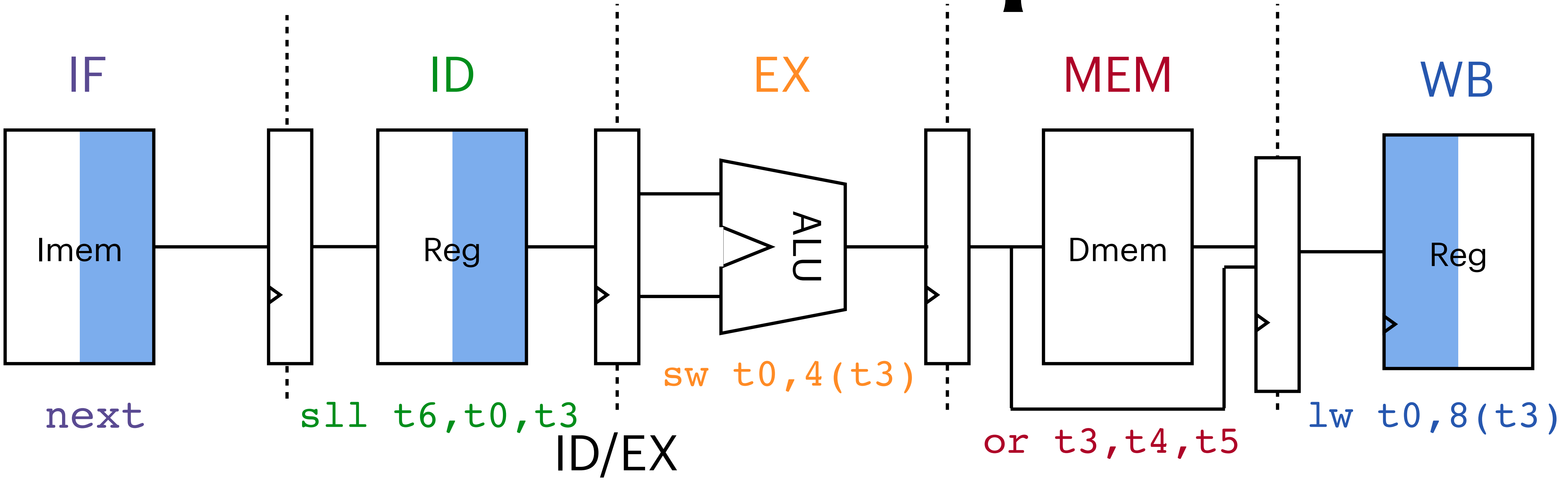
A hazard is a situation in which a planned instruction cannot execute in the "proper" clock cycle.

1. Structural hazard:
   - Hardware does not support access across multiple instructions in the same cycle

2. **Data hazard**:
   - Instructions have data dependency
   - Occurs when an instruction reads a register before a previous instruction has finished writing to that register

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

`t0` as an Example



next

sll t6,t0,t3

sw t0,4(t3)

or t3,t4,t5

lw t0,8(t3)

