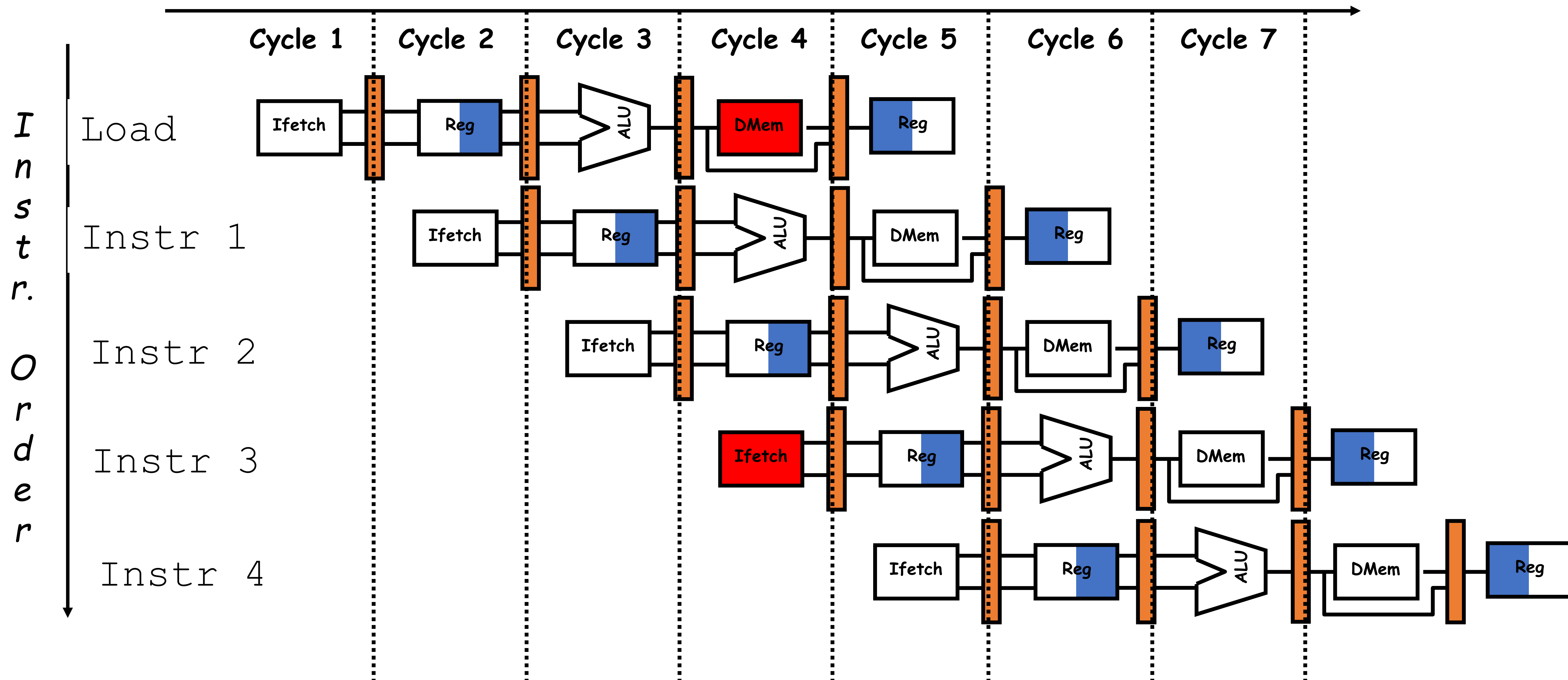
Lecture-6 (Pipeline Hazards)
CS422-Spring 2018

