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-2024/index.html
School of Information Science and Technology (SIST)
ShanghaiTech University

2024/4/9

Administratives

- No Lab this week, instead, we check Project 1.1 this week at the lab sessions.
- Again, all the deadlines are hard deadlines! Start early!
- HW 4 ddl April 29th
- Proj1.2 ddl April 25th
- Proj2.1 is released with deadline of May 7th. (near the second mid-term, so start early)
- Discussion (teaching center 301) schedule
    - Datapath for proj 2.1 has completed
    - Next discussion is about pipeline
    - The same content for Friday and the next Monday.

Policy on Assignments and Independent Work

- ALL LABS/HOMEWORKS/ASSIGNMENTS, AND SOME PROJECTS WILL BE DONE INDEPENDENTLY
- SOME PROJECTS WILL BE DONE WITH YOUR PARTNER (WITHIN THE SAME LAB SESSION)
- With the exception of laboratories and assignments that explicitly permit you to work in groups, all homework are to be YOUR work and your work ALONE.
- PARTNER TEAMS MAY NOT WORK WITH OTHER PARTNER TEAMS
- You can discuss your assignments with other students, and credit will be assigned to students who help others by answering questions on Piazza (participation), but we expect that what you hand in is yours.
- Level of detail allowed to discuss with other students: Concepts (Material taught in the class/in the text book)! Pseudocode or code is NOT allowed!
- Use the Office Hours of the TA and the Profs. if you need help with your homework/project!
- Rather submit an incomplete homework with maybe 0 points than risking an F!
- It is NOT acceptable to copy solutions from other sources (other students/web/AI-generated).
- You can never look at homework/ project code not by you/ your team!
- You cannot give your code to anybody else –> secure your computer when not around it
- It is NOT acceptable to copy (or start your) solutions from the Web/other students/AI-generated.
- It is NOT acceptable to use PUBLIC github archives (giving your answers away)
- It is NOT acceptable to give anyone other than your project partner access to your gitlab!
- Protect your code and password! Do not allow other students peeping at your screen.
- We have tools and methods, developed over many years, for detecting this. You WILL be caught, and the penalties WILL be severe.
- At the **minimum F** in the course, and **a letter to your university record** documenting the incidence of cheating.
- Both Giver and Receiver are equally culpable and suffer equal penalties

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 100 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

Different if given IPC

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

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

“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

Different if given IPC

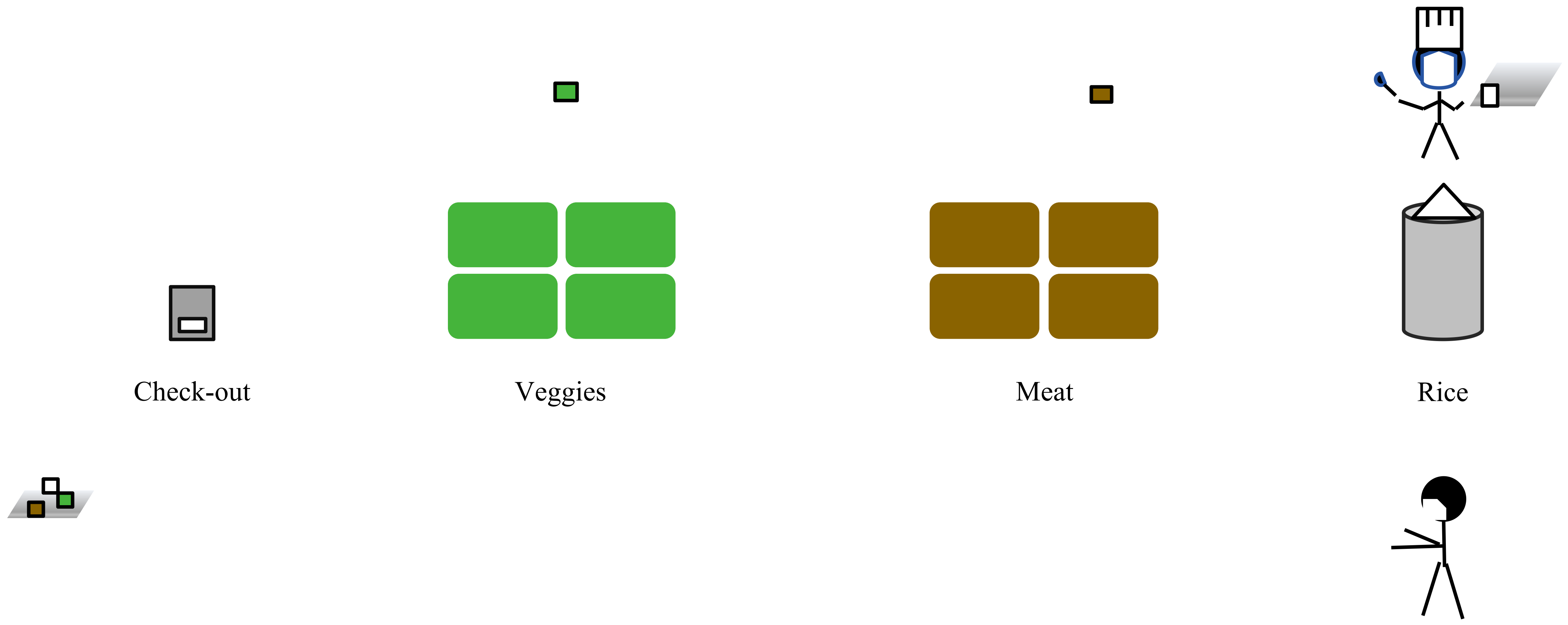
- **Calculate CPU (execution) time**
  - CPU frequency 2.5 GHz
  - Program A has in total 10^6 instructions

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

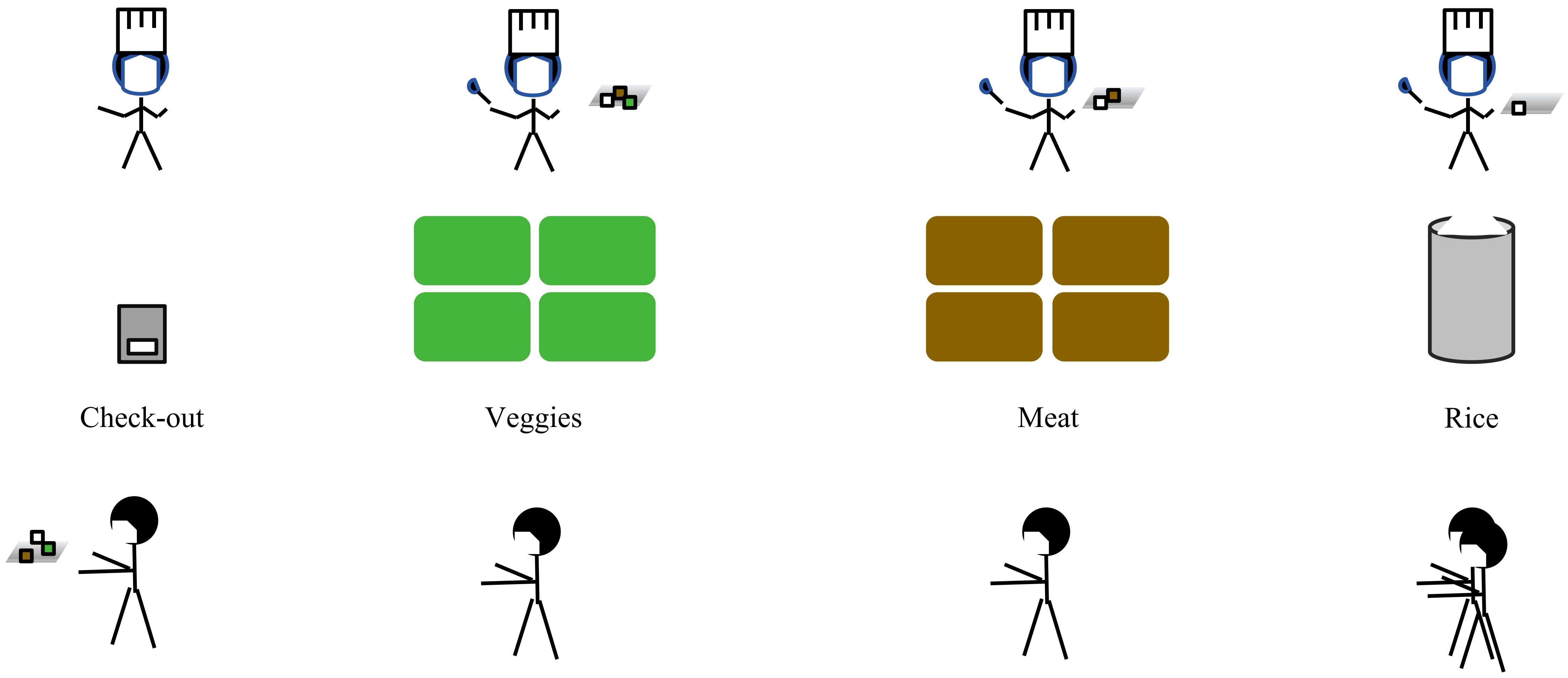
Performance

- 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)
  - Performance measurement & improvement
  - Dependability via redundancy