At the posedge of clock cycle 7, `t0` is updated

At the same time, ID/EX register captures the previous `t0`, which is erroneous

Data hazard

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

Clock cycle 5

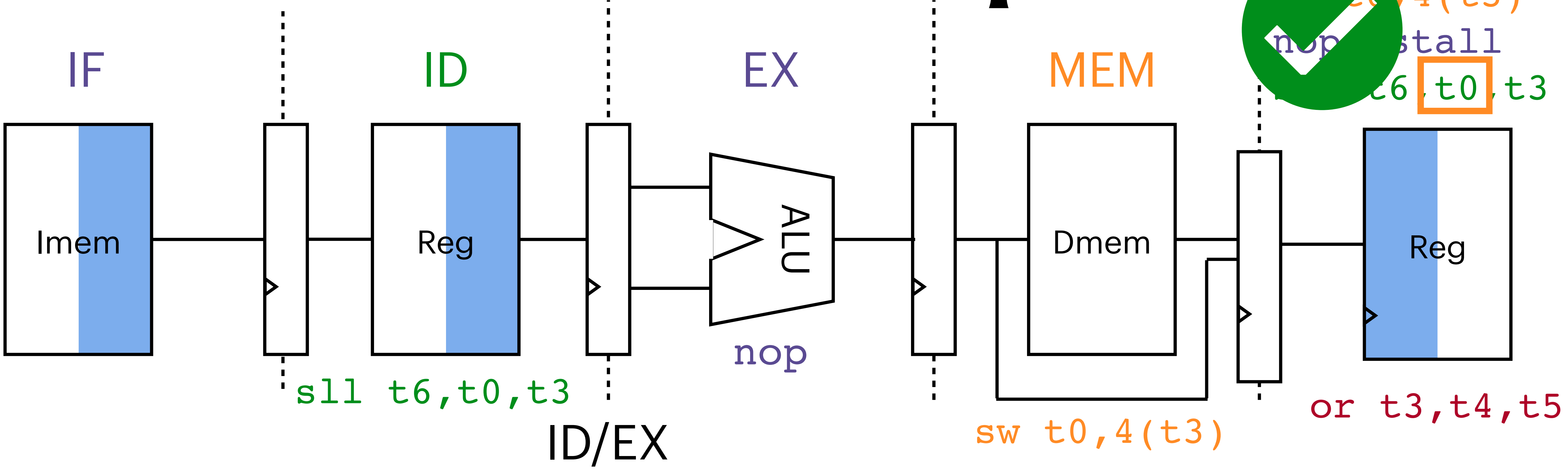
IF sll **ID sw** **EX or** **MEM lw** **WB add**

Clock cycle 6

IF next **ID sll** **EX sw** **MEM or** **WB lw**

`t0` as an Example

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



Clock cycle 5

IF nop **ID sw** **EX or** **MEM lw** **WB add**

Clock cycle 6

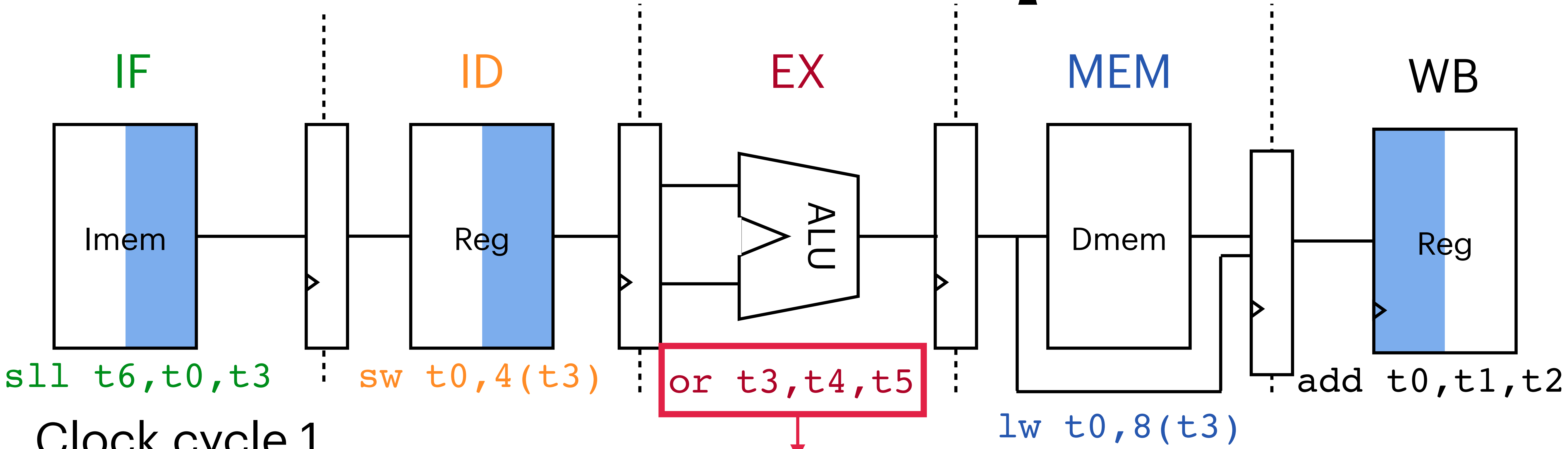
IF sll **ID nop** **EX sw** **MEM or** **WB lw**

Clock cycle 7: `t0` updated at the beginning of this clock cycle (posedge)

