<u>Poll:</u> How was the midterm compared to practice exams?

EECS 370 - Lecture 13

Pipelining and Data Hazards

Announcements

- P2
  - P2L+P2R due next Thursday
- Lab 7
  - No prelab this week
  - Will just be providing help on P2R
  - **Attendance is required unless you've completed P2R by the time your lab meets**

Single vs Multi-cycle Performance

1 ns – Register File read/write time

2 ns – ALU/adder

2 ns – memory access

0 ns – MUX, PC access, sign extend, ROM

1. Assuming the above delays, what is the best cycle time that the LC2k multi-cycle datapath could achieve? Single cycle?

   MC: MAX(2, 1, 2, 2, 1) = 2ns

   SC: 2 + 1 + 2 + 2 + 1 = 8 ns

2. Assuming the above delays, for a program consisting of 25 LW, 10 SW, 45 ADD, and 20 BEQ, which is faster?

   SC: 100 cycles * 8 ns = 800 ns

   MC: (25*5 + 10*4 + 45*4 + 20*4)cycles * 2ns = 850 ns

Single and Multi-cycle performance

- Wait, multi-cycle is worse??
- For our ISA, most instructions take about the same time
- Multi-cycle shines when some instructions take much longer
- E.g. if we add a long latency instruction like multiply:
  - Let's say operation takes 10 ns, but could be split into 5 stages of 2 ns
  - SC: clock period = 16 ns, performance is 1600 ns
  - MC: clock period = 2 ns, performance is 850 ns

Performance Metrics – Execution time

- What we really care about in a program is **execution time**
  - **Execution time** = total instructions executed X CPI x clock period
  - The "Iron Law" of performance
- CPI = **average** number of clock **cycles per instruction** *for an application*
- To calculate multi-cycle CPI we need:
  - Cycles necessary for each type of instruction
  - Mix of instructions executed in the application (dynamic instruction execution profile)

<u>Poll:</u> What are the units of (instructions executed x CPI x clock period)?

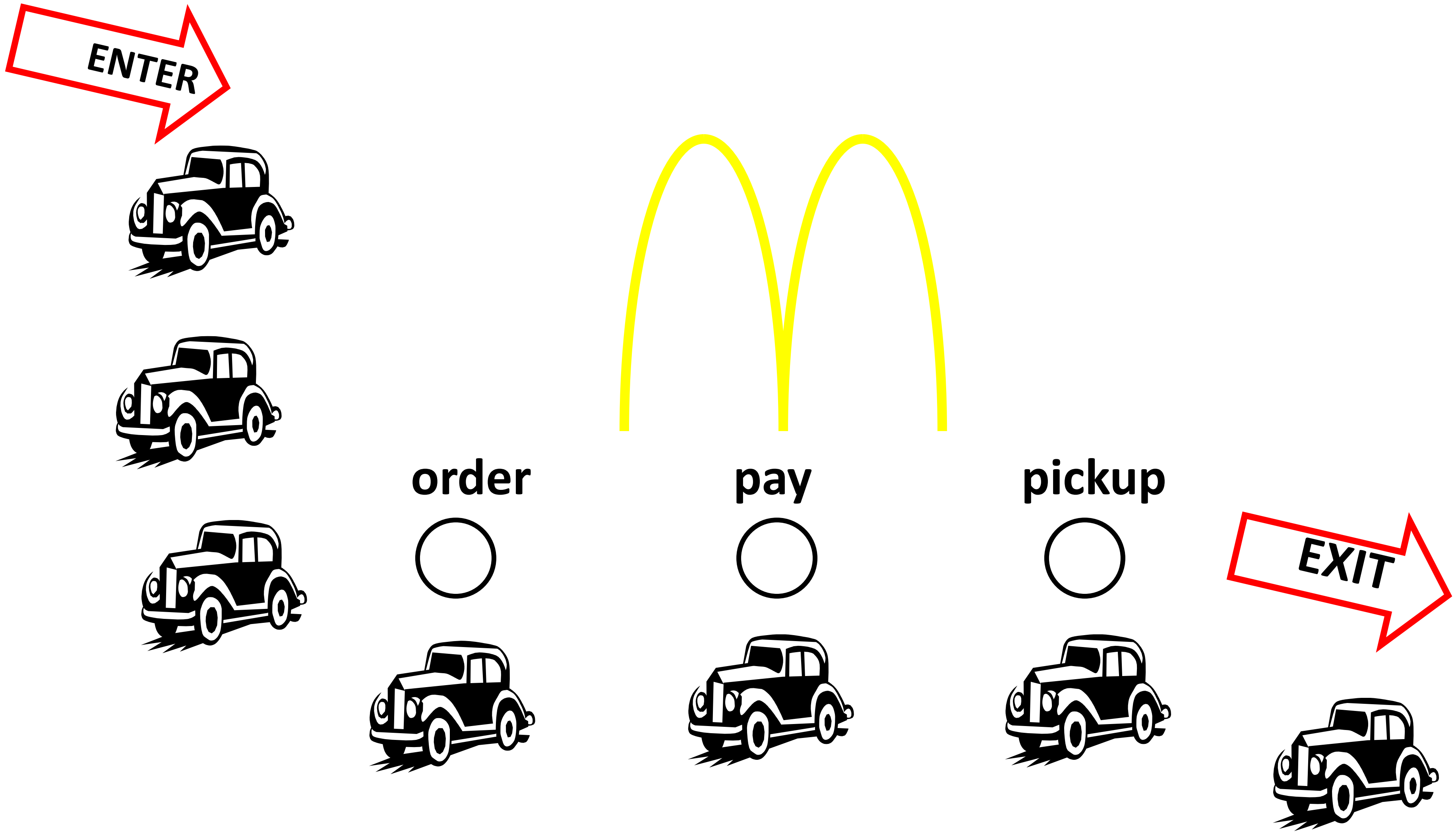
Datapath Summary

- Single-cycle processor
  - CPI = 1 (by definition)
  - clock period = ~10 ns
- Multi-cycle processor
  - CPI = ~4.25
  - clock period = ~2 ns
- Better design:
  - CPI = 1
  - clock period = ~2ns
- How??
  - Work on multiple instructions at the same time