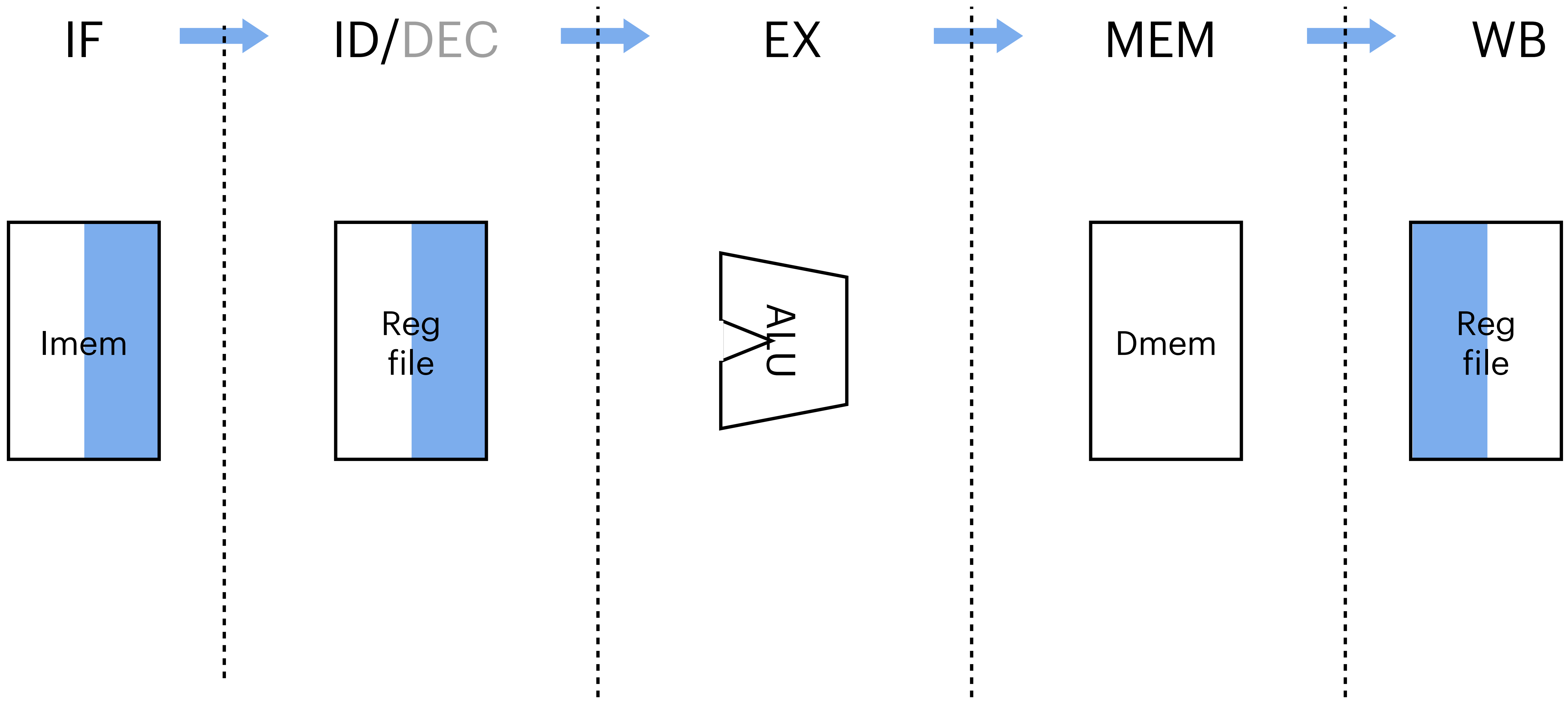
Pipe that line!



Pipe that line!



