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

CS 110
Computer Architecture
Pipeline

Instructors:
Siting Liu & Yuan Xiao
Course website: https://faculty.sist.shanghaitech.edu.cn/liust/courses/CS110.html
School of Information Science and Technology (SIST)
ShanghaiTech University

2025/4/14

Administratives

- Lab 7: project 1.1 check.
- HW 4 released, ddl April 21st! Start early!
  - The template is updated, please use the most up-to-date starter file
- Project 1.2 released, DDL April 27th.
- Mid-term I Apr. 23rd 8:00am-10:00am (Location: TBD)
  - Prepare your cheat sheet in advance!
  - Materials until Apr. 21st lecture.
- Discussion (SPST 4-122 18:00 - 19:40) schedule
  - Apr. 17th on mid-term I review by Chaofan Li;
  - Apr. 24th on datapath by Yuxuan Li.

Outline

- Starting this lecture, we will improve the performance of our CPU
- Performance evaluation
- Pipeline
- Hazards

Performance

- Recall the great ideas in CA
  - Abstraction (layers of representation/interpretation)
  - Moore's law (designing through trends)
  - Make the common case fast
  - Principle of locality (memory hierarchy)
  - Parallelism (pipeline as a special case)
  - Performance measurement & improvement
  - Dependability via redundancy

“Iron law” of performance

- **CPU (execution) time** (ignore I/O, operating system overhead etc.)

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

- Can be obtained by profiling or hardware counter
- ISA (e.g., RISC vs. CISC)
- The program itself
- Compiler
- Programming language
- etc.

- Short as “CPI”
- Microarchitecture implementation or circuit design/ISA
- Compiler
- Program
- Programming language

- Microarchitecture implementation or circuit design/ISA

“Iron law” Example

- **Calculate average CPI**

Program A	A-instruction	B-instruction	C-instruction
CPI	2	2	4
Percentage	20%	40%	40%

- **Calculate CPU (execution) time**
  - CPU frequency 2.5 GHz
  - Program A has in total 10^6 instructions

“Iron law” Example

- **Calculate average CPI**

Program A	A-instruction	B-instruction	C-instruction
CPI	2	2	4
Percentage	20%	40%	40%

Average CPI = 2*20%+2*40%+4*40% = 2.8

- **Calculate CPU (execution) time**
  - CPU frequency 2.5 GHz
  - Program A has in total 10^6 instructions

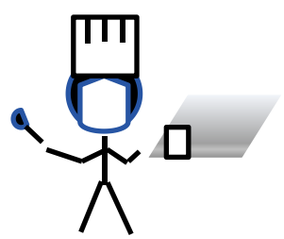
Different equation if given IPC!

CPU time = Instruction count * Average CPI * Clock cycle
= 10^6 * 2.8 * 1/2.5G
= 1.12 * 10^{-3} = 1.12 ms

Performance

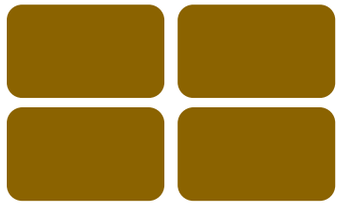
- Recall the great ideas in CA
  - Abstraction (layers of representation/interpretation)
  - Moore's law (designing through trends)
  - Make the common case fast
  - Principle of locality (memory hierarchy)
  - Parallelism (**pipeline** as a special case, instruction-level parallelism)
  - Performance measurement & improvement
  - Dependability via redundancy

Pipe that line!

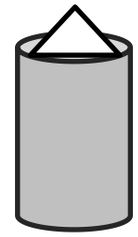


Check-out

Veggies

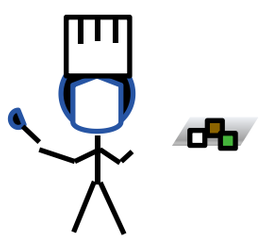


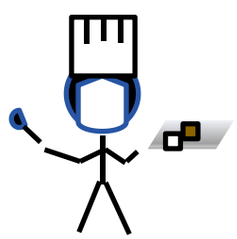
Meat

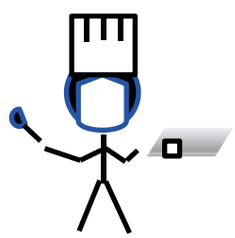


Rice

Pipe that line!



