Georgia Tech Computer Science

CS2200
Systems and Networks
Spring 2024

Lecture 11: Pipeline Hazards

Alexandros (Alex) Daglis
School of Computer Science
Georgia Institute of Technology
adaglis@gatech.edu

Lecture slides adapted from Bill Leahy and Charles Lively of Georgia Tech

Announcements

- Don't forget your Midterm I tomorrow!
- During lab time: 6:30–7:45pm
- Go to your assigned lab section
- Will start on time, don't be late!

Hazards

Pipeline Hazards

- Structural hazards
  - Datapath/resource limitations
- Data hazards
  - Program limitations
- Control hazards
  - Program limitations

Bill's Mega-Sandwich Shop





Station 1
(place order)
New (5th order)
"What will you have?"

station II
(select bread)
4th order

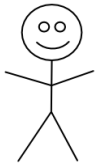
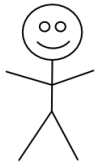
station III
(cheese)
3rd order

And sometimes fries on the side

station IV
(meat)
2nd order

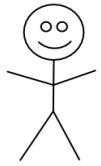
station V
(veggies)
1st order



Stay on the same order!

Stay on the same order!



No work for one cycle

Customers behind

Pipeline Stall

Customers ahead

Bill's Mega-Sandwich Shop





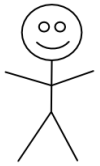
Station 1
(place order)
New (5th order)
"What will you have?"

station II
(select bread)
4th order

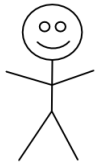
station III
(cheese)
3rd order

And sometimes fries on the side

station IV
(meat)
2nd order



station V
(veggies)
1st order



No work for one cycle

Pipeline resumed

Session ID: dag2200

Fries with that?

If the cheese person has to add fries to some orders, then the pipeline…

13% A. Continues to produce 1 sandwich per clock cycle

11% B. Produces more than 1 sandwich per clock cycle

76% C. Produces less than 1 sandwich per clock cycle

Bill's Mega-Sandwich Shop





Station 1
(place order)
New (5th order)
"What will you have?"

station II
(select bread)
4th order

station III
(cheese)
3rd order

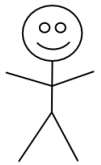
And sometimes fries on the side

station IV
(meat)
2nd order

station V
(veggies)
1st order



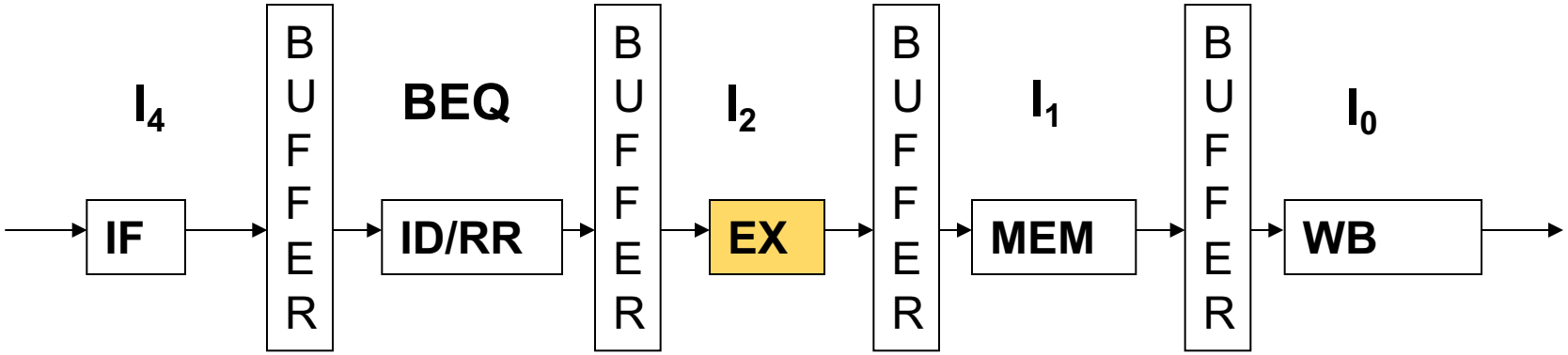




This is a structural hazard – not enough resources

The effect:
➔ Introduces “bubbles” into the pipeline
➔ Reduces efficiency

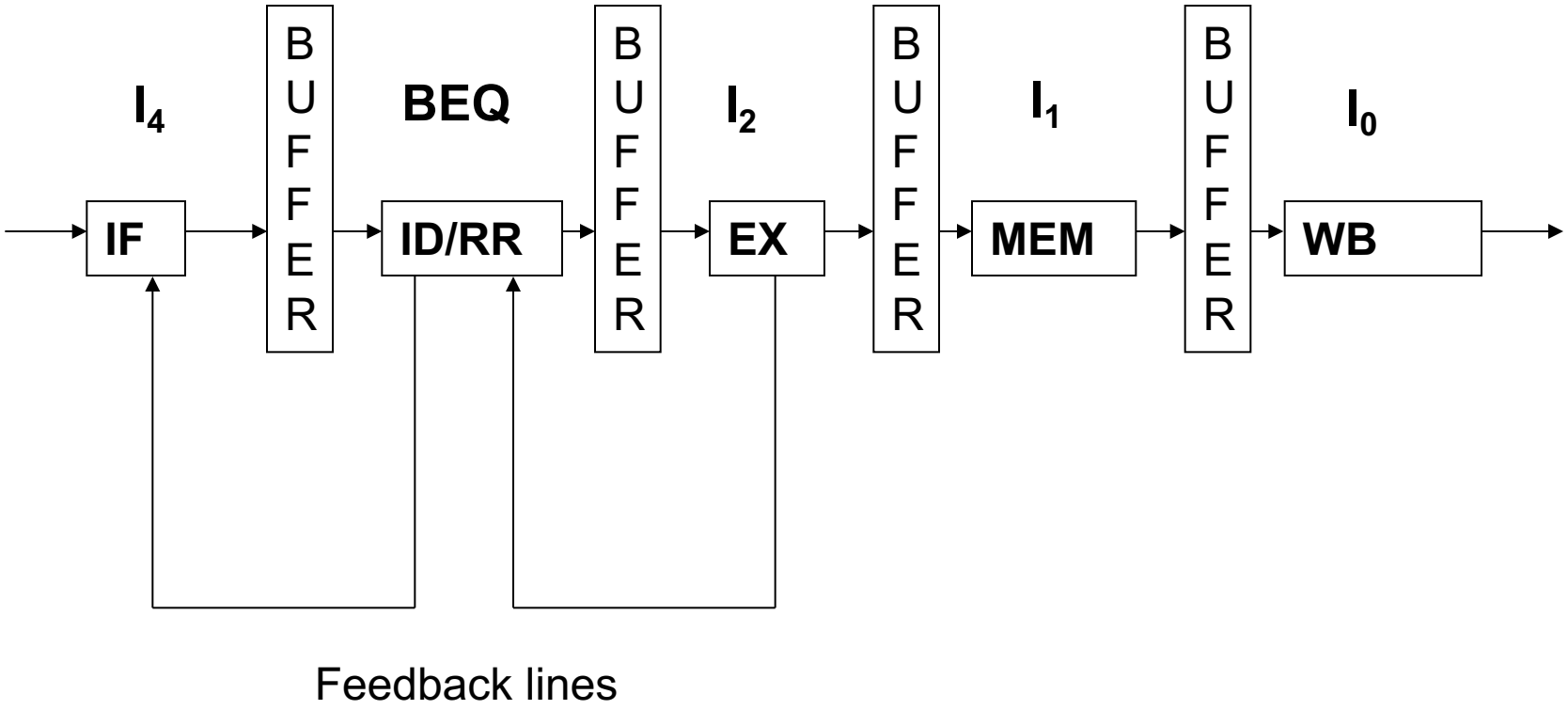
Structural Hazard



- Potentially need two arithmetic ops
  - A – B (always needed)
  - PC + offset (sometimes needed)
- Where is this going to mean trouble?