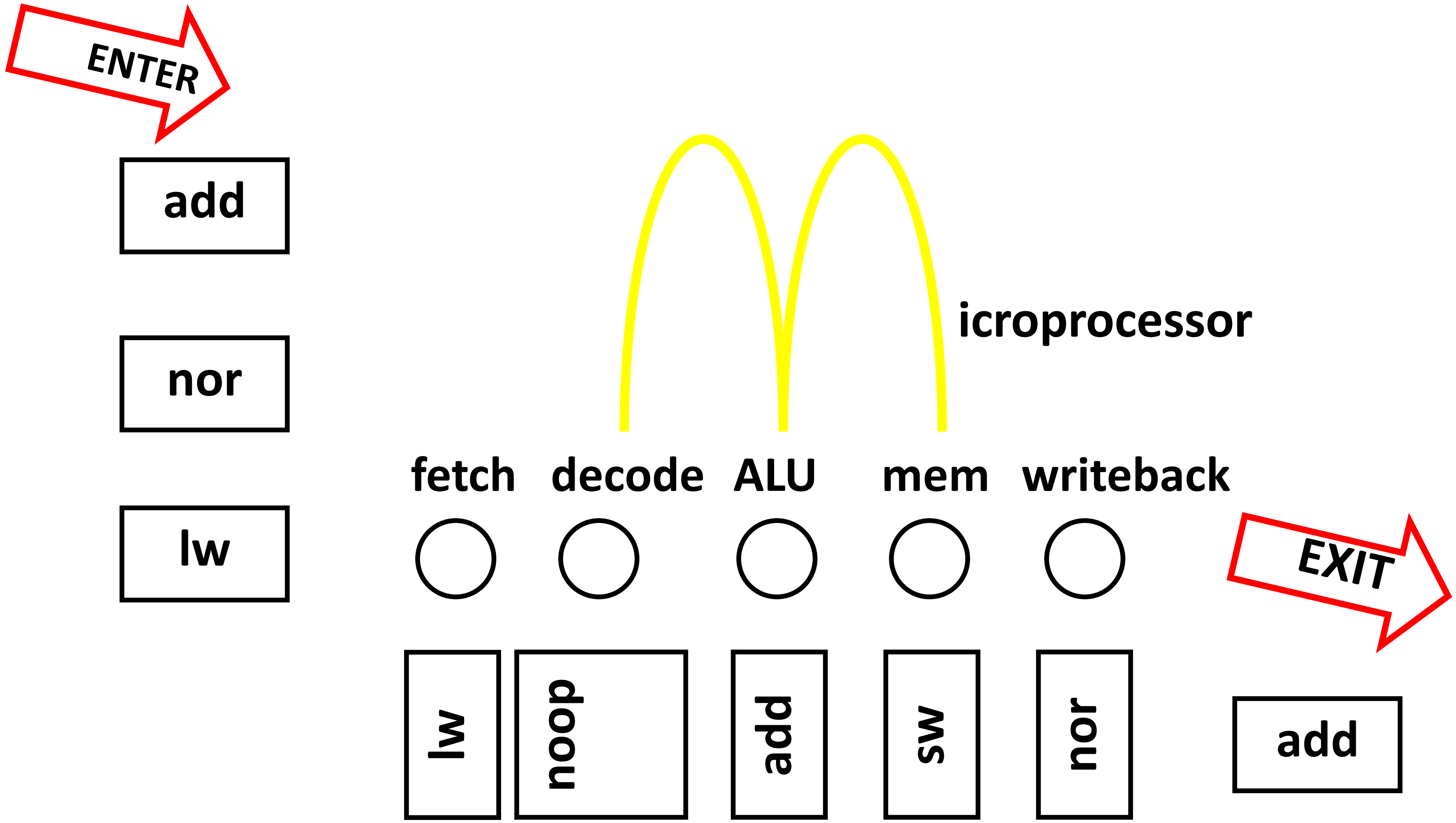
Pipelining

- Want to execute an instruction?
  - Build a processor (multi-cycle)
  - Find instructions
  - Line up instructions (1, 2, 3, …)
  - Overlap execution
    - Cycle #1: Fetch 1
    - Cycle #2: Decode 1 Fetch 2
    - Cycle #3: ALU 1 Decode 2 Fetch 3
    - . . . . . .
  - This is called pipelining instruction execution.
  - Used extensively for the first time on IBM 360 (1960s).
  - CPI approaches 1.

Pipelining



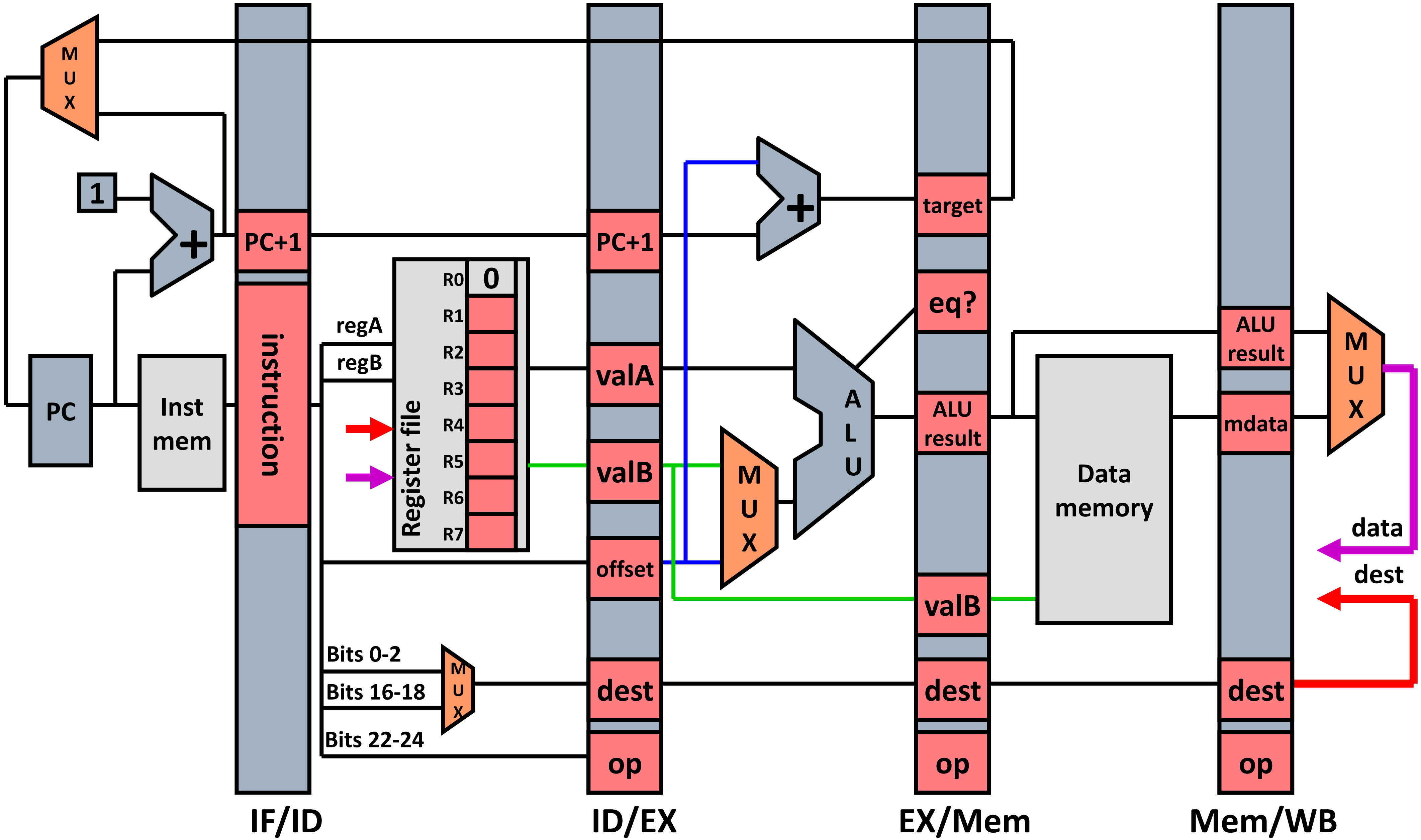
Pipelining


ENTER
add
nor
lw
icroprocessor
fetch
decode
ALU
mem
writeback
lw
noop
add
sw
nor
EXIT
add

Pipelined implementation of LC2K

- Break the execution of the instruction into cycles.
  - Similar to the multi-cycle datapath
- Design a separate datapath stage for the execution performed during each cycle.
  - Build pipeline registers to communicate between the stages.
  - Whatever is on the left gets written onto the right during the next cycle
  - Kinda like the **Instruction Register** in our multi-cycle design, but we'll need one for each stage

