Microprocessor & Computer Architecture (µpCA)

UE19CS252

Dr. D. C. Kiran

Department of
Computer Science and Engineering

PES UNIVERSITY ONLINE

Microprocessor & Computer Architecture (μpCA)

Pipeline Processor: Data Hazard 2

Dr. D. C. Kiran

Department of Computer Science and Engineering

Microprocessor & Computer Architecture (µpCA)

Syllabus

~~Unit 1: Basic Processor Architecture and Design~~

Unit 2: Pipelined Processor and Design

- ~~3-Stage ARM Processor~~
- ~~5-Stage Pipeline Processor~~
- ~~Introduction to Pipeline Processor~~
- ~~What May Go Wrong?~~
- ~~Introduction to Hazards, Stalls,~~
- ~~Structural Hazards~~
- ~~Data Hazard~~
  - ~~RAW, WAR, WAW Hazards~~
- Attacking Data Hazard
  - Software Approach
  - Hardware Approach

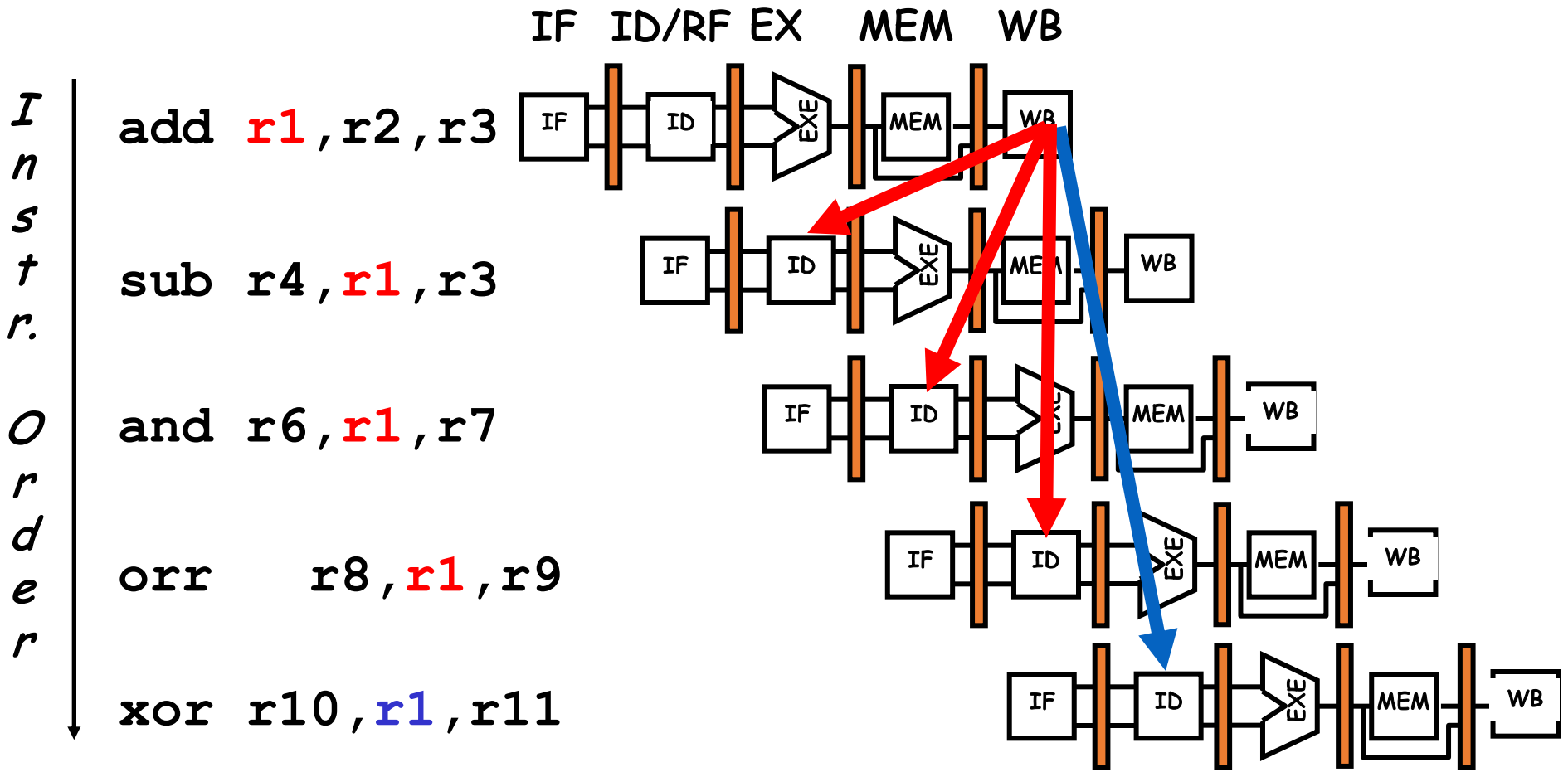
Microprocessor & Computer Architecture (µpCA)

Text 1: "Computer Organization and Design", Patterson, Hennessey, 5th Edition, Morgan Kaufmann, 2014.

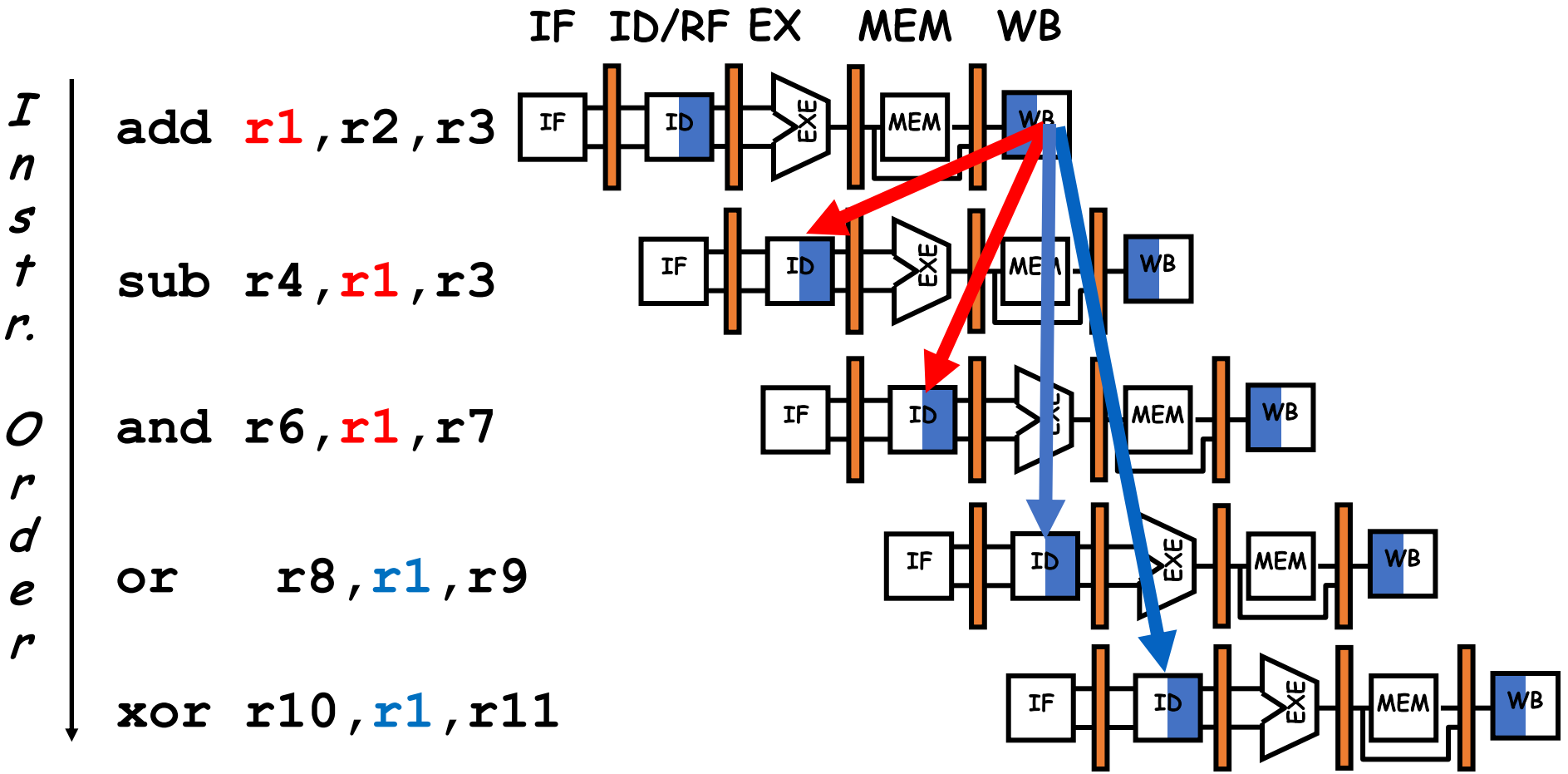
Reference 1:"Computer Architecture: A Quantitative Approach", Hennessey, Patterson, 5th Edition, Morgan Kaufmann, 2011.