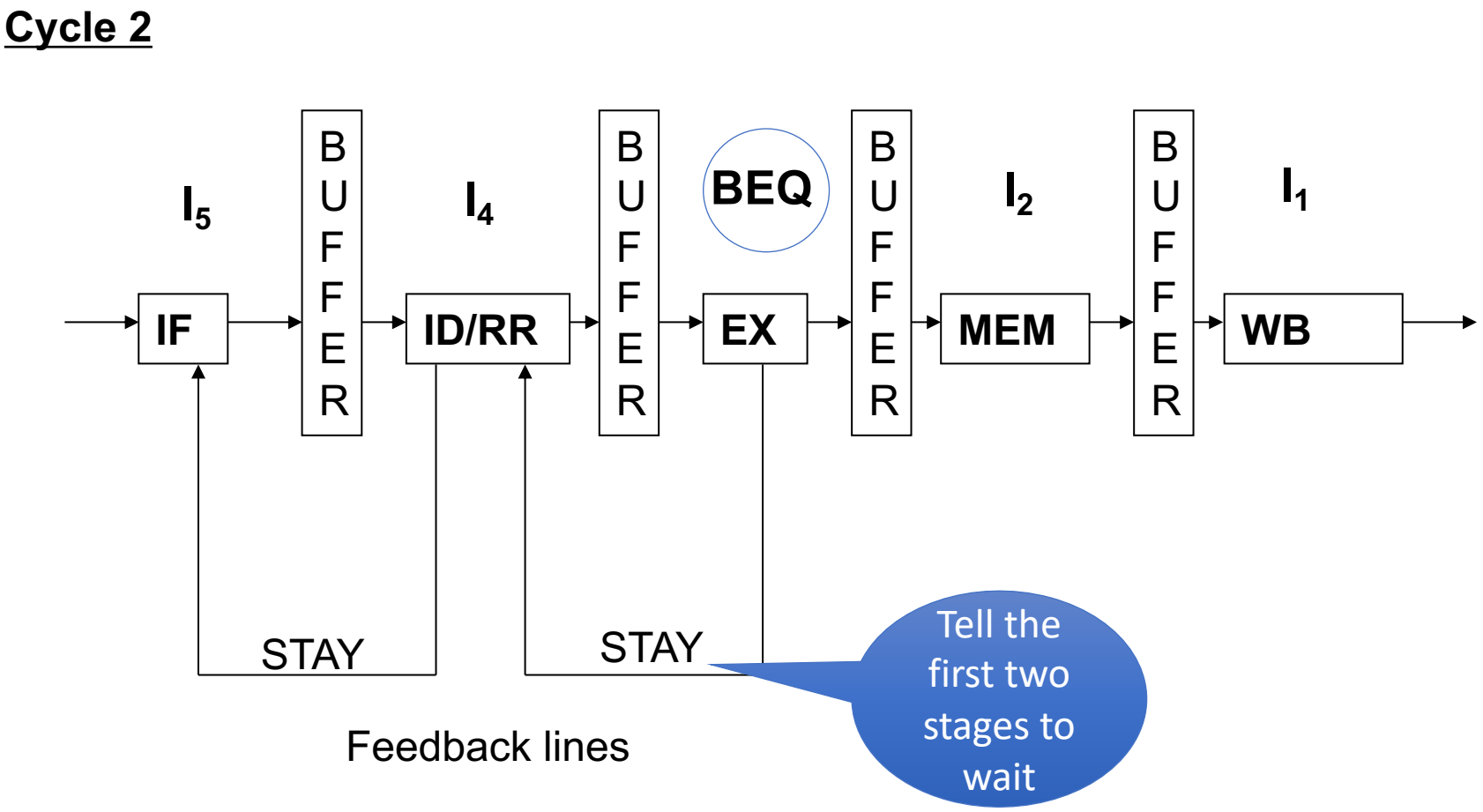
Structural Hazard

Cycle 1



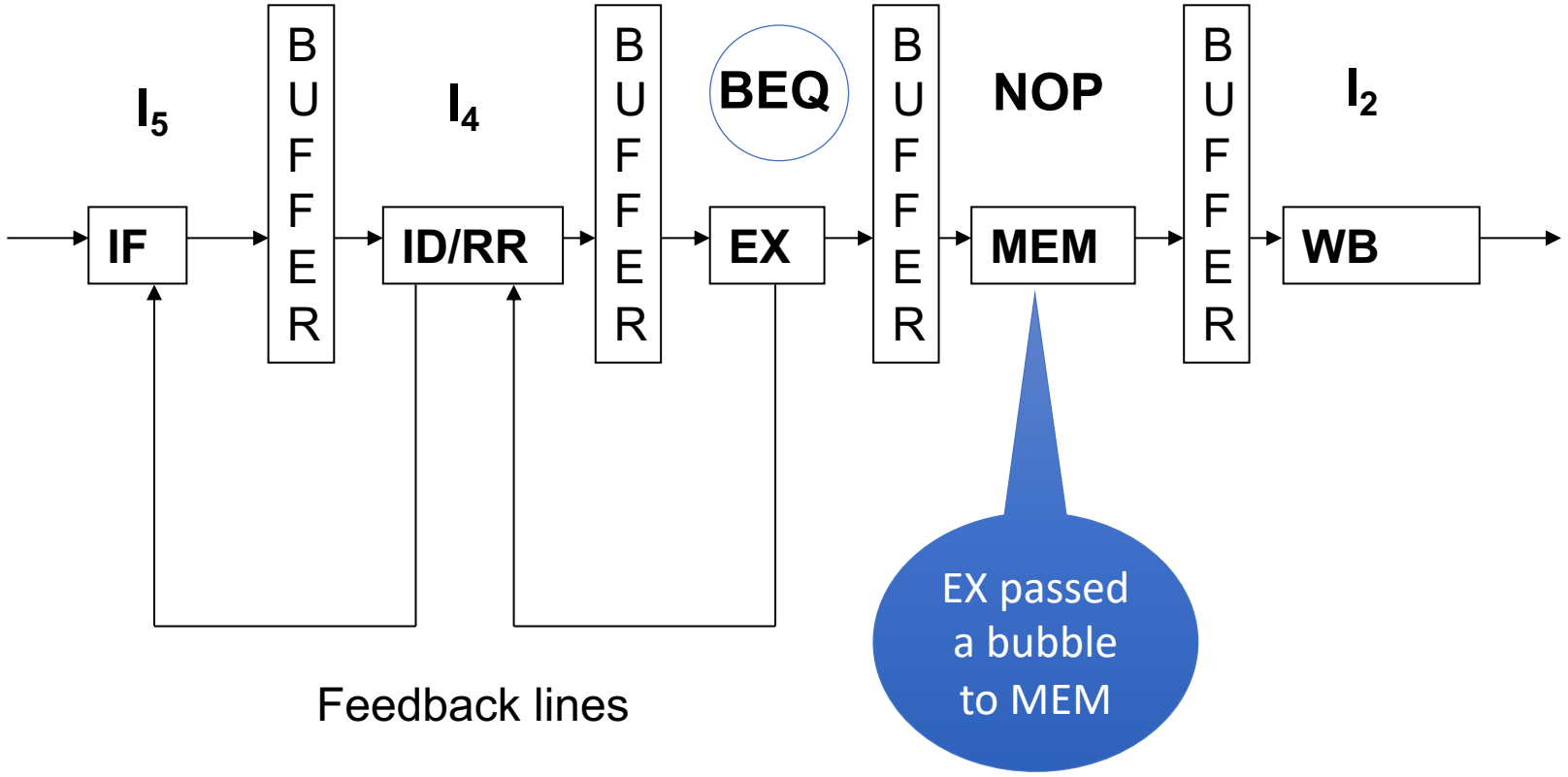
Feedback lines

If BEQ branches, we will need to use the ALU twice!