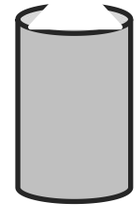




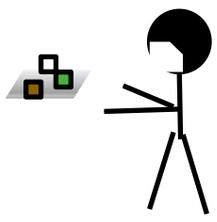
Check-out

Veggies

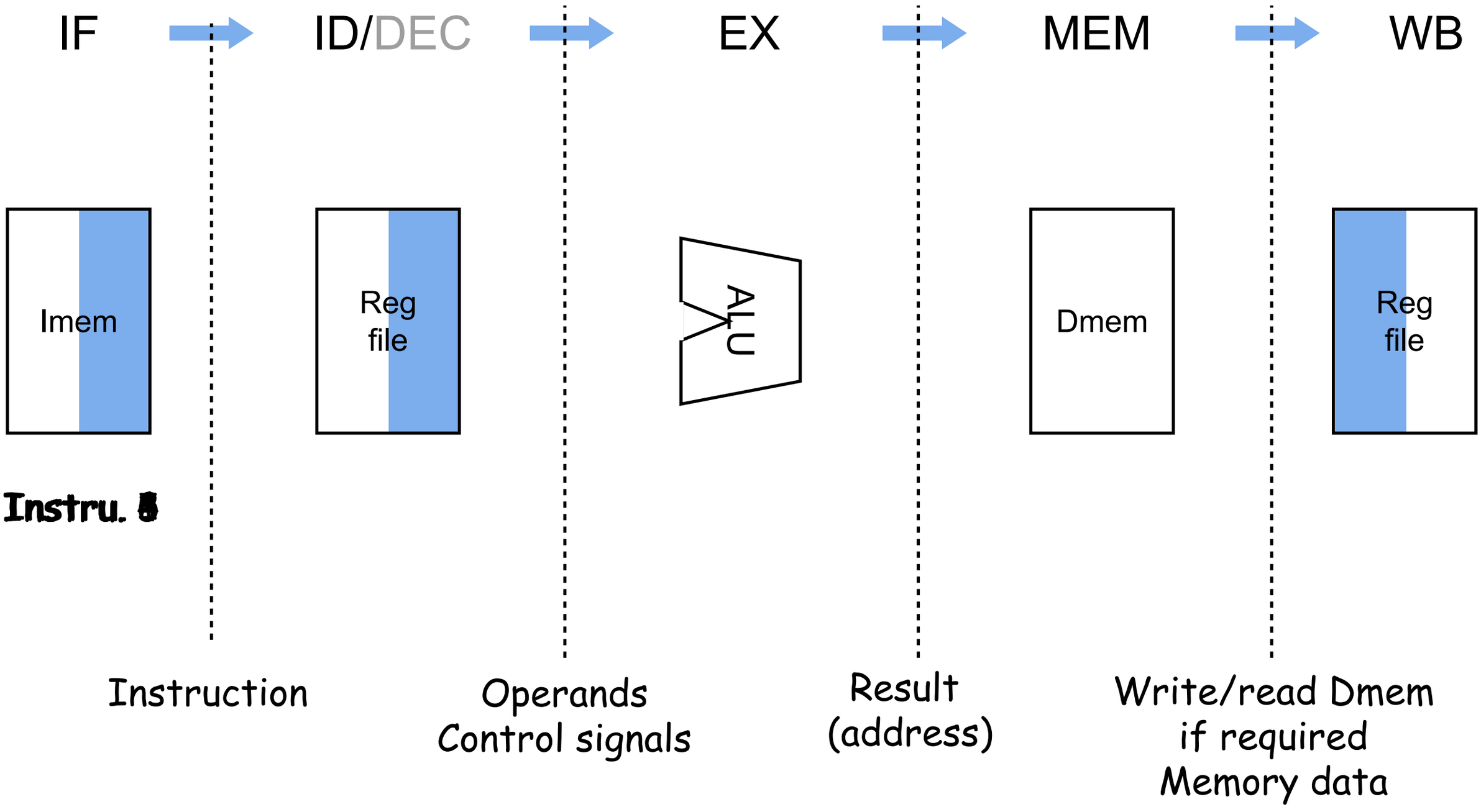
Meat



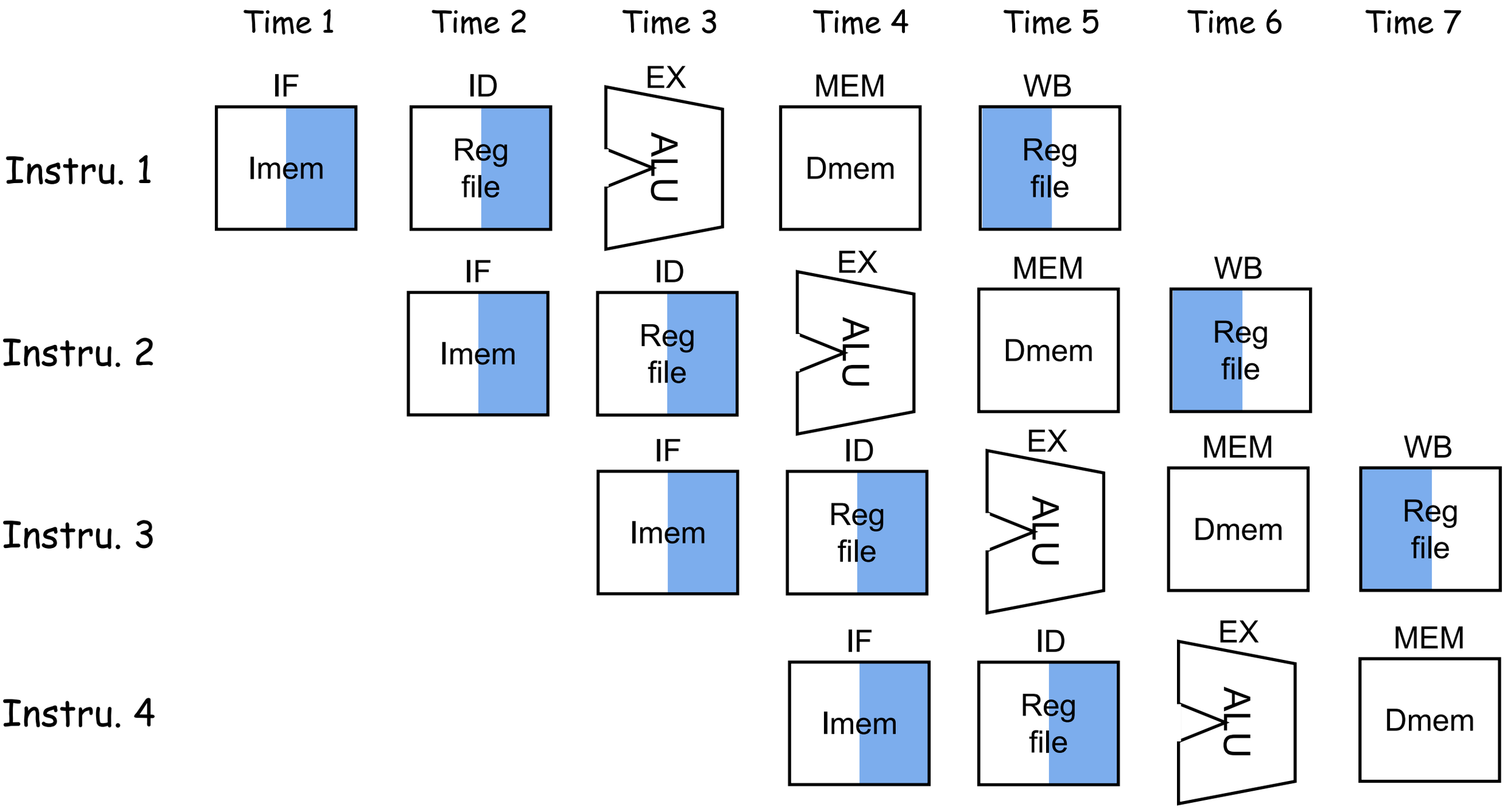
Rice



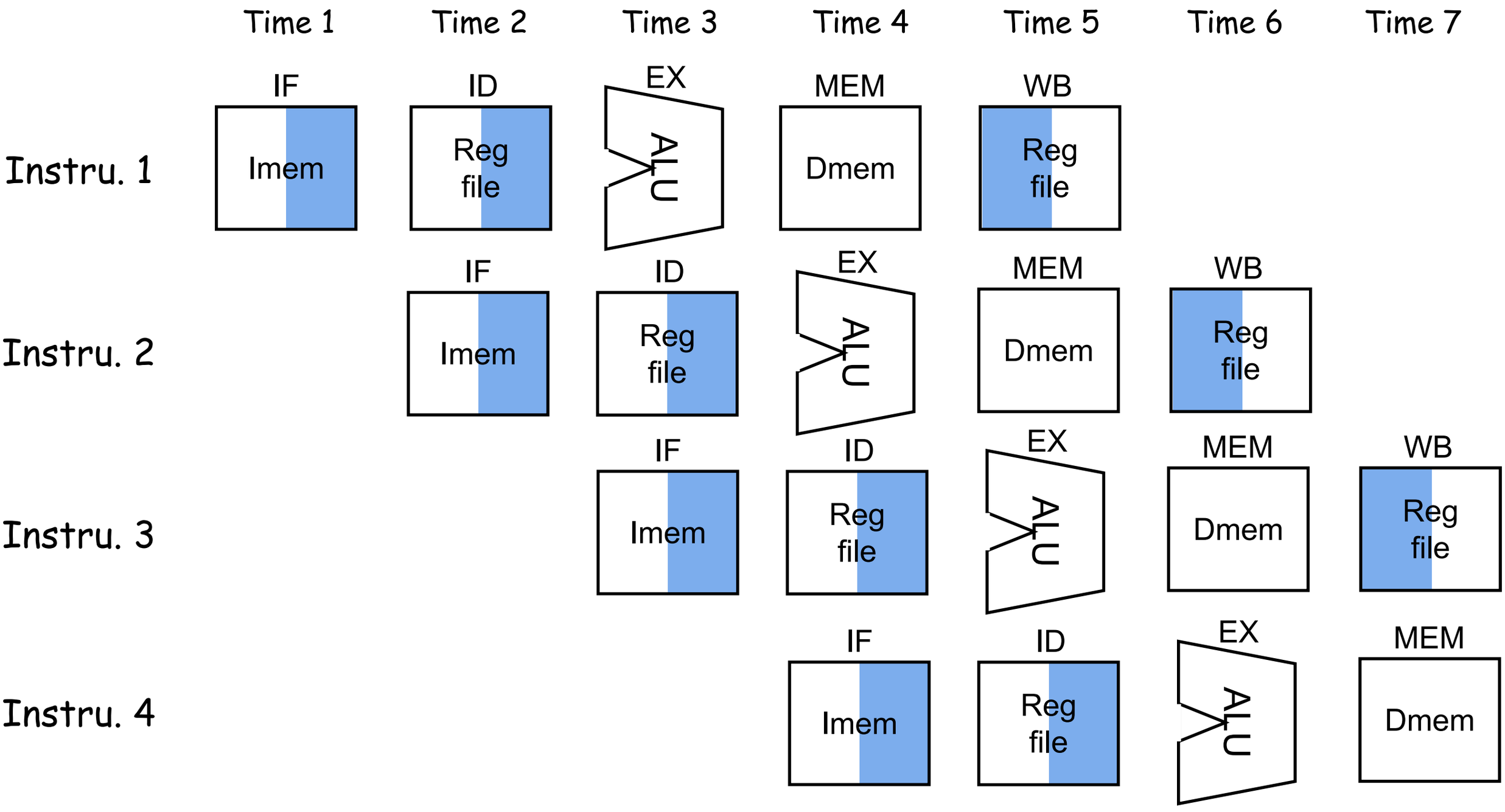
Simplify the CPU with 5 stages



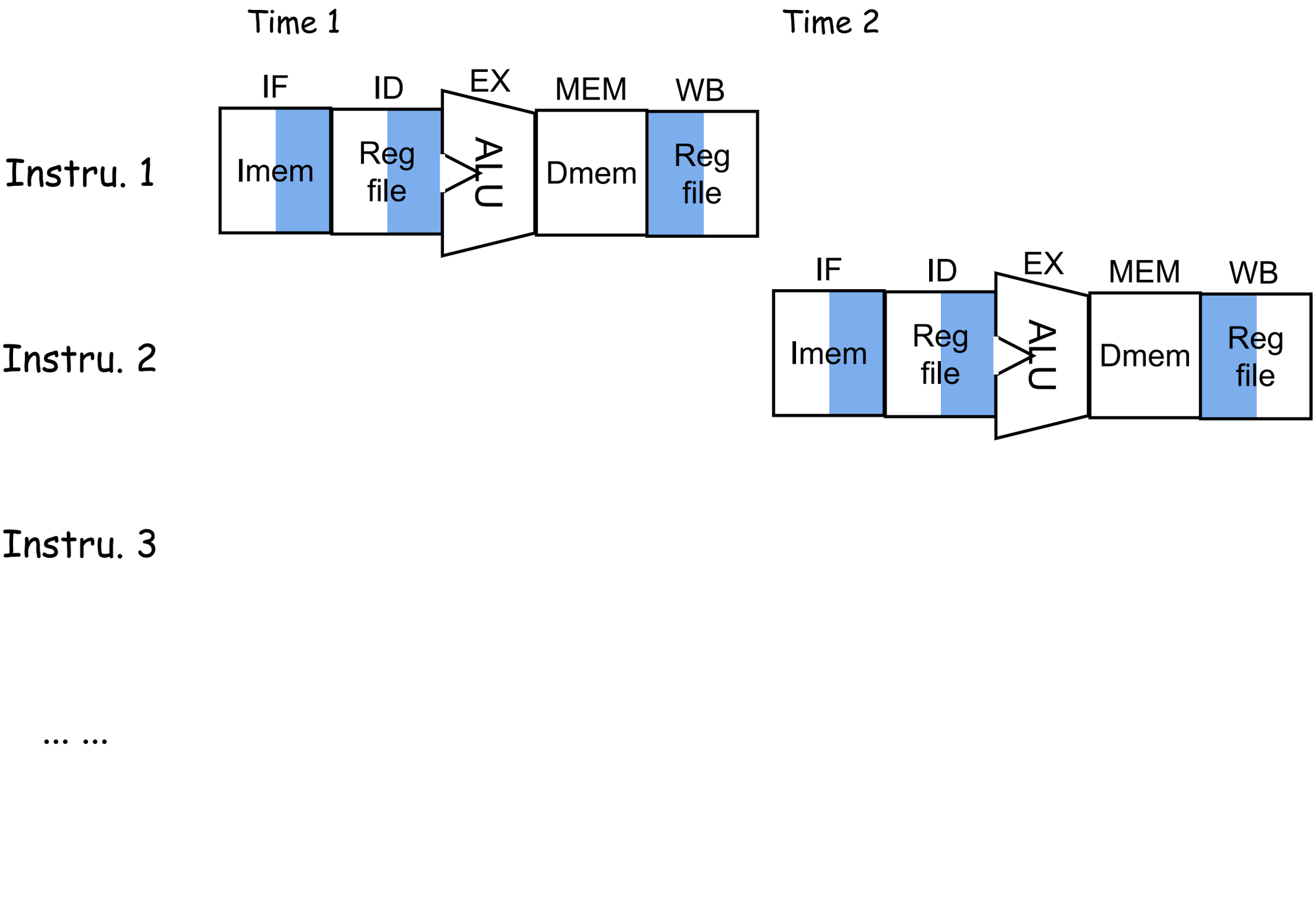
Make an analogy



Make an analogy



Previously for a single-cycle CPU



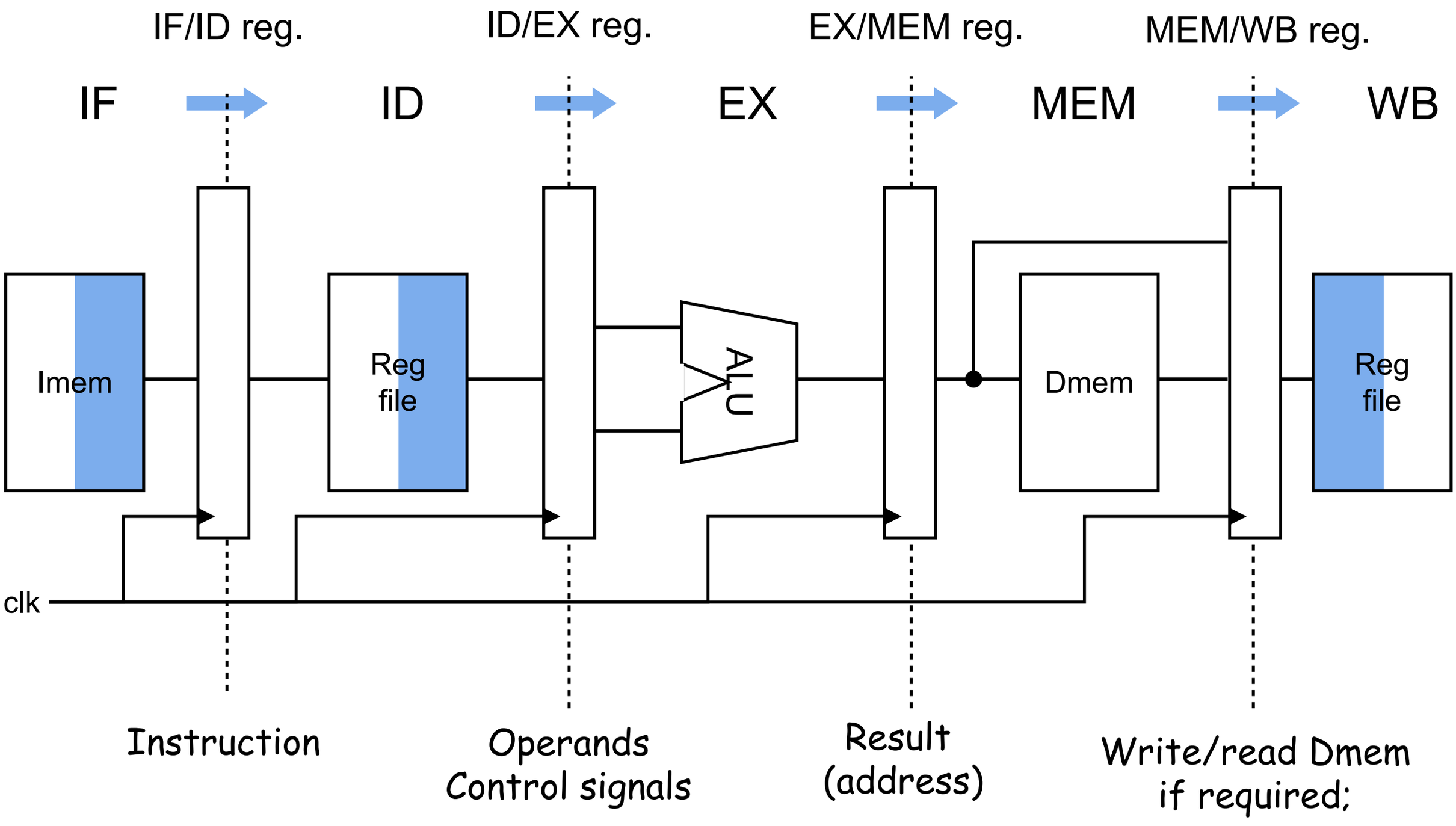
Observations

- Each instruction still take 5 stages/time slots to complete, no matter pipelined or not
- Period of the time slot is different
  - Single-cycle CPU: $t_{IF} + t_{ID} + t_{EX} + t_{MEM} + t_{WB}$
  - Pipelined CPU: $\max\{t_{IF}, t_{ID}, t_{EX}, t_{MEM}, t_{WB}\}$
- Use the “iron law” of performance, pipelined CPU is faster

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

- How to make the CPU pipelined?

