CS230: Digital Logic Design and Computer Architecture

Lecture 13: Data/Control Hazards

https://www.cse.iitb.ac.in/~biswa/courses/CS230/autumn23/main.html

https://www.cse.iitb.ac.in/~biswa/

The ideal world

- Uniform Sub-operations
  - –Operation (op) can be partitioned into uniform-latency sub-ops
- Repetition of Identical Operations
  - –Same ops performed on many different inputs
- Independent Operations
  - –All ops are mutually independent

The real world

- Uniform Sub-operations NO
  - Operation can be partitioned into uniform-latency sub-ops

- Repetition of Identical Operations NO
  - Same ops performed on many different inputs

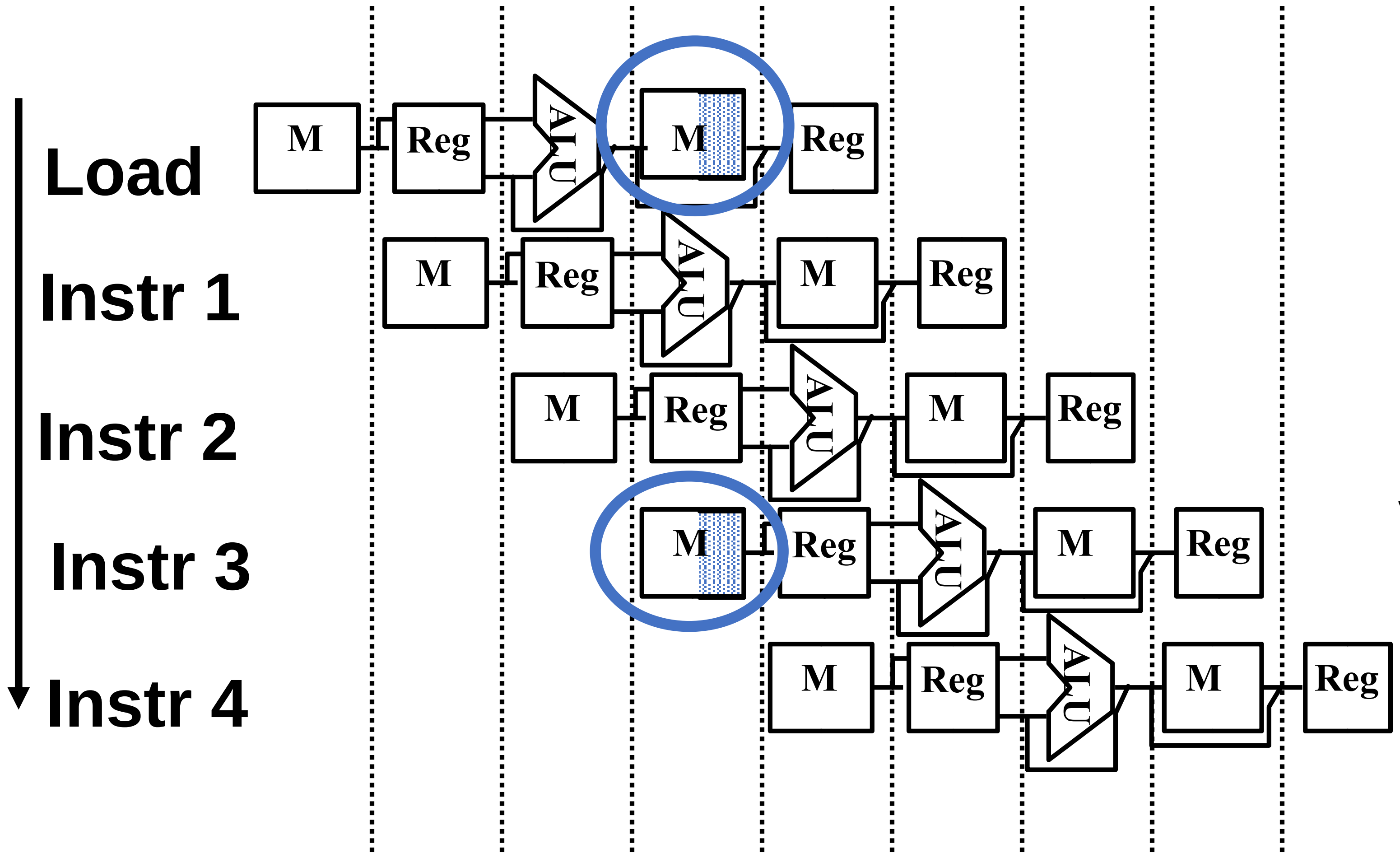
- Independent Operations NO
  - All ops are mutually independent

Structural/Data/Control Hazards

What is a hazard ? Anything that prevents an instruction to move ahead in the pipeline.