Biswa@CSE-IITK

Hazards

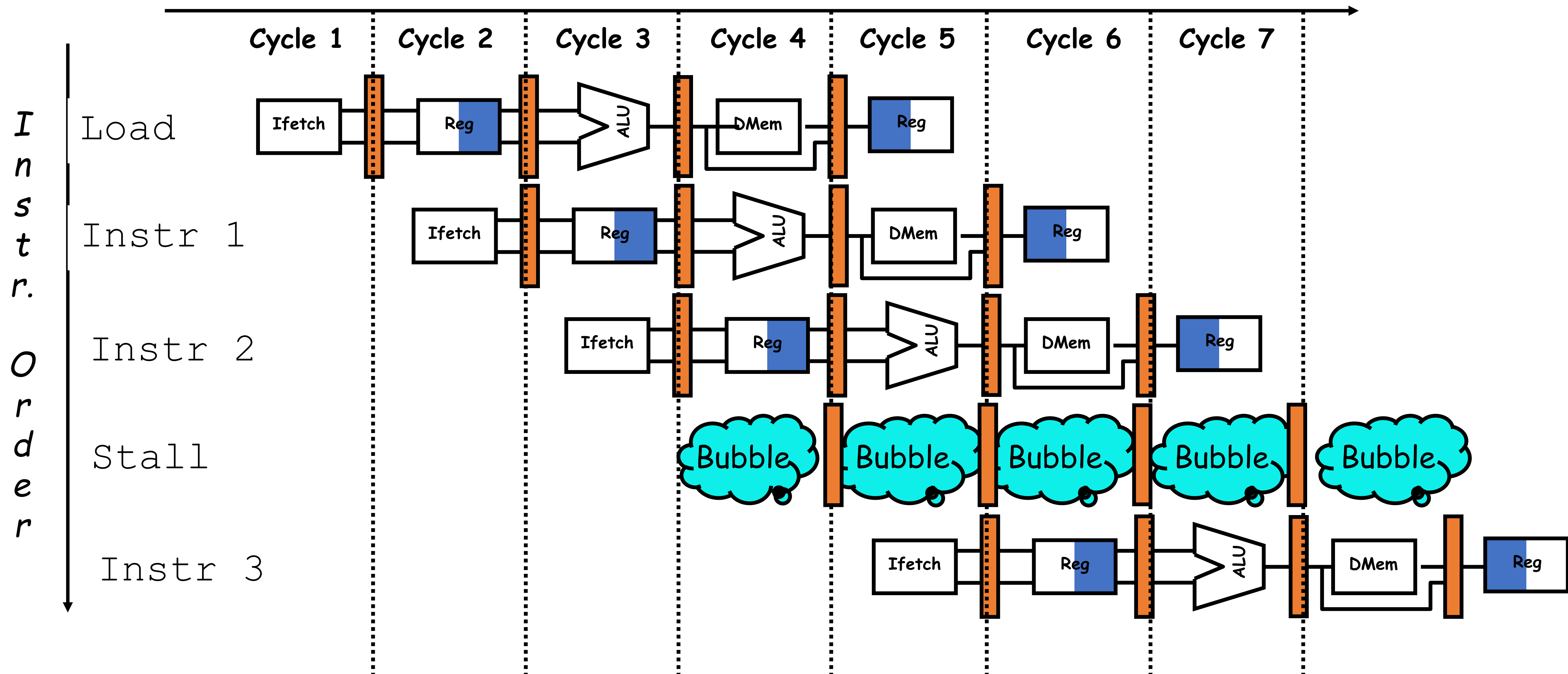
- Limits to pipelining: Hazards prevent next instruction from executing during its designated clock cycle
  - Structural hazards: HW cannot support this combination of instructions (single person to fold and put clothes away)
  - Data hazards: Instruction depends on result of prior instruction still in the pipeline (missing sock)
  - Control hazards: Caused by delay between the fetching of instructions and decisions about changes in control flow (branches and jumps).