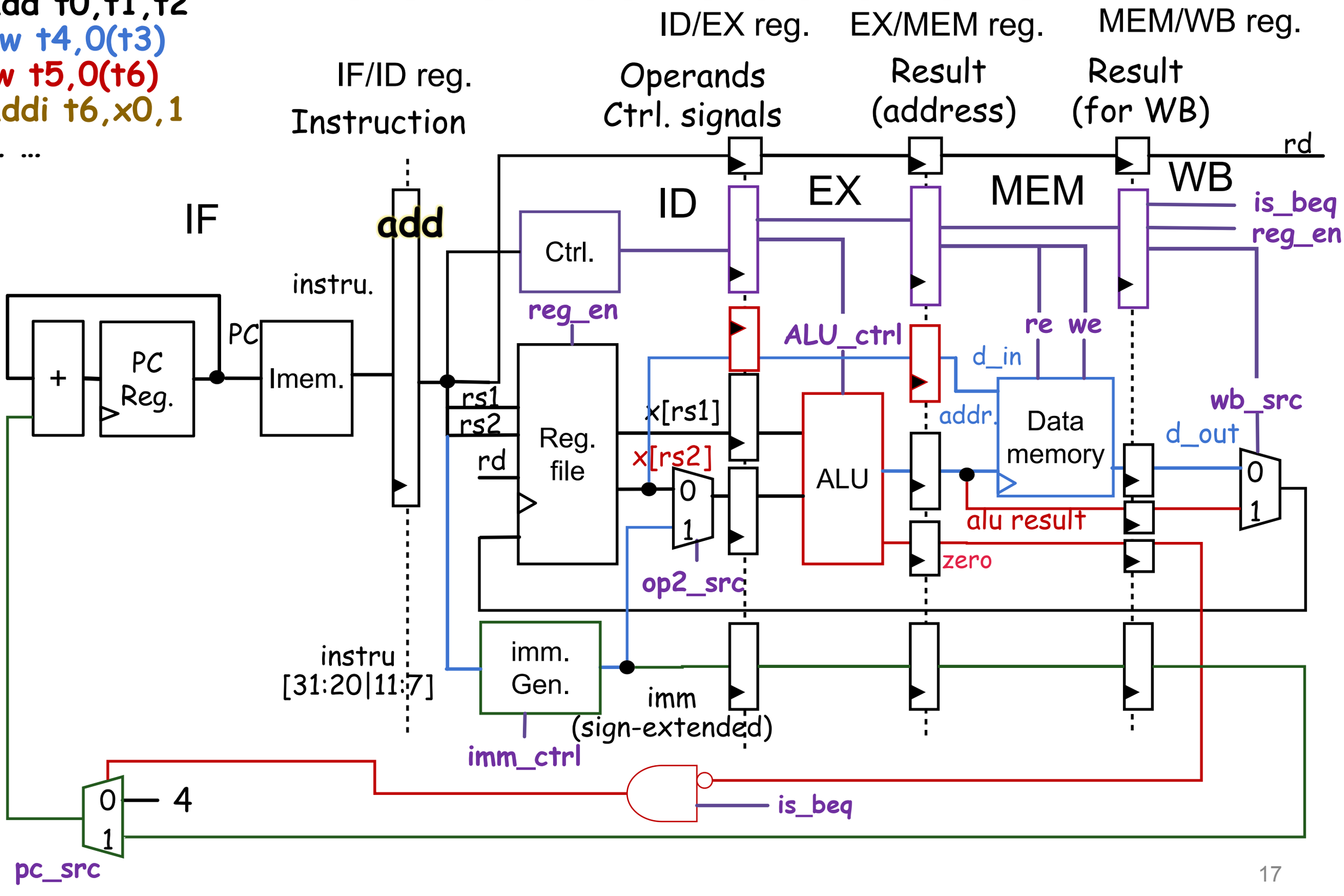
Insert pipeline registers



Critical path decided by the slowest stage

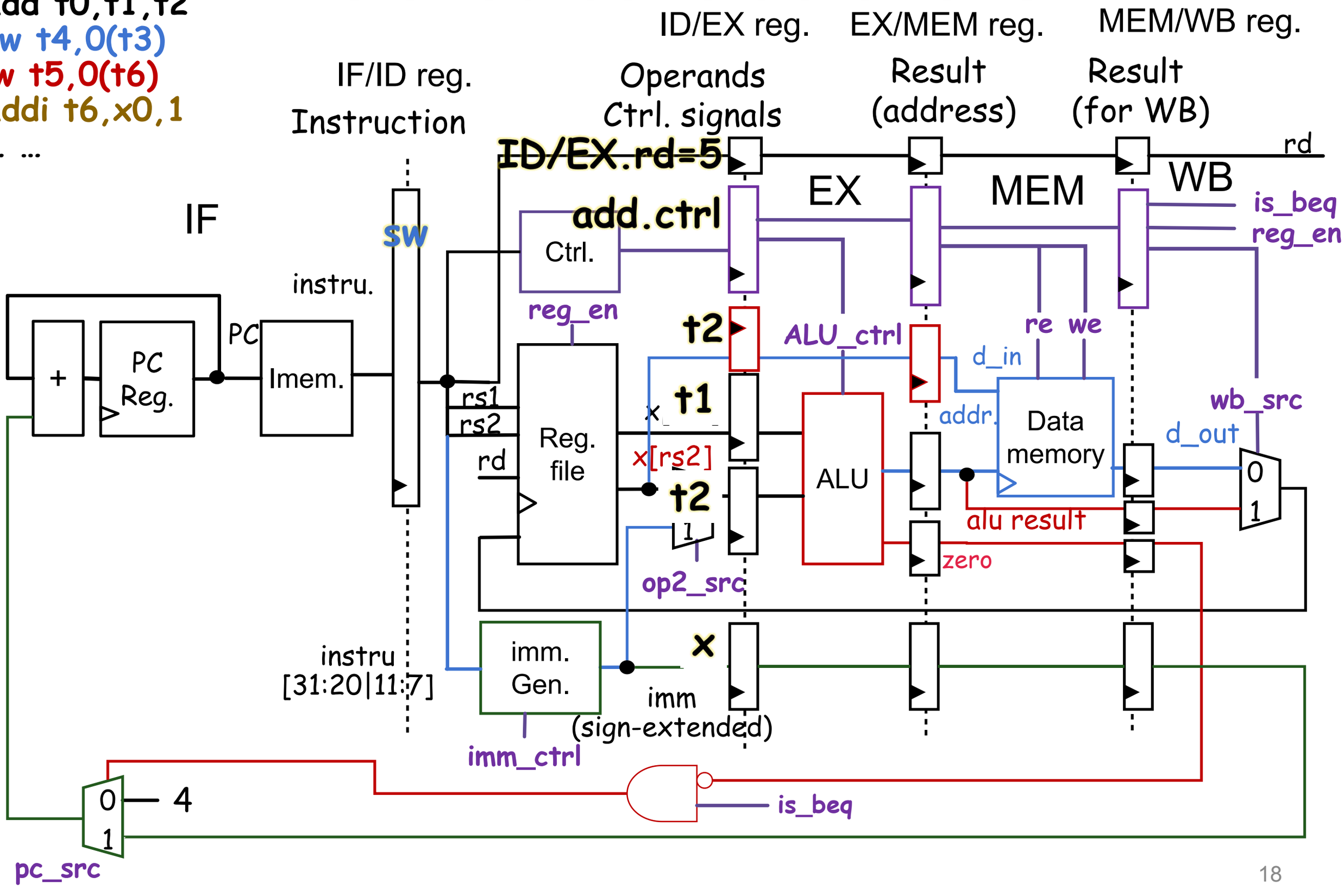
Detailed considerations

```
add t0,t1,t2
sw t4,0(t3)
lw t5,0(t6)
addi t6,x0,1
... ...
```