Our new pipelined datapath

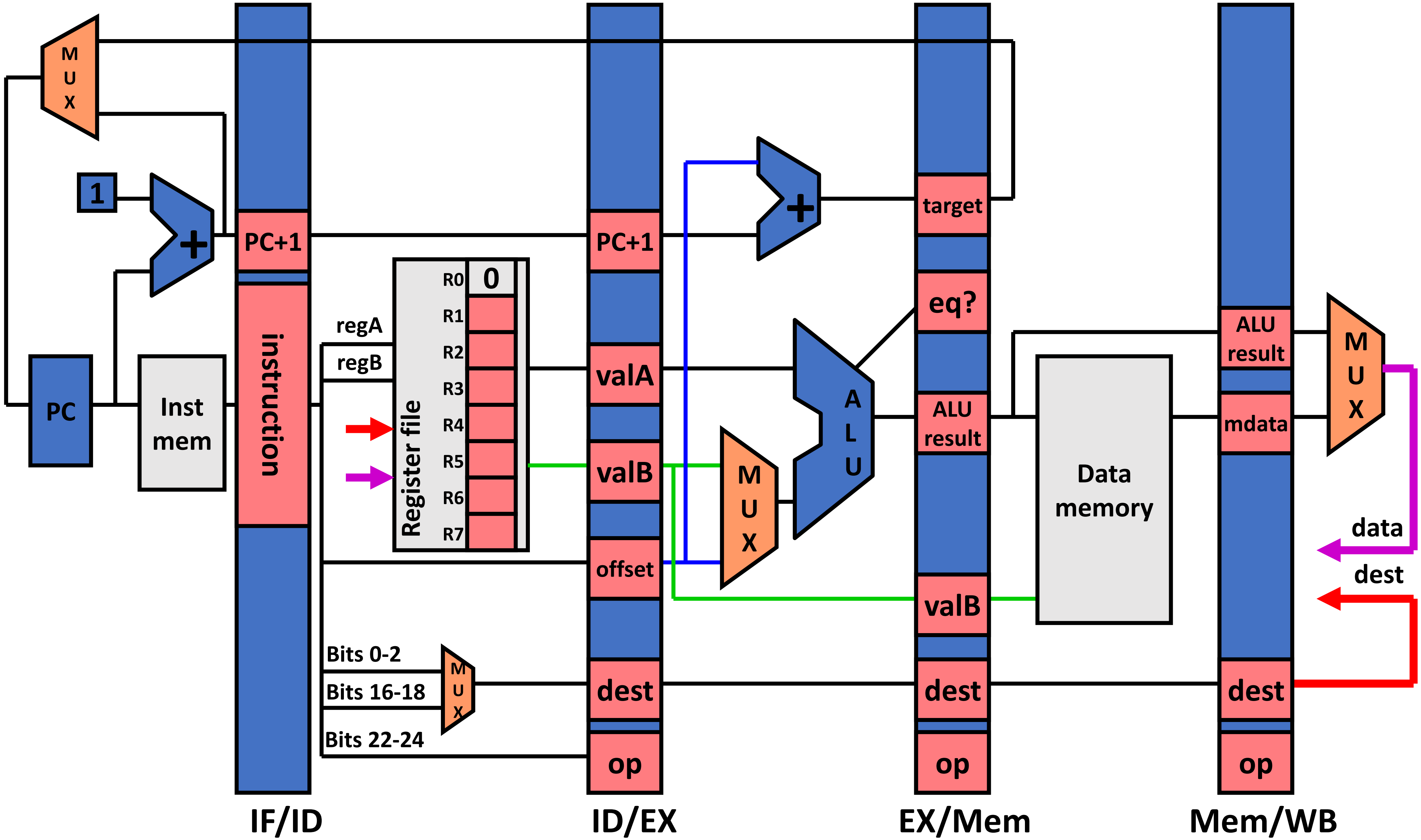


Sample Code (Simple)

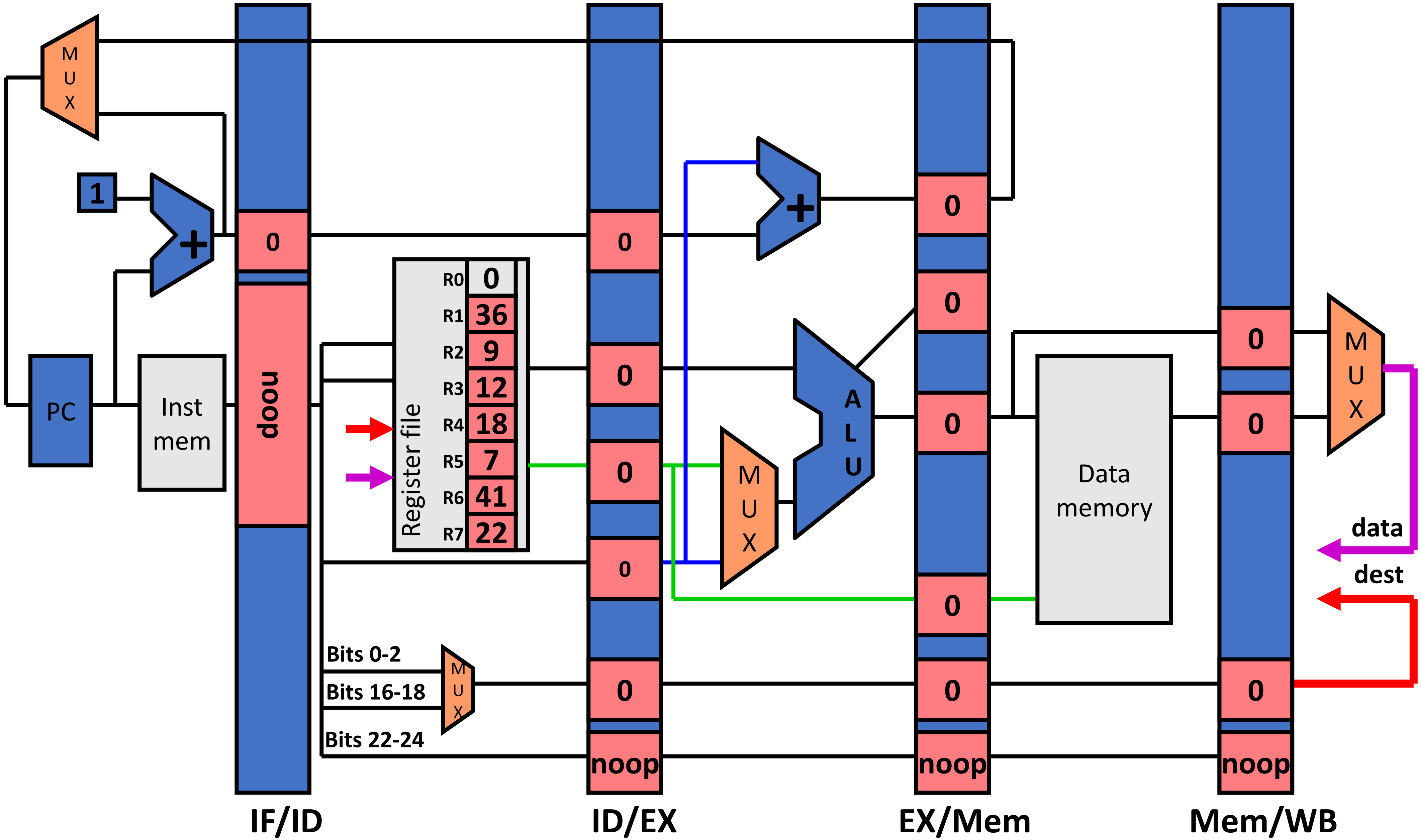
Let's run the following code on pipelined LC2K:

- add 1 2 3 ; reg 3 = reg 1 + reg 2
- nor 4 5 6 ; reg 6 = reg 4 nor reg 5
- lw 2 4 20 ; reg 4 = Mem[reg2+20]
- add 2 5 5 ; reg 5 = reg 2 + reg 5
- sw 3 7 10 ; Mem[reg3+10] =reg 7

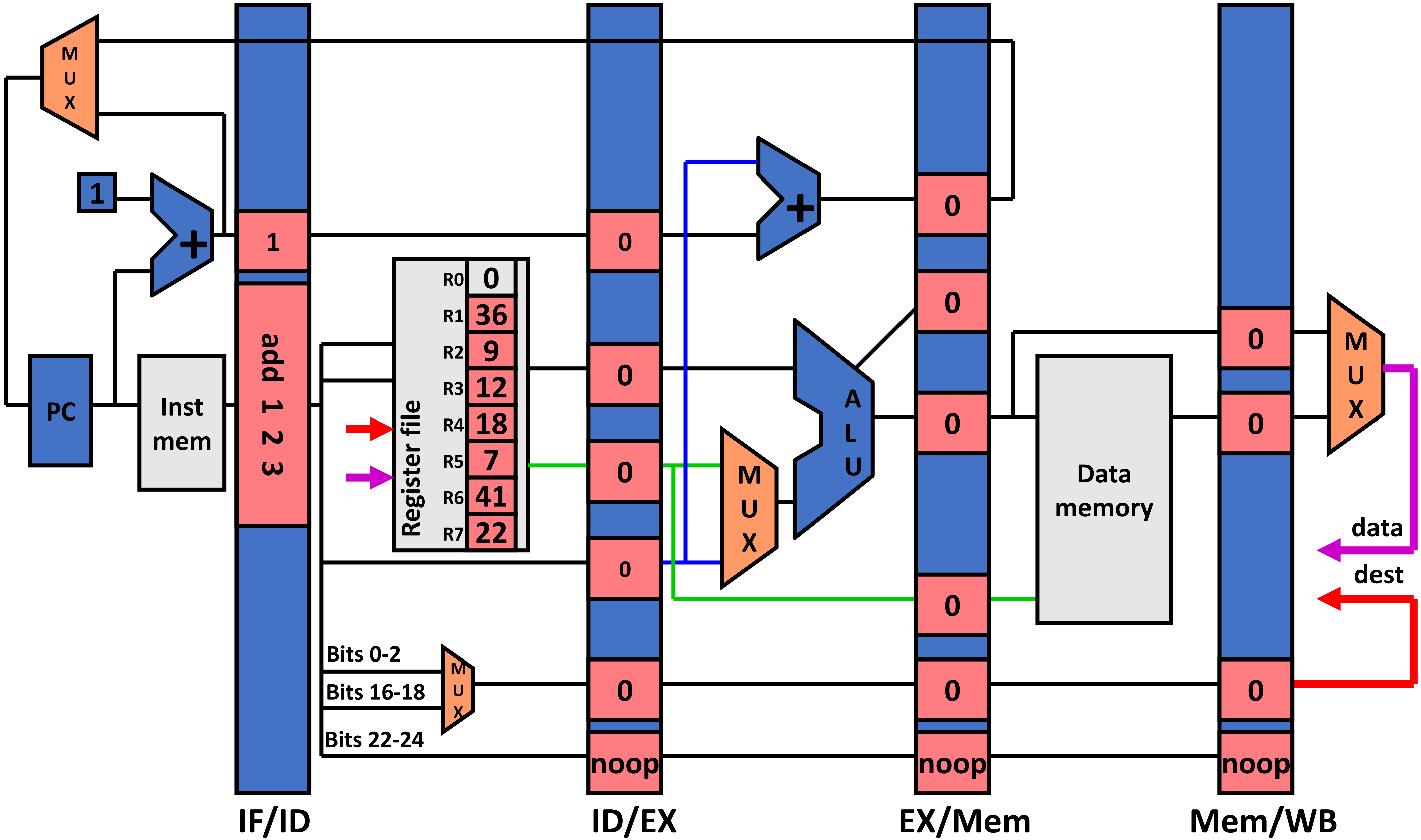
Pipeline datapath



