12. Pipeline Data Hazards

EECS 370 – Introduction to Computer Organization

Fall 2013

Profs. Valeria Bertacco, Robert Dick, and Satish Narayanasamy

EECS Department

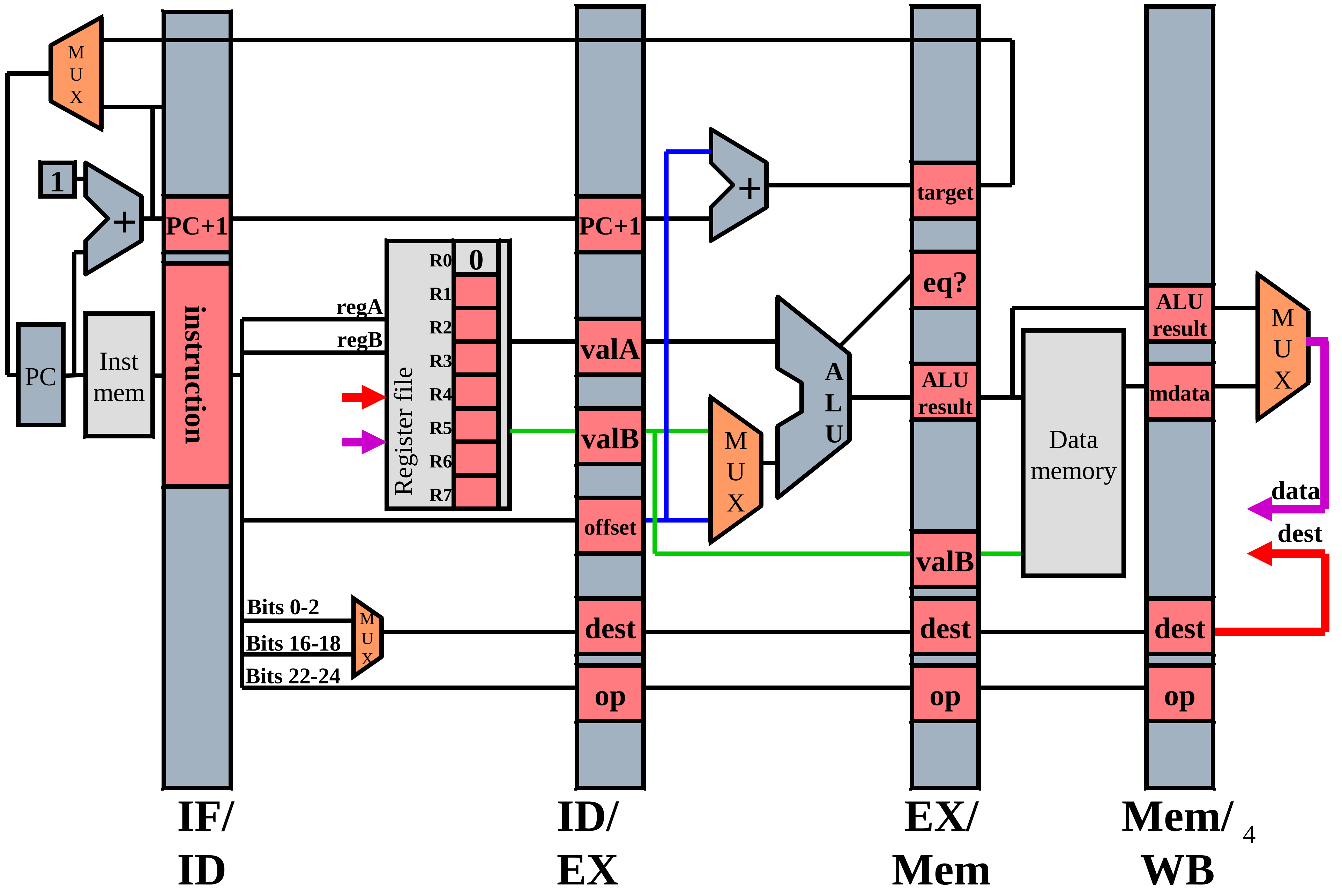
University of Michigan in Ann Arbor, USA

Announcements

- Project 2s autograder was buggy. Fixed now. Was passing invalid simulators. If your final submission was before 23 October, resubmit.
- See email for common problems that autograder previously missed.
- 24 October: Homework 4 due.
- 6:00 PM 28 October Project 2s and competition due.
- 25 October: Rest of Project 2 due.
- 5 November: Homework 5 due.
- 12 November: Project 3 due.

Pipelining - What can go wrong?

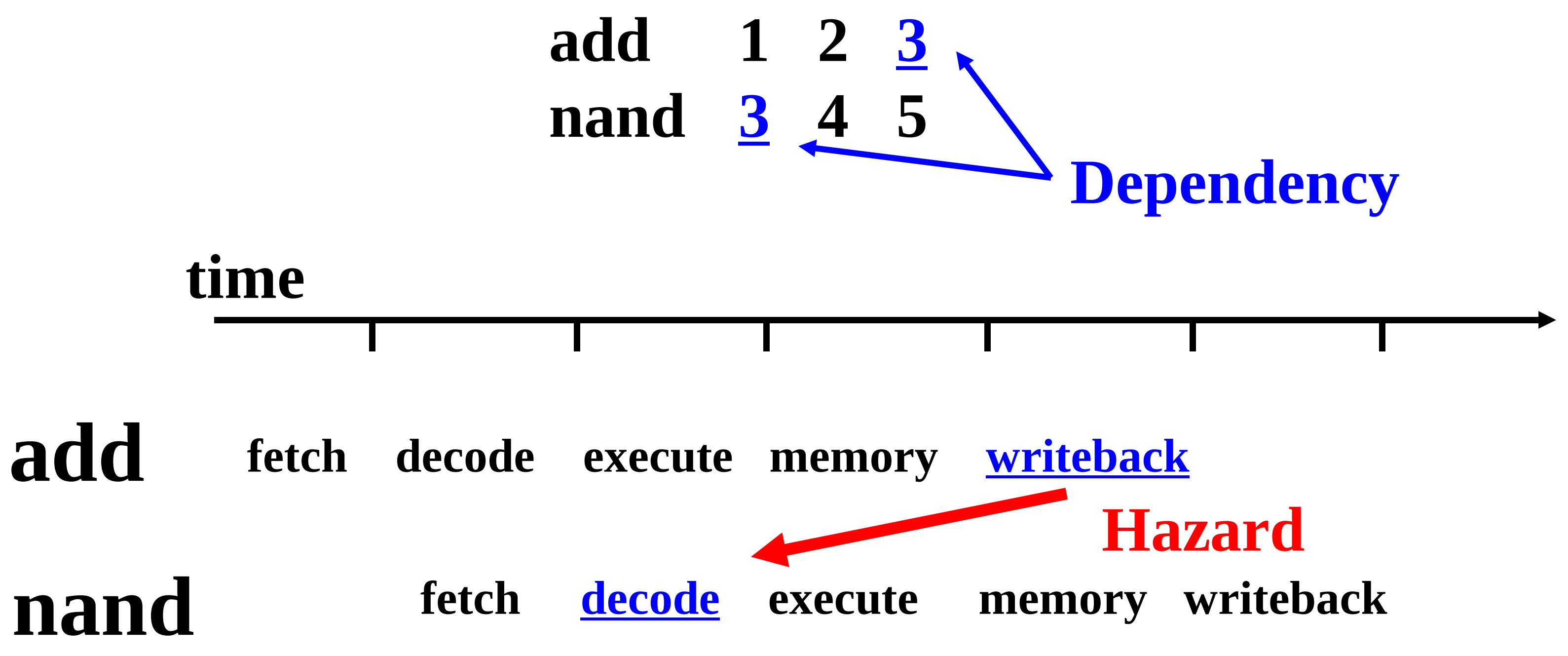
- Data hazards: since register reads occur in stage 2 and register writes occur in stage 5 it is possible to read the wrong value if is about to be written.
- Control hazards: A branch instruction may change the PC, but not until stage 4. What do we fetch before that?
- Exceptions: How do you handle exceptions in a pipelined processor with 5 instructions in flight?
- Today - Data hazards
  - What are they?
  - How do you detect them?
  - How do you deal with them?


M U X
1
+
PC+1
instruction
PC
Inst mem
regA
regB
Register file
R0
0
R1
R2
R3
R4
R5
R6
R7
PC+1
valA
valB
offset
+
M U X
A L U
target
eq?
ALU result
valB
Data memory
ALU result
mdata
M U X
data
dest
Bits 0-2
Bits 16-18
Bits 22-24
M U X
dest
op
dest
op
dest
op
IF/ ID
ID/ EX
EX/ Mem
Mem/ WB

Pipeline function for ADD

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

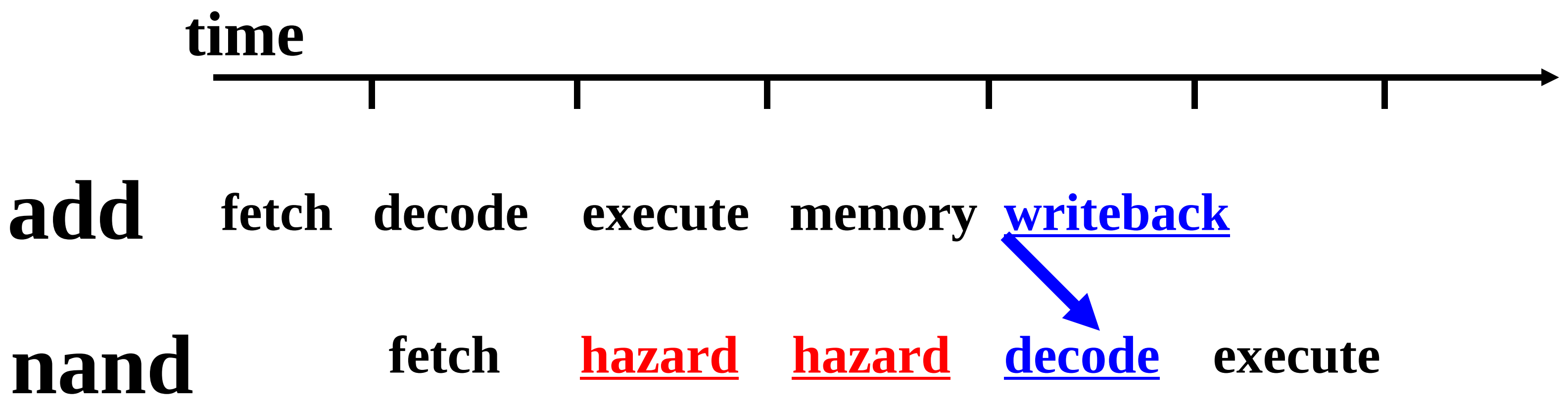
Data Hazards


add 1 2 3
nand 3 4 5
Dependency
time
add fetch decode execute memory writeback
Hazard
nand fetch decode execute memory writeback

If not careful, nand will read the wrong value of R3

Data Hazards

```
add    1  2  3
nand   3  4  5
```



Assume Register File gives the right value of R3 when read/written during same cycle. This is consistent with the book and the MIPS architecture, but not Project 3.

Class Problem 1

Which data hazards do you see?

```
add  1  2  3
nand  3  4  5
add  6  3  7
lw  3  6  10
sw  6  2  12
```

