Lecture 11: Pipelining

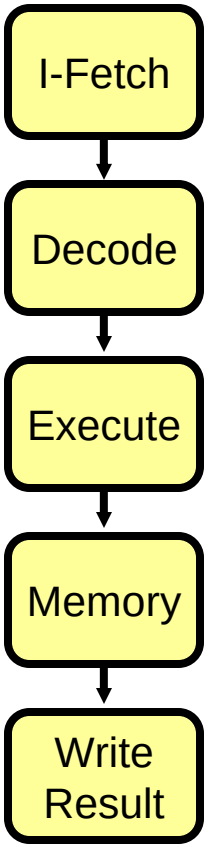
- Organizational
  - We're still grading the exam
- Last Time
  - Datapath control
  - Multicycle machine
  - Introduction to pipelining
- Today
  - Pipelining

Laundry Pipeline

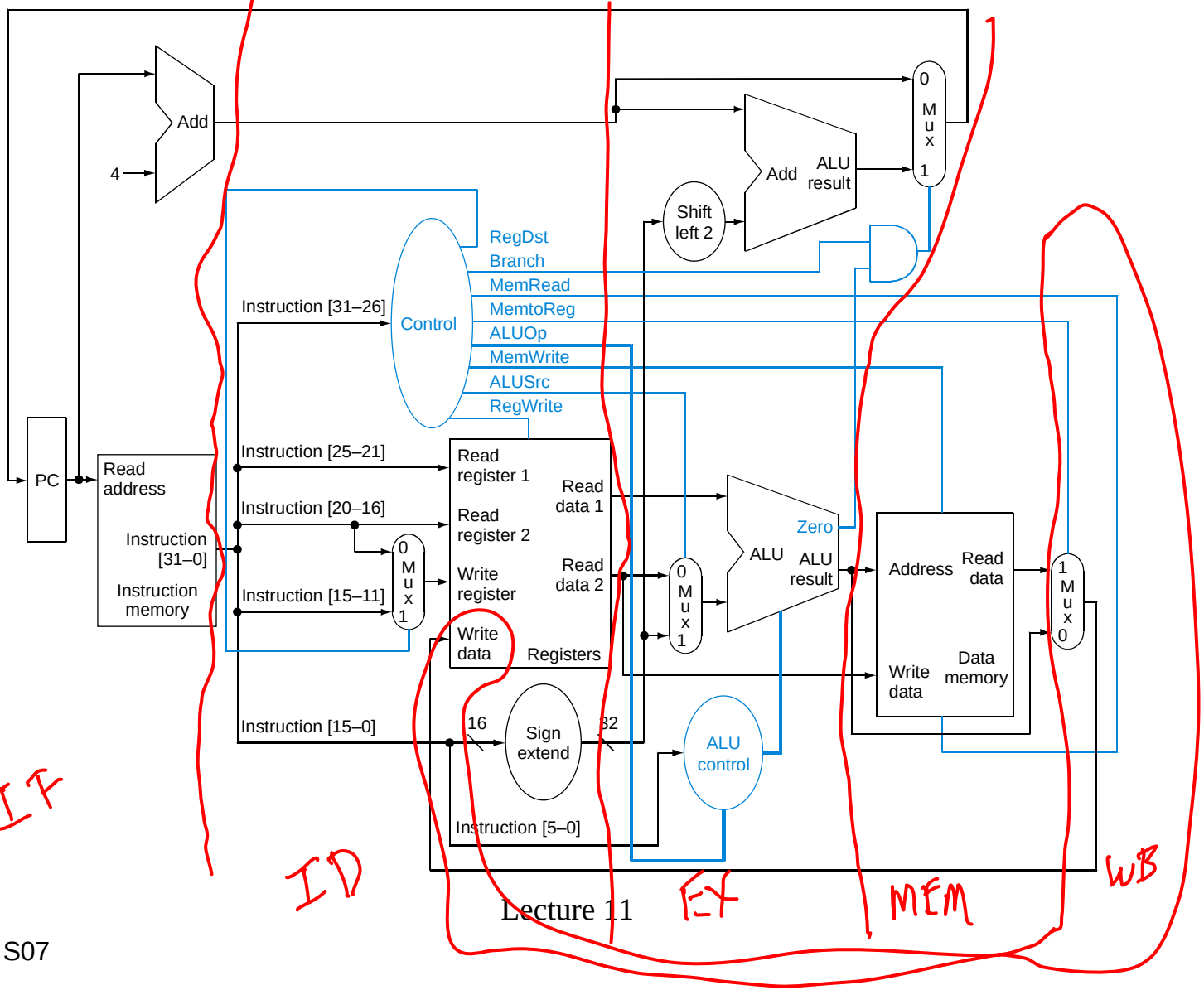
- 3 steps
  - Wash
  - Dry
  - Fold
- When first load finishes wash…
  - Immediately start the next load
- Eventually…
  - Three loads of laundry in progress at once
- Up to three times faster

Instruction Execution

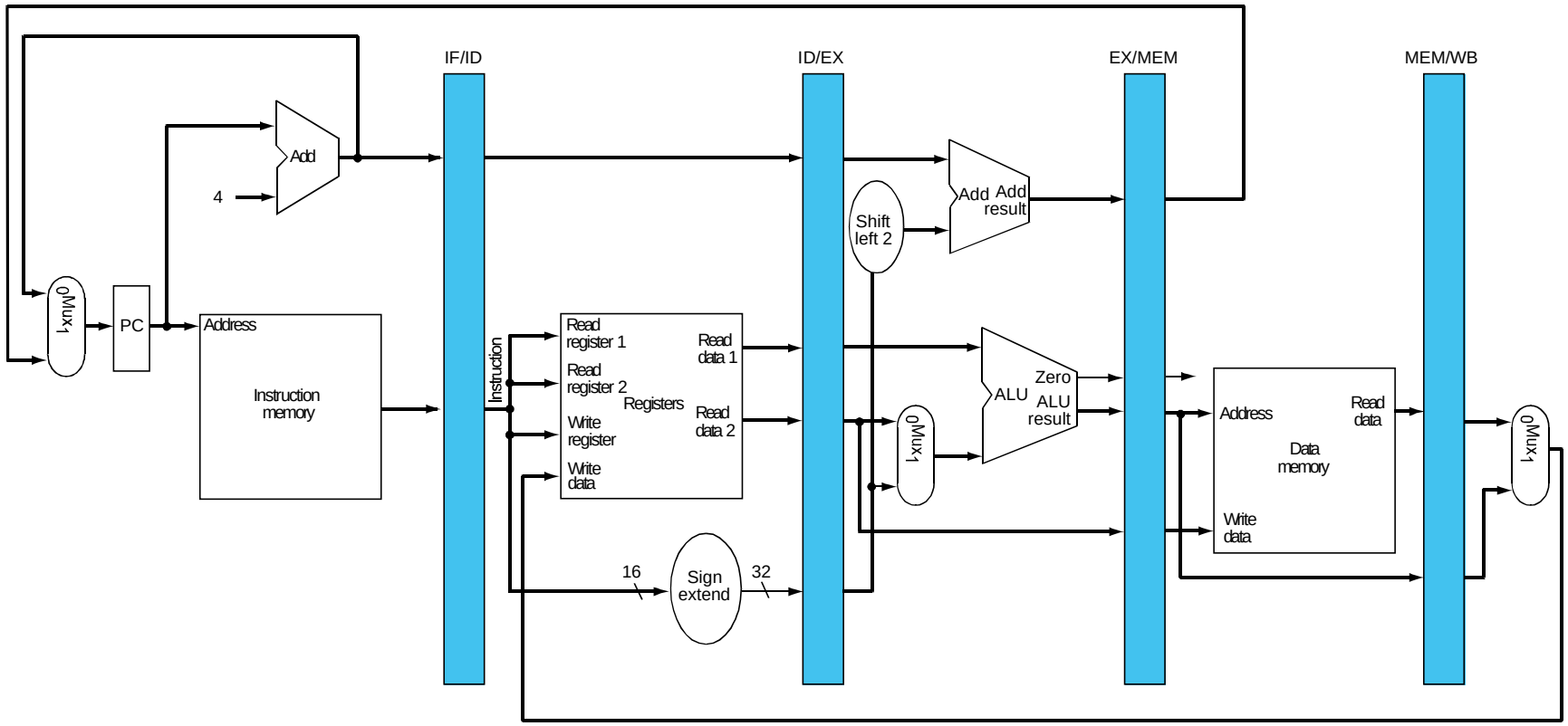
- 5 basic steps
  - fetch instruction (IF)
  - decode instruction/read registers (ID)
  - execute (EX)
  - access memory (MEM)
  - store result (WB)