Time 0 - Initial state

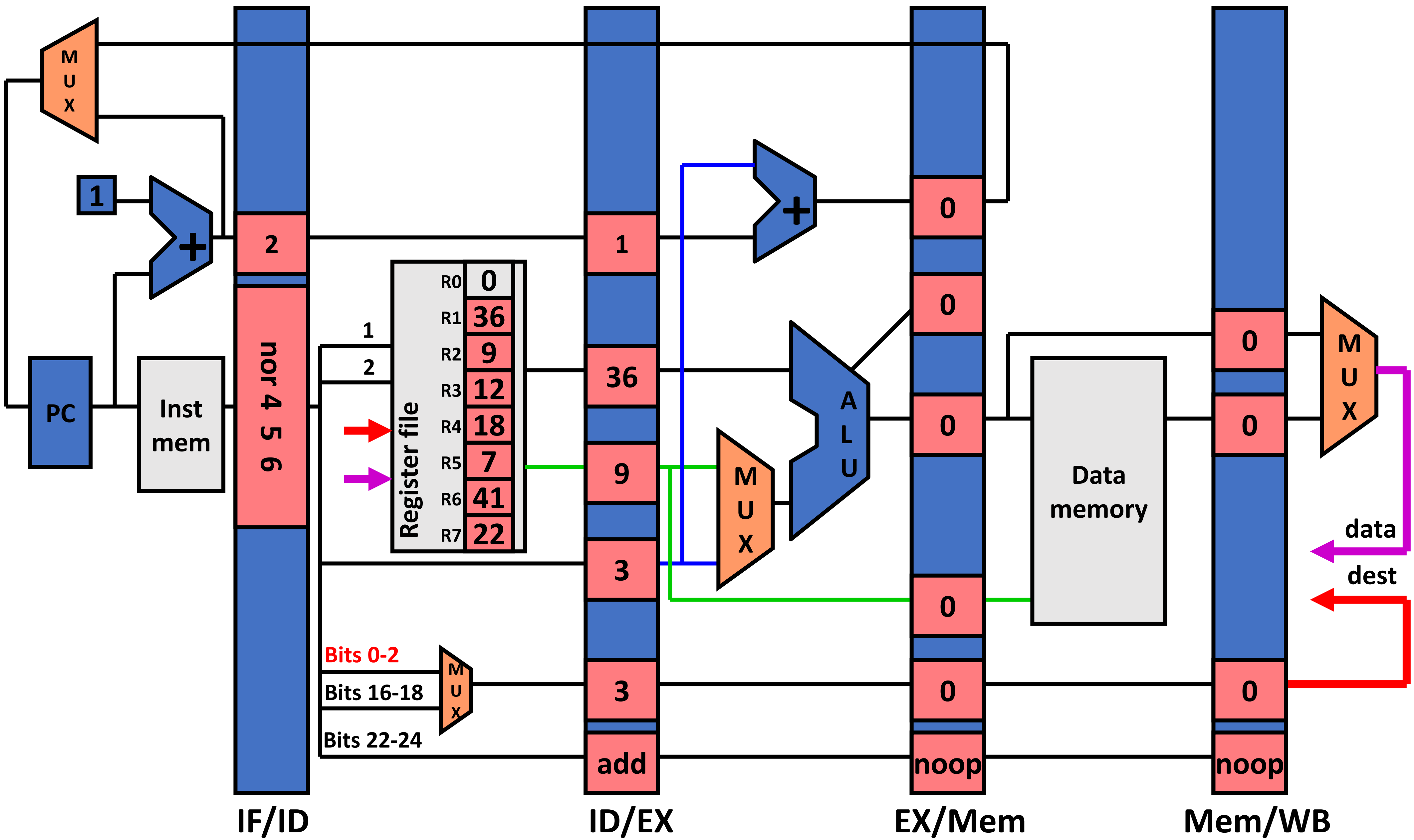


Time 1 - Fetch: add 1 2 3



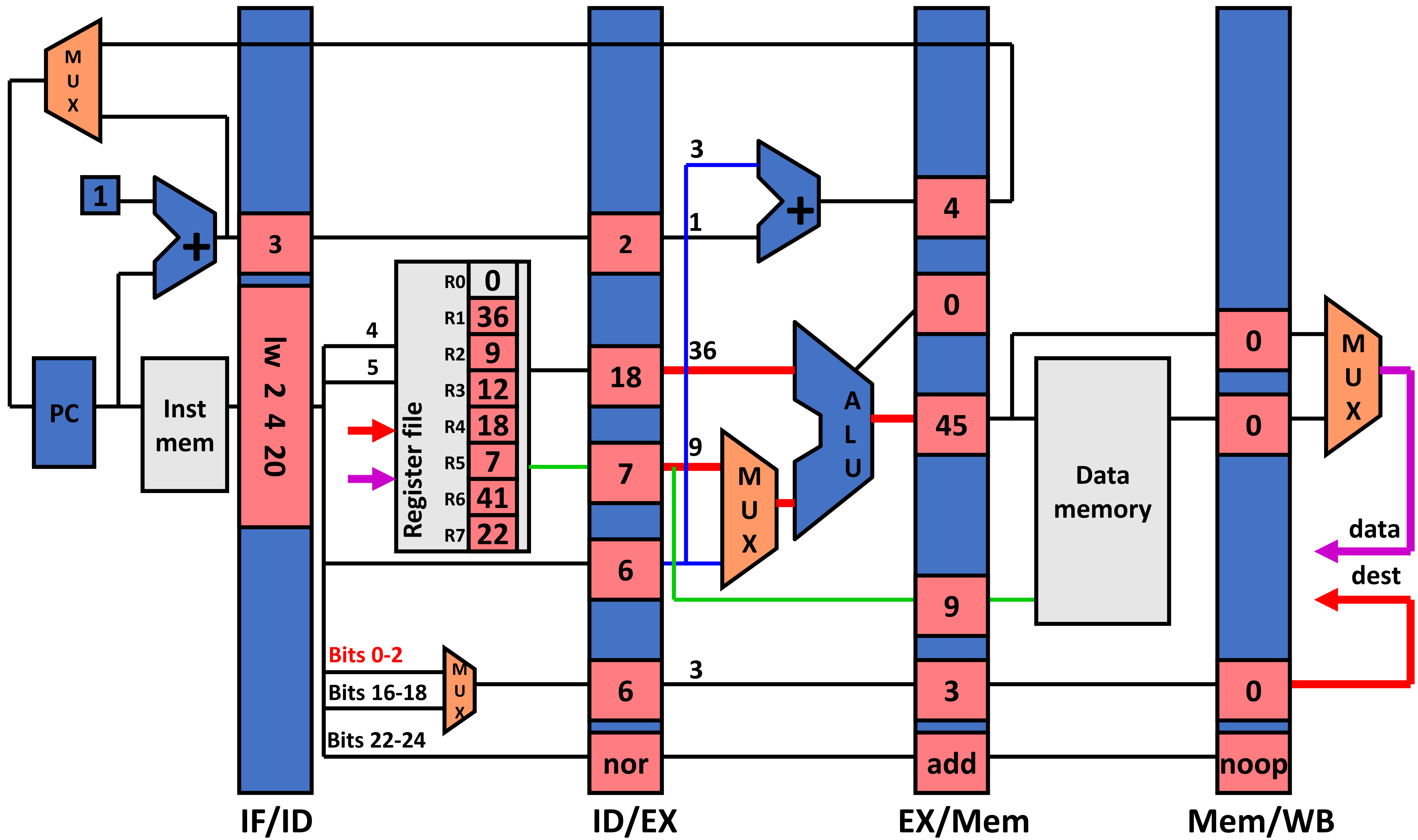
add 1 2 3

Time 2 - Fetch: nor 4 5 6



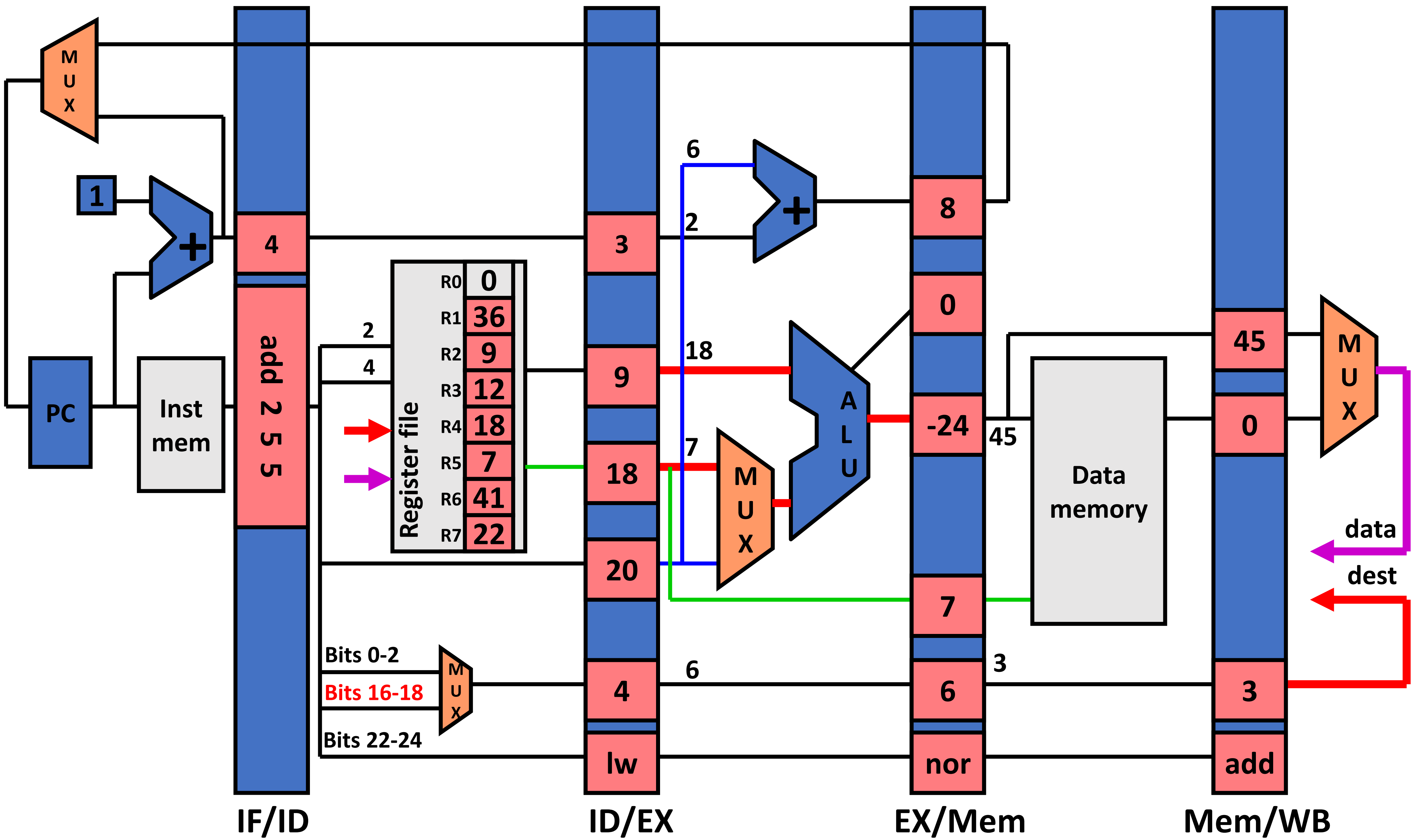