Appendix C **Pipelining: Basic and Intermediate Concepts**

How to Attack RAW Dependency?



How to Attack RAW Dependency?



Attacking Data Hazard

- In Software
  - **Solution 1: Re-order instructions**
  - Solution 2: insert independent instructions (or no-ops) Ex: MOV R0, R0
- In Hardware
  - Solution 1: Insert bubbles (i.e. stall the pipeline)
  - Solution 2: Data Forwarding

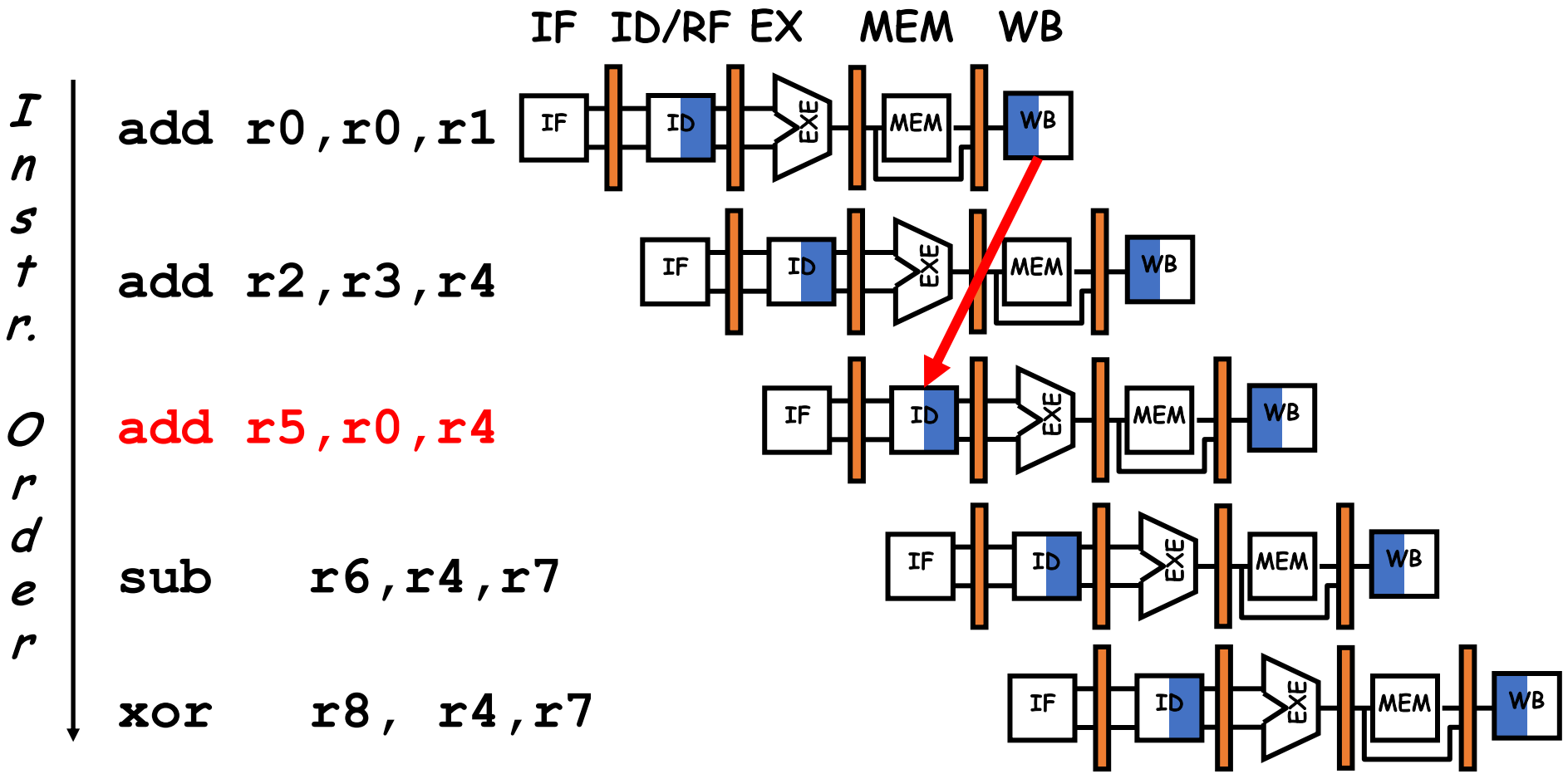
Software Solution 1→ By Compiler


```
ADD R0,R0,R1
ADD R2,R3,R4
ADD R5,R0,R4
SUB R6,R4,R7
XOR R8,R4,R7
```

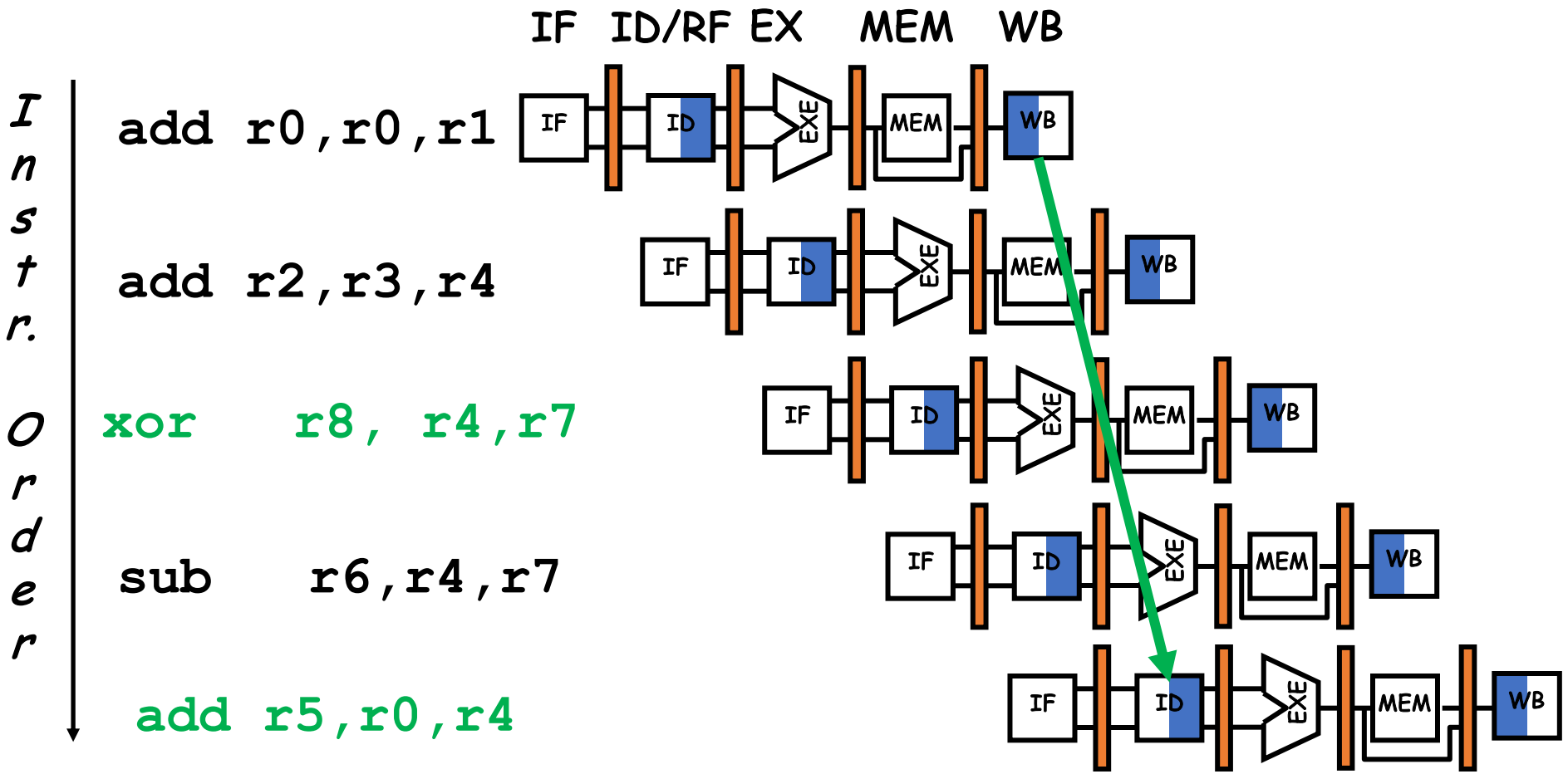
```
ADD R0,R0,R1
ADD R2,R3,R4
SUB R6,R4,R7
XOR R8,R4,R7
ADD R5,R0,R4
```

