L13_1 Pipelining_Execution-Example

EECS 370 – Introduction to Computer Organization – Fall 2020

Learning Objectives

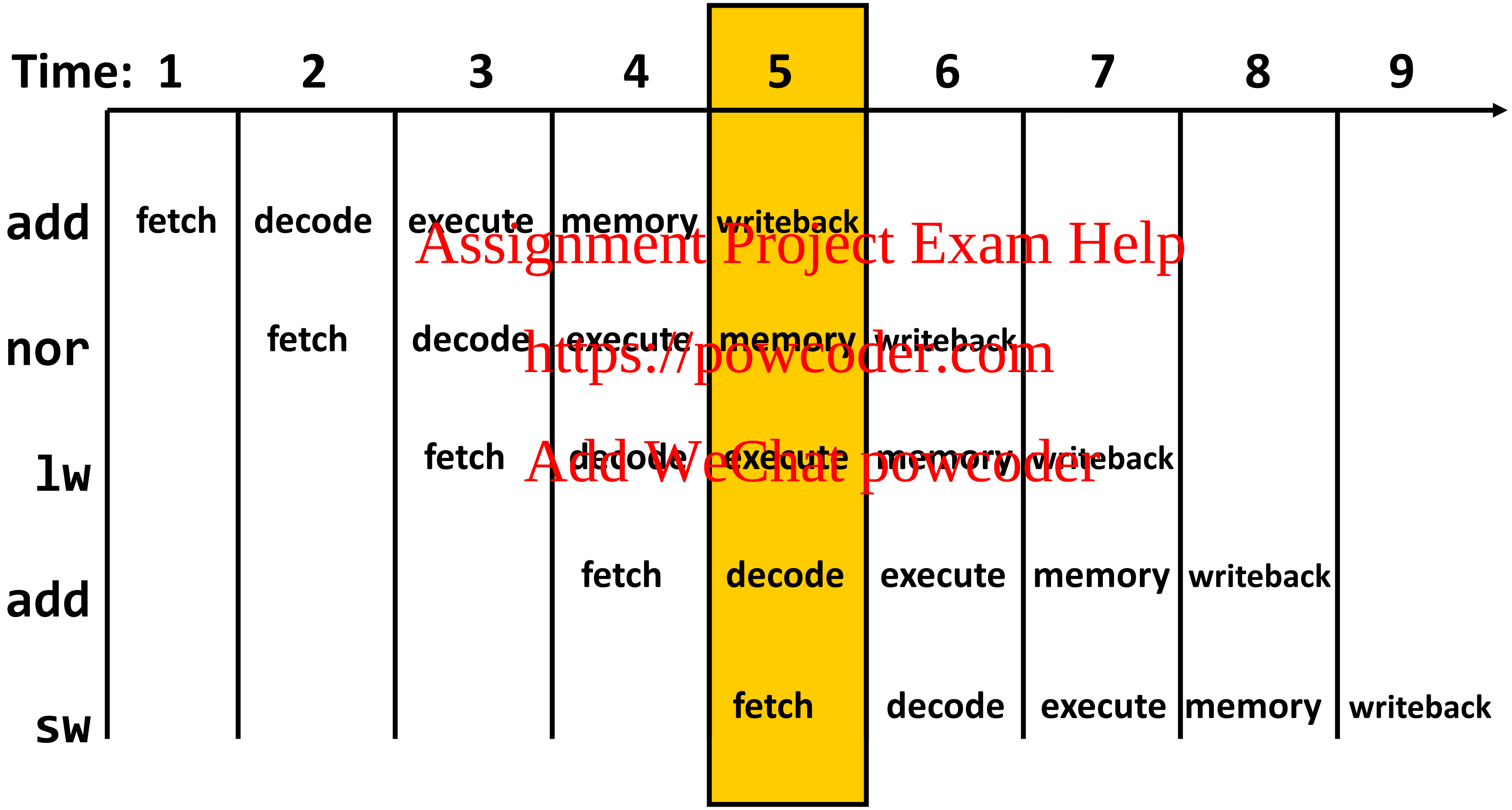
- Ability to trace the execution of instructions through the pipeline datapath implementation for LC2K.
- Understand the flow of data through the datapath and between pipeline stages.

Sample Code (Simple)

Let us run the following code on pipelined LC2K2x:

```
•    add  1    2    3   ;  reg3 = reg1 + reg2
•    nor  4    5    6   ;  reg6 = reg4 nor reg5
•    lw   2    4    20  ;  reg4 = Mem[reg2 + 20]
•    add  2    5    5   ;  reg5 = reg2 + reg5
•    sw   3    7    10  ;  Mem[reg3 + 10] = reg7
```

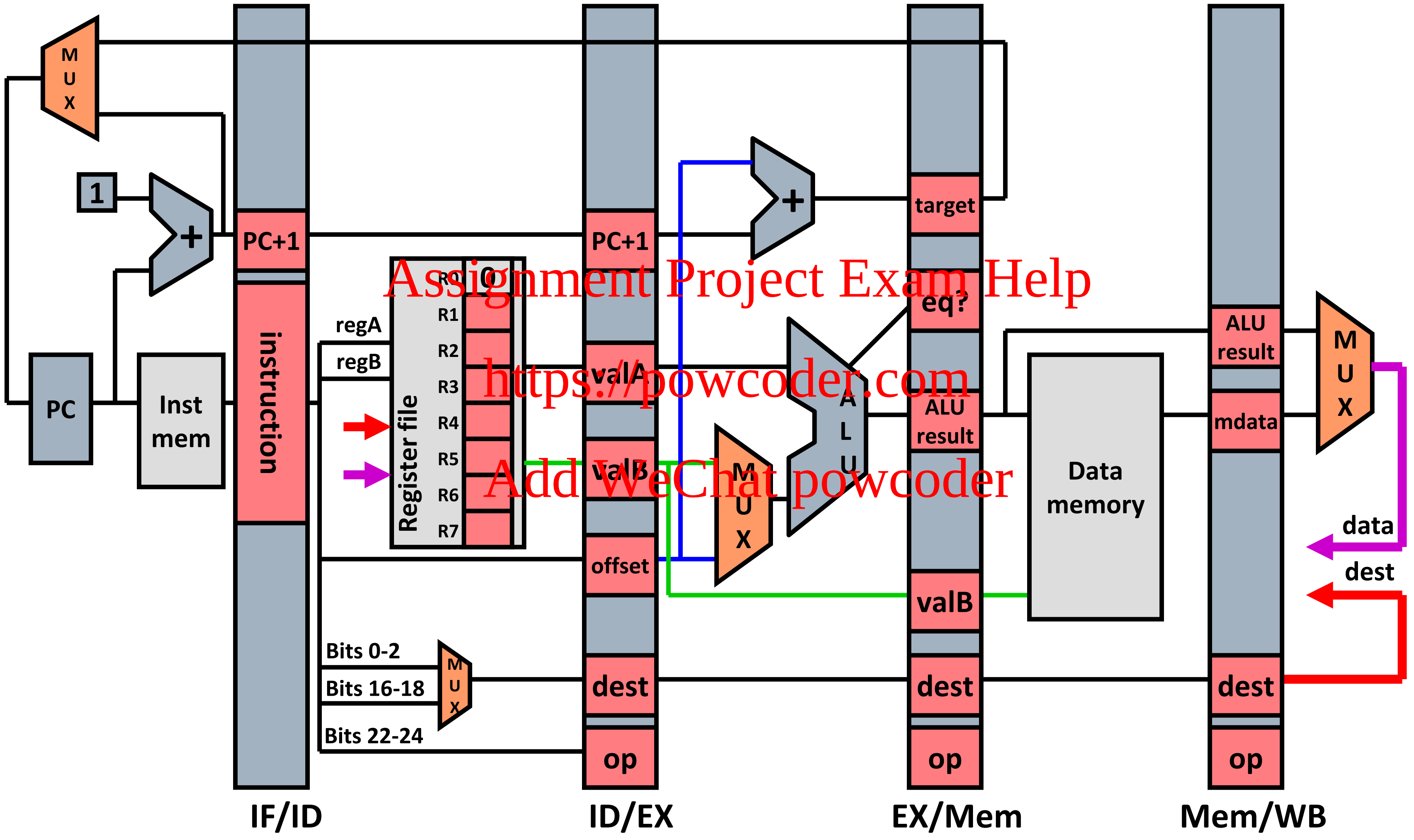
Time Graphs (a.k.a. Pipe Trace)



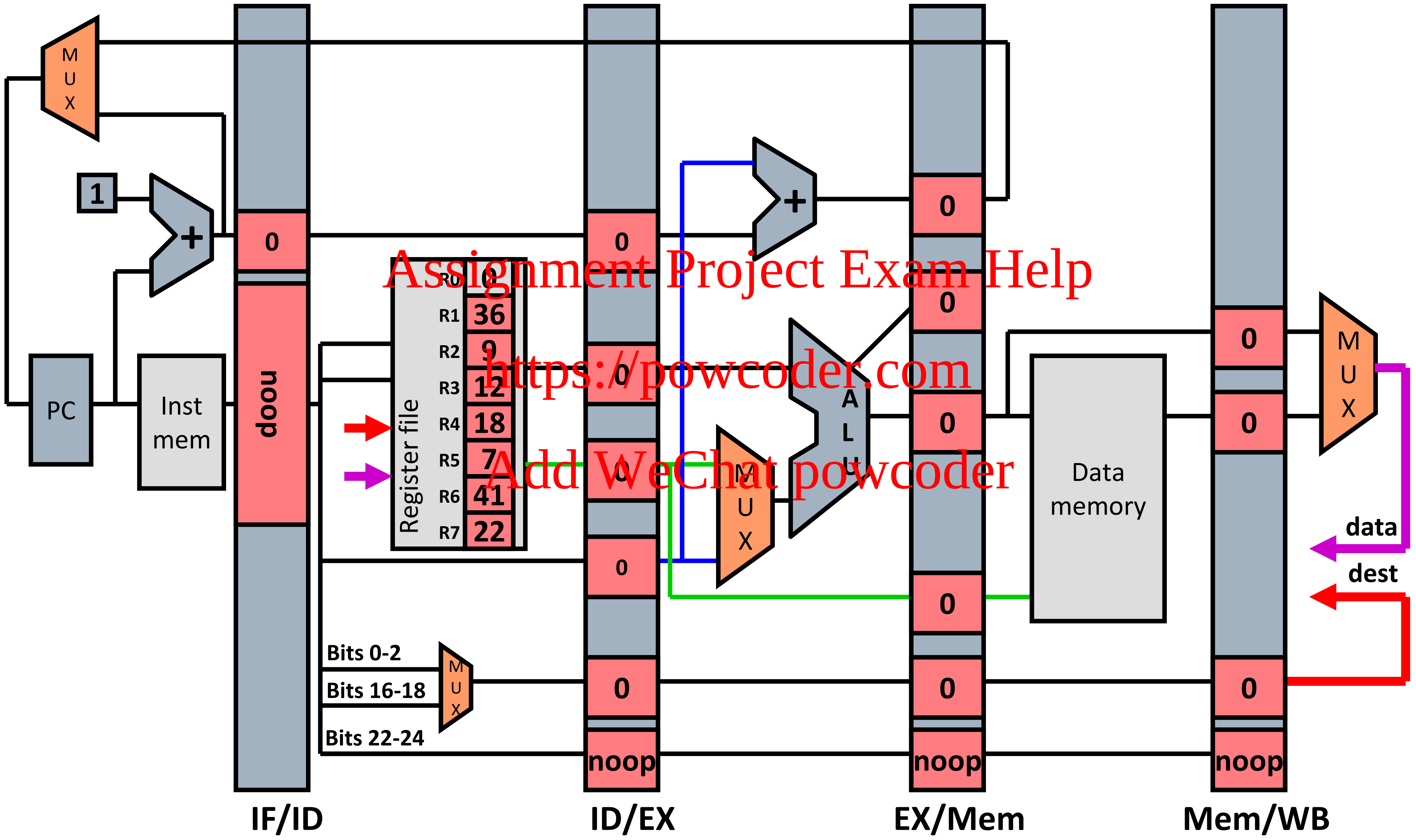
Time:	1	2	3	4	5	6	7	8	9
add	fetch	decode	execute	memory	writeback				
nor		fetch	decode	execute	memory	writeback			
lw			fetch	decode	execute	memory	writeback		
add				fetch	decode	execute	memory	writeback	
sw					fetch	decode	execute	memory	writeback