"Single cycle" implementation can be broken into 5 stages

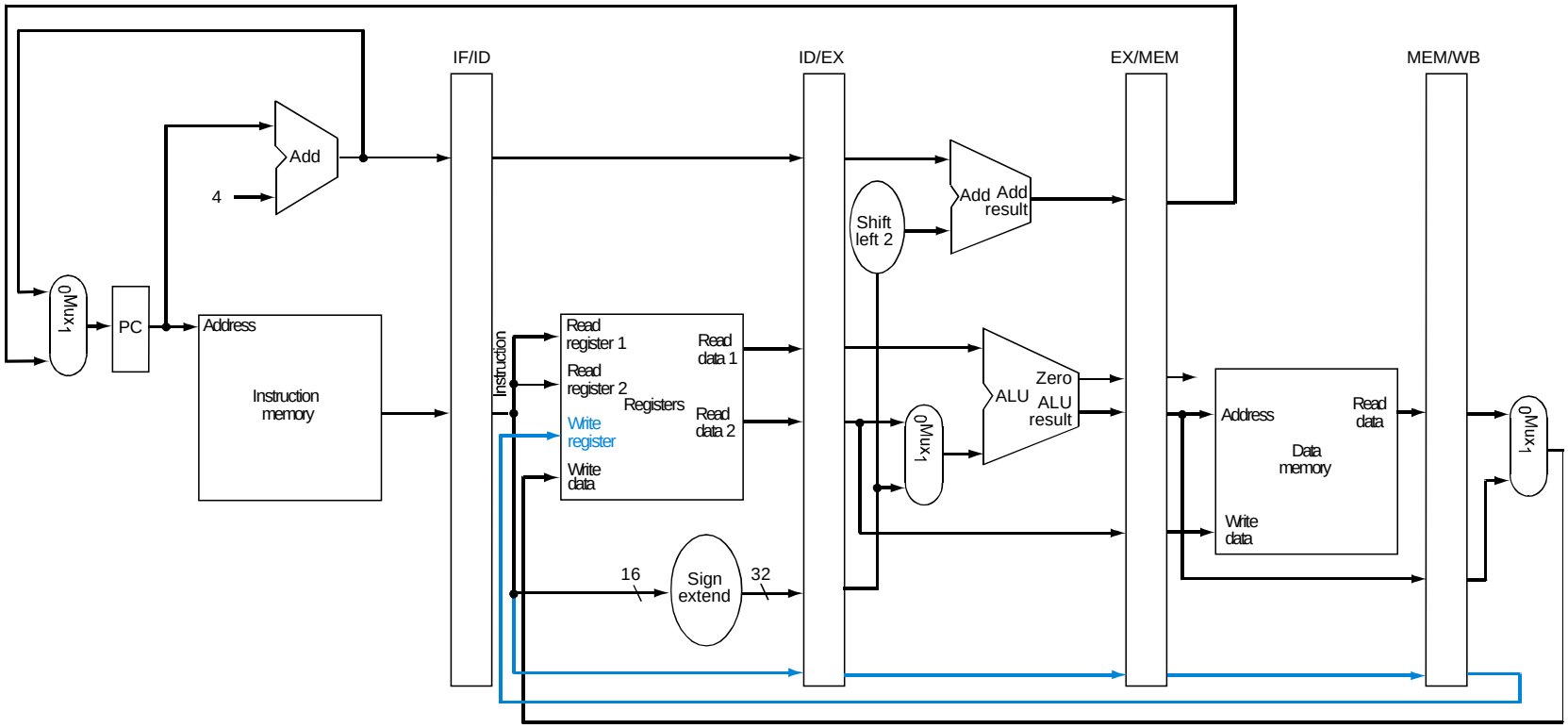


Separate into 5 stages with latches



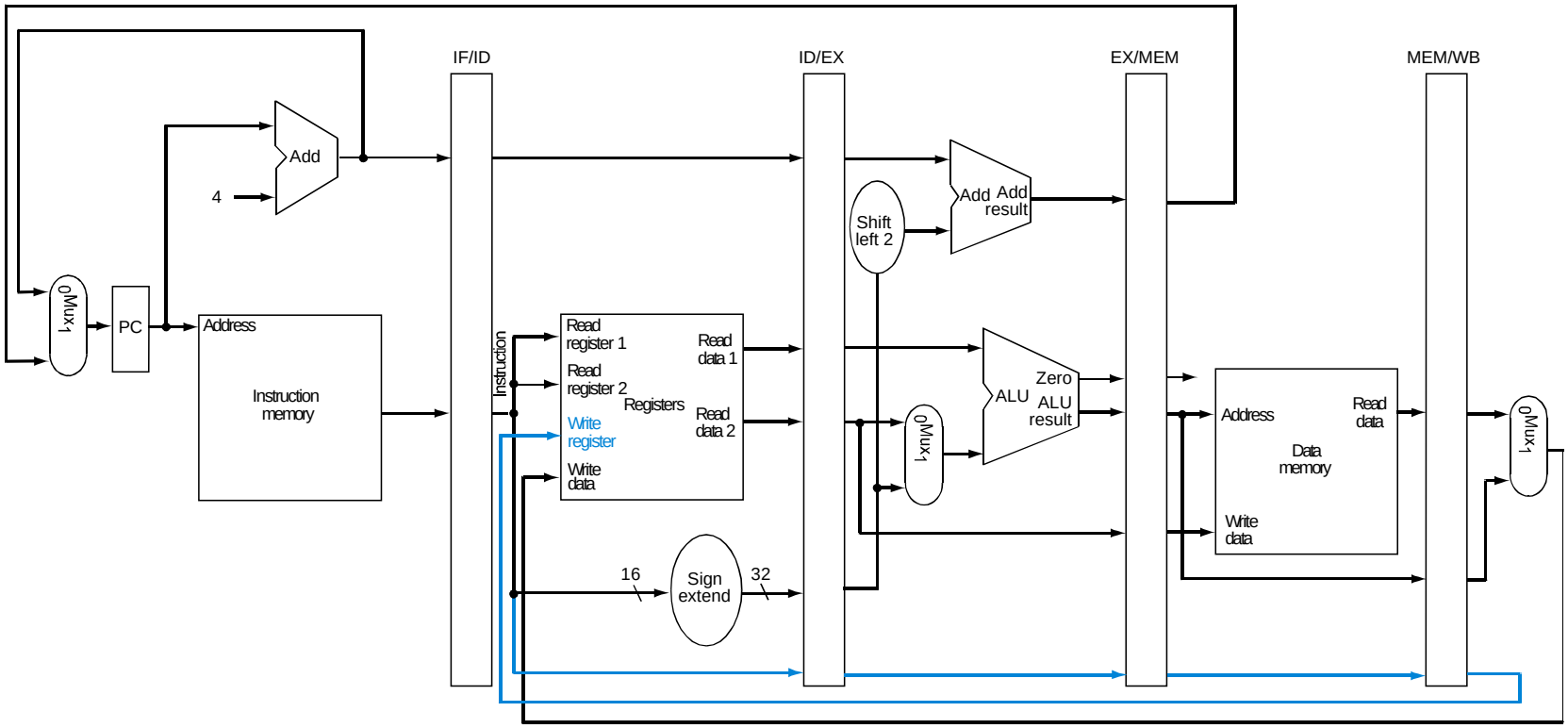
But… does the correct register get written?