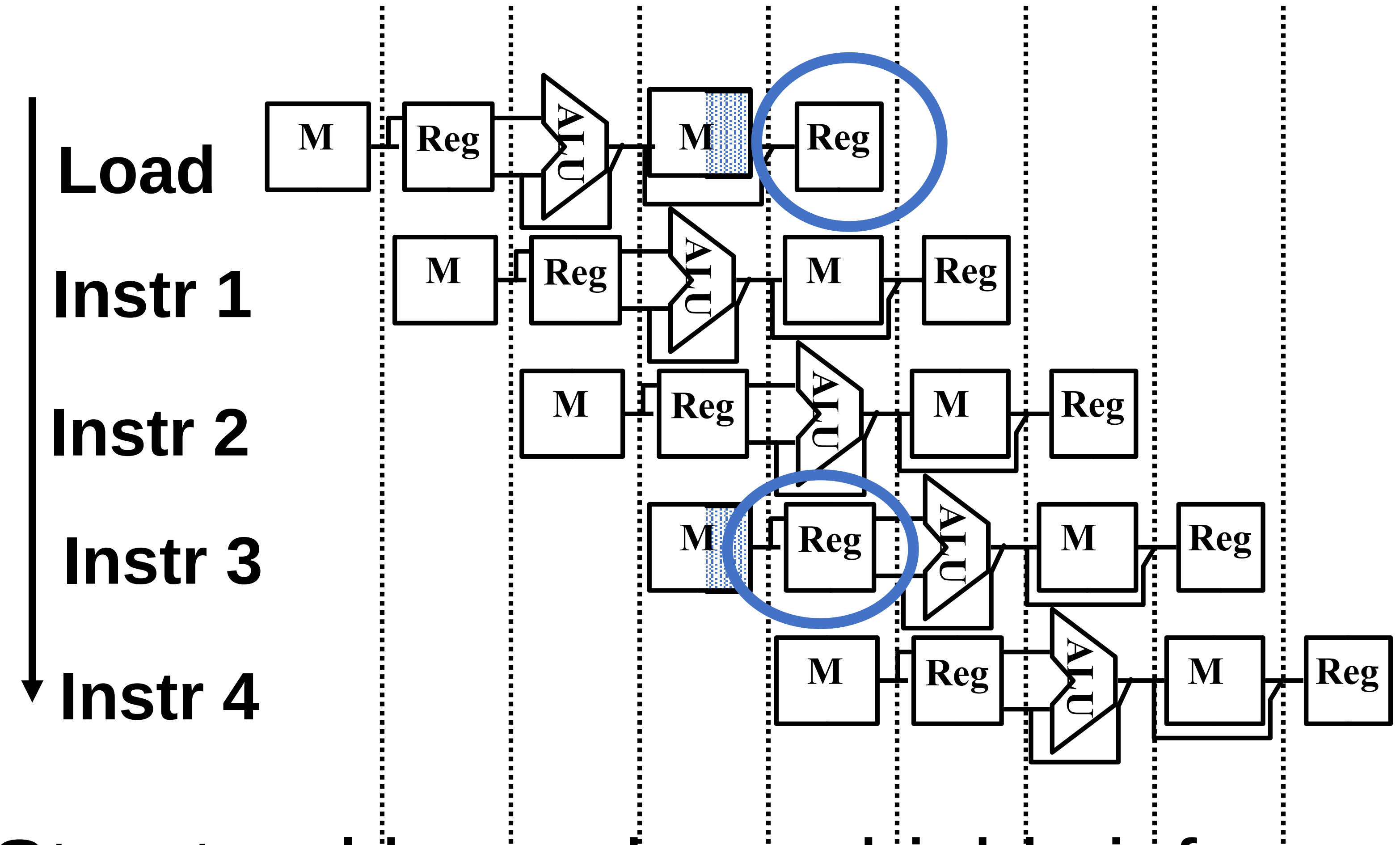
Structural: Resource conflicts, two instructions want to access the same structure on the same clock cycle.

An Example with unified (single) memory



Can’t read same memory twice in same clock cycle

What about registers?



Can read/write the register file (same register) in same clock cycle ☺ but.. Real picture is different

Remember: Edge-triggered

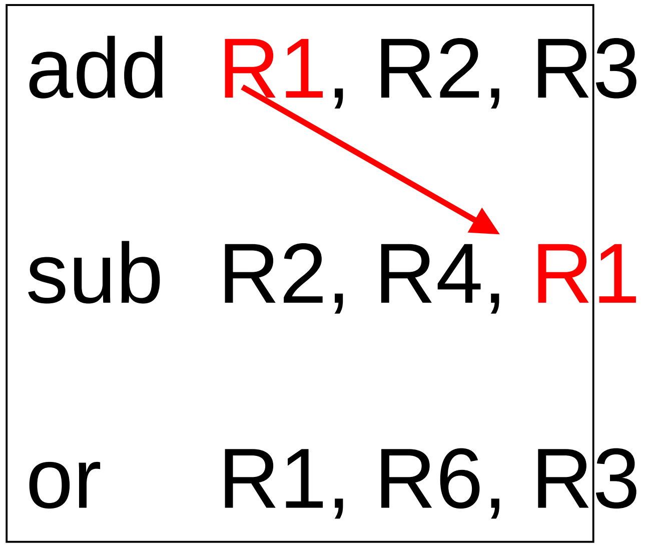
Structural hazards are highly infrequent

Data Hazards

INSTRUCTION DEPENDS ON THE RESULT (DATA) OF PREVIOUS INSTRUCTION(S).

HAZARDS HAPPEN BECAUSE OF DATA DEPENDENCES.

Data dependences (hazards)

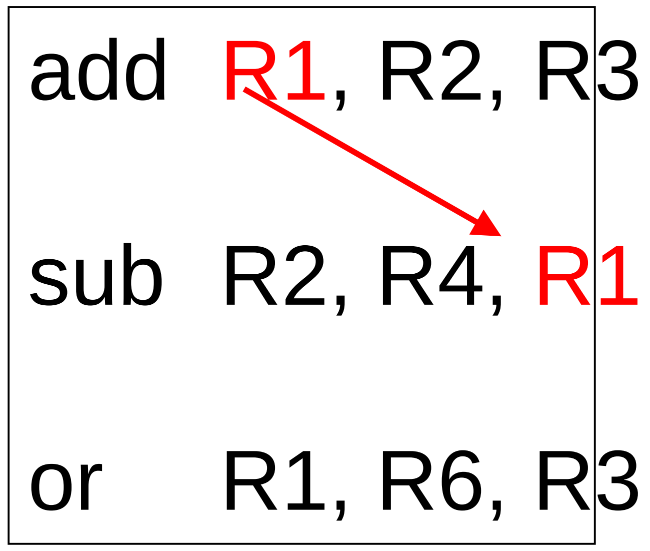


```
add  R1, R2, R3

sub  R2, R4, R1

or   R1, R6, R3
```

read-after-write (RAW)