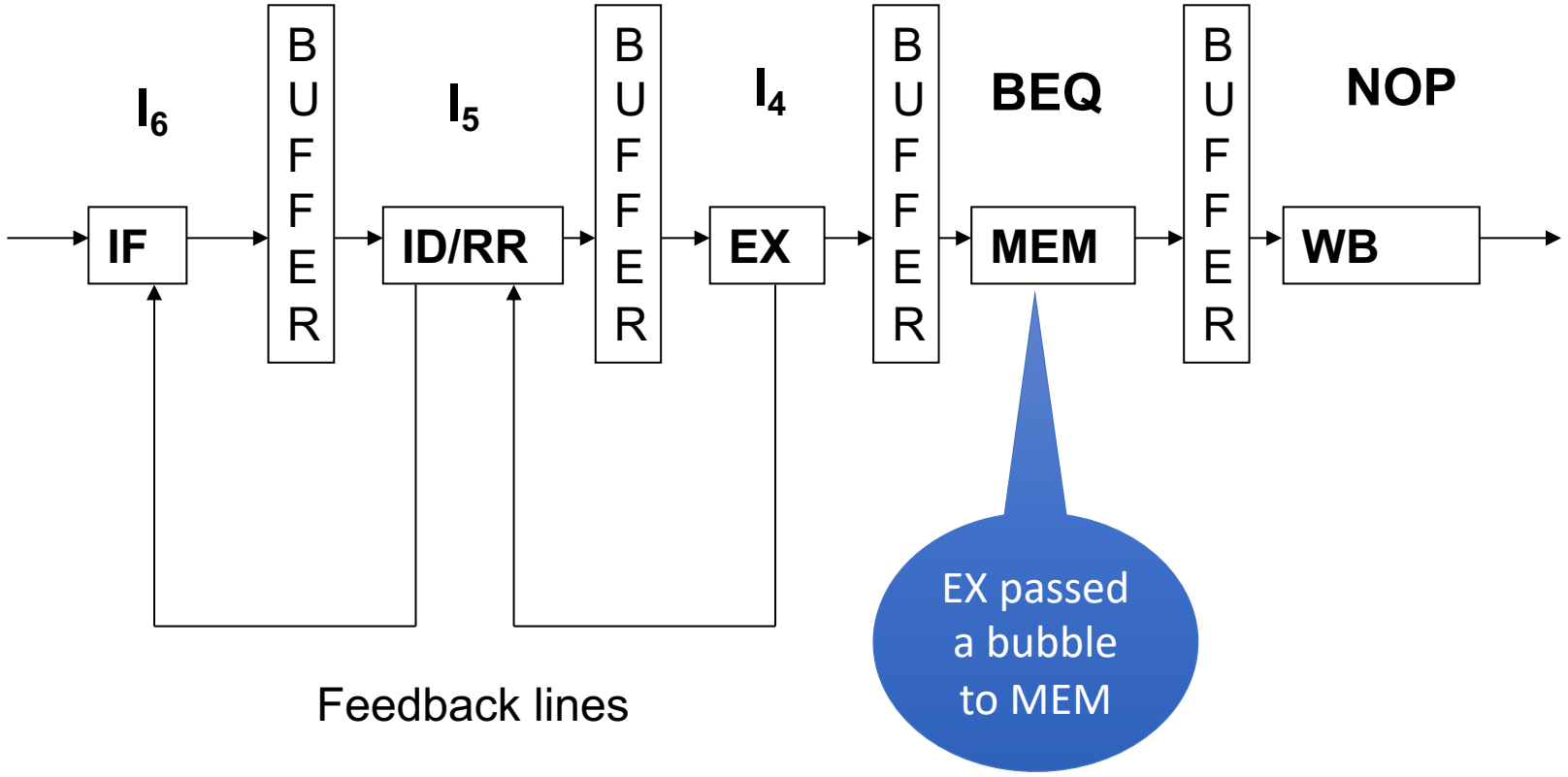
Pipeline resumes with a bubble

Cycle 3



But life goes on

Cycle 4



Bill's Mega-Sandwich Shop





Station 1
(place order)
New (5th order)
"What will you have?"

station II
(select bread)
4th order

station III
(cheese)
3rd order

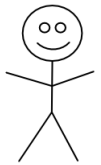
And sometimes fries on the side

station IV
(meat)
2nd order

station V
(veggies)
1st order





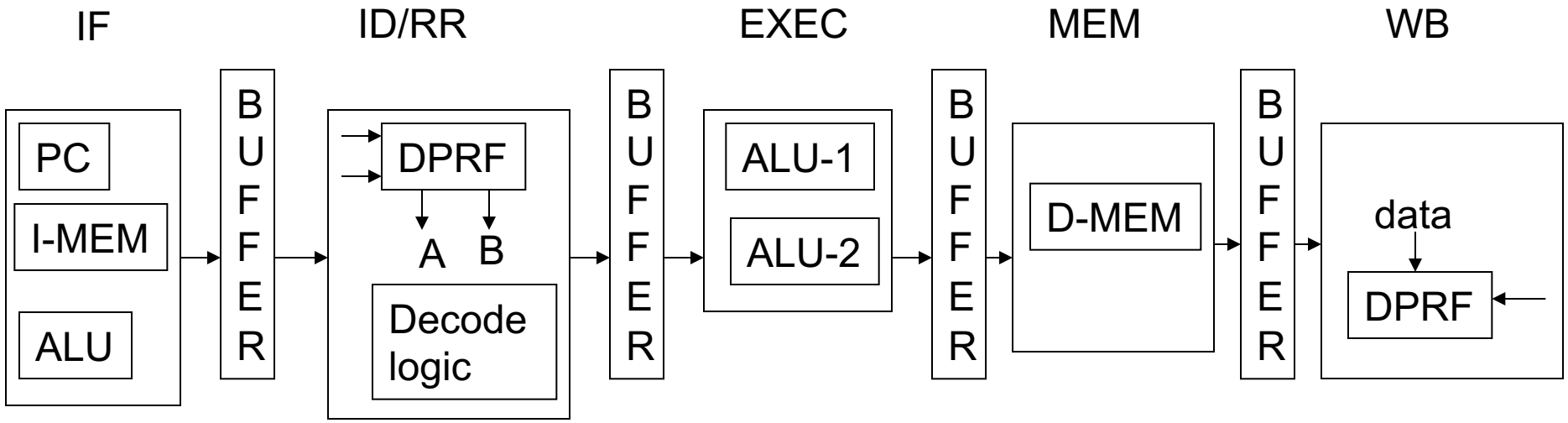
Cheese guy

Fries guy

But we can also add more “hardware”!

Throw hardware at it!

