CSE261: Computer Architecture

13. Processor (5): Pipelining #2

Seongil Wi

Department of Computer Science and Engineering

HW2

- MIPS Single-cycle CPU Implementation
- Build a working model of a MIPS processor that can simulate a 5-stage in a single clock cycle
- Due: Nov 26, 11:59 PM

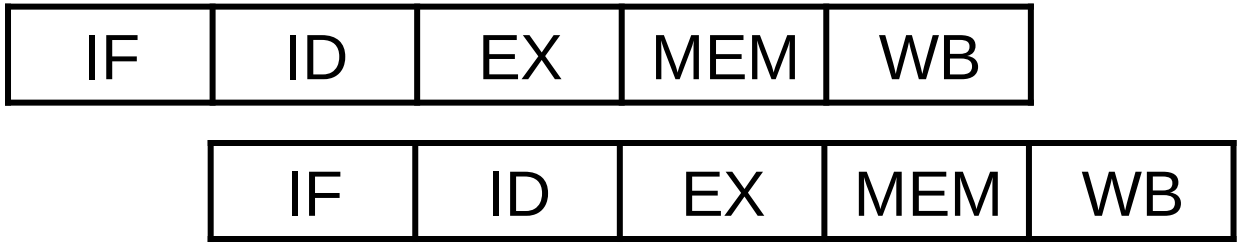
Motivation Example

```
add $s0, $t0, $t1
sub $t2, $s0, $t3
```

Pipelining is used for execution. Any problems?

Motivation Example

```
add $s0, $t0, $t1
sub $t2, $s0, $t3
```



Motivation Example

The calculated data of $s0 is available at this time

```
add $s0, $t0, $t1
sub $t2, $s0, $t3
```

IF	ID	EX	MEM	WB
IF	ID	EX	MEM	WB

The data of $s0 is needed at this time

Pipelining hazard: situations that prevent starting the next instruction in the next cycle

Pipelining Hazards

Pipelining Hazards

Situations that prevent starting the next instruction in the next cycle

- ***What types of hazards are there?***
- ***How can these hazards be resolved?***

Pipelining Hazards

Situations that prevent starting the next instruction in the next cycle

Hazard #1: Structural hazard	**Hazard #2:** Data hazard	**Hazard #3:** Control hazard

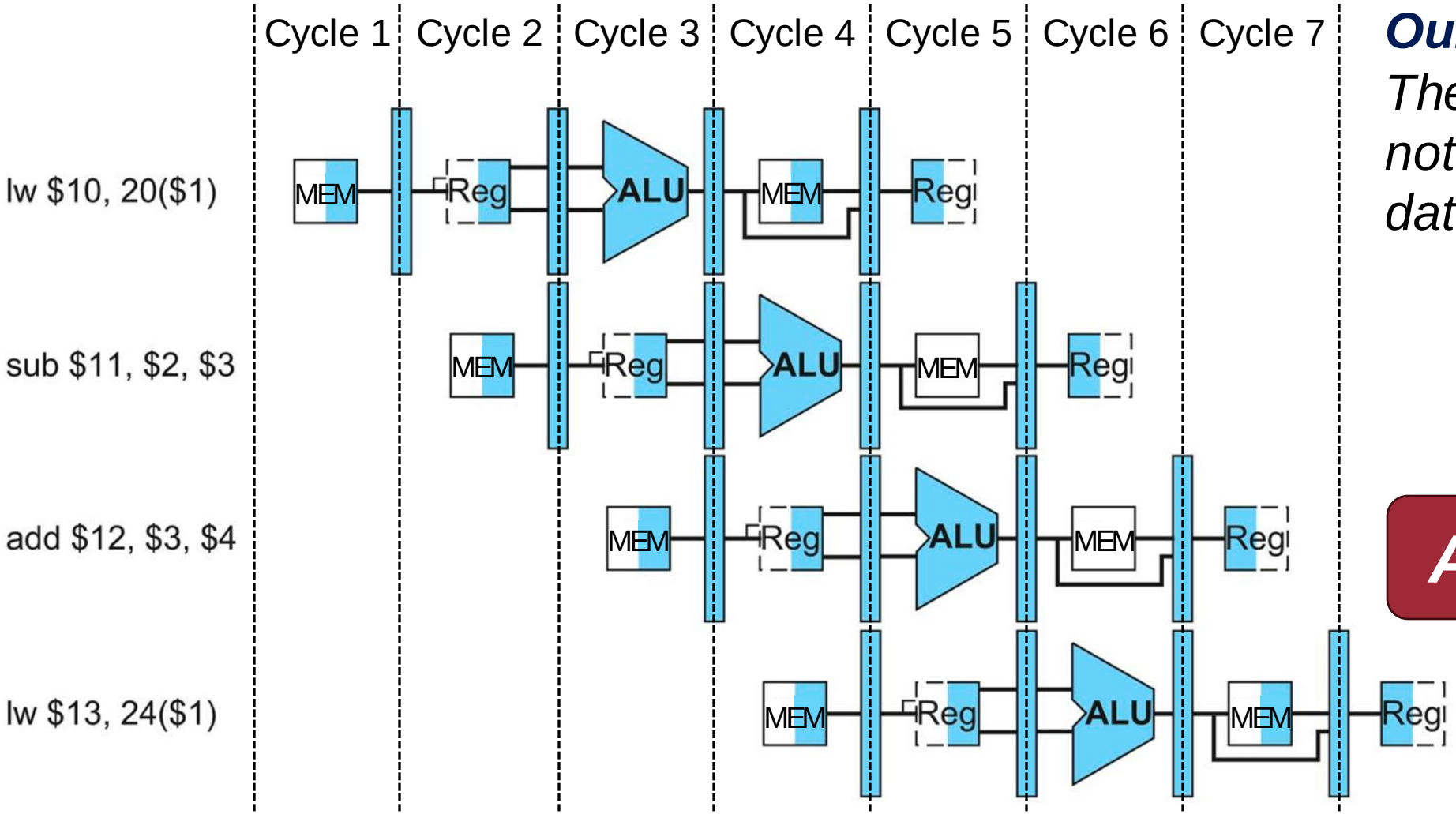
Structural Hazard

Structural Hazard

The **hardware** cannot support *the combination of instructions*

Structural Hazard

The **hardware** cannot support *the combination of instructions*

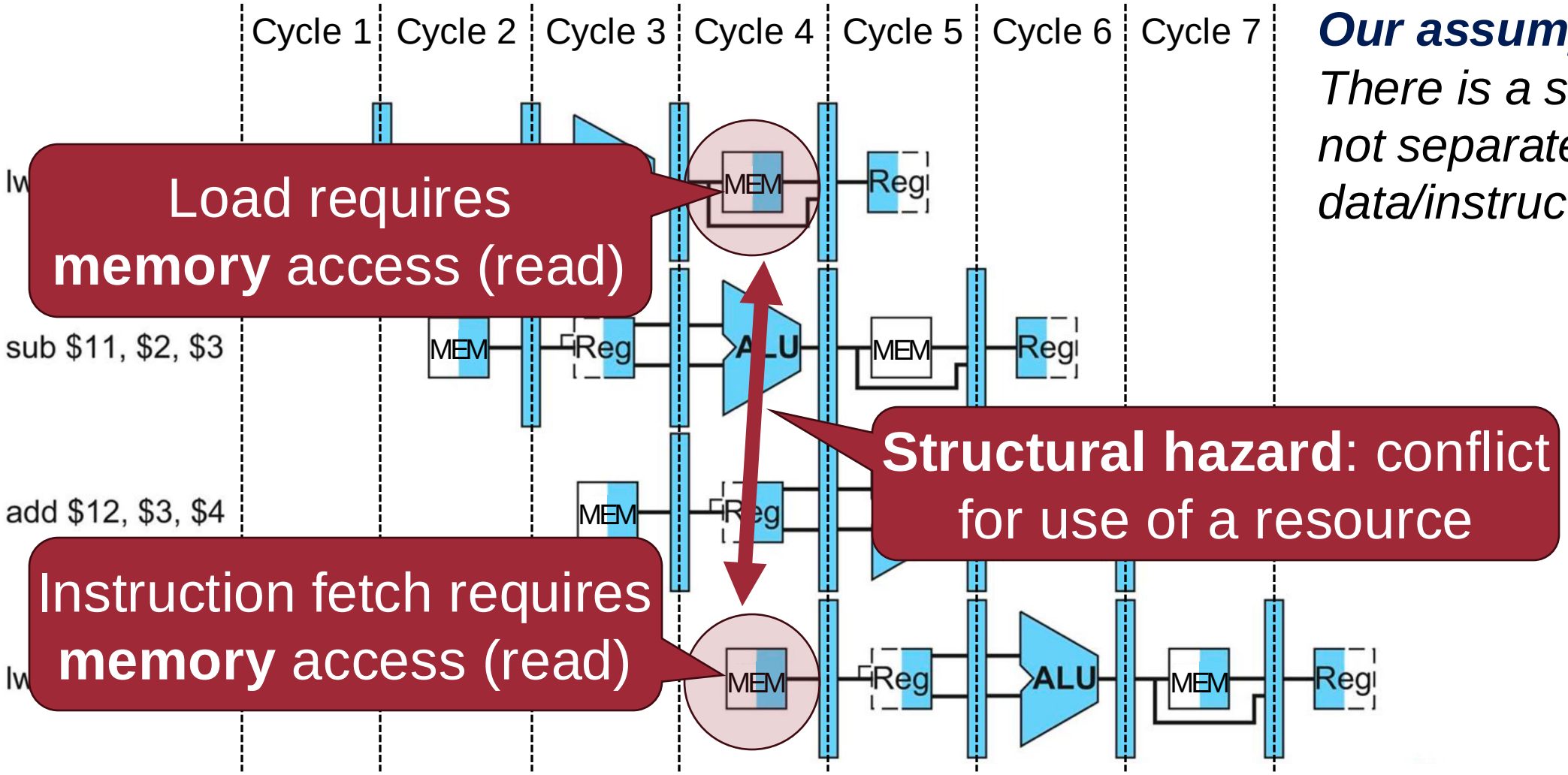


Our assumption:
There is a single memory, not separate data/instruction memories

Any problems?