Fix to write the correct register



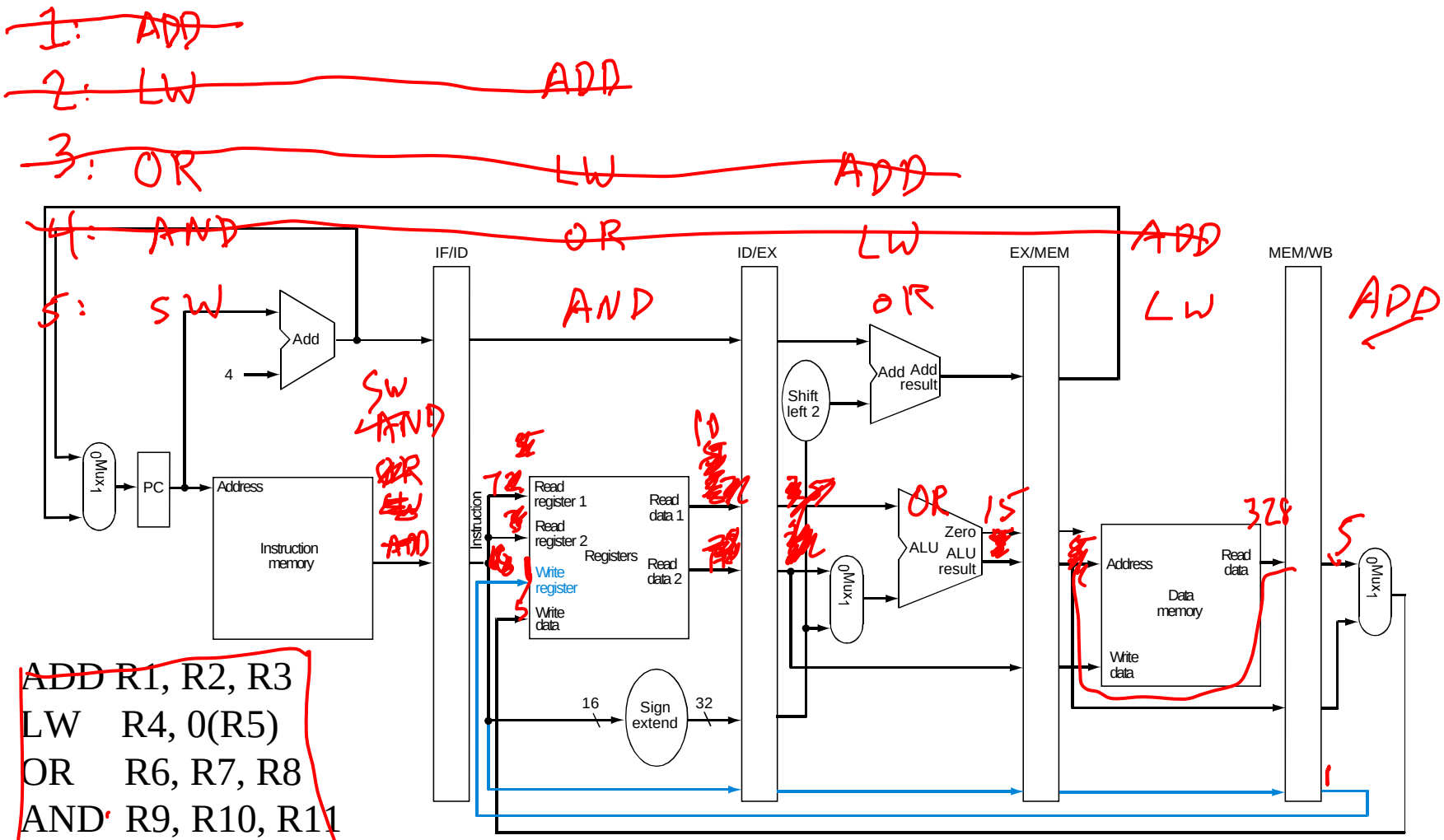
It's like attaching a note to the laundry basket used for a particular load

