Slide su Pipeline e Hazard in datapath,
Parte 1/3, di:

http://www.pitt.edu/~juy9/142/slides/L8-Pipeline.pdf

Instruction execution review

- Executing a MIPS instruction can take up to five steps.

Step	Name	Description
Instruction Fetch	IF	Read an instruction from memory.
Instruction Decode	ID	Read source registers and generate control signals.
Execute	EX	Compute an R-type result or a branch outcome.
Memory	MEM	Read or write the data memory.
Writeback	WB	Store a result in the destination register.

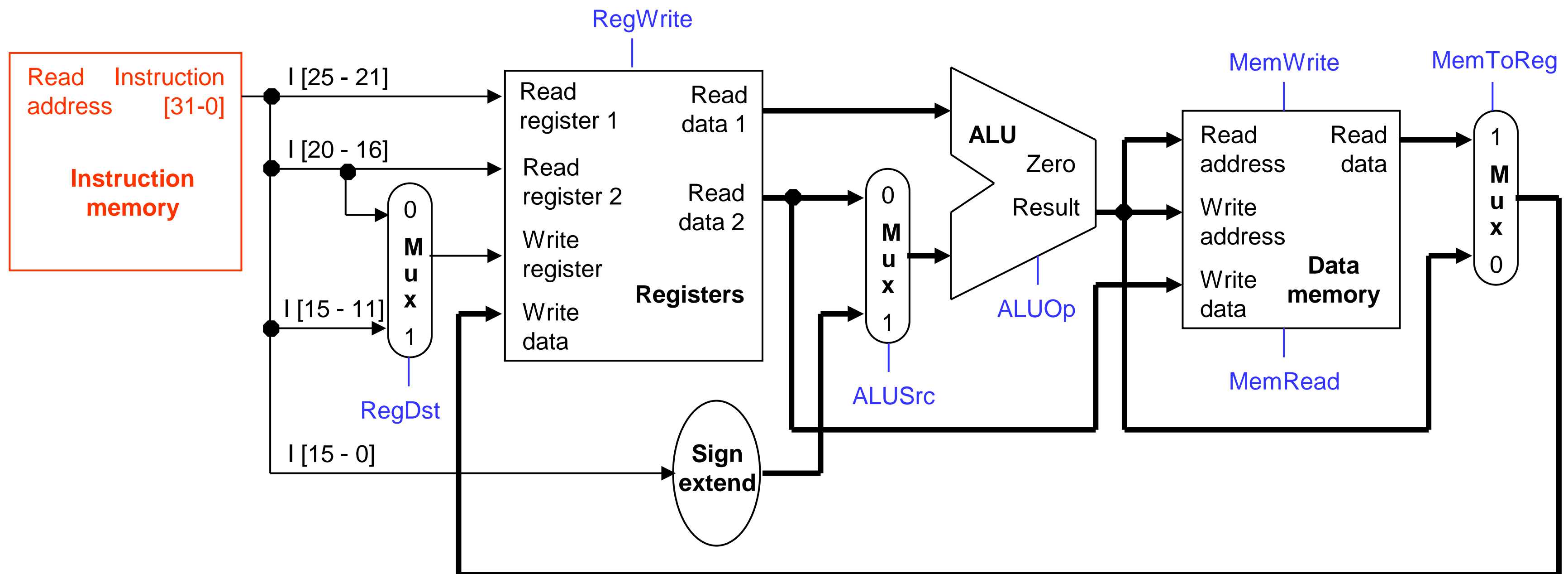
- However, as we saw, not all instructions need all five steps.

Instruction	Steps required				
beq	IF	ID	EX		
R-type	IF	ID	EX		WB
sw	IF	ID	EX	MEM	
lw	IF	ID	EX	MEM	WB

©1998 Morgan Kaufmann Publishers

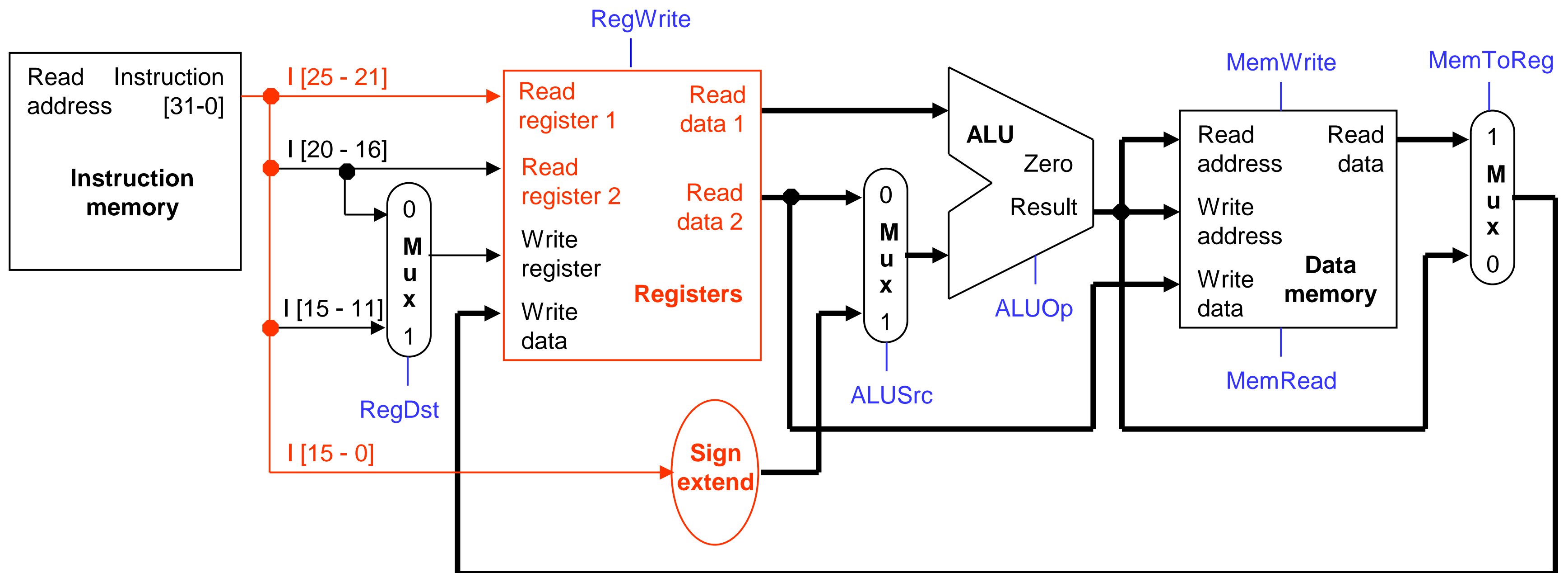
Review: Instruction Fetch (IF)

- Let's quickly review how lw is executed in the single-cycle datapath.
- We'll ignore PC incrementing and branching for now.
- In the Instruction Fetch (IF) step, we read the instruction memory.



Instruction Decode (ID)

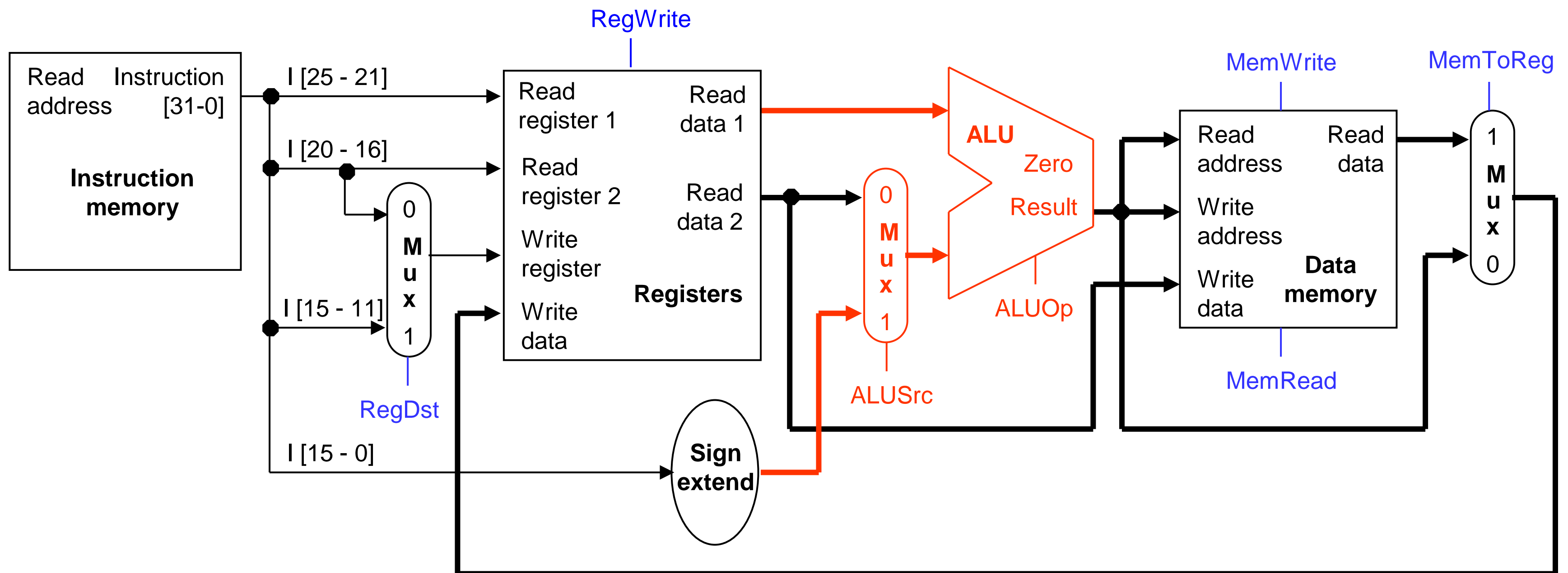
- The Instruction Decode (ID) step reads the source register from the register file.