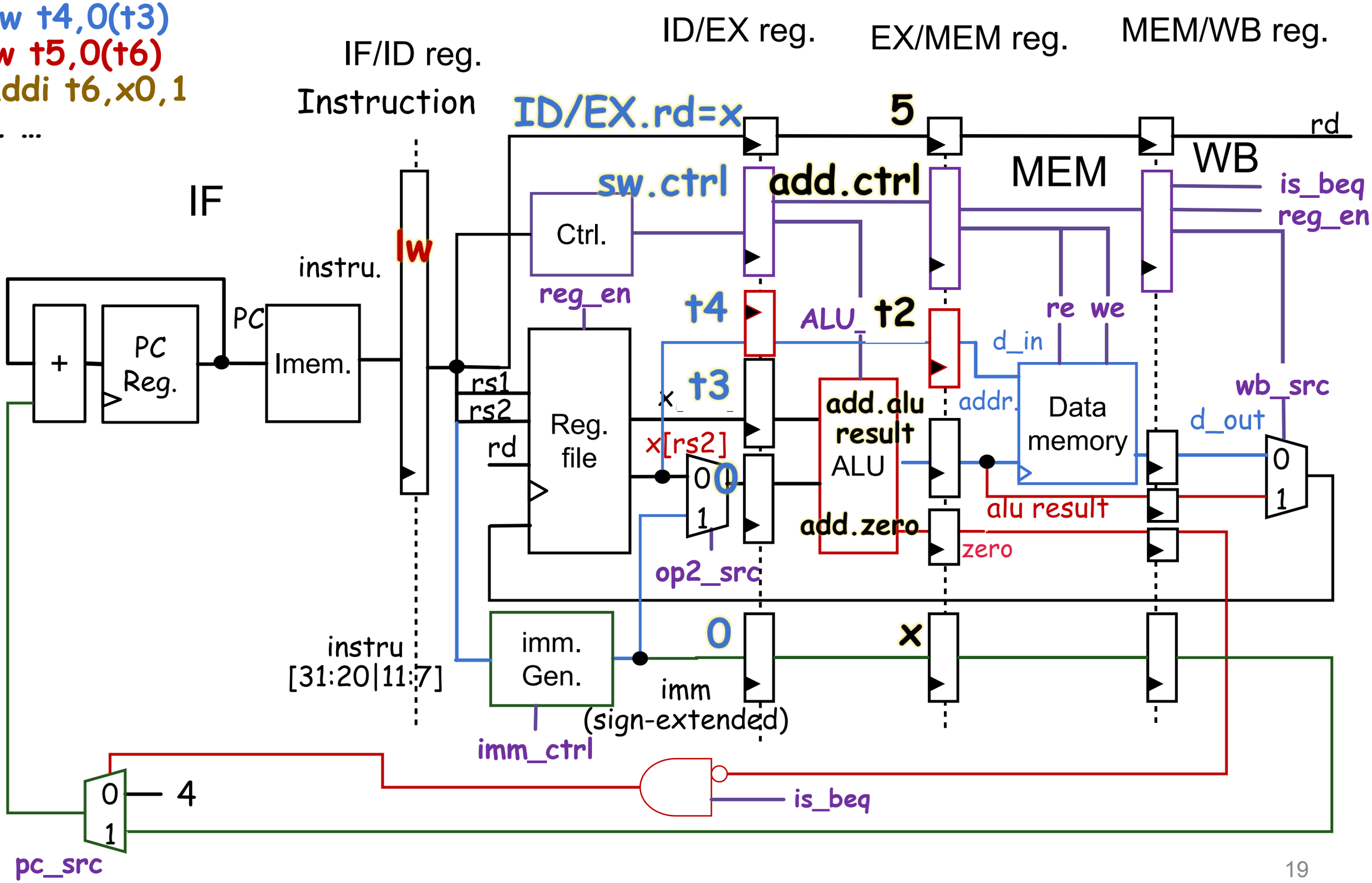
Detailed considerations

```
add t0,t1,t2
sw t4,0(t3)
lw t5,0(t6)
addi t6,x0,1
... ...
```



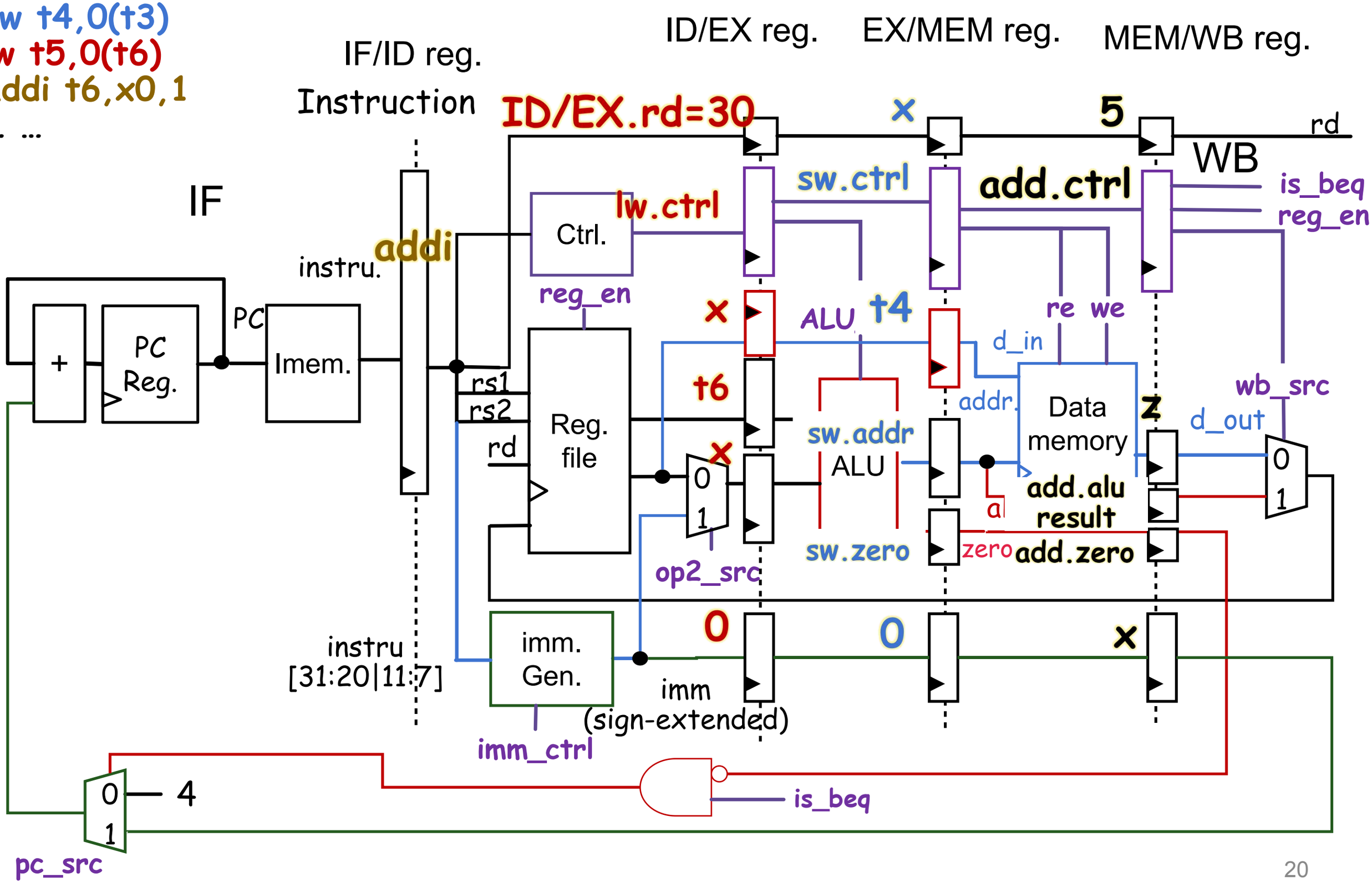
Detailed considerations

```
add t0,t1,t2
sw t4,0(t3)
lw t5,0(t6)
addi t6,x0,1
... ...
```



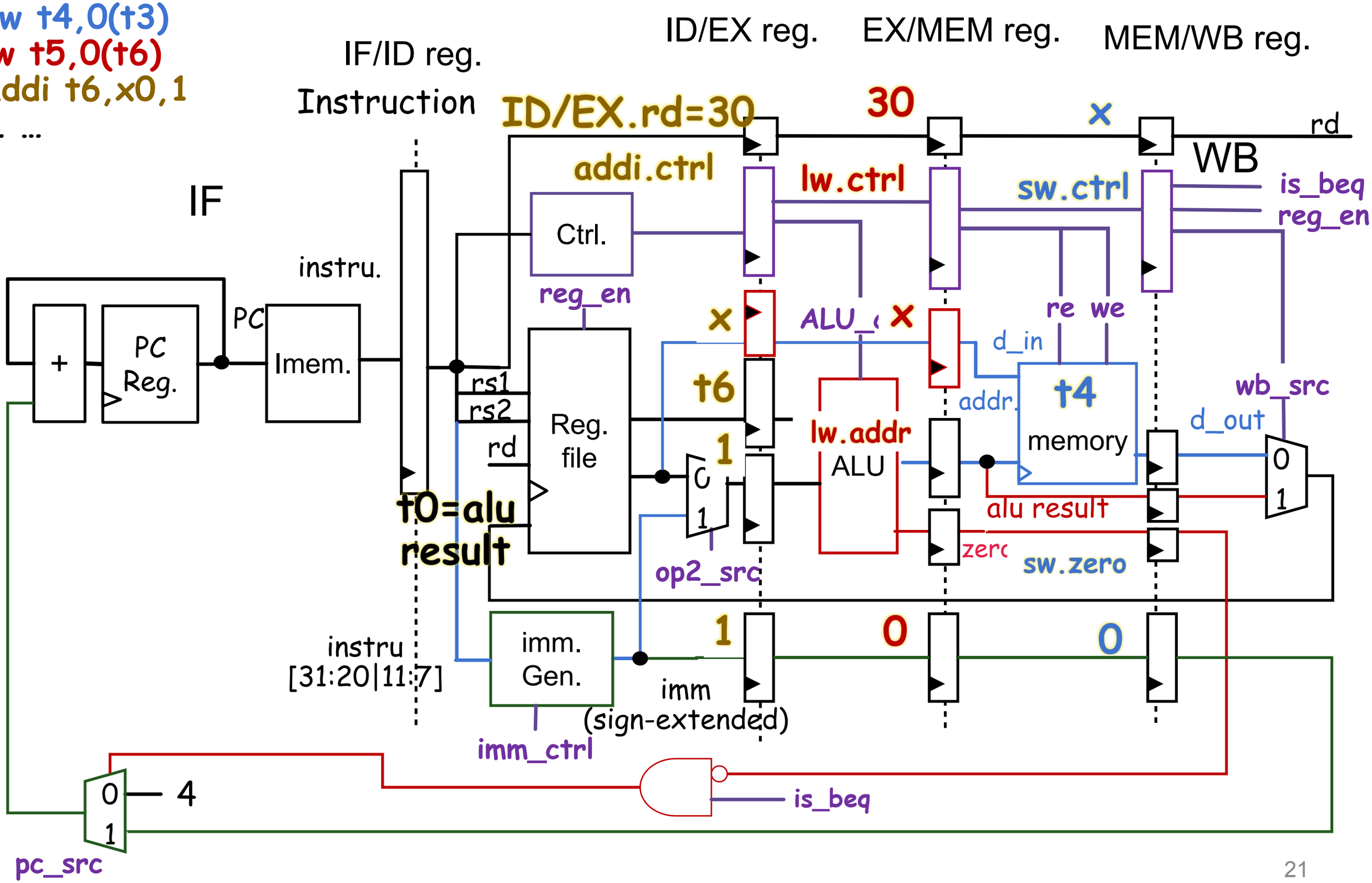
Detailed considerations

```
add t0,t1,t2
sw t4,0(t3)
lw t5,0(t6)
addi t6,x0,1
... ...
```