Structural Hazard

The **hardware** cannot support *the combination of instructions*



Our assumption:
There is a single memory, not separate data/instruction memories

Pipelining Hazards Summary

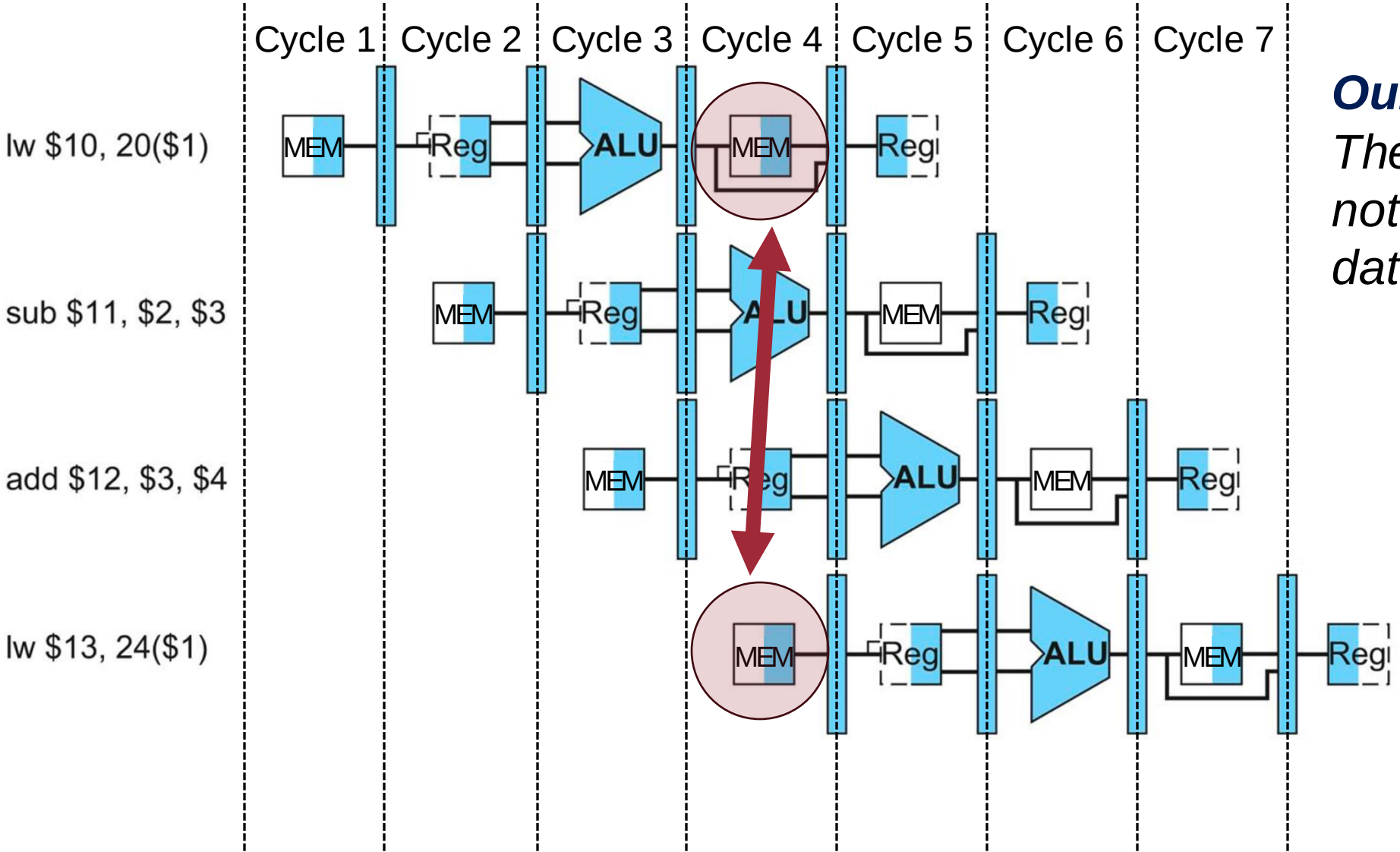
Situations that prevent starting the next instruction in the next cycle

Hazard #1: Structural hazard	Hazard #2: Data hazard	Hazard #3: Control hazard
Conflict for use of a hardware resource		

Solution:

- Stall
- Resource duplication

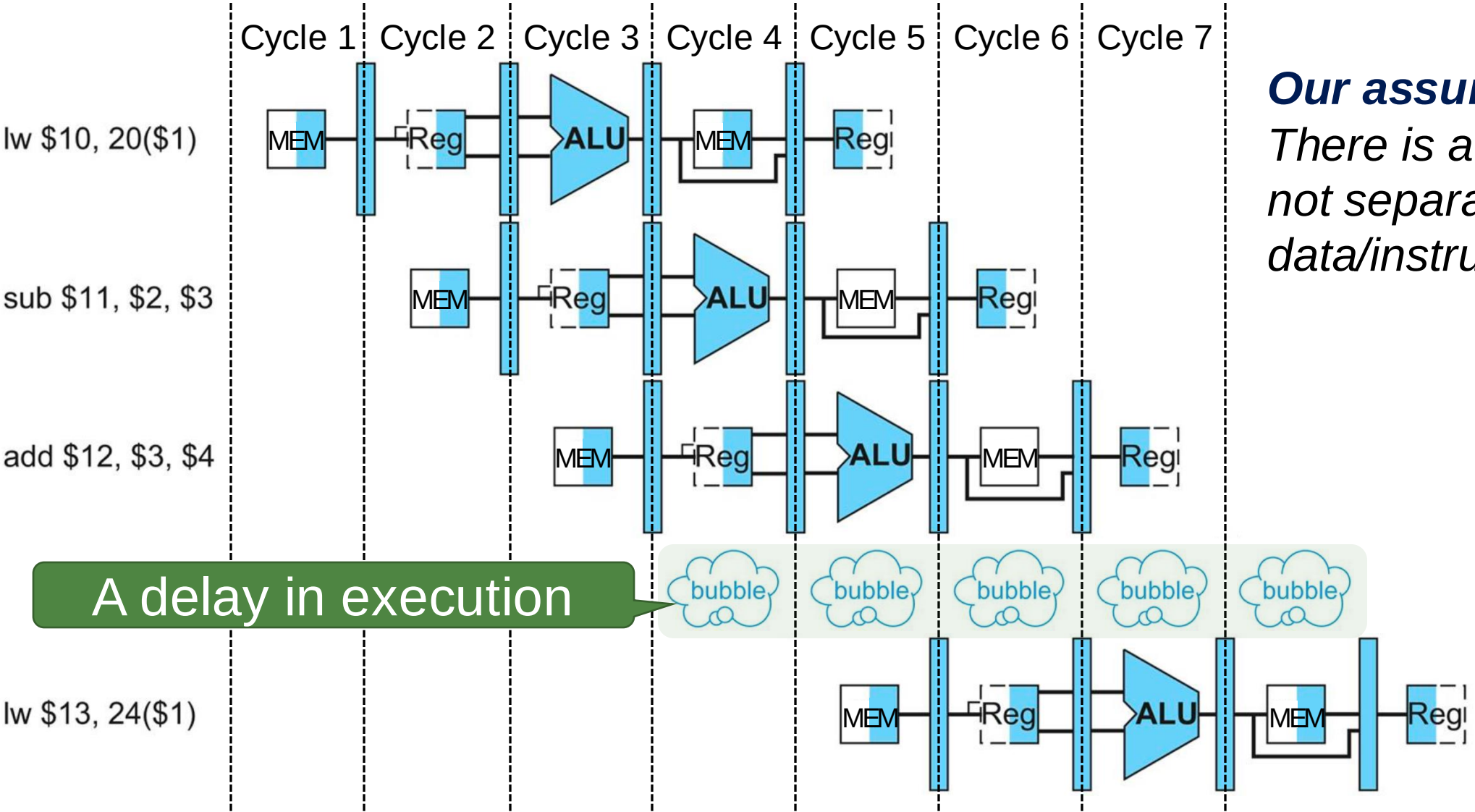
Solution for Structural Hazard: Stall



Our assumption:

There is a single memory, not separate data/instruction memories

Solution for Structural Hazard: Pipeline Stall

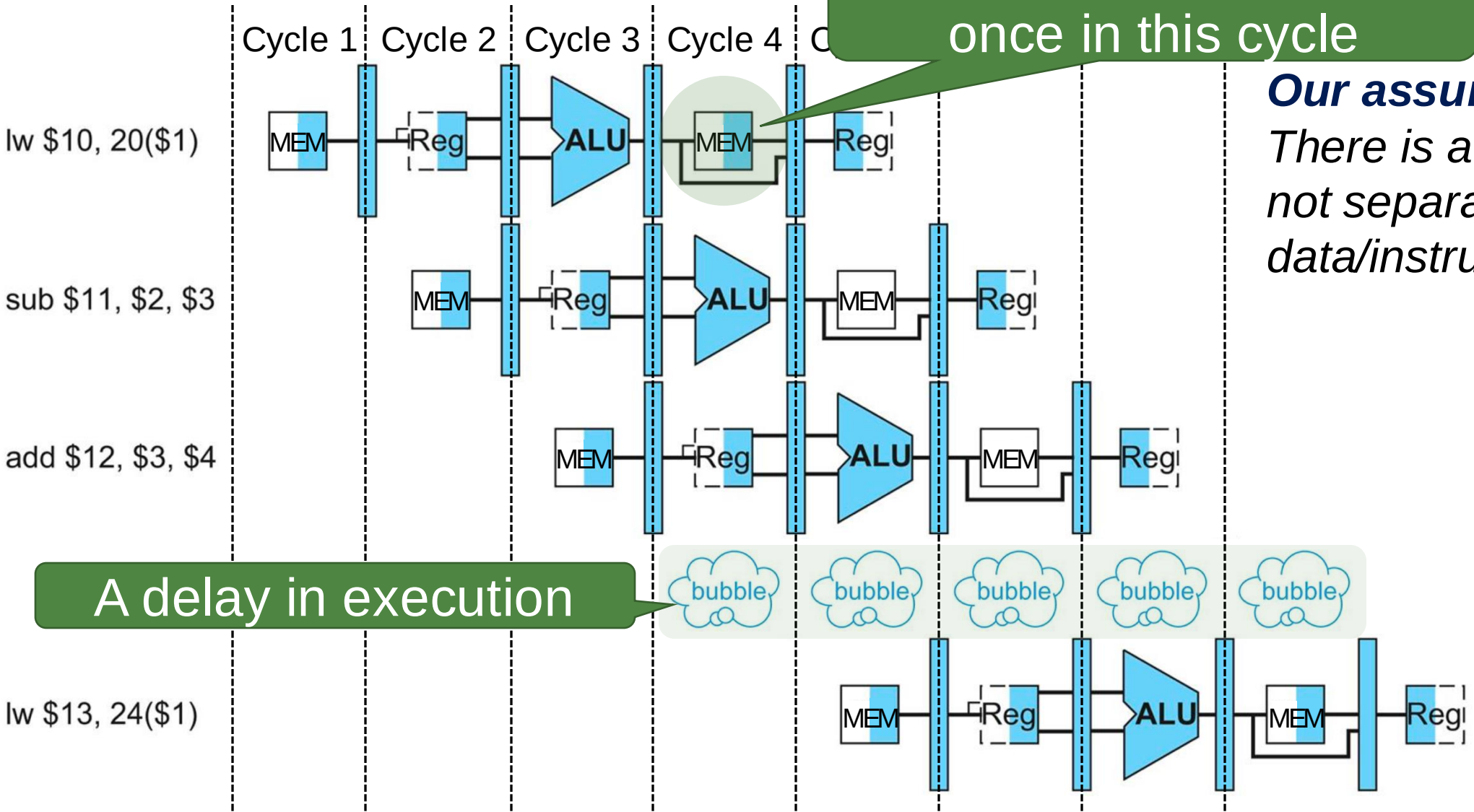


Our assumption:

There is a single memory, not separate data/instruction memories

Solution for Structural Hazard: Pipeline Stall

The memory is accessed once in this cycle

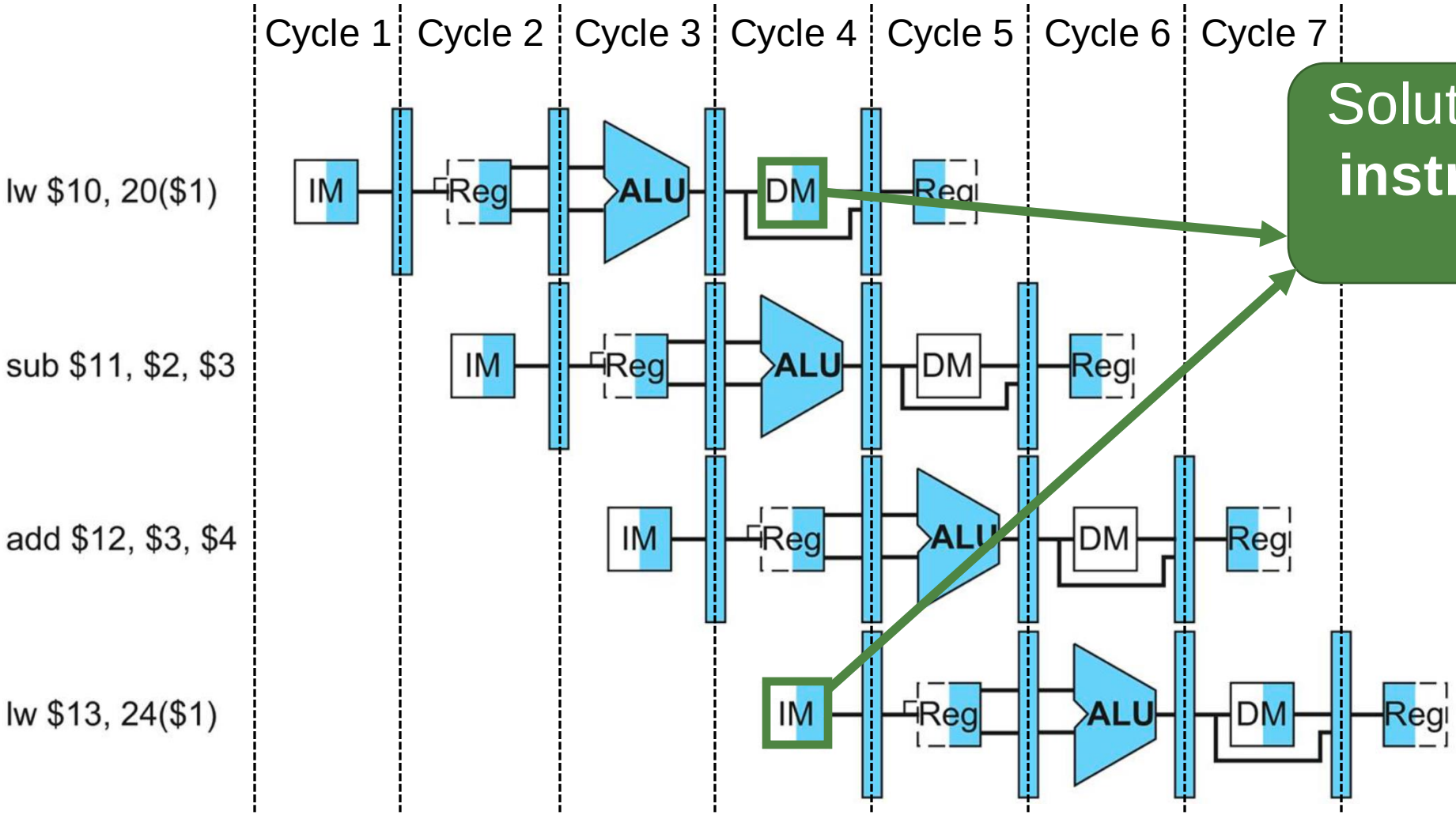


A delay in execution

Our assumption:

There is a single memory, not separate data/instruction memories

Solution for Structural Hazard: Resource Duplication



Solution: use separate **instruction and data memory**

Pipelining Hazards Summary

Situations that prevent starting the next instruction in the next cycle

Hazard #1: Structural hazard	Hazard #2: Data hazard	Hazard #3: Control hazard
Conflict for use of a hardware resource		

Solution:

- Stall
- Resource duplication

Data Hazard

Data Hazard

A planned instruction **cannot execute** because <u>data is not yet available</u>

Data Hazard

A planned instruction **cannot execute** because <u>data is not yet available</u>

```
add $s0, $t0, $t1
sub $t2, $s0, $t3
```

Data Hazard

A planned instruction **cannot execute** because <u>data is not yet available</u>

Read after Write (RAW) dependency

```
add $s0, $t0, $t1
sub $t2, $s0, $t3
```

Data Hazard

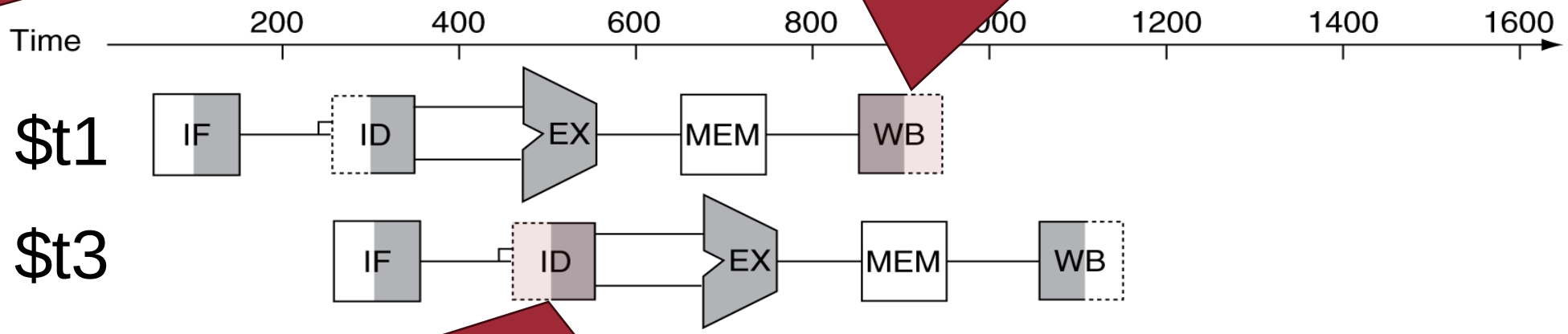
A planned instruction **cannot execute** because data is not yet available

Read after Write (RAW) dependency

The calculated data of $s0 is available in the 5th clock cycle

add $s0, $t0, $t1

sub $t2, $s0, $t3



The data of $s0 calculated in the previous instruction is needed in the 3rd clock cycle