©1998 Morgan Kaufmann Publishers

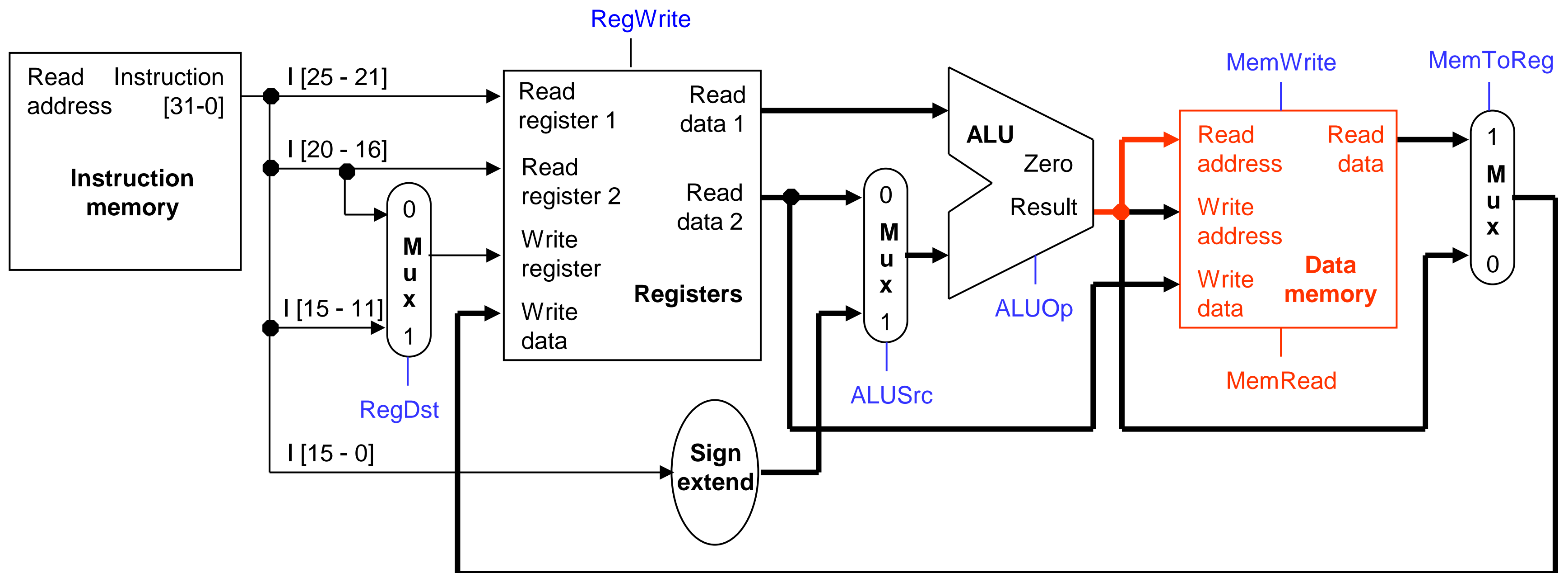
Execute (EX)

- The third step, Execute (EX), computes the effective memory address from the source register and the instruction's constant field.



Memory (MEM)

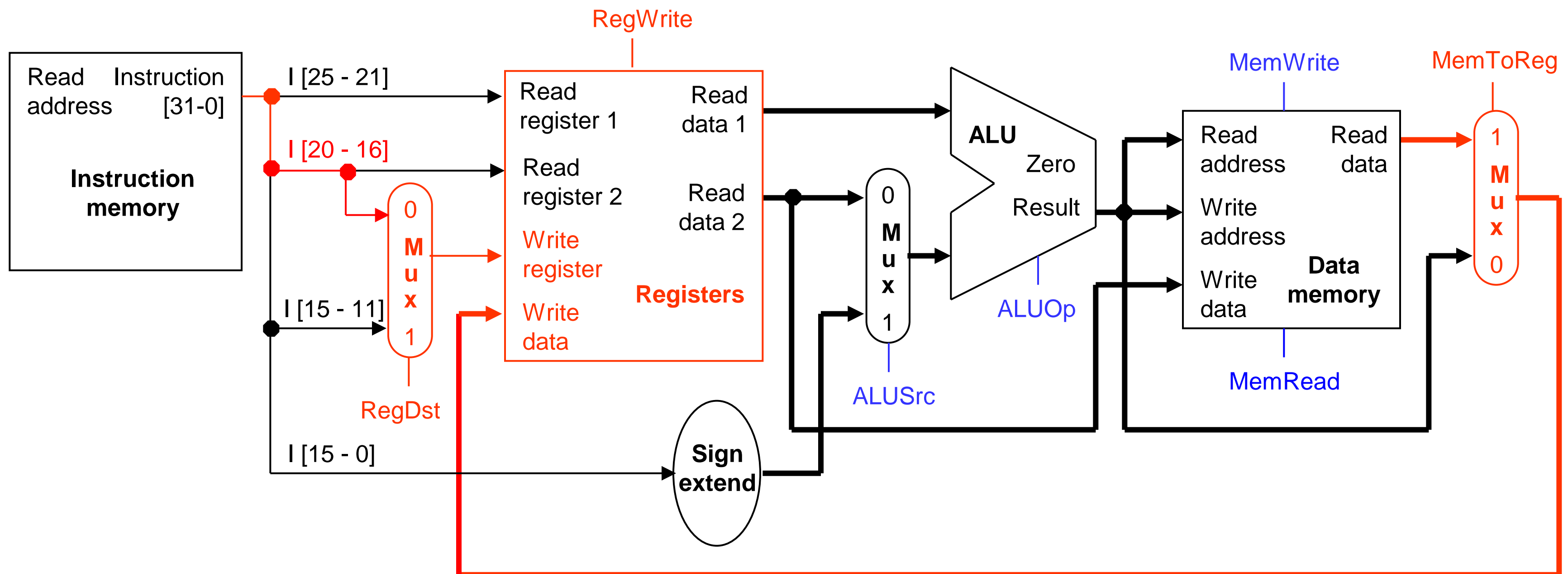
- The Memory (MEM) step involves reading the data memory, from the address computed by the ALU.



©1998 Morgan Kaufmann Publishers

Writeback (WB)

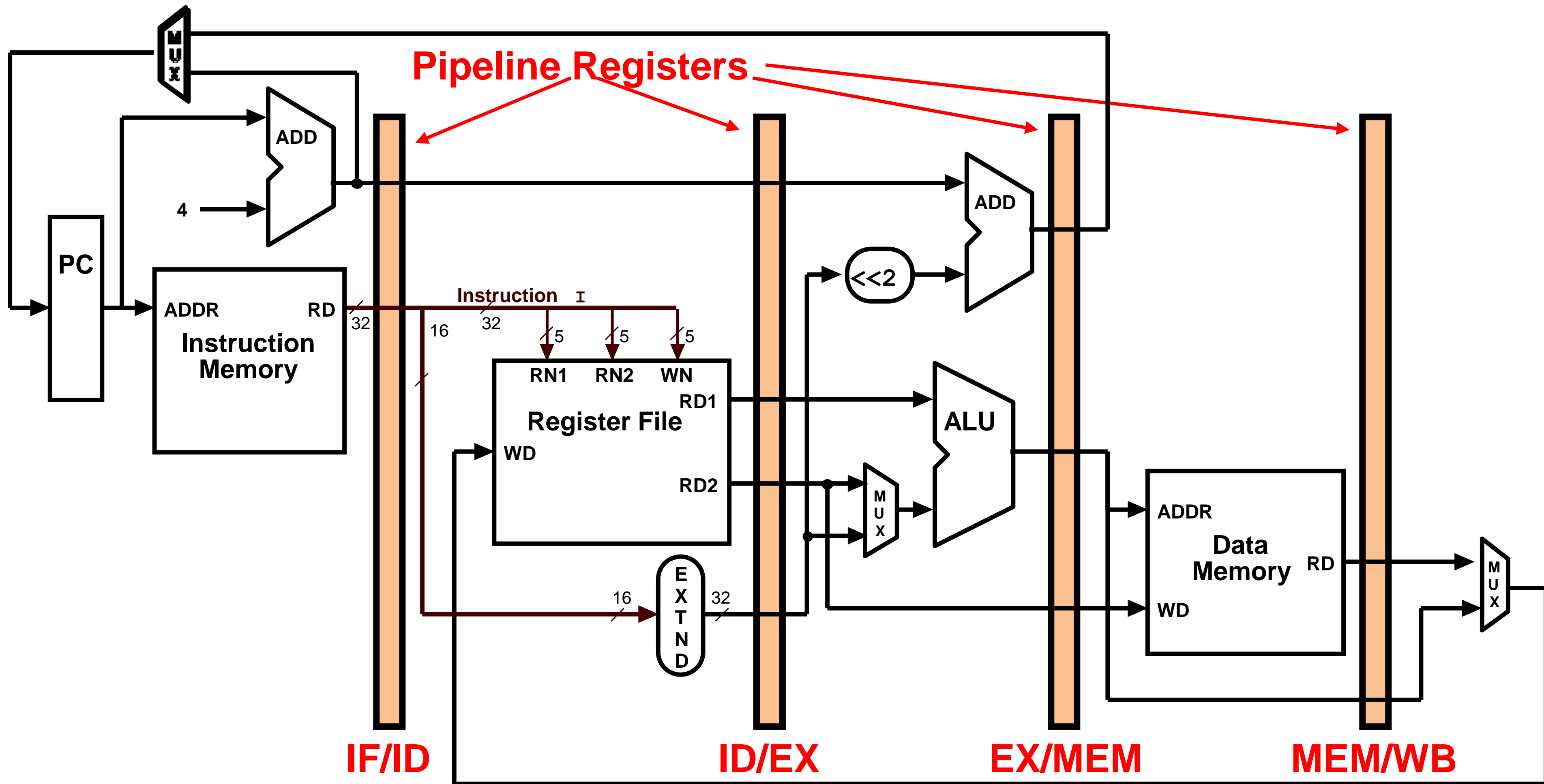
- Finally, in the Writeback (WB) step, the memory value is stored into the destination register.