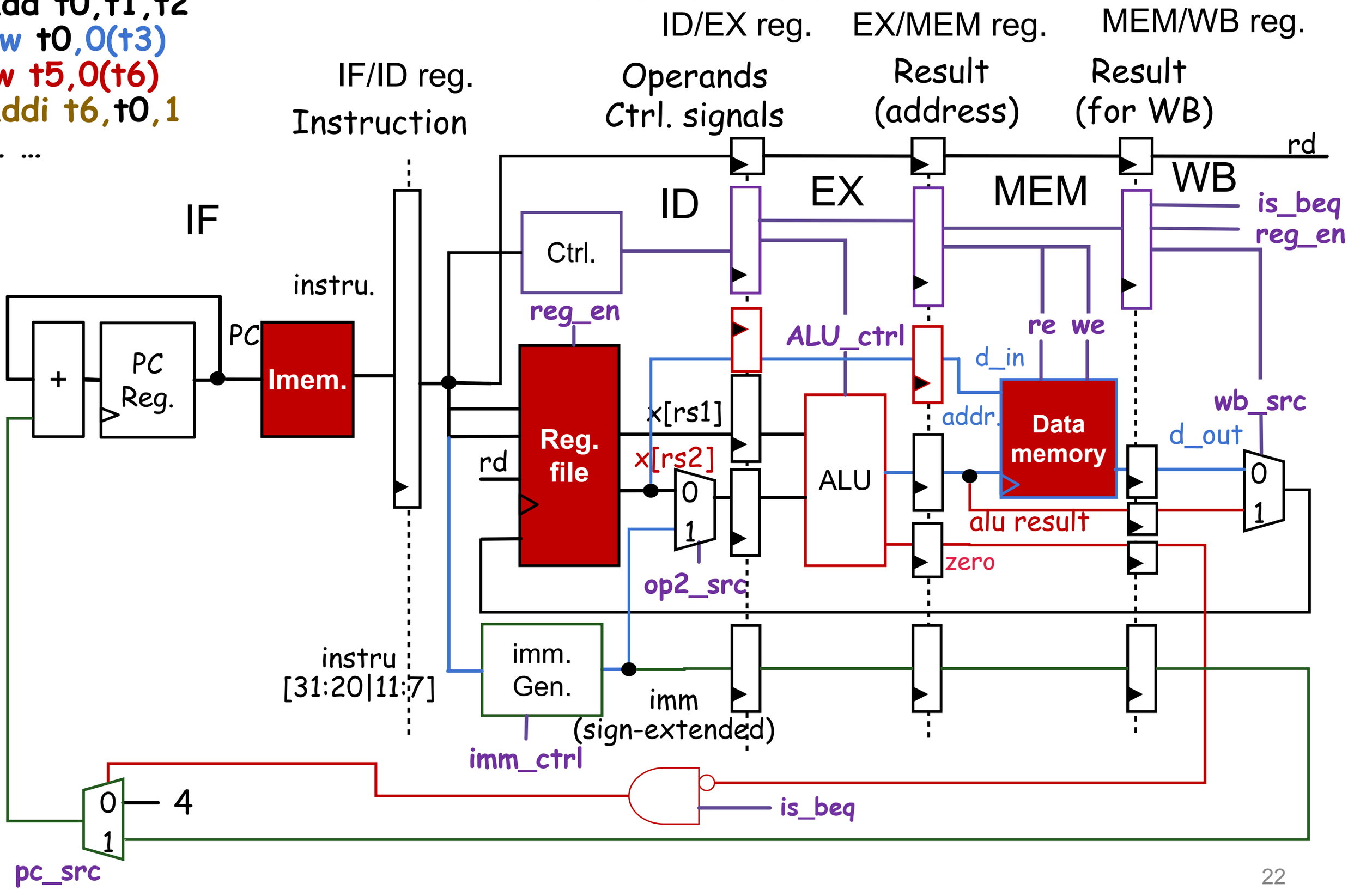
Detailed considerations

```
add t0,t1,t2
sw t4,0(t3)
lw t5,0(t6)
addi t6,x0,1
... ...
```



Hazards ahead!!!

```
add t0,t1,t2
sw t0,0(t3)
lw t5,0(t6)
addi t6,t0,1
... ...
```



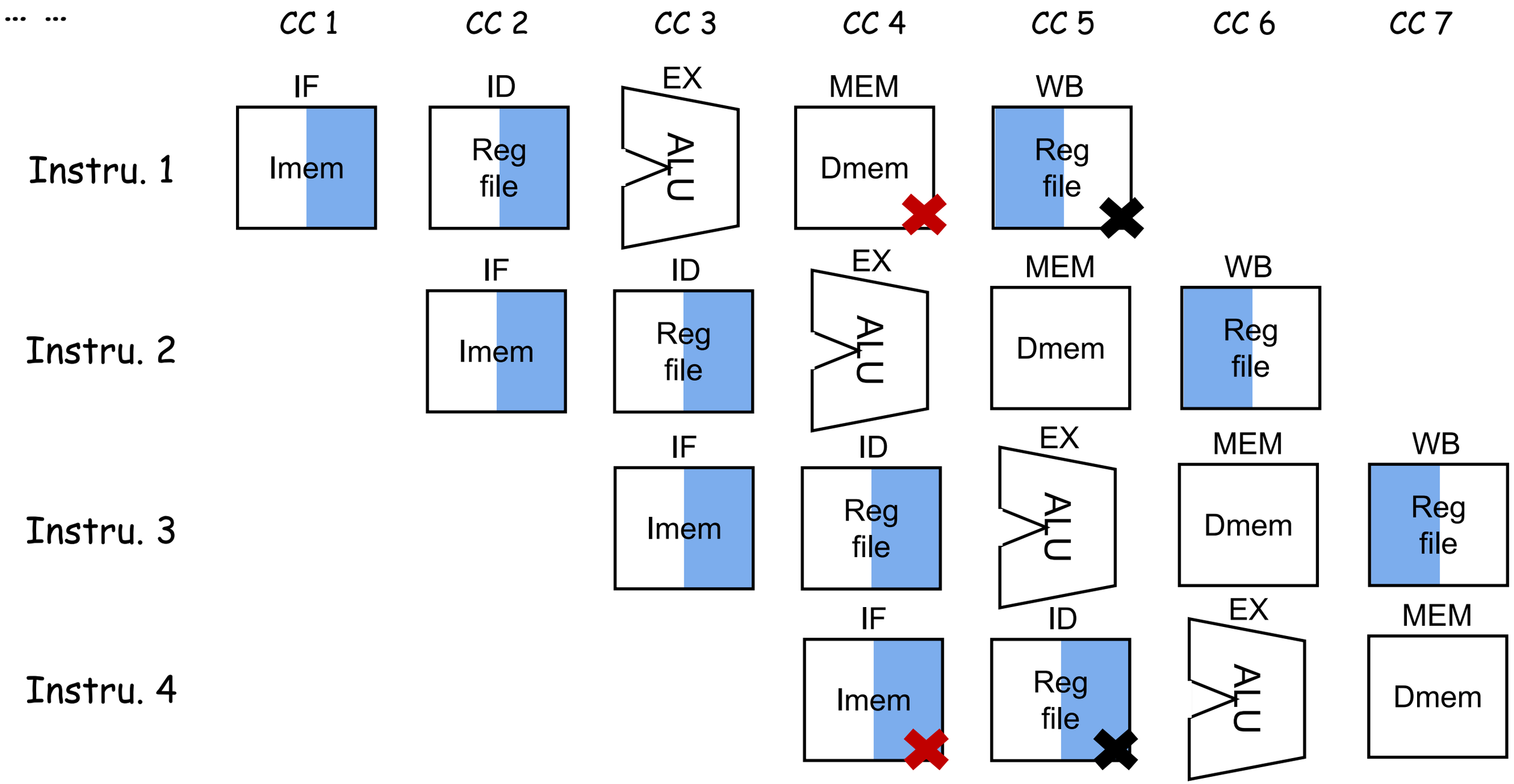
Hazards ahead!!!

- A **hazard** is a situation in which a planned instruction cannot execute in the “proper” clock cycle.

- **Structural hazards**
- Data hazards
- Control hazards

Structural hazards

```
lw t0,0(t2)
sw t0,0(t3)
lw t5,0(t6)
addi t6,t0,1
... ...
```



Structural hazards

- Caused by hardware limitations. Two or more instructions in the pipeline compete for a single physical resource
- Can be solved by
  - Seperate instruction and data memory (real CPU uses instruction cache and data cache)
  - or using dual-port memory (input multiple addresses, output multiple data) (general ways to solve structural hazards, add more hardware)
  - Assume register file write at rising edge (in the textbook “the first half clock cycle”), read arbitrarily (in the textbook “the second half clock cycle”), and design the hardware with this feature
  - Instructions take turns to use the physical resource (wait/stall)

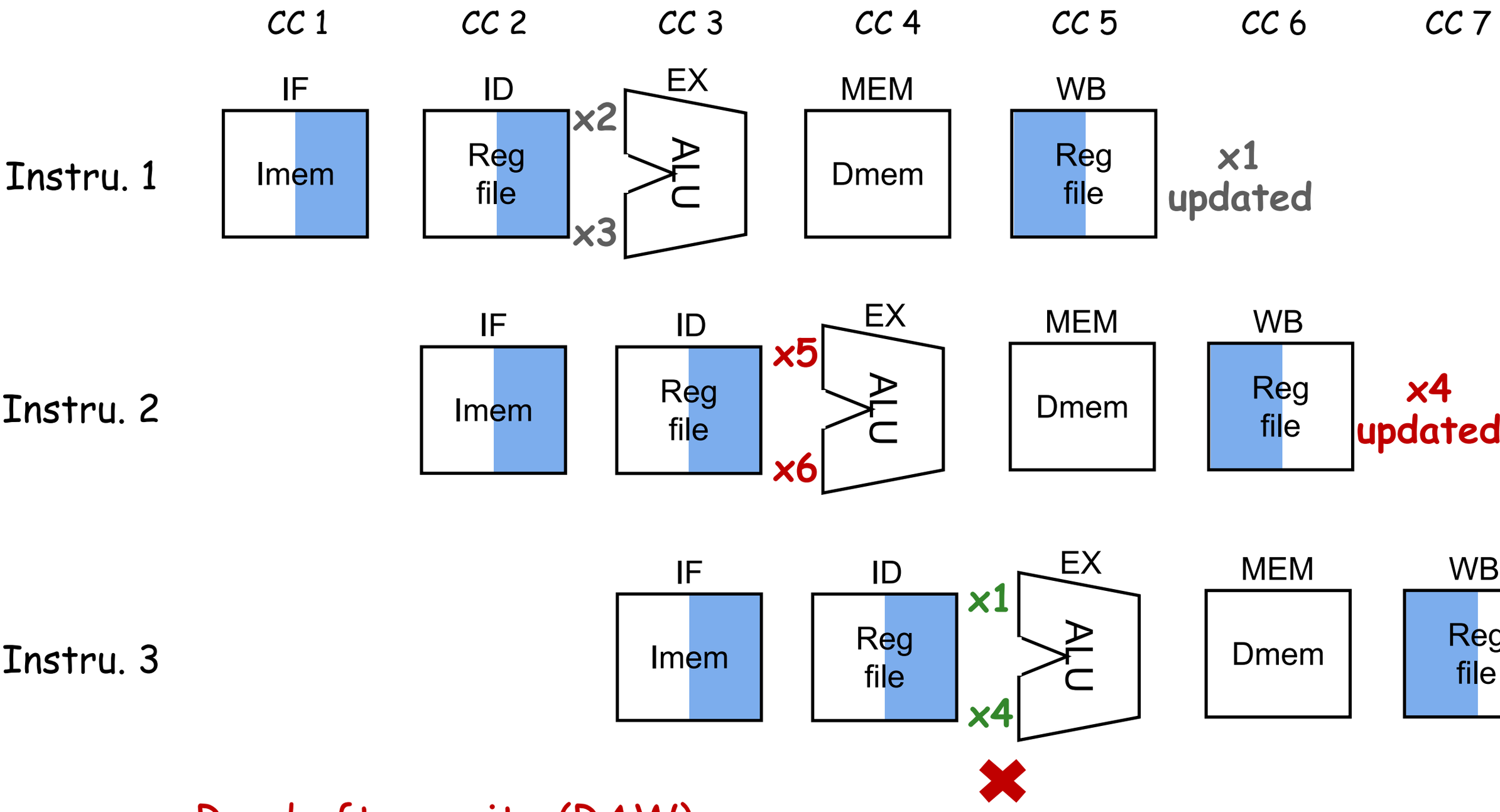
Hazards ahead!!!