Simplify the CPU with 5 stages



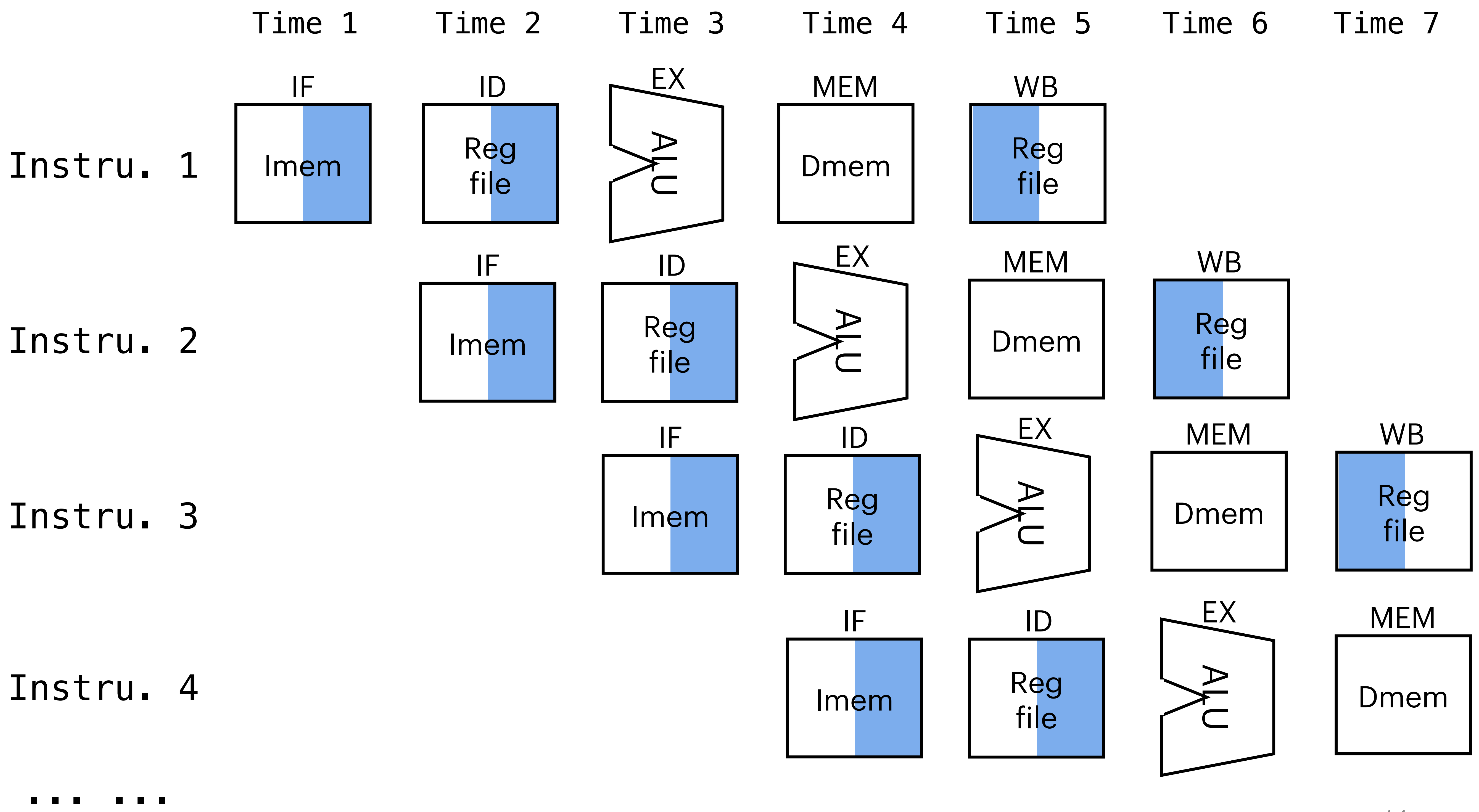
Instruction

Operands
Control signals

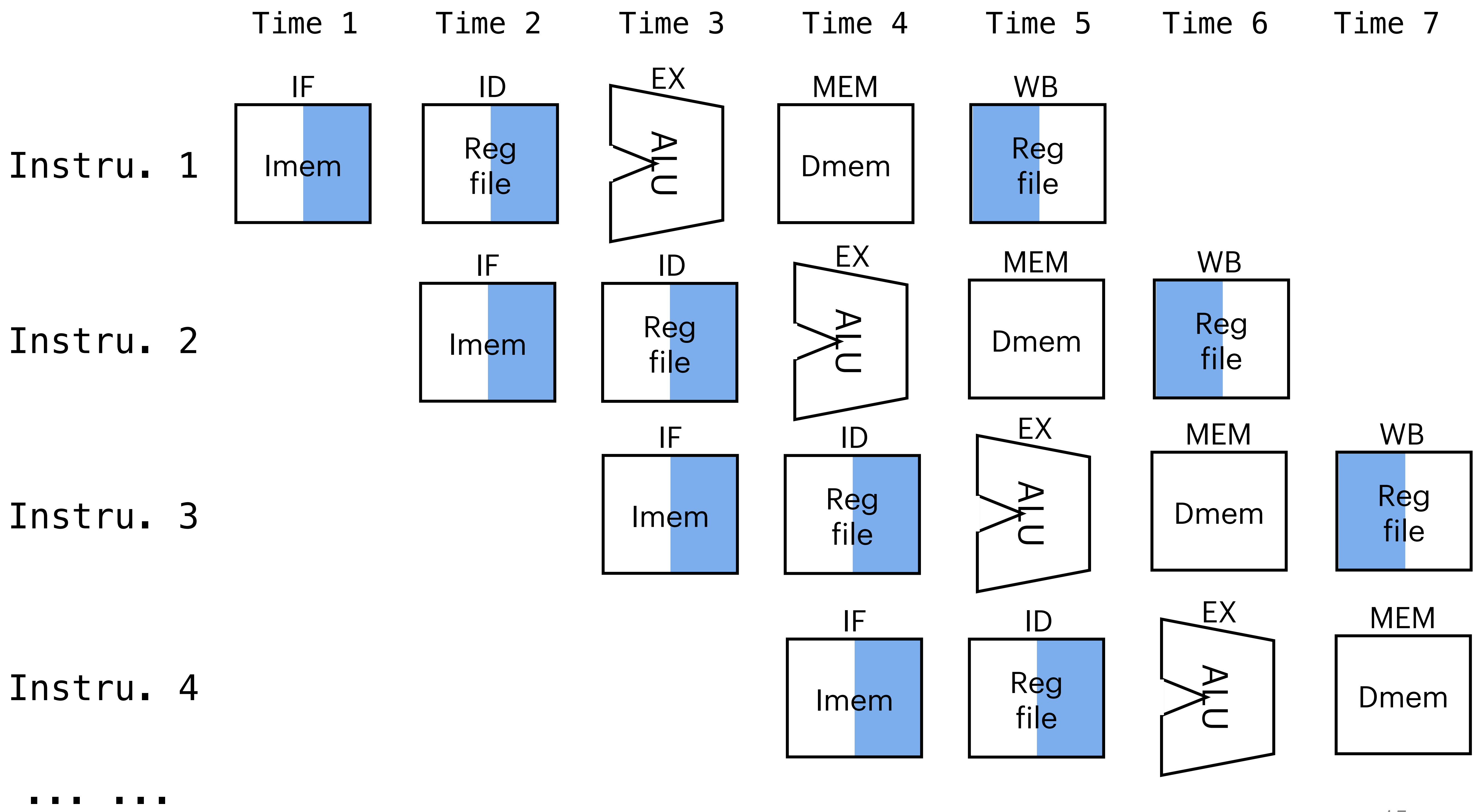
Result
(address)

Write/read Dmem
if required
Memory data

Make an analogy



Make an analogy



Previously for a single-cycle CPU

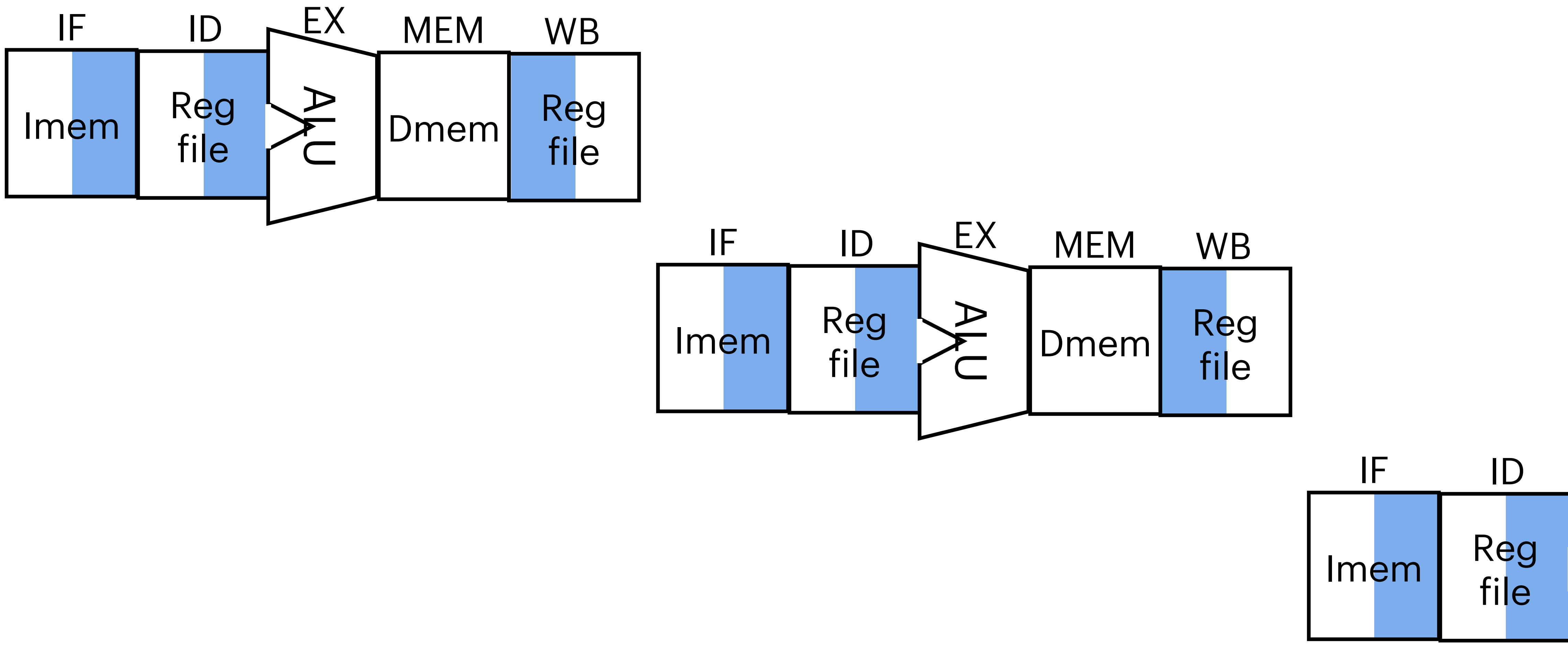
Time 1 Time 2

Instru. 1

Instru. 2

Instru. 3

... ...



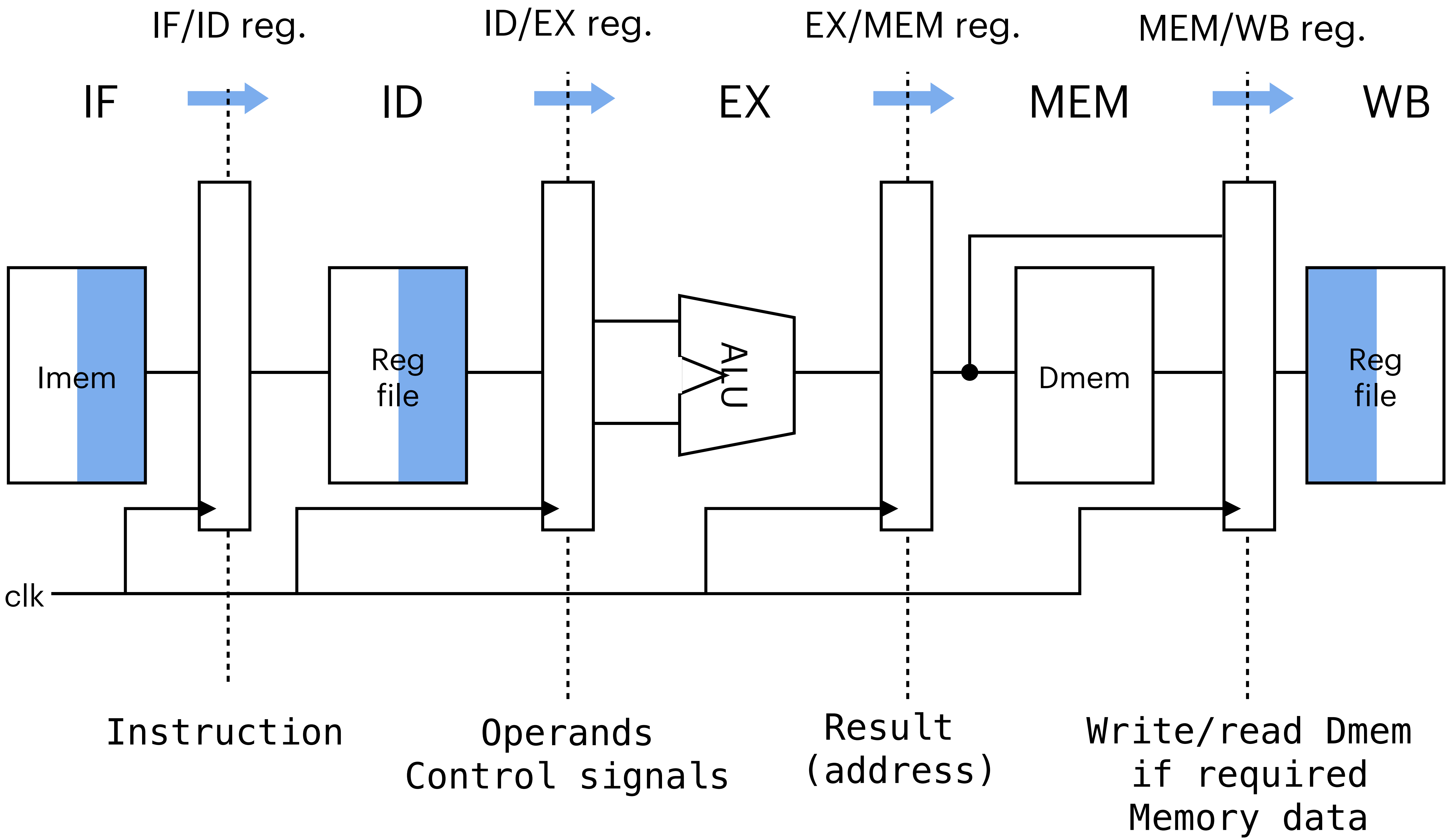
Observations

- 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?