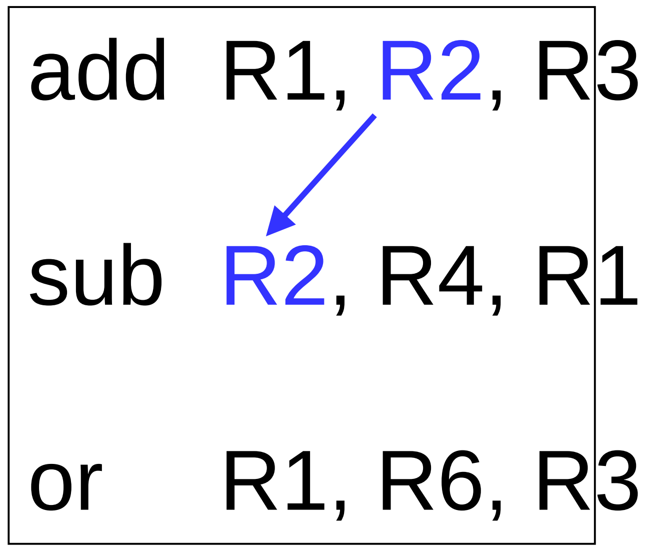
True dependence

Data dependences (hazards)



```
add  R1, R2, R3

sub  R2, R4, R1

or   R1, R6, R3
```

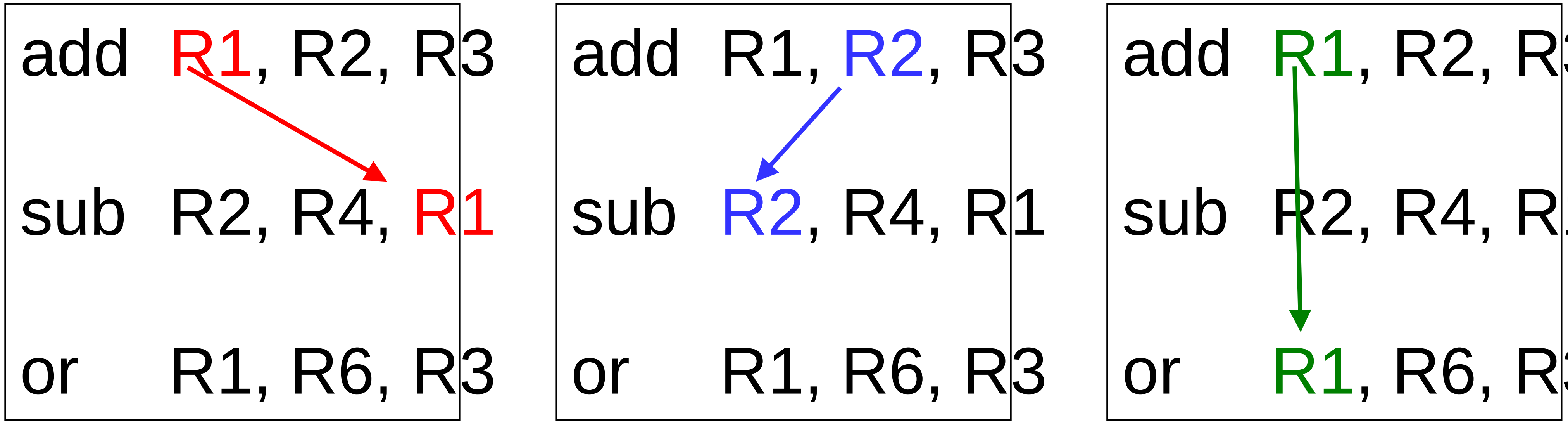
read-after-write
(RAW)
True dependence



```
add  R1, R2, R3

sub  R2, R4, R1

or   R1, R6, R3
```

write-after-read
(WAR)
anti dependence

Data dependences (hazards)



```
add  R1, R2, R3

sub  R2, R4, R1

or   R1, R6, R3
```

read-after-write
(RAW)
True dependence

```
add  R1, R2, R3

sub  R2, R4, R1

or   R1, R6, R3
```

write-after-read
(WAR)
anti dependence

```
add  R1, R2, R3

sub  R2, R4, R1

or   R1, R6, R3
```

write-after-write
(WAW)
output dependence

Data Hazards

Read-After-Write (*RAW*)

- Read must wait until earlier write finishes

Anti-Dependence (*WAR*)

- Write must wait until earlier read finishes

- *Output Dependence* (*WAW*)
  - Earlier write can't overwrite later write

  (WAW hazard: not possible with vanilla 5-stage pipeline)

Data Hazards (Examples)

Time (clock cycles)