A vertical slice reports the entire activity of the pipeline at time 5

Pipeline Datapath

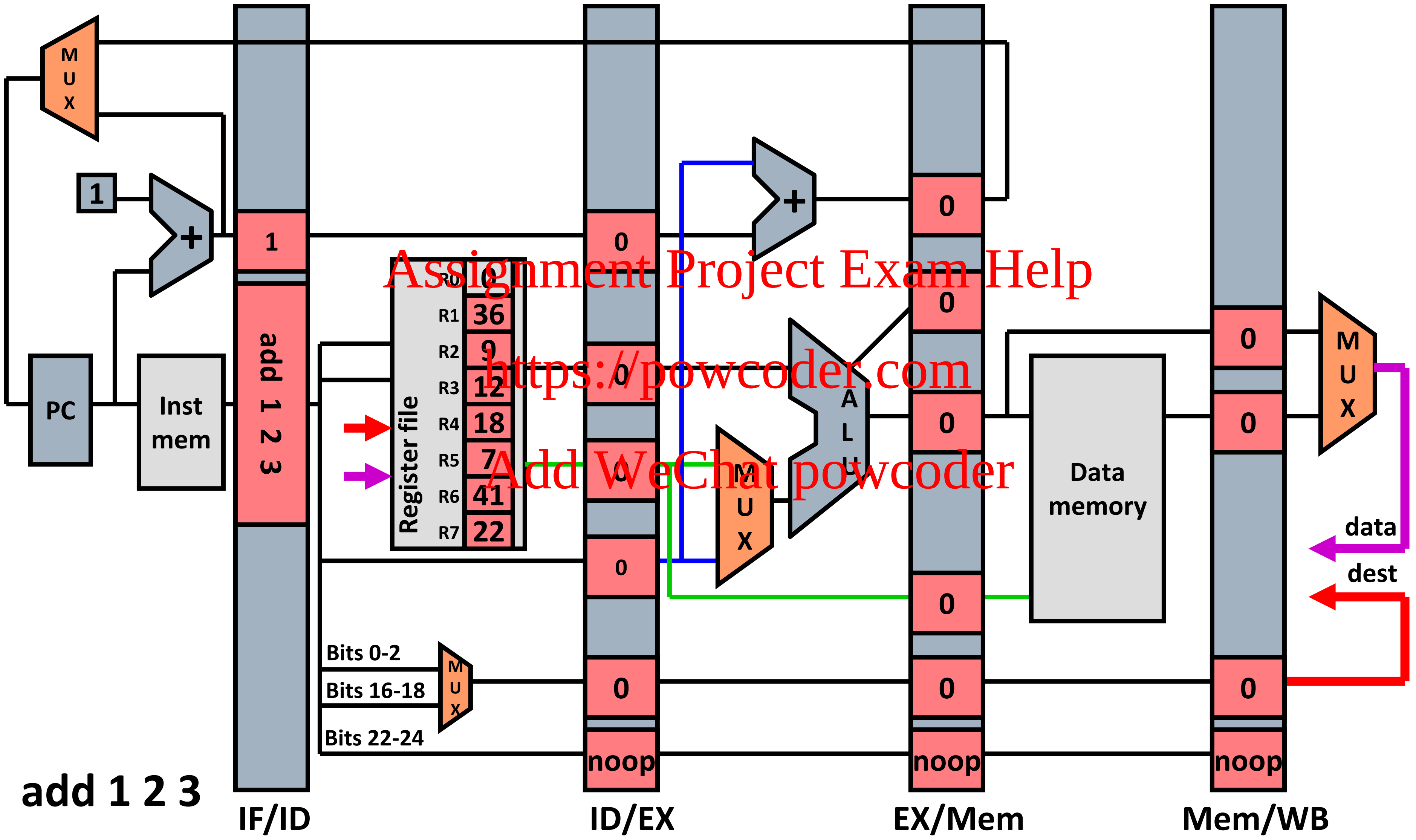


Time 0 - Initial State

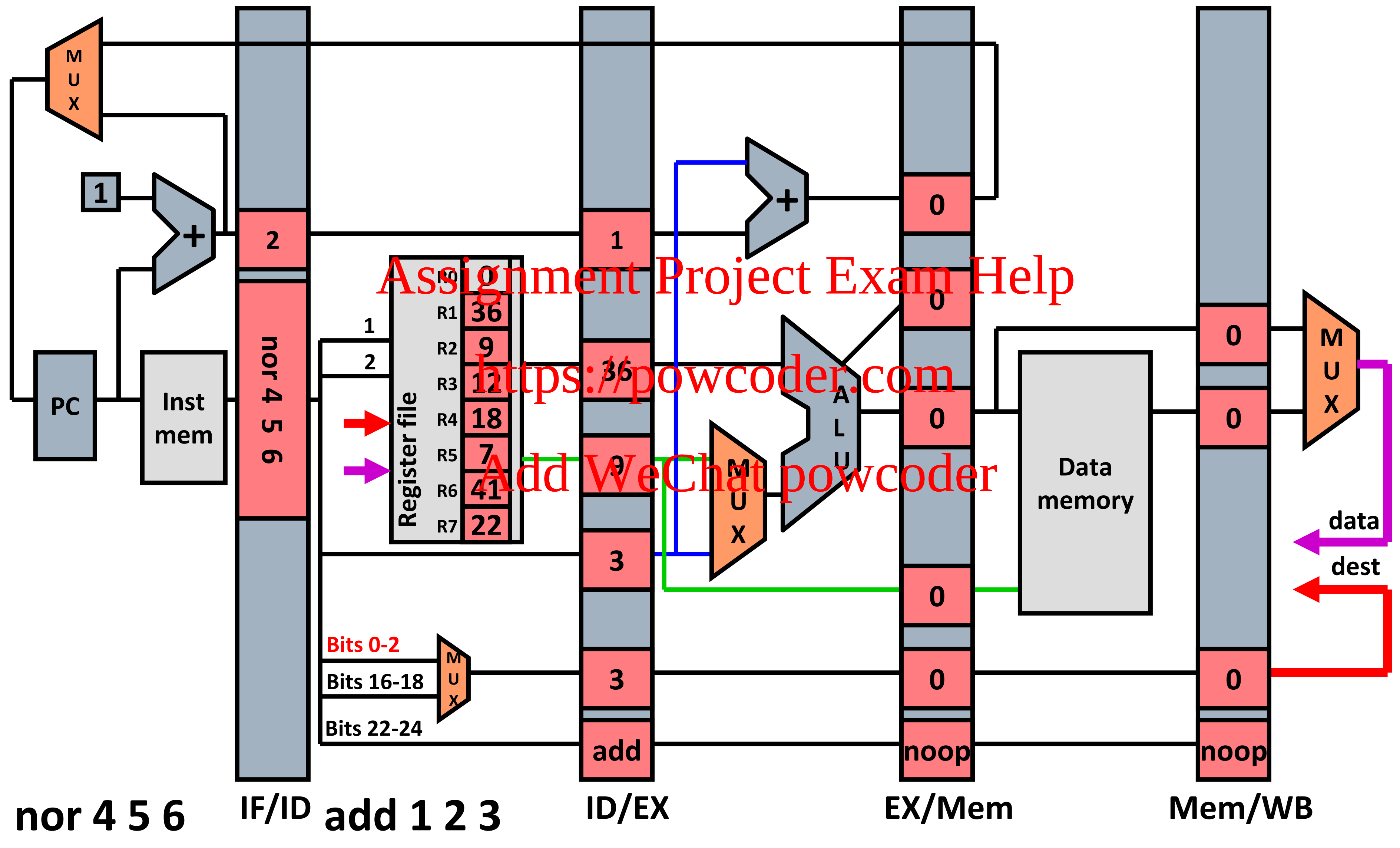


Pipelining

Time 1 - Fetch: **add 1 2 3**

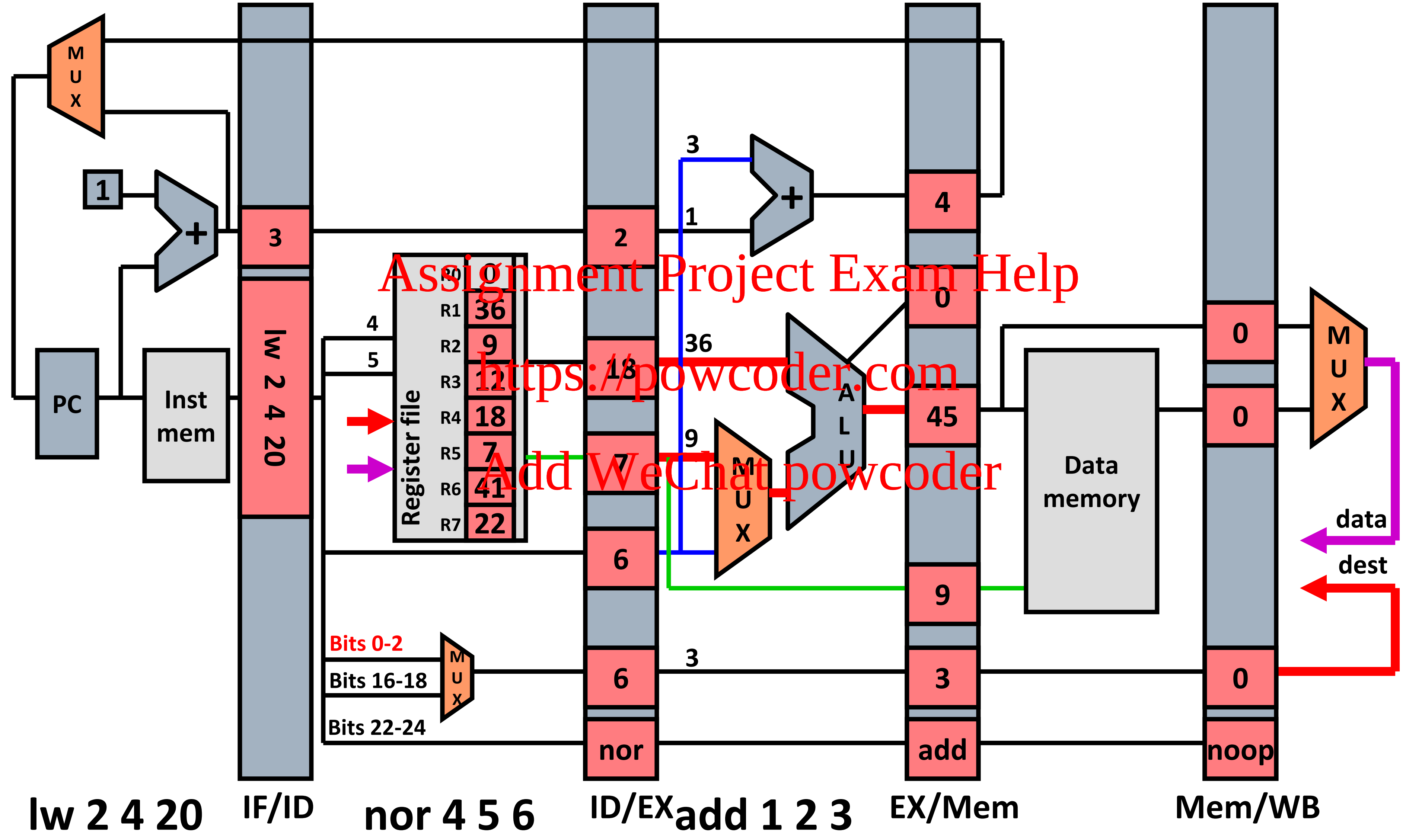


Time 2 - Fetch: **nor 4 5 6**



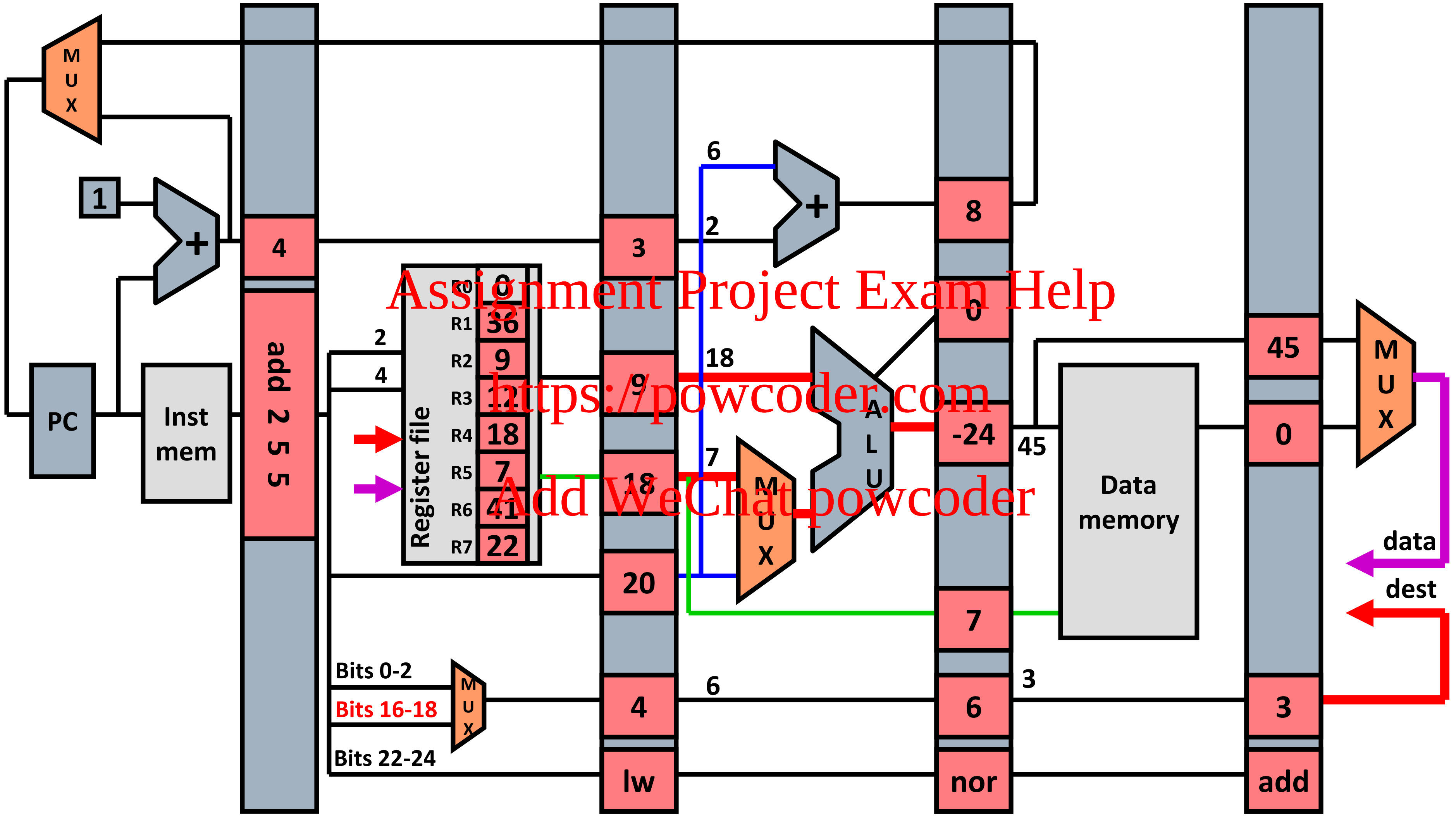