One Memory Port/Structural Hazards

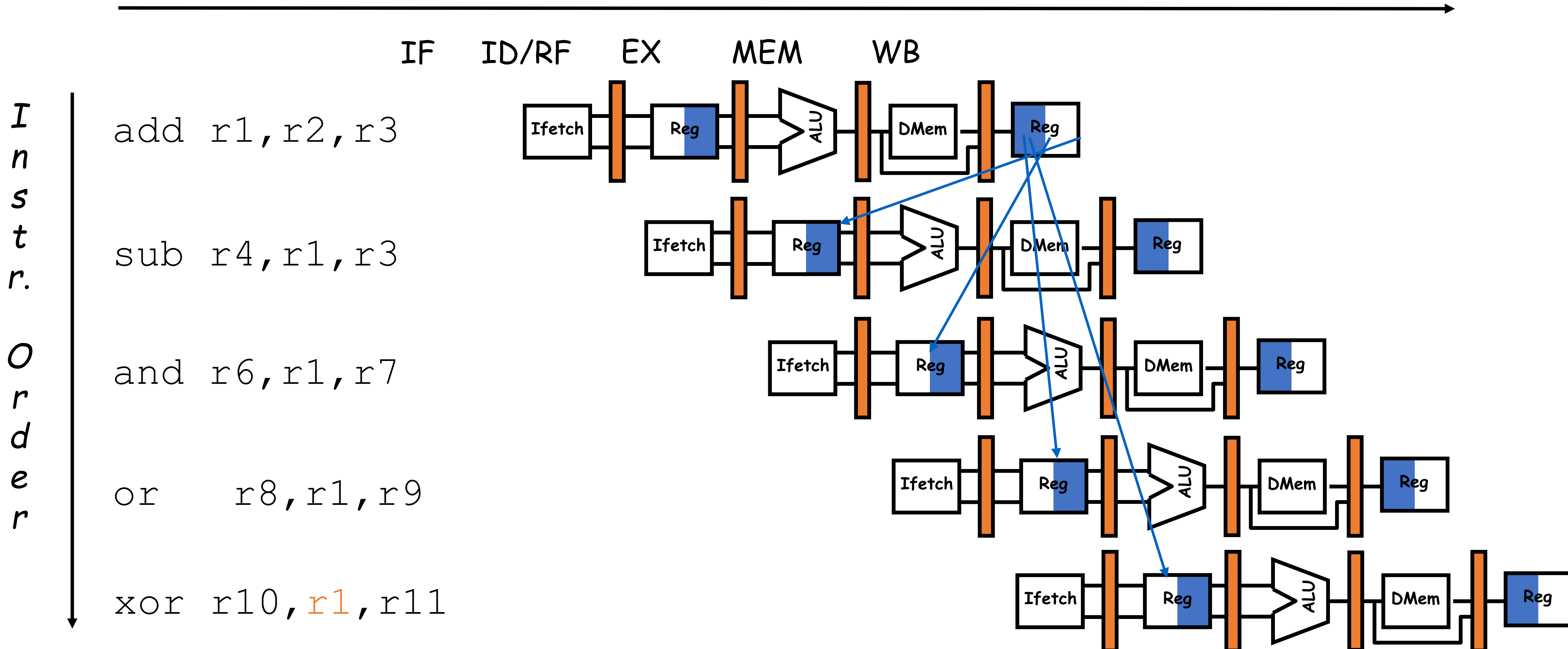


Why Separate Data and Instruction Caches?

Bubble

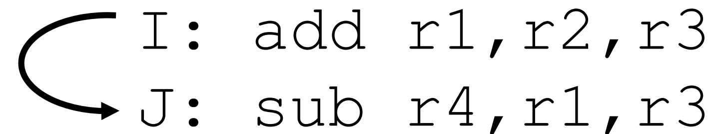


Data Hazards



RAW

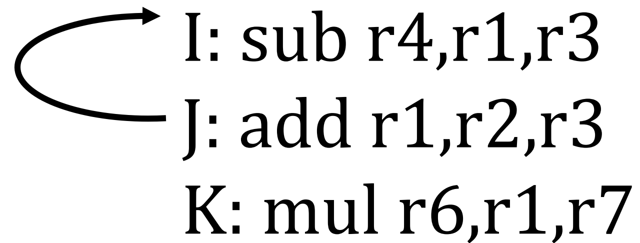
- Read After Write (RAW)
  $Instr_J$ tries to read operand before $Instr_I$ writes it



- Caused by a "Dependence" (in compiler nomenclature). This hazard results from an actual need for communication.

WAR

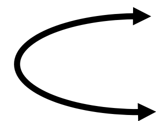
- Write After Read (WAR)
  $Instr_J$ writes operand *before* $Instr_I$ reads it



- Called an "anti-dependence" by compiler writers.
  This results from reuse of the name "r1".
- Can't happen in MIPS 5 stage pipeline because:
  - All instructions take 5 stages, and
  - Reads are always in stage 2, and
  - Writes are always in stage 5