What about here?

```
add 1  2  3
beq 3  4  1
add  3  5  6
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, augment the pipeline to get the values where they are needed with reduced stalls

Handling data hazards I: Avoid all hazards

- Assume the programmer (or the compiler) knows about the processor implementation.
  - Make sure no hazards exist.
    - Put noops between any dependent instructions.

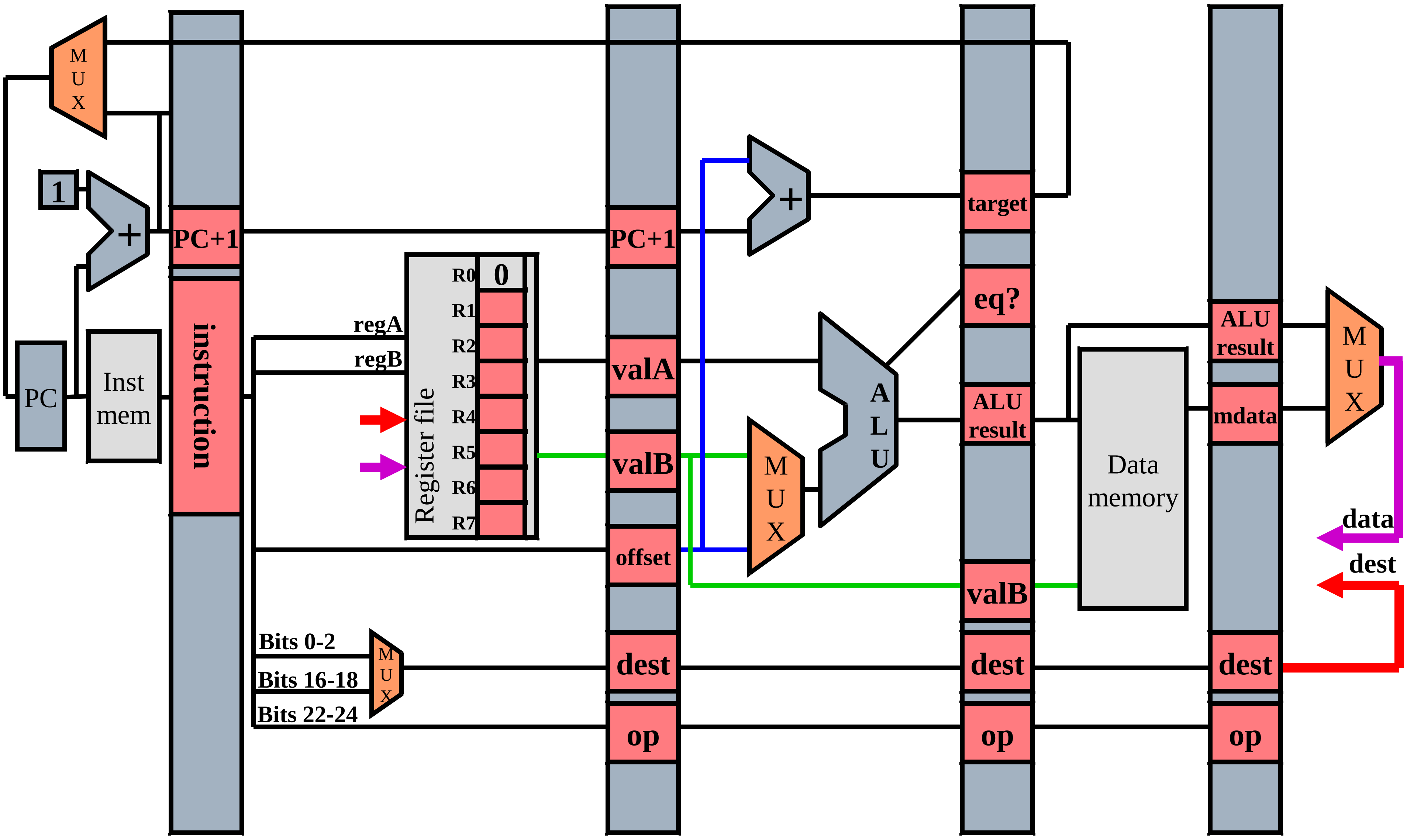
```
add    1   2   3   ←——   write R3 in cycle 5
noop
noop
nand   3   4   5   ←——   read R3 in cycle 5
```

Problems with this solution

- Old programs (legacy code) may not run correctly on new implementations
  - Longer pipelines need more noops
- Programs get larger as noops are included