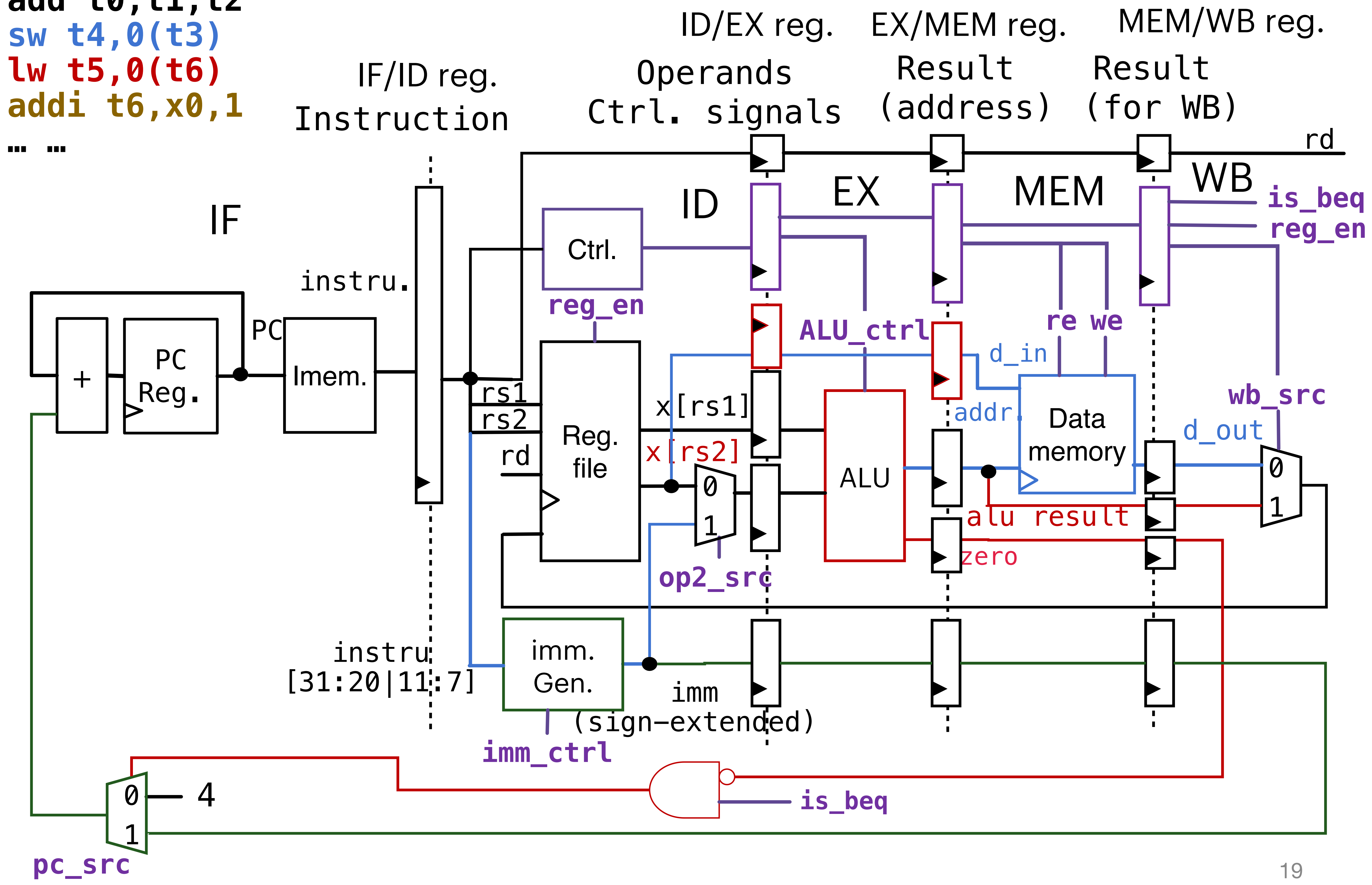
Insert pipeline registers



Critical path decided by the slowest stage

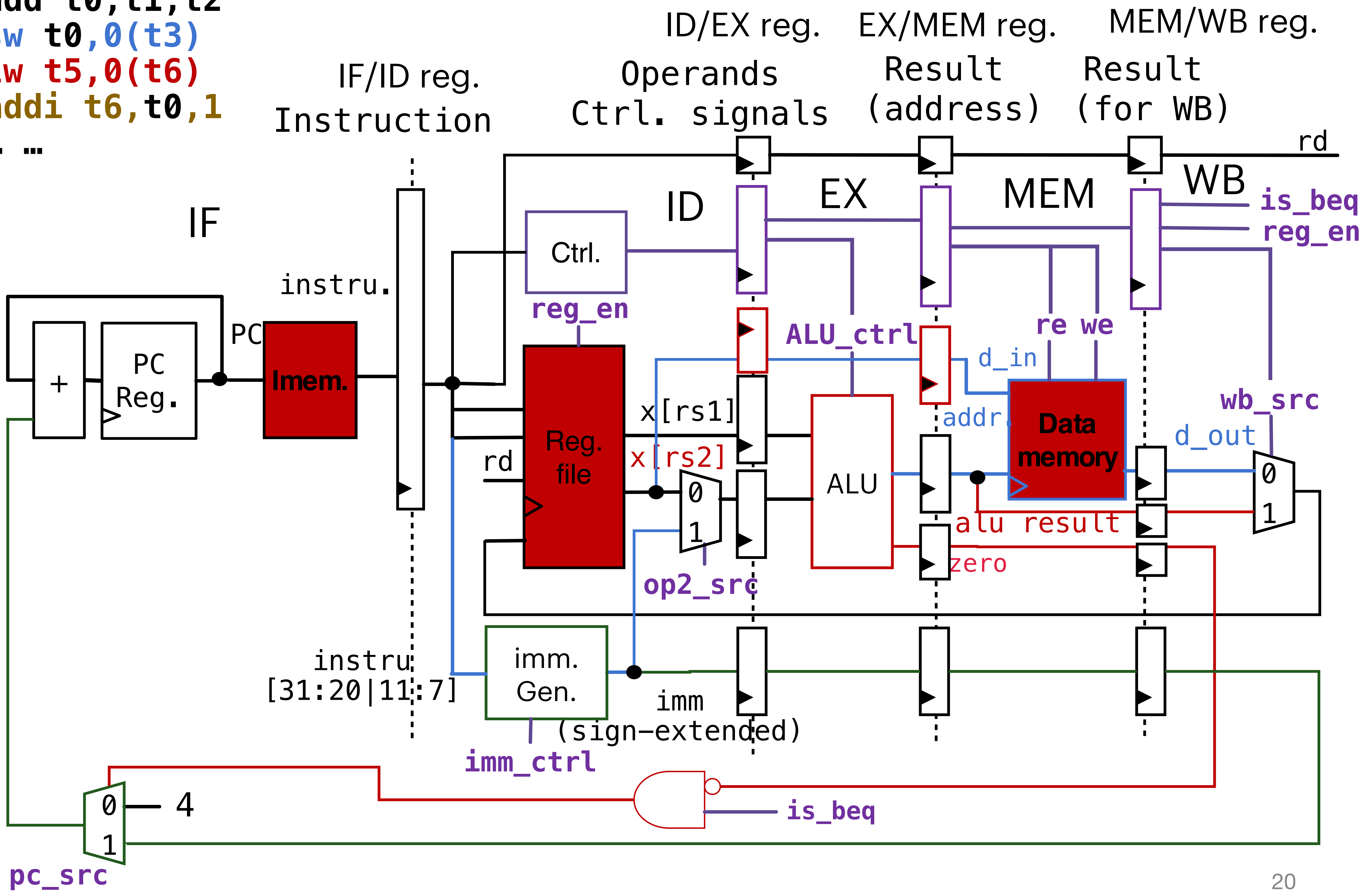
Detailed considerations

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



Hazards ahead!!!