Data Hazard

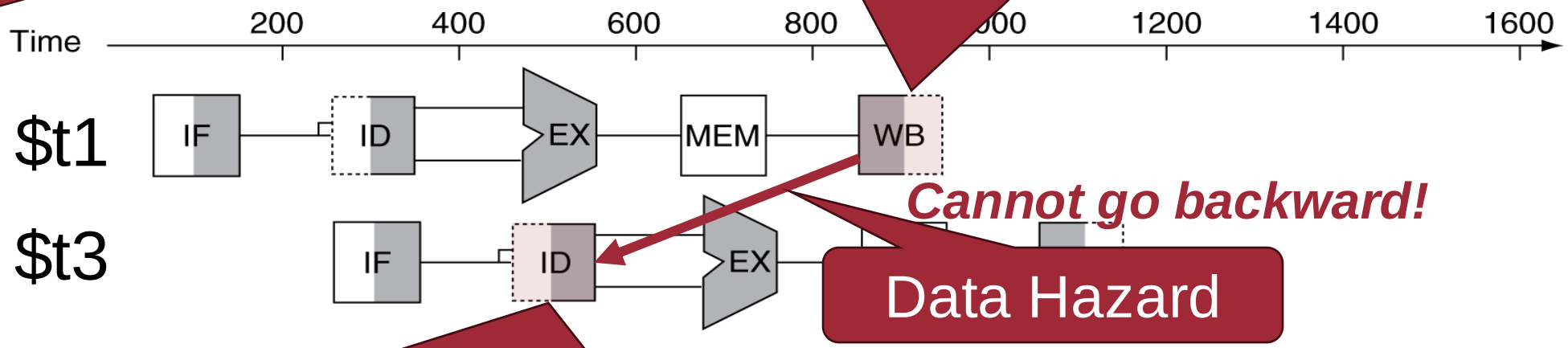
A planned instruction **cannot execute** because data is not yet available

Read after Write (RAW) dependency

The calculated data of $s0 is available in the 5th clock cycle

add $s0, $t0, $t1

sub $t2, $s0, $t3



Cannot go backward!

Data Hazard

The data of $s0 calculated in the previous instruction is needed in the 3rd clock cycle

Pipelining Hazards Summary

Situations that prevent starting the next instruction in the next cycle

Hazard #1: Structural hazard	Hazard #2: Data hazard	Hazard #3: Control hazard
Conflict for use of a hardware resource	An instruction cannot execute because data is not yet available	
Solution: • Stall • Resource duplication	**Solution:** • Stall • Forwarding • Compiler optimization	

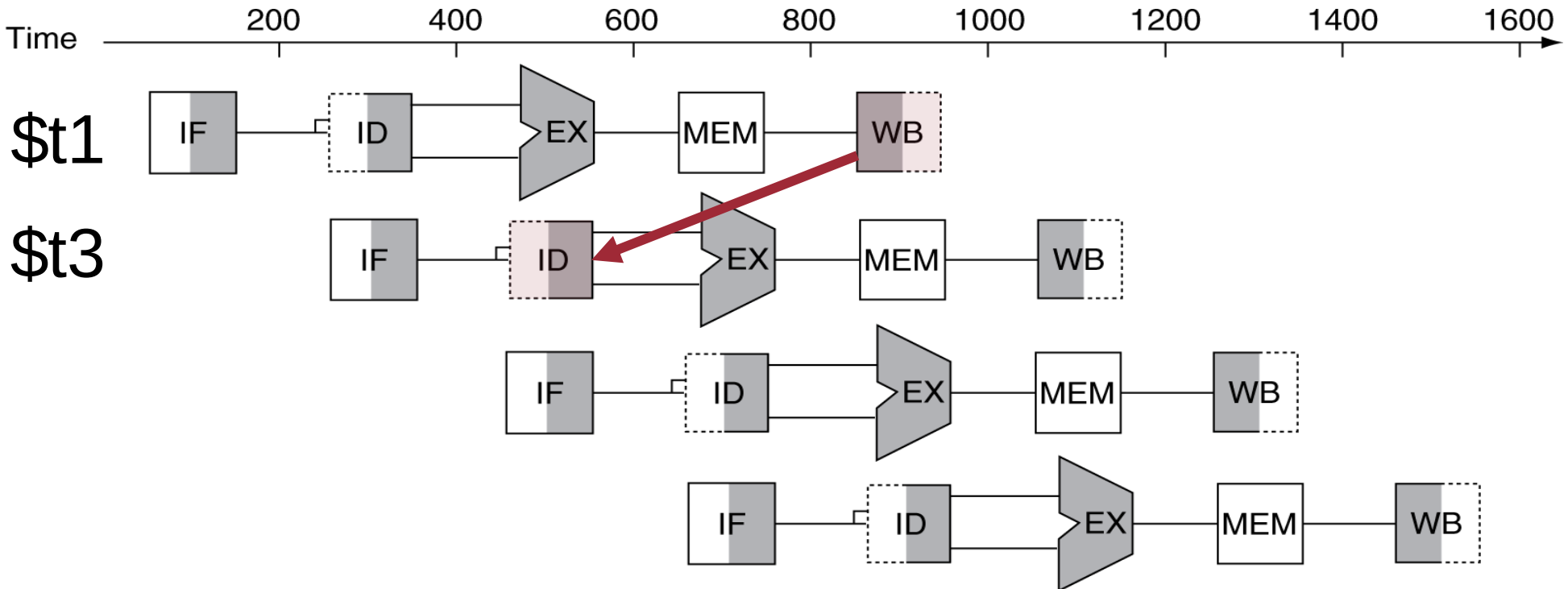
Solution for Data Hazard: Pipeline Stall

add $s0, $t0, $t1

sub $t2, $s0, $t3

…

…



Solution for Data Hazard: Pipeline Stall

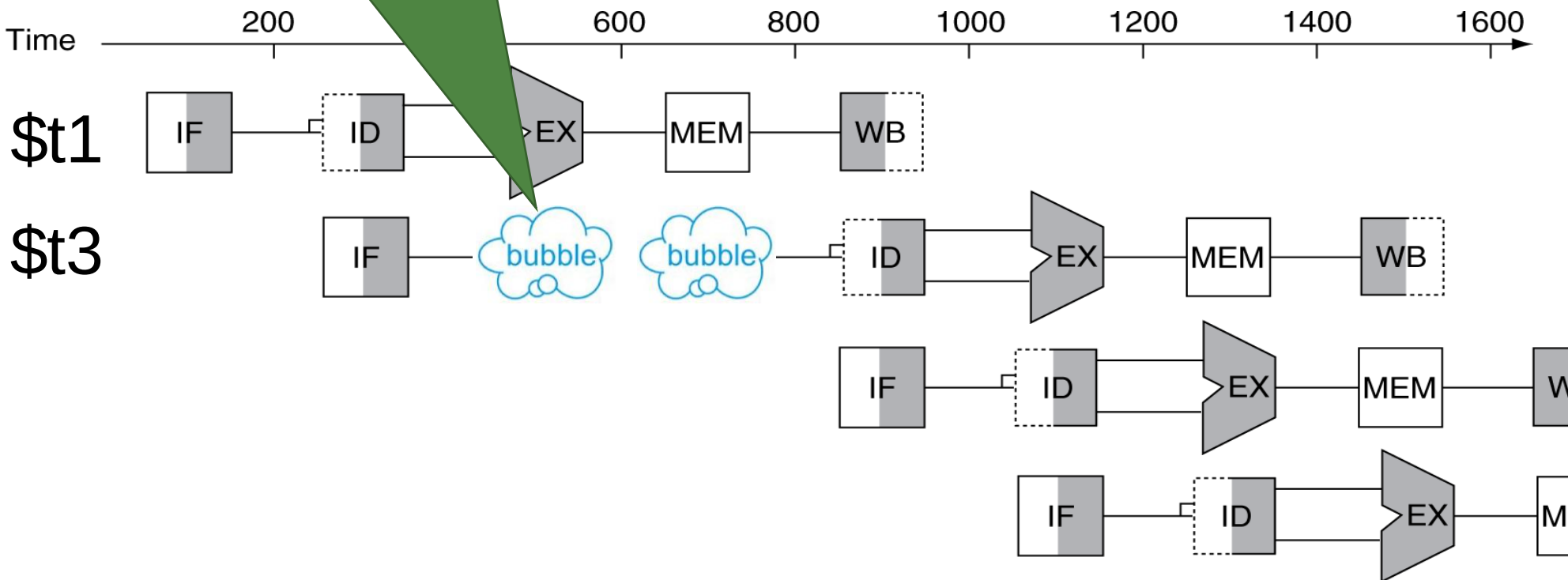
A delay in execution
(2 clock cycle)

add $s0, $t0, $t1

sub $t2, $s0, $t3

…

…



Solution for Data Hazard: Pipeline Stall

Performance may degrade.
Is there a better way?
Forwarding!