nor 4 5 6 **add 1 2 3**

Time 3 - Fetch: lw 2 4 20



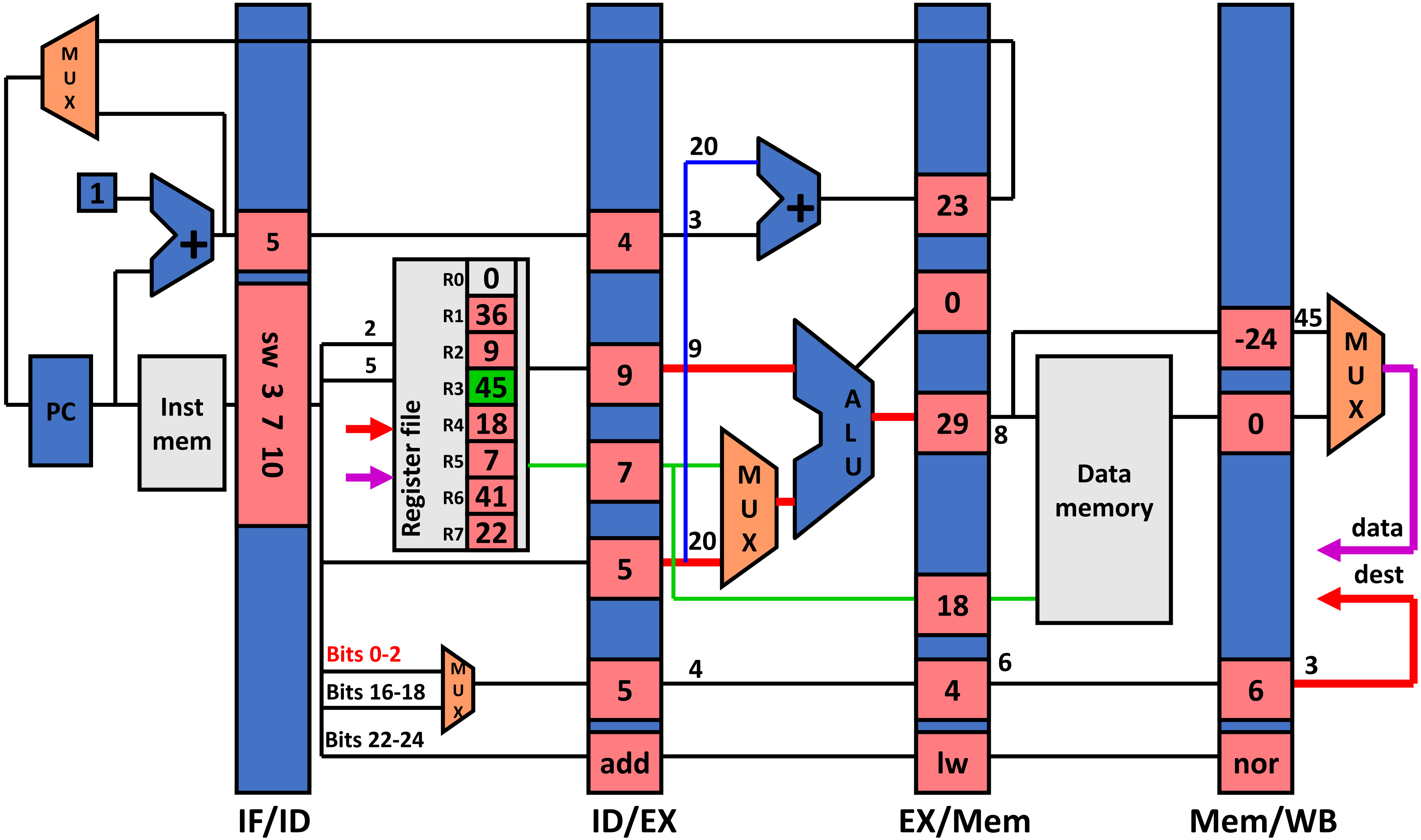
lw 2 4 20 nor 4 5 6 add 1 2 3

Time 4 - Fetch: add 2 5 5



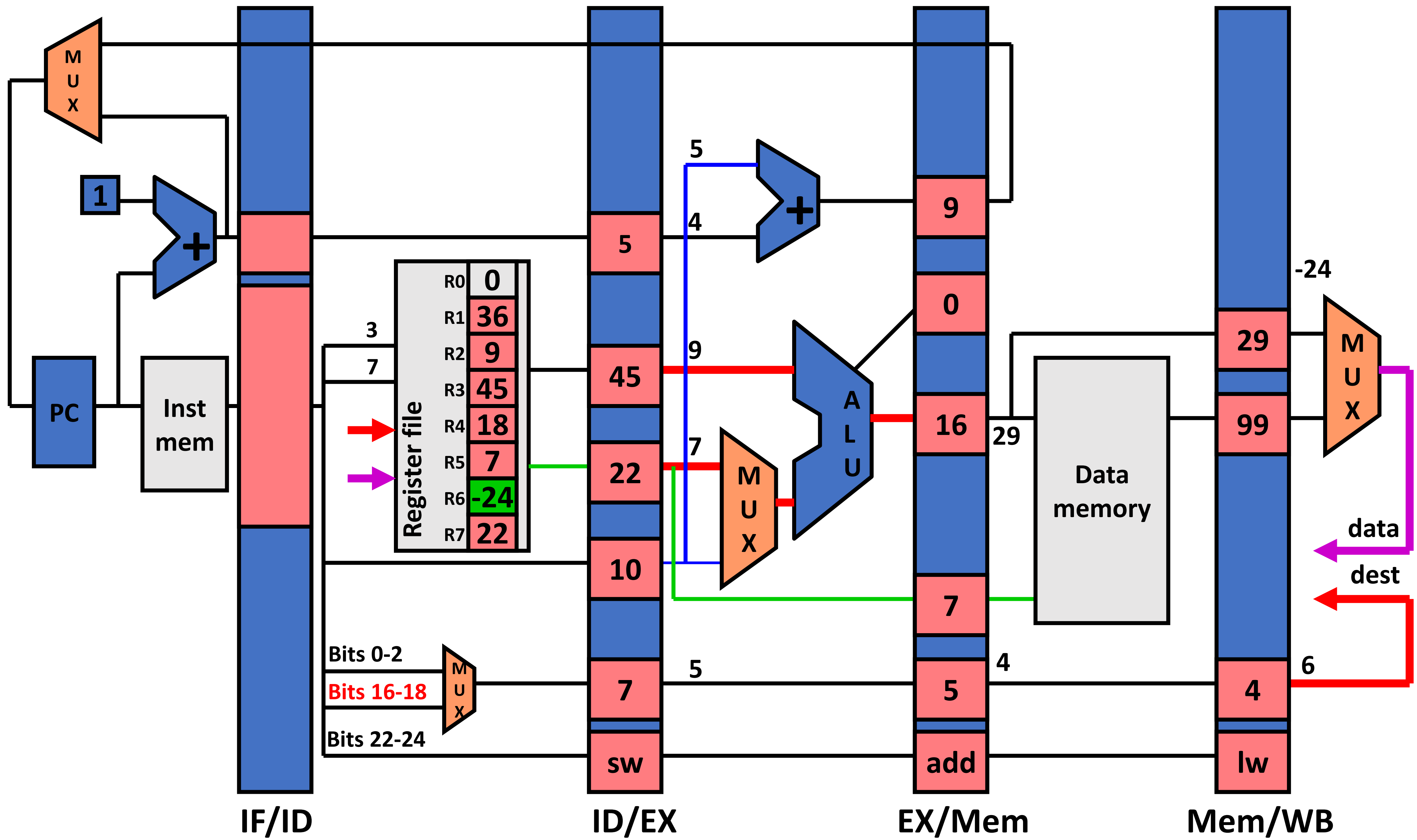
add 2 5 5 | lw 2 4 20 | nor 4 5 6 | add 1 2 3