Out of Order Execution by Compiler

Software Solution 1→ By Compiler



Software Solution 1→ By Compiler



Attacking Data Hazard

- In Software
  - Solution 1: Re-order instructions
  - **Solution 2: insert independent instructions (or no-ops) Ex: MOV R0, R0**
- In Hardware
  - Solution 1: Insert bubbles (i.e. stall the pipeline)
  - Solution 2: Data Forwarding

Software Solution 2→ By Compiler

Insert NOP or MOV R0, R0

- The compiler can guarantee that no data hazards exist adding NOP or MOV instructions where needed. [Instruction Scheduling]

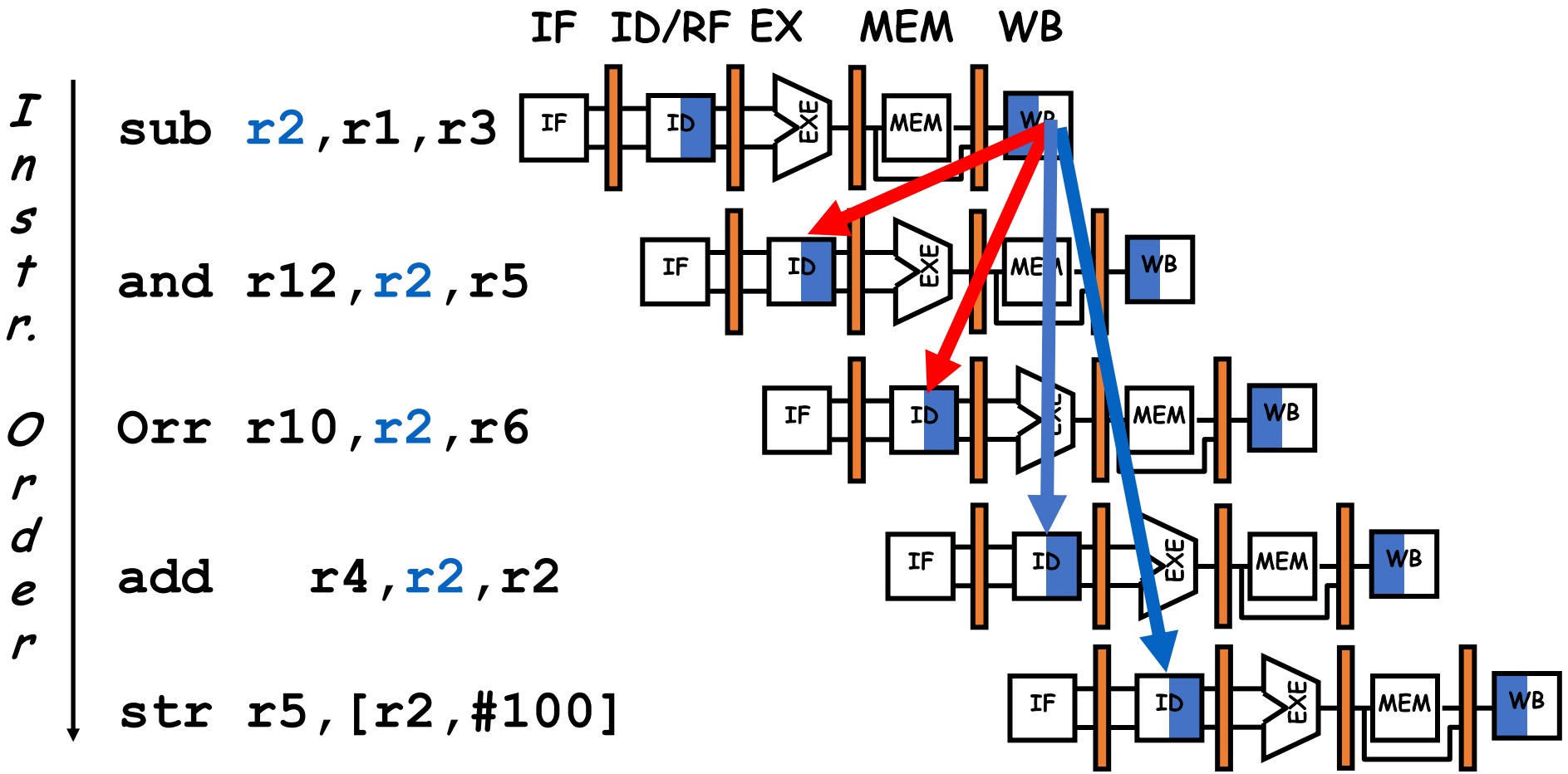
```
sub  R2, R1, R3
and  R12, R2, R5
orr  R10, R6, R2
add  R4, R2, R2
str  R5, [R2, #100]
```

```
sub  R2, R1, R3
NOP or MOV R0, R0
NOP or MOV R0, R0

and  R12, R2, R5
orr  R10, R6, R2
add  R4, R2, R2
str  R5, [R2, #100]
```

Microprocessor & Computer Architecture (µpCA)