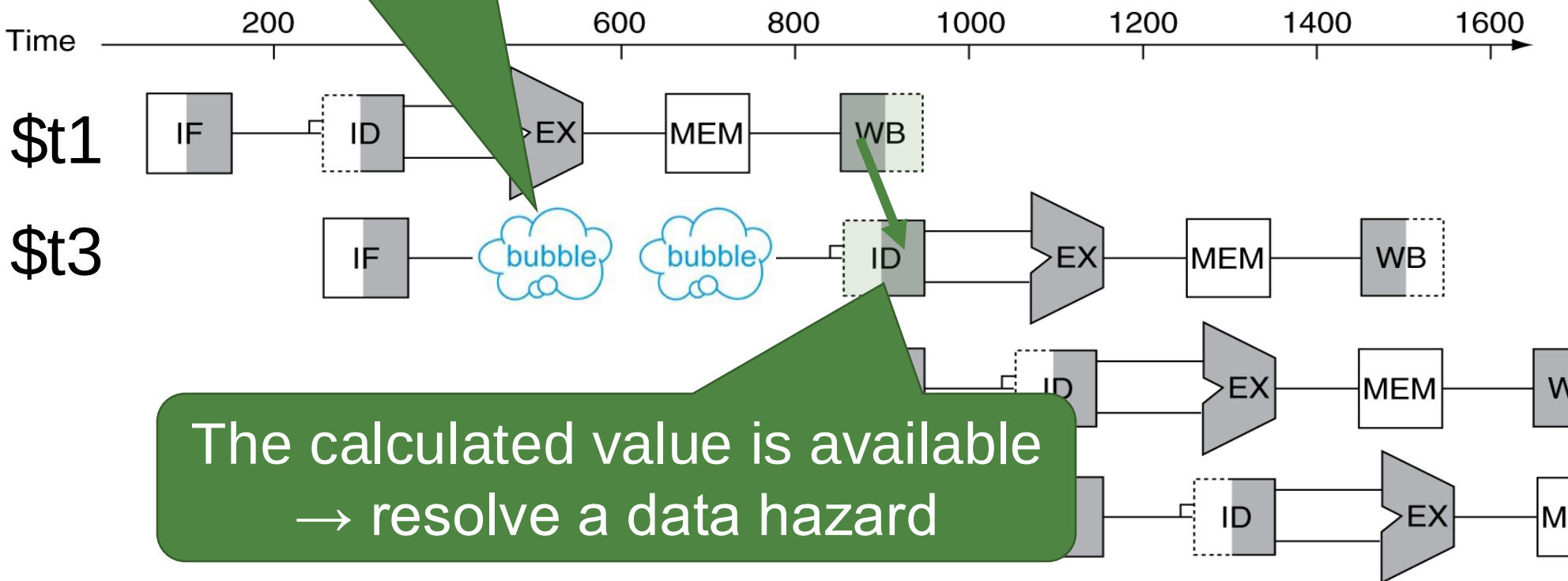
A delay in execution
(2 clock cycle)

add $s0, $t0, $t1

sub $t2, $s0, $t3

…

…



The calculated value is available
→ resolve a data hazard

Solution for Data Hazard: Forwarding (bypassing)

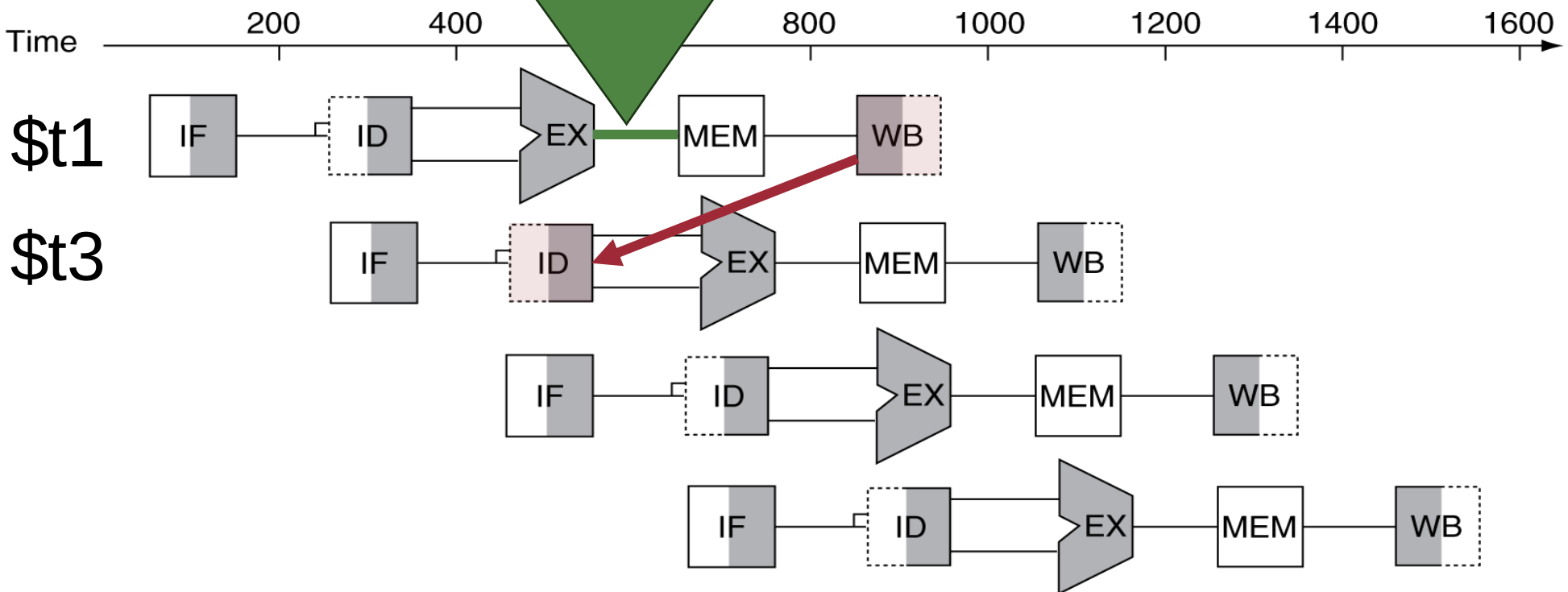
Observation: the result data is available in the 3rd cycle

add $s0, $t0, $t1

sub $t2, $s0, $t3

…

…



Solution for Data Hazard: Forwarding (bypassing)

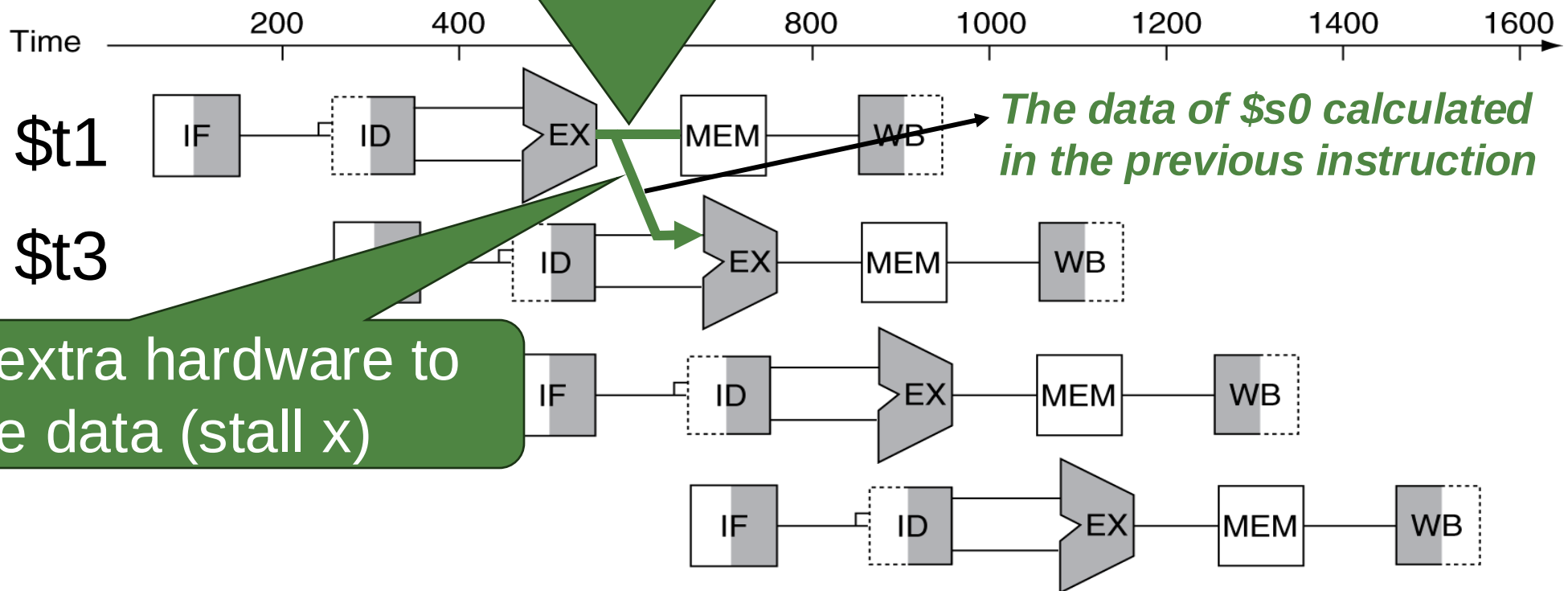
- Use result when it is computed!

Observation: the result data is available in the 3rd cycle

add $s0, $t0, $t1

sub $t2, $s0, $t3

...