IF next **ID sll** **EX nop** **MEM sw** **WB or**

Covered by RegFile write-then-read in the same clock cycle

Another Example



t3 has not been calculated/updated

Clock cycle 1

IF add

Clock cycle 2

IF lw **ID add**

Clock cycle 3

IF or **ID lw** **EX add**

Clock cycle 4

IF sw **ID or** **EX lw** **MEM add**

Clock cycle 5

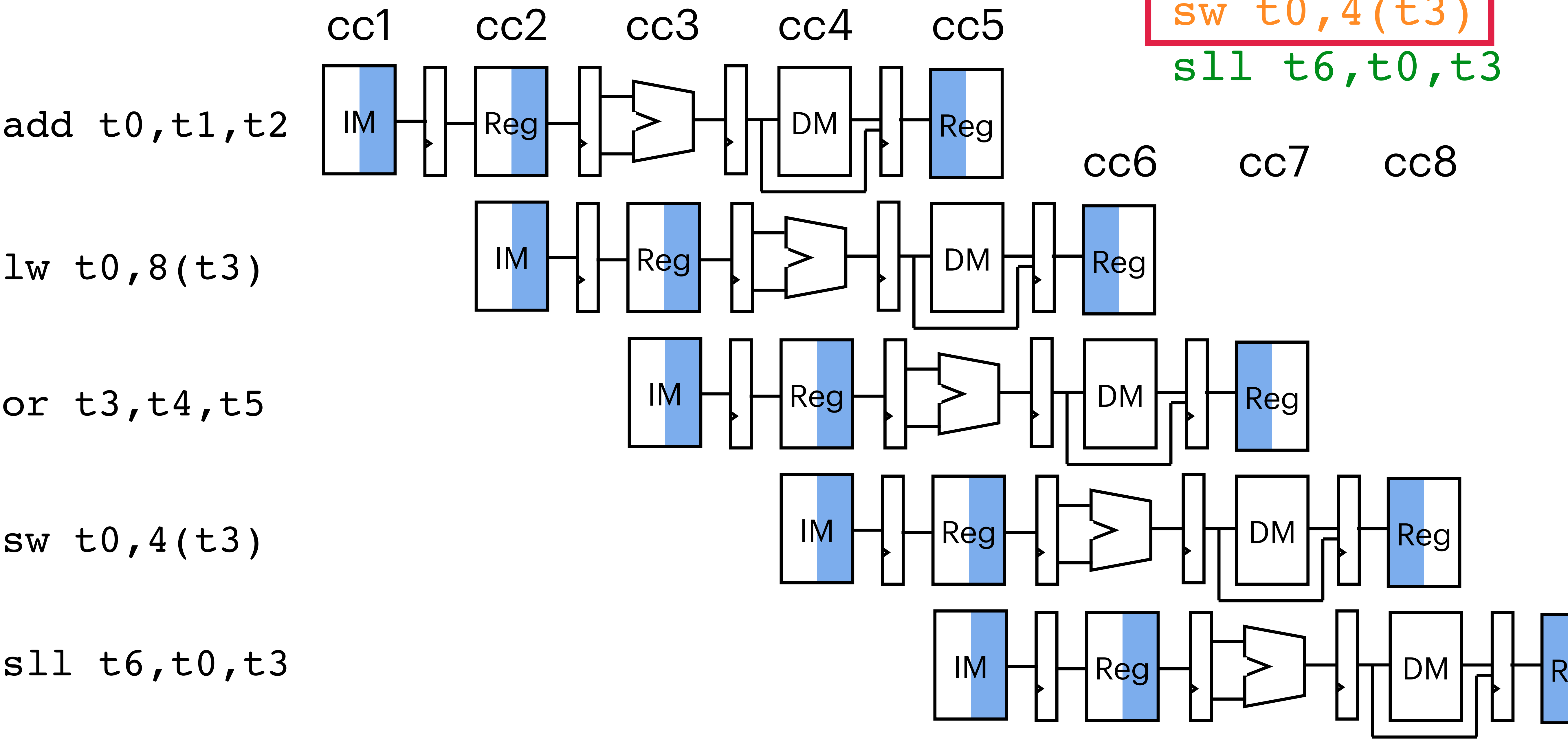
IF sll **ID sw** **EX or** **MEM lw** **WB add**