Time 3 - Fetch: `lw 2 4 20`

Pipelining



Time 4 - Fetch: **add 2 5 5**

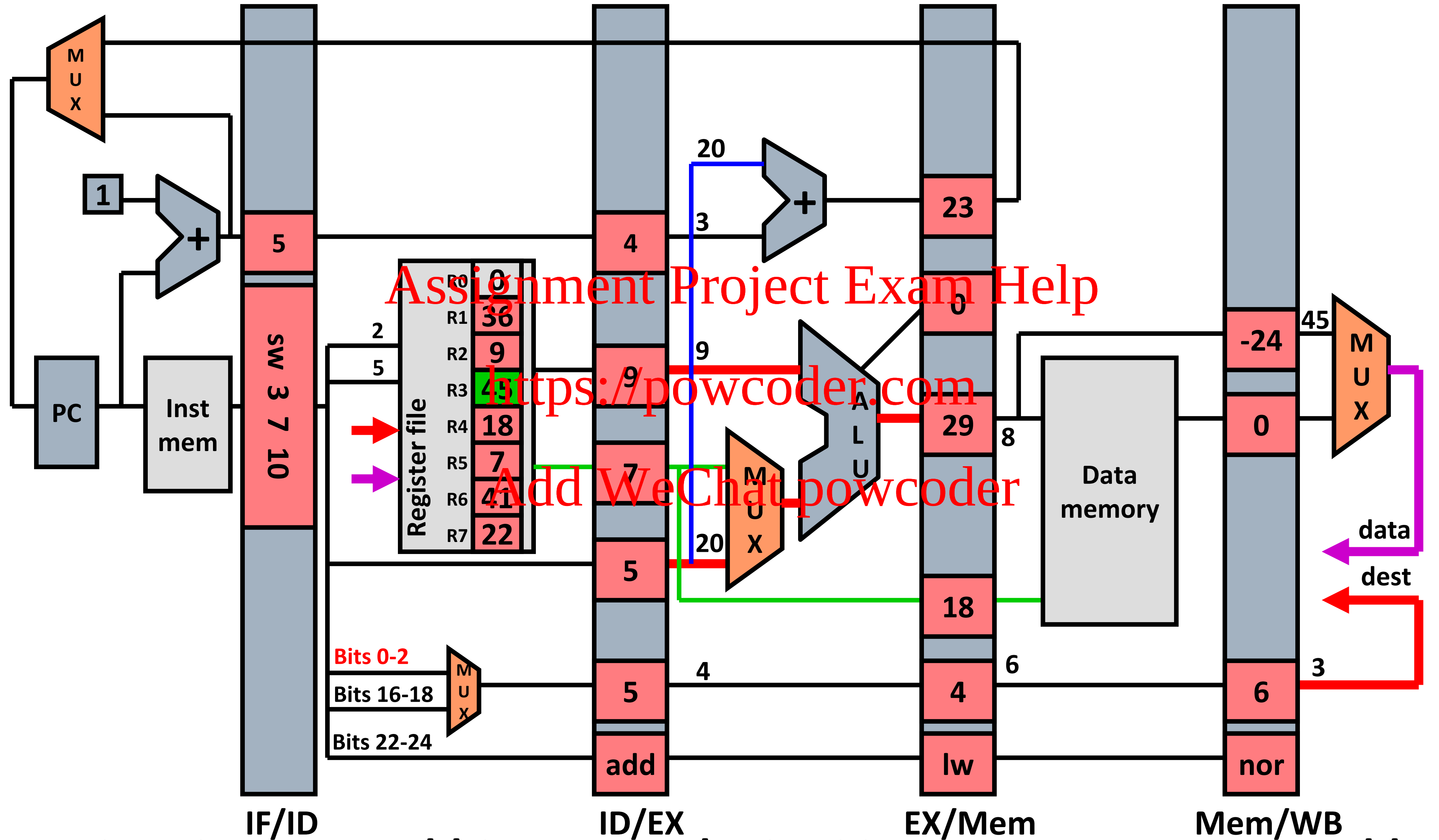
Pipelining



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

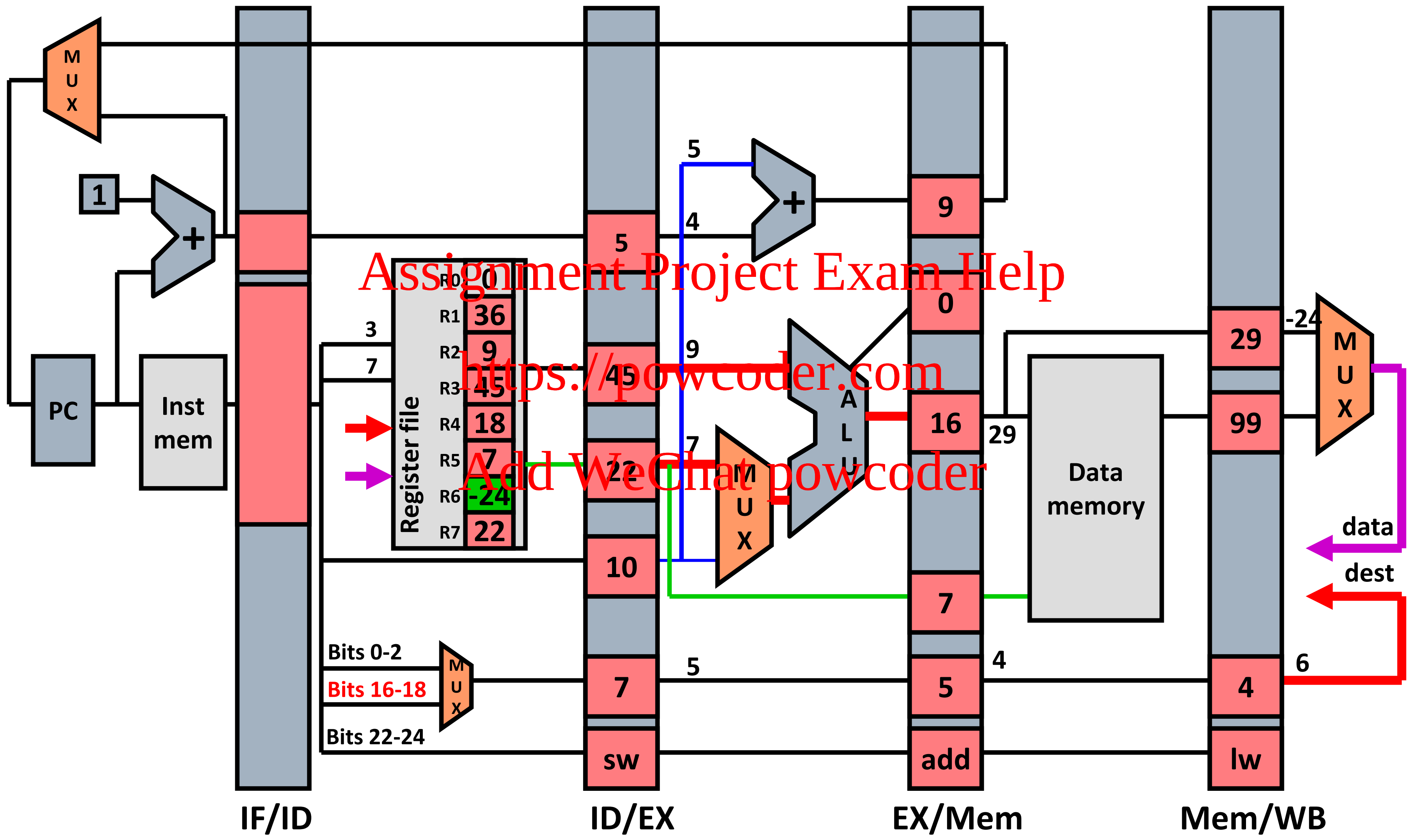
Time 5 - Fetch: `sw 3 7 10`

Pipelining



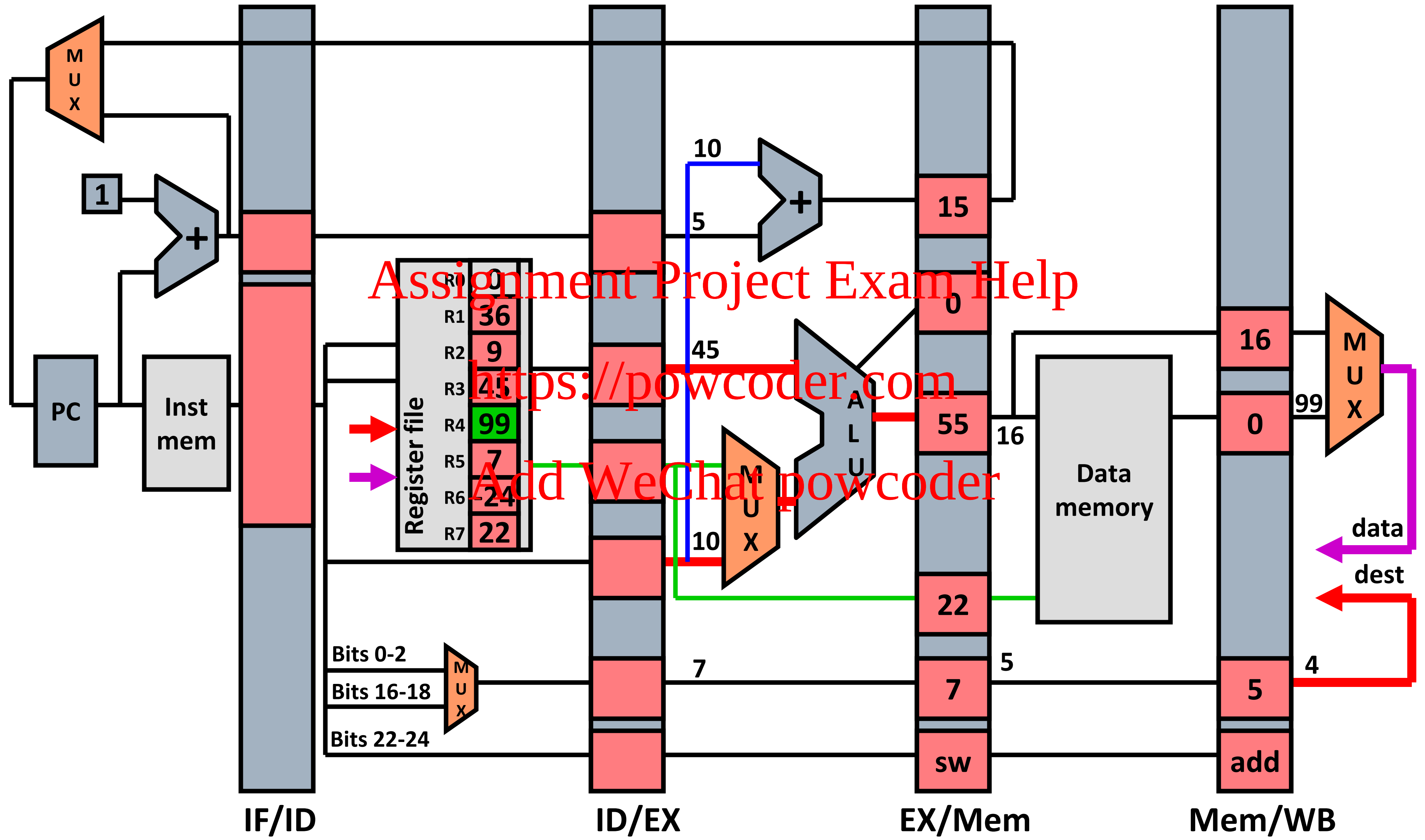
sw 3 7 10 add 2 5 5 lw 2 4 20 nor 4 5 6 add 1 2 3