One instruction as it moves through pipeline



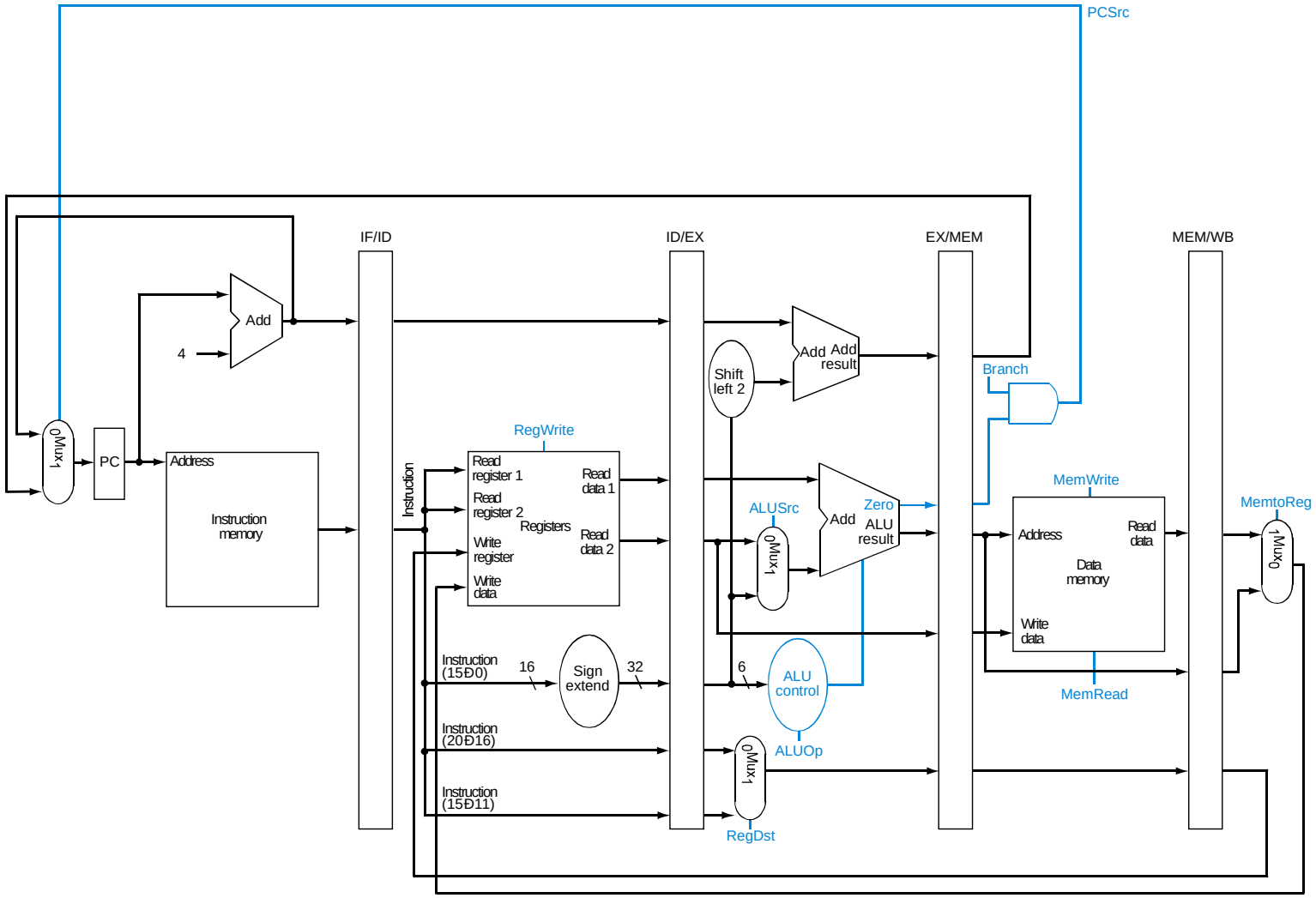
ADD R1, R2, R3

Several instructions in progress

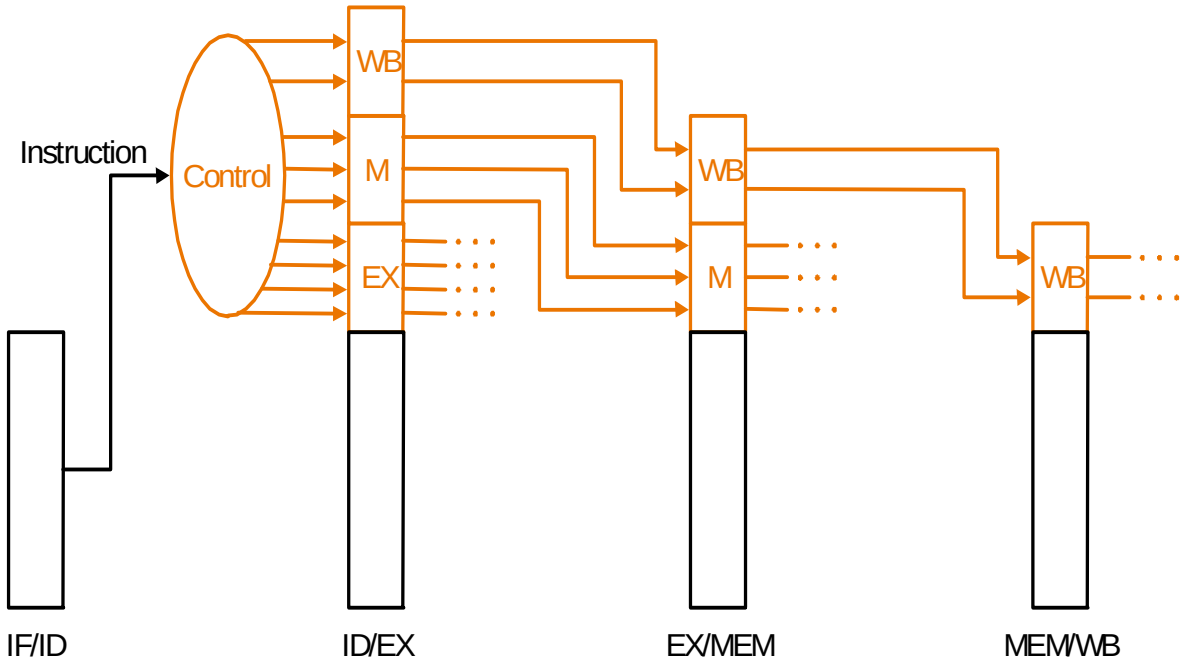


ADD R1, R2, R3
LW R4, 0(R5)
OR R6, R7, R8
AND R9, R10, R11
SW R12, 0(R13)

How do we control the pipeline?



Pass control signals down the pipeline

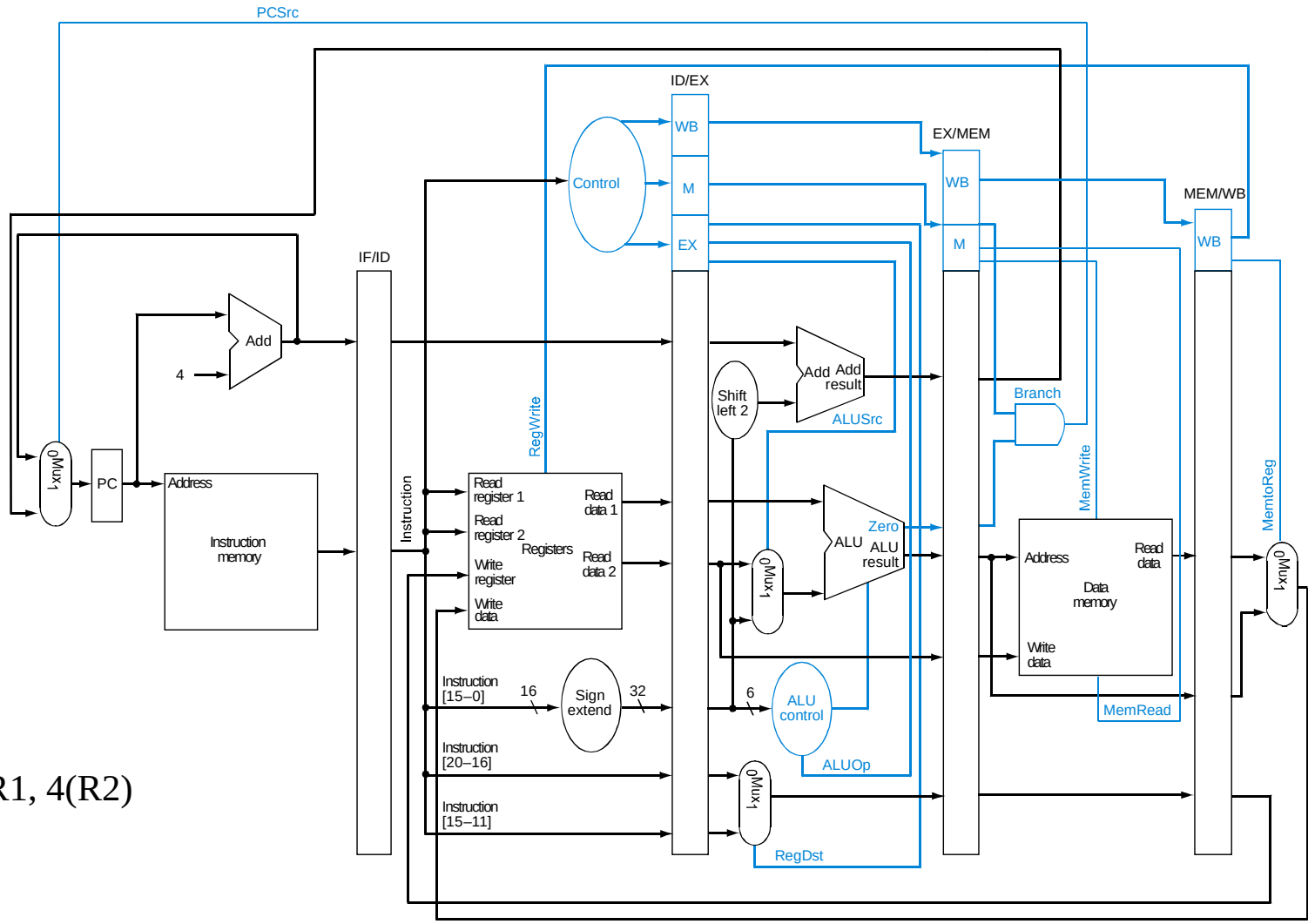


Laundry Analogy…

Attach instructions to basket:

-“wash with cold water”
-“tumble dry low”
- “iron on high”
- “hang in hall closet”

Datapath with control



LW R1, 4(R2)