Instr. Order

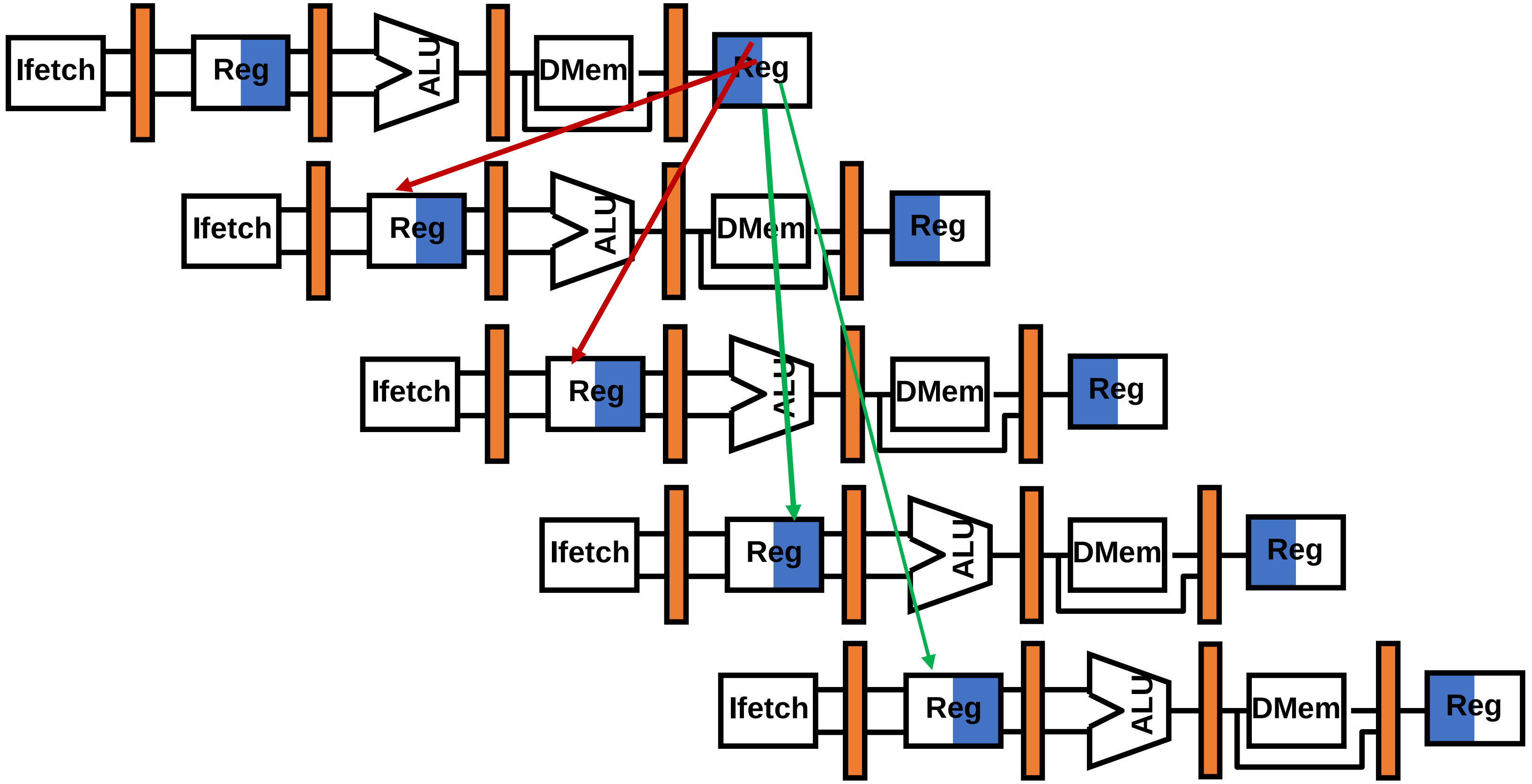
add r1,r2,r3

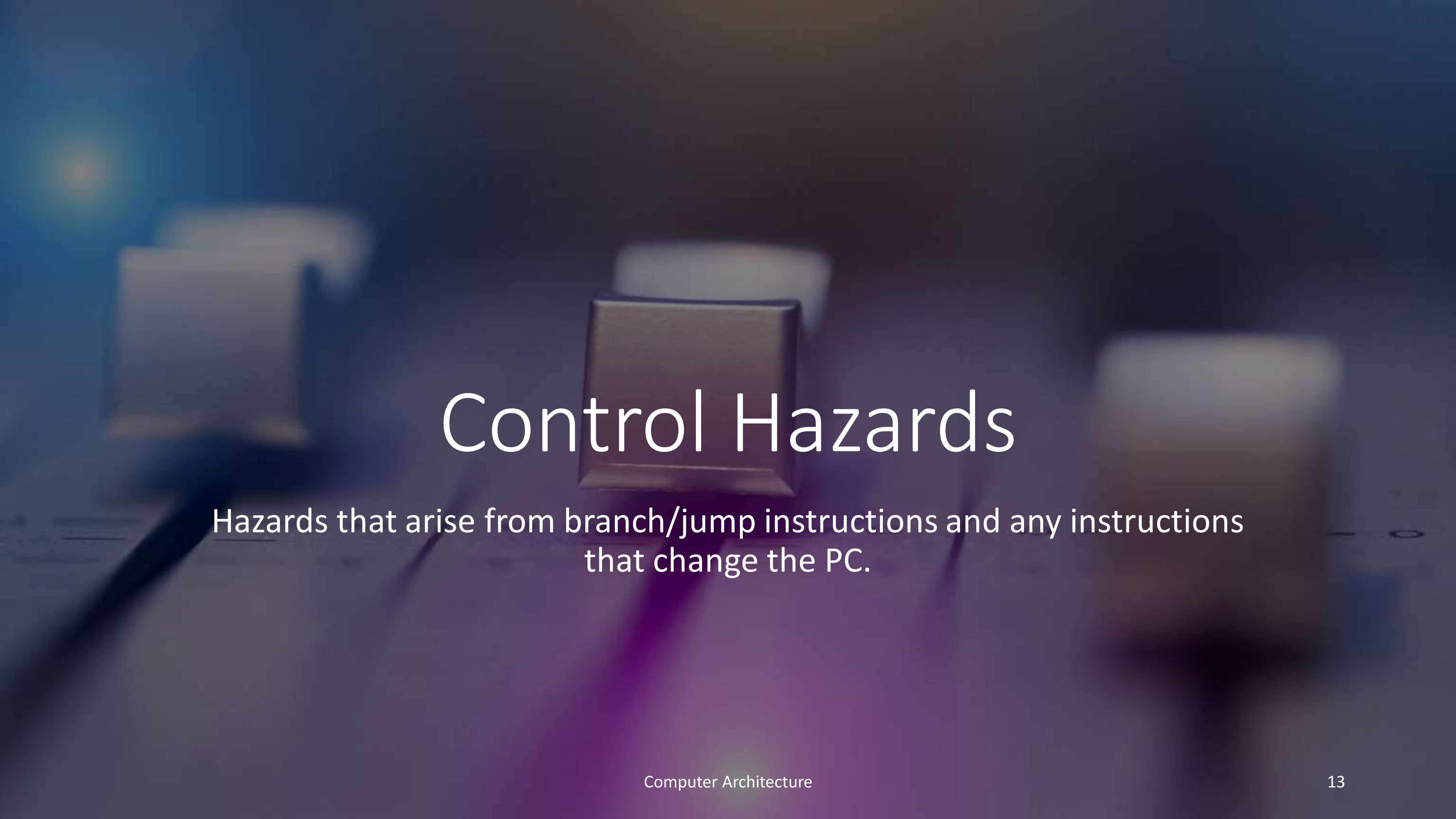
sub r4,r1,r3

and r6,r1,r7

or   r8,r1,r9

xor r10,r1,r11





Control Hazards

Hazards that arise from branch/jump instructions and any instructions that change the PC.