WAW

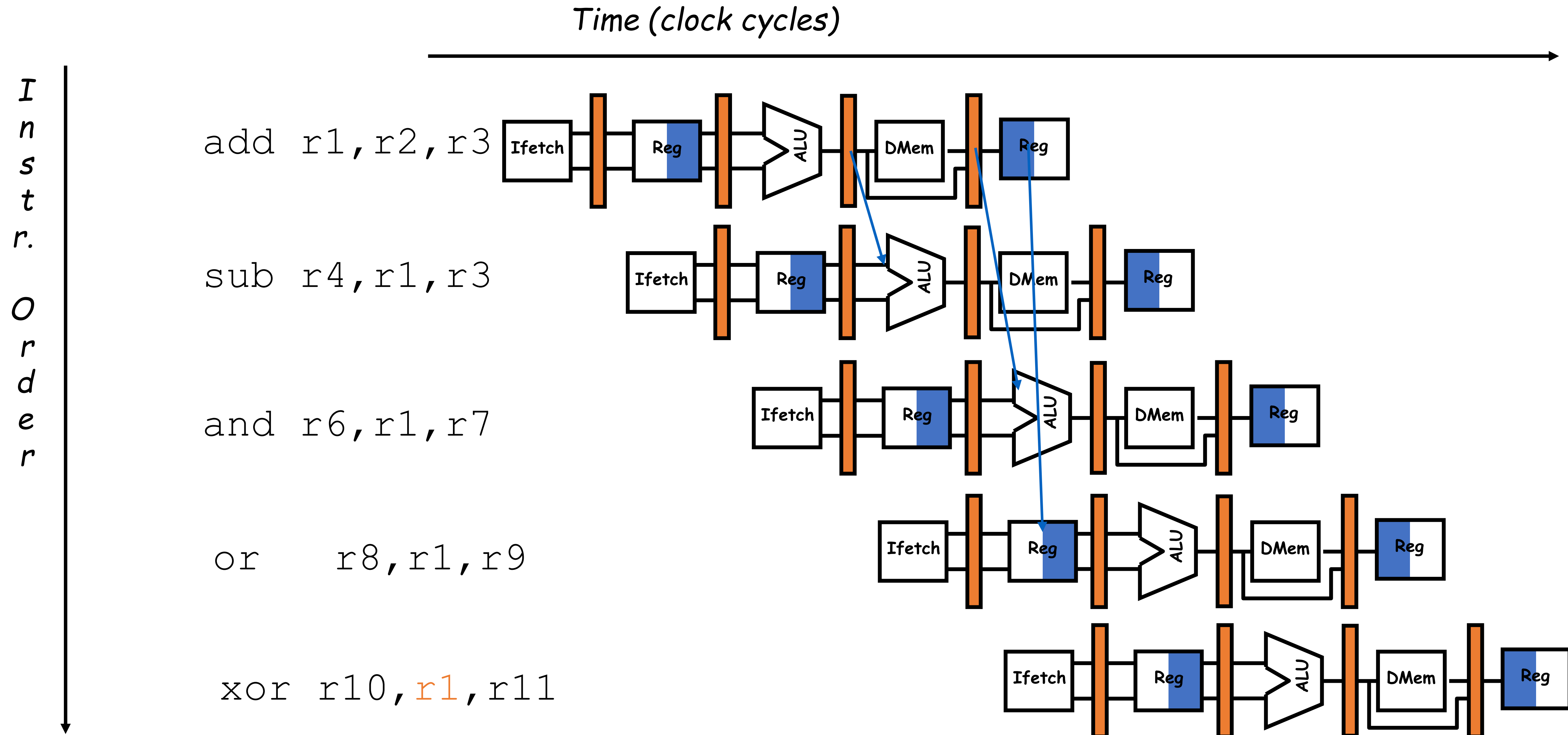
- Write After Write (WAW)
  Instr_J writes operand *before* Instr_I writes it.