- So we add another ALU to the EX stage

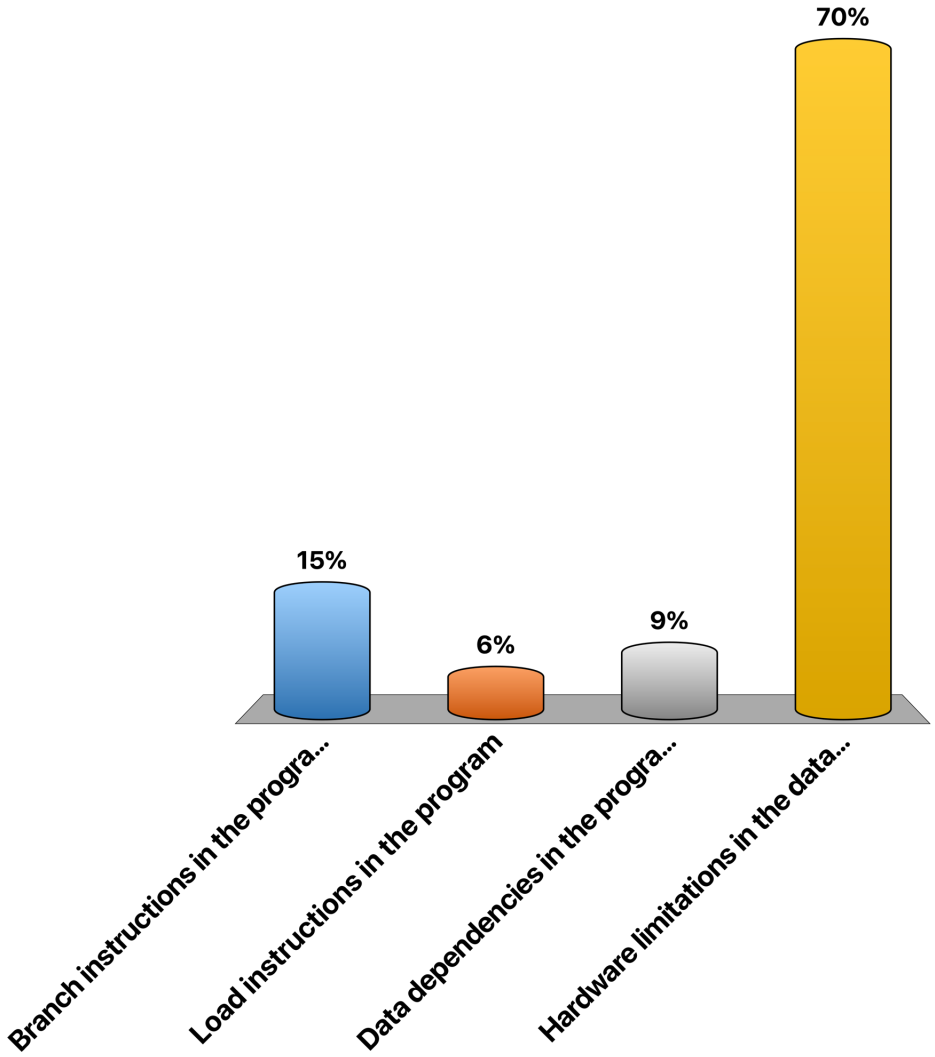


Performance problems due to structural hazards can be addressed by adding more hardware

Session ID: dag2200

Pipeline structural hazards are caused by…

A. Branch instructions in the program
B. Load instructions in the program
C. Data dependencies in the program
D. Hardware limitations in the datapath



Data Hazard

RAW

I_1: R1 ← R2 + R3

I_2: R4 ← R1 + R5

True dependency

WAR

I_1: R4 ← R1 + R5

I_2: R1 ← R2 + R3

Anti dependency

WAW

I_1: R1 ← R4 + R5

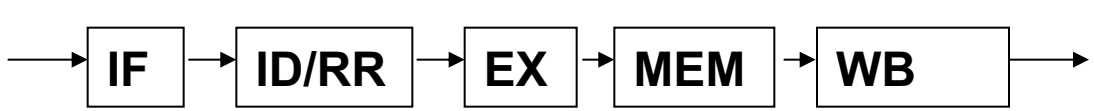
I_2: R1 ← R2 + R3

Output dependency

Dealing with a RAW dependency

I_1: R1 ← R2 + R3

I_2: R4 ← R1 + R5

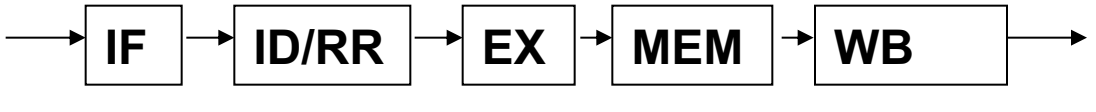


Cycle 1

At this point, R1 hasn't even been computed

I_2 I_1

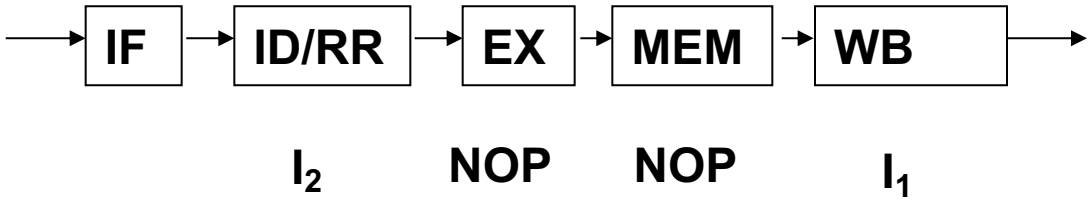
If we don't stall the pipeline here, the execution of I2 will result in a semantic inconsistency – wrong data values will be loaded



Cycle 2

I_2 I_1

Fix – introduce bubbles



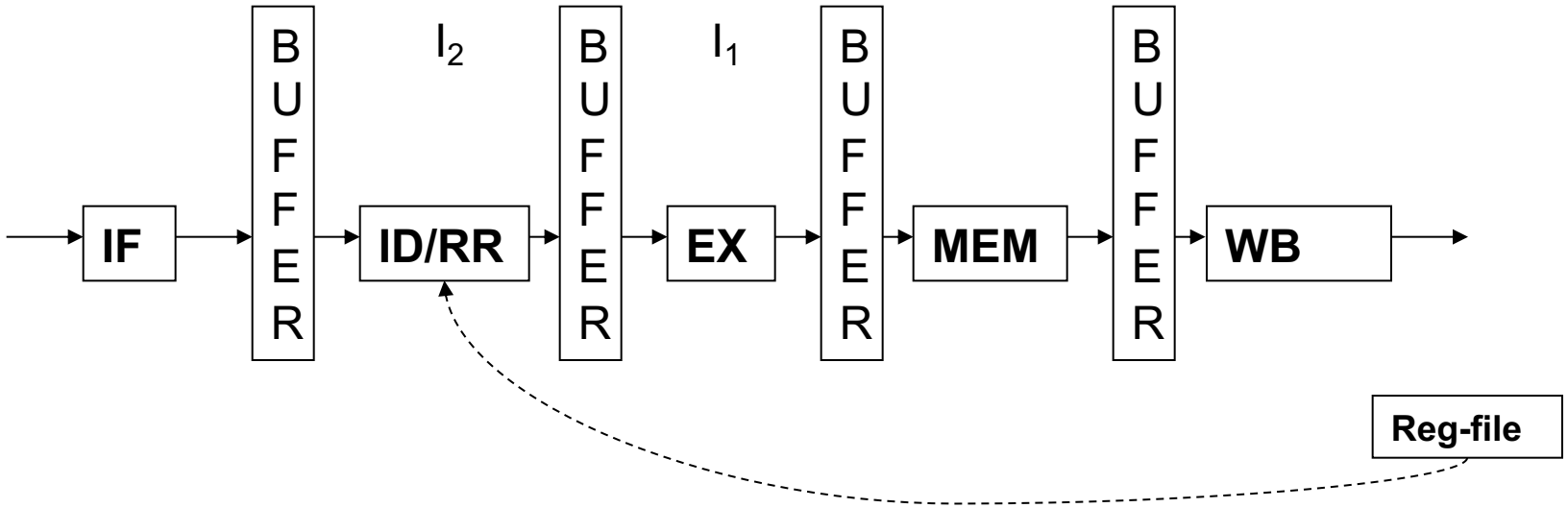
- We need some hardware help to know when to introduce bubbles
- So we're going to add a "busy" bit to the register file
- ID/RR sets it and WB clears it