Time 6 – No More Instructions



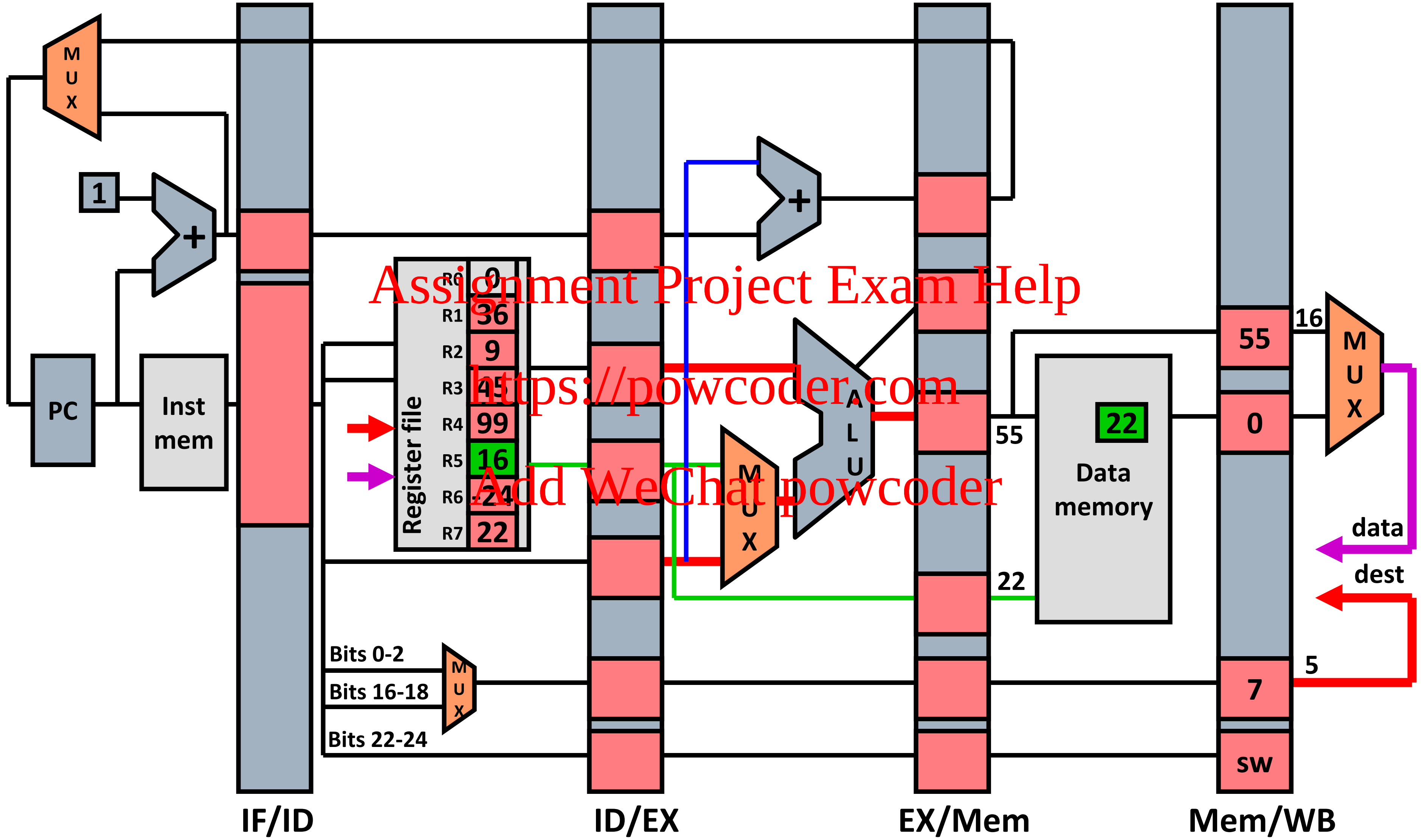
sw 3 7 10 add 2 5 5 lw 2 4 20 nor 4 5 6

Time 7 – No More Instructions



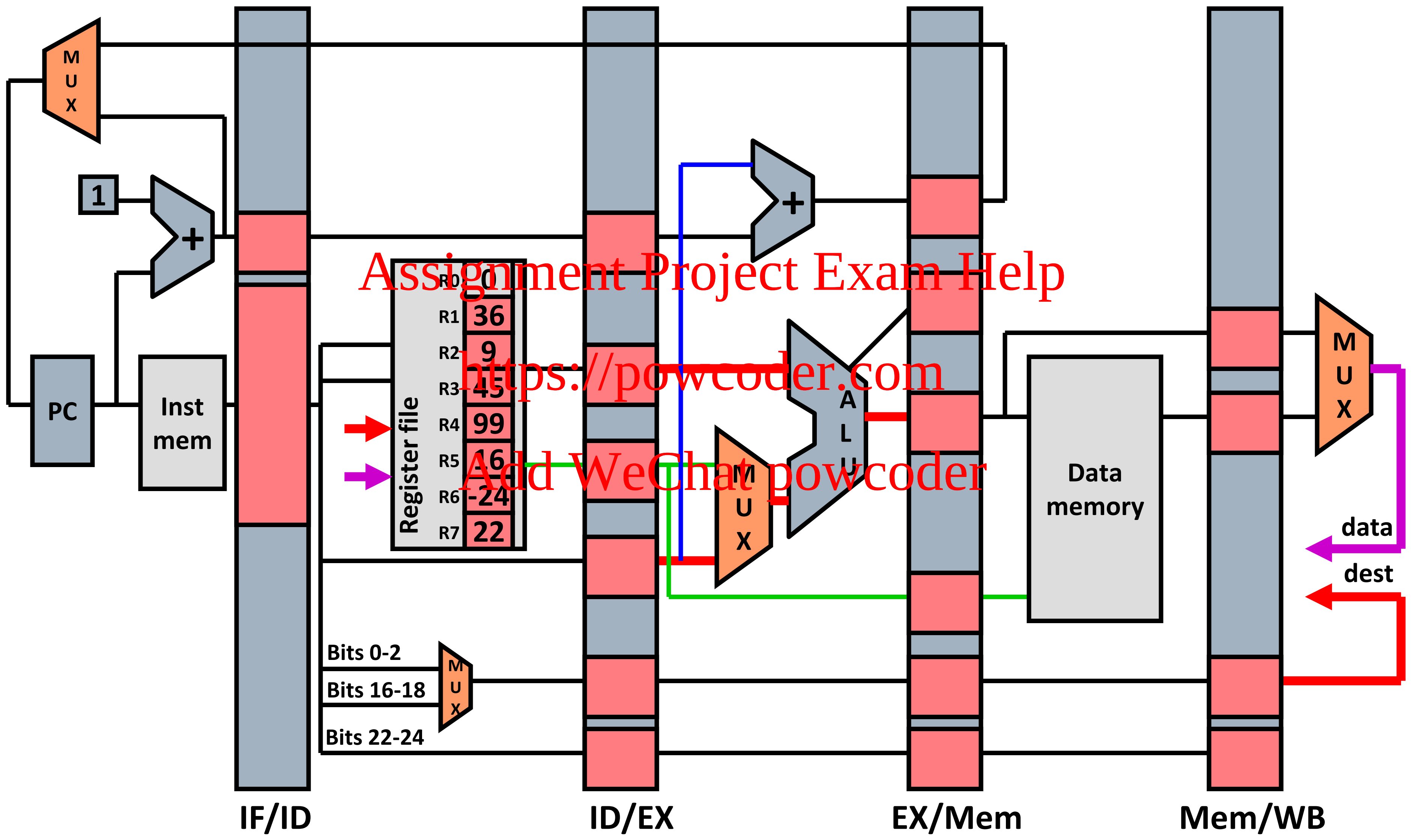
sw 3 7 10 **add 2 5 5** **lw 2 4 20**

Time 8 – No More Instructions



sw 3 7 10 add 2 5 5

Time 9 – No More Instructions



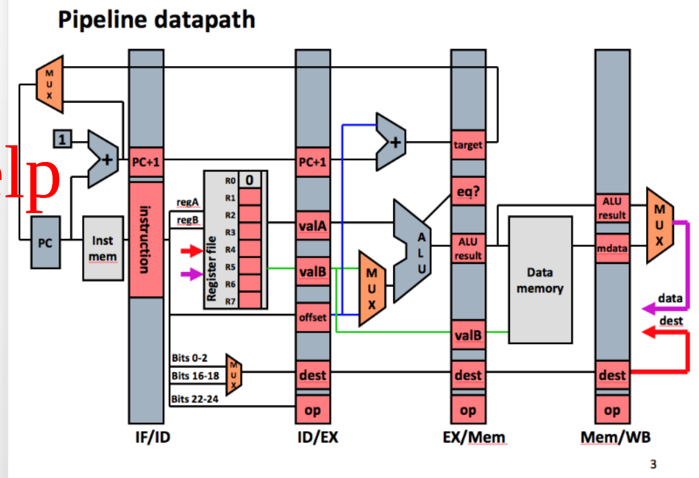
Time Graphs (a.k.a. Pipe Trace)

Time:	1	2	3	4	5	6	7	8	9
add	fetch	decode	execute	memory	writeback				
nor		fetch	decode	execute	memory	writeback			
lw			fetch	decode	execute	memory	writeback		
add				fetch	decode	execute	memory	writeback	
sw					fetch	decode	execute	memory	writeback

A vertical slice reports the entire activity of the pipeline at time 5

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 it 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?



Logistics

- There are 3 videos for lecture 13
  - L13_1 – Pipelining_Execution-Example
  - L13_2 – Data-Hazards
  - L13_3 – Data-Hazards_Detect-and-Stall
- There is one worksheet for lecture 13
  1. L13 worksheet

L13_2 Data-Hazards

EECS 370 – Introduction to Computer Organization – Fall 2020

Learning Objectives

- Ability to identify data dependencies between instructions.
- To differentiate between data dependencies and data hazards.
- To identify the approaches to resolving data hazards.

Data Hazards

- Register reads occur in stage 2 and register writes occur in stage 5
  - It is possible to read the wrong value if it is about to be written.

- Data hazards occur when the pipeline must be stalled because one step must wait for another to complete.
  - Data hazards arise from the dependence of one instruction on an earlier one that is still in the pipeline

Example: AND gate using NOR

```
nor  1  1 [2]
nor  3  3 [4]
nor [2][4] 5
```

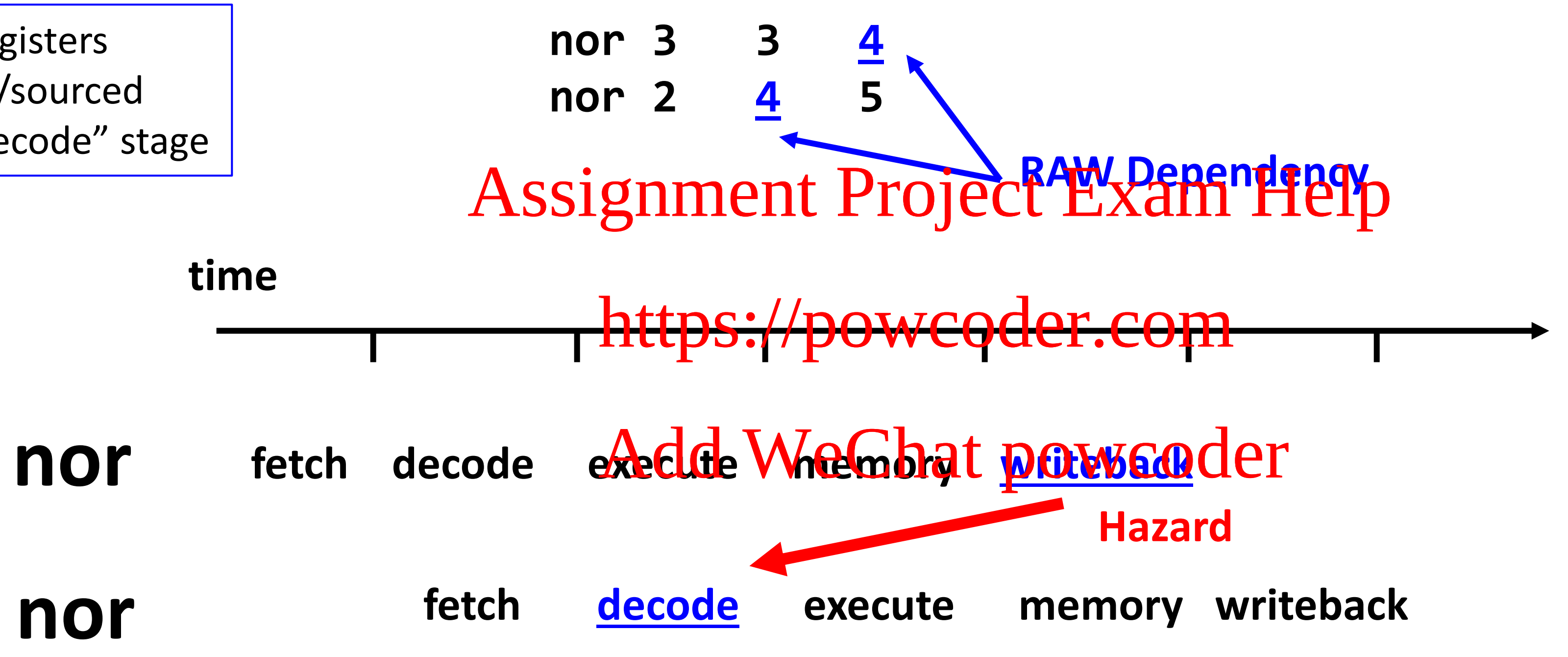
Pipeline Function for **nor**

- Fetch: read instruction from memory
- Decode: **read source operands from reg**
- Execute: calculate nor
- Memory: pass results to next stage
- Writeback: **write sum into register file**