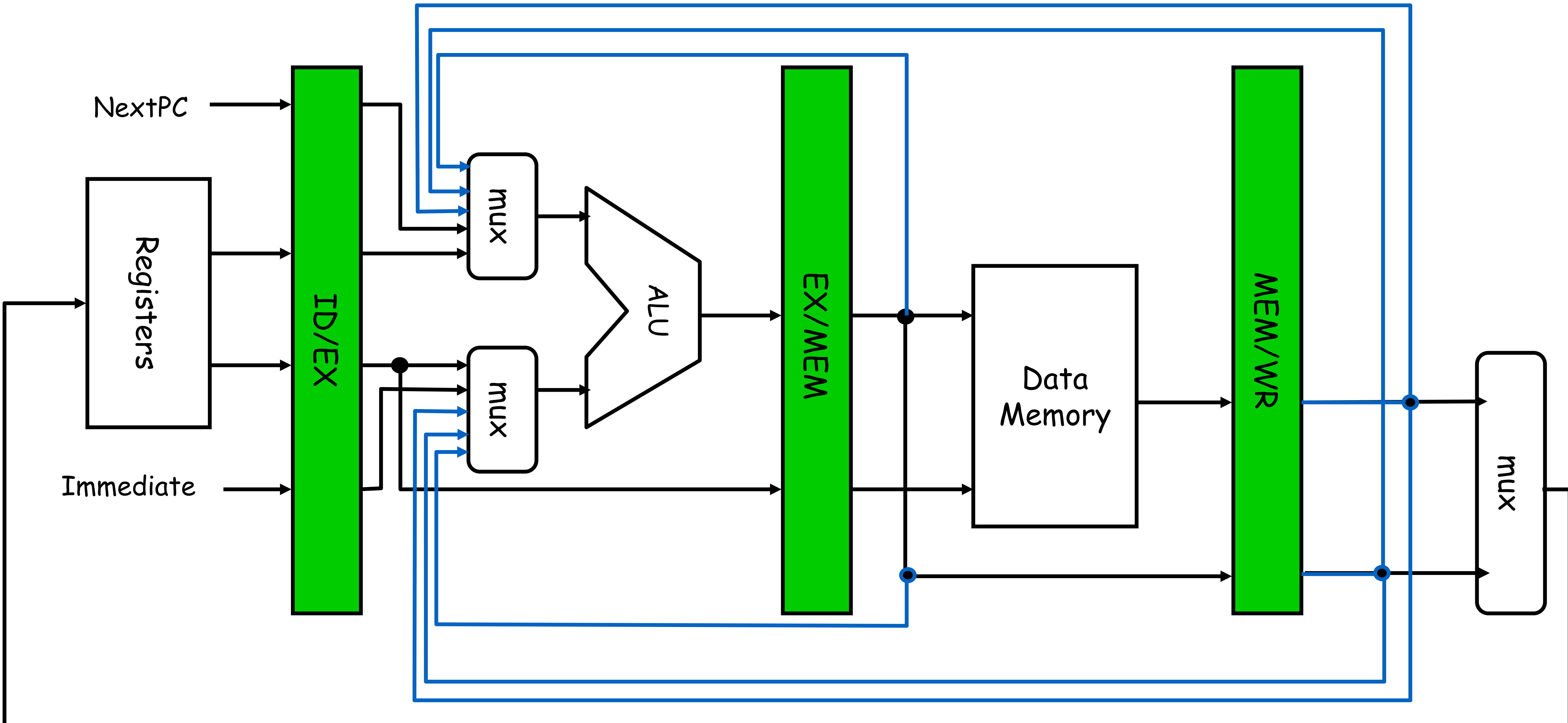
I: sub r1,r4,r3
J: add r1,r2,r3
K: mul r6,r1,r7

- Called an "output dependence" by compiler writers
  This also results from the reuse of name "r1".
- Can't happen in MIPS 5 stage pipeline because:
  - All instructions take 5 stages, and
  - Writes are always in stage 5
- Will see WAR and WAW in more complicated pipes