Register file

	B
	B
	B
	B
	B
	B
	B
	B

RAW Hazard – Cycle 1

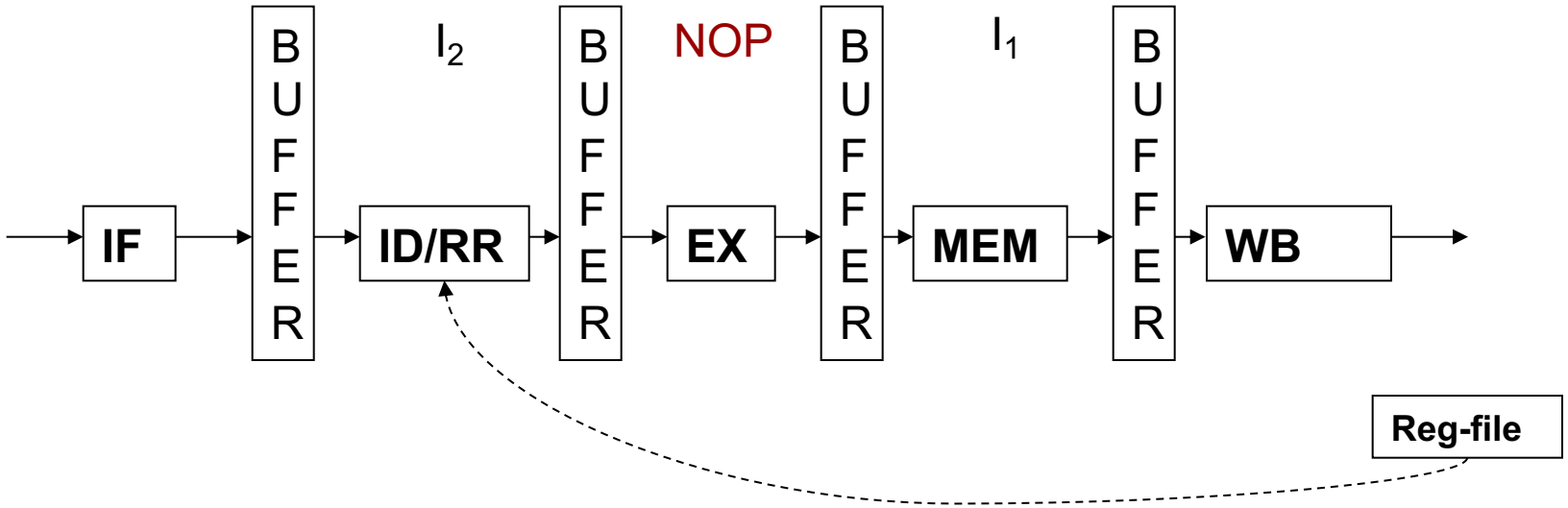
I_1: R1 ← R2 + R3
I_2: R4 ← R1 + R5



R1 not ready to be read

RAW Hazard – Cycle 2

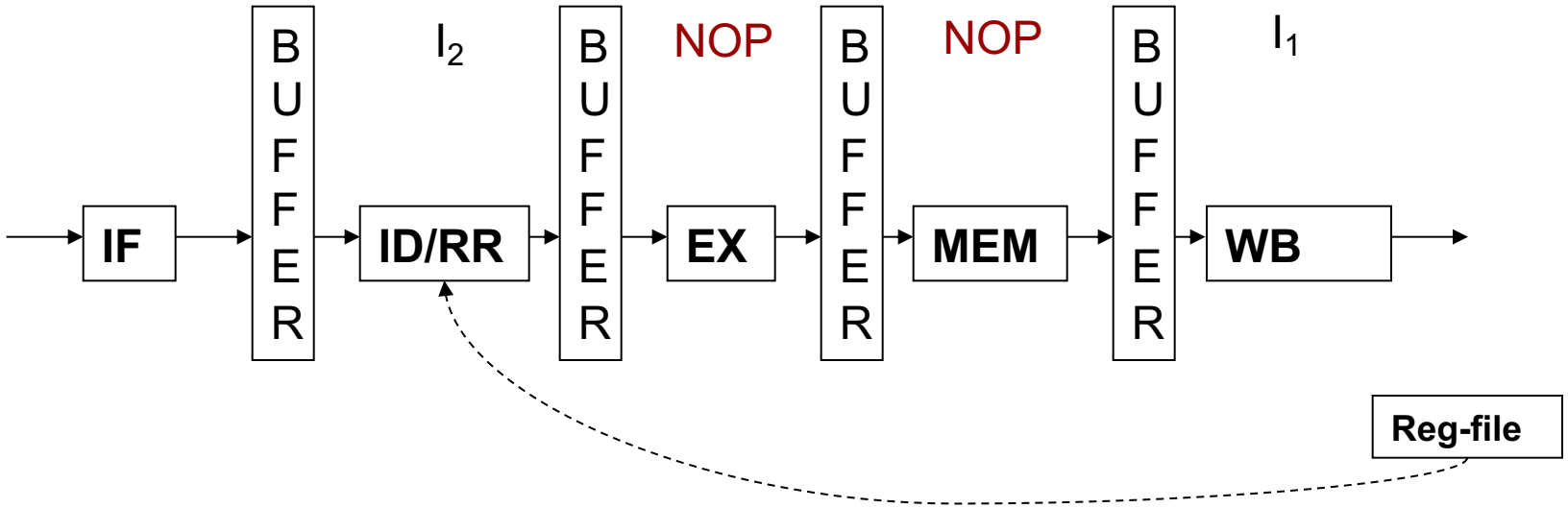
I_1: R1 ← R2 + R3
I_2: R4 ← R1 + R5



R1 not ready to be read

RAW Hazard – Cycle 3

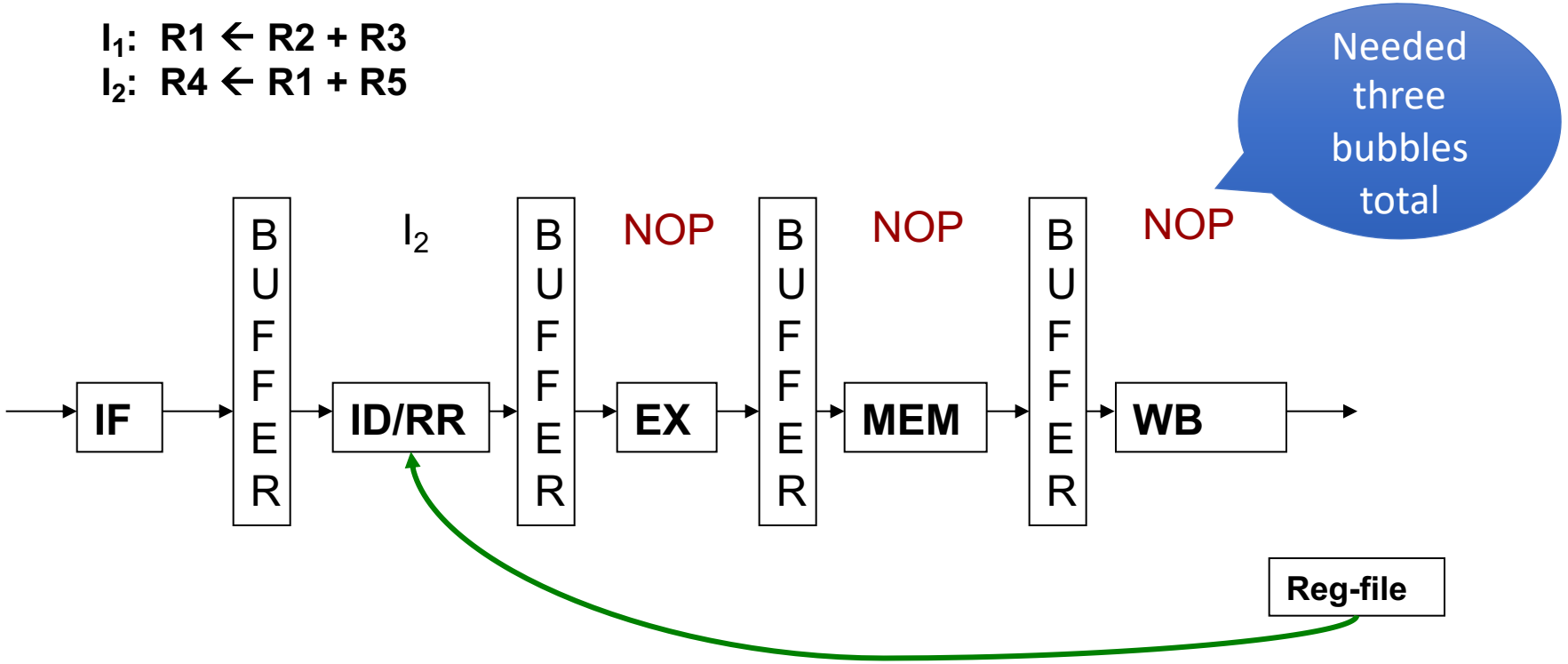
I_1: R1 ← R2 + R3
I_2: R4 ← R1 + R5



R1 not ready to be read

RAW Hazard – Cycle 4

I_1: R1 ← R2 + R3
I_2: R4 ← R1 + R5



R1 is ready to be read

A question…

I_1: R1 ← R2 + R3
I_2: R4 ← R4 + R3
I_3: R5 ← R5 + R3
I_4: R6 ← R1 + R6

- Say we insert 2 instructions between "definition" and "use" of a register
- Does this help to reduce "bubbles"?

With 2 instruction separation...

I_1: R1 ← R2 + R3
I_2: R4 ← R4 + R3
I_3: R5 ← R5 + R3
I_4: R6 ← R1 + R6

6% A. Continue to have 3 bubbles in pipeline

20% B. Reduce the number of bubbles to 0

56% C. Reduce the number of bubbles to 1