  - Especially a problem for machines that try to execute more than one instruction every cycle
  - Intel EPIC: Often 25% - 40% of instructions are noops
- Program execution is slower
  - CPI is 1, but some instructions are noops

Handling data hazards II: Detect and stall until ready

- Detect:
  - Compare regA with previous DestRegs
    - 3 bit operand fields
  - Compare regB with previous DestRegs
    - 3 bit operand fields
- Stall:
  - Keep current instructions in fetch and decode
  - Pass a noop to execute

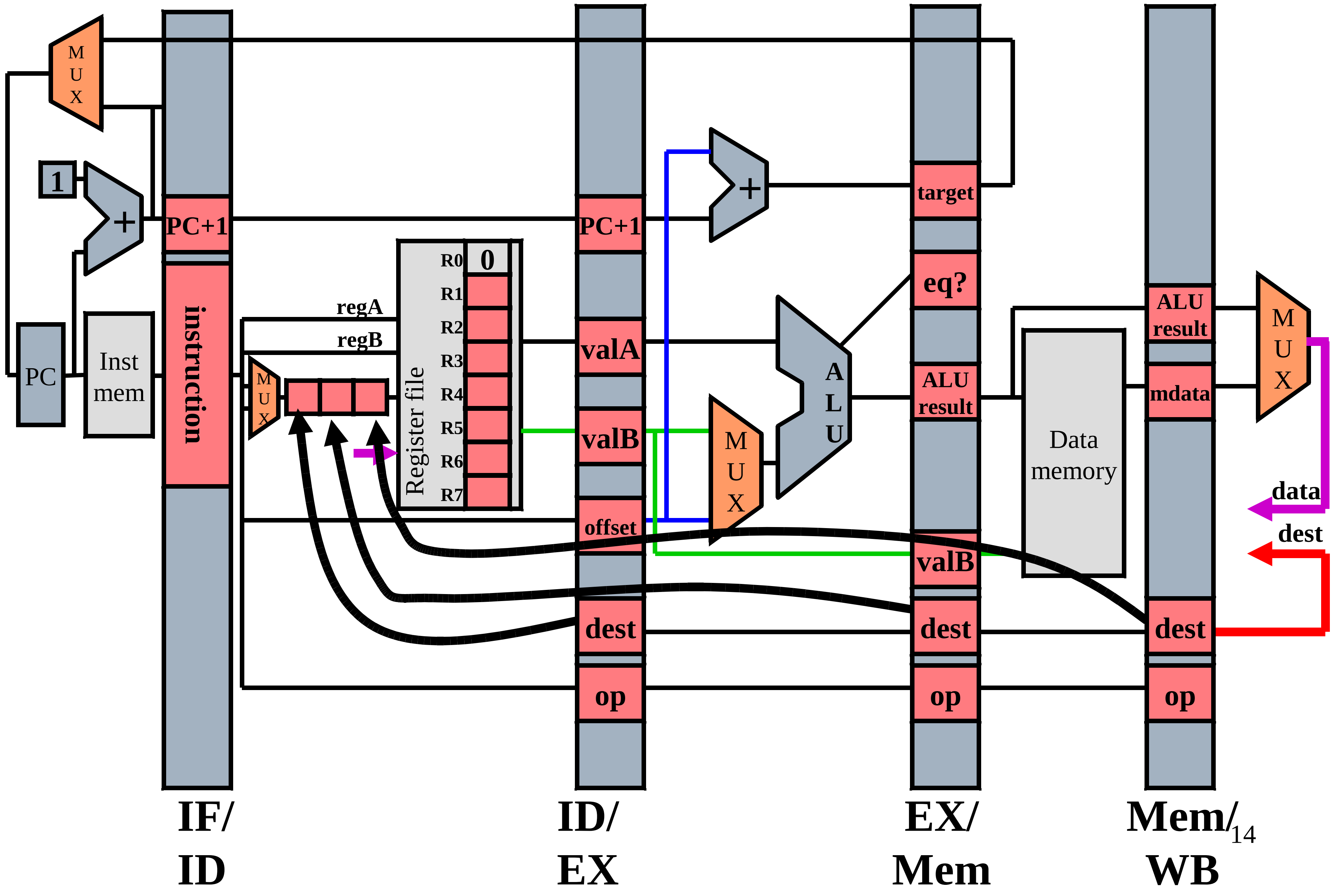


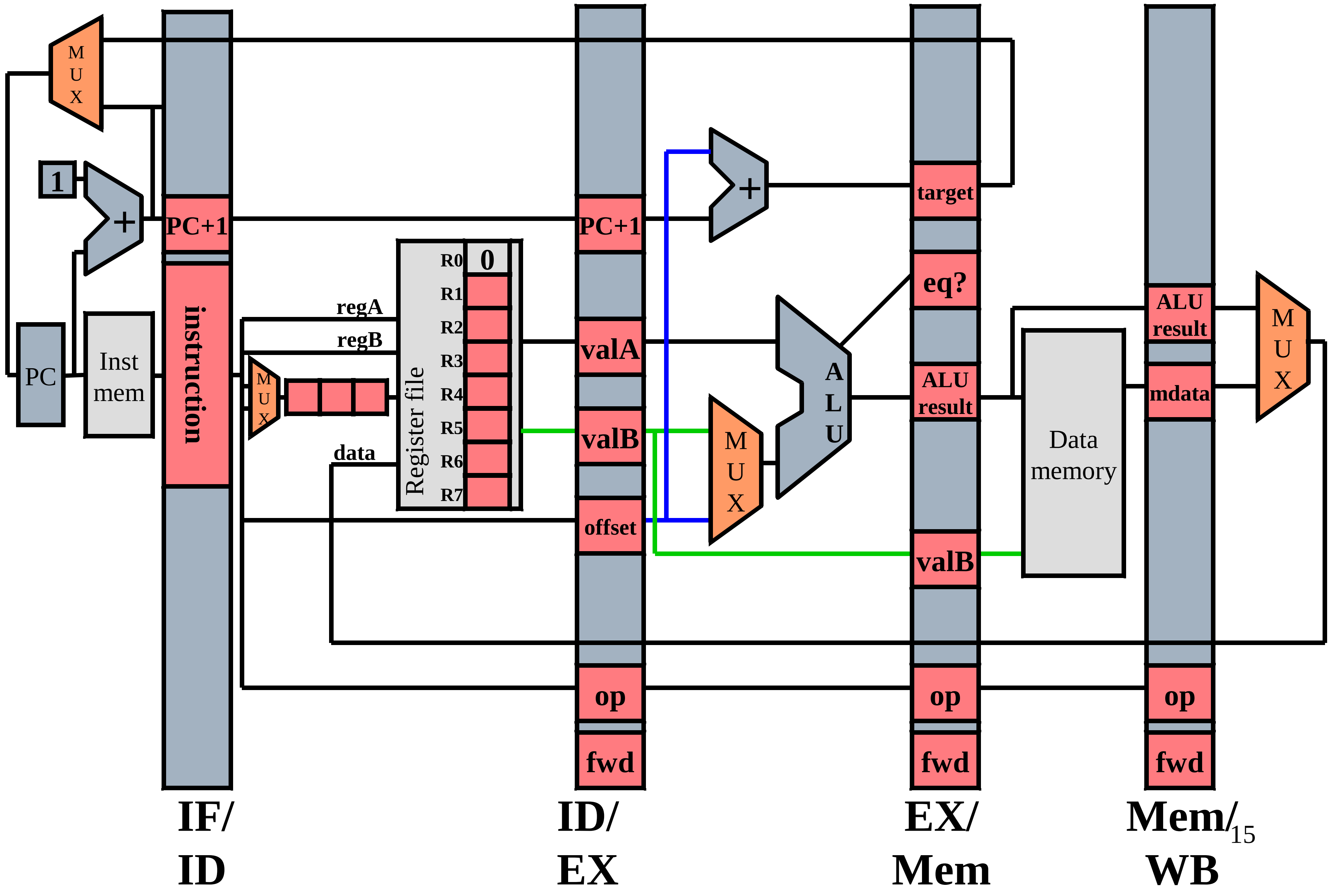
IF/ ID

ID/ EX

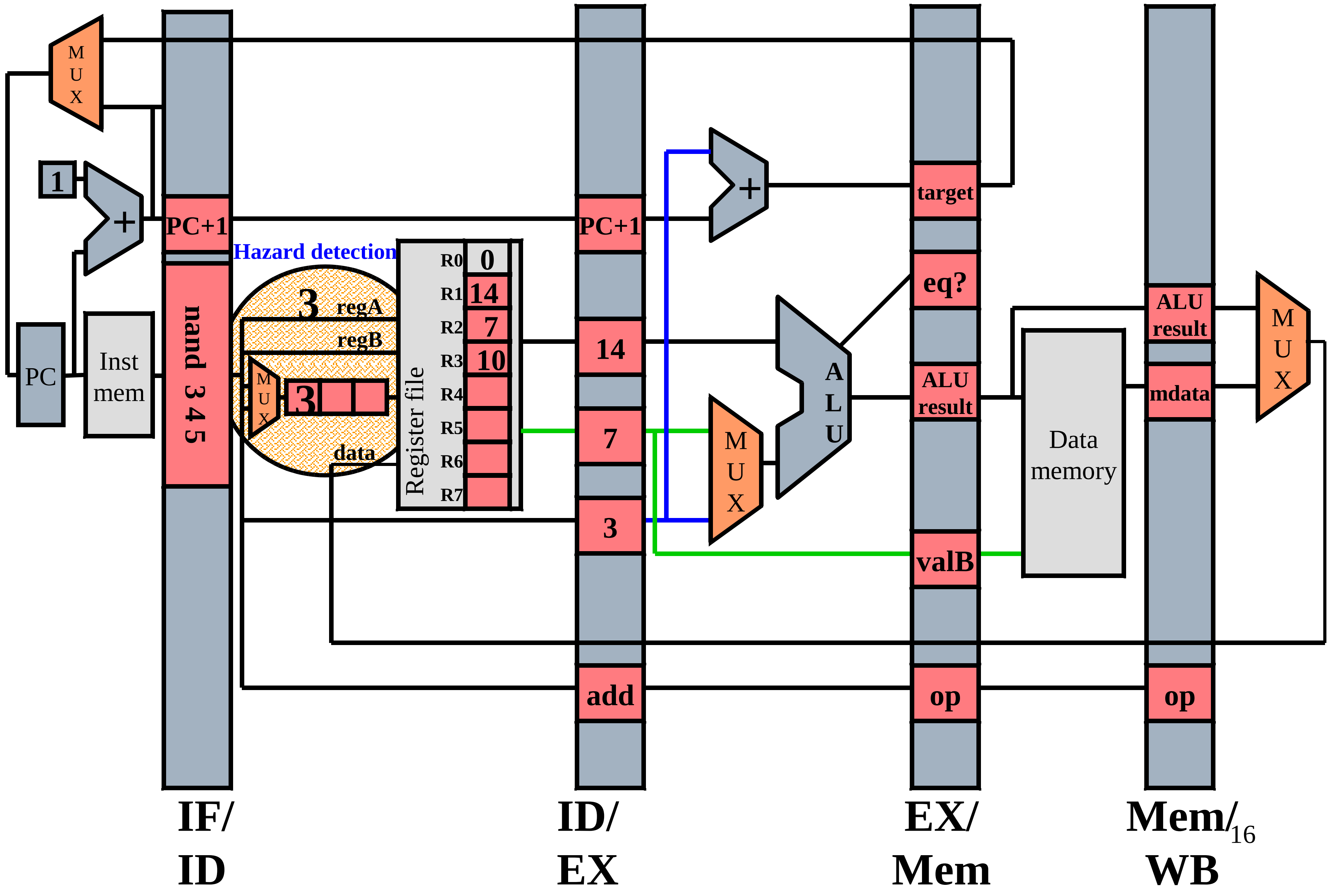
EX/ Mem

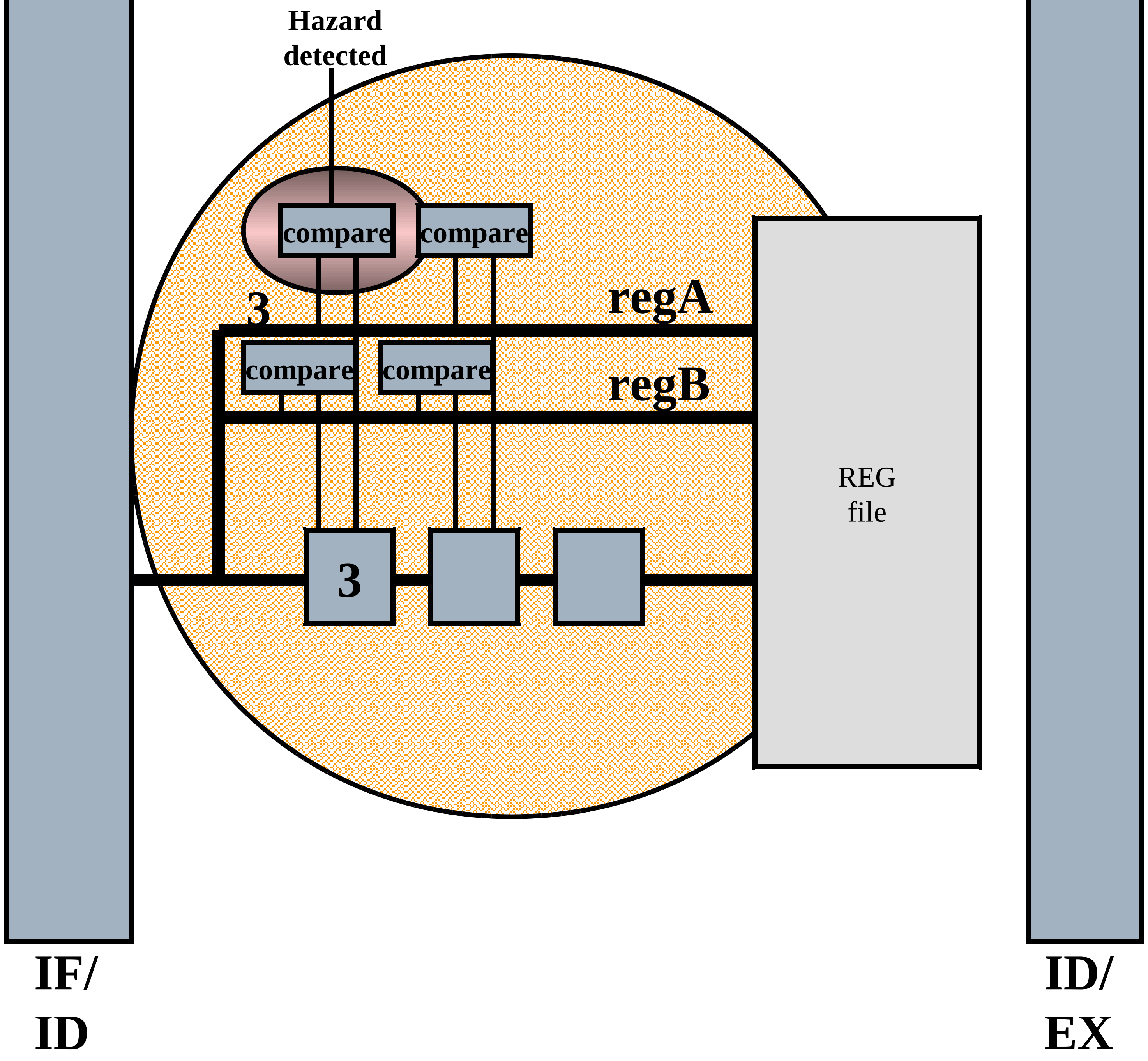
Mem/ WB

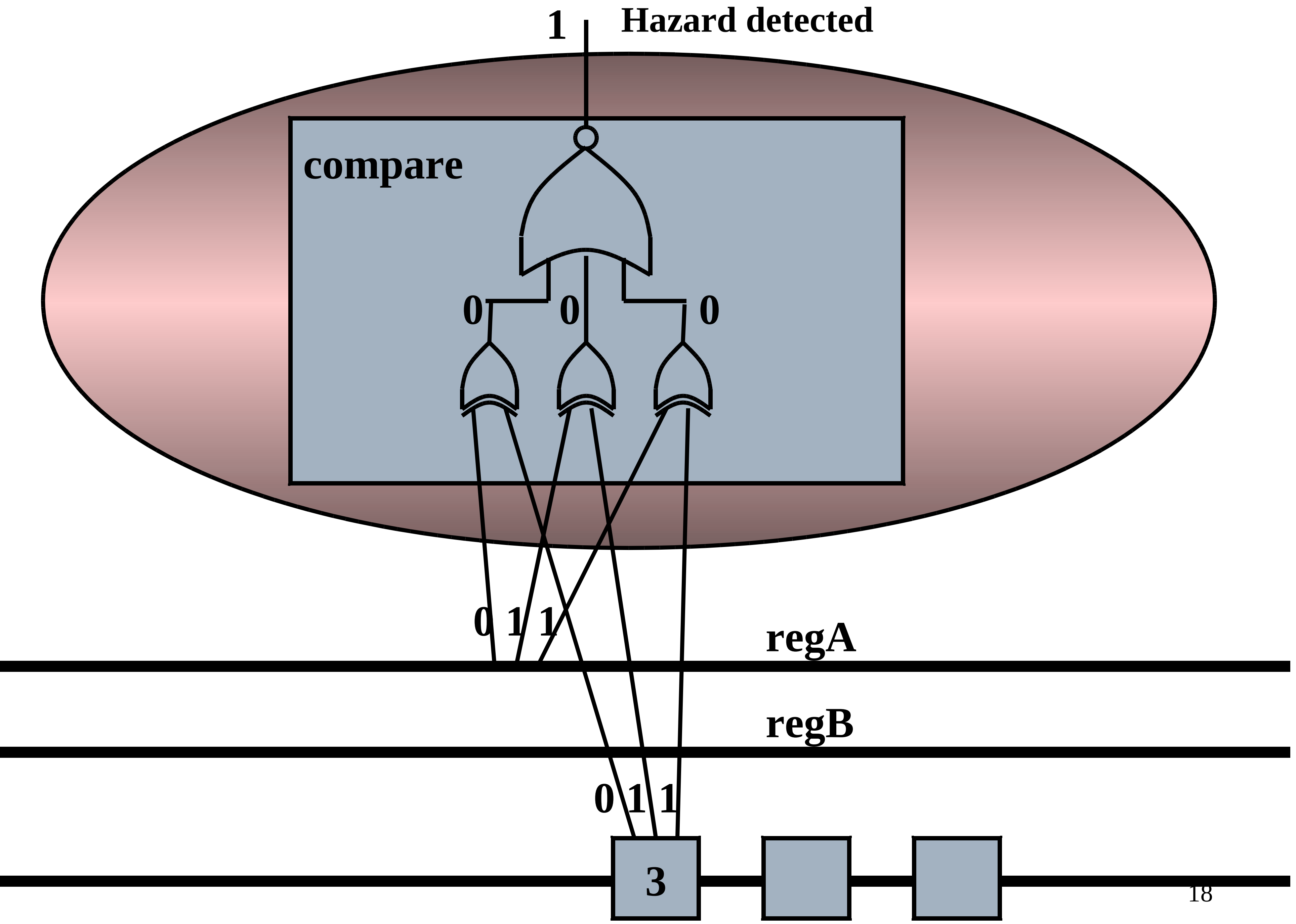

M U X
1
+
PC
Inst mem
PC+1
instruction
regA
regB
M U X