An Example

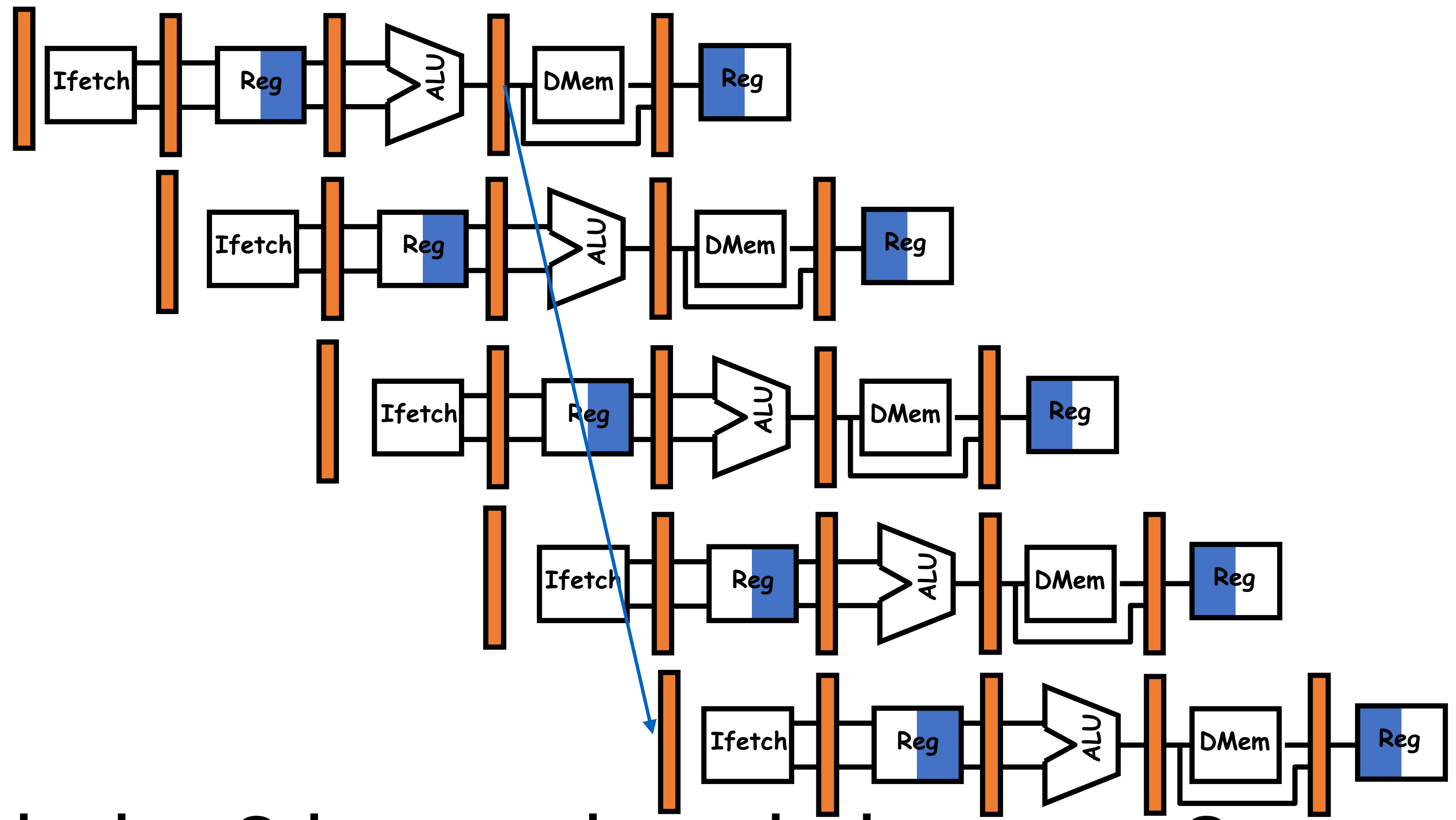
10: beq r1,r3,36

14: and r2,r3,r5 ☹

18: or r6,r1,r7 ☹

22: add r8,r1,r9 ☹

50: xor r10,r1,r11



What do you do with the 3 instructions in between?

How do you do it?

What happens on a hazard?

Instruction cannot move forward

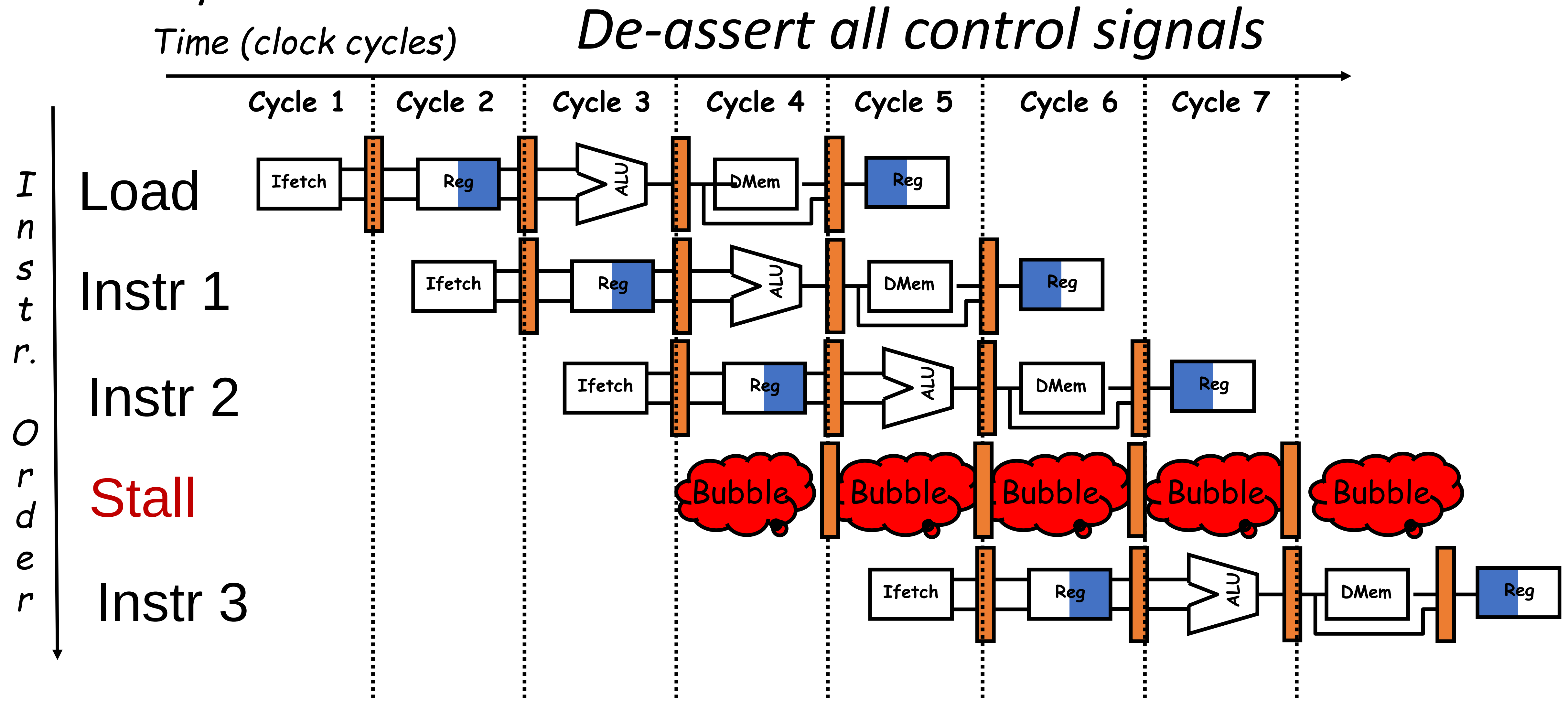
Instruction must wait to get the hazard resolved.