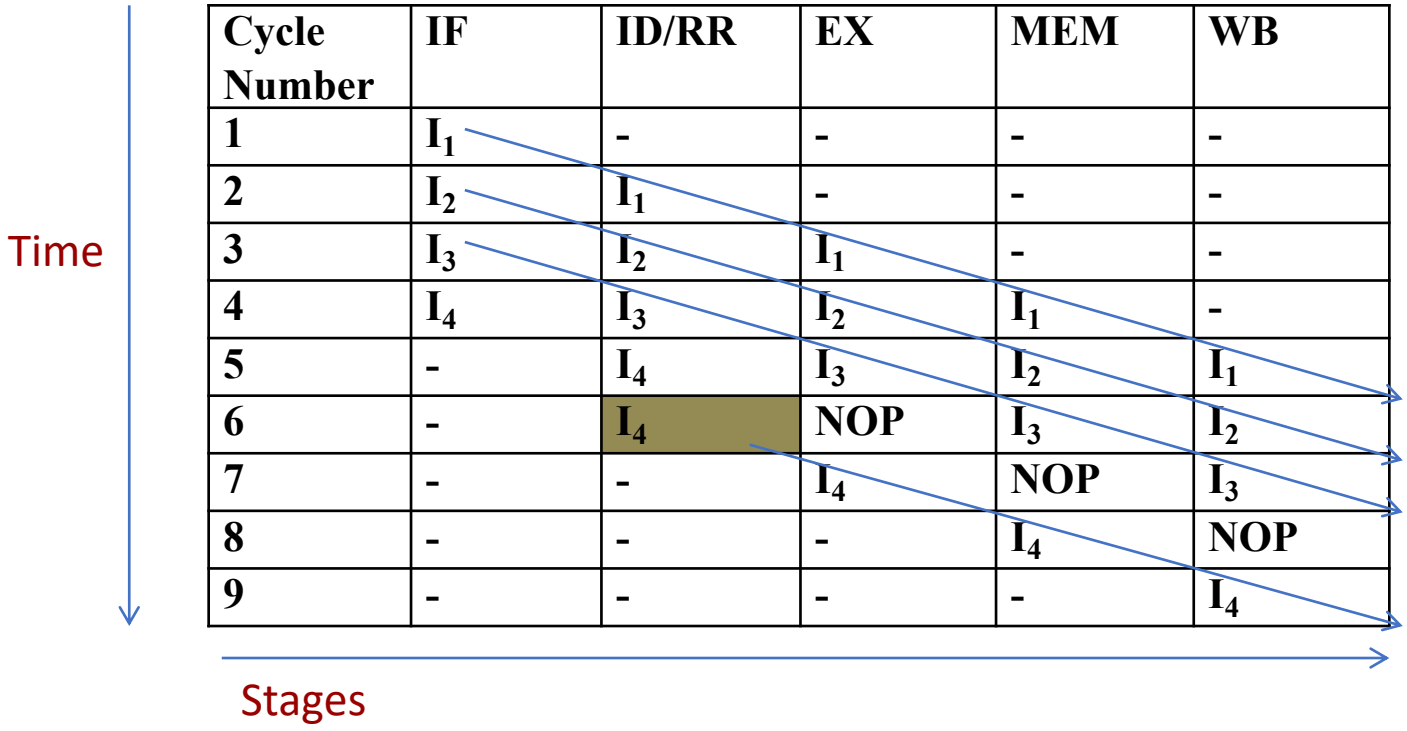
18% D. Reduce the number of bubbles to 2

Two instruction separation

I_1: R1 ← R2 + R3
I_2: R4 ← R4 + R3
I_3: R5 ← R5 + R3
I_4: R6 ← R1 + R6

- We can restructure the program to reduce "bubbles"
- This is something we expect modern compilers to be able to do for us.

Waterfall Diagram



Cycle Number	IF	ID/RR	EX	MEM	WB
1	I_1	-	-	-	-
2	I_2	I_1	-	-	-
3	I_3	I_2	I_1	-	-
4	I_4	I_3	I_2	I_1	-
5	-	I_4	I_3	I_2	I_1
6	-	I_4	NOP	I_3	I_2
7	-	-	I_4	NOP	I_3
8	-	-	-	I_4	NOP
9	-	-	-	-	I_4

I_1: R1 <- R2 + R3

I_2: R4 <- R4 + R3

I_3: R5 <- R5 + R3

I_4: R6 <- R1 + R6

If we can't restructure our program?

- Can we throw hardware at the problem and eliminate bubbles?

Register forwarding

- Let's add an RP (read-pending) signal associated with each register to indicate an instruction is stalled in ID/RR for this register value
- While B is a flip-flop, RP is a signal line