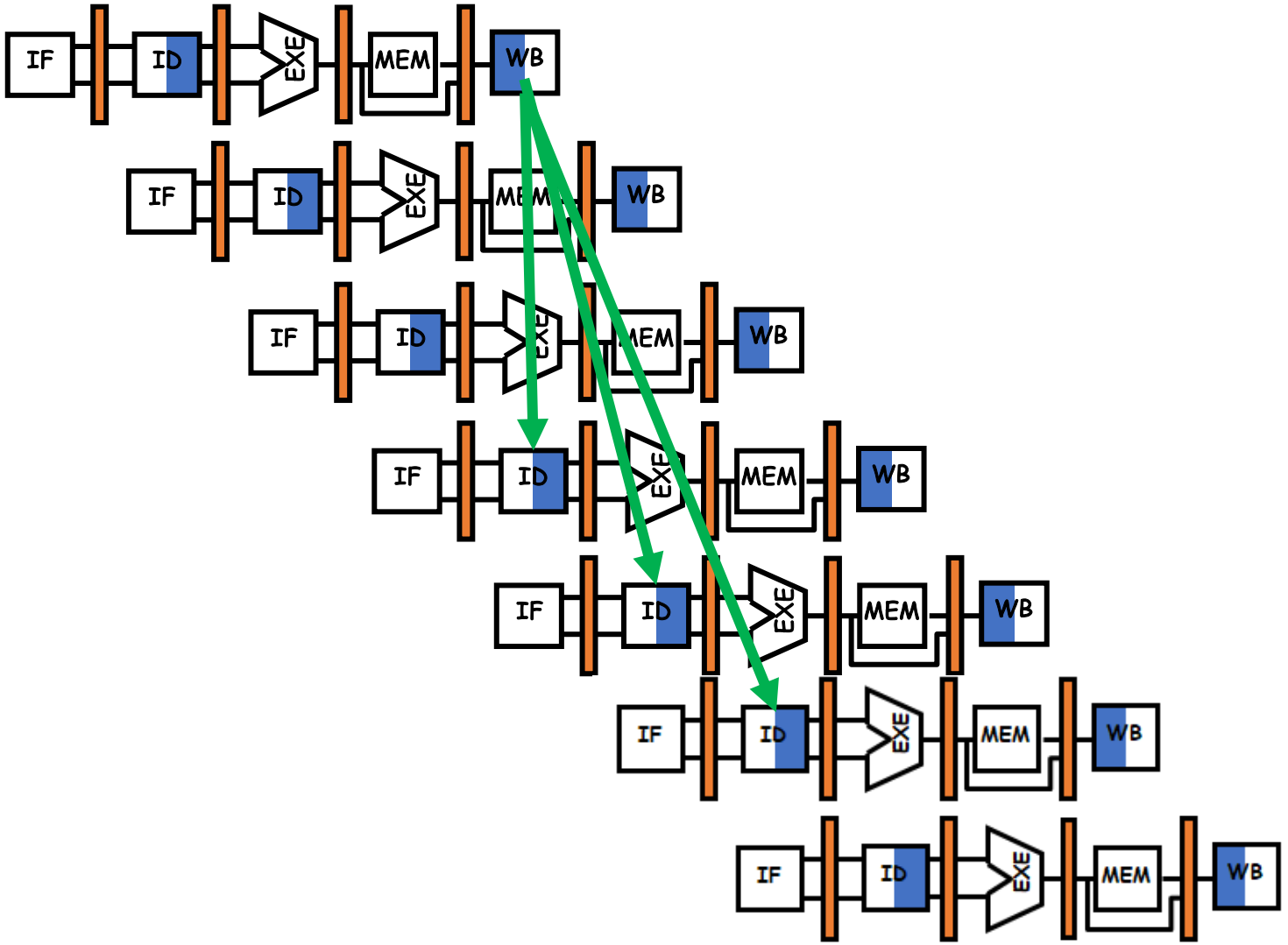
Software Solution 2→ By Compiler



Microprocessor & Computer Architecture (µpCA)

Software Solution 2→ By Compiler

```
sub r2,r1,r3
mov r0,r0
mov r0,r0
and r12,r2,r5
orr r10,r2,r6
add   r4,r2,r2
str r5,[r2,#100]
```



Where are NOPs needed?

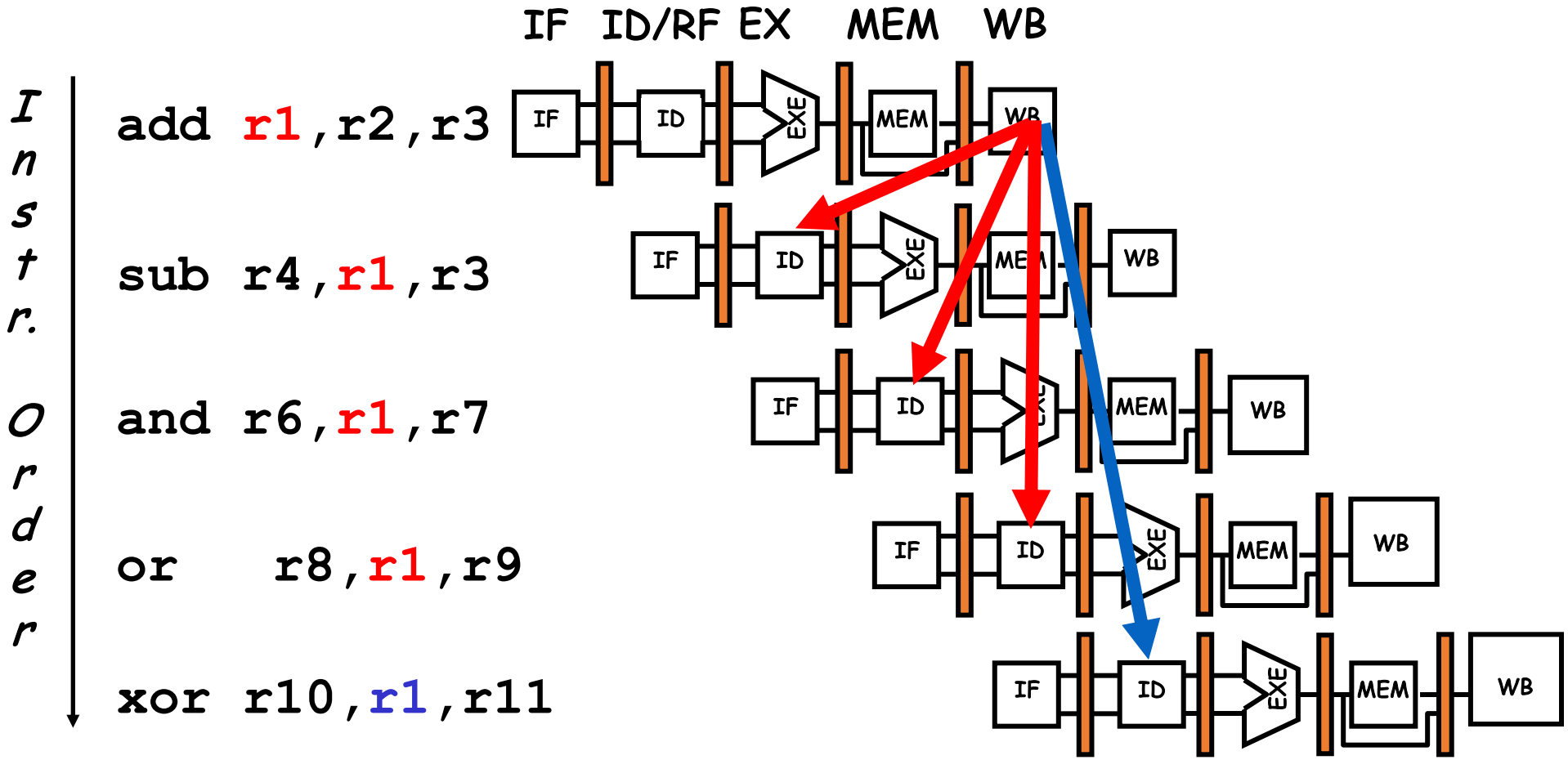
```
sub   R2, R1, R3
and   R4, R2, R5
orr   R8, R2, R6
add   R9, R4, R2
rsb   R1, R6, R7
```

```
sub   R2, R1, R3
NOP
NOP
and   R4, R2, R5
orr   R8, R2, R6
rsb   R1, R6, R7
add   R9, R4, R2
```