Register file
R0
0
R1
R2
R3
R4
R5
R6
R7
PC+1
valA
valB
offset
dest
op
+
M U X
A L U
target
eq?
ALU result
valB
dest
op
Data memory
ALU result
mdata
dest
op
M U X
data
dest
IF/ ID
ID/ EX
EX/ Mem
Mem/ WB


MUX
1
+
PC+1
PC
Inst mem
instruction
MUX
regA
regB
data
Register file
R0
0
R1
R2
R3
R4
R5
R6
R7
PC+1
valA
valB
offset
op
fwd
+
MUX
ALU
target
eq?
ALU result
valB
op
fwd
Data memory
ALU result
mdata
op
fwd
MUX
IF/ ID
ID/ EX
EX/ Mem
Mem/ WB

First half of cycle 3




Hazard detected
compare
compare
3
regA
compare
compare
regB
REG file
3
IF/ ID
ID/ EX


1
Hazard detected
compare
0
0
0
0 1 1
regA
regB
0 1 1
3

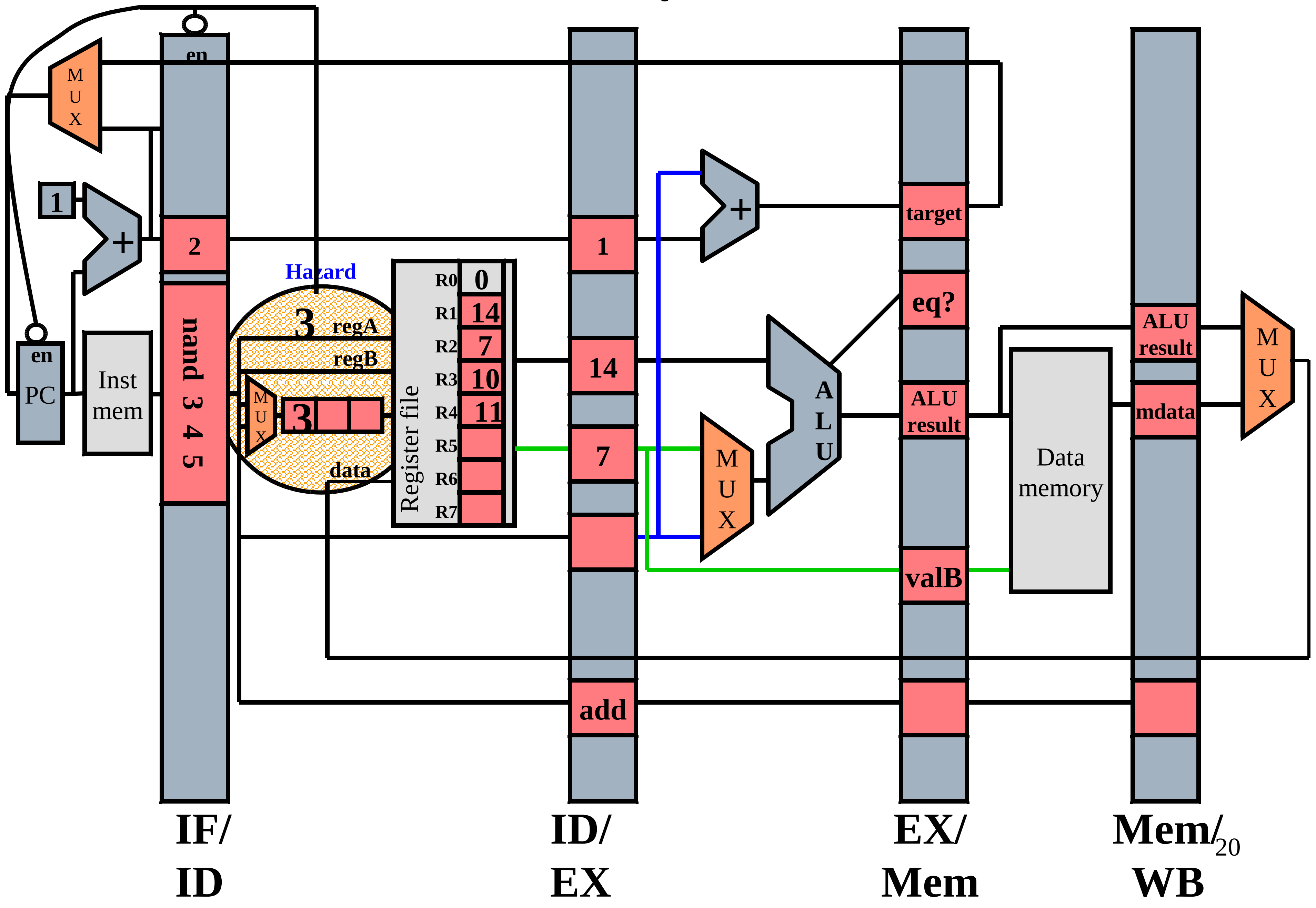
Handling data hazards II: Detect and stall until ready

- Detect:
  - Compare regA with previous DestReg
    - 3 bit operand fields