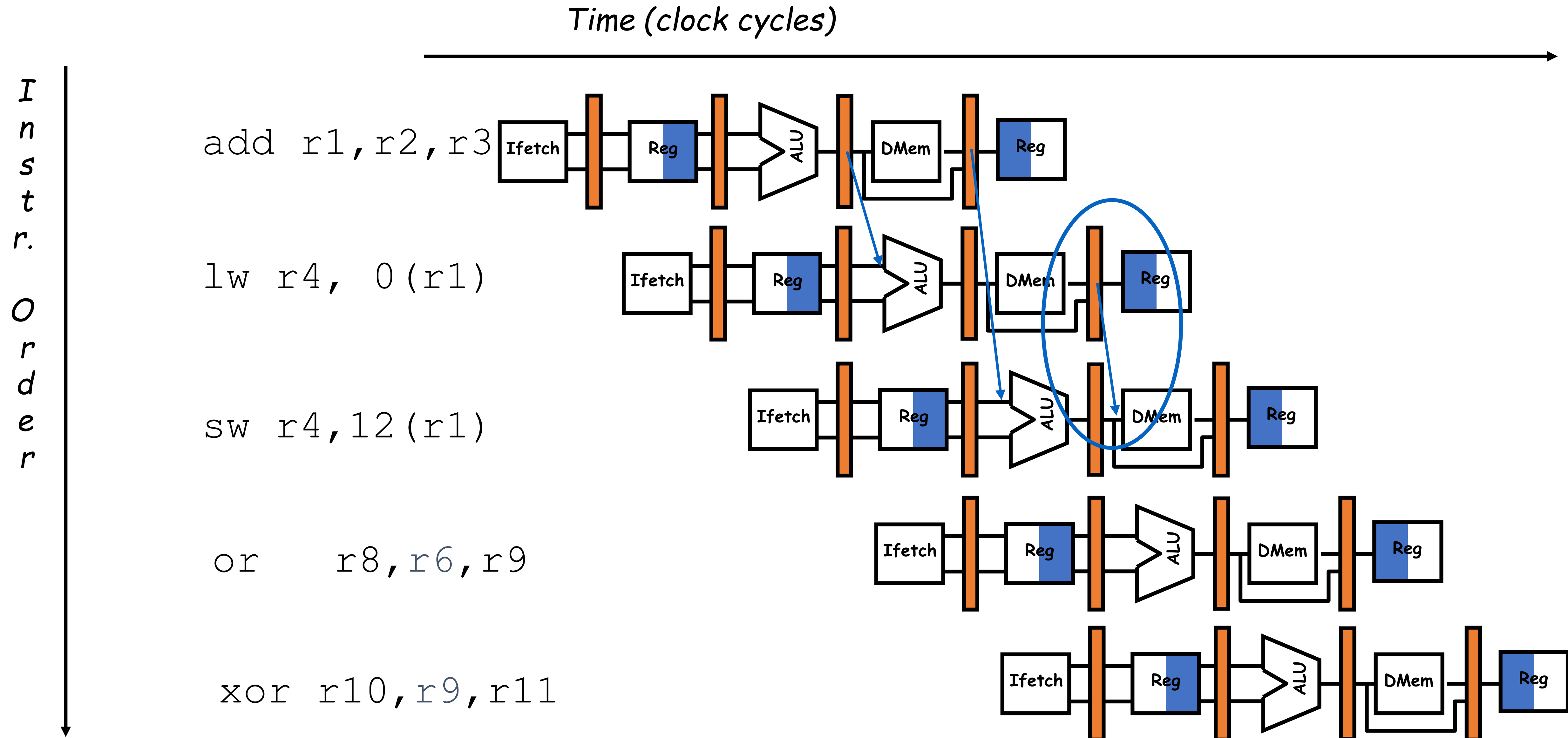
Data Forwarding



Hardware Change



Forwarding to Avoid LW-SW ?



Even With Forwarding ?

Time (clock cycles)