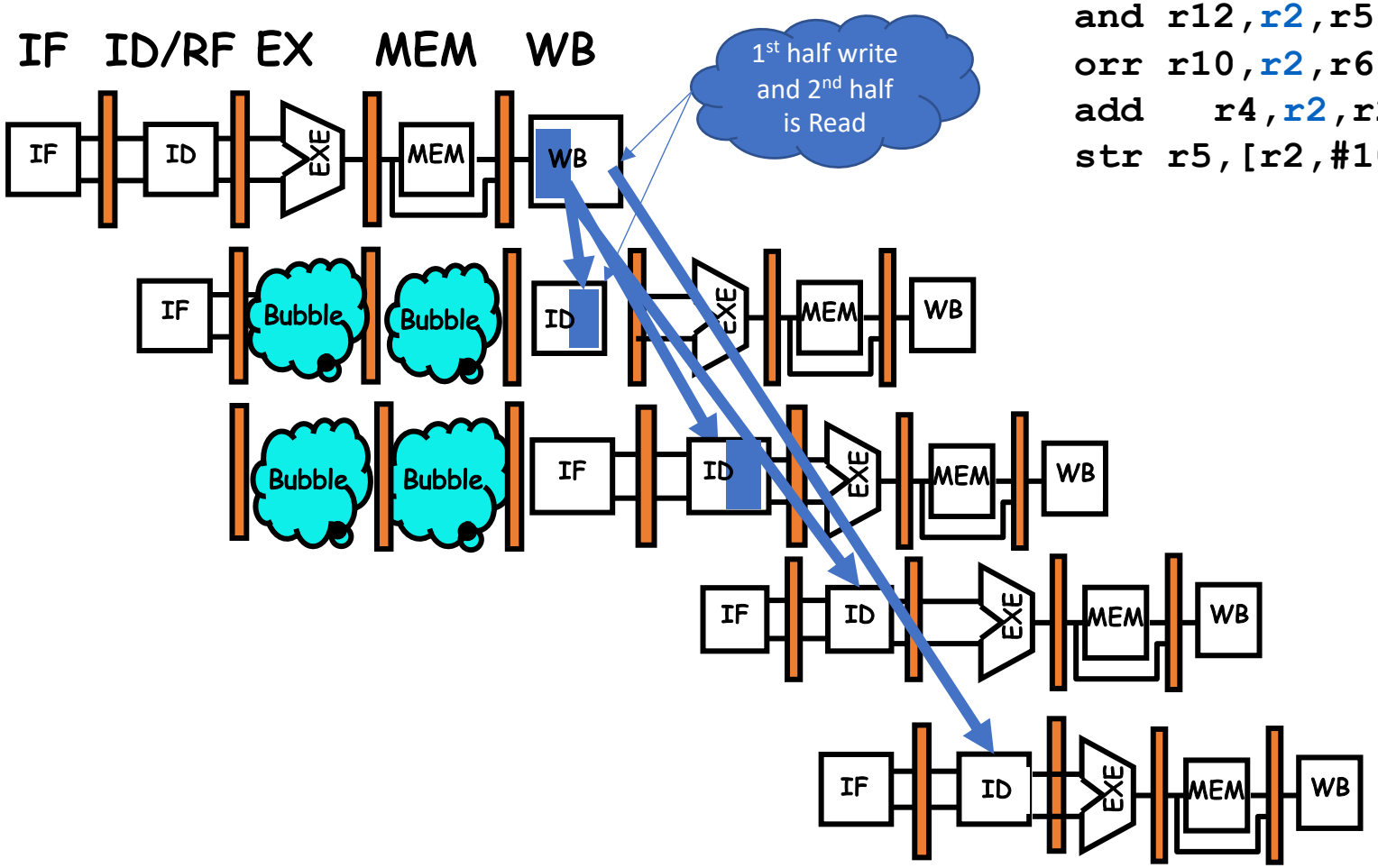
Attacking Data Hazard

- In Software
  - Solution 1: Re-order instructions
  - Solution 2: Insert independent instructions (or no-ops) Ex: MOV R0, R0
- **In Hardware**
  - **Solution 1: Insert bubbles (i.e. stall the pipeline)**
  - Solution 2: Data Forwarding

Hardware Solution 1



Hardware Solution 1




```
sub r2,r1,r3
and r12,r2,r5
orr r10,r2,r6
add   r4,r2,r2
str r5,[r2,#100]
```

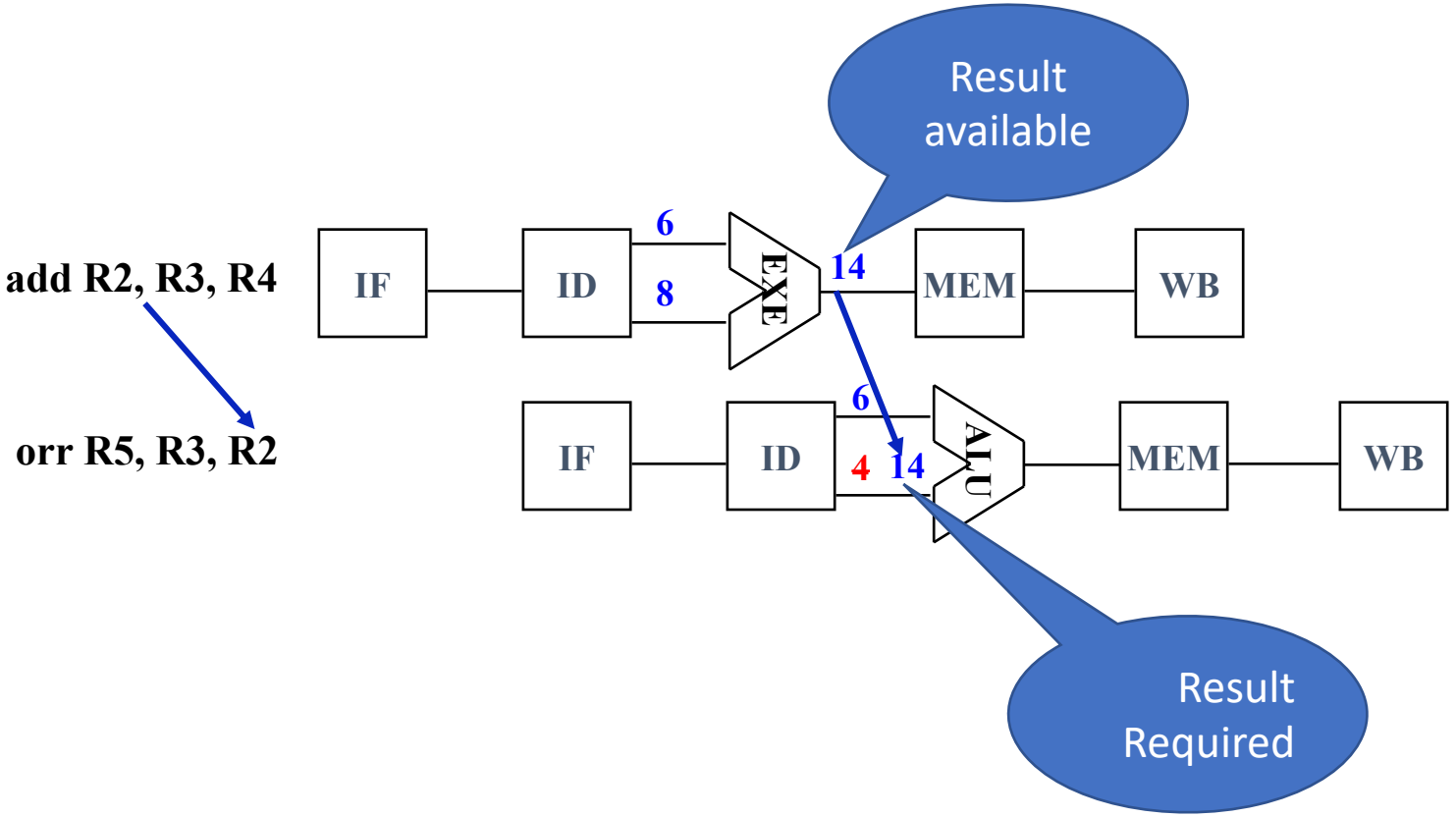
Attacking Data Hazard

- In Software
  - Solution 1: Re-order instructions
  - Solution 2: Insert independent instructions (or no-ops) Ex: MOV R0, R0
- **In Hardware**
  - Solution 1: Insert bubbles (i.e. stall the pipeline)
  - **Solution 2: Data Forwarding**