The data of $s0 calculated in the previous instruction

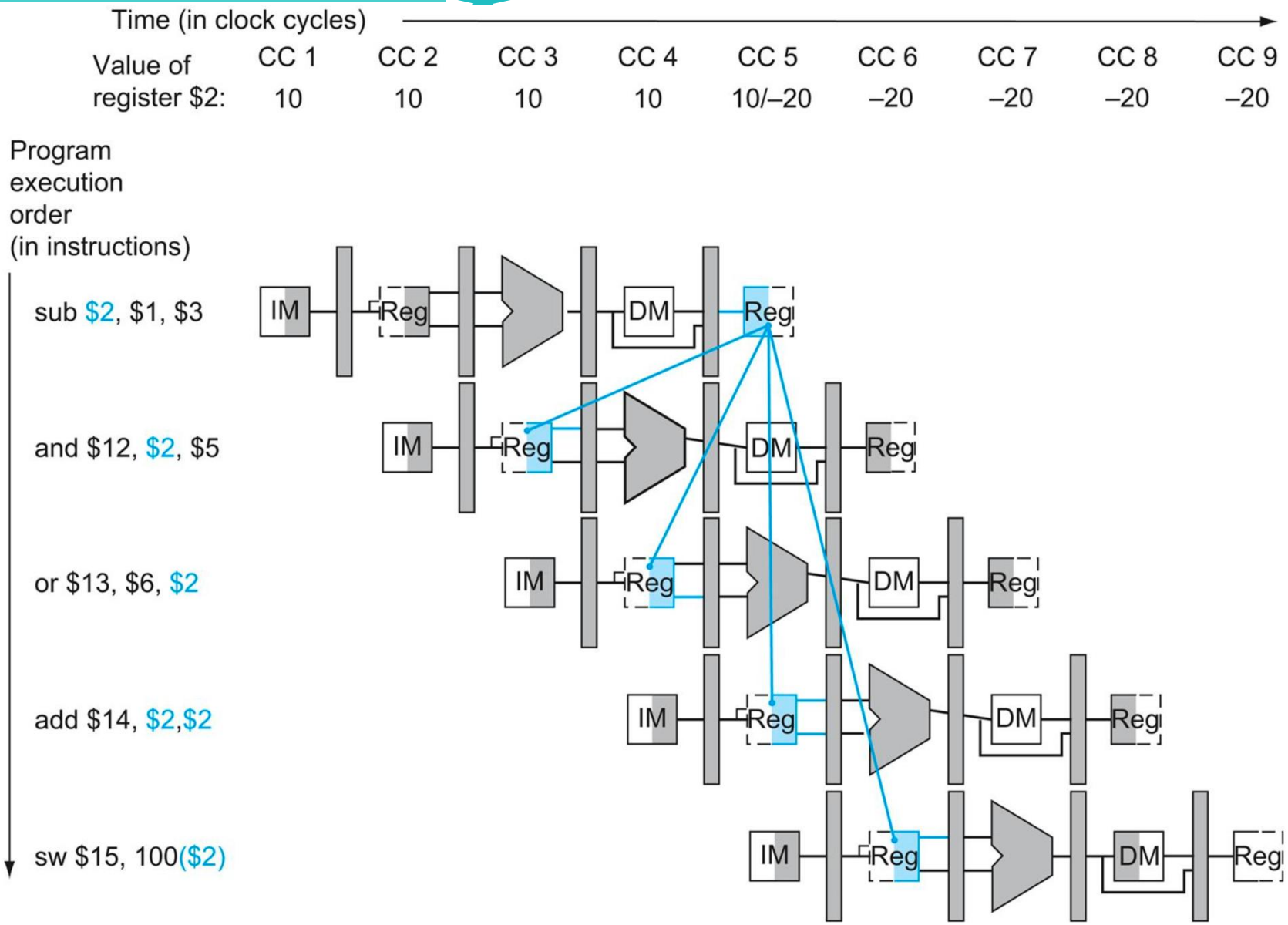
Idea: adding extra hardware to forward the data (stall x)

Data Hazard Example

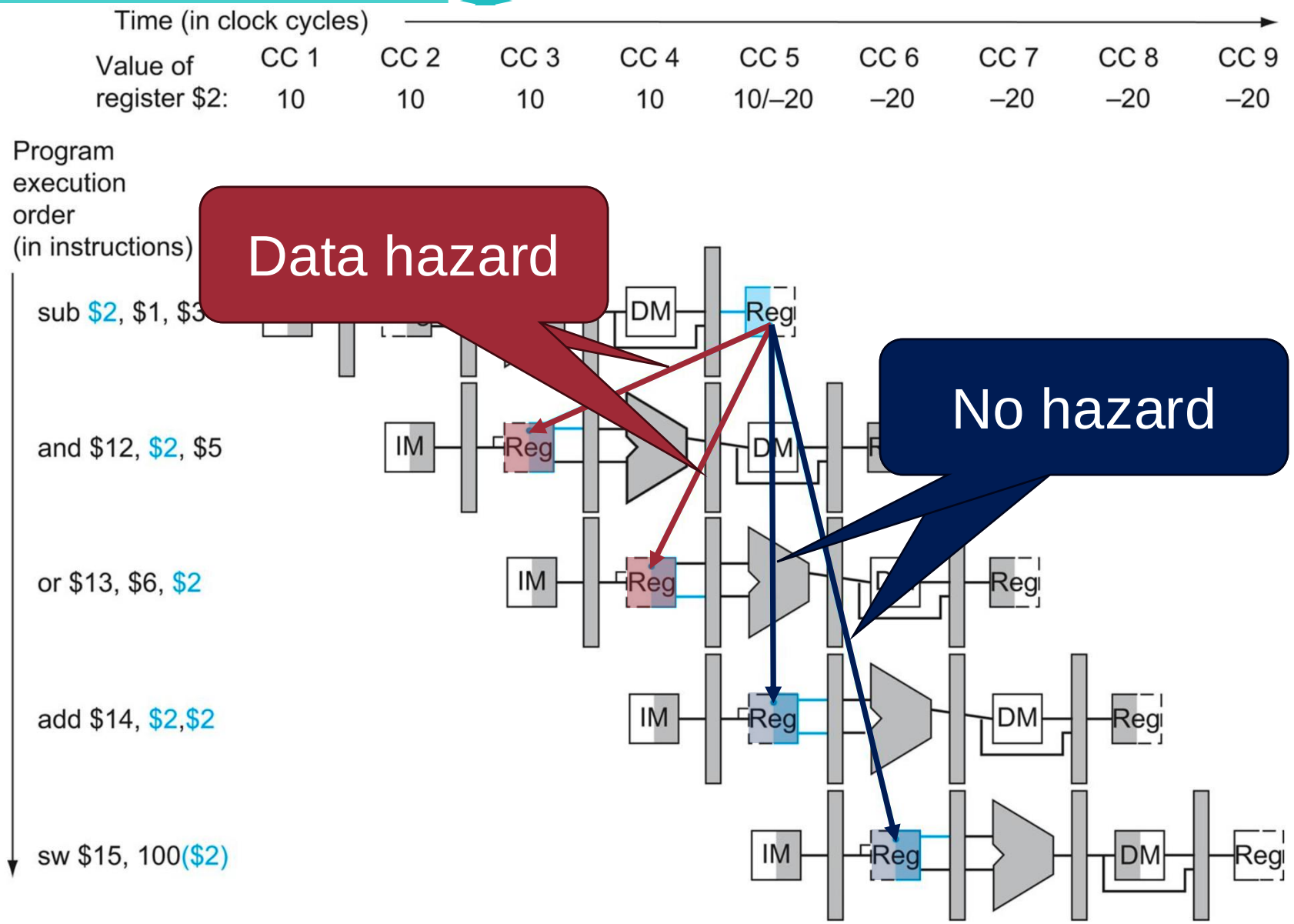
10 → -20 after sub

```
sub    $2, $1,$3        # Register $2 written by sub
and    $12,$2,$5        # 1st operand($2) depends on sub
or     $13,$6,$2        # 2nd operand($2) depends on sub
add    $14,$2,$2        # 1st($2) & 2nd($2) depend on sub
sw     $15,100($2)      # Base ($2) depends on sub
```