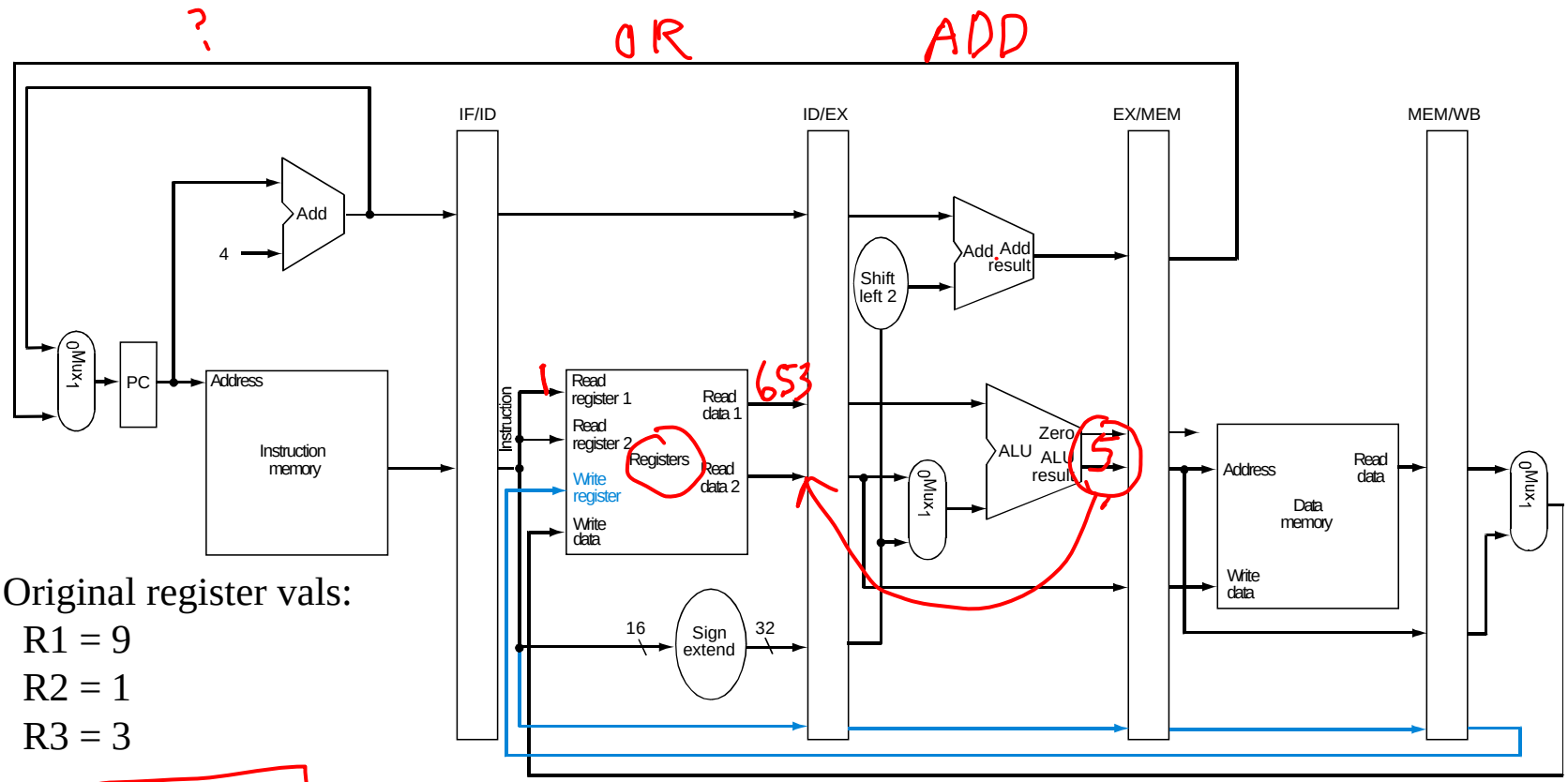
A fly in the soup…

- What happens when two instructions depend on one another?

```
ADD R1, R2, R3
OR  R4, R1, R5
```

- Let's see…

R1 = 653



Original register vals:

R1 = 9

R2 = 1

R3 = 3

ADD R1, R2, R3

OR R4, R1, R5

ADD R1, ..

NOP

NOP

OR .., R1

That was a *Data Hazard*

- Data Hazard:
  - An instruction depends on a result computed by a (nearby) previous instruction
  - If we're not careful, we get "stale" data instead of the correct data.

Three kinds of Pipeline Hazards

- Data hazards
  - an instruction uses the result of a previous instruction (RAW)

```
ADD   R1, R2, R3        or      SW    R1, 3(R2)
ADD   R4, R1, R5                LW    R3, 3(R2)
```

- Control hazards
  - the location of an instruction depends on a previous instruction

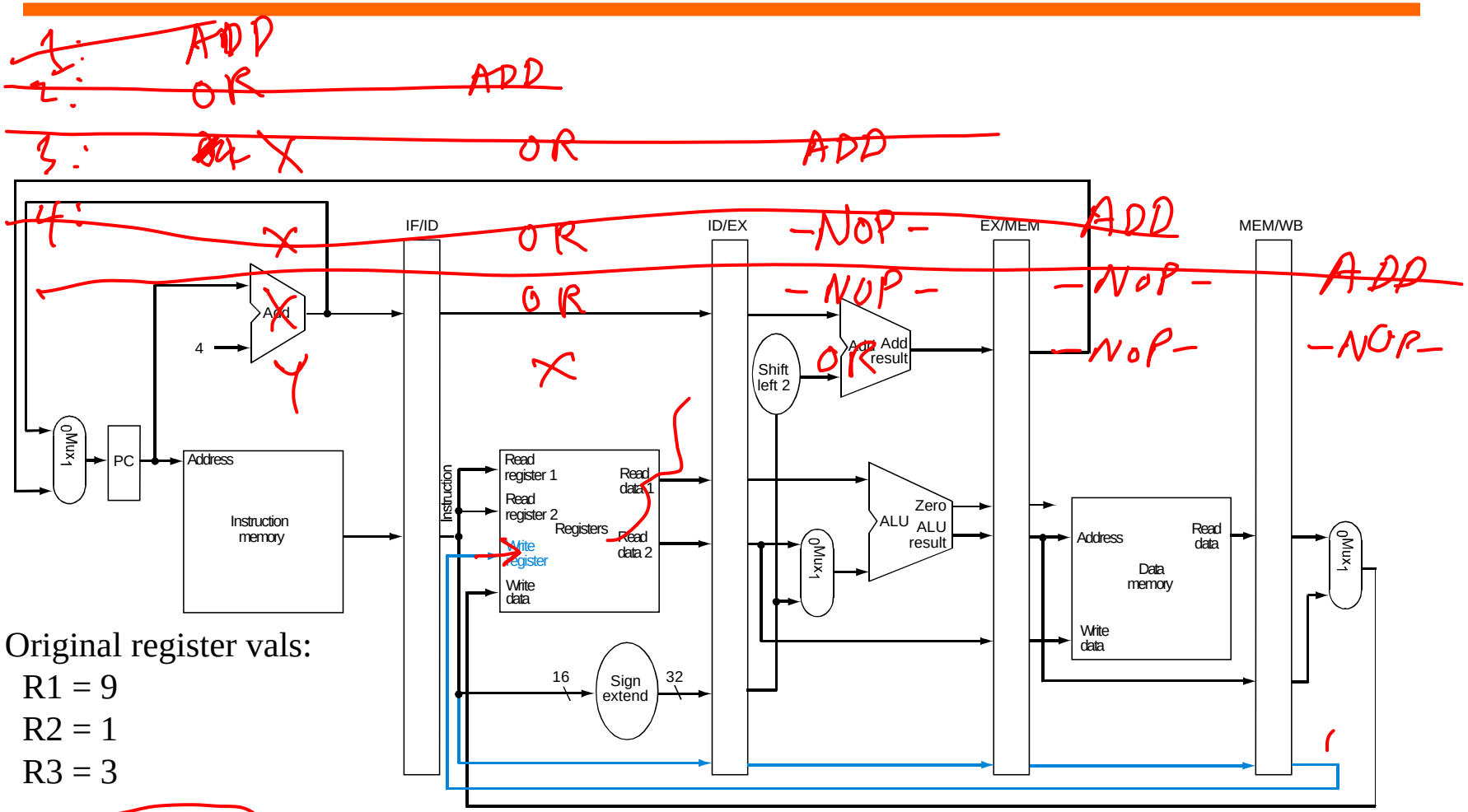
```
        JMP   LOOP
        …
LOOP:   ADD   R1, R2, R3
```