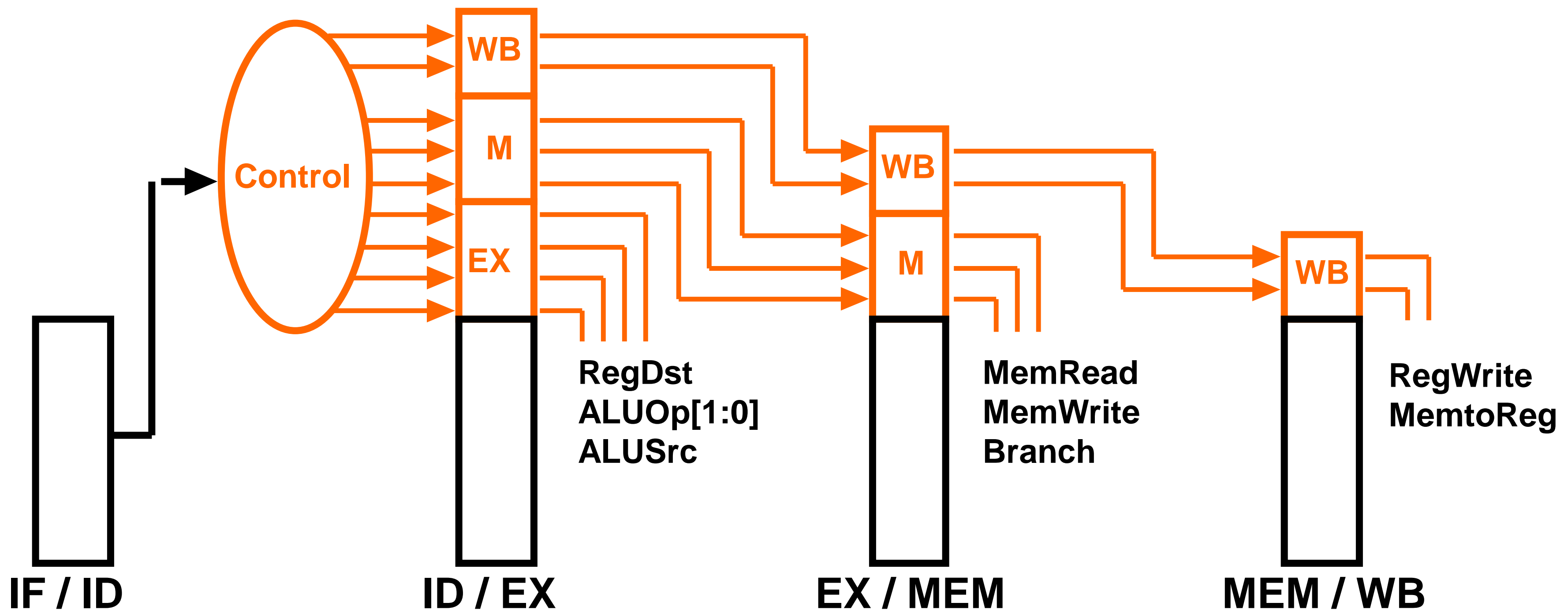
Slide su Pipeline e Hazard in datapath,
Parte 2/3, di:

http://www.eng.auburn.edu/~uguin/teaching/E6200_Spring_2017/lectures/lec5_pipelining.pdf

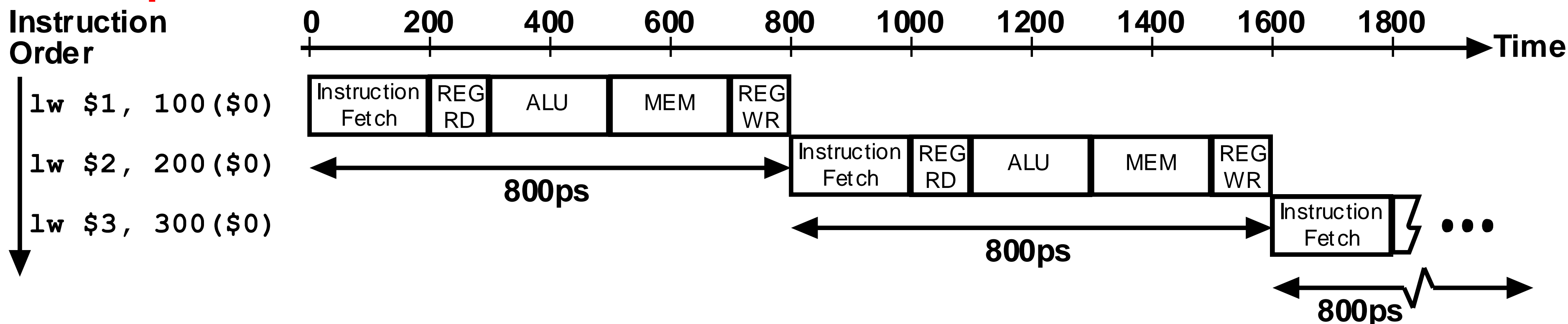
Basic Pipelined Processor

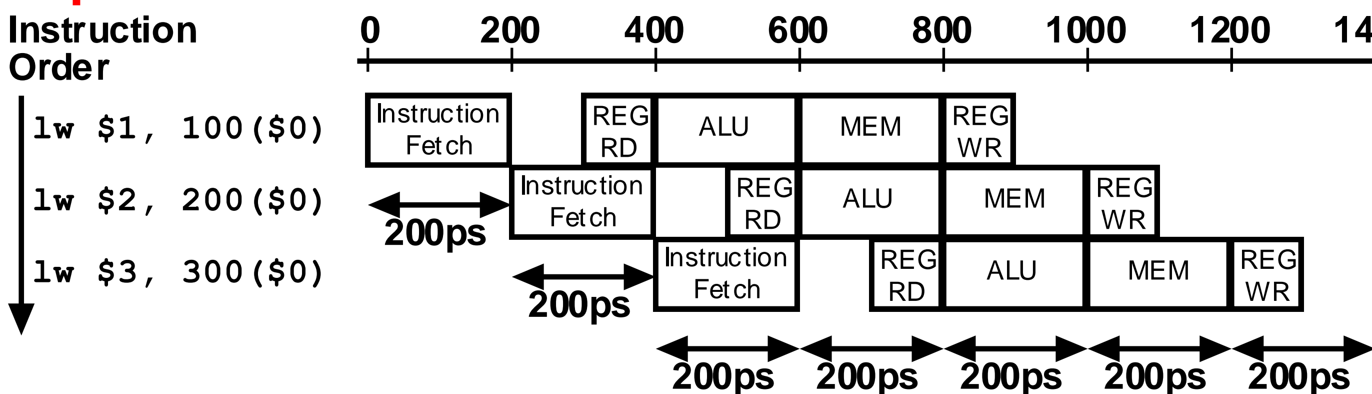


Control for Pipelined Datapath



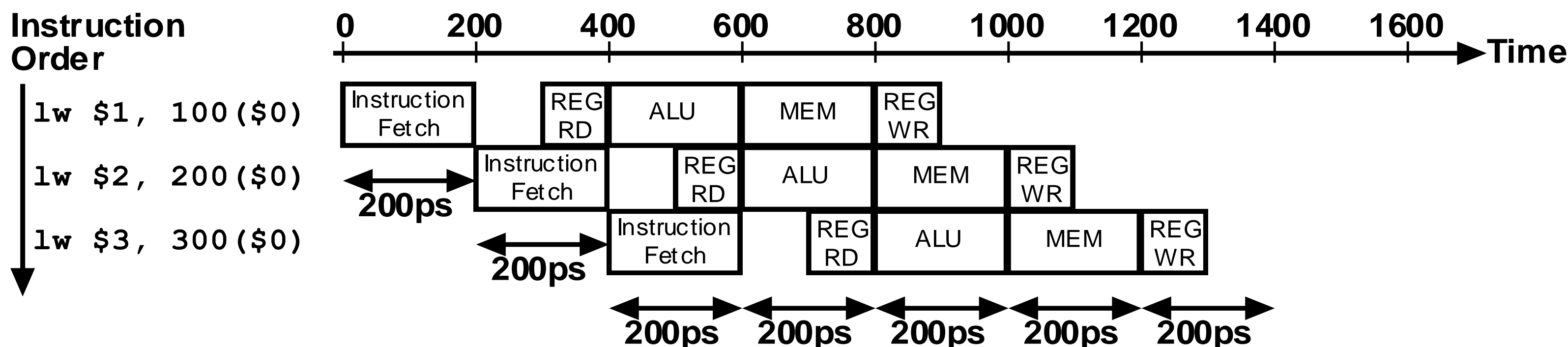
Single-Cycle vs. Pipelined Execution





Note: REGRD is at the end of a stage but REGWR is at the beginning of a stage

Single-Cycle vs. Pipelined Execution (cont.)



- Time taken in pipeline stages is limited by the slowest operation
  - Either ALU operation or Memory access
    - Time taken in ALU stage (i.e. EX) is used as pipeline clock cycle in the following discussion
    - If most memory access is cache access, MEM < ALU
- Assumptions (Fig 4.27 on p.276)
  - Write to the register/memory occurs in the first half of the clock cycle
  - Read from register/memory occurs in the second half of the clock cycle
  - If no such assumption, Cycle 5 of the following example will have issues
    - Executing Multiple Instructions Clock Cycle 5, where the register file is used for 2 instructions at their different stages (ID and WB)
  - How to design such an assumption?