- Structural hazards
- **Data hazards**
- Control hazards

Data hazards

```
add x1, x2, x3
add x4, x5, x6
add x7, x1, x4
```

R-type

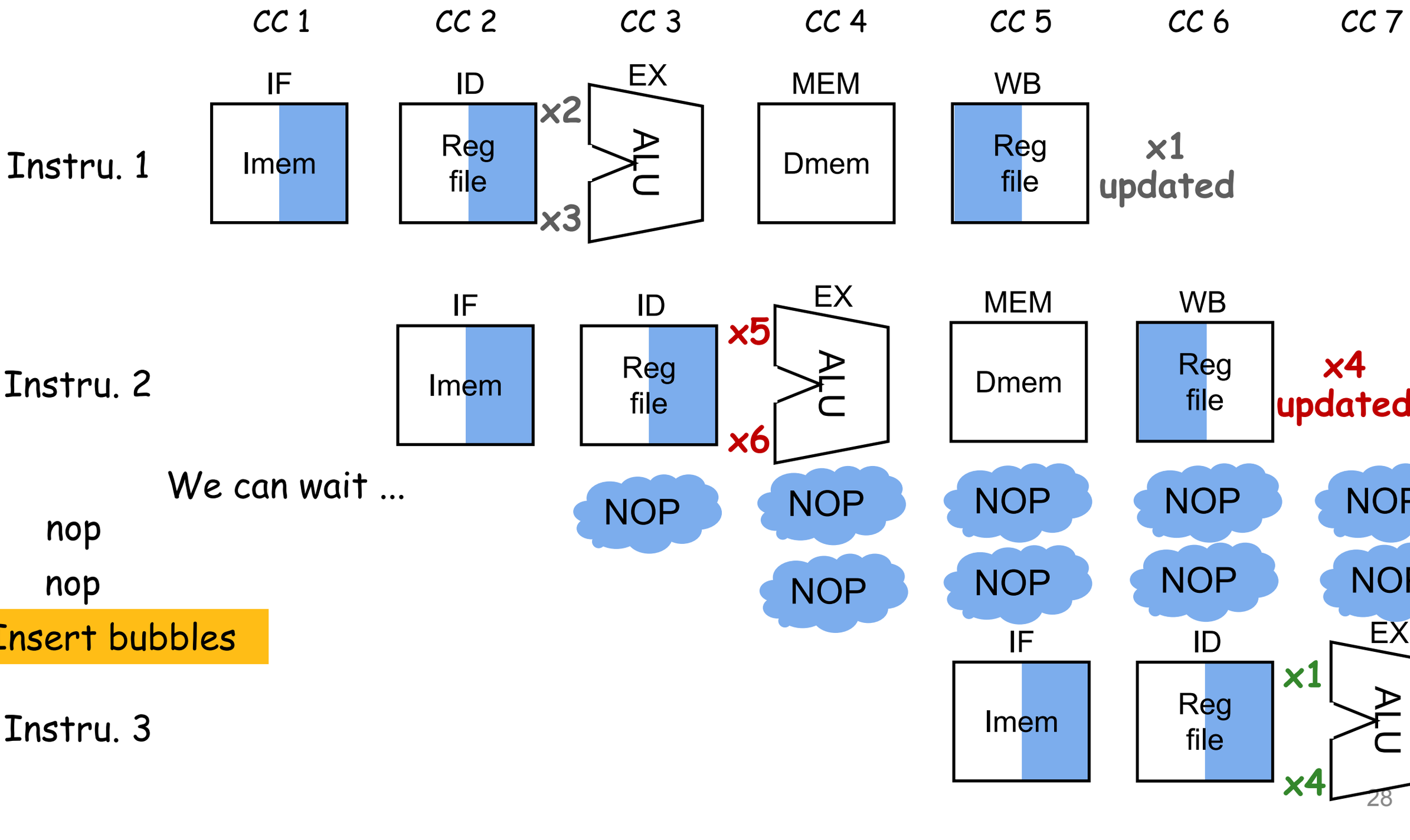


Read after write (RAW)

Data hazards -- solution 1

```
add x1, x2, x3
add x4, x5, x6
add x7, x1, x4
```

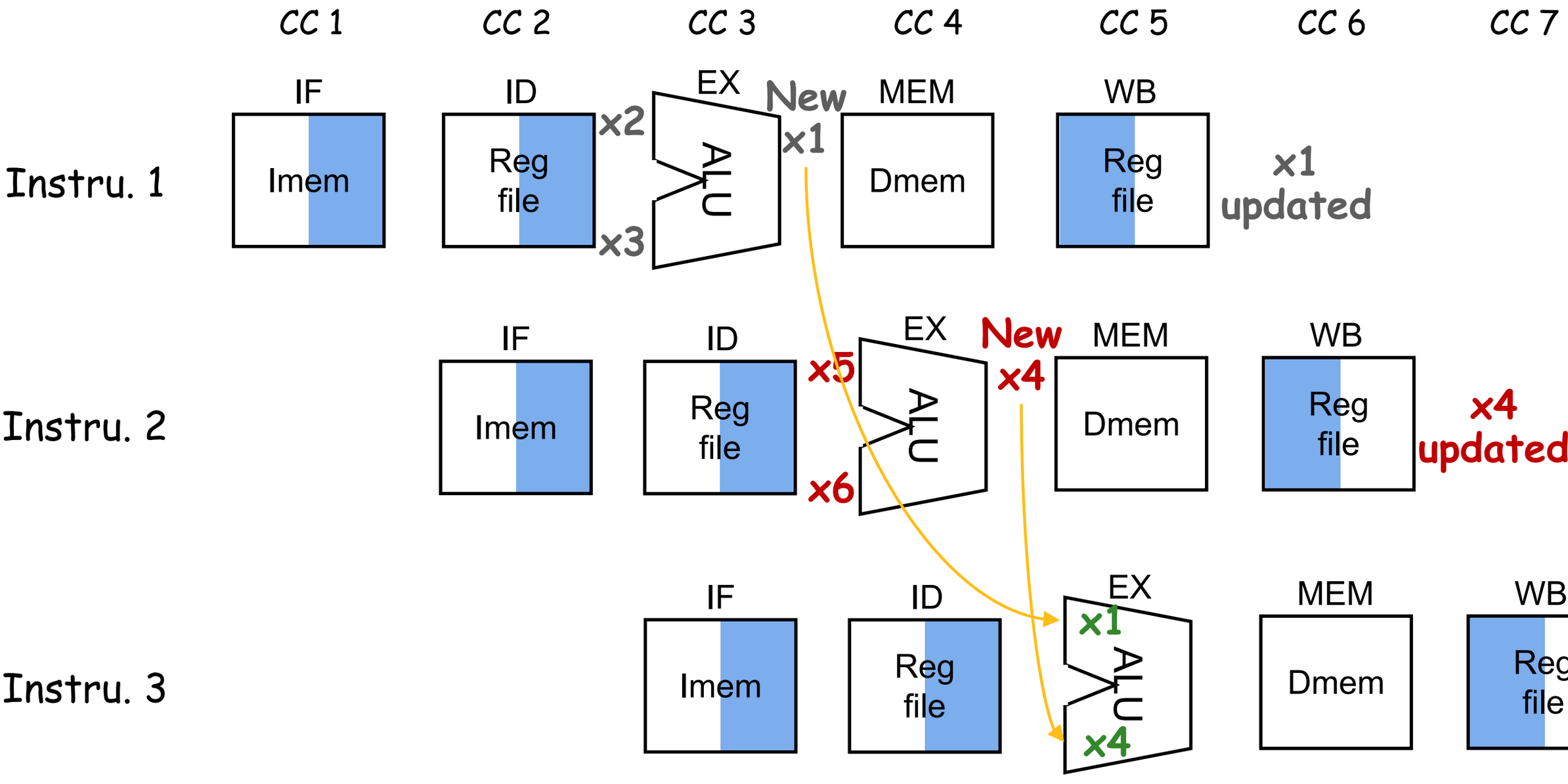
R-type



Data hazards -- solution 2

```
add x1, x2, x3
add x4, x5, x6
add x7, x1, x4
```

R-type

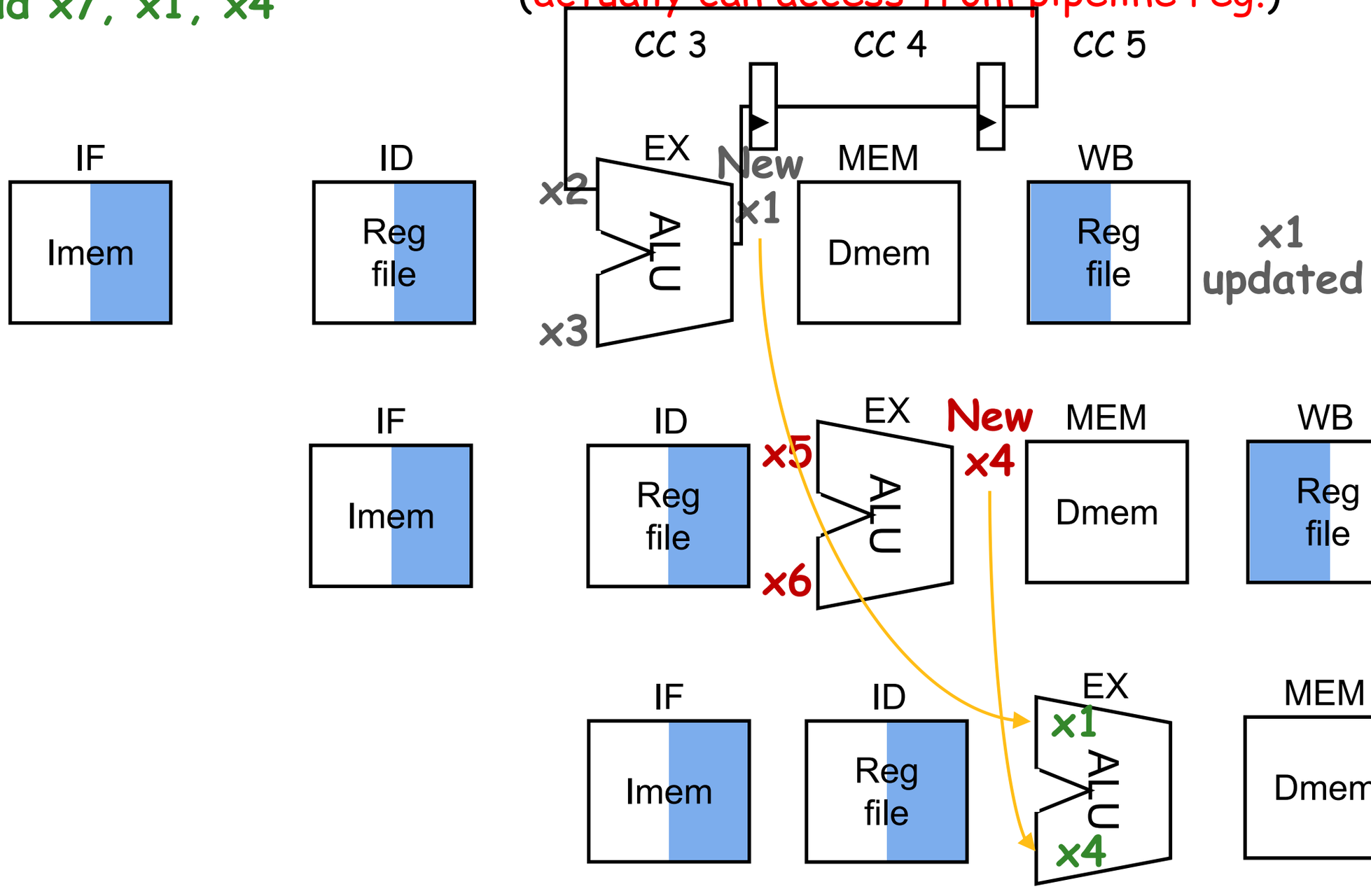


Forwarding or bypass

Data hazards -- solution 2

```
add x1, x2, x3
add x4, x5, x6
add x7, x1, x4
```

Add registers to store the updated values
(actually can access from pipeline reg.)