Data Dependency - Example

Data Hazards

Recall: registers are read /sourced In the “decode” stage

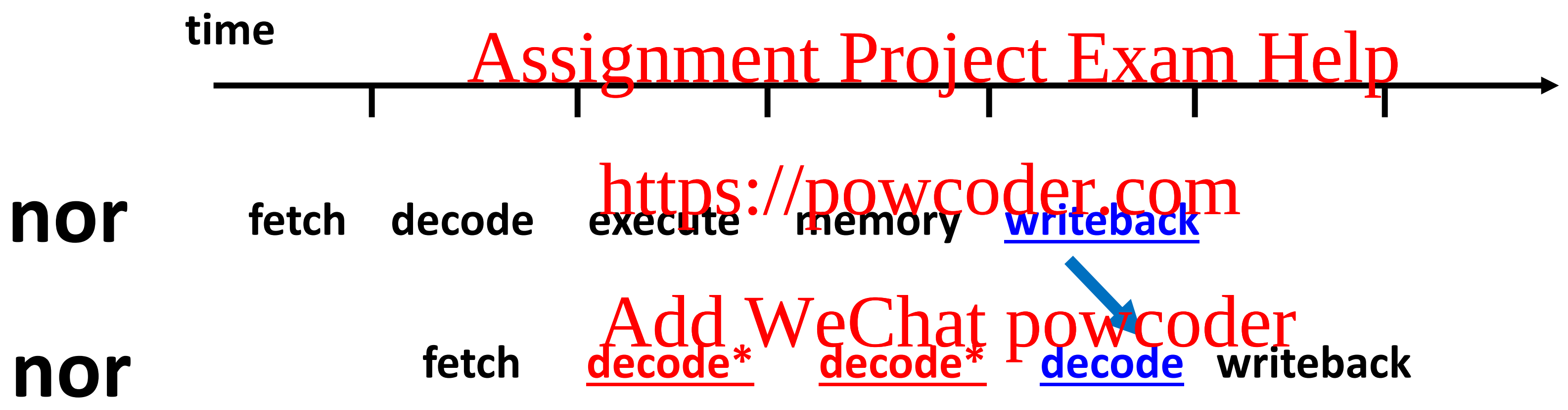


If not careful, nor will read a stale value of register 4

RAW dependency: **R**ead-**A**fter-**W**rite

Data Hazard - Solution

```
nor  3    3    4
nor  2    4    5
```



Assume Register File gives the right value of register 4 when read/written during <u>same</u> cycle. This is consistent with most processors (ARM/x86), <u>but not Project 3</u>.

Data Dependency and Data Hazard

Data Hazards

- Data Dependency: one instruction uses the result of a previous one
  - Does not necessarily cause a problem
- Data Hazard: one instruction has a data dependency that will cause a problem if we do not "deal with it"

Data Dependency - Example

Data Hazards

Problem: Which of these instructions has a data dependency on an earlier one?
Which of those are data hazards?

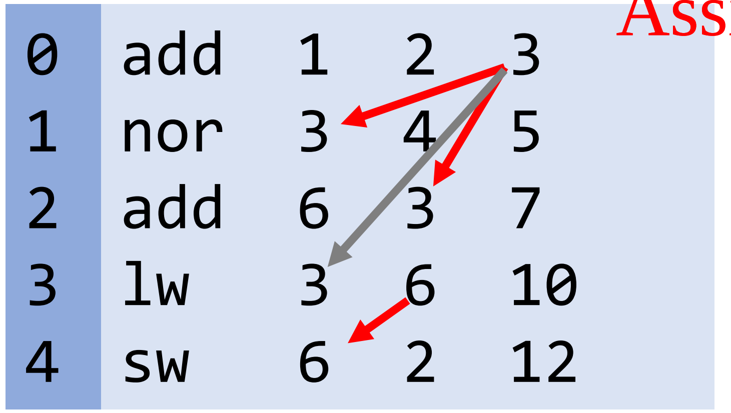
```
0  add  1  2  3
1  nor  3  4  5
2  add  6  3  7
3  lw   3  6  10
4  sw   6  2  12
```

```
0  add 1  2  3
1  beq 3  4  1
2  add 3  5  6
3  add 3  6  7
```

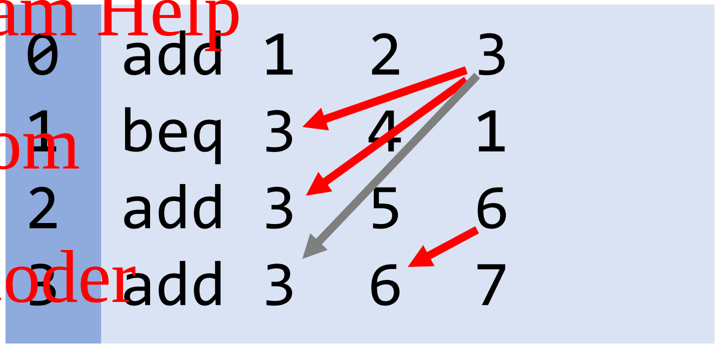
Data Hazards

Data Dependency - Example

Problem: Which of these instructions has a data dependency on an earlier one?
Which of those are data hazards?



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



```
0  add 1  2  3
1  beq 3  4  1
2  add 3  5  6
3  add 3  6  7
```

Three Approaches to Handling Data Hazards

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)