  - Compare regB with previous DestReg
    - 3 bit operand fields
- Stall:

  Keep current instructions in fetch and decode

  Pass a noop to execute

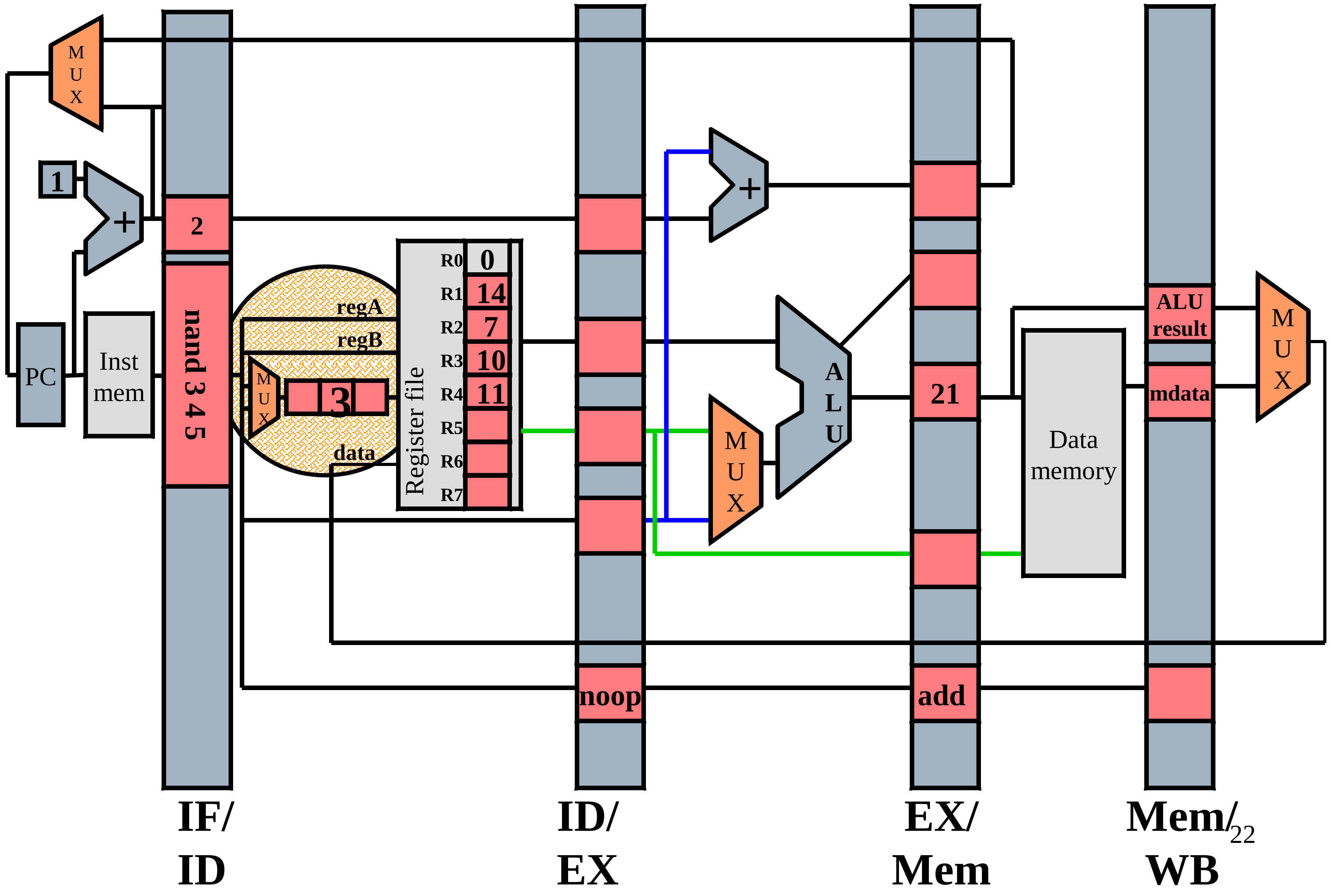
First half of cycle 3



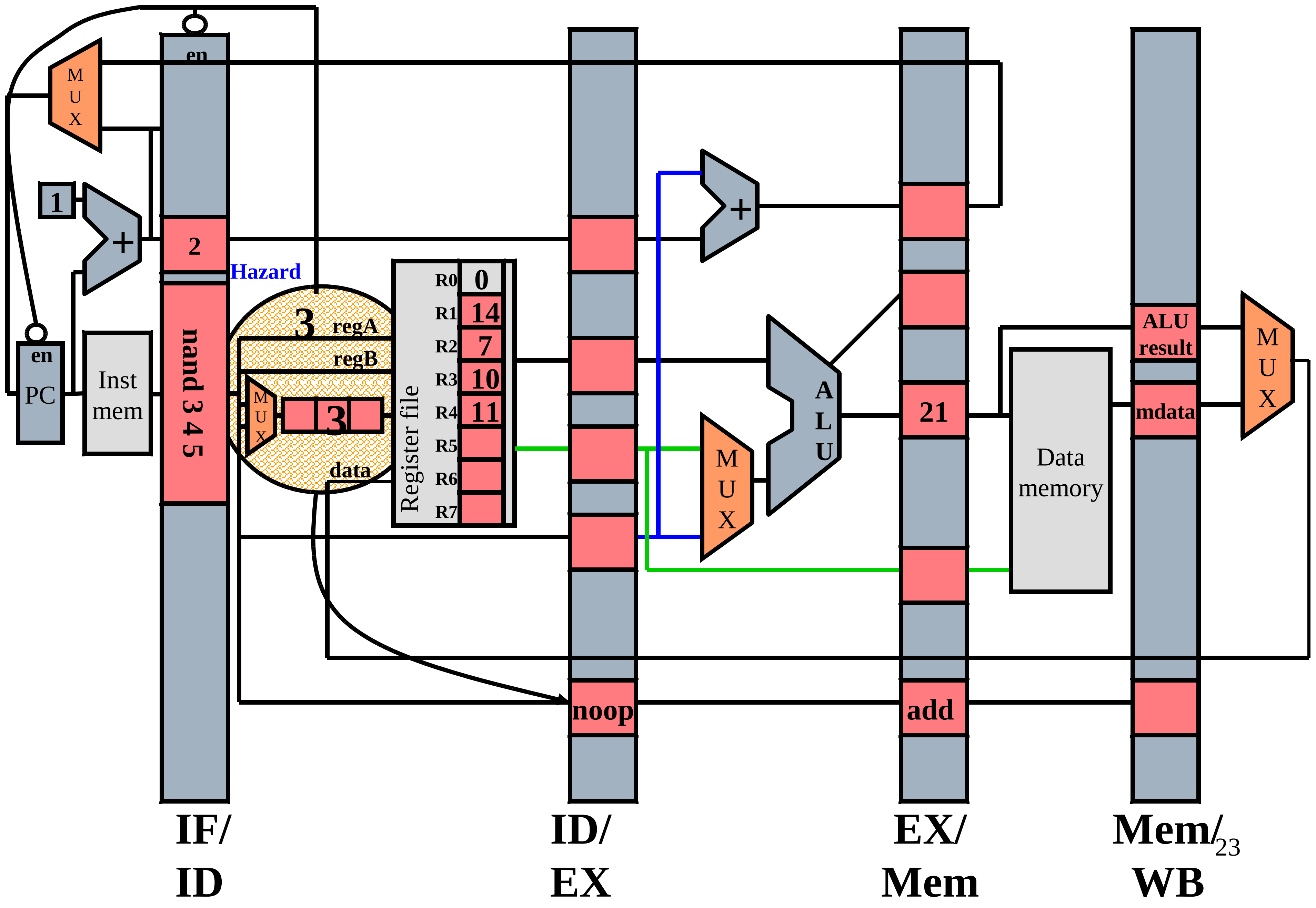
Handling data hazards II: Detect and stall until ready

- Detect:
  - Compare regA with previous DestReg
    - 3 bit operand fields
  - Compare regB with previous DestReg
    - 3 bit operand fields
- Stall:
  - Keep current instructions in fetch and decode
  - Pass a noop to execute

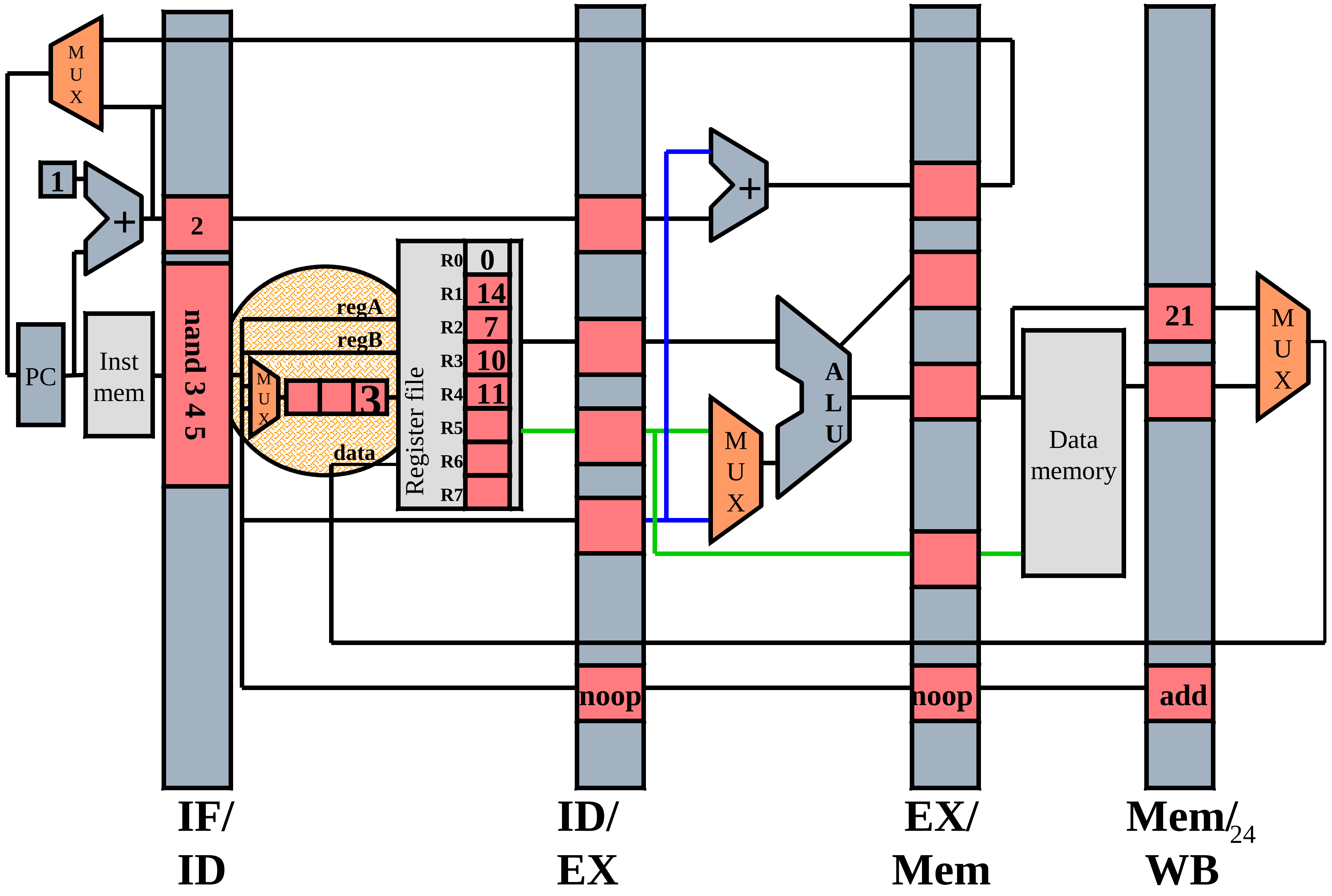
End of cycle 3



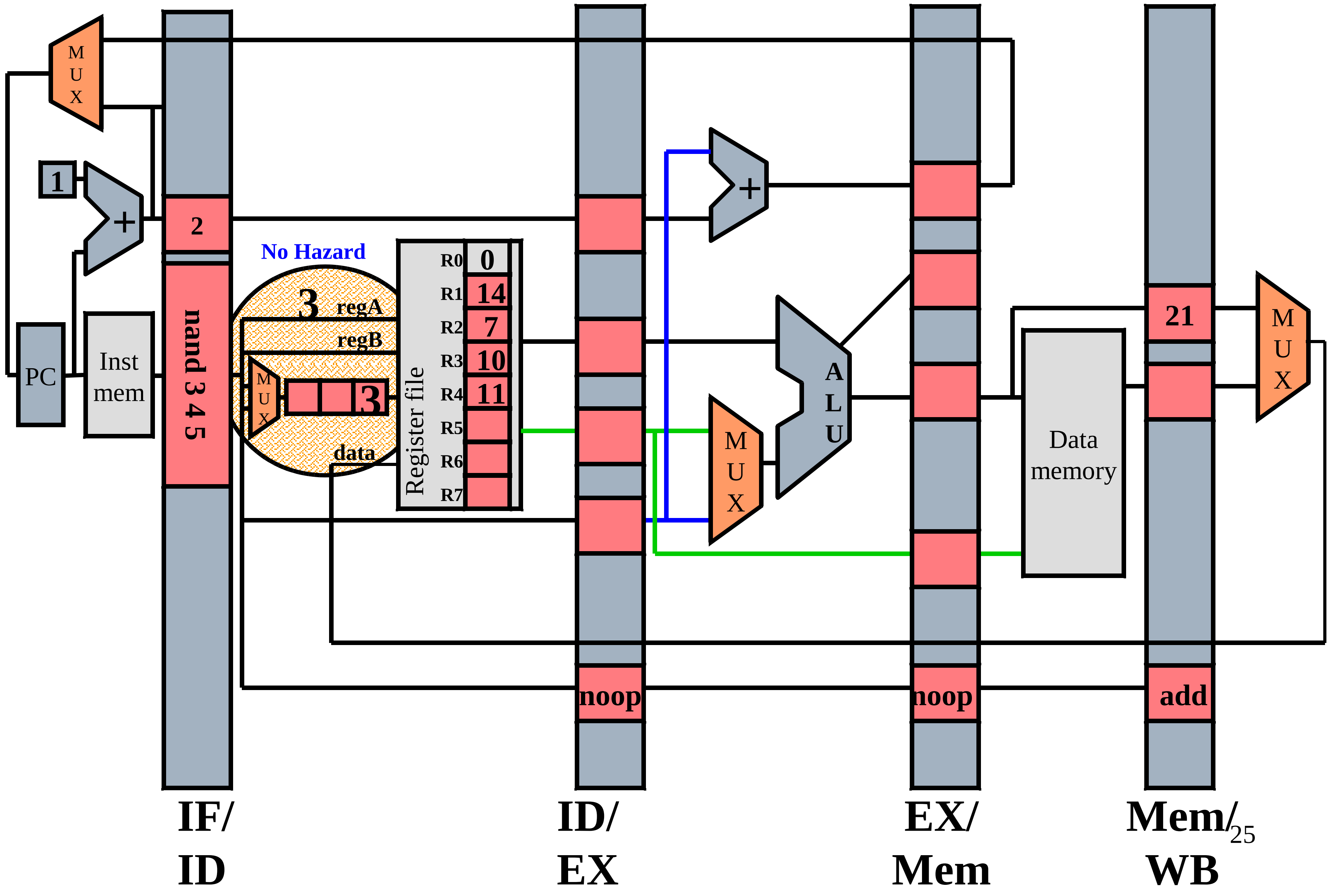
First half of cycle 4



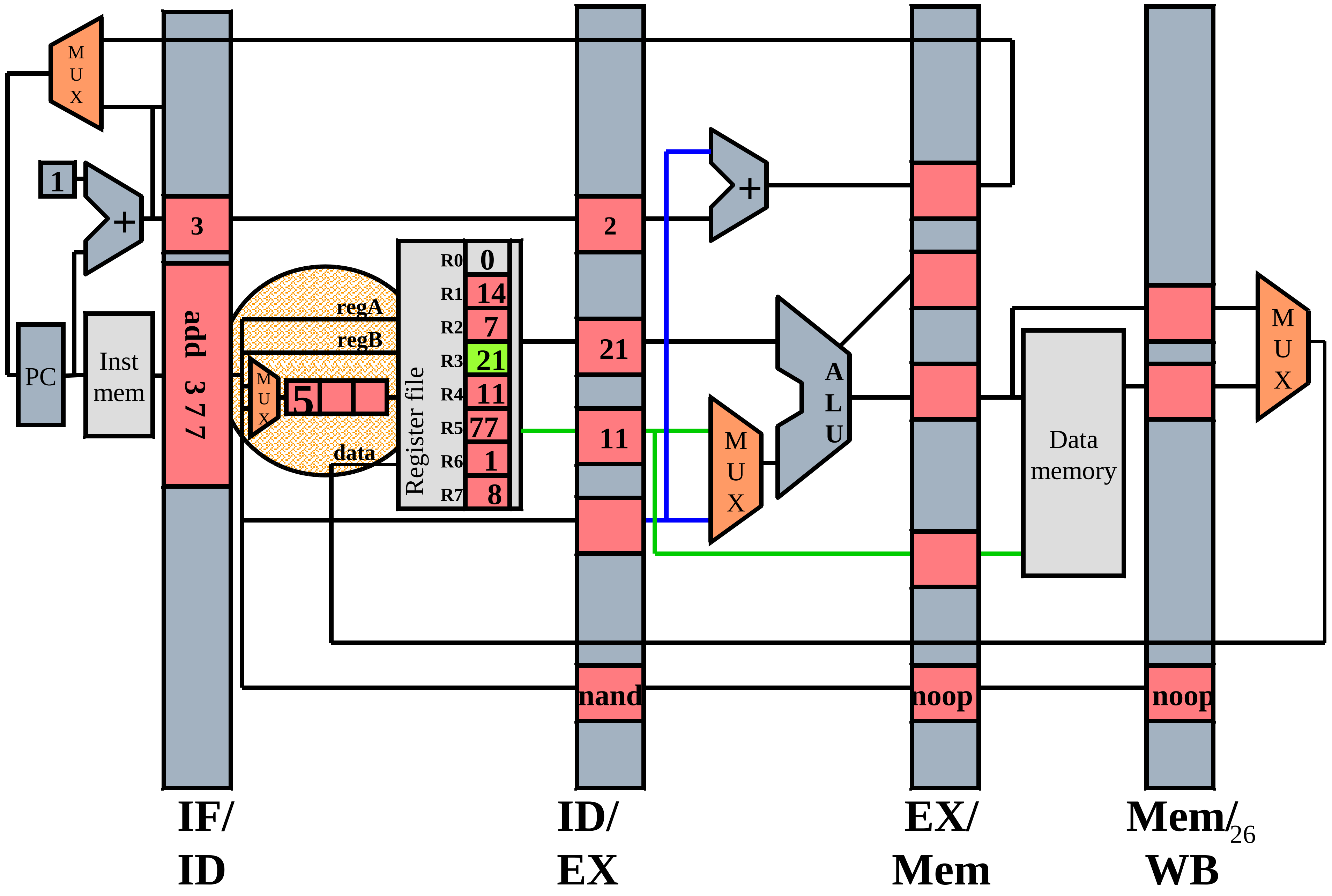
End of cycle 4



First half of cycle 5



End of cycle 5