The pipeline must stall ☹

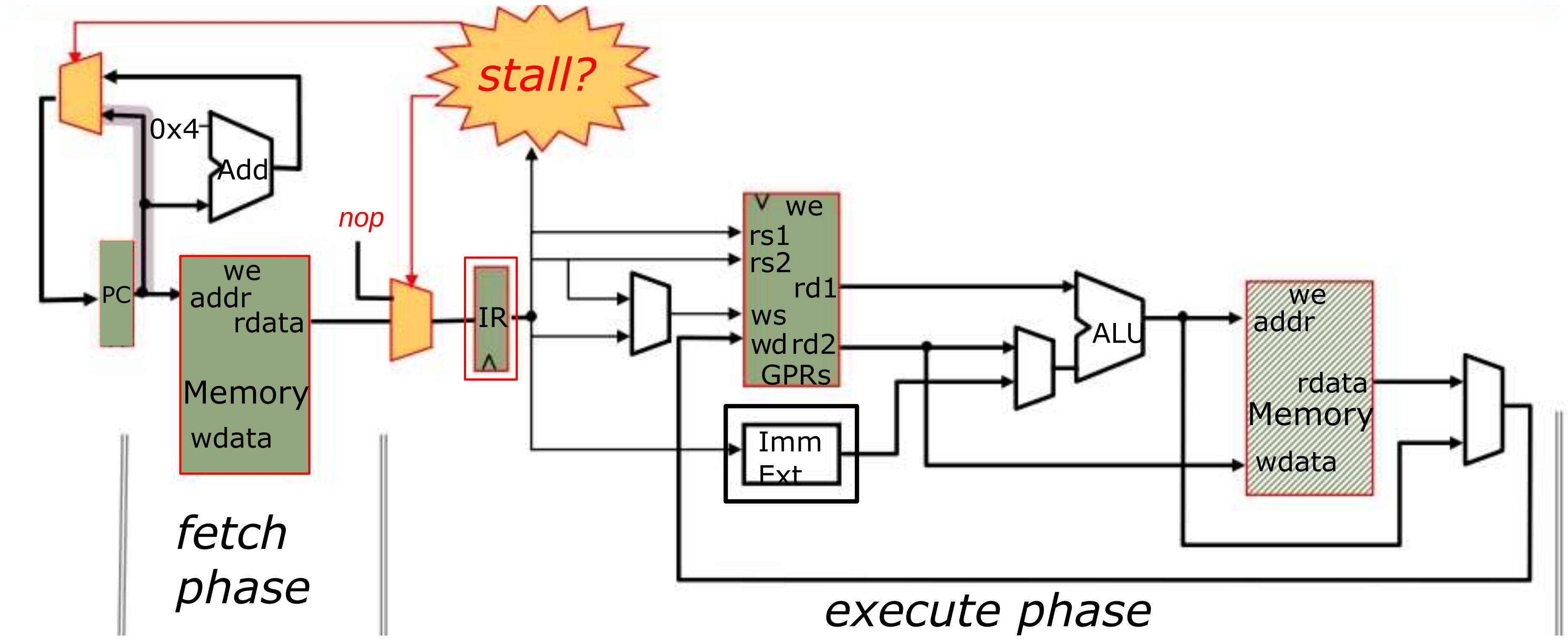
It is like air bubbles in pipelines

Stall/Bubble

De-assert all control signals



How to implement a stall?



Don’t fetch a new instruction and don’t change the PC

insert a nop (compiler way)

An example of an NOP

sll $0 $0 (in MIPS)

Simple Example (no bubbles)

add r3, r2, r1

add r6, r5, r4

C1	C2	C3	C4	C5	C6	C7	C8
IF1	ID1	IE1	IM1	IWB1			
	IF2	ID2	IE2	IM2	IWB2		

Simple Example (2 bubbles)

add r3, r2, r1

add r6, r5, r3

C1	C2	C3	C4	C5	C6	C7	C8
IF1	ID1	IE1	IM1	IWB1			
	IF2	ID2	ID2	ID2	IE2	IM2	IWB2

Control Hazard and NOPs

time

	t0	t1	t2	t3	t4	t5	t6	t7	. . . .
(I_1) 096: ADD	IF_1	ID_1	EX_1	MA_1	WB_1				
(I_2) 100: J 200		IF_2	ID_2	EX_2	MA_2	WB_2			
(I_3) 104: ADD			IF_3	nop	nop	nop	nop		
(I_4) 304: ADD				IF_4	ID_4	EX_4	MA_4	WB_4	

time

Resource Usage		t0	t1	t2	t3	t4	t5	t6	t7	. . . .
	IF	I_1	I_2	I_3	I_4	I_5				
	ID		I_1	I_2	nop	I_4	I_5			
	EX			I_1	I_2	nop	I_4	I_5		
	MA				I_1	I_2	nop	I_4	I_5	
	WB					I_1	I_2	nop	I_4	I_5

nop $\Rightarrow$ *pipeline bubble*

What happens to the speedup?