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

Another Example

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

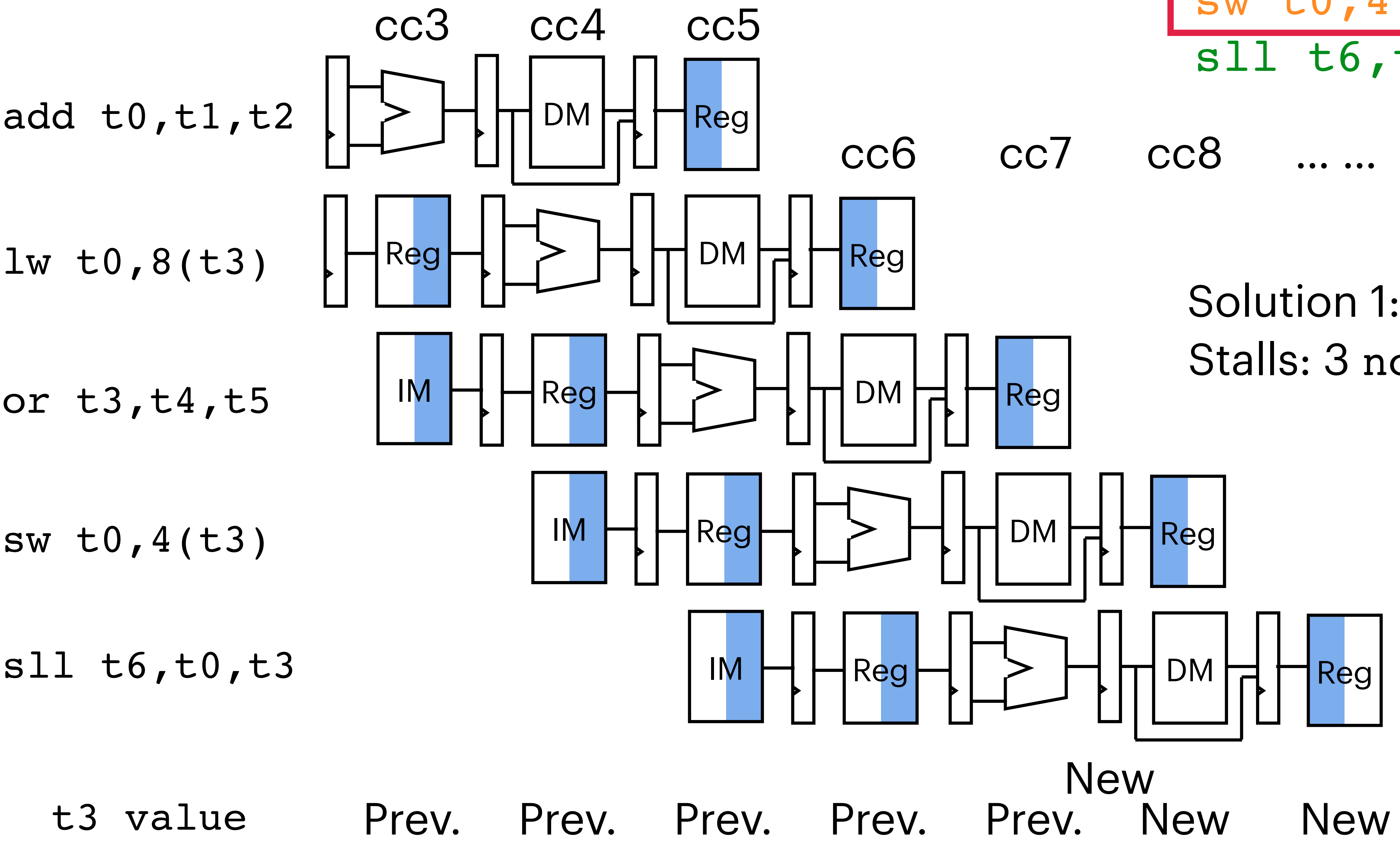
Expand the pipeline both temporally and spatially



Solution 1

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

Starting from cc3



Solution 1:
Stalls: 3 nops