Time 5 - Fetch: sw 3 7 10



sw 3 7 10 **add 2 5 5** **lw 2 4 20** **nor 4 5 6** **add**

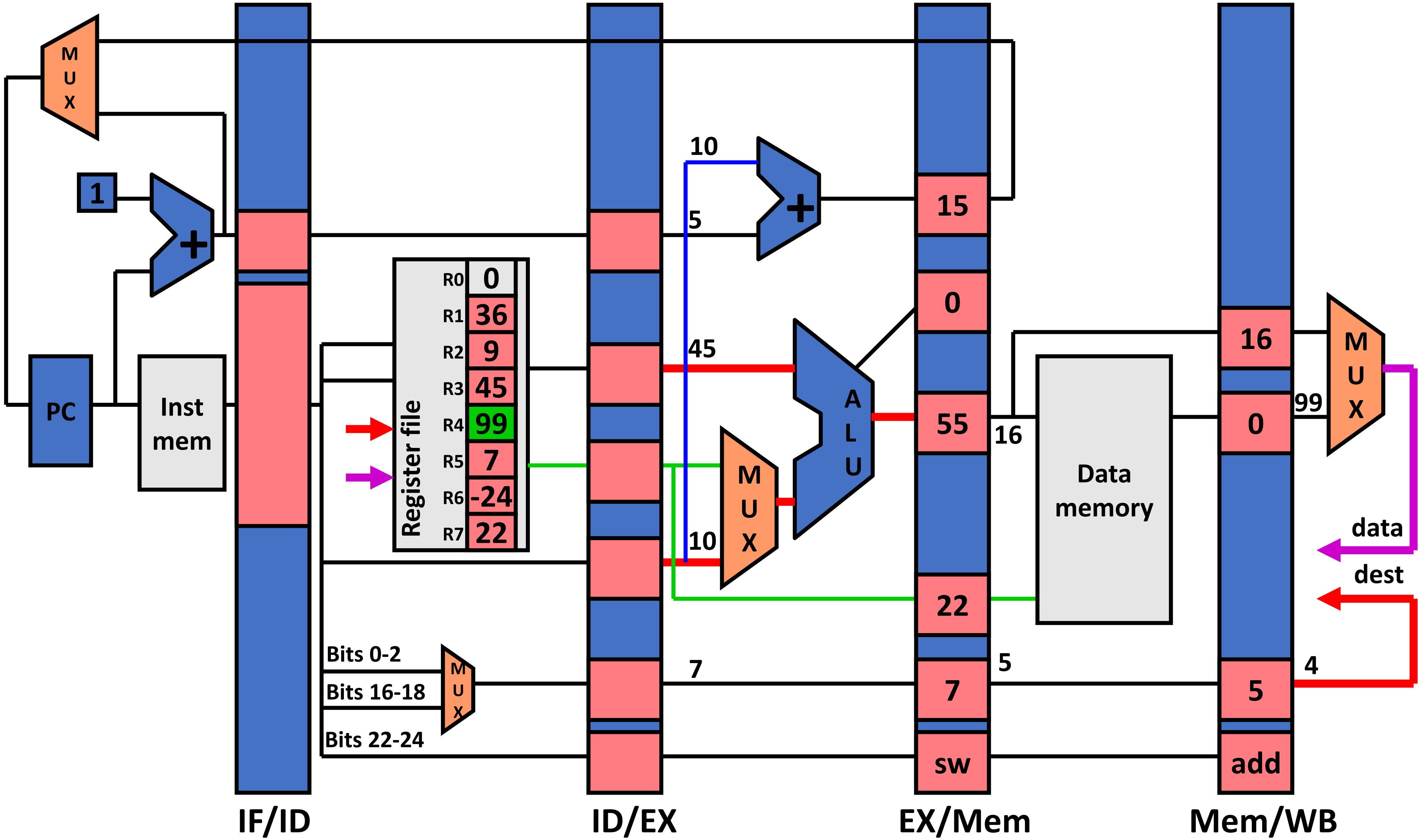
Time 6 – no more instructions



sw 3 7 10 add 2 5 5 lw 2 4 20 nor

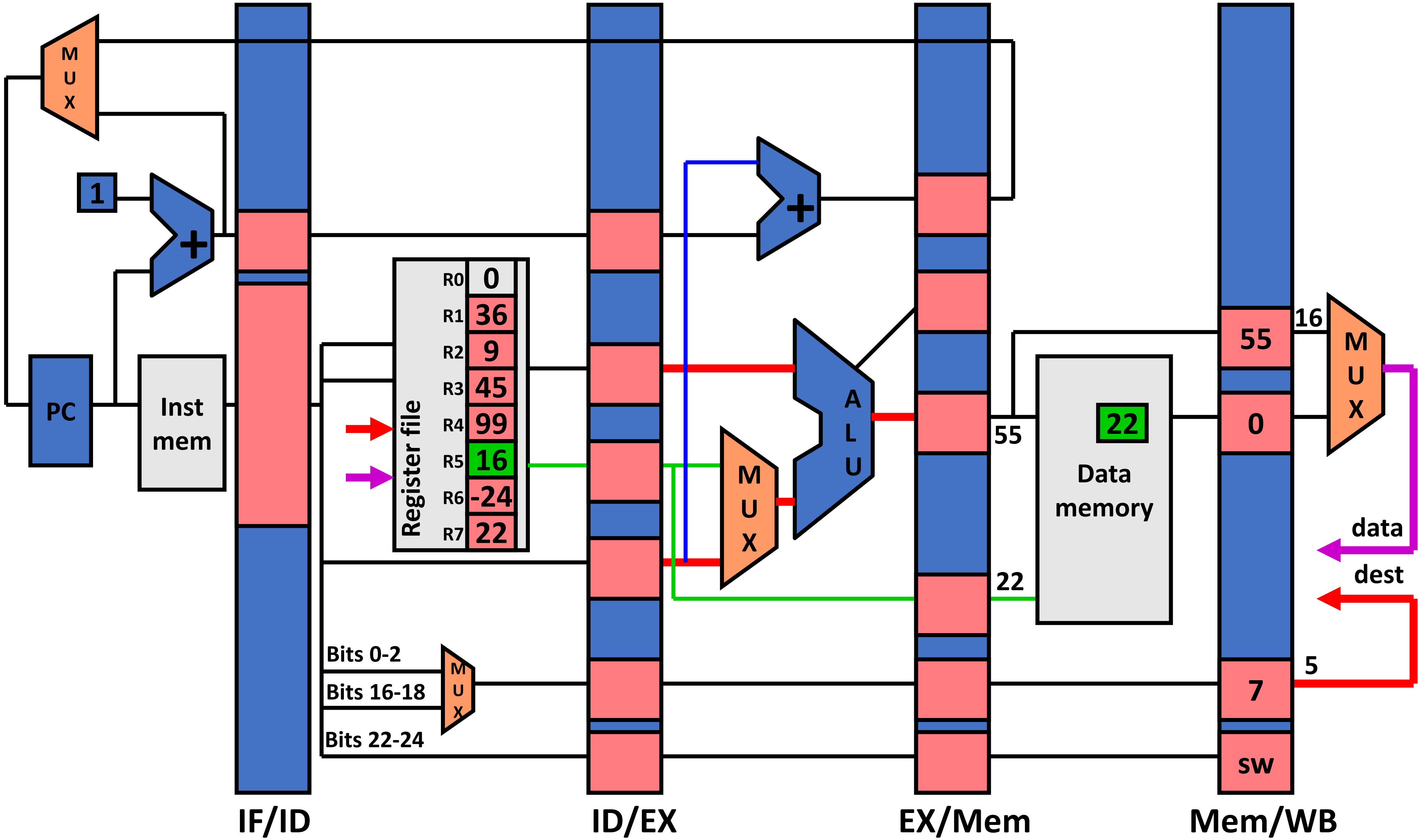
Time 7 – no more instructions

Poll: What all happens on the next clock cycle?