$$\text{Speedup} = \frac{\text{CPI unpipelined}}{\text{CPI pipelined}} = \frac{\text{CPI unpipelined}}{\text{ideal CPI} + \text{stalls/instructions}}$$

Ideal CPI=1, assume stages are perfectly balanced

Data Hazard Detector and stalls

Execute to decode:

EX/MEM.RegisterRd = ID/EX.RegisterRs

EX/MEM.RegisterRd = ID/EX.RegisterRt

Memory to decode:

MEM/WB.RegisterRd = ID/EX.RegisterRs

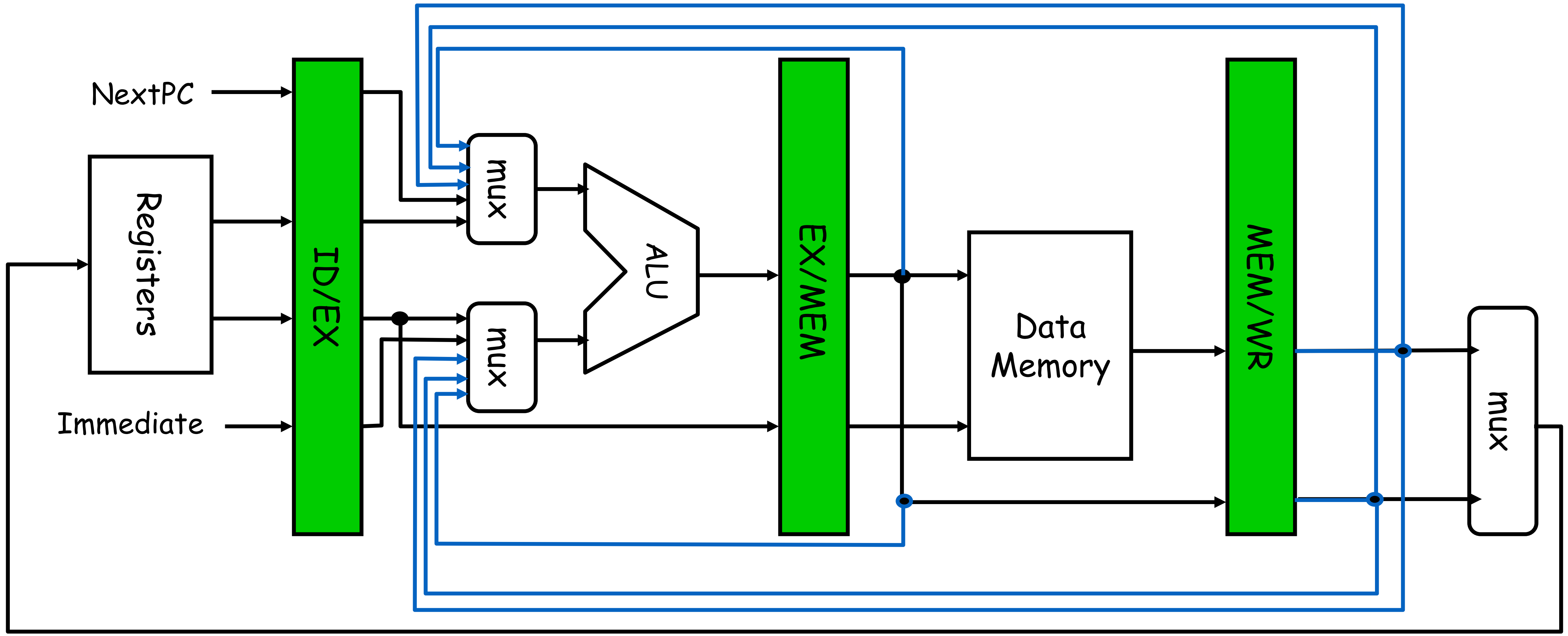
MEM/WB.RegisterRd = ID/EX.RegisterRt

what about instructions that do not write into the registers?