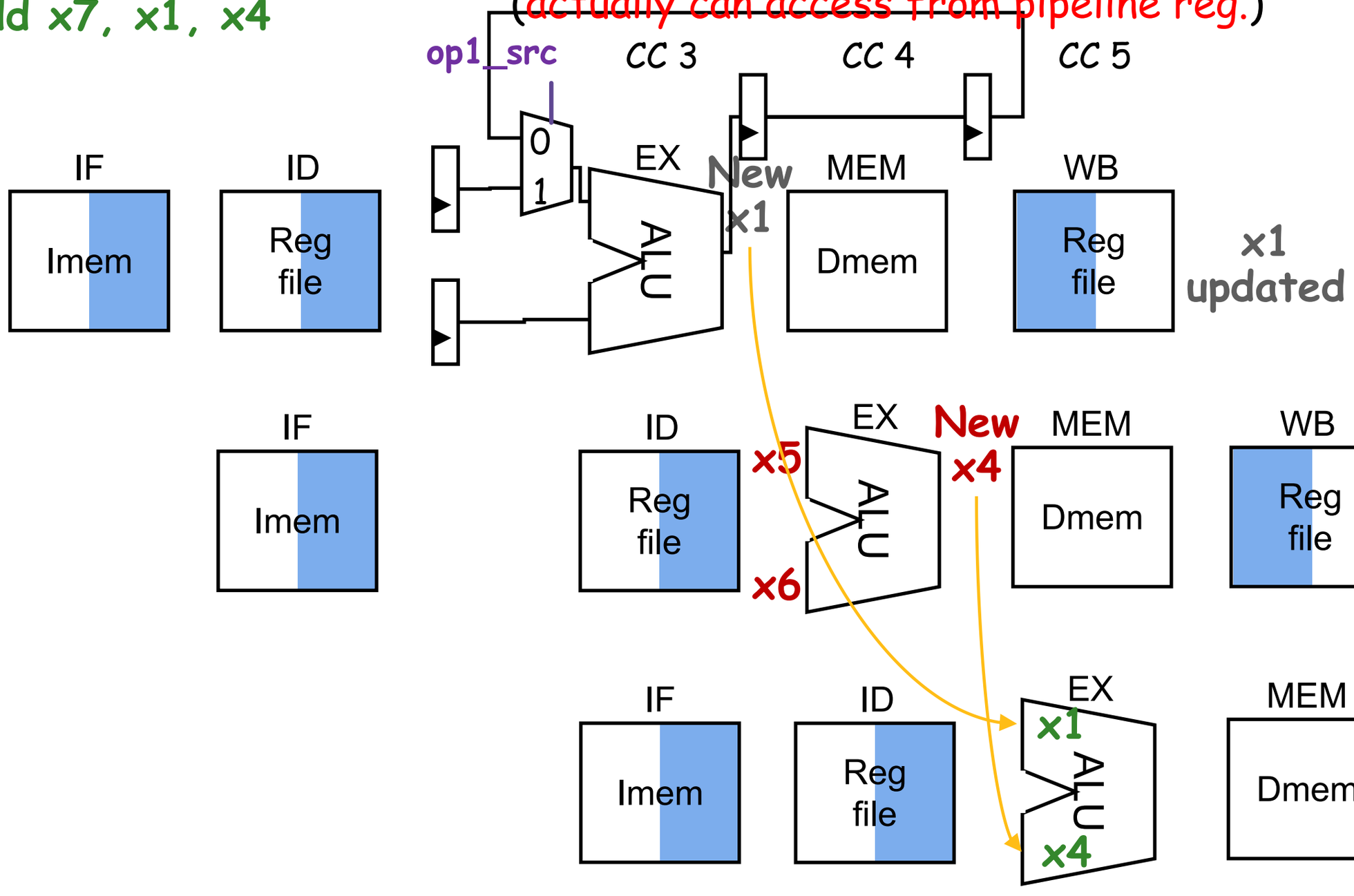


Forwarding or bypass

Data hazards -- solution 2

```
add x1, x2, x3
add x4, x5, x6
add x7, x1, x4
```

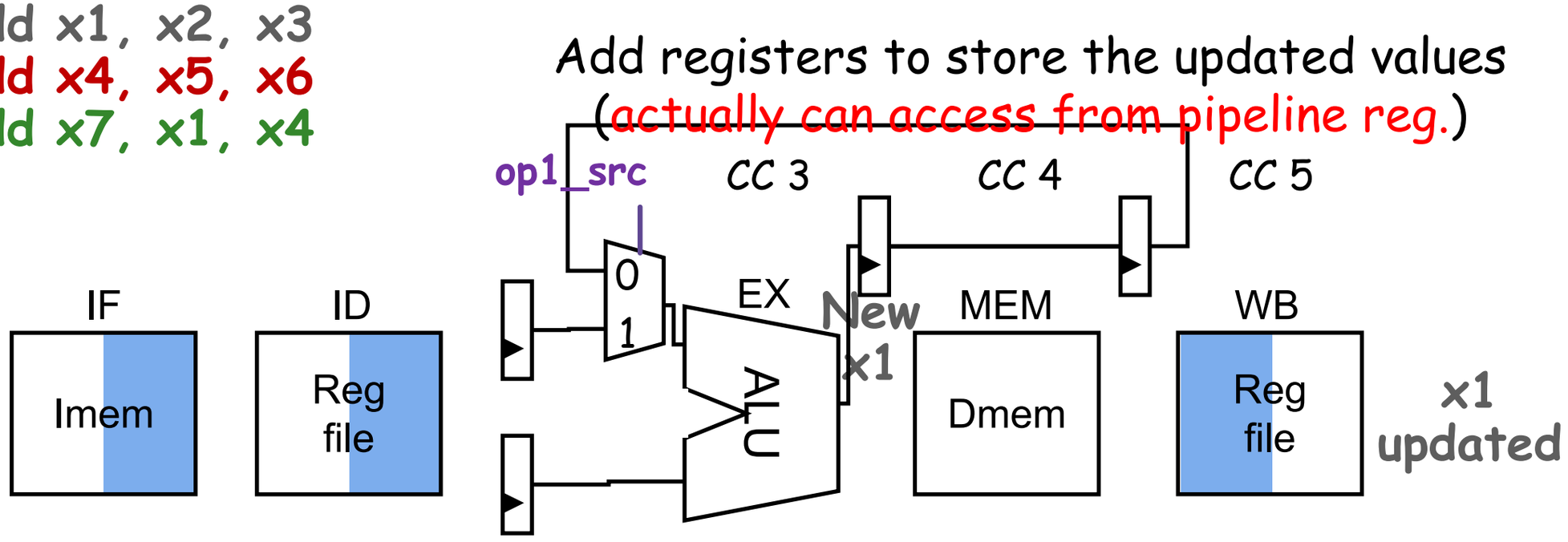
Add registers to store the updated values
(actually can access from pipeline reg.)



Forwarding or bypass

Data hazards -- solution 2

```
add x1, x2, x3
add x4, x5, x6
add x7, x1, x4
```



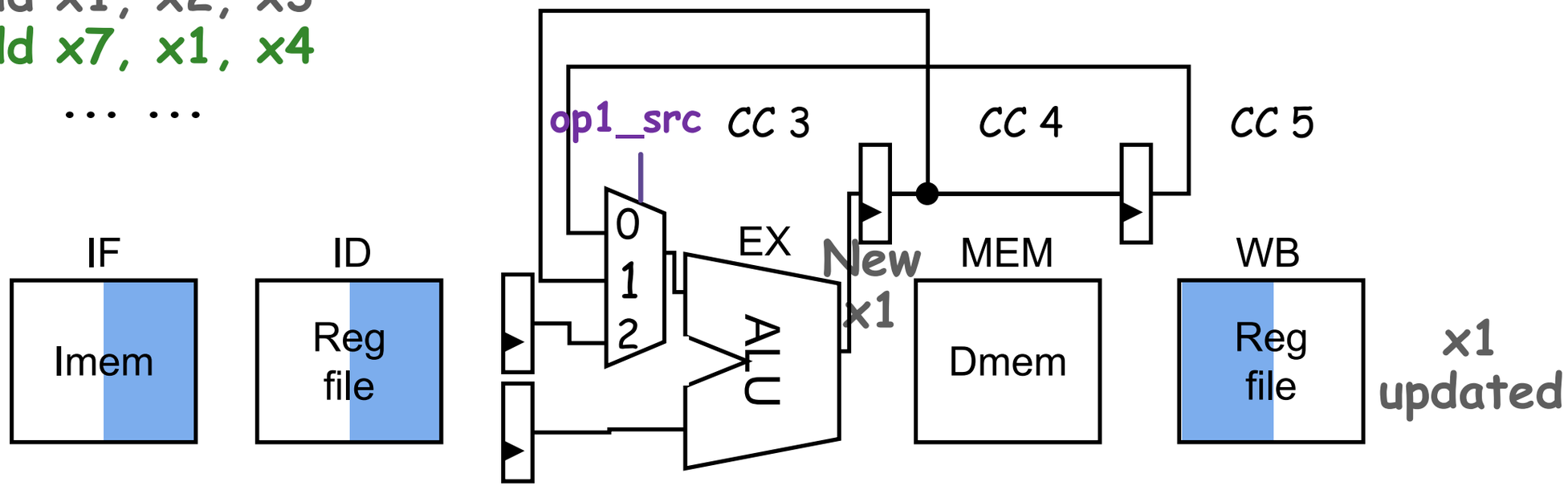
- How to decide **op1_src**?
  - Select the forwarded value or the value from ID/EX register
  - 1. rd of the add instruction (may be the other type instructions) equals rs1 in add (may be the other type instructions) instruction
  - 2. Ignore write to x0
  - 3. The first instruction must write the register and the third instruction must read the register

Forwarding or bypass

Forwarding control logic

Data hazards -- solution 2

```
add x1, x2, x3
add x7, x1, x4
... ...
```



- How to decide **op1_src**?
  - Select the forwarded value or the value from EX/MEM register
  - 1. rd of the add instruction (may be the other type instructions) equals rs1 in add (may be the other type instructions) instruction
  - 2. Ignore write to x0
  - 3. The first instruction must write the register and the second instruction must read the register