Time Graph

Time:	1	2	3	4	5	6	7	8	9	10	11	12	13
add 1 2 3	IF	ID	EX	ME	WB								
nand 3 4 5		IF	no op	no op	ID	EX	ME	WB					
add 6 3 7					IF	ID	EX	ME	WB				
lw 3 6 10						IF	ID	EX	ME	WB			
sw 6 2 12							IF	no op	no op	ID	EX	ME	WB

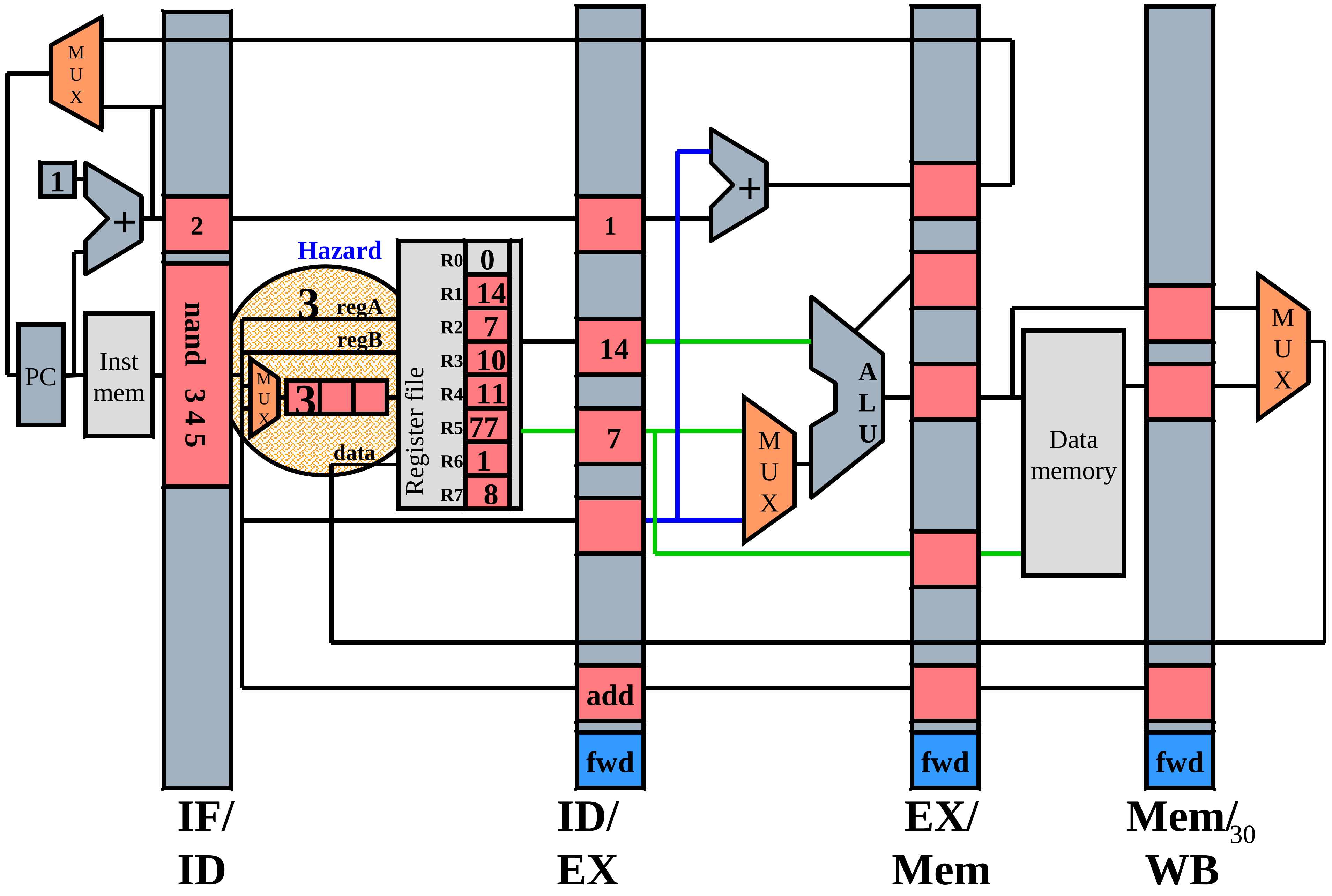
Problems with detect and stall

- CPI increases every time a hazard is detected!
- Is that necessary? Not always!
  - Re-route the result of the add to the nand
    - nand no longer needs to read R3 from reg file
    - It can get the data later (when it is ready)
    - This lets us complete the decode this cycle
      - But we need more control to remember that the data that we aren't getting from the reg file at this time will be found elsewhere in the pipeline at a later cycle.