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

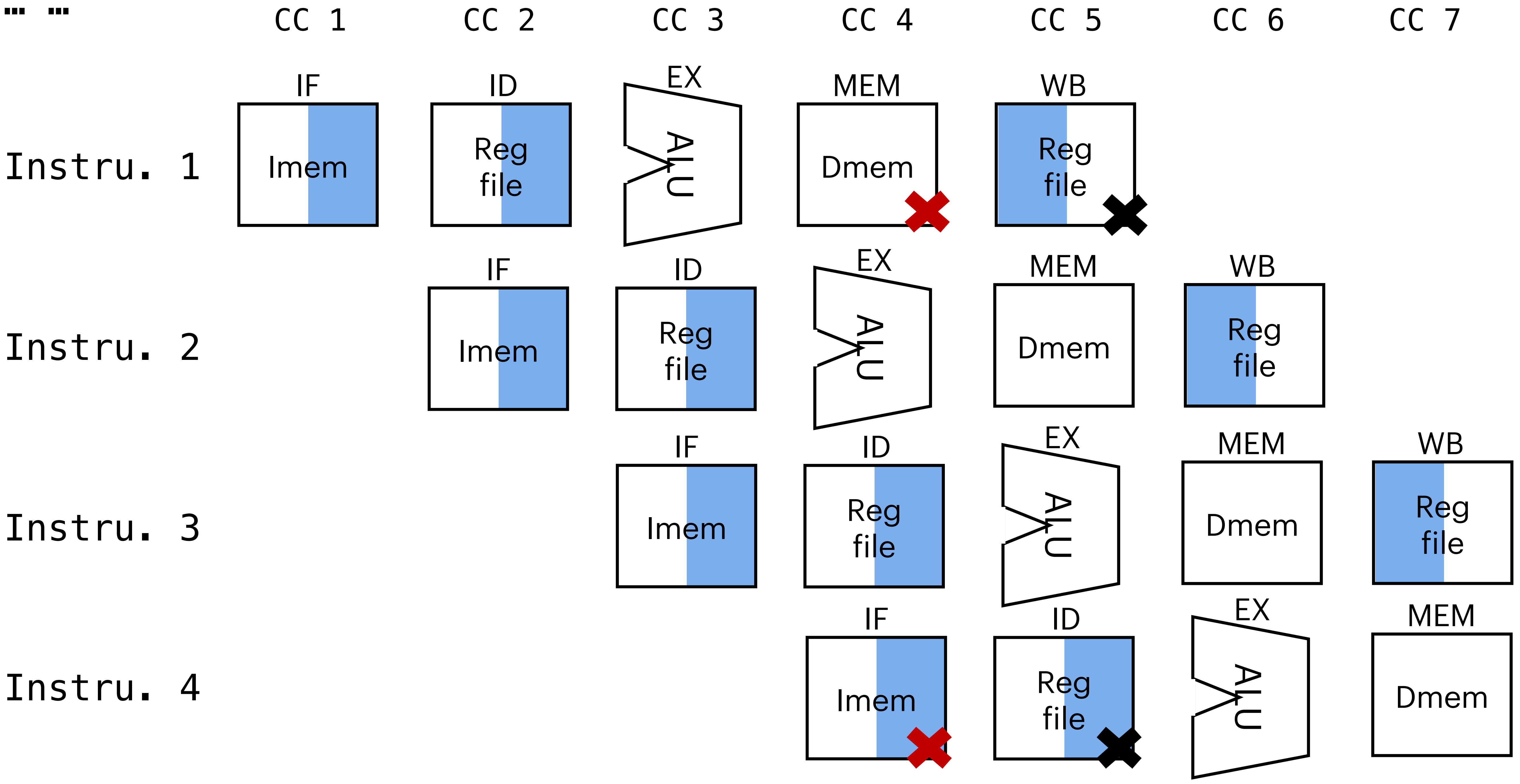


Hazards ahead!!!

- **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)

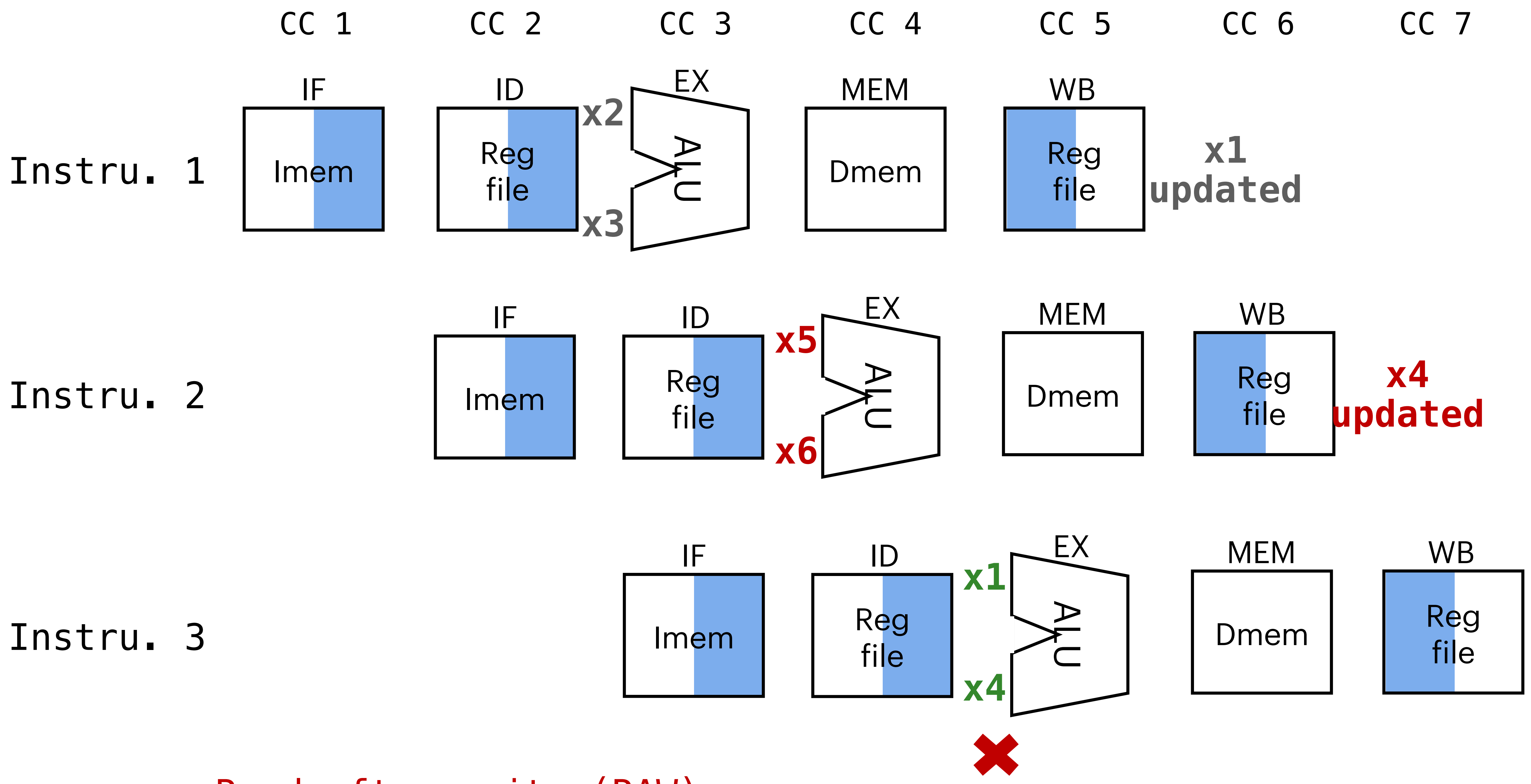
Hazards ahead!!!