- Structural hazards
  - two instructions need access to the same resource
    - e.g., single memory shared for instruction fetch and load/store

Resolving Hazards: Pipeline Stalls

- Can resolve any type of hazard
  - data, control, or structural
- Detect the hazard
- Freeze the pipeline up to the dependent stage until the hazard is resolved

Example pipeline stall



Original register vals:

R1 = 9

R2 = 1

R3 = 3

ADD R1, R2, R3

OR R4, R1, R5

READ from R1

Write to R1

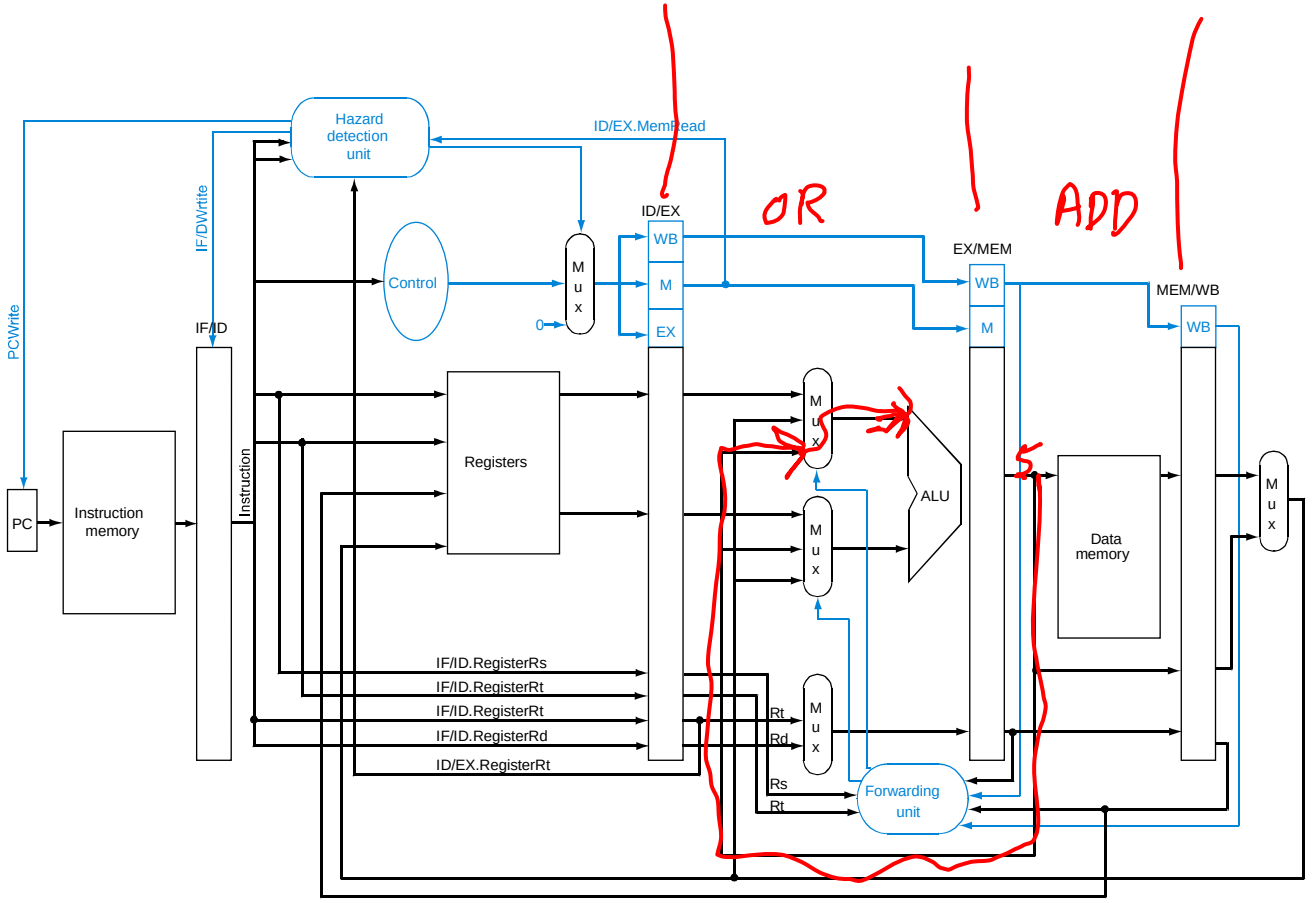
Can we avoid these stalls?

- If data is available elsewhere in the pipeline, there is no need to stall
  - Let's look for this on previous slide

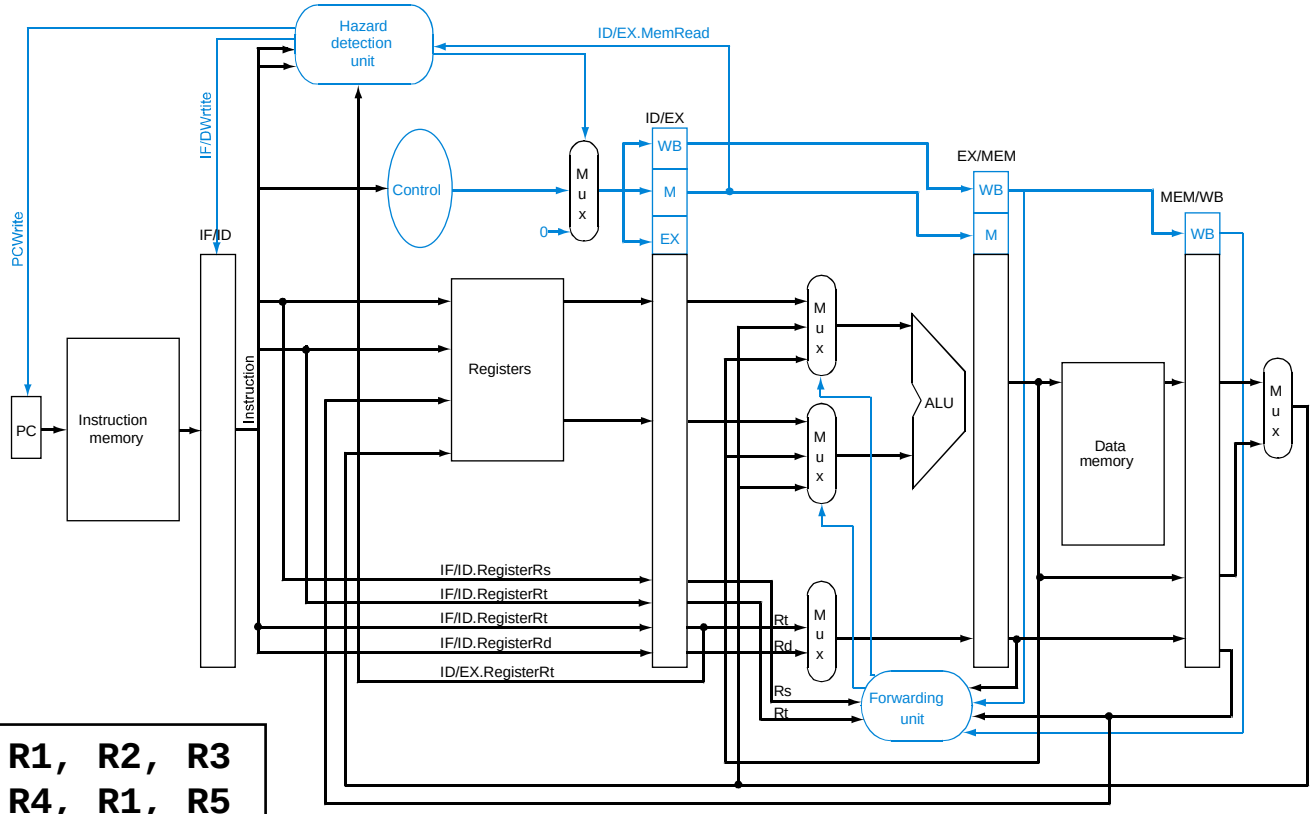
Resolving Hazards: Bypass (Forwarding)