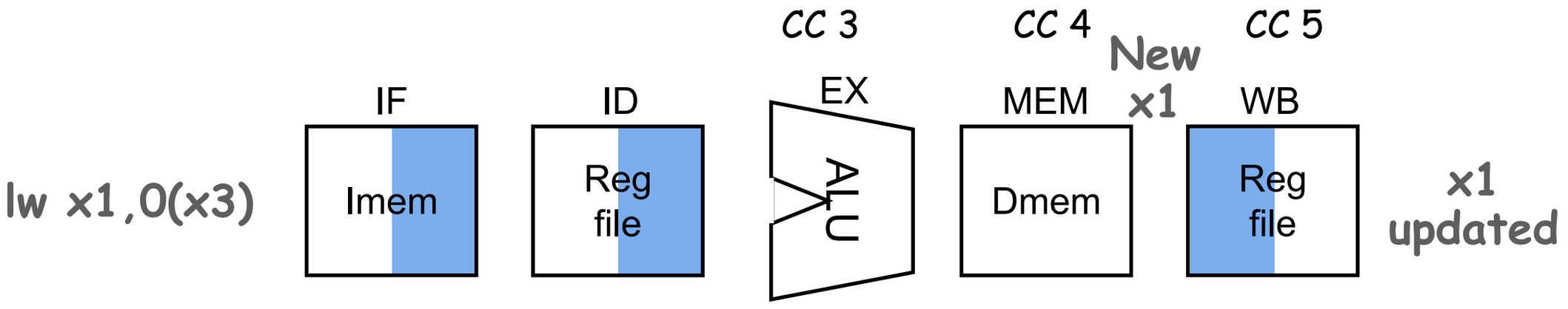
Forwarding or bypass

Forwarding control logic

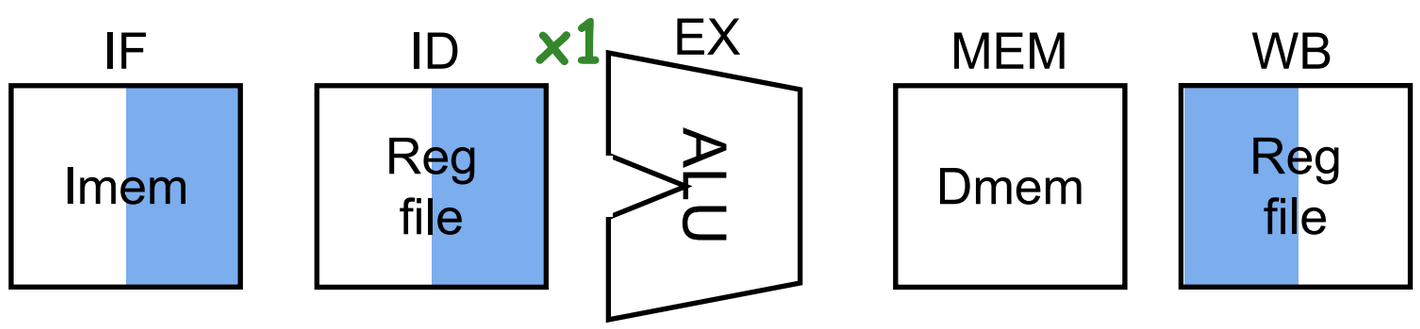
Data hazards

lw x1,0(x3)
add x7,x1,x4
... ...

Load type RAW



add x7,x1,x4

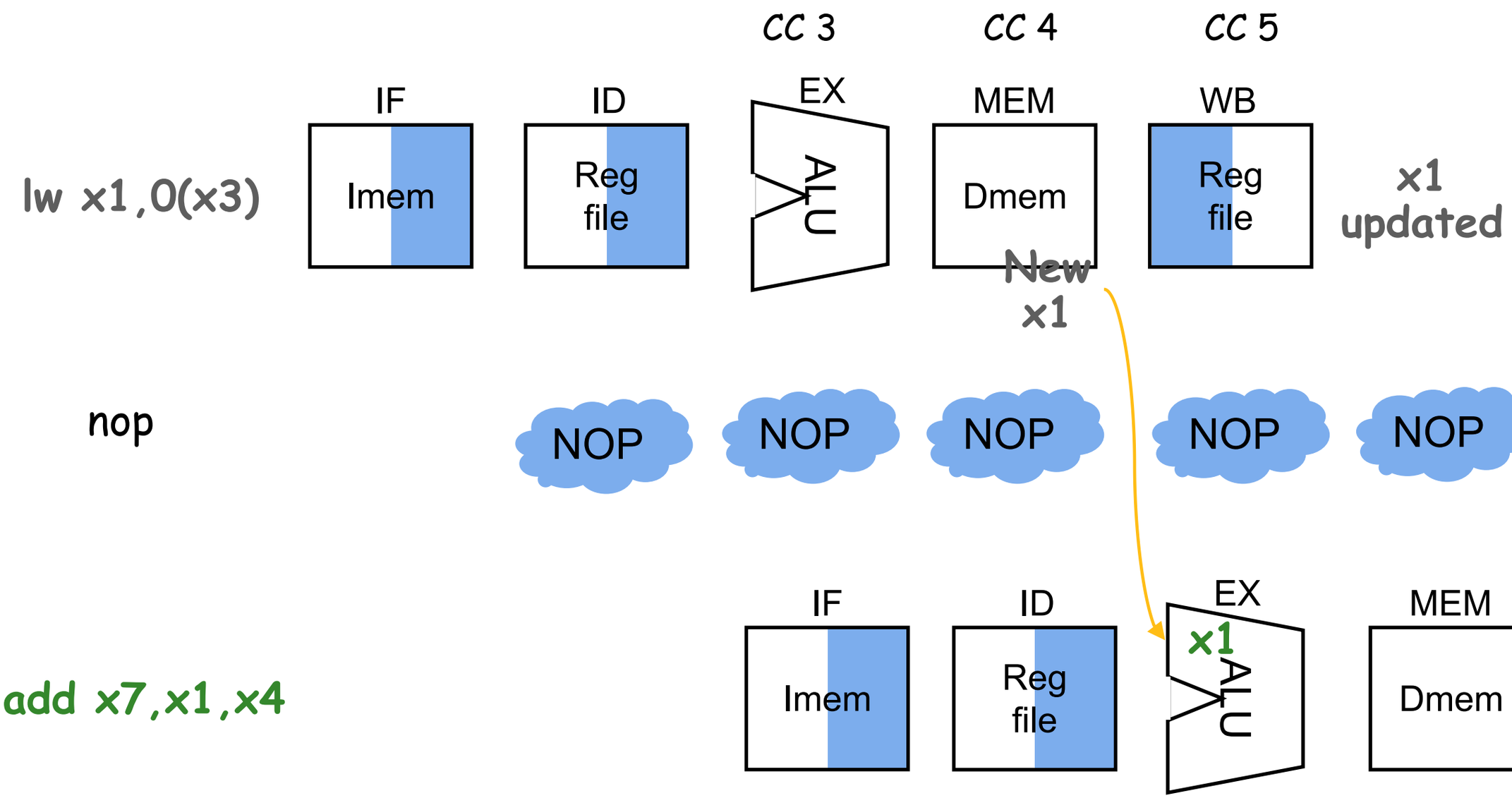


Forwarding cannot solve this, leading to a "load delay slot"

Data hazards -- solution 1

lw x1,0(x3)
add x7,x1,x4
... ...

Load type RAW



Insert nop and forward x1

Data hazards -- solution 2 warm up

- Identify all the data hazards in the following code

Original Order:

```
lw  t1, 0(t0)
lw  t2, 4(t0)
add t3, t1, t2
sw  t3, 12(t0)
lw  t4, 8(t0)
add t5, t1, t4
sw  t5, 16(t0)
```

Data hazards -- solution 2 warm up

- Identify all the data hazards in the following code

Original Order:

```
lw  t1, 0(t0)
lw  t2, 4(t0)
add t3, t1, t2
sw  t3, 12(t0)
lw  t4, 8(t0)
add t5, t1, t4
sw  t5, 16(t0)
```

Data hazards -- solution 2 warm up

- Identify all the data hazards in the following code

Original Order:

```
lw  t1, 0(t0)
lw  t2, 4(t0)
add t3, t1, t2
sw  t3, 12(t0)
lw  t4, 8(t0)
add t5, t1, t4
sw  t5, 16(t0)
```

Data hazards -- solution 2 warm up

- Identify all the data hazards in the following code

Original Order:

```
lw  t1, 0(t0)
lw  t2, 4(t0)
add t3, t1, t2
sw  t3, 12(t0)
lw  t4, 8(t0)
add t5, t1, t4
sw  t5, 16(t0)
```

Data hazards -- solution 2 warm up

- Identify all the data hazards in the following code

Original Order:

```
lw  t1, 0(t0)
lw  t2, 4(t0)
add t3, t1, t2
sw  t3, 12(t0)
lw  t4, 8(t0)
add t5, t1, t4
sw  t5, 16(t0)
```

Data hazards -- solution 2 warm up

- Identify all the data hazards in the following code
- Which of the hazards cannot be completely solved by forwarding?

Original Order:

```
lw  t1, 0(t0)
lw  t2, 4(t0)
add t3, t1, t2
sw  t3, 12(t0)
lw  t4, 8(t0)
add t5, t1, t4
sw  t5, 16(t0)
```

Data hazards -- solution 2 warm up

- Identify all the data hazards in the following code
- Which of the hazards cannot be completely solved by forwarding?

Original Order:

```
lw  t1, 0(t0)
lw  t2, 4(t0)
add t3, t1, t2
sw  t3, 12(t0)
lw  t4, 8(t0)
add t5, t1, t4
sw  t5, 16(t0)
```

Data hazards -- solution 2

- Code scheduling: Put unrelated instruction into load delay slot
  - No performance loss!

Original Order:

```
lw  t1, 0(t0)
lw  t2, 4(t0)
add t3, t1, t2
sw  t3, 12(t0)
lw  t4, 8(t0)
add t5, t1, t4
sw  t5, 16(t0)
```

Code scheduling:

```
lw  t1, 0(t0)
lw  t2, 4(t0)
lw  t4, 8(t0)
add t3, t1, t2
sw  t3, 12(t0)
add t5, t1, t4
sw  t5, 16(t0)
```

- Code scheduling can be accomplished by the compiler

Data hazards

- How many clock cycles to complete the code before and after code scheduling? Assume no instruction in the pipeline initially.

Original Order:

```
lw  t1, 0(t0)
lw  t2, 4(t0)
add t3, t1, t2
sw  t3, 12(t0)
lw  t4, 8(t0)
add t5, t1, t4
sw  t5, 16(t0)
```

Code scheduling:

```
lw  t1, 0(t0)
lw  t2, 4(t0)
lw  t4, 8(t0)
add t3, t1, t2
sw  t3, 12(t0)
add t5, t1, t4
sw  t5, 16(t0)
```

Summary on data hazards

- Instructions have data dependency
- Occurs when an instruction reads a register before a previous instruction has finished writing to that register (RAW)
- There can also be WAW/WAR hazards depending on the pipeline design
- For load-type RAW data hazards, there is a load delay slot unavoidable
- Can be solved by forwarding or code scheduling

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

Question

IF = 400 ps	ID = 200 ps	EX = 200 ps	MEM = 500 ps	WB = 200 ps

What is the maximum frequency of this 5-stage RISC-V CPU before/after pipelining?