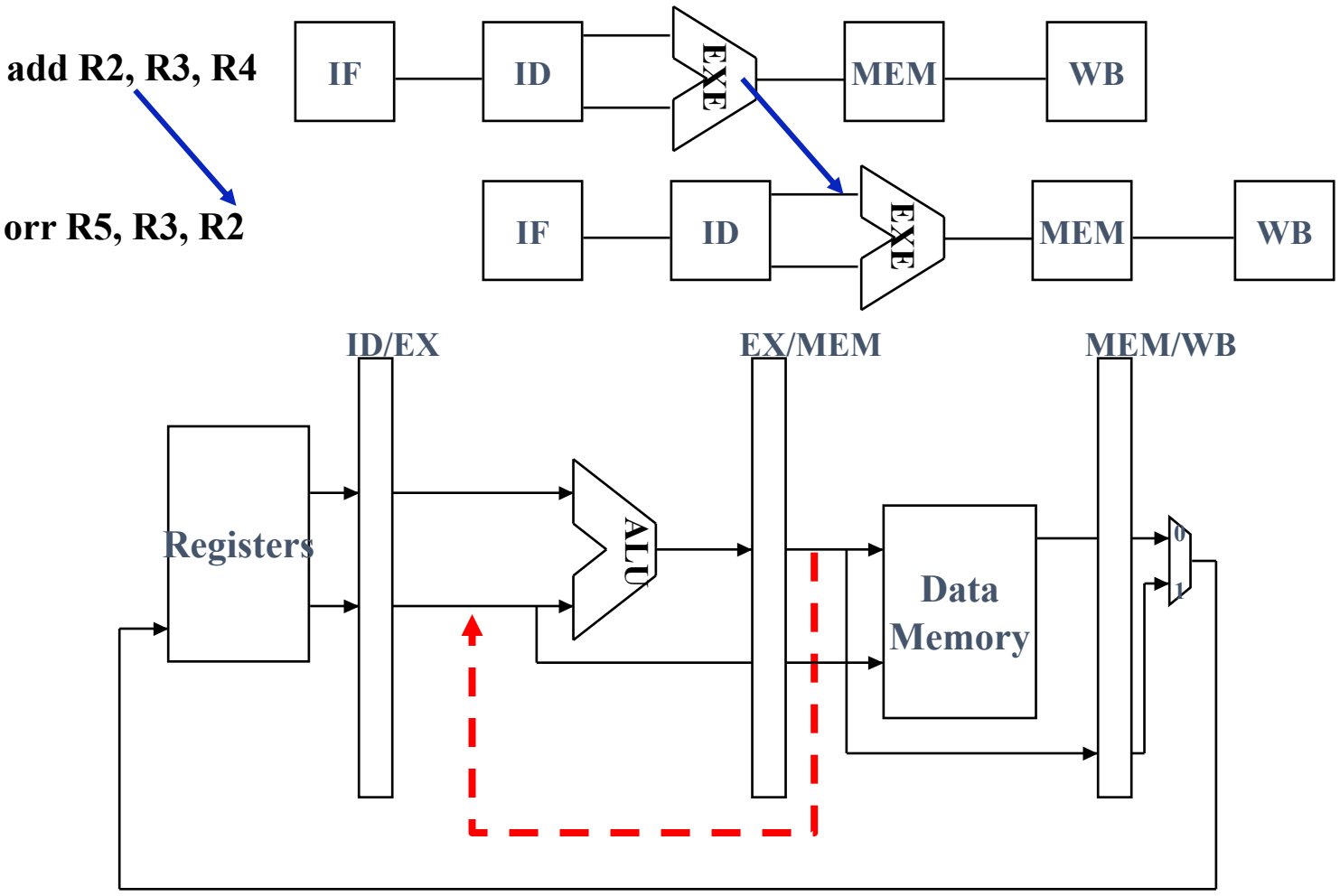
Hardware Solution 2: Data Forwarding

We could avoid stalling if we could get to EX stage the ALU output from "previous instruction" to ALU input for the "current instruction"

Suppose initially, register i holds the number 2i (ie. R3 = 6; R6=12; R8 = 16…)