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

Data hazards

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

R-type

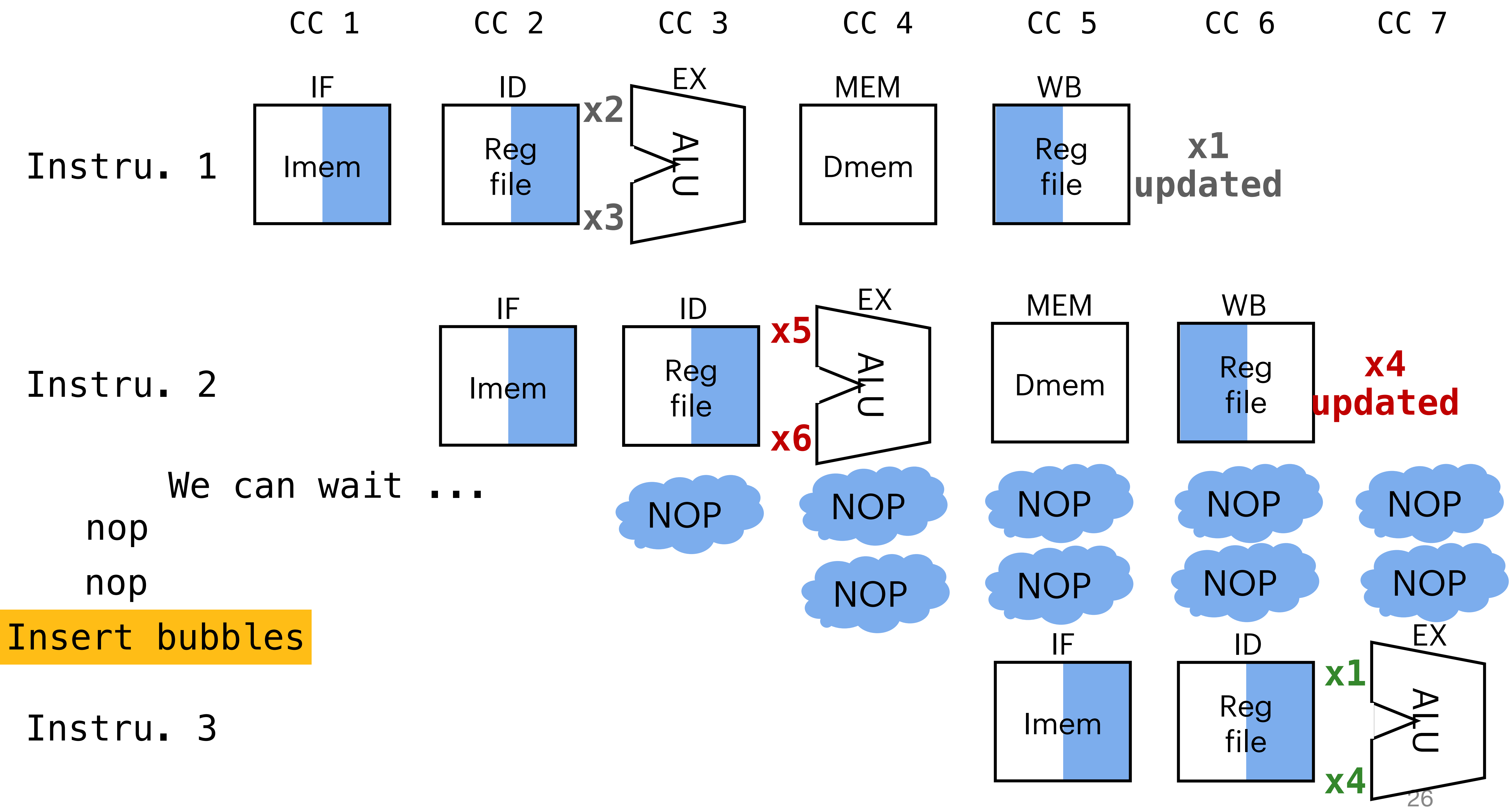


Read after write (RAW)

Data hazards -- solution 1

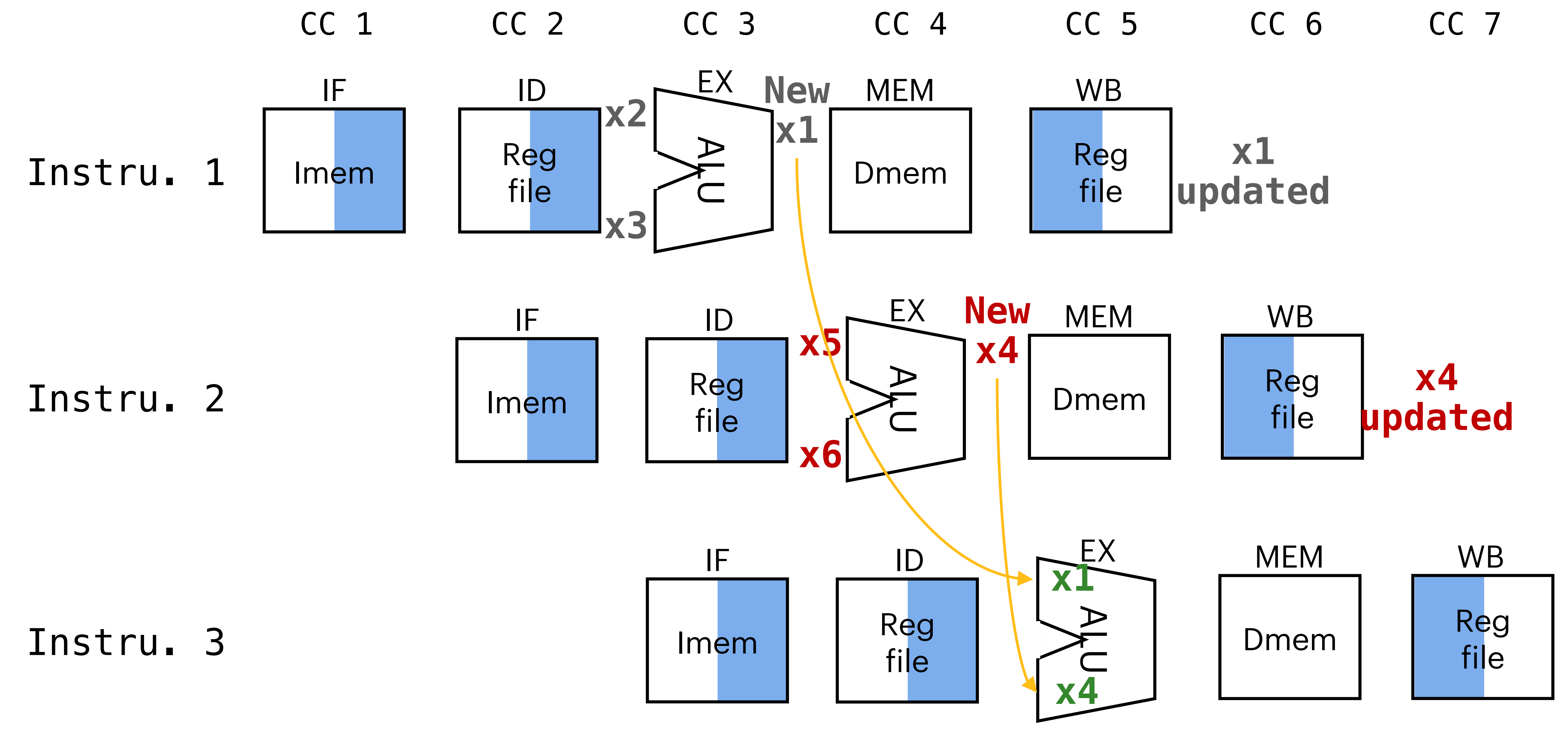
```
add x1, x2, x3
add x4, x5, x6
add x7, x1, x4
```

R-type



Data hazards -- solution 2

```
add x1, x2, x3
add x4, x5, x6
add x7, x1, x4      R–type
```