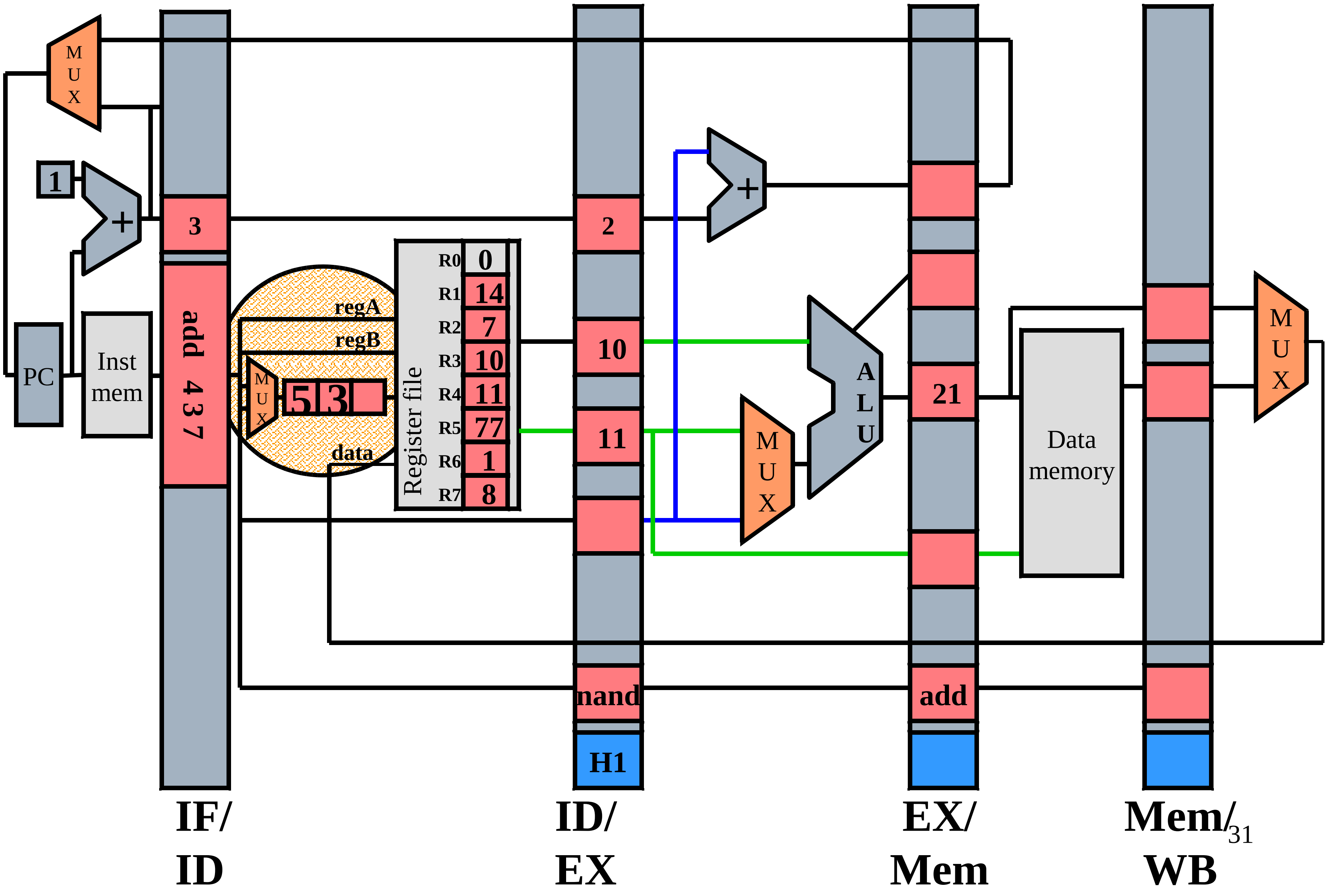
Handling data hazards III: Detect and forward

- Detect: same as detect and stall
  - Except that all 4 hazards are treated differently
    - I.e., you can't logical-OR the 4 hazard signals
- Forward:
  - New bypass datapaths route computed data to where it is needed
  - New MUX and control to pick the right data
- Beware: Stalling may still be required even in the presence of forwarding

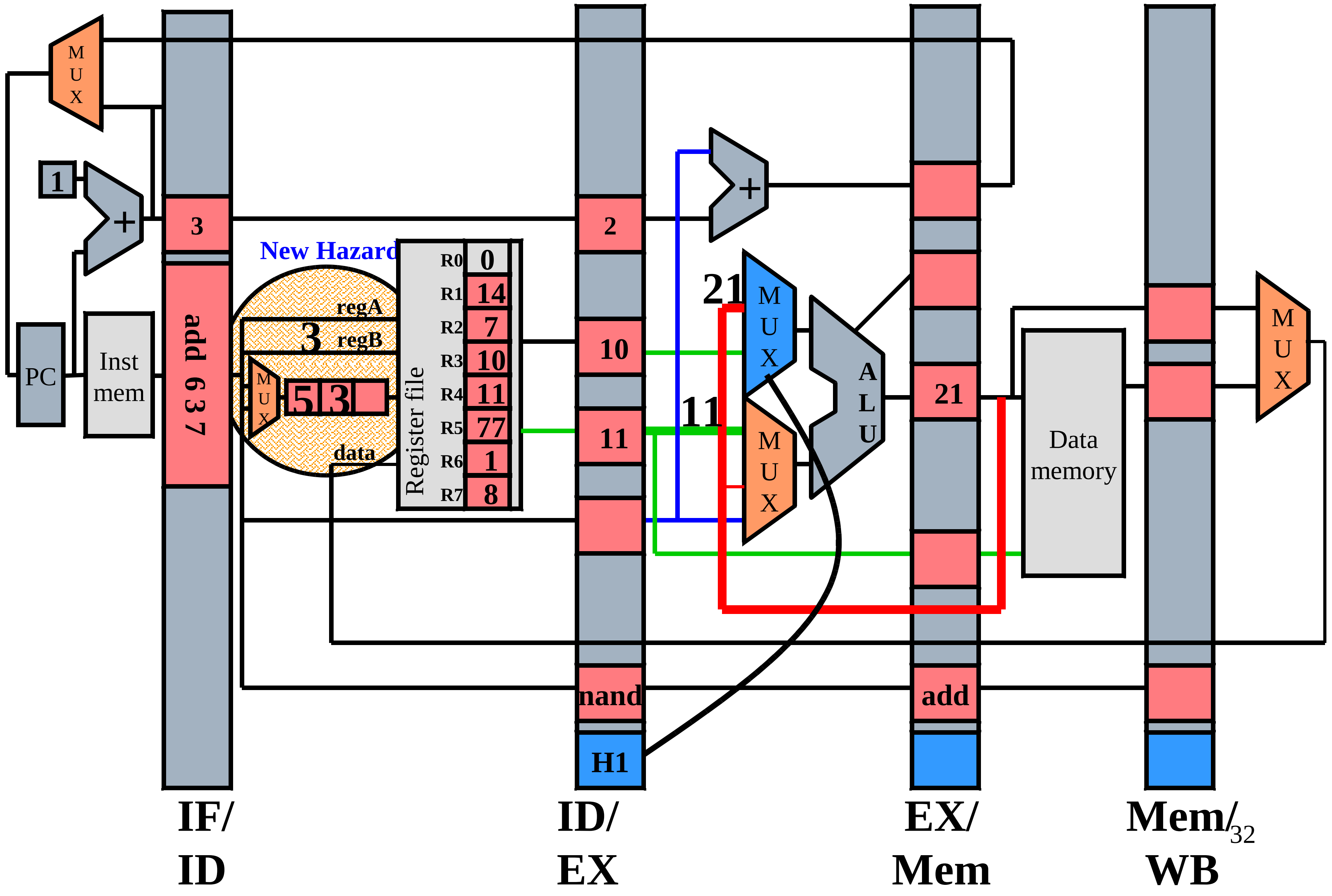
First half of cycle 3



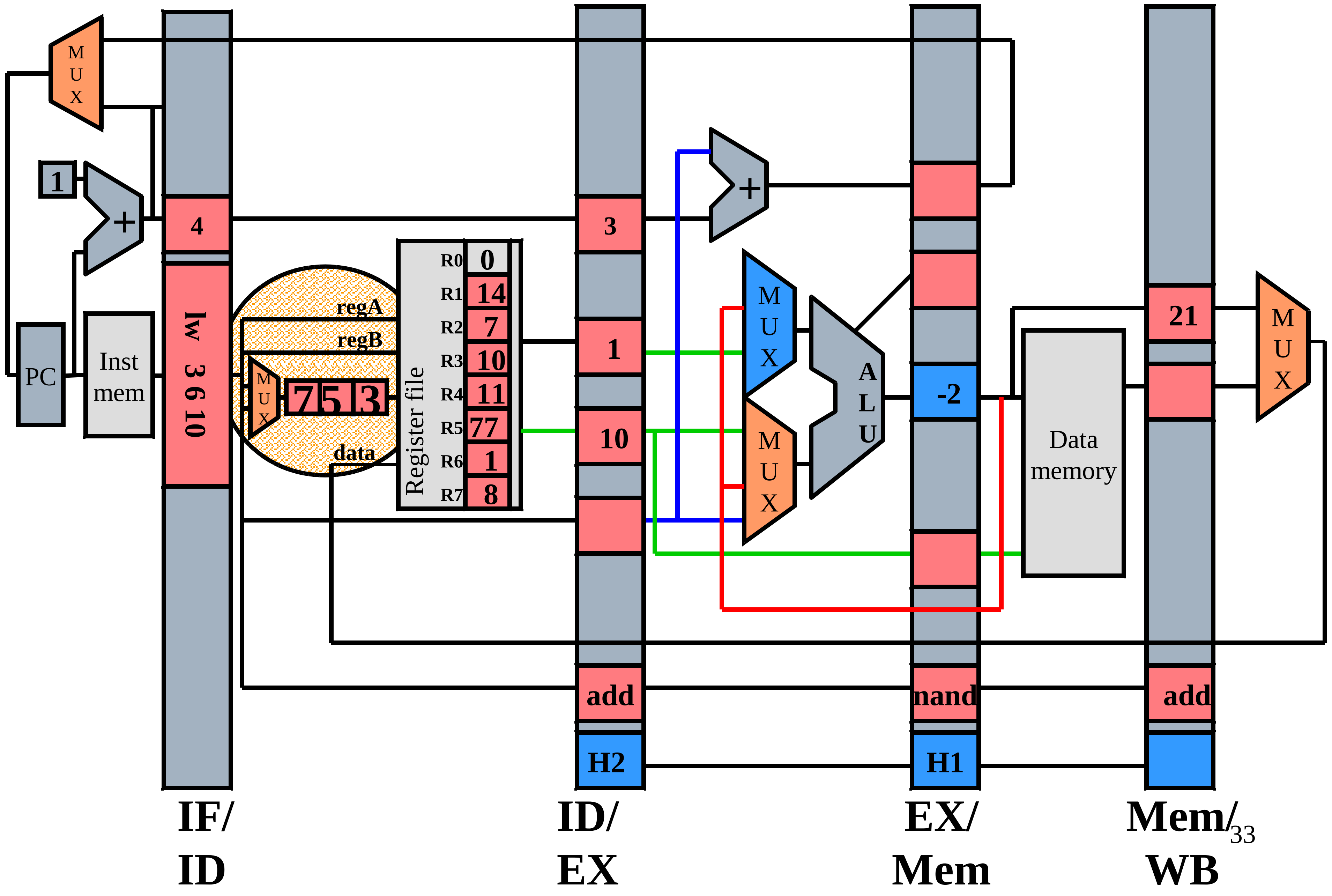
End of cycle 3



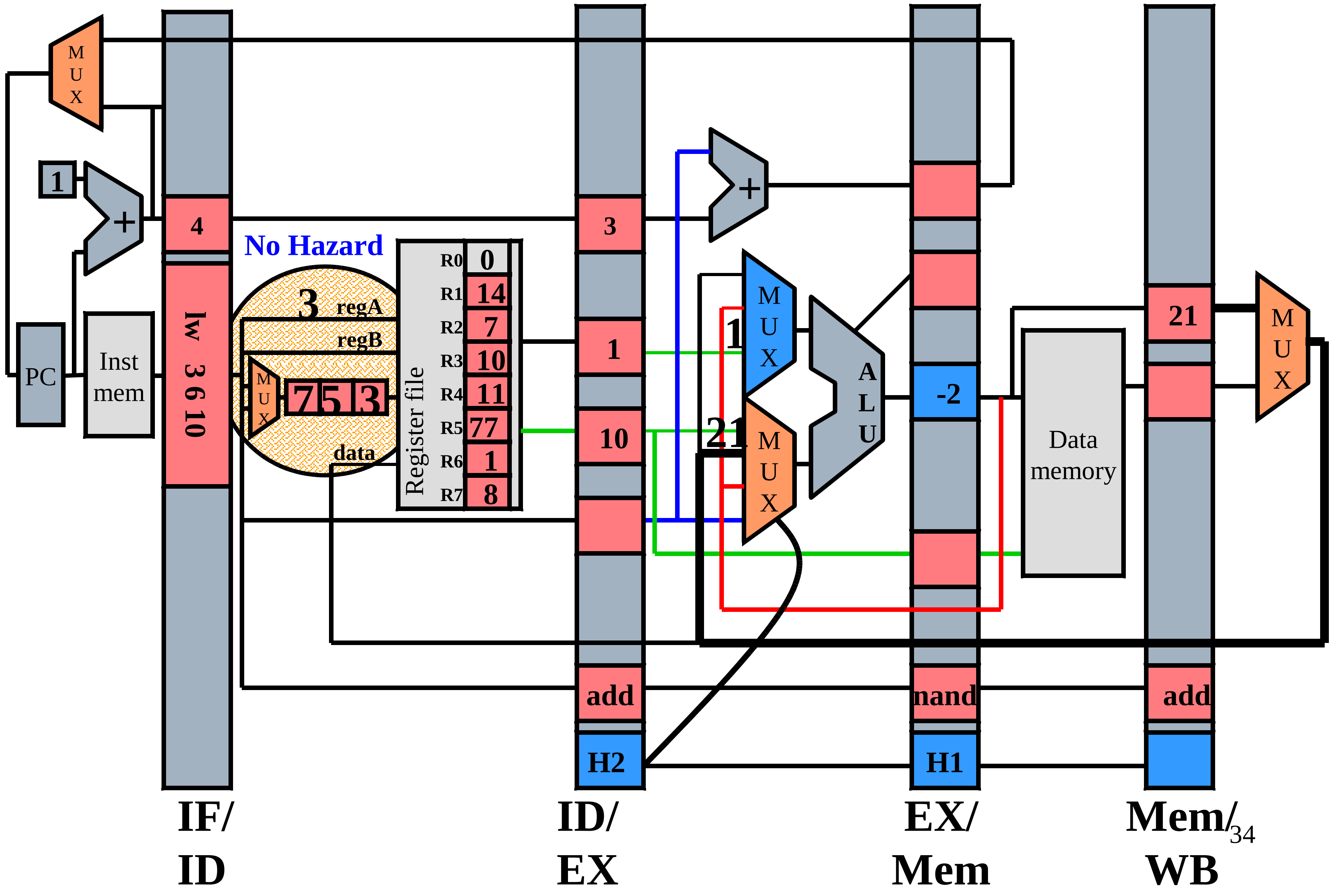
First half of cycle 4



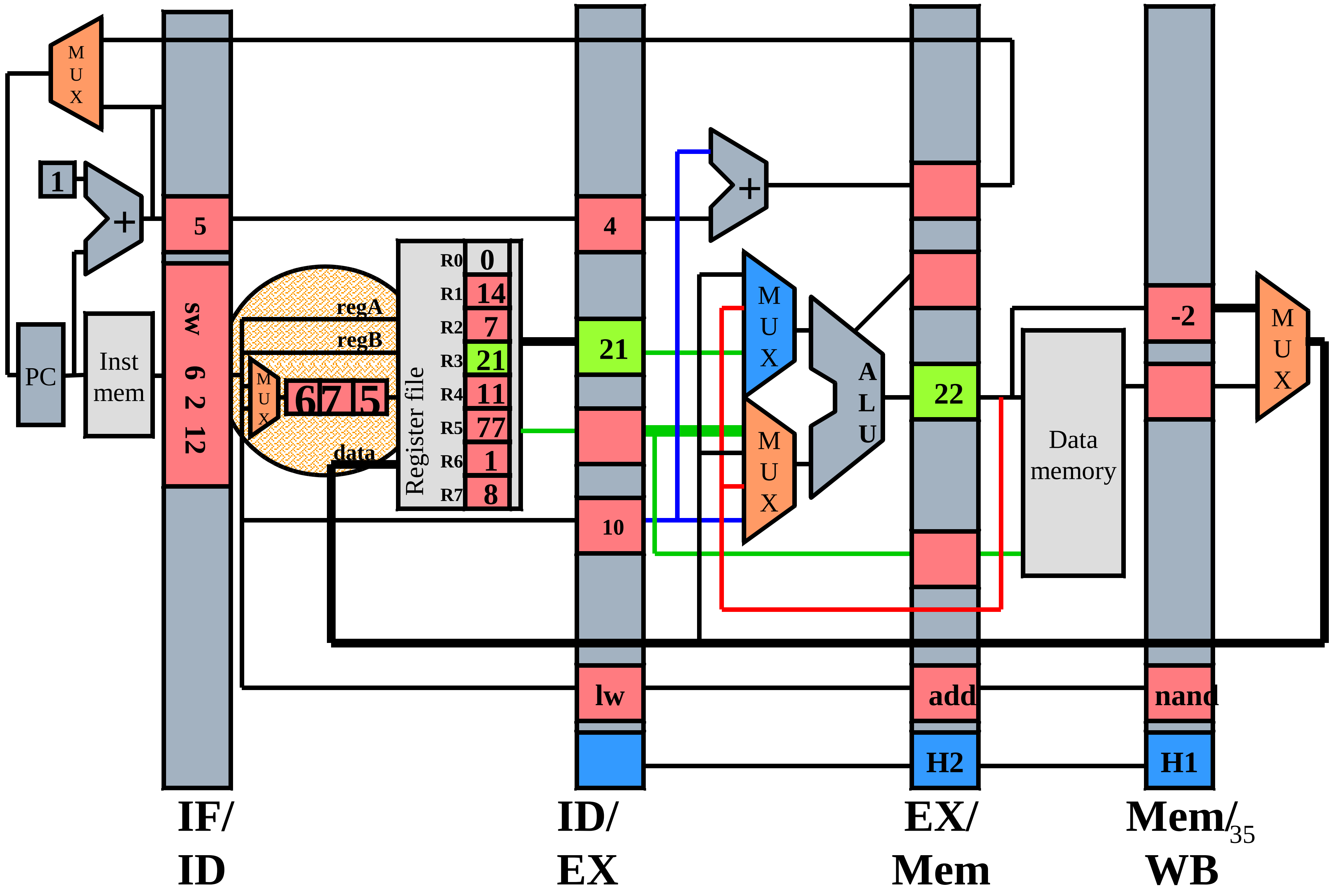
End of cycle 4



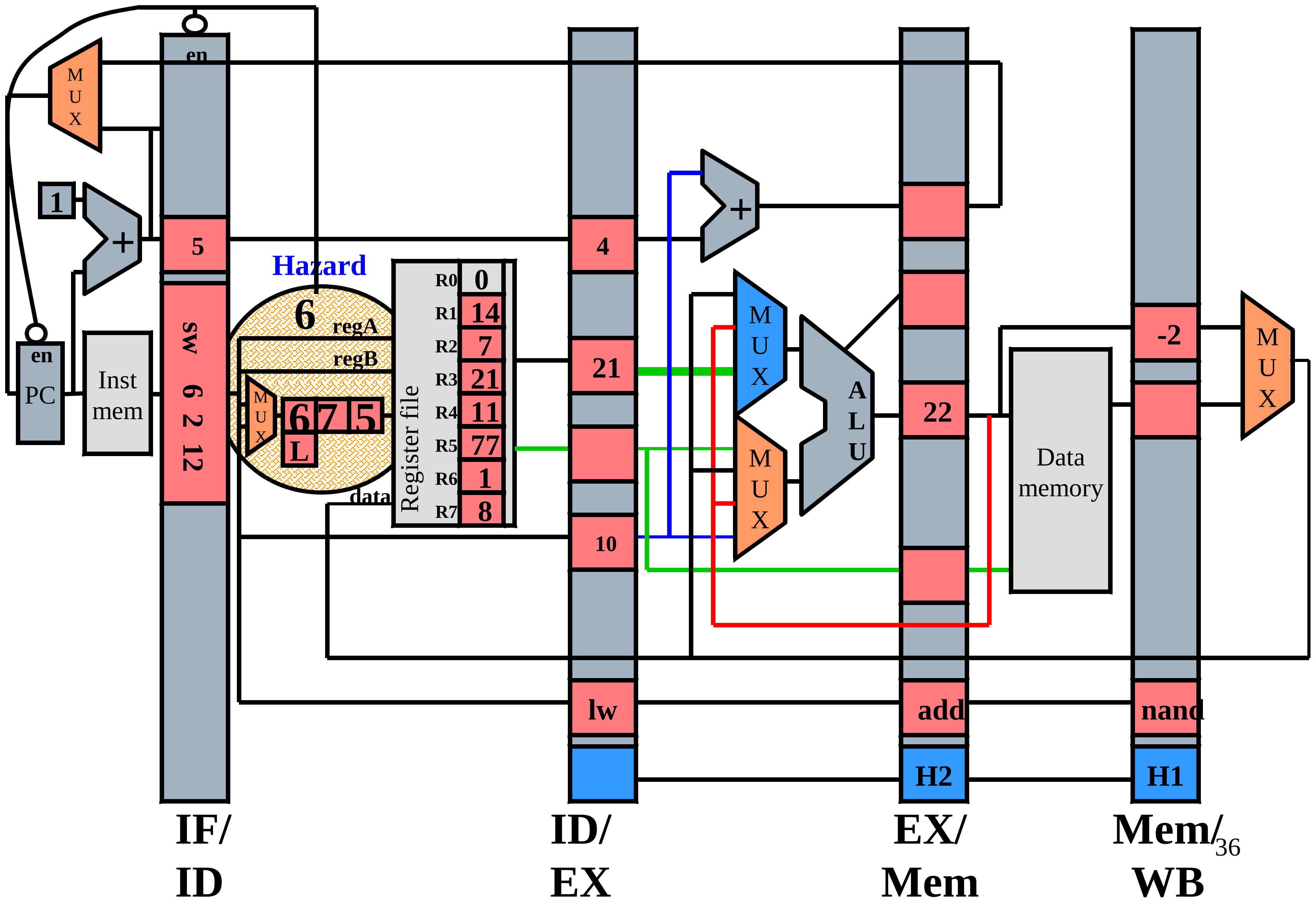
First half of cycle 5



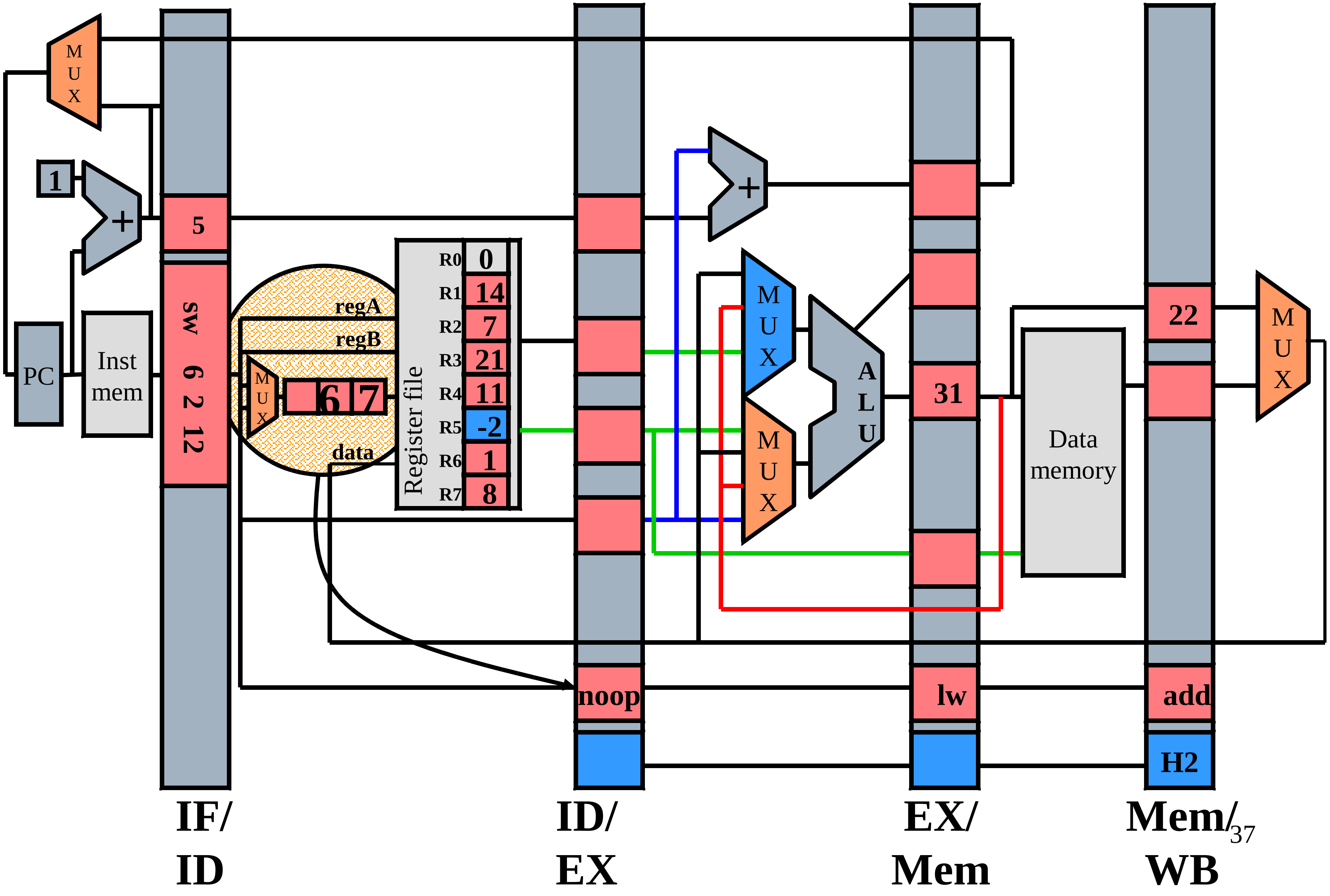