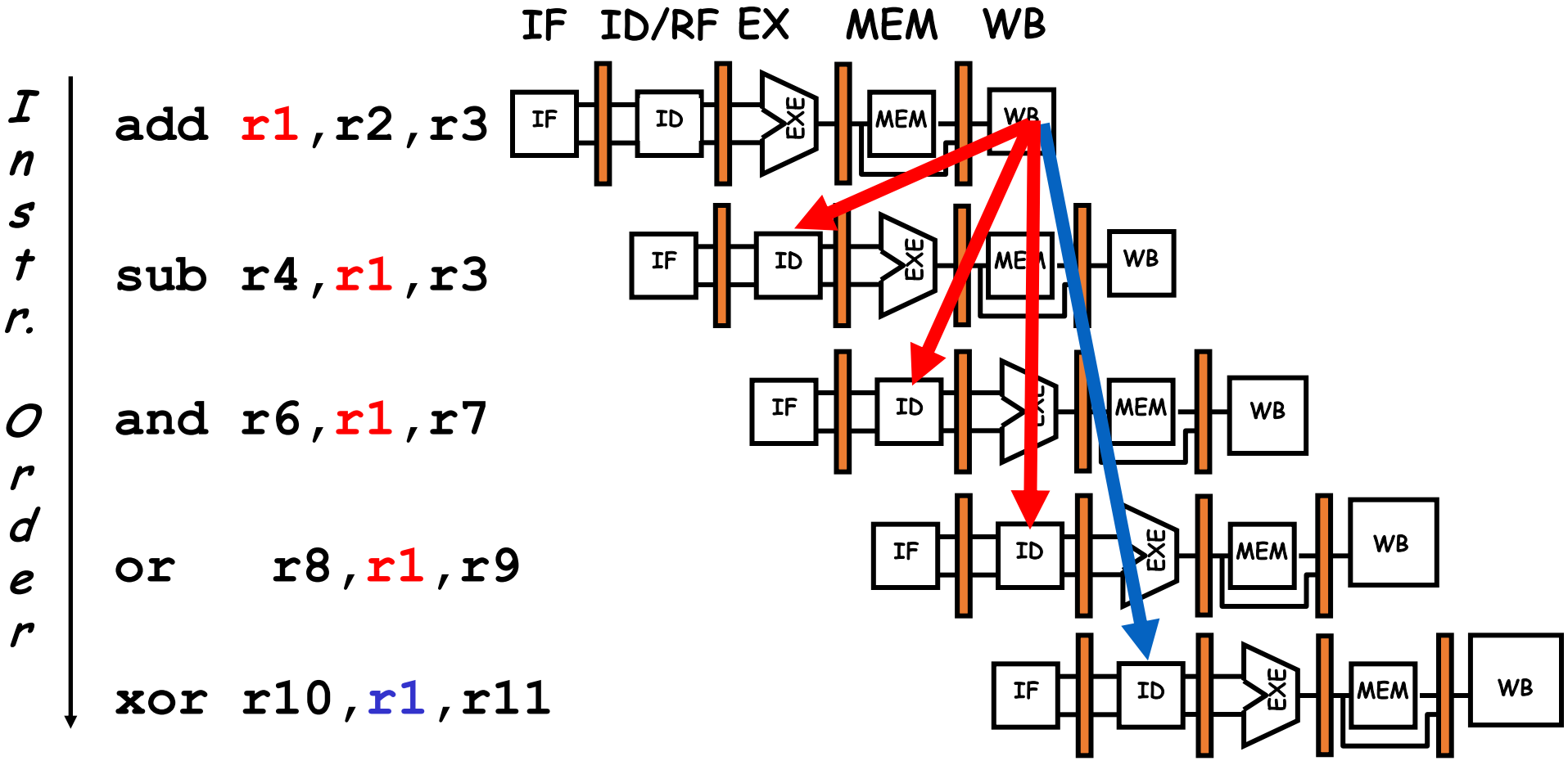
Hardware Solution 2: Data Forwarding



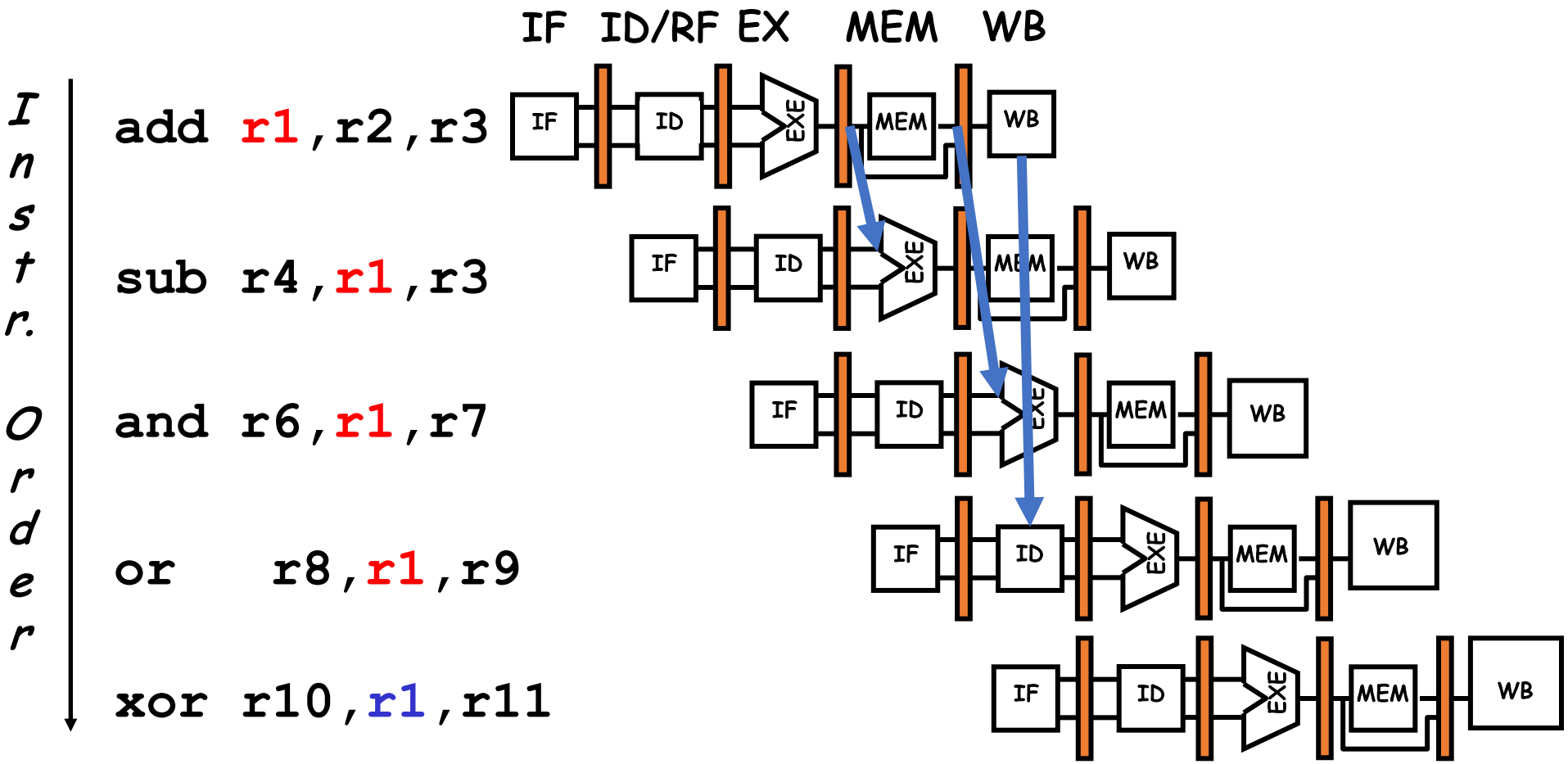
Hardware Solution 2: Data Forwarding

- Also known as:
  - Register –bypassing
  - Short-circuiting
- Forwarding handles hazards at both
  - EX stage
  - MEM stage