Comments about Pipelining

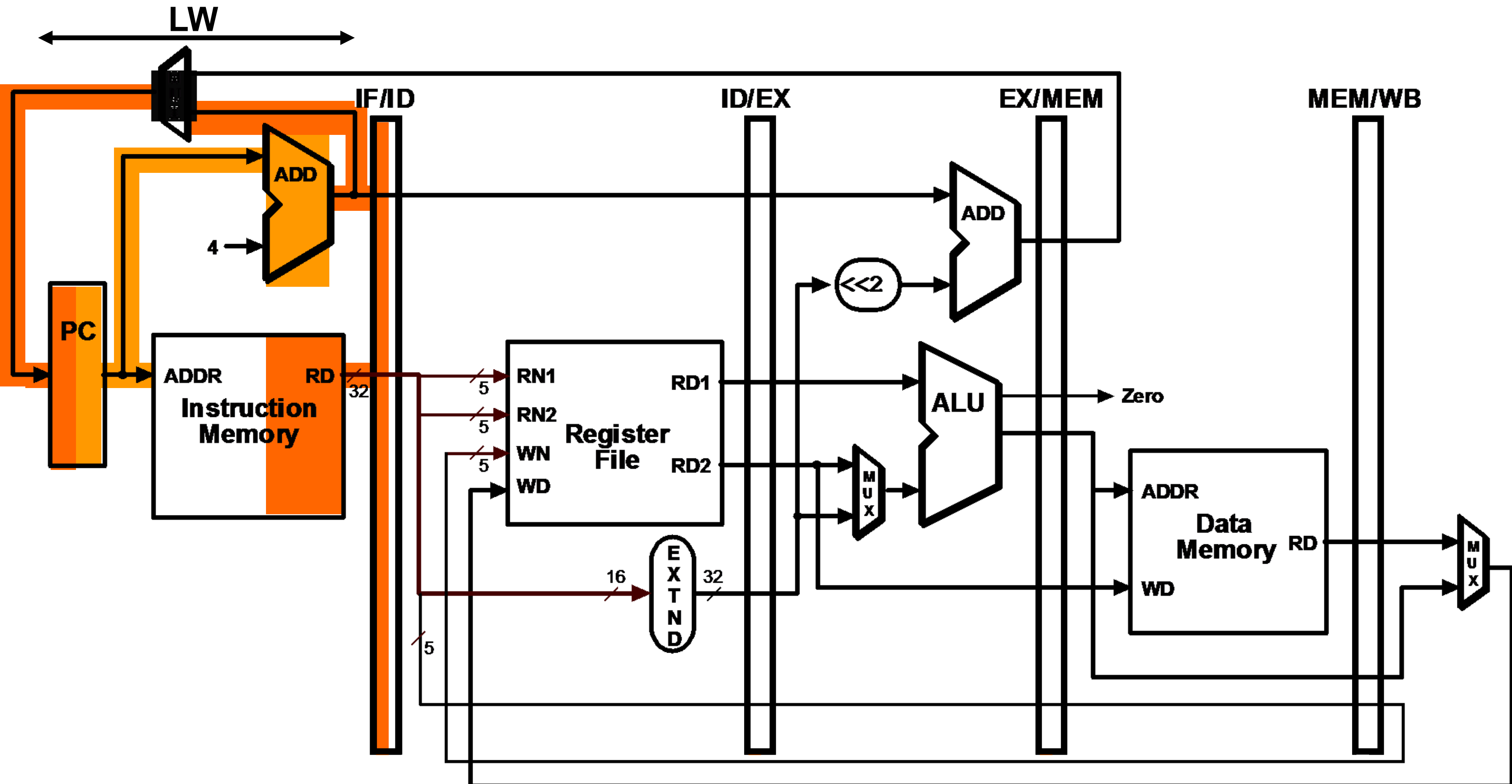
- The good news
  - Multiple instructions are being processed at the same time
  - This works because stages are isolated by registers
  - Best case speedup of #Stages
- The bad news
  - Instructions interfere with each other - Hazards
    - Different instructions may need the same piece of hardware (e.g., memory) in same clock cycle --- Structure Hazard
    - Not sure which is the next instruction for the next instruction fetch (IF) until EX of the branch instruction --- Control Hazard
    - Instruction may require a result produced by an earlier instruction that is not yet complete --- Data Hazard
  - Worst case: Must suspend execution - Stall

Example - Executing Multiple Instructions

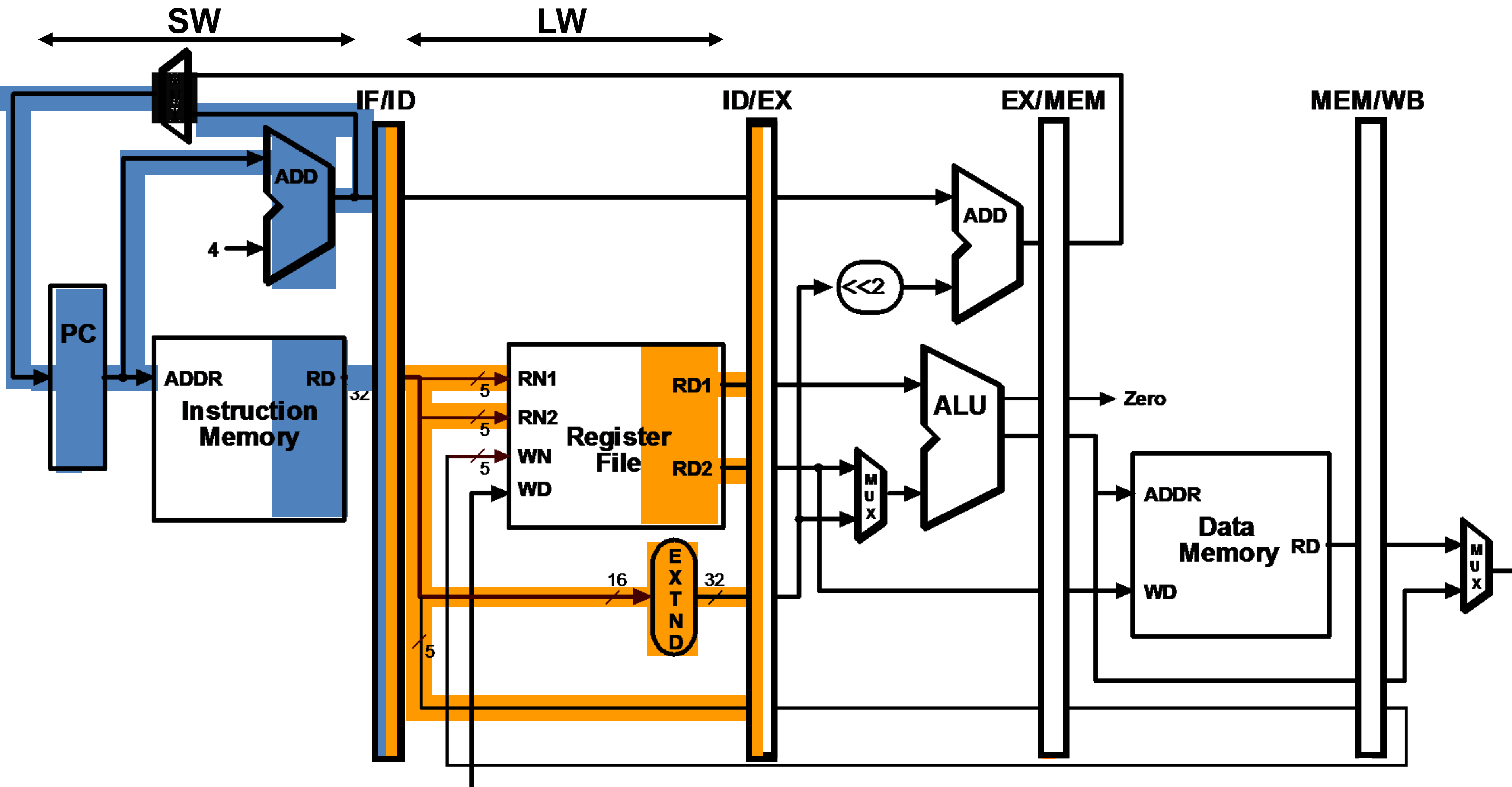
- Consider the following instruction sequence

```
lw   $r0, 10($r1)
sw   $r3, 20($r4)
add  $r5, $r6, $r7
sub  $r8, $r9, $r10
```

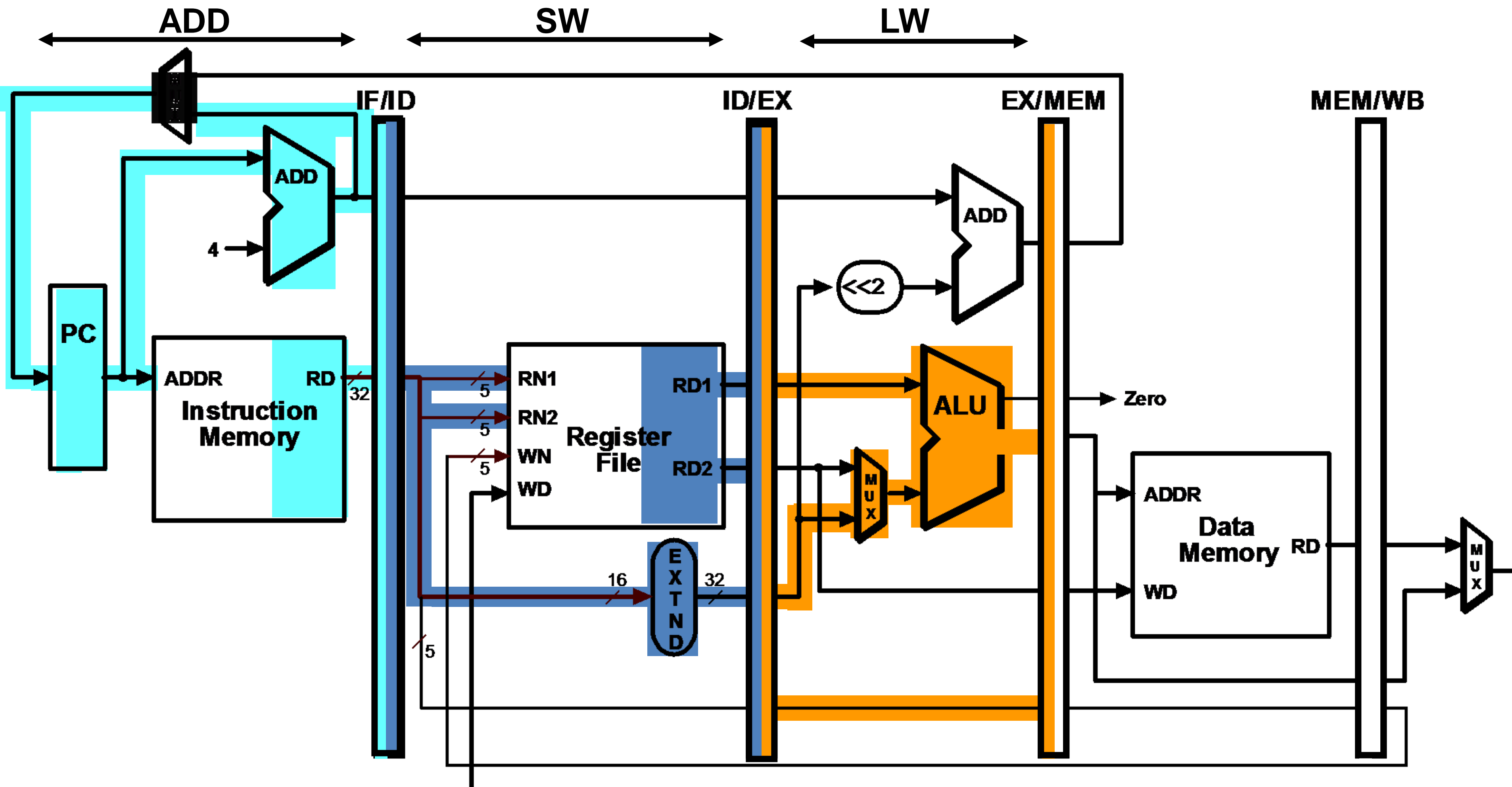
Executing Multiple Instructions Clock Cycle 1