Data Hazards

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 register 3 in cycle 5
noop
noop
nor   3   4   5      ← read register 3 in cycle 5
```

Avoid all Hazards: Problems

- 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

Logistics

- There are 3 videos for lecture 13
  - L13_1 – Pipelining_Execution-Example
  - L13_2 – Data-Hazards
  - L13_3 – Data-Hazards_Detect-and-Stall
- There is one worksheet for lecture 13
  1. L13 worksheet

L13_3 Data-Hazards_Detect-and-Stall

EECS 370 – Introduction to Computer Organization – Fall 2020

Learning Objectives

- To identify and understand the pipeline datapath components necessary to facilitate detection and stalling for data hazards.

Handling Data Hazards `II`: Detect and Stall

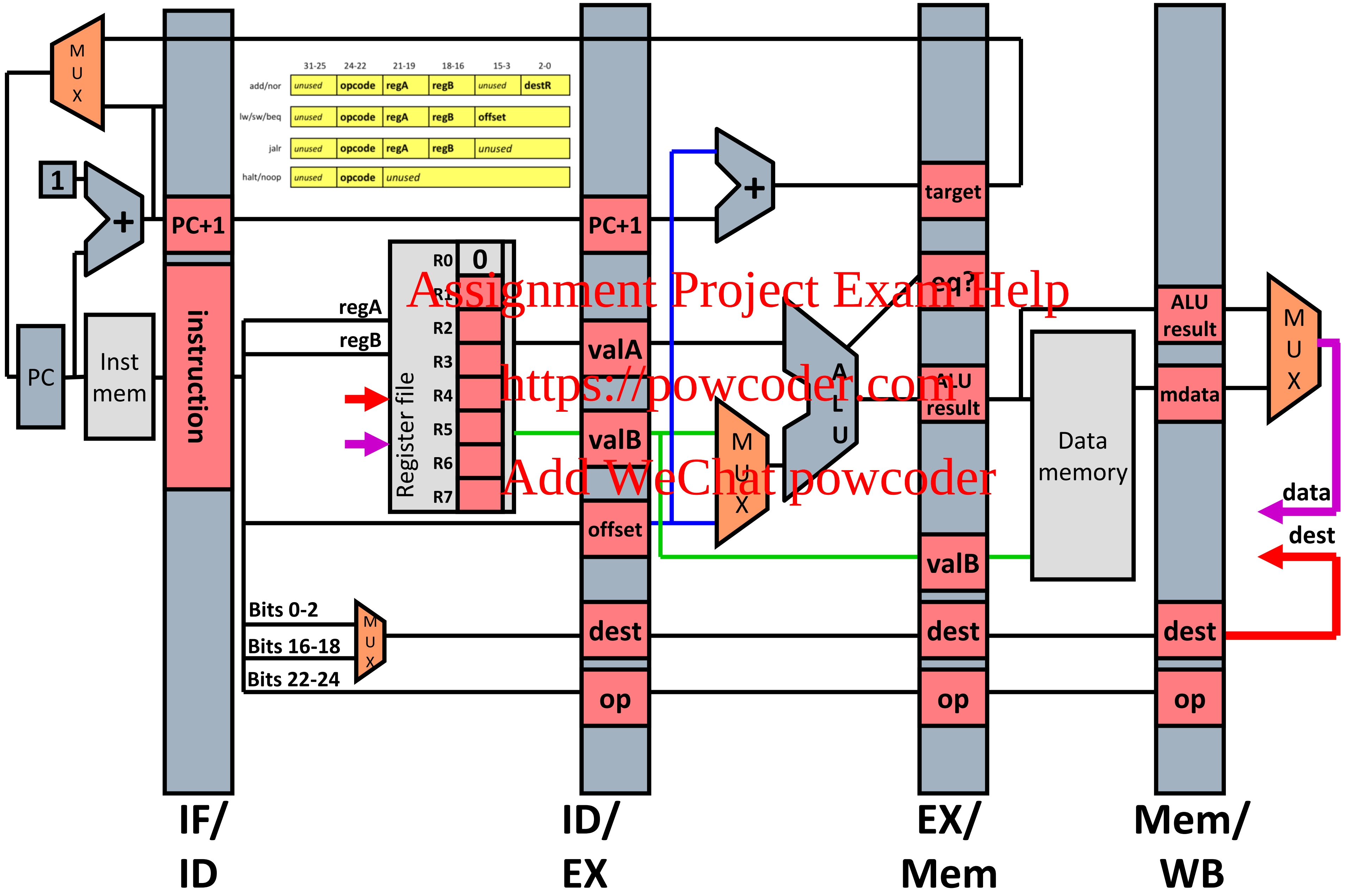
- Detect:
  - Compare `regA` with previous `destRegs`
    - 3 bit operand fields
  - Compare `regB` with previous `destRegs`
    - 3 bit operand fields
- Stall:
  - Keep current instructions in fetch and decode
  - Pass a noop to execute
- How do we modify the pipeline to do this?

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								
nor 3 4 5		IF	ID*	ID*	ID	EX	ME	WB					

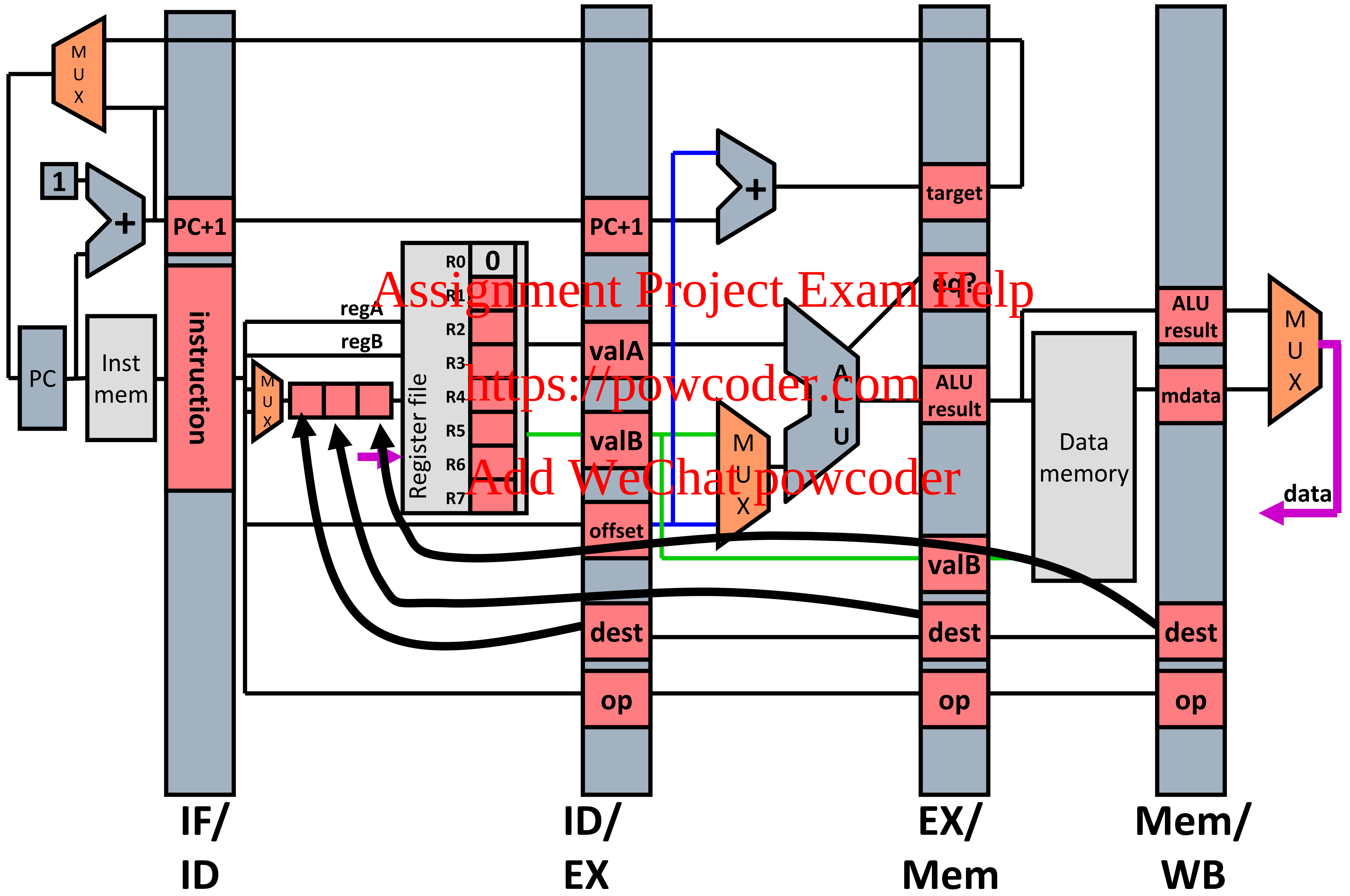
Problem: Our pipeline currently does not handle hazards – let us fix it

Data Hazards



Problem: Our pipeline currently does not handle hazards – let us fix it

Data Hazards



Example

- Let’s run this program with a data hazard through our 5-stage pipeline

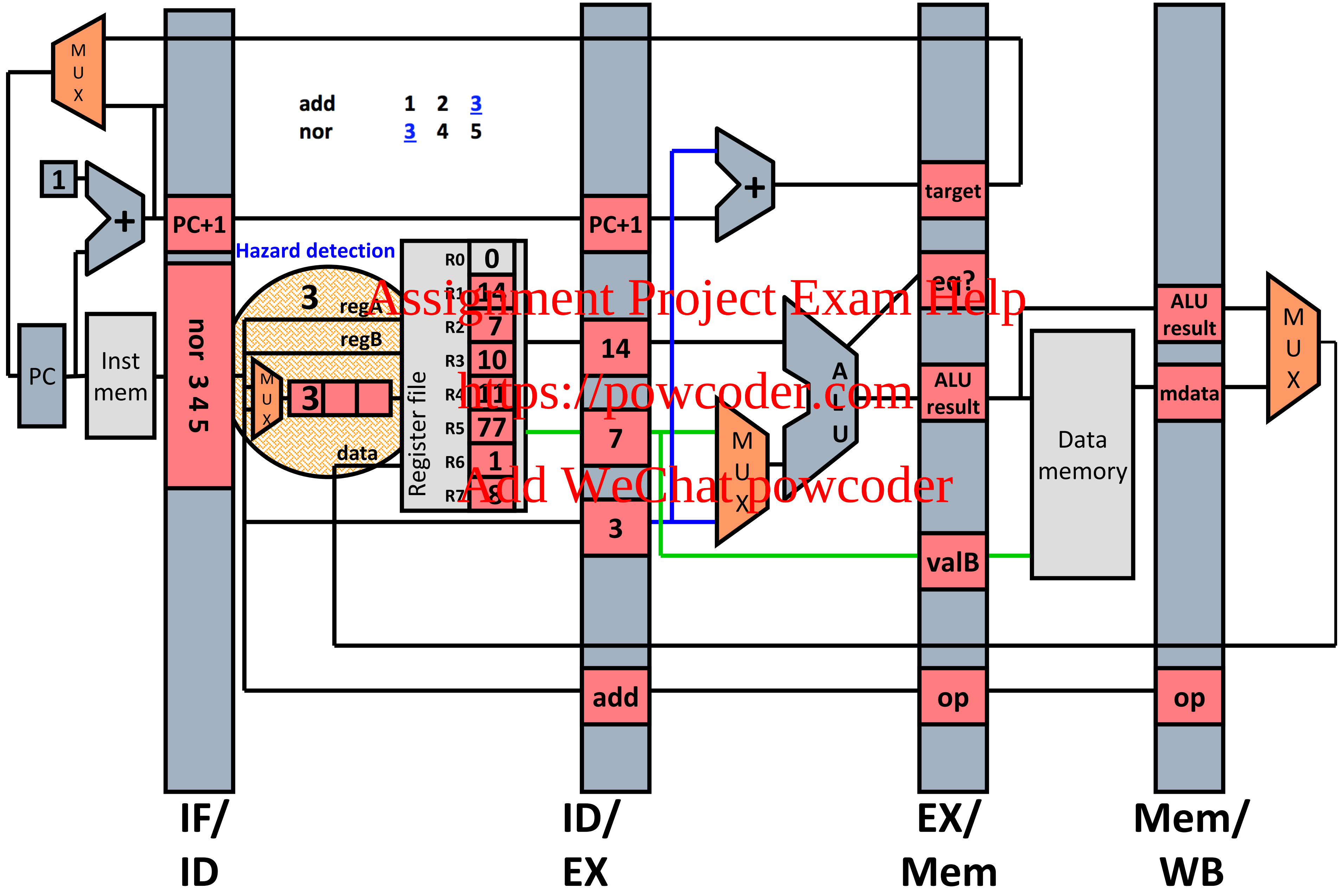