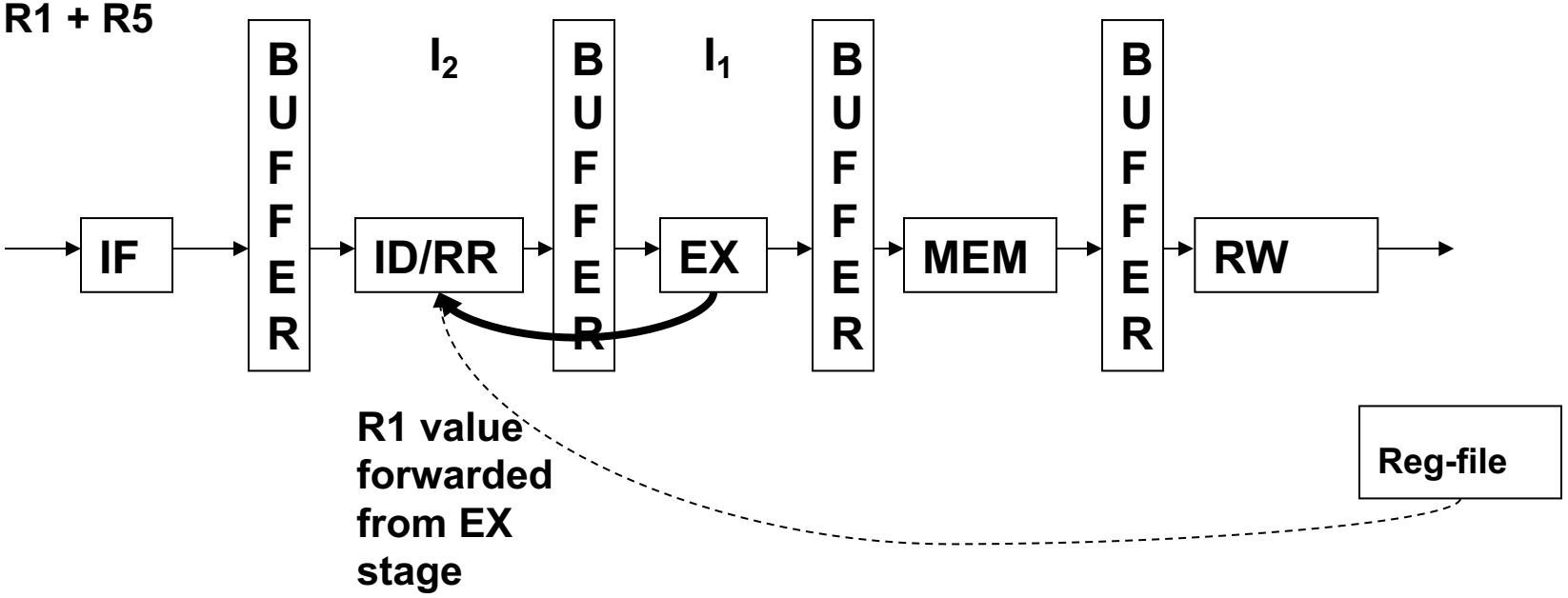
- It's a little more complicated than the next diagram shows. (ID/RR will advertise which registers are needed on two address lines and the first pipeline stage that has that Rx value can gate its result onto a forwarding bus.)

Register file

	B	RP
	B	RP
	B	RP
	B	RP
	B	RP
	B	RP
	B	RP
	B	RP

Forwarding example

I_1: R1 ← R2 + R3
I_2: R4 ← R1 + R5

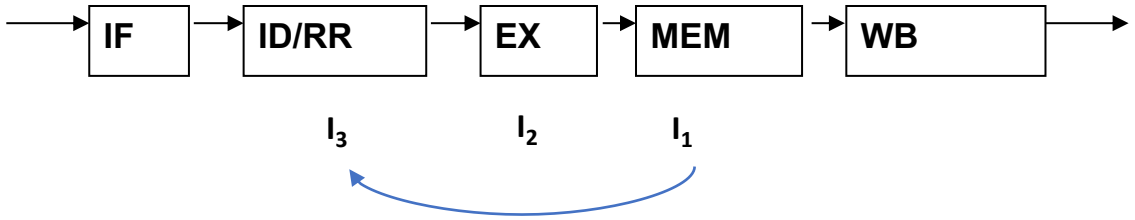


Forwarding logic in ID/RR:
If RP == 1 then
  use forwarded value
else
  read entry from DPRF

R1 not ready to be read

What if an instruction intervenes?

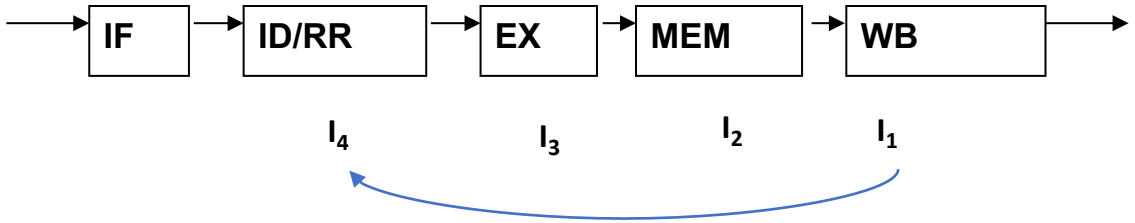
I_1: R1 ← R2 + R3
I_2: R4 ← R4 + R3
I_3: R6 ← R1 + R6



Now we need to forward from MEM to ID/RR

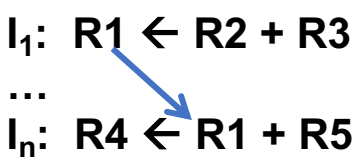
What if two instructions intervene?

I_1: R1 ← R2 + R3
I_2: R4 ← R4 + R3
I_3: R5 ← R5 + R3
I_4: R6 ← R1 + R6



Now we need to forward from WB to ID/RR

Bubbles introduced by RAW



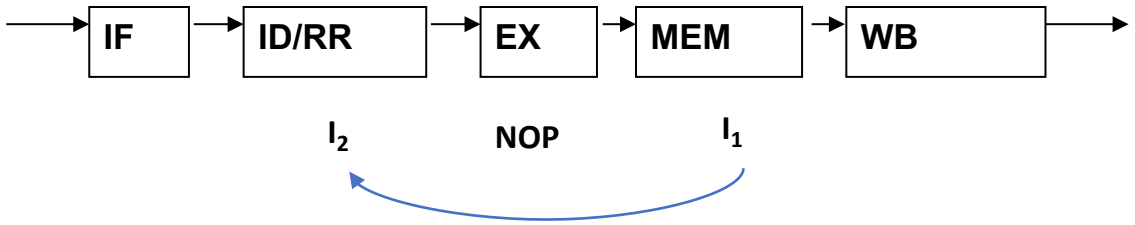
Number of unrelated instructions Between I_1 and I_n	Number of bubbles Without forwarding	Number of bubbles With forwarding
0	3	0
1	2	0
2	1	0
3 or more	0	0

… but forwarding cannot eliminate ALL bubbles

LW instructions are special

I_1: LW R1, 0(R2)

I_2: R4 ← R1 + R5



Even with data forwarding, we have to insert a bubble!

Why? We don't get the data until I1 gets to MEM!

So LW changes the table

RAW Hazard

I_1: R1 ← R2 + R3
…
I_2: R4 ← R1 + R5

Number of unrelated instructions Between I_1 and I_2	Number of bubbles Without forwarding	Number of bubbles With forwarding
0	3	0
1	2	0
2	1	0
3 or more	0	0

RAW Hazard

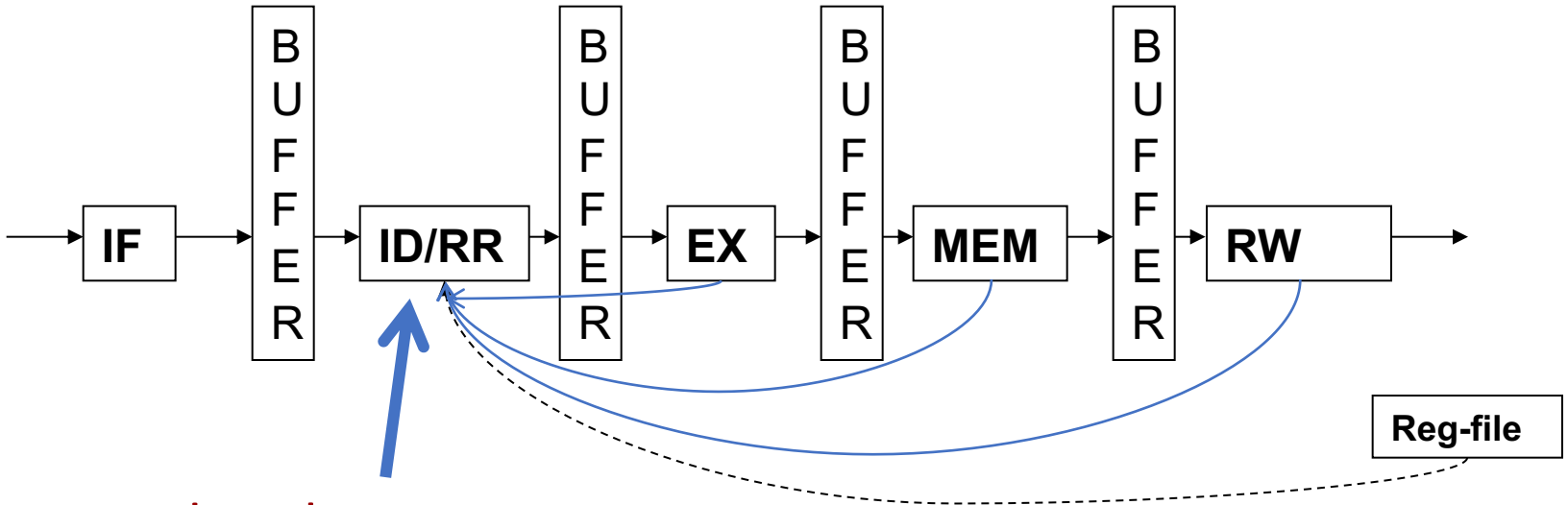
I_1: LW R1, 0(R2)
…
I_2: R4 ← R1 + R5

Number of unrelated instructions Between I_1 and I_2	Number of bubbles Without forwarding	Number of bubbles With forwarding
0	3	1
1	2	0
2	1	0
3 or more	0	0