- Detect condition
- Bypass (or forward) data directly to the consuming pipeline stage
- Bypass eliminates stalls for *single-cycle* operations
  - reduces longest stall to N-1 cycles for N-cycle operations

Add multiplexors in front of ALU that can accepted bypassed values



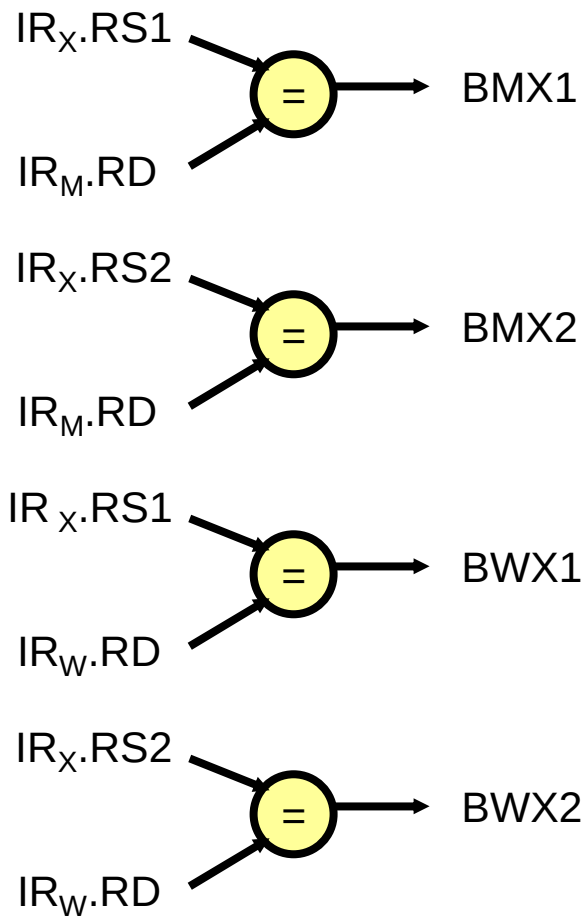
Example of bypassing



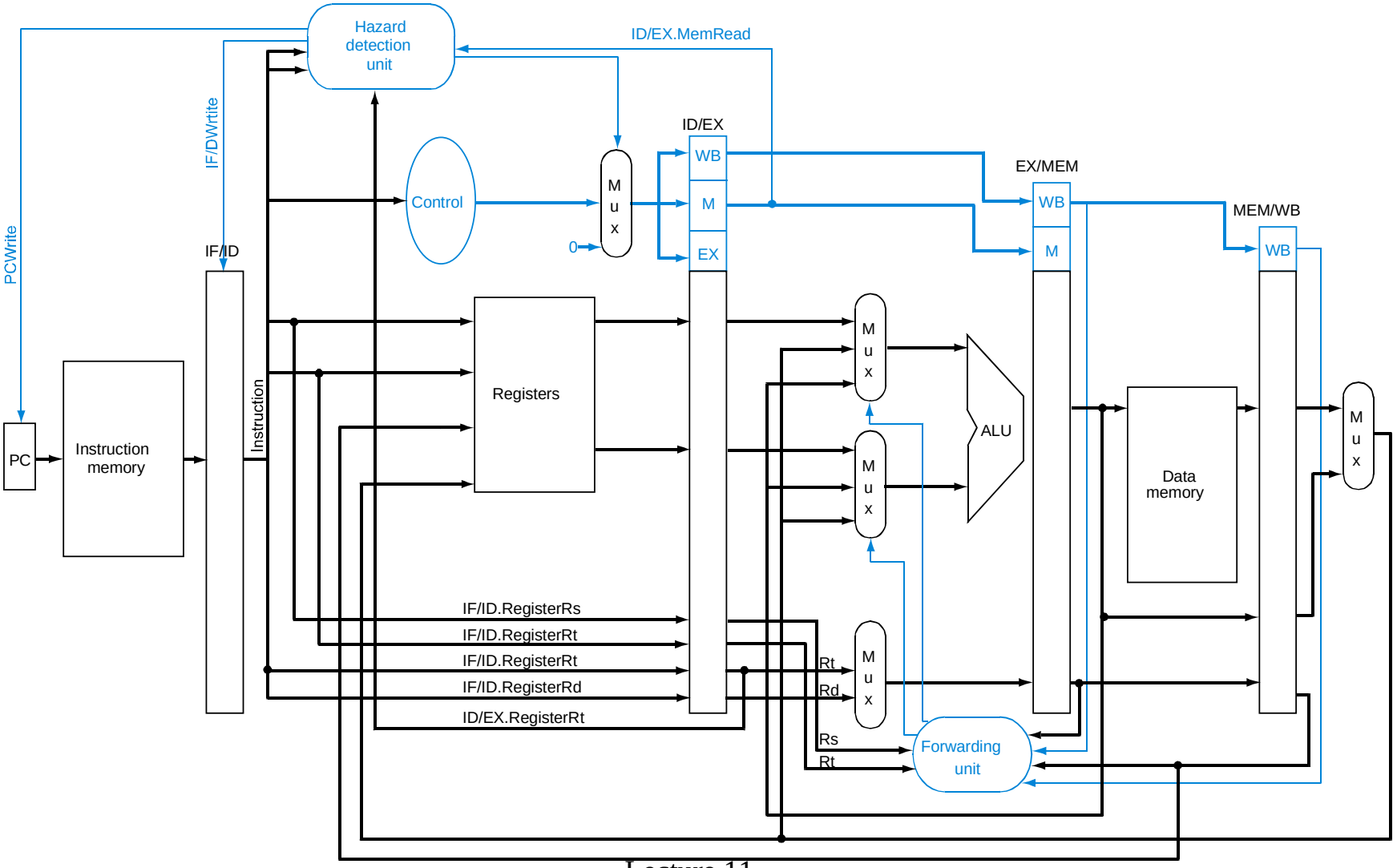

```
ADD     R1, R2, R3
ADD     R4, R1, R5
SUB     R5, R1, R6
XOR     R7, R8, R1
```

Control of Bypass (show with next figure)

- Compare source register fields of IR_X to destination register fields of IR_M and IR_W.
- If match and fields *active,* enable appropriate bypass path