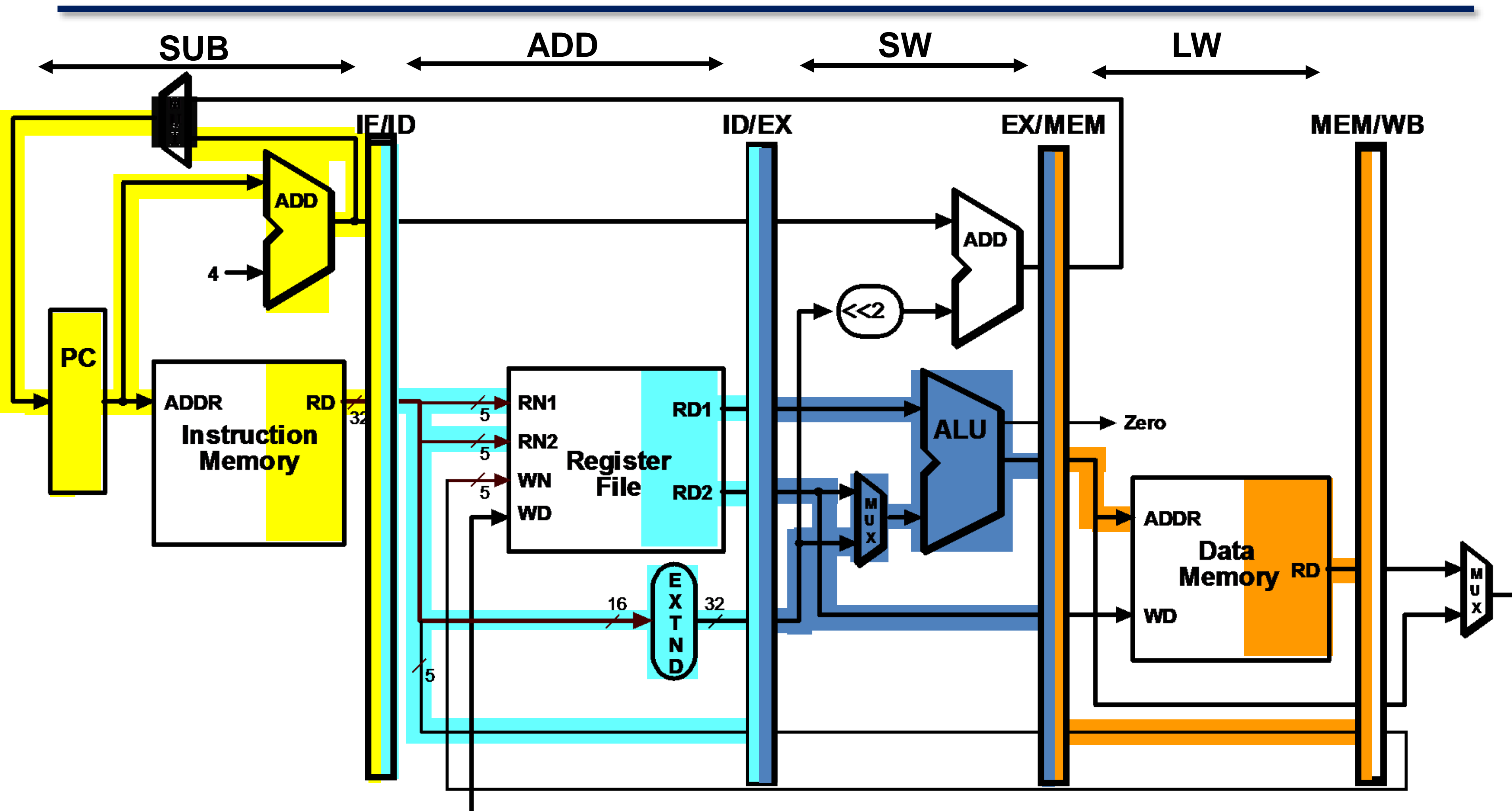
Executing Multiple Instructions Clock Cycle 2



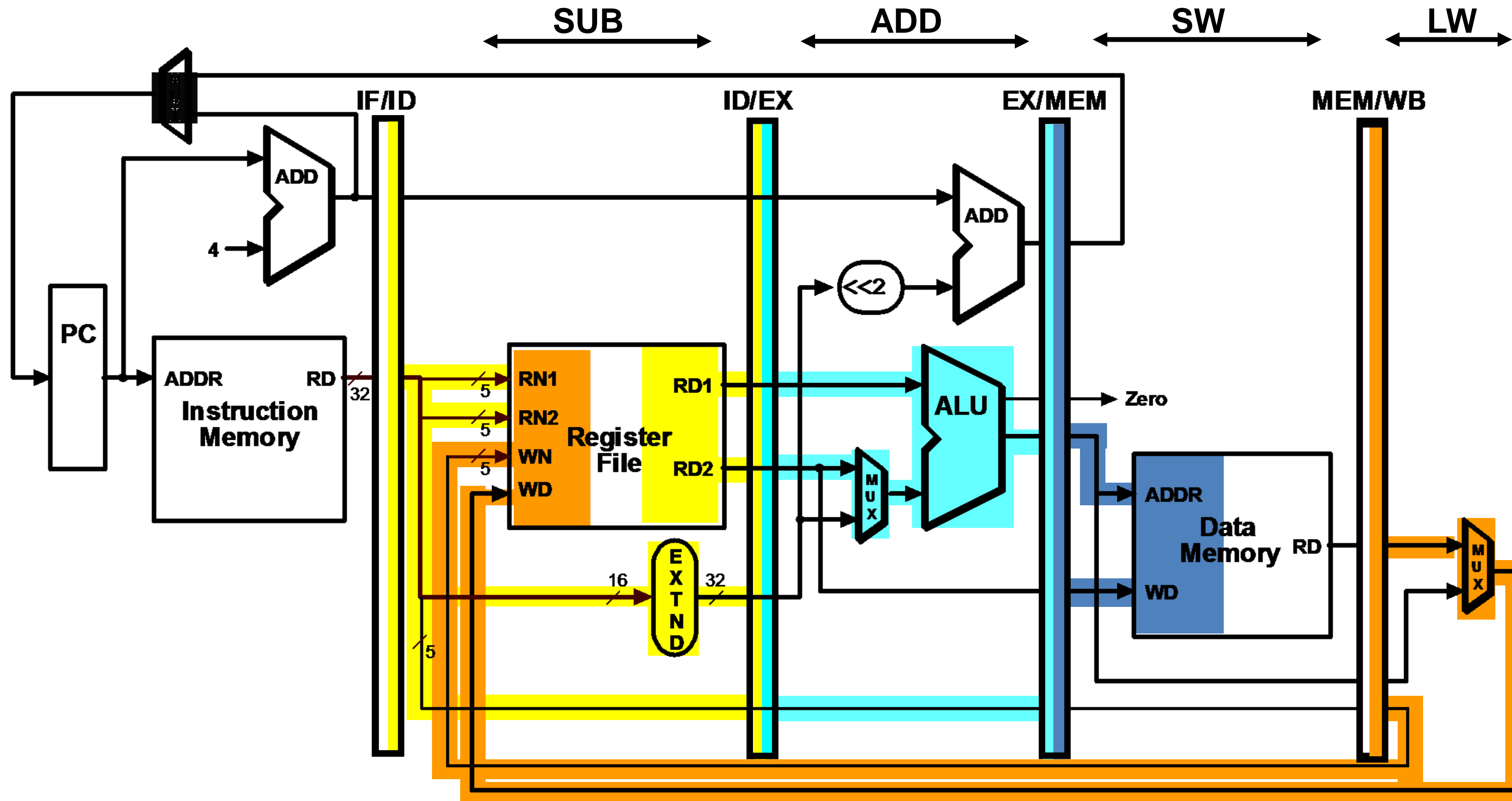
Executing Multiple Instructions Clock Cycle 3