Data Hazard Example



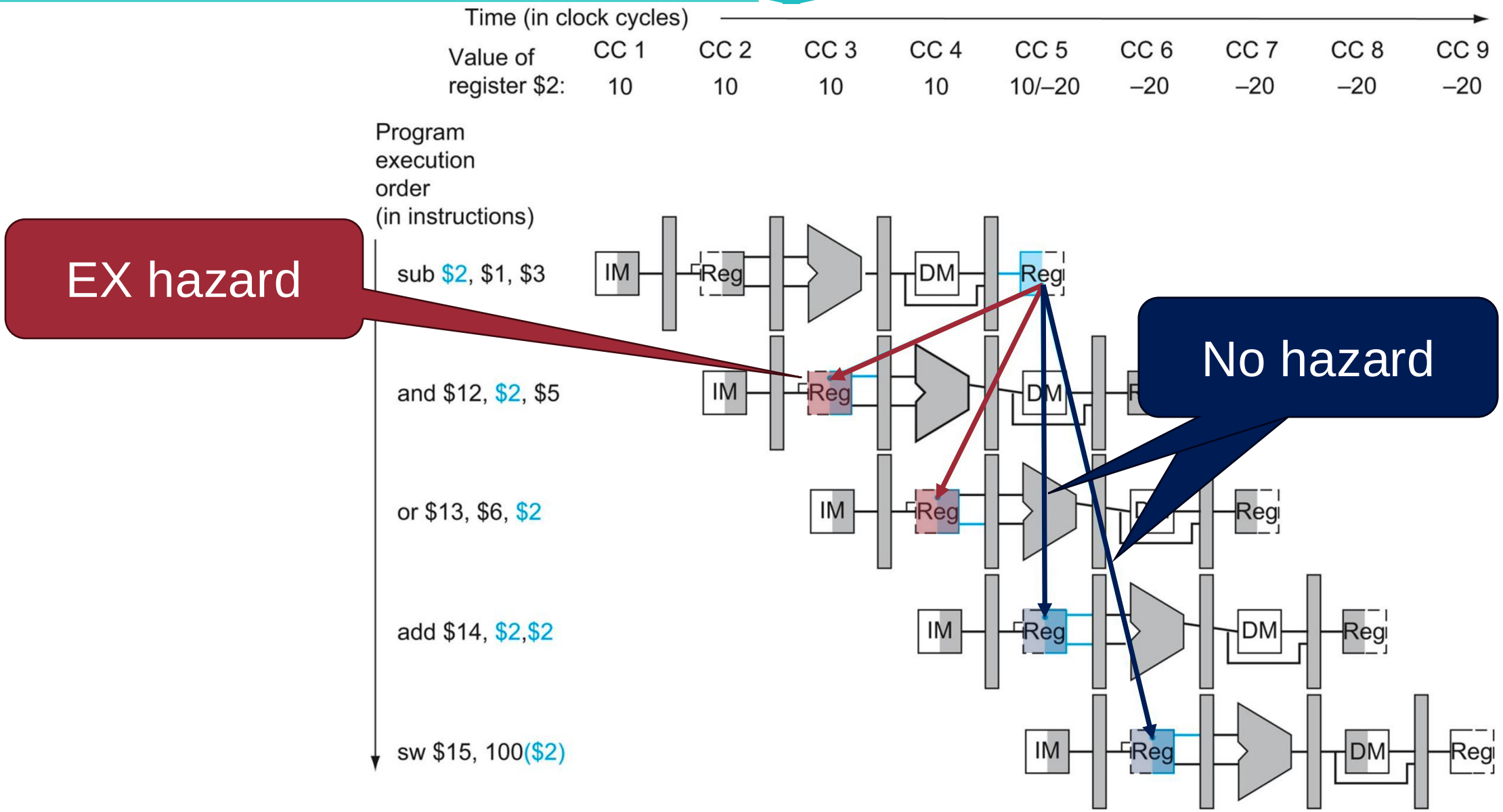
Data Hazard Example: without Forwarding



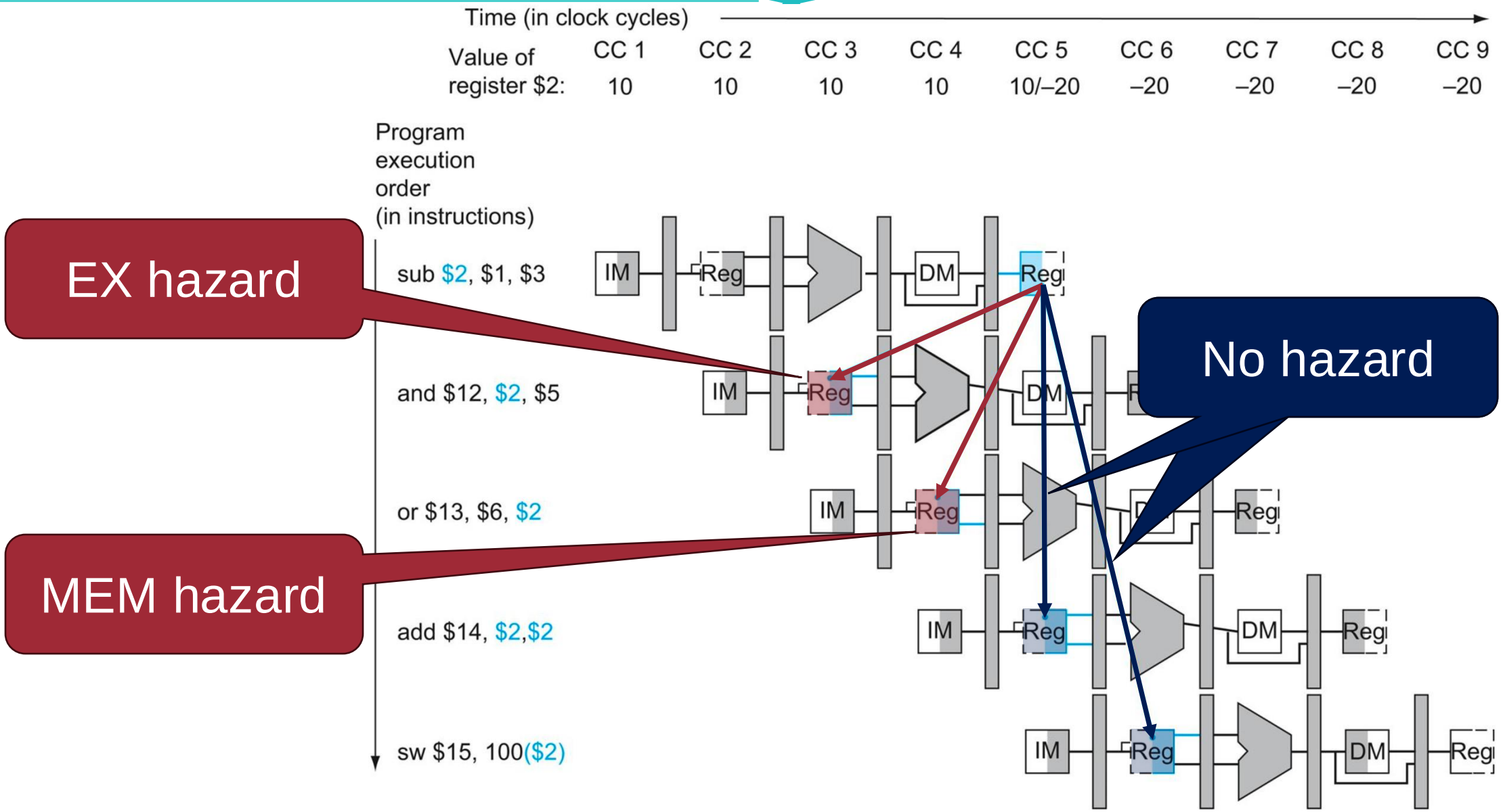
FYI: Three Types of Data Hazard

1. EX Hazard
2. MEM Hazard
3. Load-Use Hazard

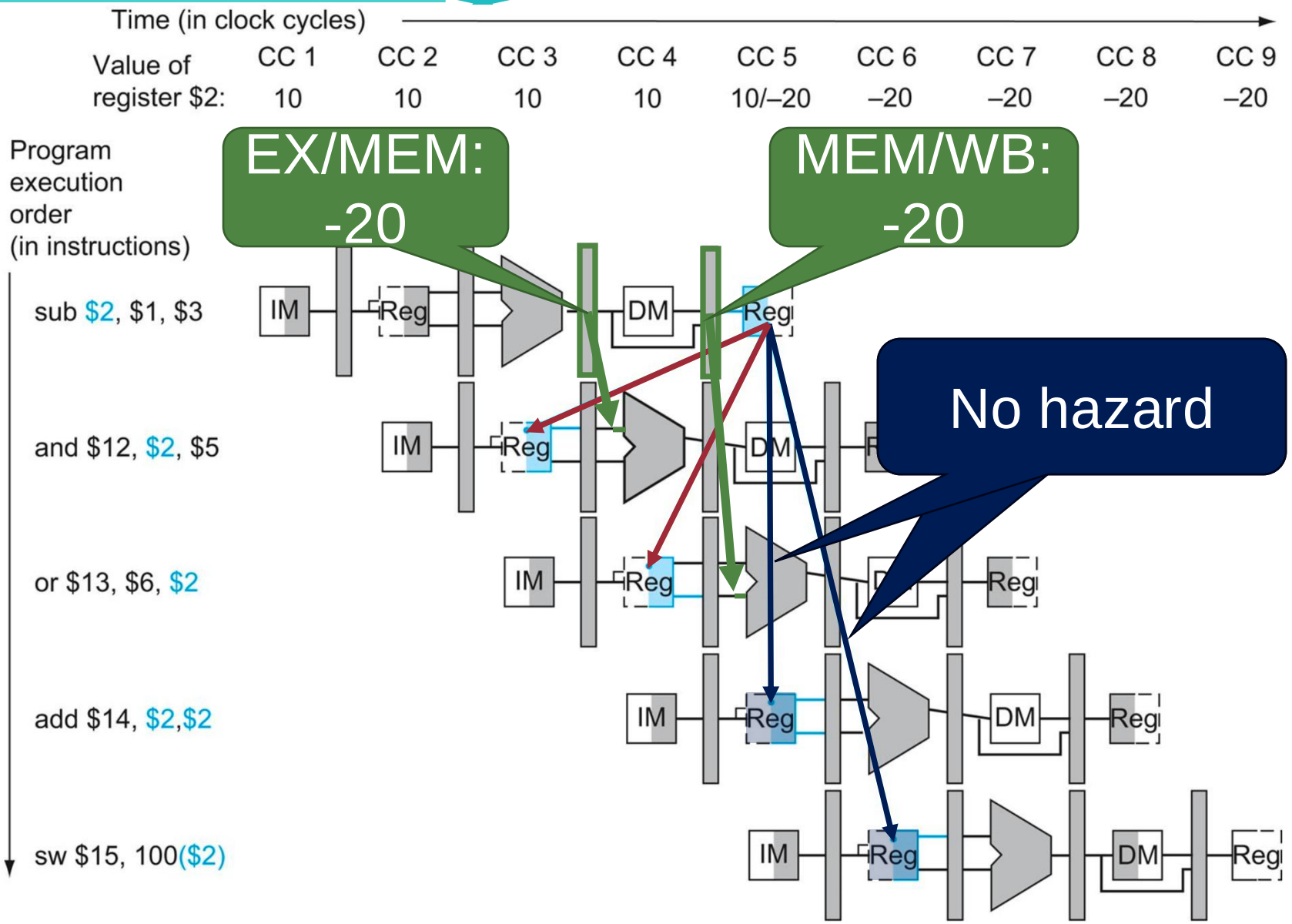
Data Hazard #1: EX Hazard



Data Hazard #2: MEM Hazard



Data Hazard Example: with Forwarding



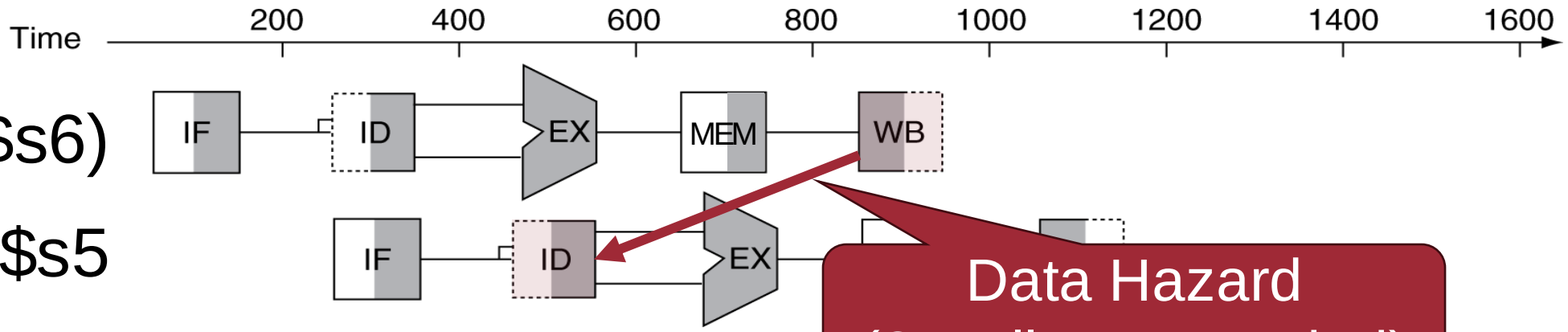
Data Hazard #3: Load-Use Data Hazard

Load

lw $s1, 32($s6)

sub $s4, $s1, $s5

Use



Data Hazard
(2 stalls are needed)