End of cycle 5



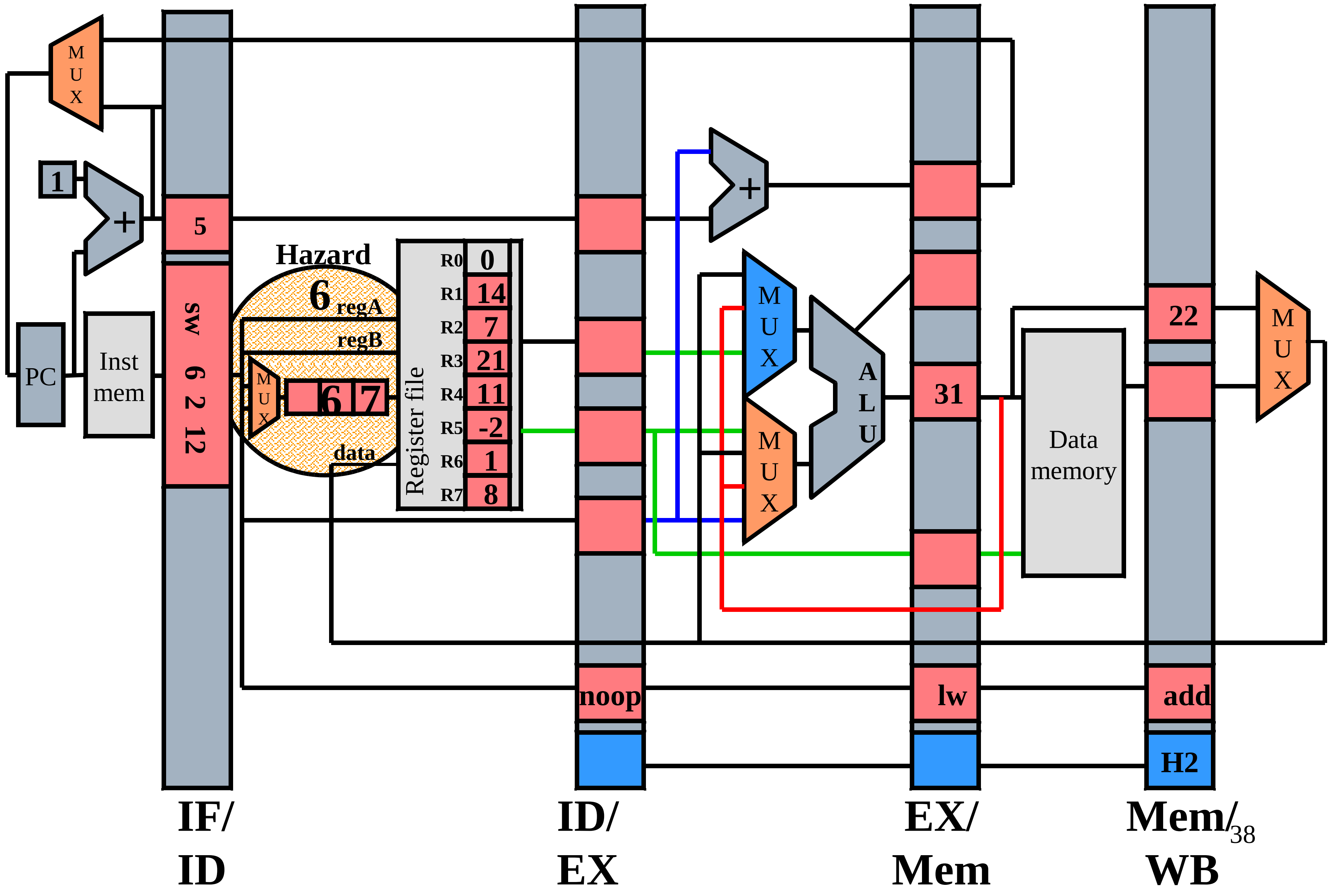
First half of cycle 6



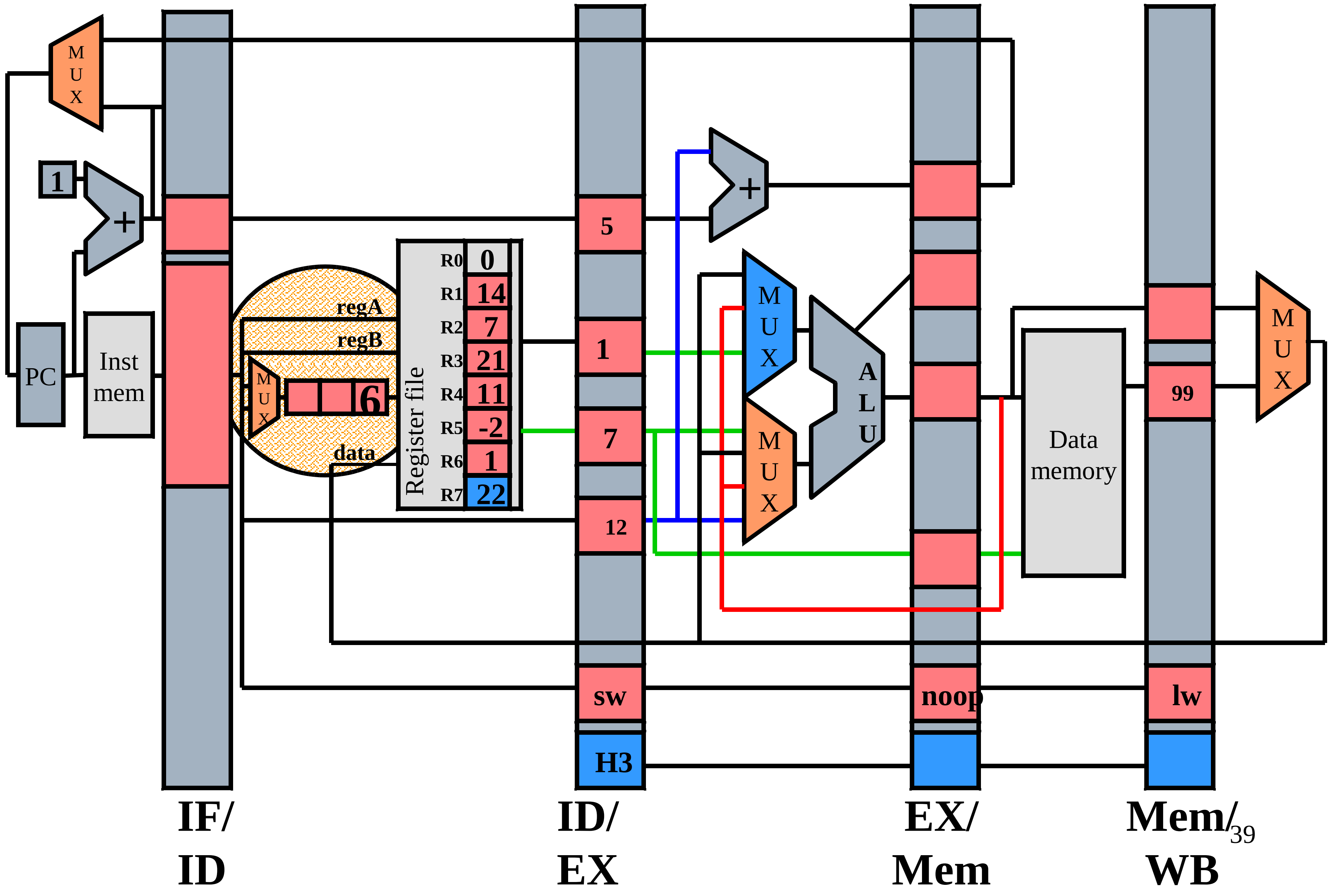
End of cycle 6



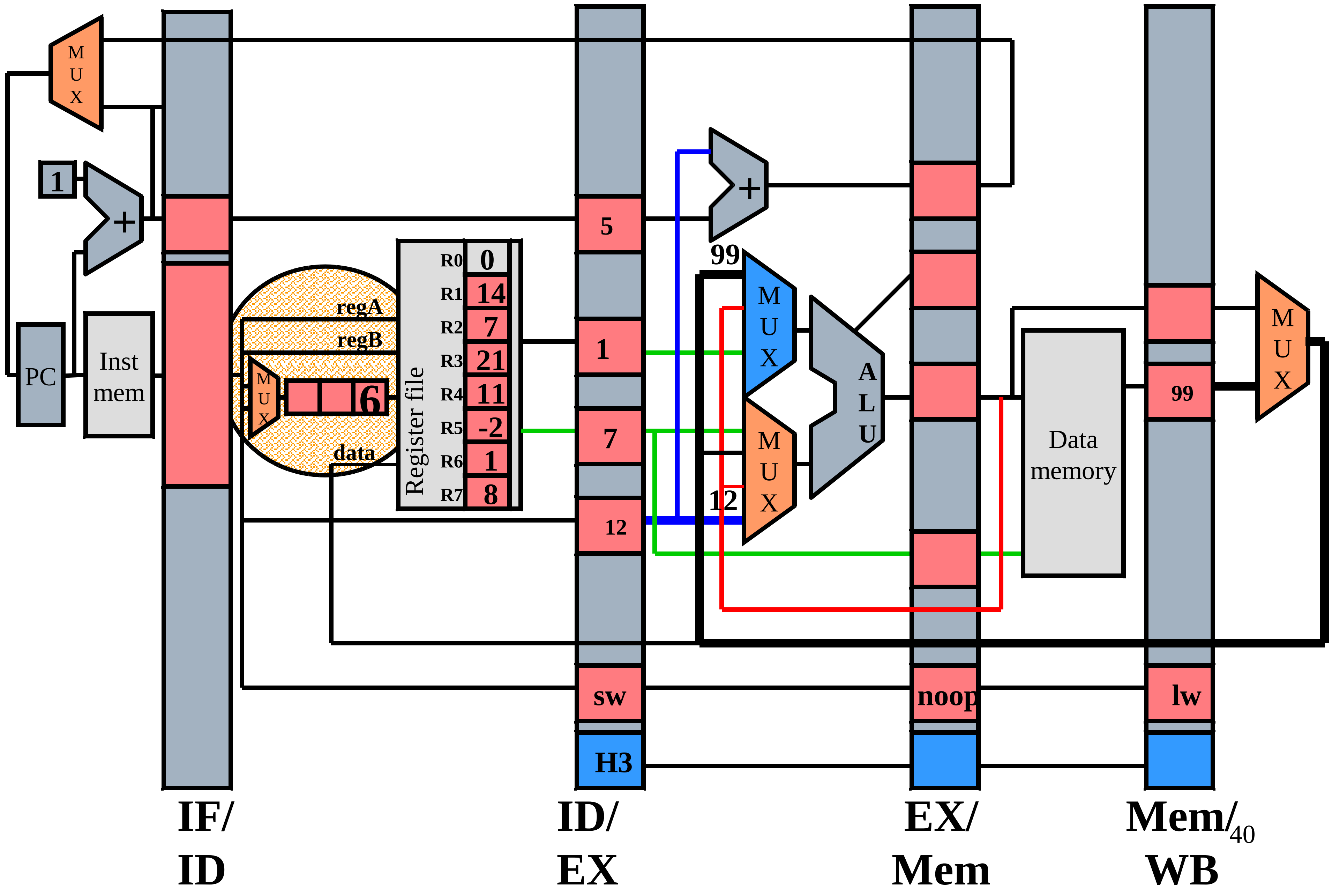
First half of cycle 7



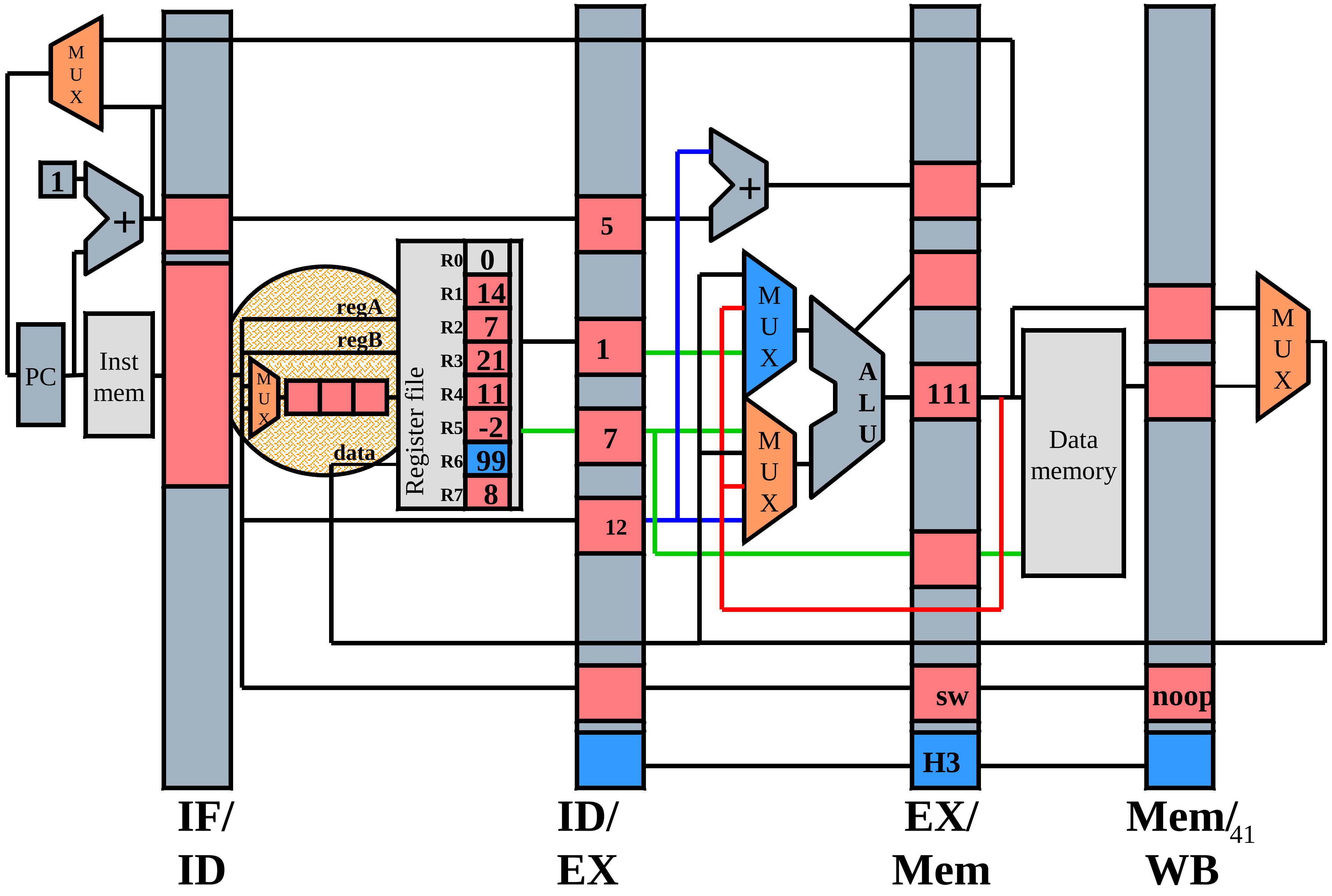
End of cycle 7



First half of cycle 8



End of cycle 8



Time Graph

Time:	1	2	3	4	5	6	7	8	9	10	11	12	13
add 1 2 3	IF	ID	EX	ME	WB								
nand 3 4 5		IF	ID	EX	ME	WB							
add 6 3 7			IF	ID	EX	ME	WB						
lw 3 6 10				IF	ID	EX	ME	WB					
sw 6 2 12					IF	no op	ID	EX	ME	WB			

Class Problem 2

```
add  1  2  3
lw   3  4  1
lw   4  5  6
add 6   1  7
sw   5  2  12
```

How many cycles to execute this code using detect and stall?

How many cycles to execute this code using detect and forward?

Next time

- Control hazards