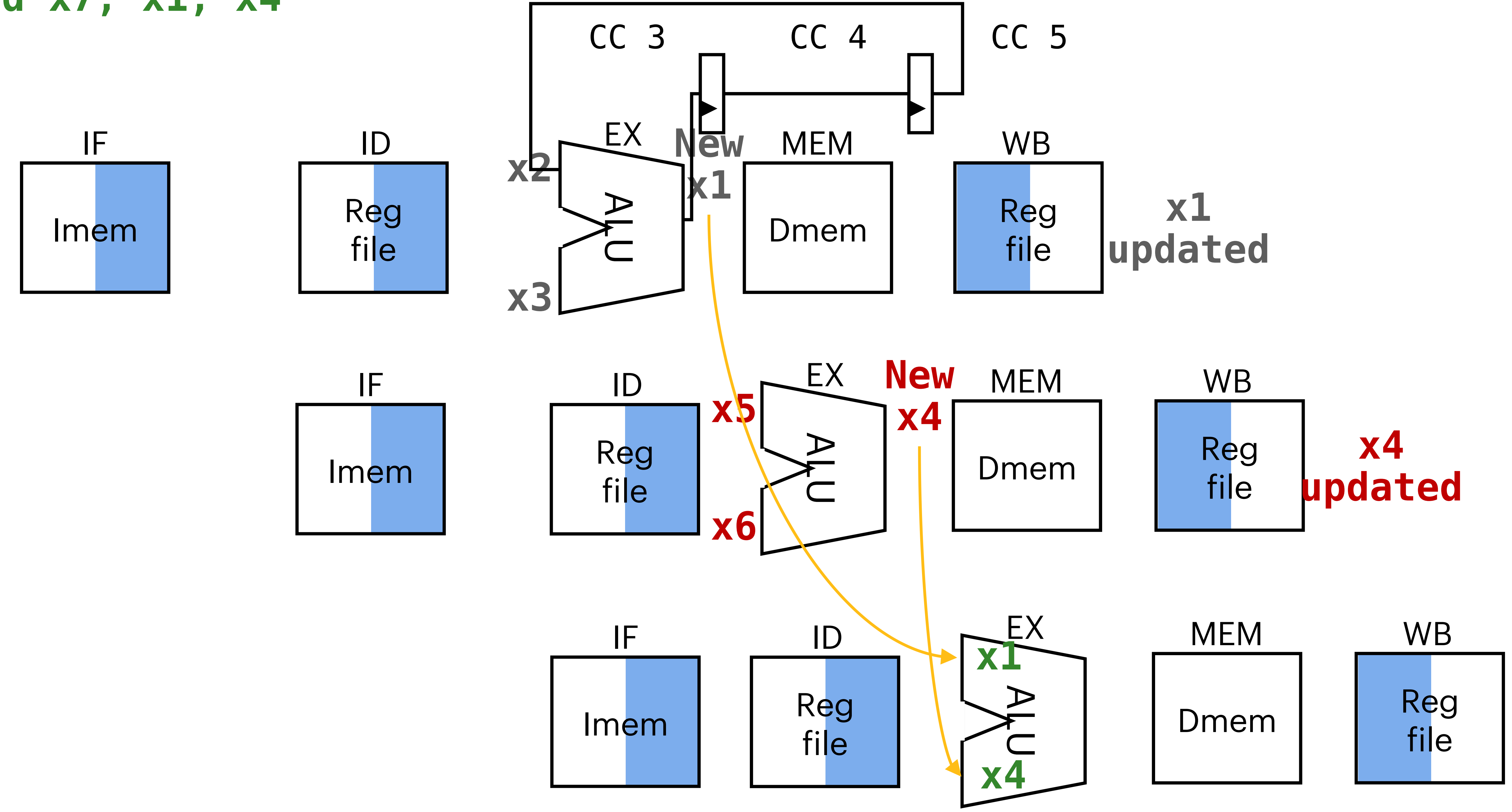


Forwarding or bypass

Data hazards -- solution 2

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

Add registers to store the updated values

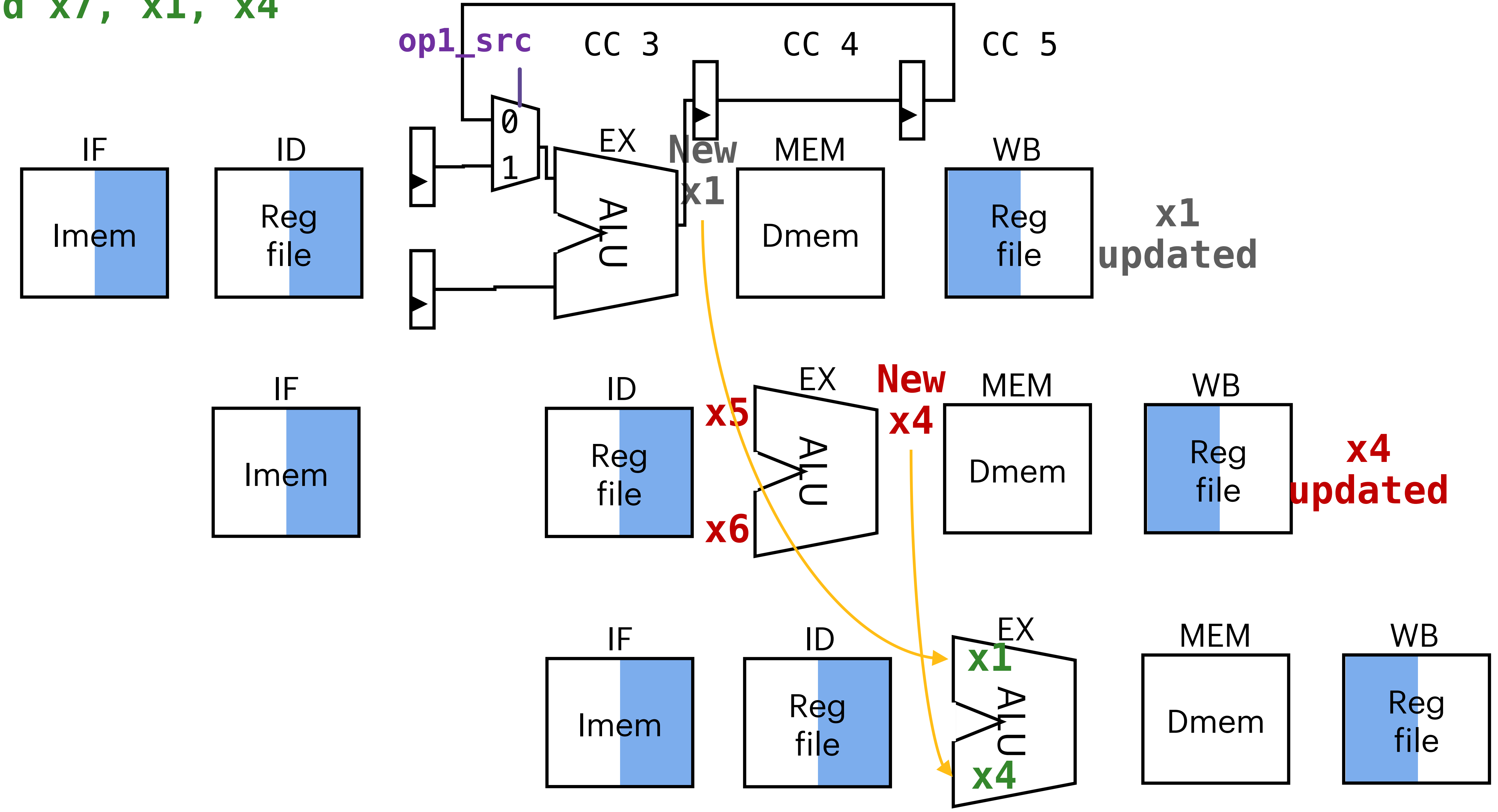


Forwarding or bypass

Data hazards -- solution 2

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

Add registers to store the updated values

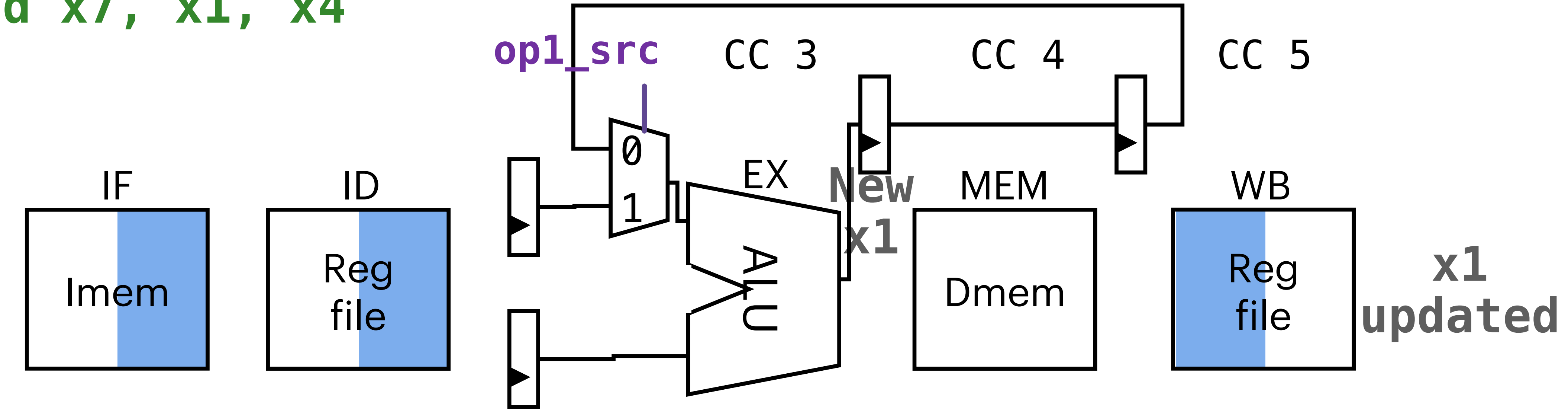


Forwarding or bypass

Data hazards -- solution 2

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

`Add registers to store the updated values`



- 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