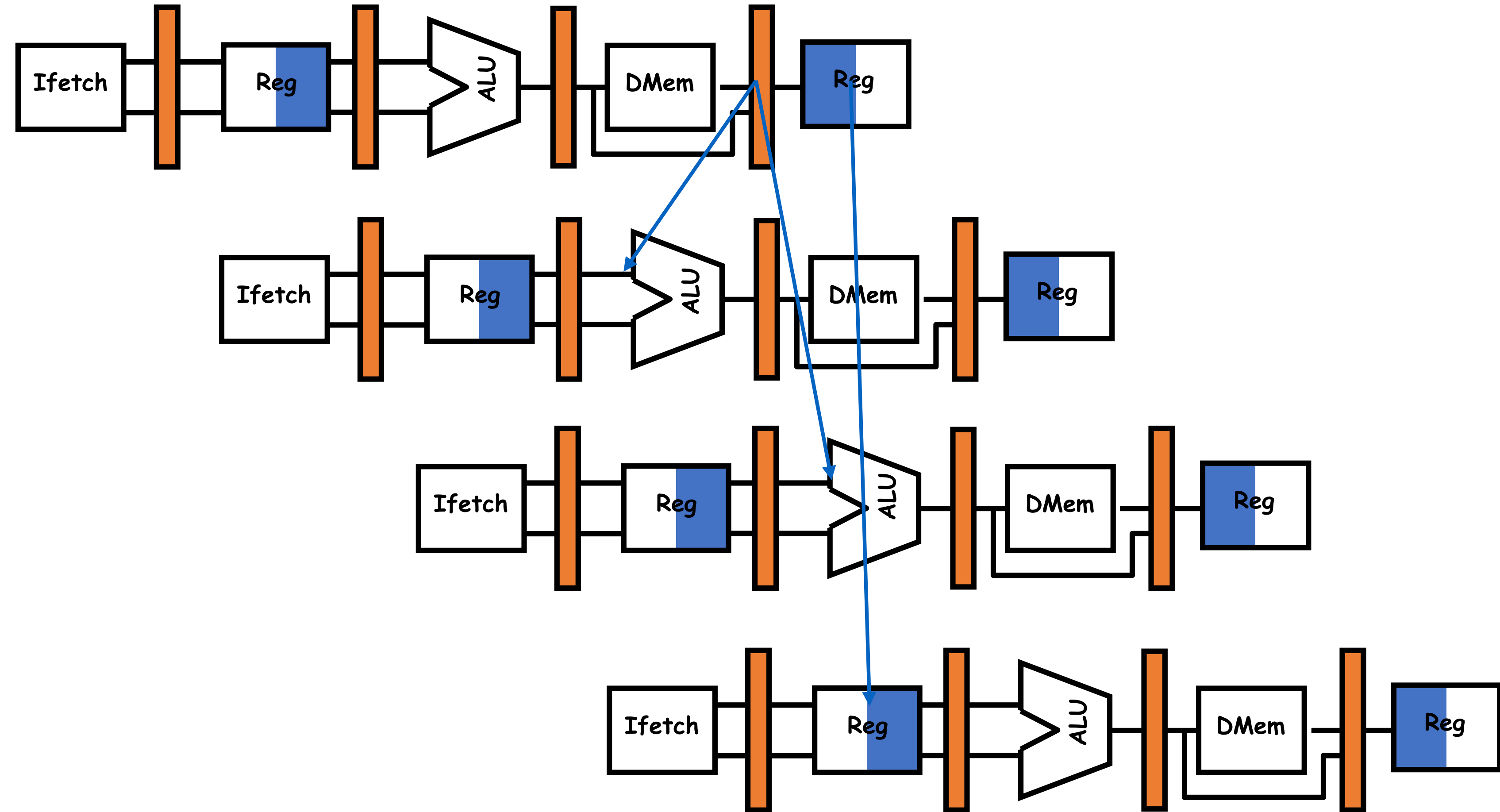
Instr. Order

```
lw r1, 0(r2)

sub r4,r1,r6

and r6,r1,r7

or   r8,r1,r9
```