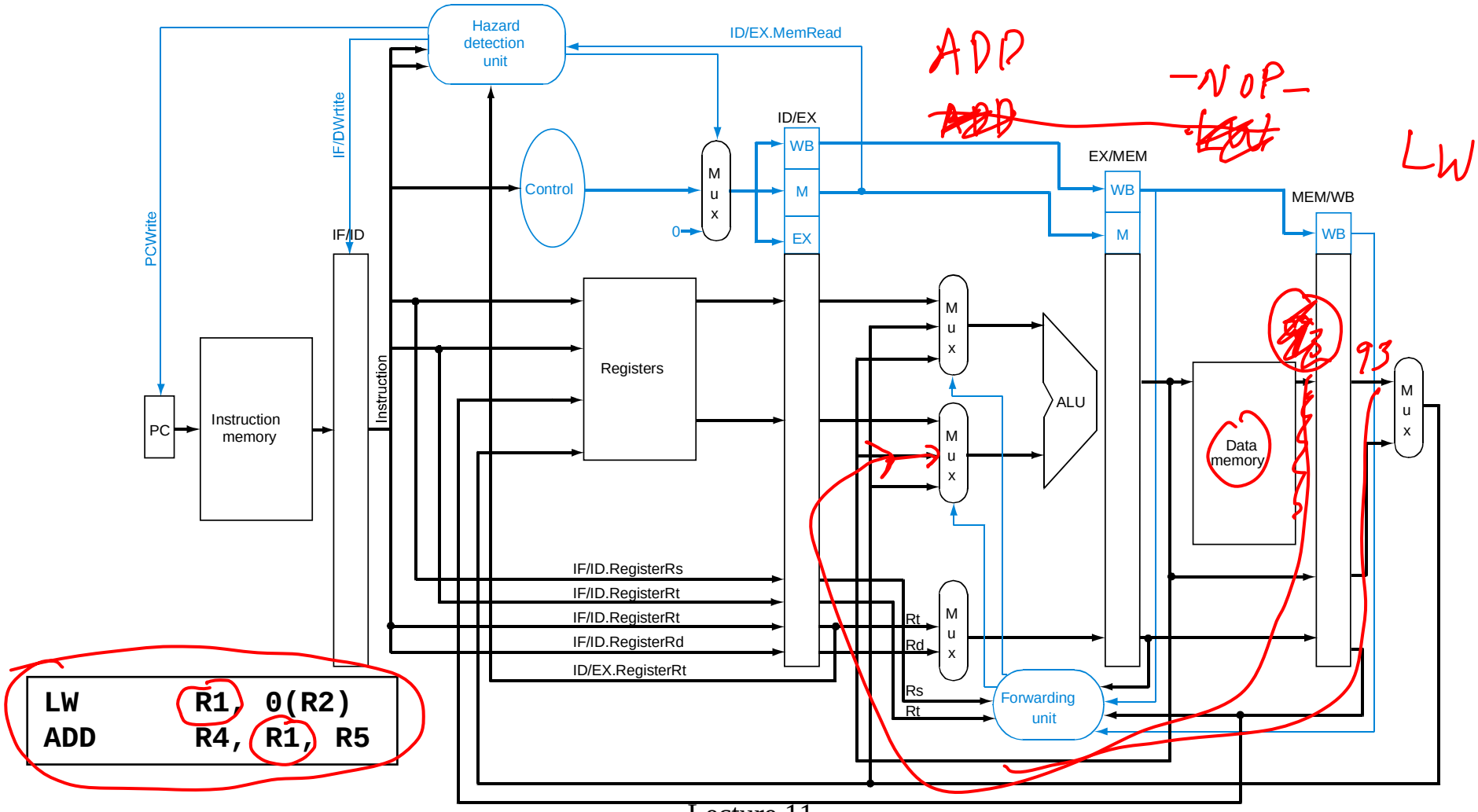


"Forwarding unit" compares register #'s



Memory data hazards

Example of memory data hazard



What can we do?

- Currently, we can't immediately use a value that has been loaded.

```
LW      R1, 0(R2)
ADD     R4, R1, R5
```

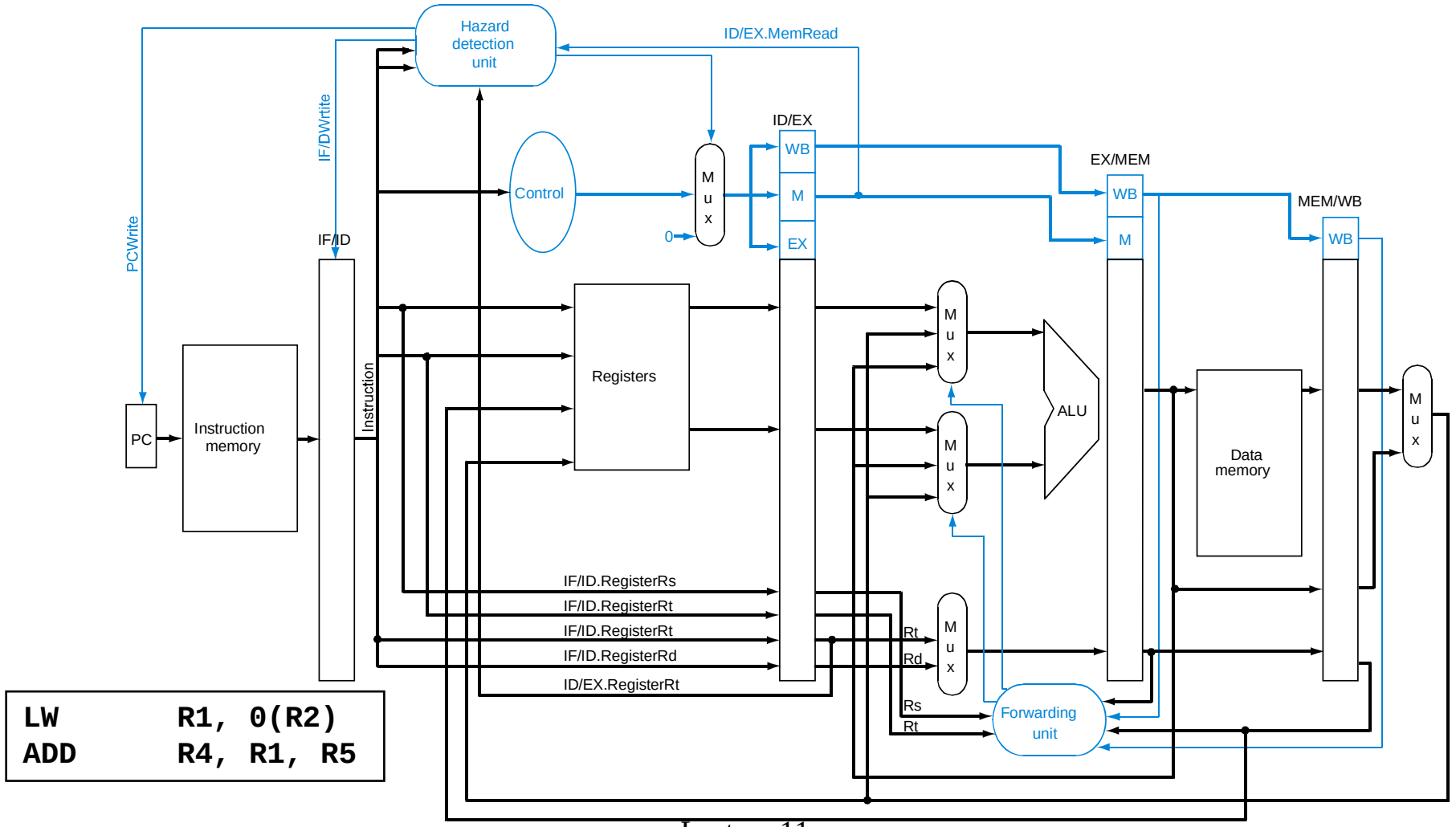
- But, it would work if we had a NOP…

```
LW      R1, 0(R2)
NOP
ADD     R4, R1, R5
```

Two choices

- Change the ISA manual:
  - You can't immediately use the result of a LOAD
- Change the hardware:
  - Detect the problem
  - Somehow make things work (but slower)
    - "Automatically" insert the NOP

The "Hazard detection unit" inserts a NOP



Summary

- Pipelining – work on multiple instructions at once
- Data Hazard detection
- Data Hazard avoidance (forwarding/bypassing)

- Next Time
  - Control hazards
  - Branch prediction
  - Reading assignment: P&H 6.4-6.6, 6.8 – 6.12