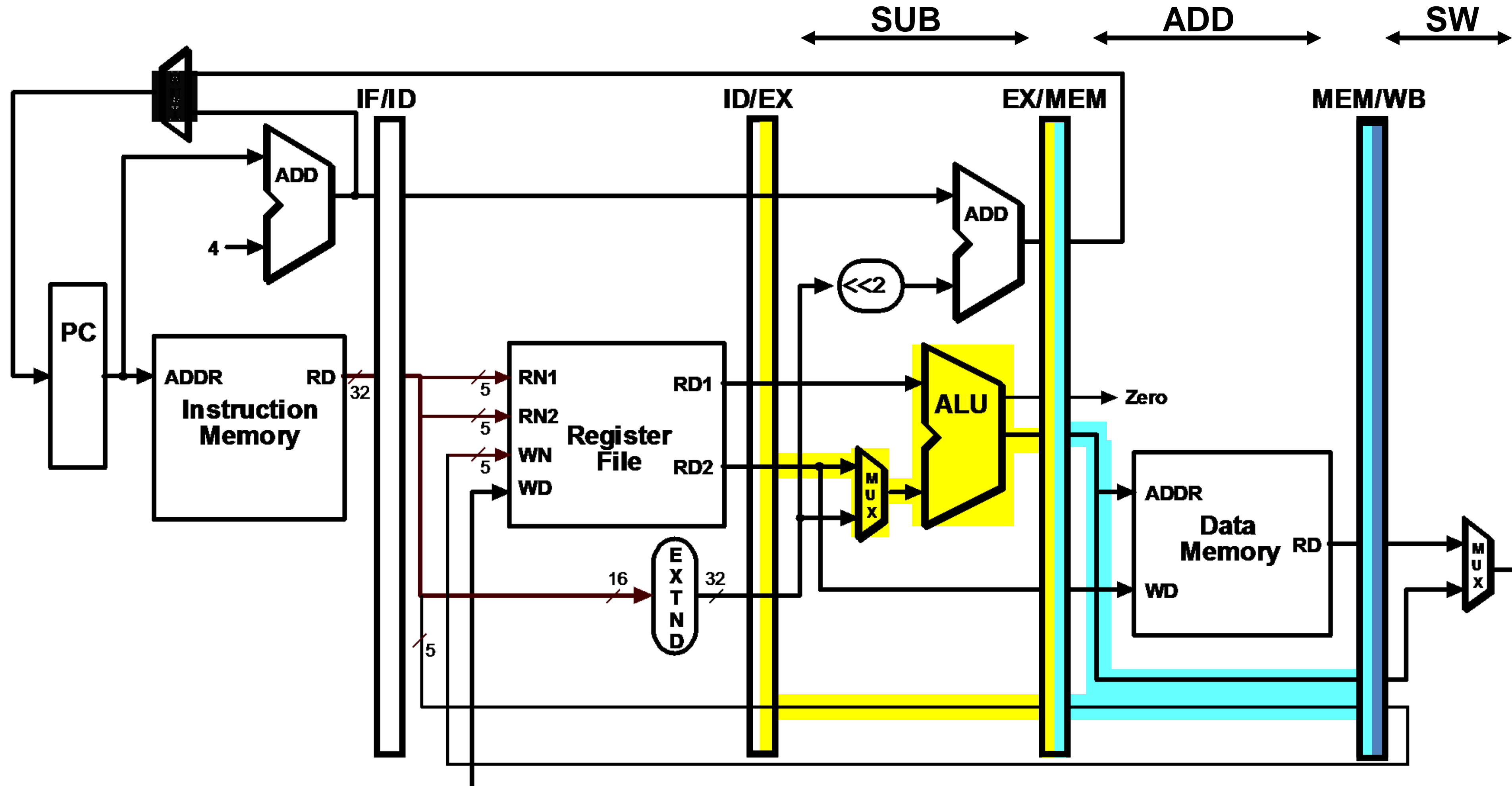
Executing Multiple Instructions Clock Cycle 4



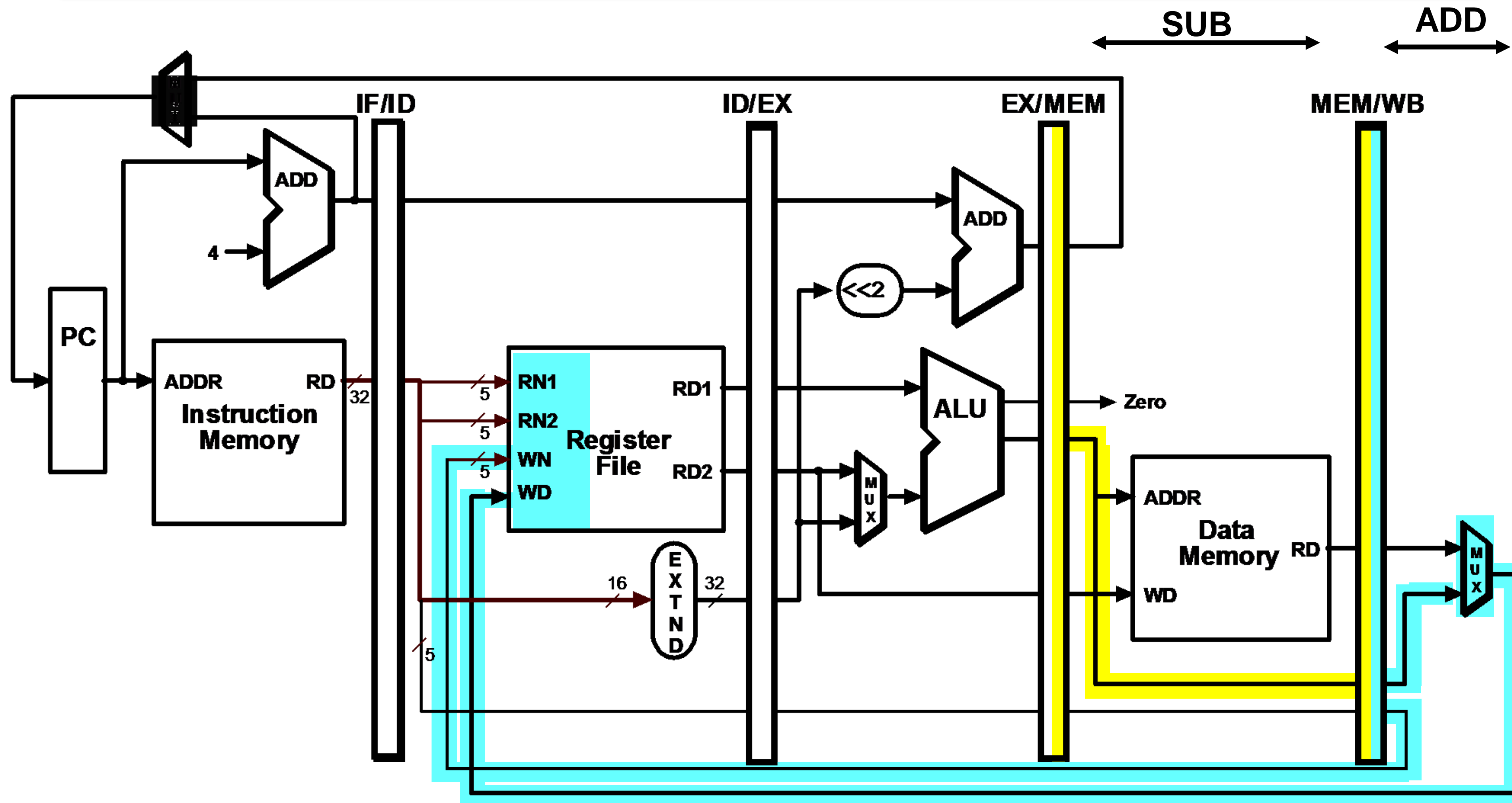
Executing Multiple Instructions Clock Cycle 5



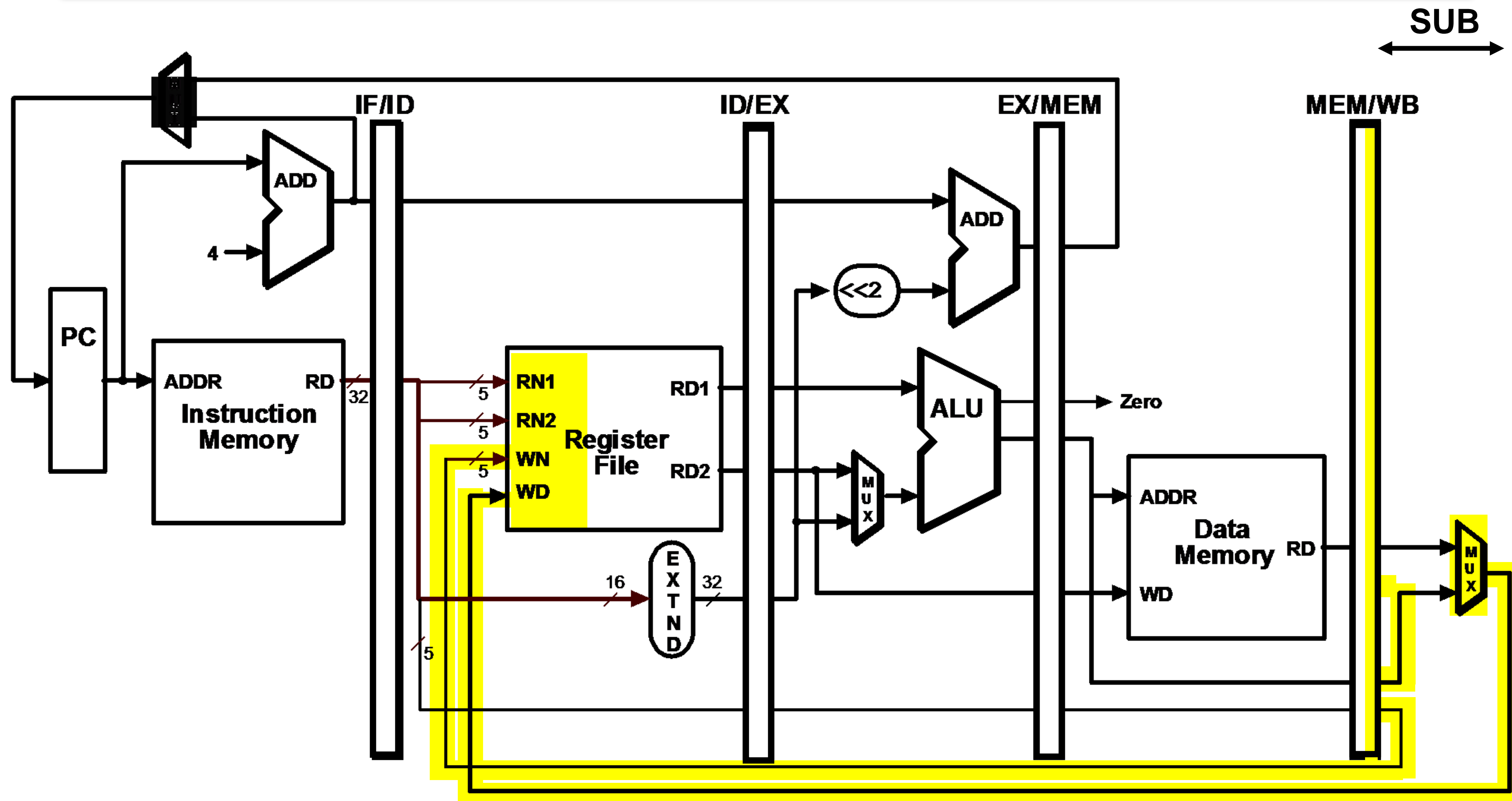
Executing Multiple Instructions Clock Cycle 6