```
add 1 2 3
nor 3 4 5
```

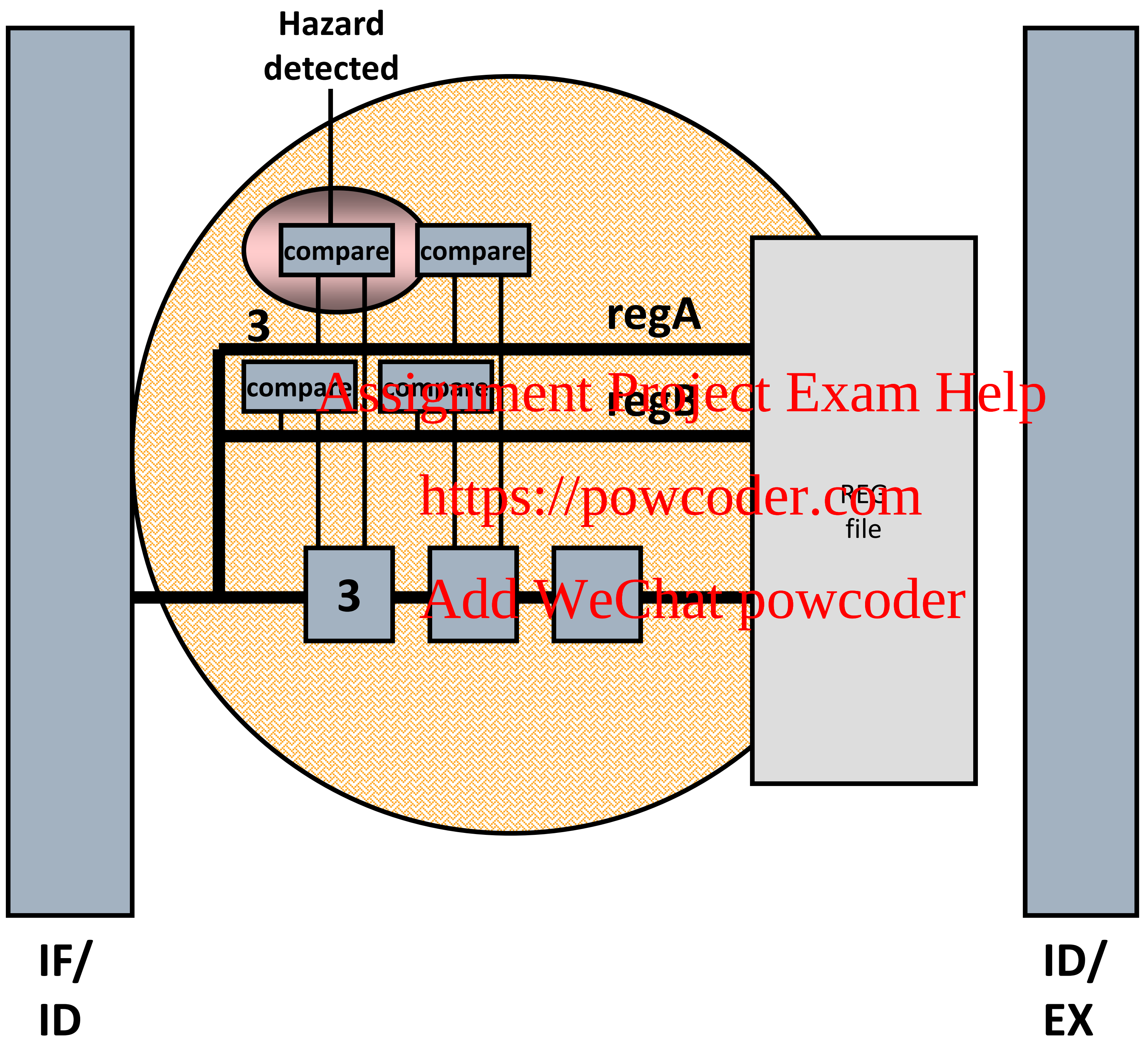
- We will start at the beginning of cycle 3, where add is in the EX stage, and nor is in the ID stage, about to read a register value

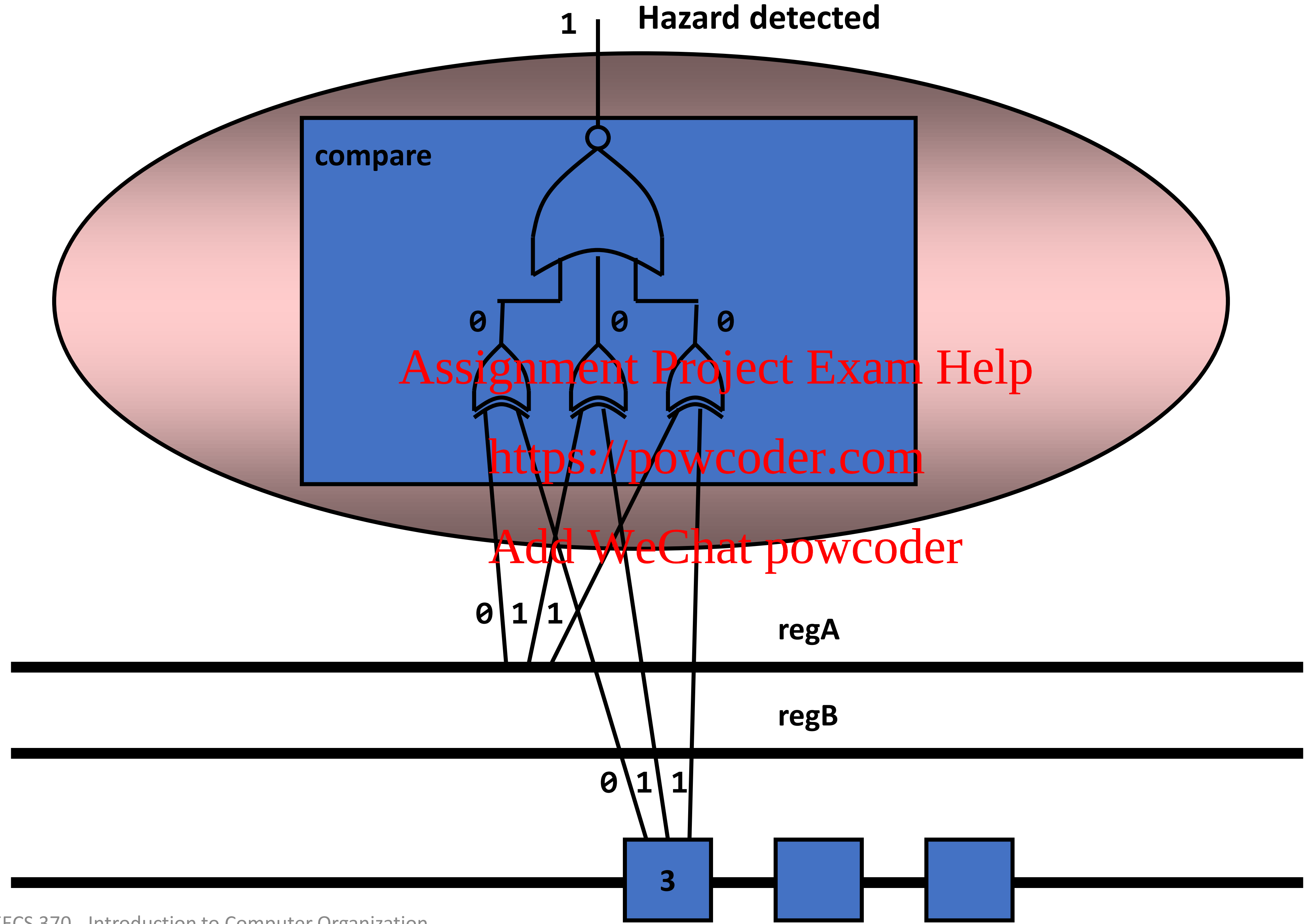
Time:	1	2	3
add 1 2 3	IF	ID	EX
nor 3 4 5		IF	ID

Hazard!

First half of cycle 3




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

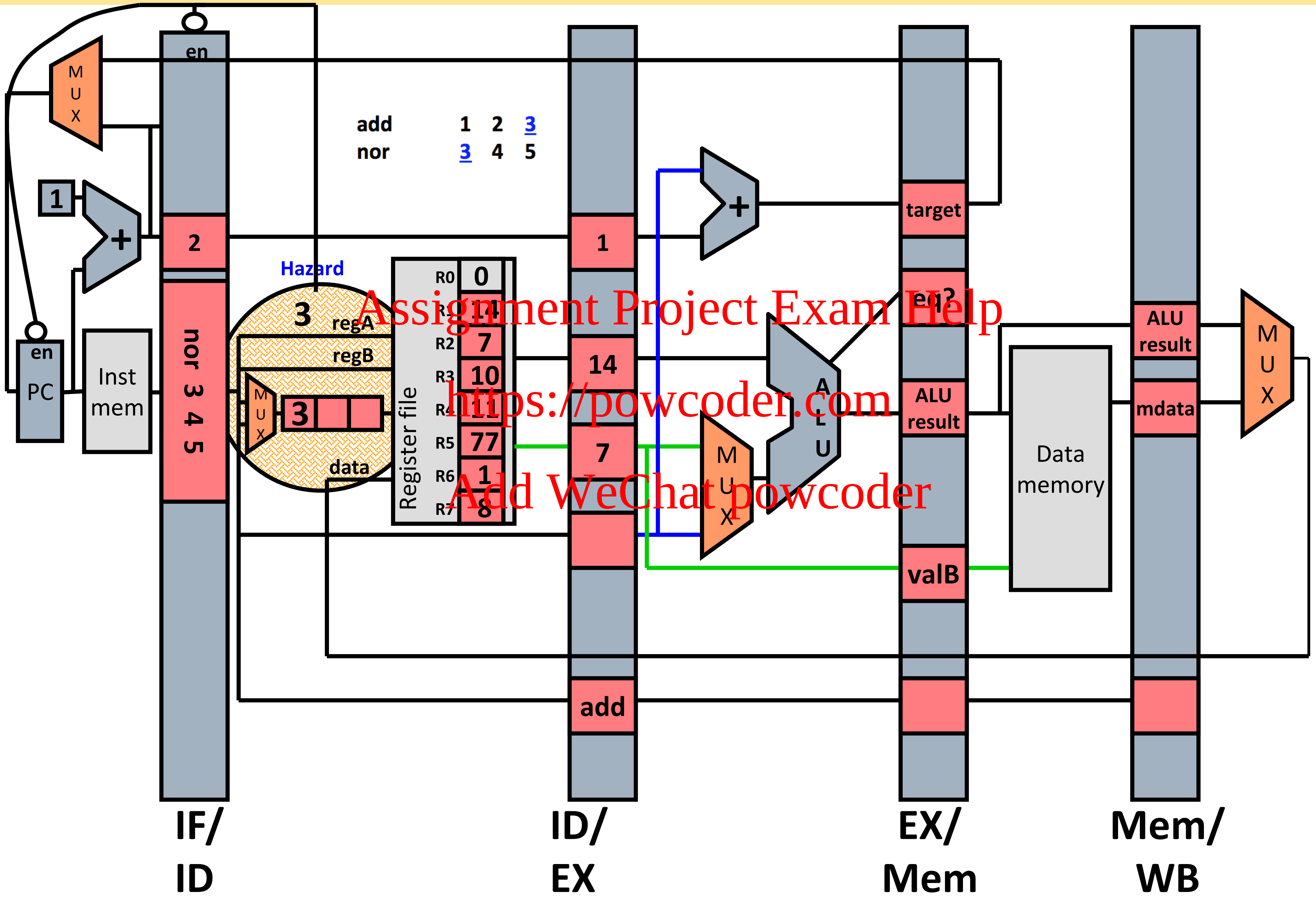

1
Hazard detected
compare
0
0
0
0
1
1
regA
regB
0
1
1
3

Handling Data Hazards II: Detect and <u>**Stall**</u>

Data Hazards

- Detect:
  - Compare regA with previous destRegs
    - 3 bit operand fields
  - Compare regB with previous destRegs
    - 3 bit operand fields
- Stall:
  - **<u>Keep current instructions in fetch and decode</u>**
  - Pass a noop to execute

First half of cycle 3

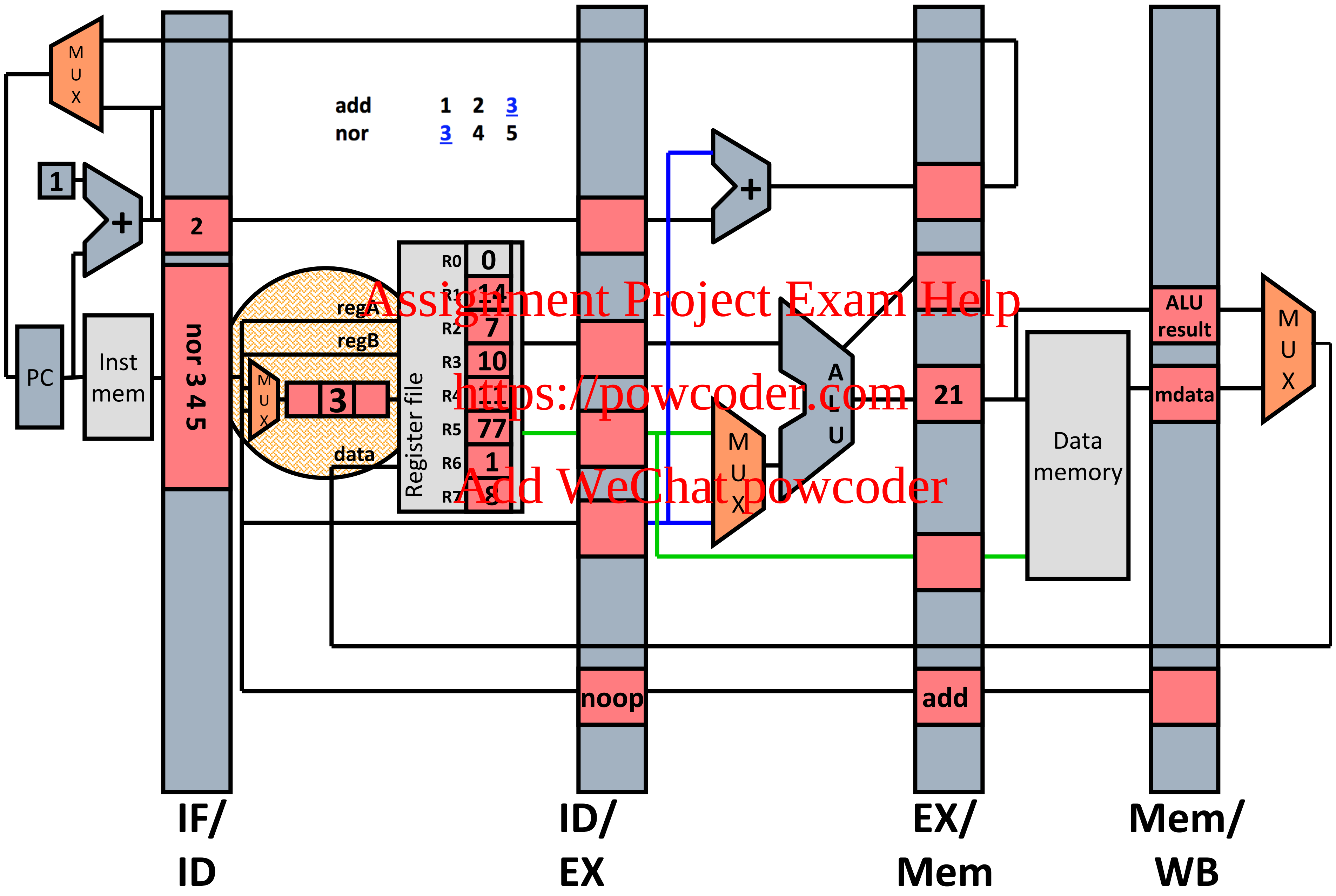


Handling Data Hazards II: Detect and **Stall**

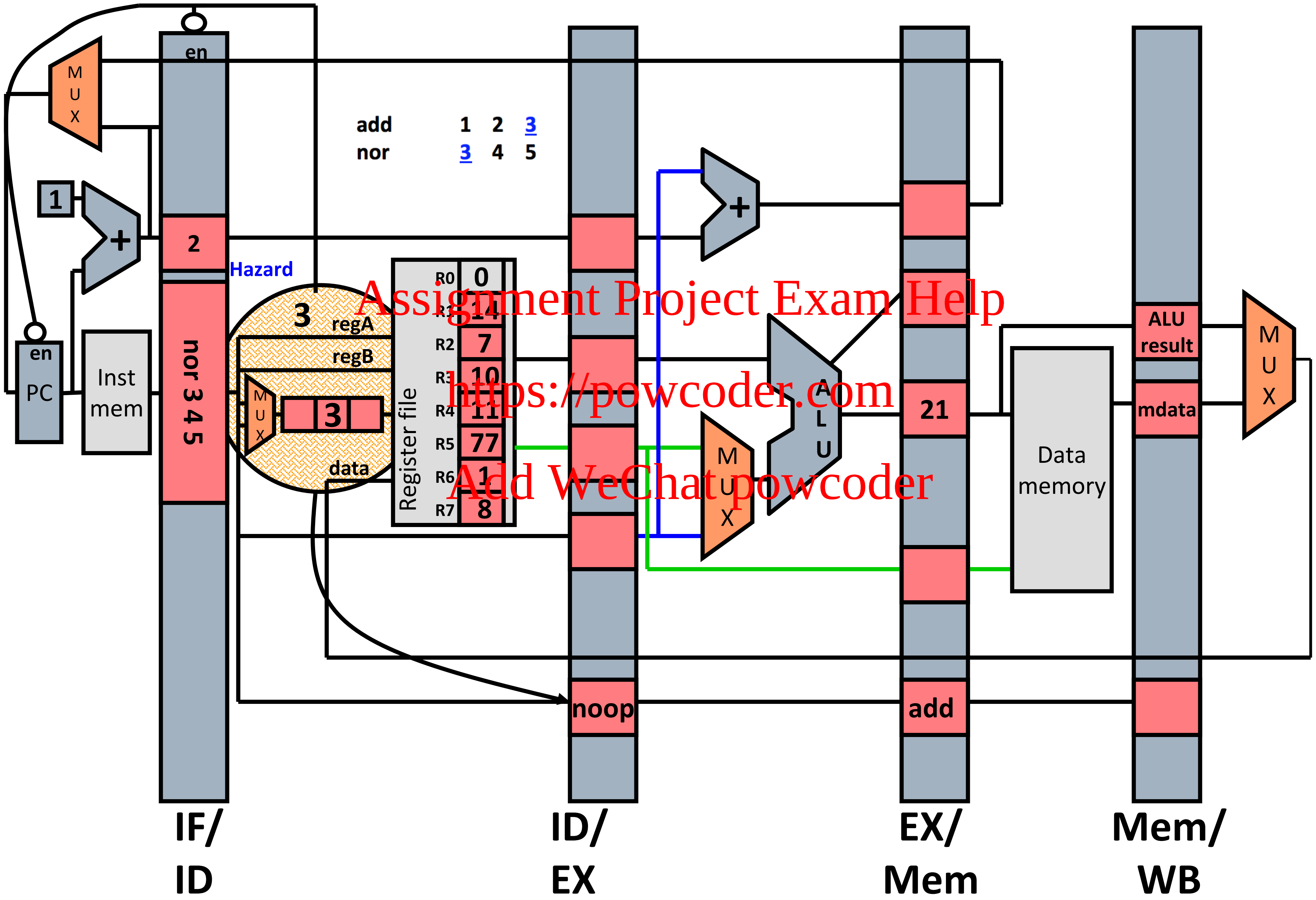
Data Hazards

- 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**

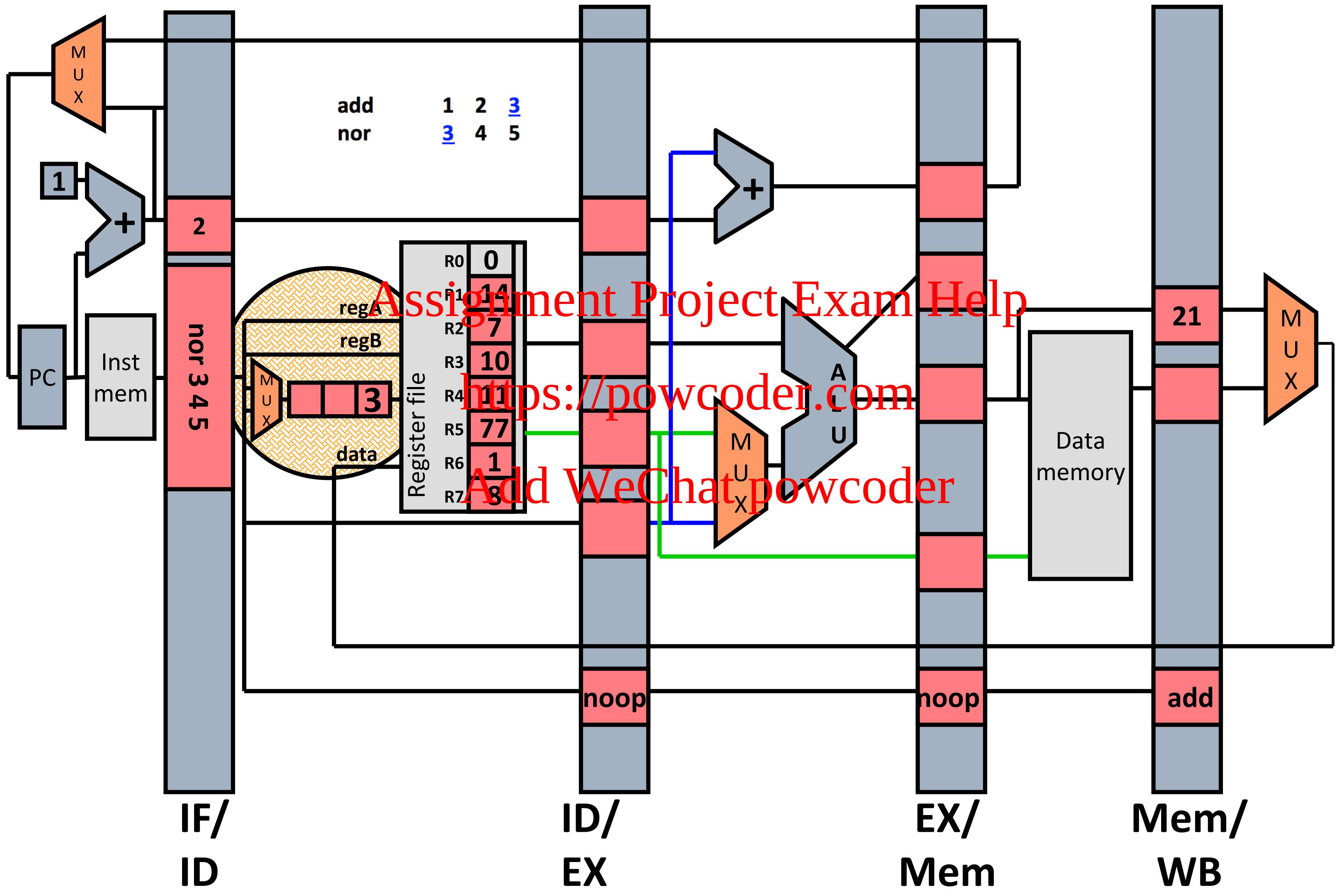
End of cycle 3



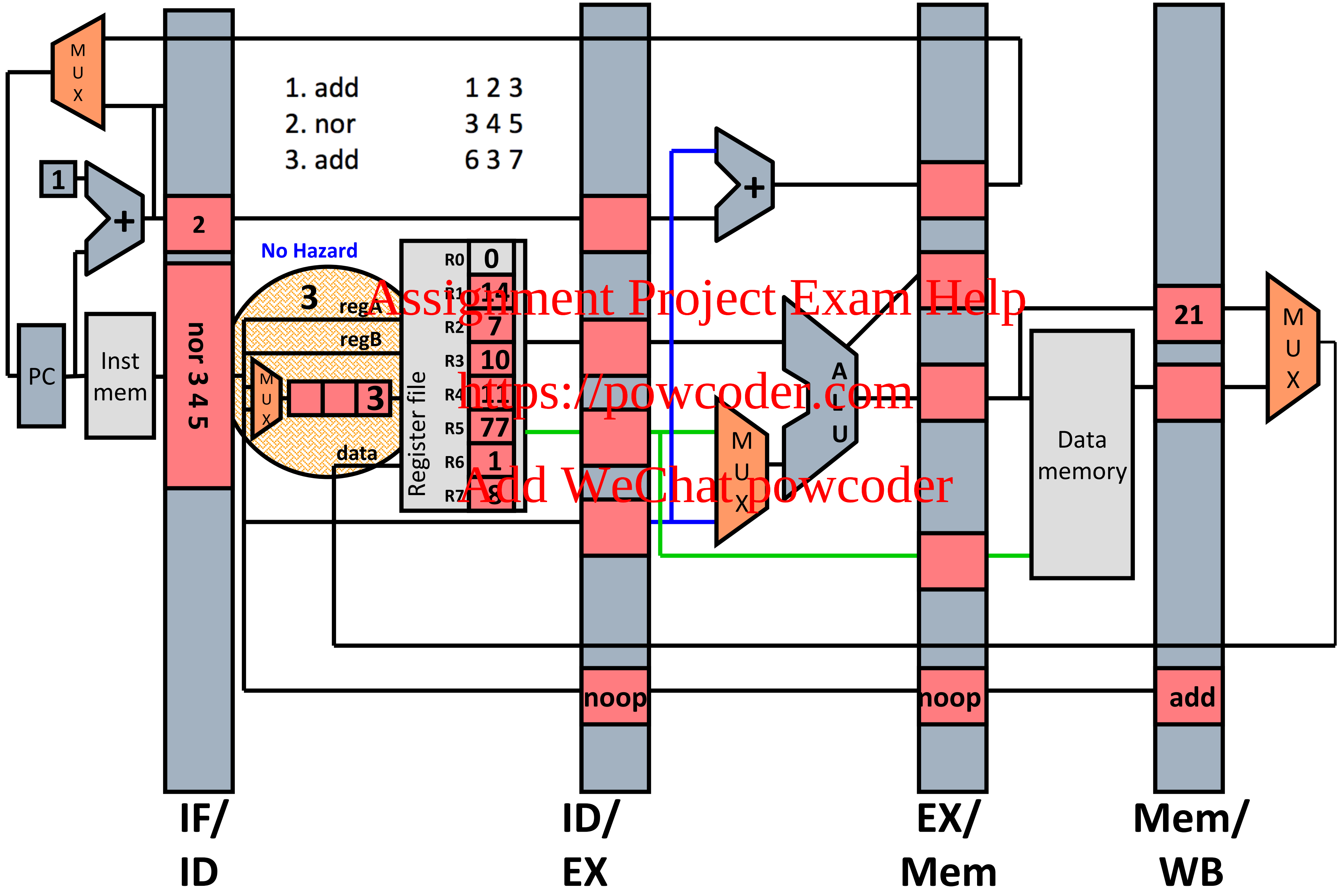
First half of cycle 4



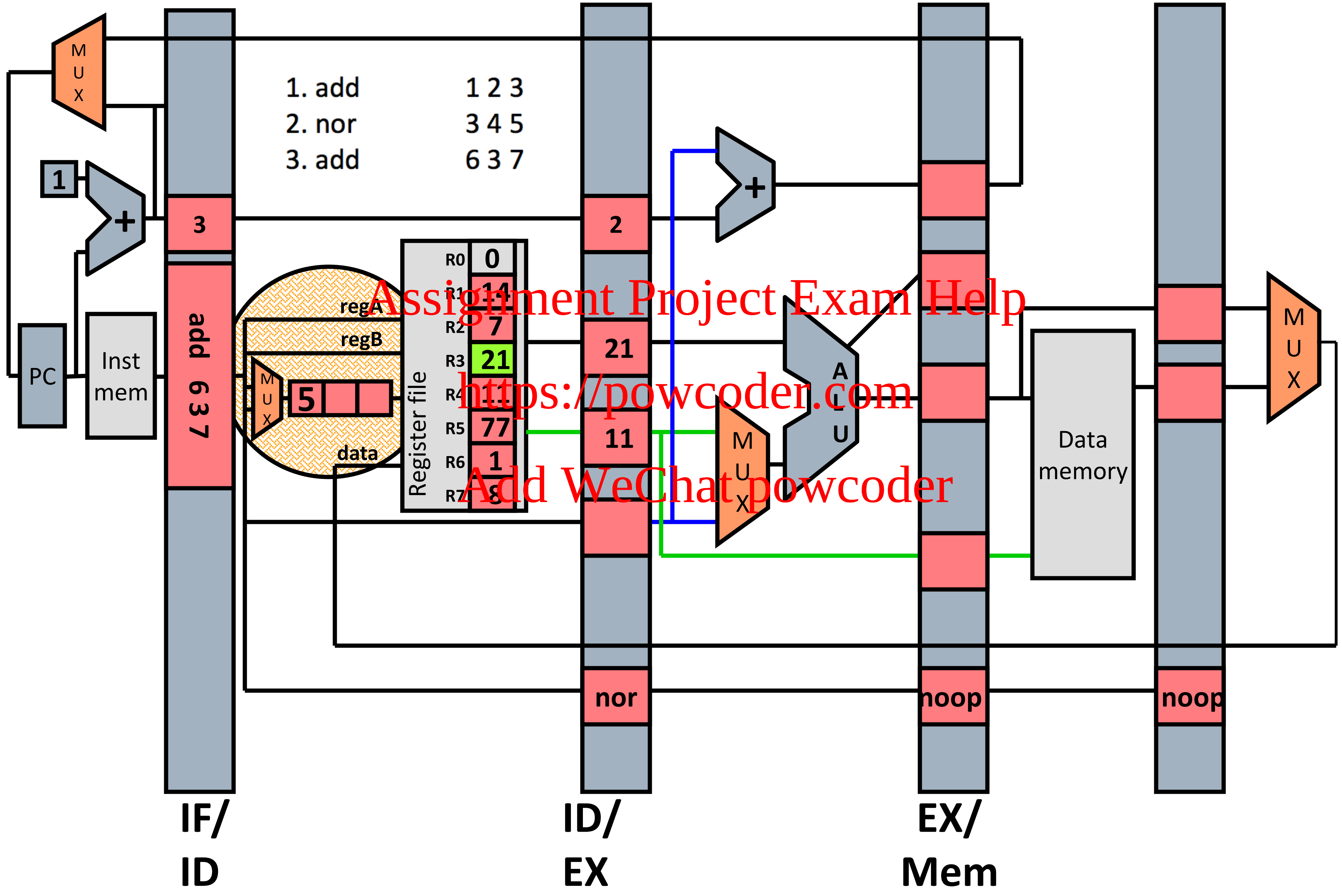
End of cycle 4



First half of cycle 5



End of cycle 5



Time Graph

Data Hazards

Time:	1	2	3	4	5	6	7	8	9	10	11	12	13
add 1 2 3	IF	ID	EX	ME	WB								
nor 3 4 5		IF	ID*	ID*	ID	EX	ME	WB					

Time Graph: Exercise

Data Hazards

Time:	1	2	3	4	5	6	7	8	9	10	11	12	13
add 1 2 3	IF	ID	EX	ME	WB								