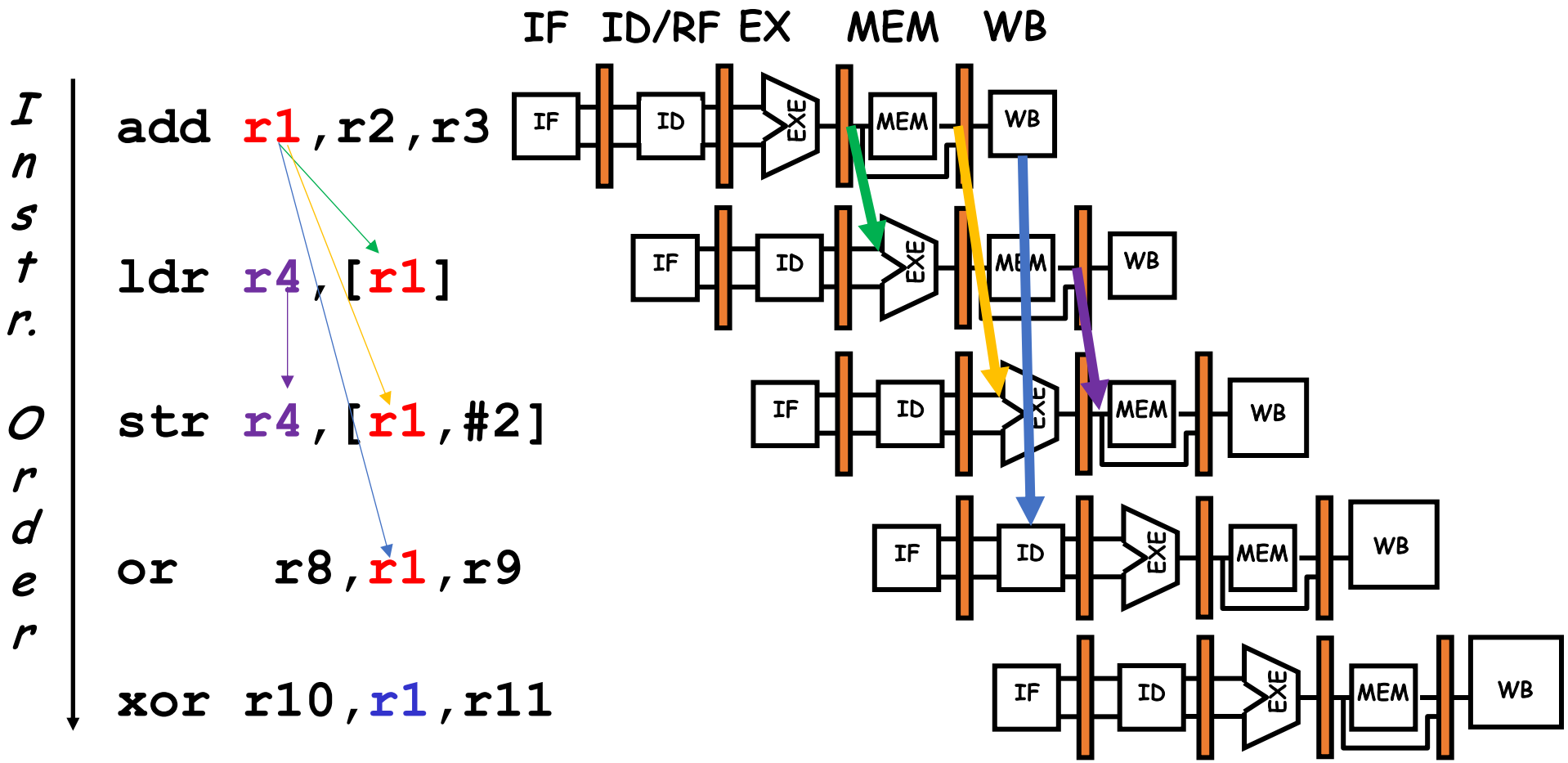
Hardware Solution 2: Data Forwarding



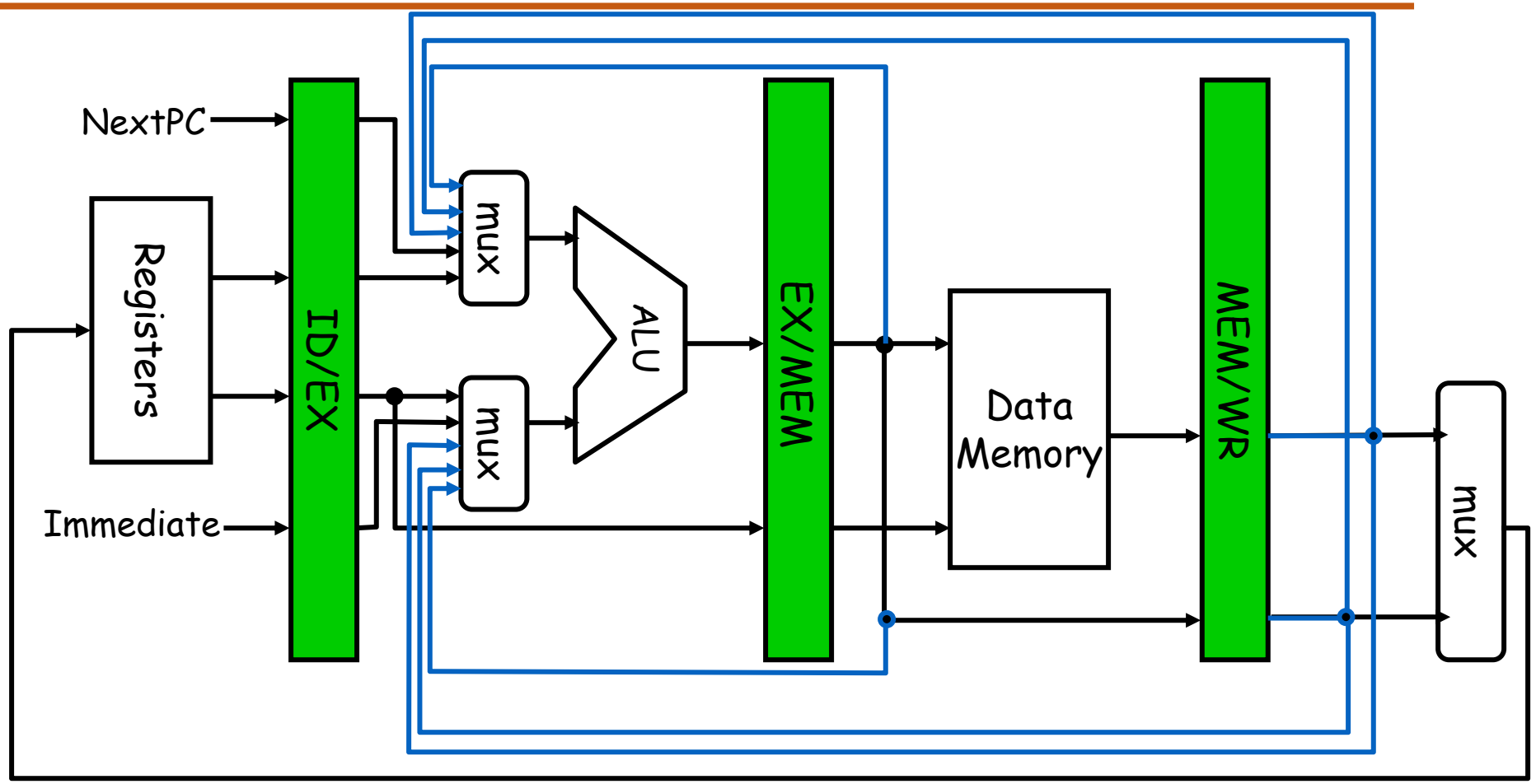
Hardware Solution 2: Data Forwarding



Hardware Solution 2: Data Forwarding



Hardware change for Forwarding



What circuit detects and resolves this hazard?

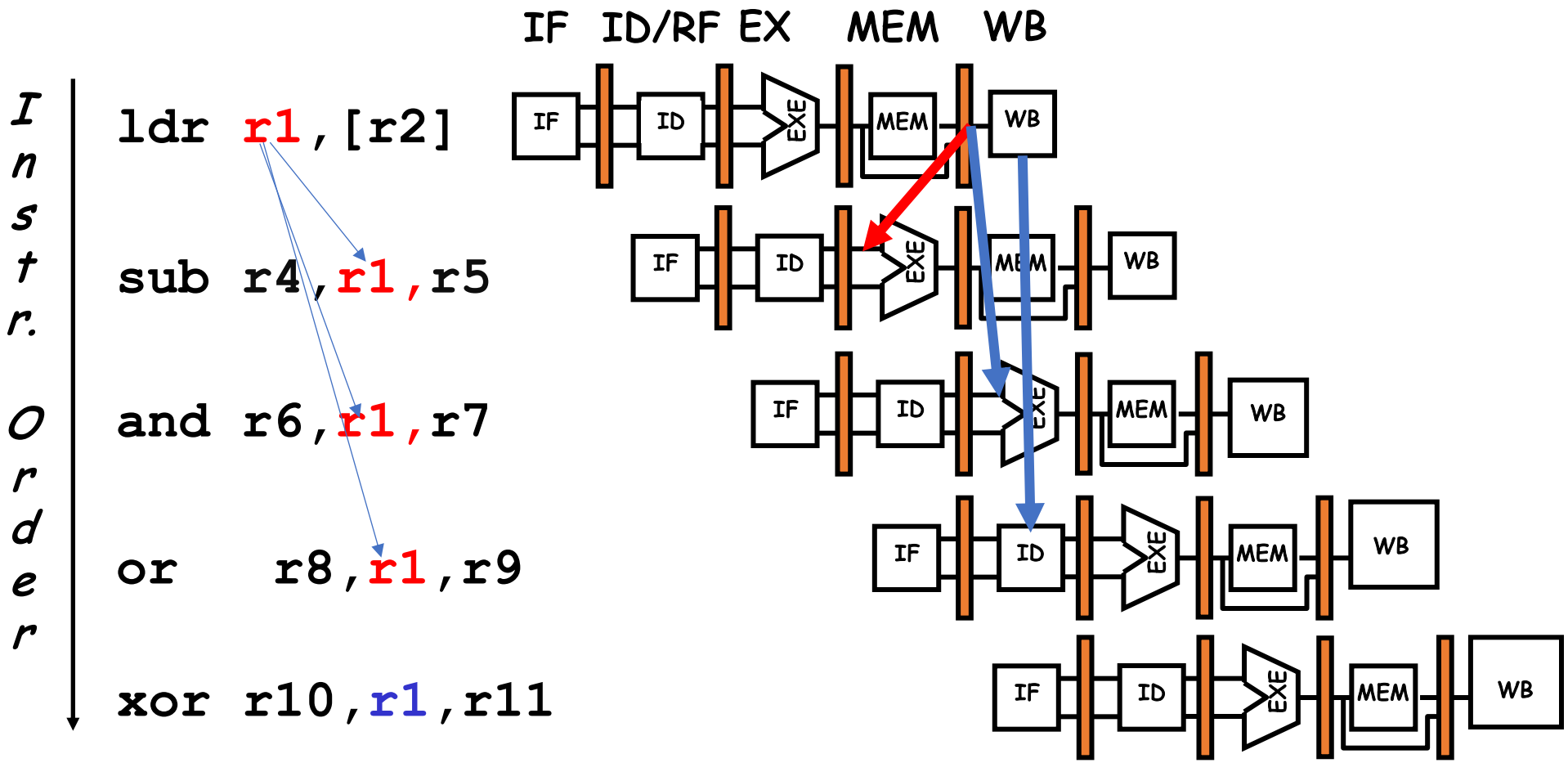
Does Forwarding eliminate all hazards??

Consider this example:

```
LDR     R0, [R1, #60]
ADD     R2, R0, R4
```

NO! You may need to stall after loads

Hardware Solution 2: Data Forwarding

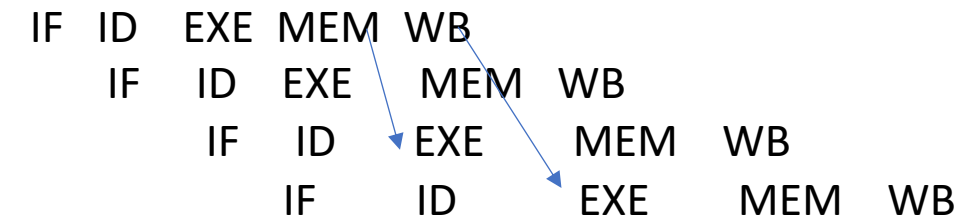


Think About It

```
ldr    R7, [R2]
ldr    R6, [R2, #4]
add    R4, R5, R6
str    R6, [R2, #4]
```

With forwarding we need to find only one independent instructions to place between them,
swapping the ldr instructions works:

```
ldr    R6, [R2, #4]
ldr    R7, [R2]
add    R4, R5, R6
str    R6, [R2, #4]
```



Without forwarding we need two independent instructions to place between them, so in addition a nop is added.

```
ldr    R6, [R2, #4]
ldr    R7, [R2]
nop
add    R4, R5, R6
str    R6, [R2, #4]
```

Why?

Think About It

a = b + c;

d = e – f;

Before:

```
ldr   R2, =a
ldr   R3, =b
add   R1, R2, R3
str   R1, =a
ldr   R2, =e
ldr   R3, =f
sub   R1, R2, R3
str   R1, =d
```

Eliminate dependency by renaming registers

```
ldr   R2, =a
ldr   R3, =b
add   R1, R2, R3
str   R1, =a
ldr   R5, =e
ldr   R6, =f
sub   R4, R5, R6
str   R4, =d
```

After Reordering & Assuming Data Forwarding

```
ldr   R2, =a
ldr   R3, =b
ldr   R5, =e
add   R1, R2, R3
ldr   R6, =f
str   R1, =a
sub   R4, R5, R6
str   R4, =d
```

Draw the timing diagram check, if stalls are required?

Next Session

Control Hazards

THANK YOU

Dr. D. C. Kiran
Department of Computer Science and Engineering

dckiran@pes.edu

9829935135