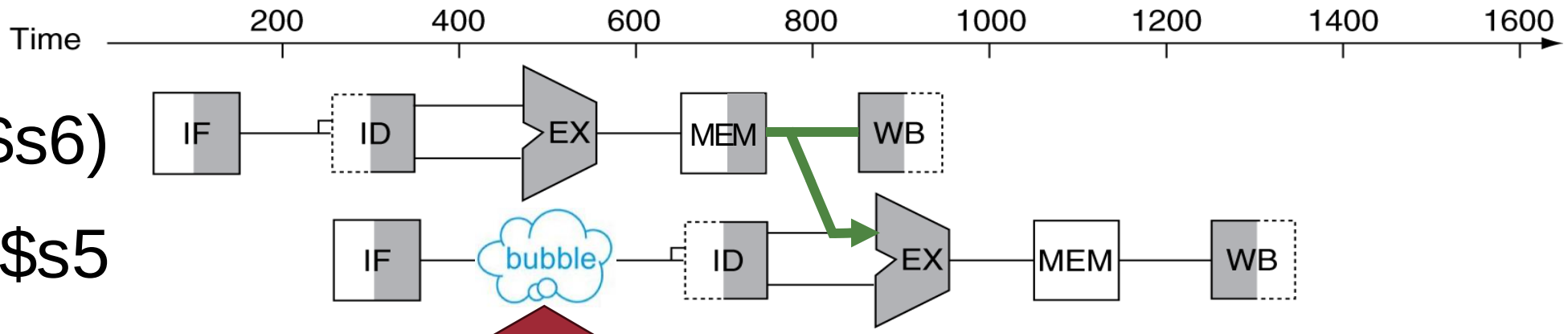
Load-Use Data Hazard with Forwarding

Load

lw $s1, 32($s6)

sub $s4, $s1, $s5

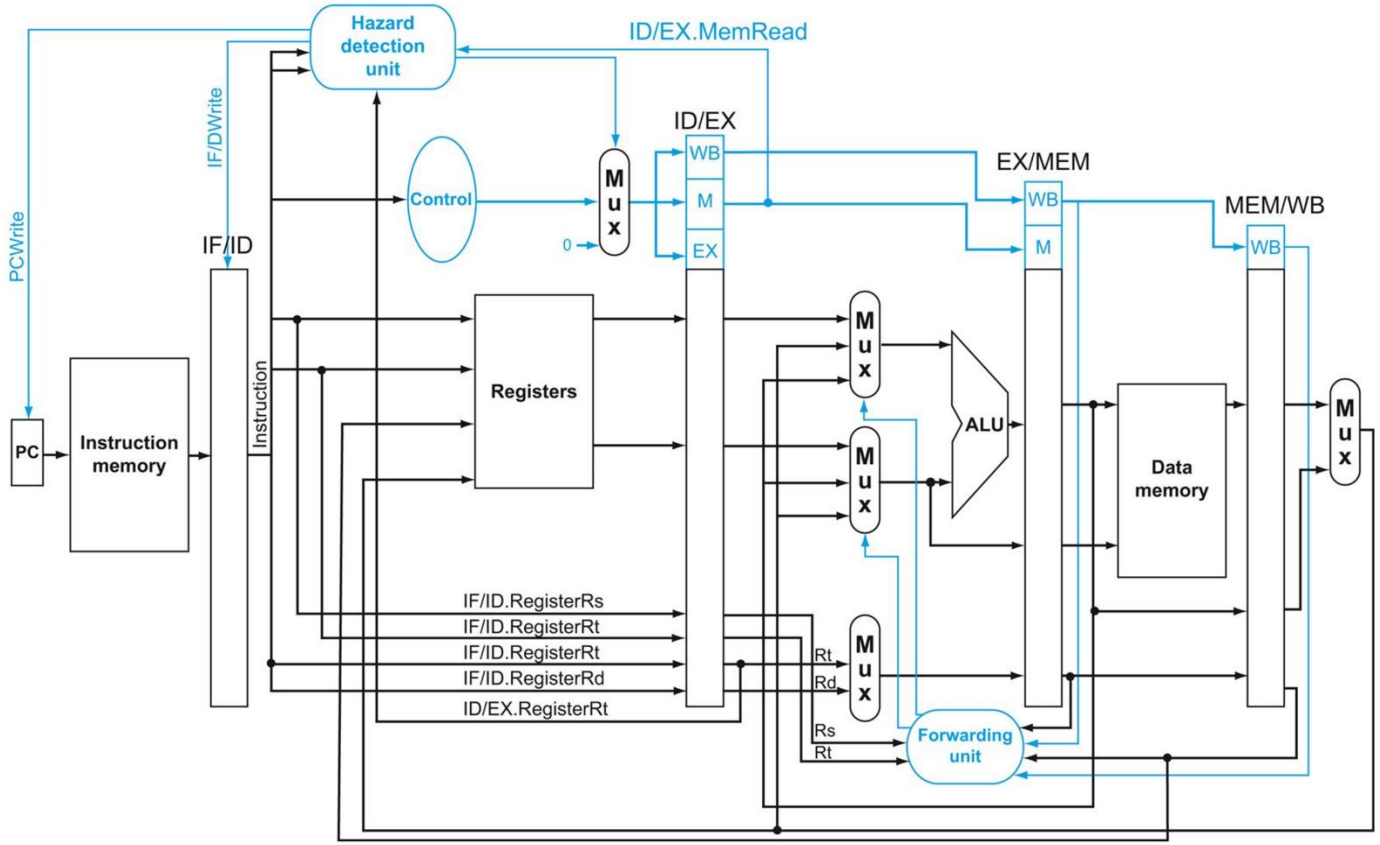
Use



`lw` still cause a hazard: we need a stall even with forwarding when <u>a load</u> tries to use the data

Question

- How should the **hardware** be modified to detect data hazards?
- How should the **hardware** be modified to support forwarding?
- How should the **hardware** be modified to support stalls?



Next lecture!
(in details)

Solution for Data Hazard: Forwarding

Forwarding is effective, but we don't have a hardware expert. In this situation, is there any way to resolve the data hazard in software manner?

Compiler Optimization!

Solution for Data Hazard: Compiler Optimization

- Reorder code to avoid use of load result in the next instruction
- C code for `v[3] = v[0] + v[1]; v[4] = v[0] + v[2];`

```
      lw  $t1, 0($t0)
      lw  $t2, 4($t0)
1 stall
      add $t3, $t1, $t2
      sw  $t3, 12($t0)
      lw  $t4, 8($t0)
1 stall
      add $t5, $t1, $t4
      sw  $t5, 16($t0)
```

13 cycles

Solution for Data Hazard: Compiler Optimization

- Reorder code to avoid use of load result in the next instruction
- C code for `v[3] = v[0] + v[1]; v[4] = v[0] + v[2];`

```
lw   $t1, 0($t0)
lw   $t2, 4($t0)
add  $t3, $t1, $t2
sw   $t3, 12($t0)
lw   $t4, 8($t0)
add  $t5, $t1, $t4
sw   $t5, 16($t0)
```

1 stall

1 stall

Idea: code reordering to avoid stalls (by compiler)

Compiler requires knowledge of the pipeline structure!

13 cycles

Solution for Data Hazard: Compiler Optimization

- Reorder code to avoid use of load result in the next instruction
- C code for `v[3] = v[0] + v[1]; v[4] = v[0] + v[2];`

```
lw  $t1, 0($t0)
lw  $t2, 4($t0)
add $t3, $t1, $t2
sw  $t3, 12($t0)
lw  $t4, 8($t0)
add $t5, $t1, $t4
sw  $t5, 16($t0)
```

1 stall

1 stall

```
lw  $t1, 0($t0)
lw  $t2, 4($t0)
lw  $t4, 8($t0)
add $t3, $t1, $t2
sw  $t3, 12($t0)
add $t5, $t1, $t4
sw  $t5, 16($t0)
```

13 cycles

Solution for Data Hazard: Compiler Optimization

- Reorder code to avoid use of load result in the next instruction
- C code for `v[3] = v[0] + v[1]; v[4] = v[0] + v[2];`

```
lw  $t1, 0($t0)
lw  $t2, 4($t0)
add $t3, $t1, $t2
sw  $t3, 12($t0)
lw  $t4, 8($t0)
add $t5, $t1, $t4
sw  $t5, 16($t0)
```

1 stall

1 stall

13 cycles

```
lw  $t1, 0($t0)
lw  $t2, 4($t0)
lw  $t4, 8($t0)
add $t3, $t1, $t2
sw  $t3, 12($t0)
add $t5, $t1, $t4
sw  $t5, 16($t0)
```

No stall

No stall

11 cycles

Pipelining Hazards Summary

Situations that prevent starting the next instruction in the next cycle

Hazard #1: Structural hazard

Conflict for use of a hardware resource

Next lecture! (in details)

Solution:

- Stall
- Resource duplication

Hazard #2: Data hazard

An instruction cannot execute because data is not yet available

Solution:

- Stall
- Forwarding
- Compiler optimization

Hazard #3: Control hazard

Next lecture!

Question?