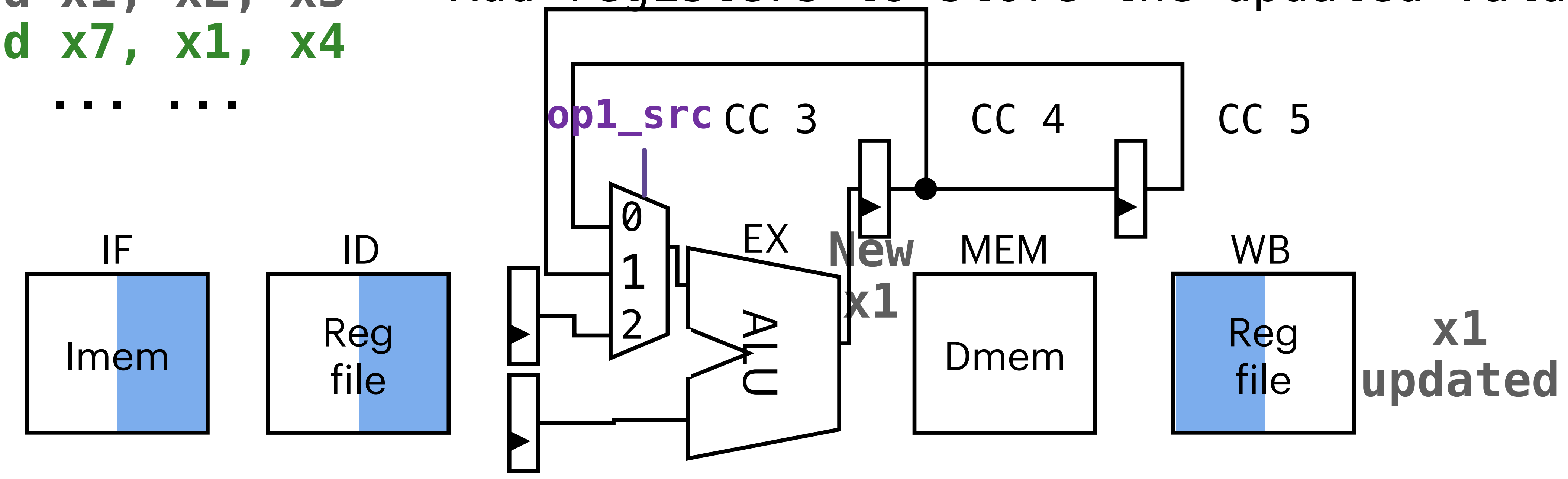
`Forwarding or bypass`

Forwarding control logic

Data hazards -- solution 2

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

Add registers to store the updated values



- 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

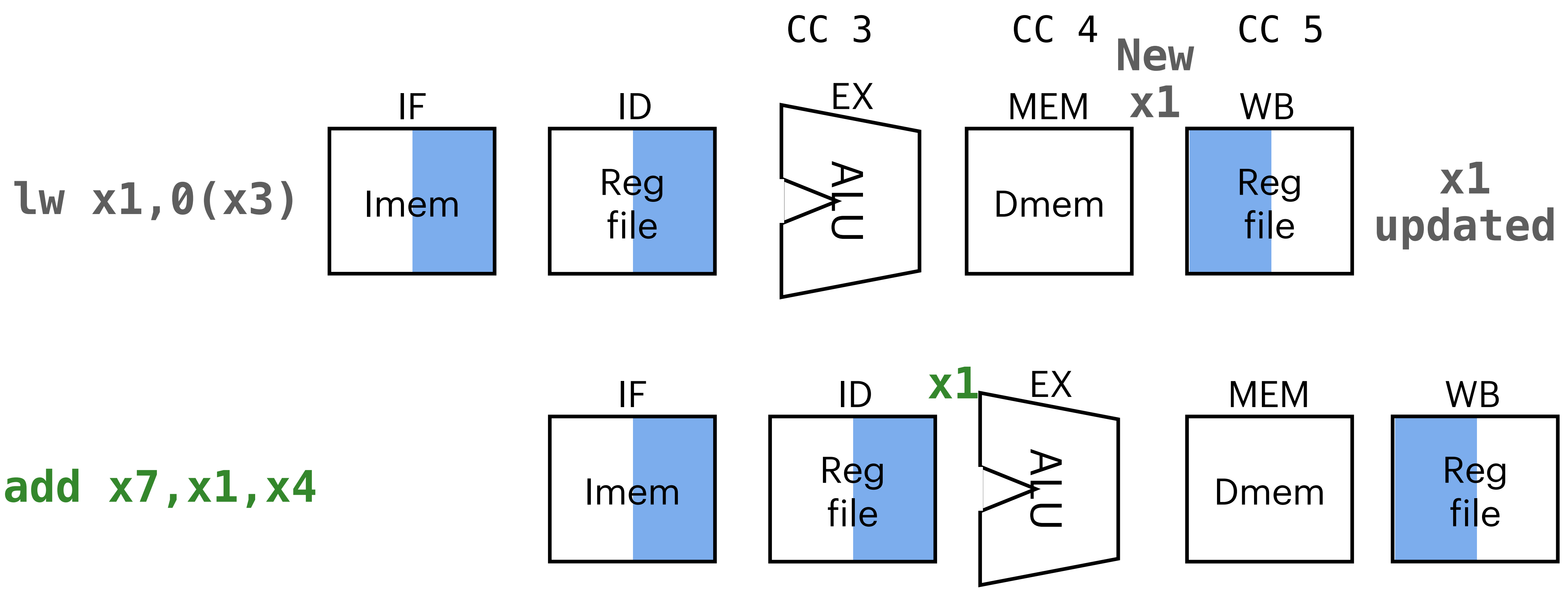
Forwarding control logic

`Forwarding or bypass`

Data hazards

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

Load type RAW

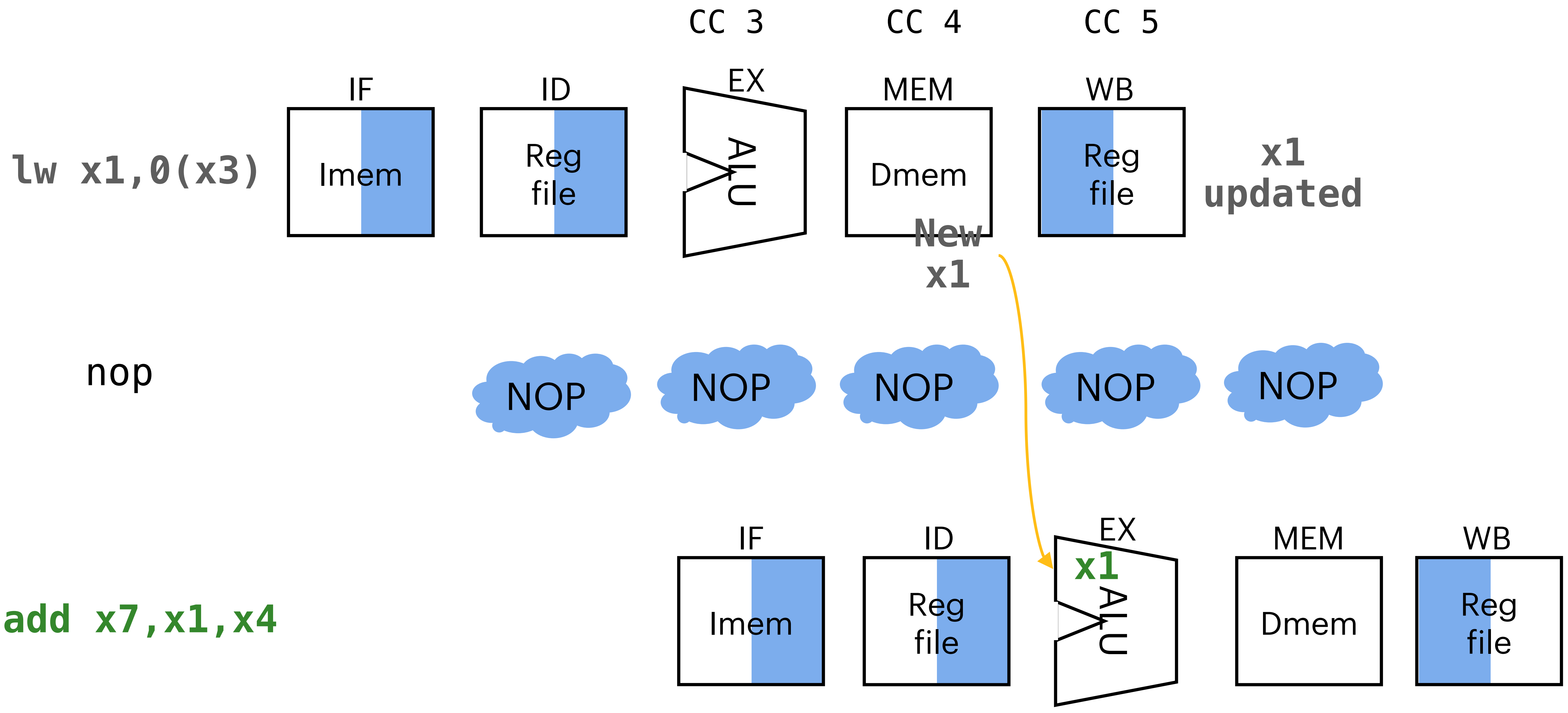


Forwarding cannot solve this, leading to a “load delay slot”

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 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