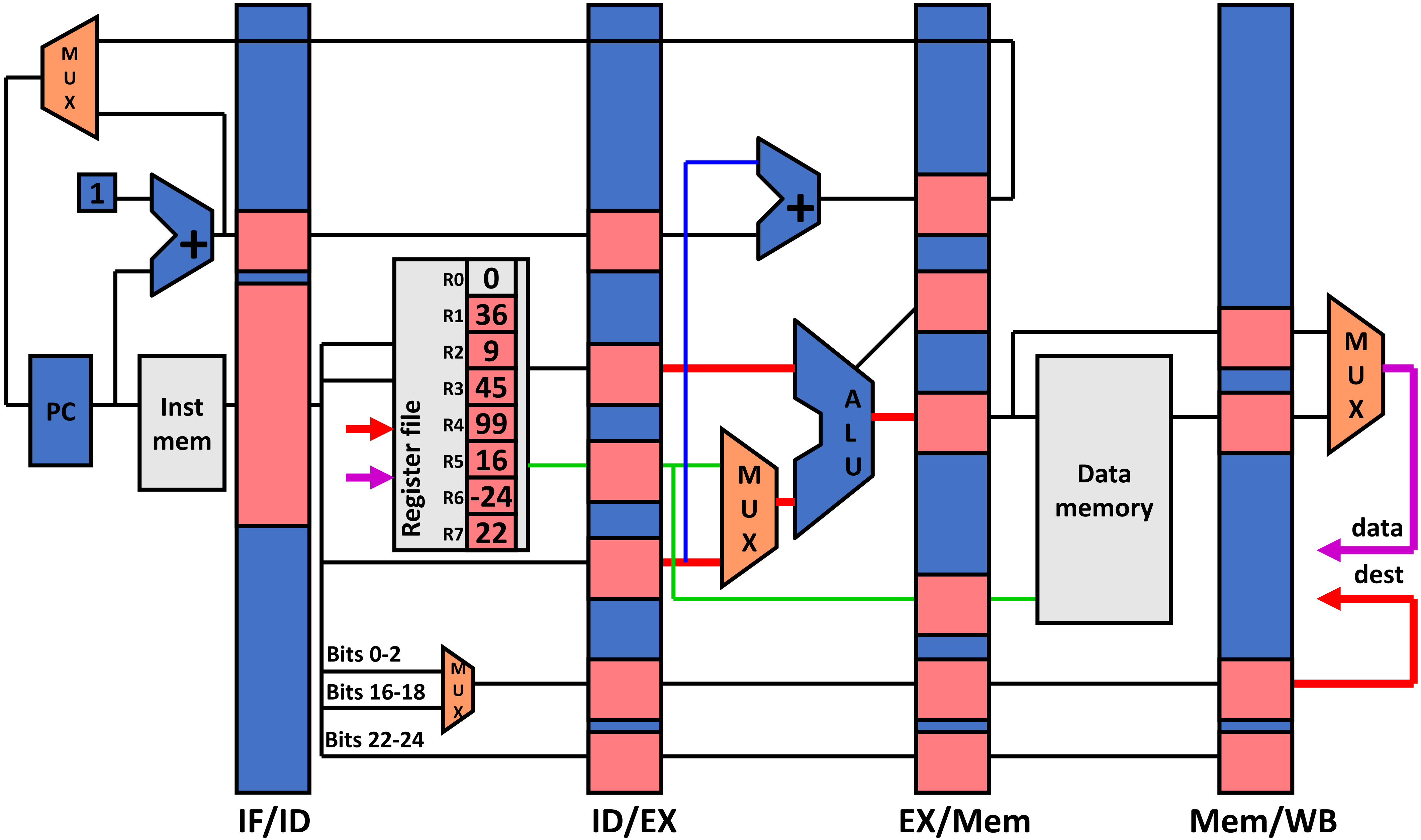
sw 3 7 10 **add 2 5 5** **lw**

Time 8 – no more instructions



sw 3 7 10 add

Time 9 – no more instructions



sw

Pipelining - What can go wrong?

- Data hazards: since register reads occur in stage 2 and register writes occur in stage 5 it is possible to read an old / stale value before the correct value is written back.
- Control hazards: A branch instruction may change the PC, but not until stage 4. What do we fetch before that?
- Exceptions: Sometimes we need to pause execution, switch to another task (maybe the OS), and then resume execution… how to we make sure we resume at the right spot
- Now - Data hazards
  - What are they?
  - How do you detect them?
  - How do you deal with them?

Pipeline function for ADD

- Fetch: read instruction from memory
- Decode: **read source operands from reg**
- Execute: calculate sum
- Memory: pass results to next stage
- Writeback: **write sum into register file**

Data Hazards

Recall: registers
are read /sourced
In the "decode" stage

add	1	2	3
nor	3	4	5

Read-after-write
(RAW) Dependency

time

add	fetch	decode	execute	memory	writeback	
nor		fetch	decode	execute	memory	writeback

Hazard

If not careful, nor will read a stale value of register 3

Data Hazards

add 1 2 3

nor 3 4 5

time

add	fetch	decode	execute	memory	writeback	
nor		fetch	decode*	decode*	decode	execute

Assume Register File gives the right value of register 3 when read/written during <u>same</u> cycle. This is consistent with most processors (ARM/x86), <u>but not Project 3</u>.

Definitions

- Data Dependency: *one instruction uses the result of a previous one*
  - Doesn't necessarily cause a problem
- Data Hazard: *one instruction has a data dependency that will cause a problem if we don't "deal with it"*

Class Problem 1

Poll: Which of these instructions has a data dependency on an earlier one? Which of those are data hazards in our 5-stage pipeline?

1. add 1 2 3
2. nor 3 4 5
3. add 6 3 7
4. lw 3 6 10
5. sw 6 2 12

What about here?

1. add 1 2 3
2. beq 3 4 1
3. add 3 5 6
4. add 3 6 7

Class Problem 1

Which read-after-write (RAW) dependences do you see?

Which of those are data hazards?

1. add 1 2 3
2. nor 3 4 5
3. add 6 3 7
4. lw 3 6 10
5. sw 6 2 12

What about here?

1. add 1 2 3
2. beq 3 4 1
3. add 3 5 6
4. add 3 6 7

Three approaches to handling data hazards

- Avoid
  - Make sure there are no hazards in the code
- Detect and Stall
  - If hazards exist, stall the processor until they go away.
- Detect and Forward
  - If hazards exist, fix up the pipeline to get the correct value (if possible)