nor 3 4 5		IF	ID*	ID*	ID	EX	ME	WB					
add 6 3 7													
lw 3 6 10													
sw 6 2 12													

1. Identify the data hazards in this extended program
2. Complete the time graph

Data Hazards

Time Graph: Exercise

Time:	1	2	3	4	5	6	7	8	9	10	11	12	13
add 1 2 3	IF	ID	EX	ME	WB								
nor 3 4 5		IF	ID*	ID*	ID	EX	ME	WB					
add 6 3 7													
lw 3 6 10													
sw 6 2 12													

Data Hazards

Time Graph: Solution

Time:	1	2	3	4	5	6	7	8	9	10	11	12	13
add 1 2 3	IF	ID	EX	ME	WB								
nor 3 4 5		IF	ID*	ID*	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	ID*	ID*	ID	EX	ME	WB

Detect and Stall - Benefits

- Benefits over "Avoid all hazards"
  - Backwards compatibility: noops will effectively be injected into pipeline by the processor, not necessary to know number of noops to insert when assembling
  - Fewer noops in code: smaller executables
    - Smaller executables
    - Less fetching of noops, fewer memory accesses

Detect and Stall - Problems

- CPI increases every time a hazard is detected!
- Is that necessary? Not always!
  - Re-route the result of the add to the nor
    - nor 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 are not getting from the reg file at this time will be found elsewhere in the pipeline at a later cycle.

Logistics

- There are 3 videos for lecture 13
  - L13_1 – Pipelining_Execution-Example
  - L13_2 – Data-Hazards
  - L13_3 – Data-Hazards_Detect-and-Stall
- There is one worksheet for lecture 13
  1. L13 worksheet