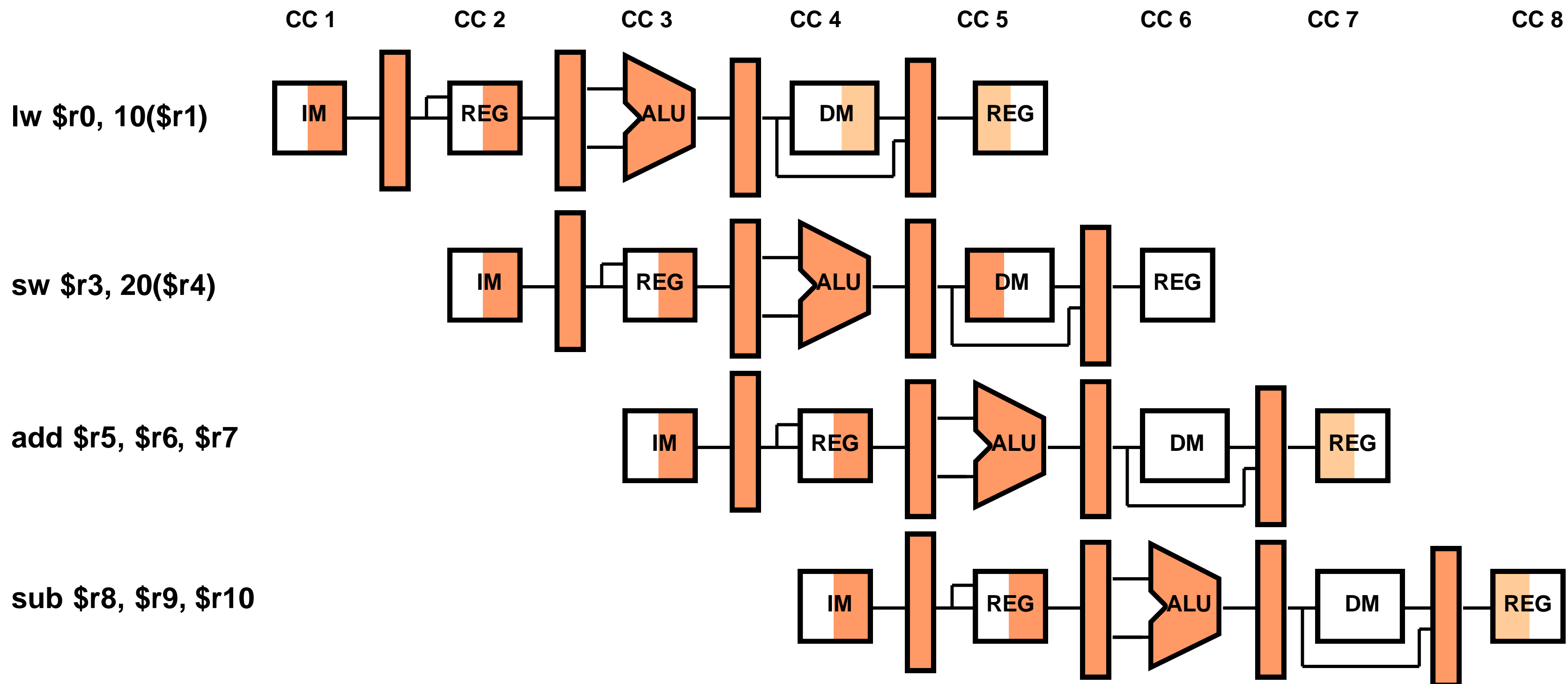
Executing Multiple Instructions Clock Cycle 7



Executing Multiple Instructions Clock Cycle 8



Compact View



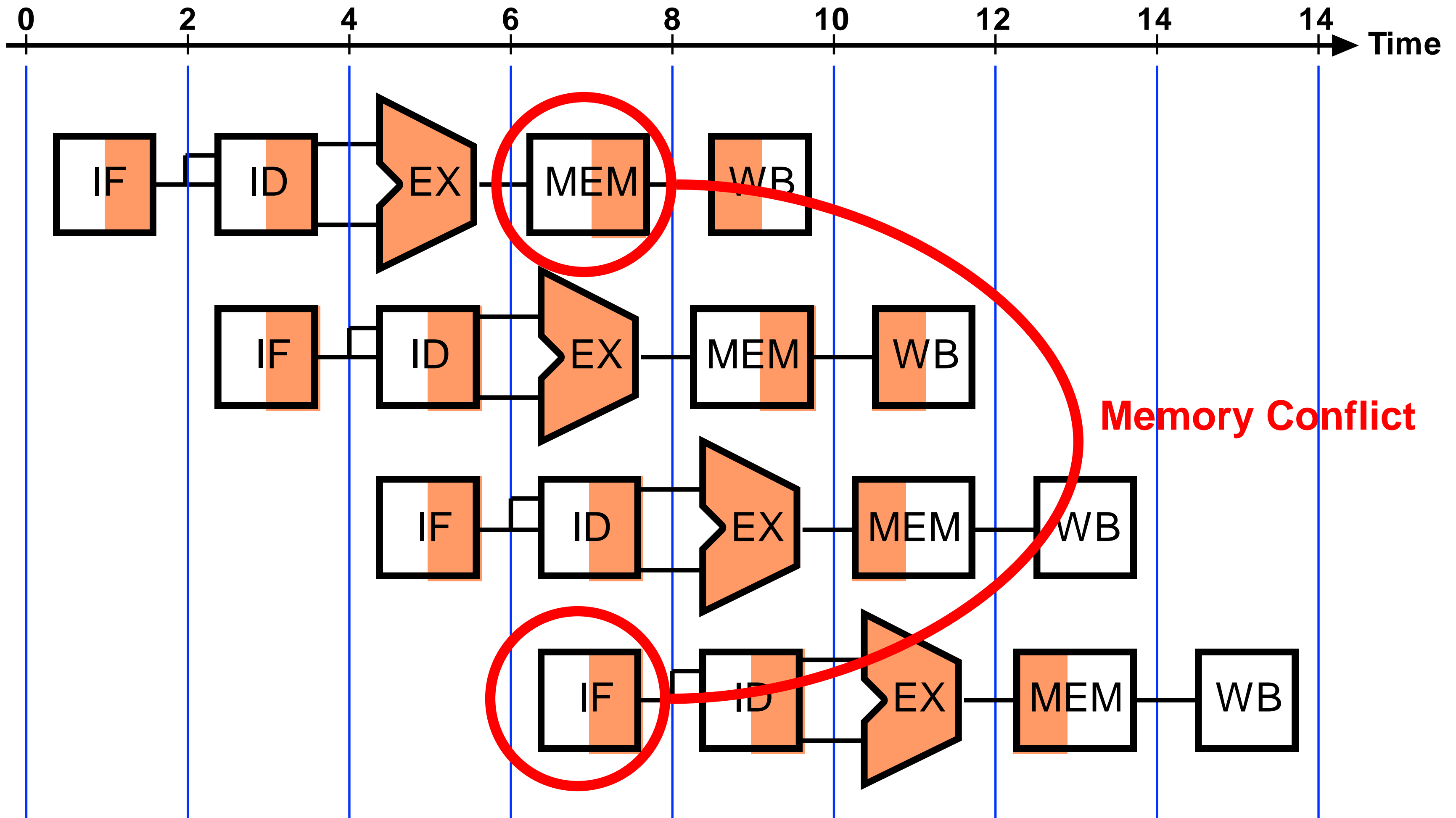
Pipeline Hazards

- Where one instruction cannot immediately follow another
- Types of hazards
  - Structural hazards - attempt to use same resource twice
  - Control hazards - attempt to make decision before condition is evaluated
  - Data hazards - attempt to use data before it is ready
- We can always resolve hazards by waiting
  - i.e. stall

Structural Hazards

- Attempt to use same resource twice at same time
- Example: A Single Memory for both instructions and data
  - Accessed by IF stage
  - Accessed at same time by MEM stage
- Solutions
  - Delay second access by one clock cycle, OR
  - Provide separate memories for instructions and data (IM and DM)
    - This is what MIPS does
    - Recall “Harvard Architecture”
    - Real pipelined processors have separate caches

Structural Hazard - Single Memory



Slide su Pipeline e Hazard in datapath,
Parte 3/3, di:

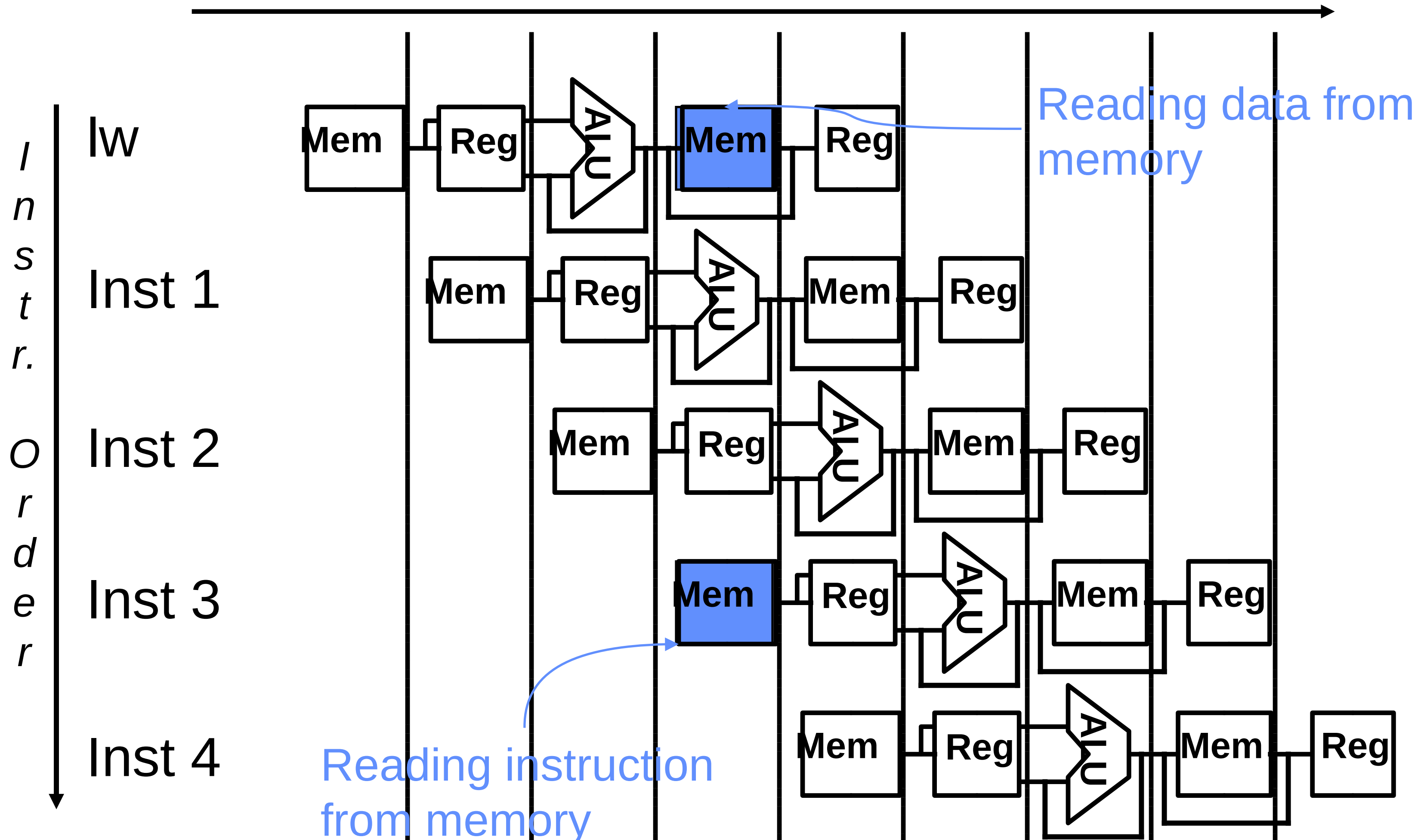
http://person.zju.edu.cn/person/fck_filebrowser.php?cmd=download&id=712274