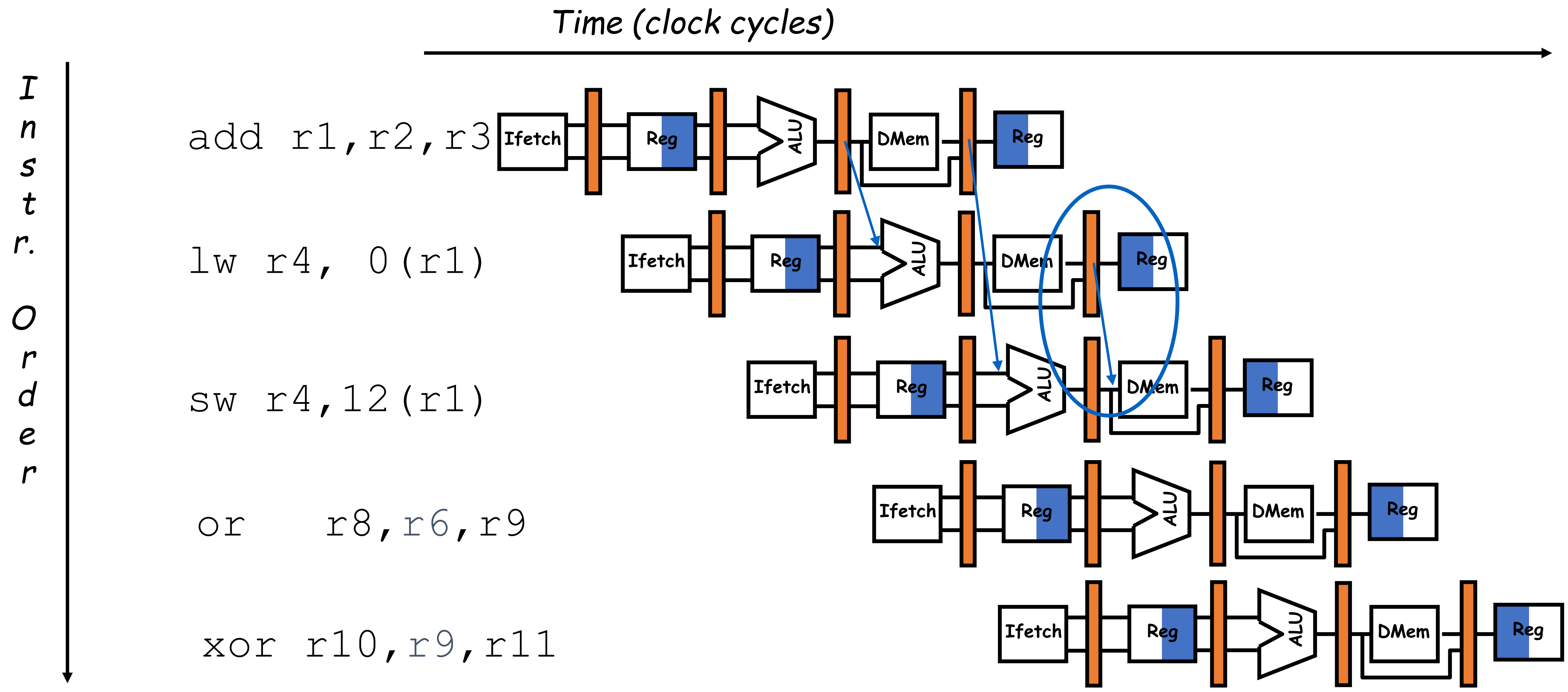
Bypassing

Route data as soon as possible after it is calculated to the earlier pipeline stage

Bypassing/forwarding: Updated Datapath



How does it help?



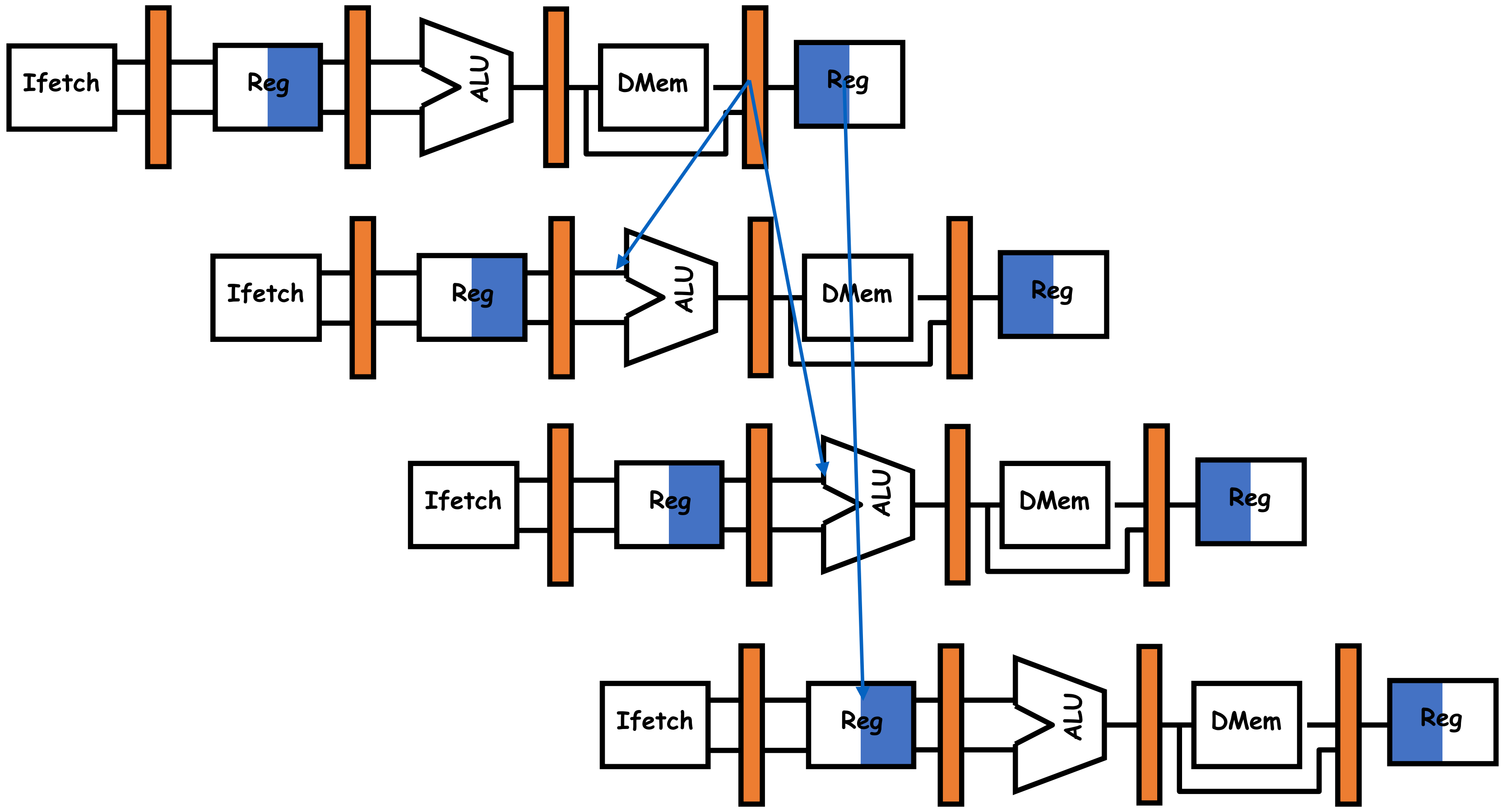
Computer Architecture

Does it help always?

Time (clock cycles)

Instr. Order

```
lw r1, 0(r2)

sub r4,r1,r6

and r6,r1,r7

or   r8,r1,r9
```


Ifetch
Reg
ALU
DMem
Reg