Generalized register forwarding

To minimize stalls due to data hazards, we need forwarding data lines from ALL stages

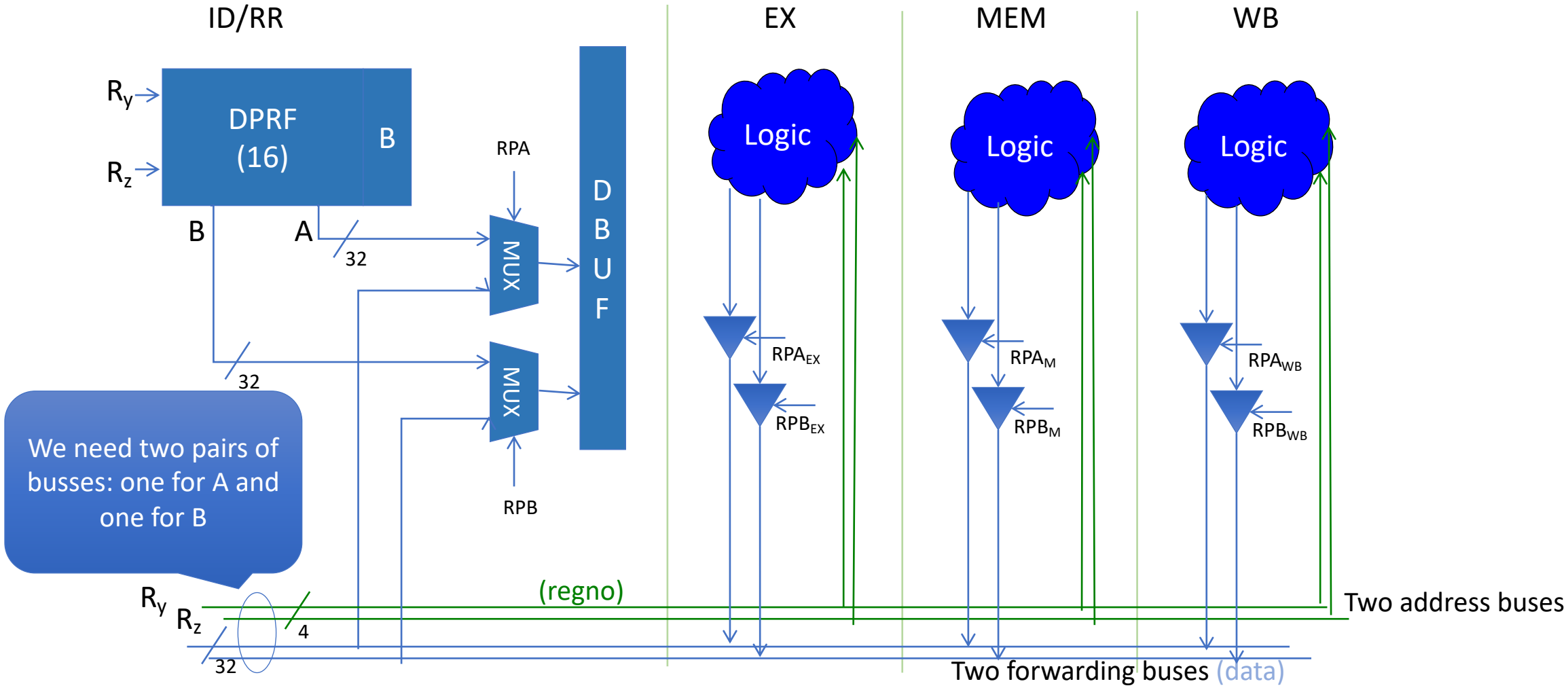
- Due to cost & complexity, hardware designer may decide to implement subset of forwarding paths



And we have two source operands to be fed

R1 not ready to be read

An example of forwarding implementation



What's going on?

This is a rough outline of what happens in the previous diagram:

1. ID/RR puts the register numbers used for A and B on the two address busses
2. The other stages compare any results for their destination register with the register numbers on the two register busses
3. If the destination register matches a register on a register forwarding bus, the stage opens the driver to allow the destination value it has onto the appropriate A and/or B forwarding data bus and signals ID/RR that a match has been found on the A or B operand via RPA/RPB.
4. There is circuitry (sort of like interrupt priority) that makes sure the first stage that matches on the forwarding register bus is the one that gets to use the forwarding data bus (Why?)
5. If the RPA/RPB signals are sent, multiplexers in the ID/RR stage select the the data from the corresponding A or B forwarding data bus instead of the register file

Other data hazards

WAR

I_1: R2 ← R1 + R3
I_2: R1 ← R4 + R5

Fortunately, this isn't a problem since we've already read the registers for I_1 into DBUF before I_2 enters the ID/RR stage (let alone the WB stage).

WAW

I_1: R1 ← R2 + R3
I_2: R1 ← R4 + R5

I_2 will reach the WB stage after I_1, so the updates to R1 will be applied in order

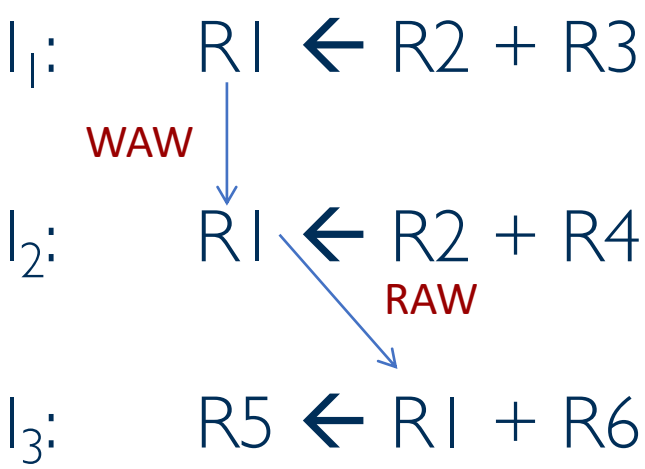
So WAR and WAW hazards and not really a concern for our simple pipeline

- But remain relevant for more complex pipelines (out-of-order execution, beyond cs2200's scope)

But why do we see WAW anyway?

- Shouldn't the compiler get rid of useless writes?
- There are unexpected code sequences
  - Exception handling code
- Check out Sec 5.13.2 page 200 and Sec 5.15.4 if you're curious

What hazards?



Register file

	B	RP
	B	RP
	B	RP
	B	RP
	B	RP
	B	RP
	B	RP
	B	RP

Must ensure that I_3 gets the correct value of R1 (from I_2 instead of I_1)

- Without forwarding, need to make sure I_1 does not clear the busy bit – I_2 should do that
  - i.e., I3 *cannot* move past ID/RR before I2 is also done with WB
- With forwarding, need to make sure R1 value is forwarded from EX stage (where I_2 is), not from MEM (where I_1 is)