Control Hazard on Branches with 3-stage Stall

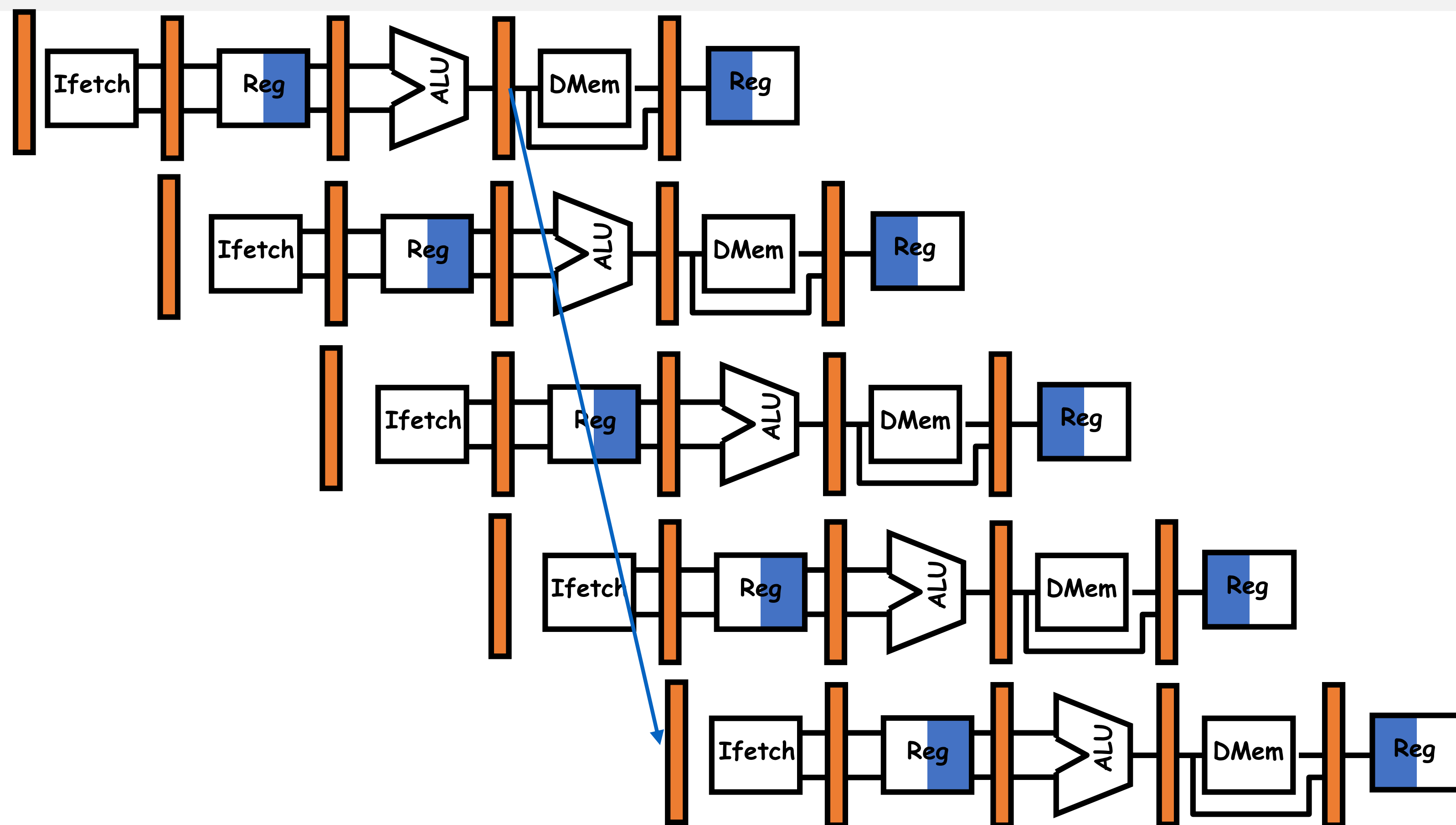
```
10: beq r1,r3,36

14: and r2,r3,r5

18: or  r6,r1,r7

22: add r8,r1,r9

36: xor r10,r1,r11
```



What do you do with the 3 instructions in between?
How do you do it?
Where is the "commit"?

Branches

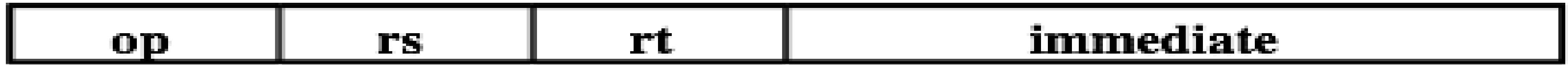
Conditional

- the target address is close to the current PC location
- branch distance from the incremented PC value fits into the immediate field
- for example: loops, if statements

Unconditional (jumps)

- transfers of control
- the target address is far away from the current PC location
- for example: subroutine calls

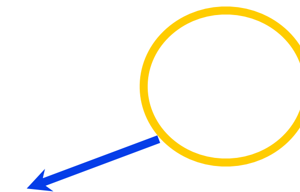
Branches



Syntax: BEQ $1, $2, 12

Action: If ($1 != $2), PC = PC + 4

Action: If ($1 == $2), PC = PC + 4 + 48



Immediate field codes # words, not # bytes.
Why is this encoding a good idea?

Increases branch range to 128 KB.

Zero-extend or sign-extend immediate field? Sign-extend.

Why is this extension method a good idea?

Supports forward and backward branches.