Can Pipelining Get Us Into Trouble?

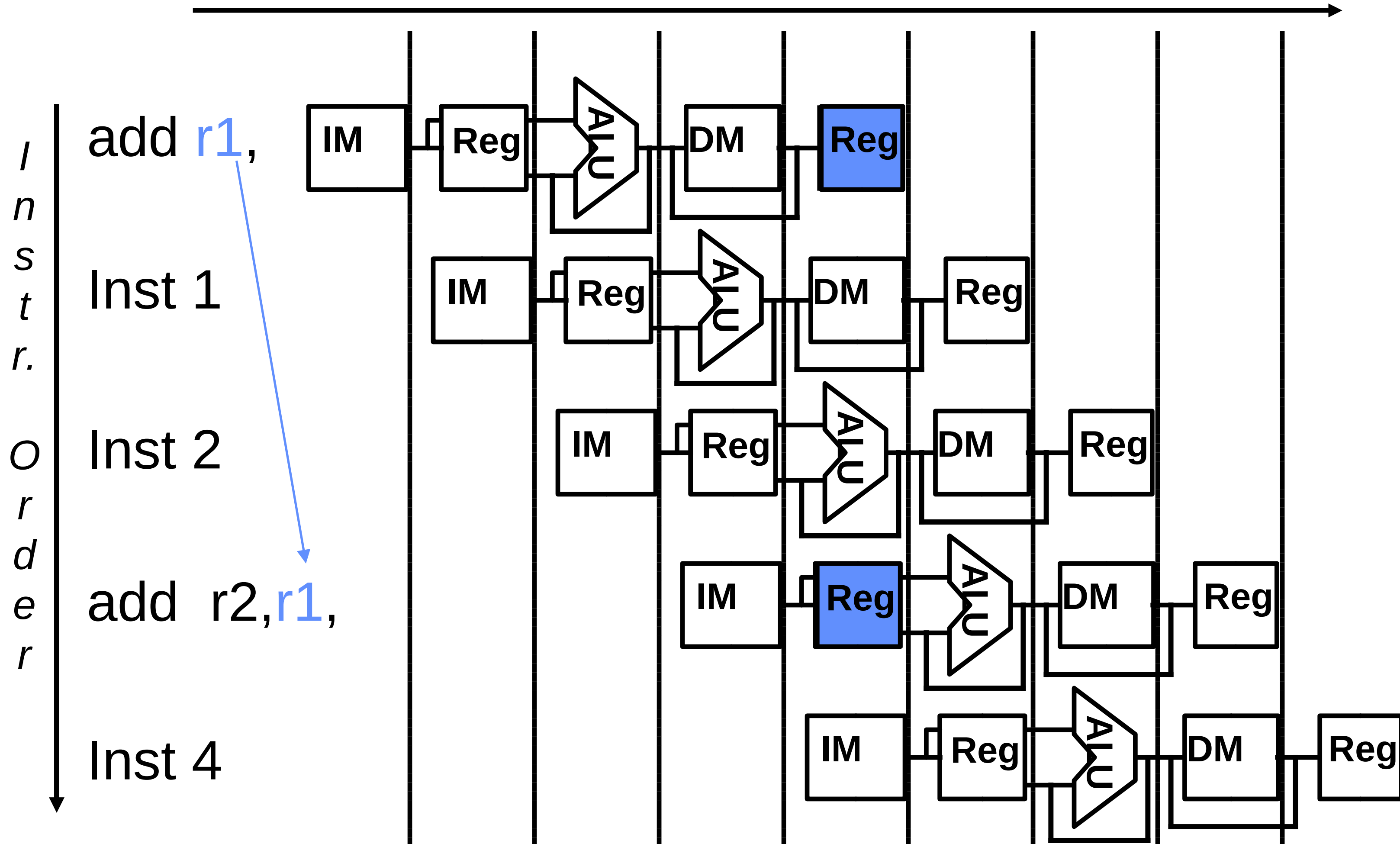
- Yes: Pipeline Hazards
  - **structural hazards: attempt to use the same resource by two different instructions at the same time**
  - **data hazards: attempt to use data before it is ready**
    - instruction source operands are produced by a prior instruction still in the pipeline
    - load instruction followed immediately by an ALU instruction that uses the load operand as a source value
  - **control hazards: attempt to make a decision before condition has been evaluated**
    - branch instructions

- Can always resolve hazards by waiting
  - **pipeline control must detect the hazard**
  - **take action (or delay action) to resolve hazards**

A Single Memory Would Be a Structural Hazard

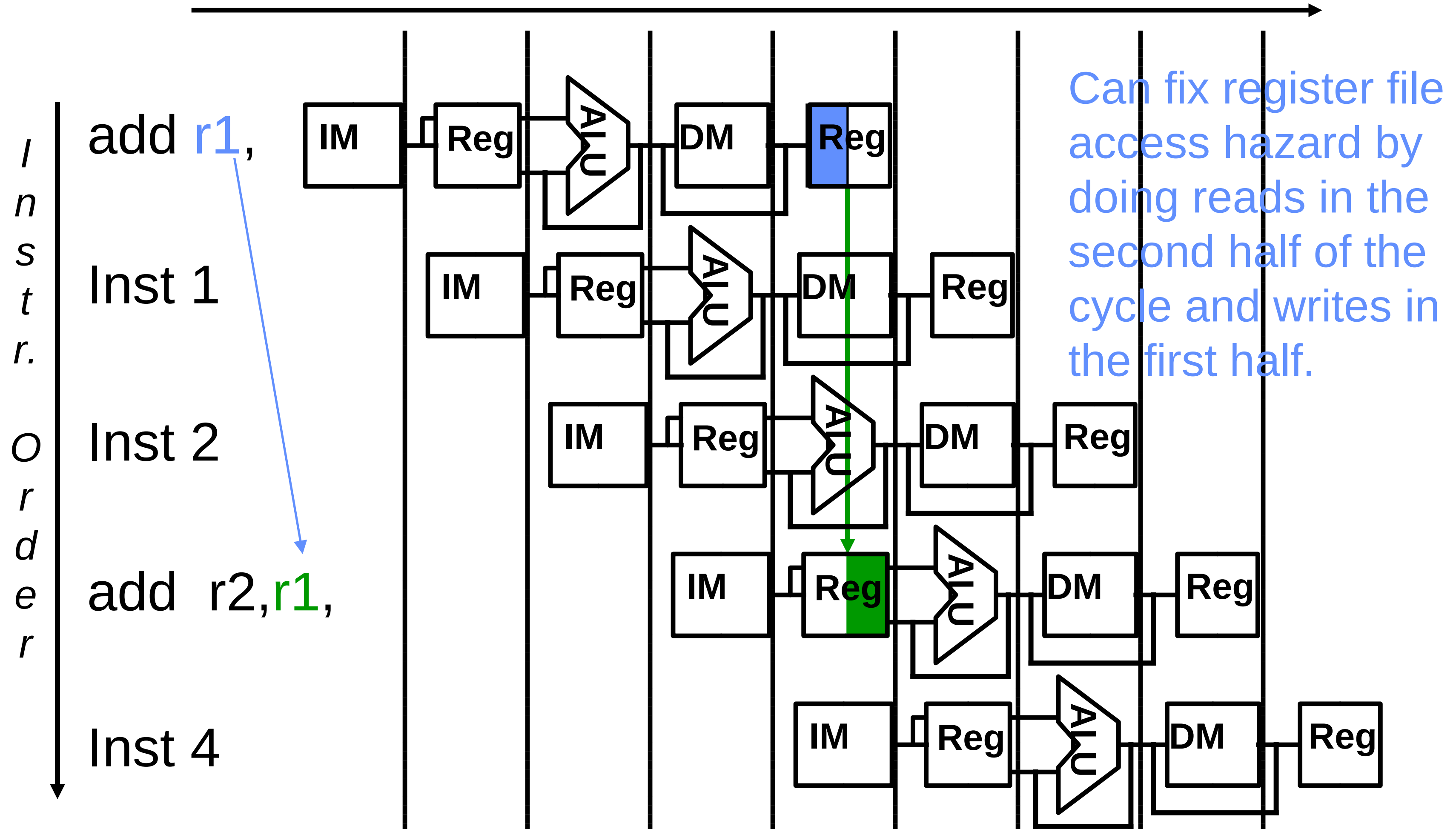


How About Register File Access?



Potential read before write data hazard

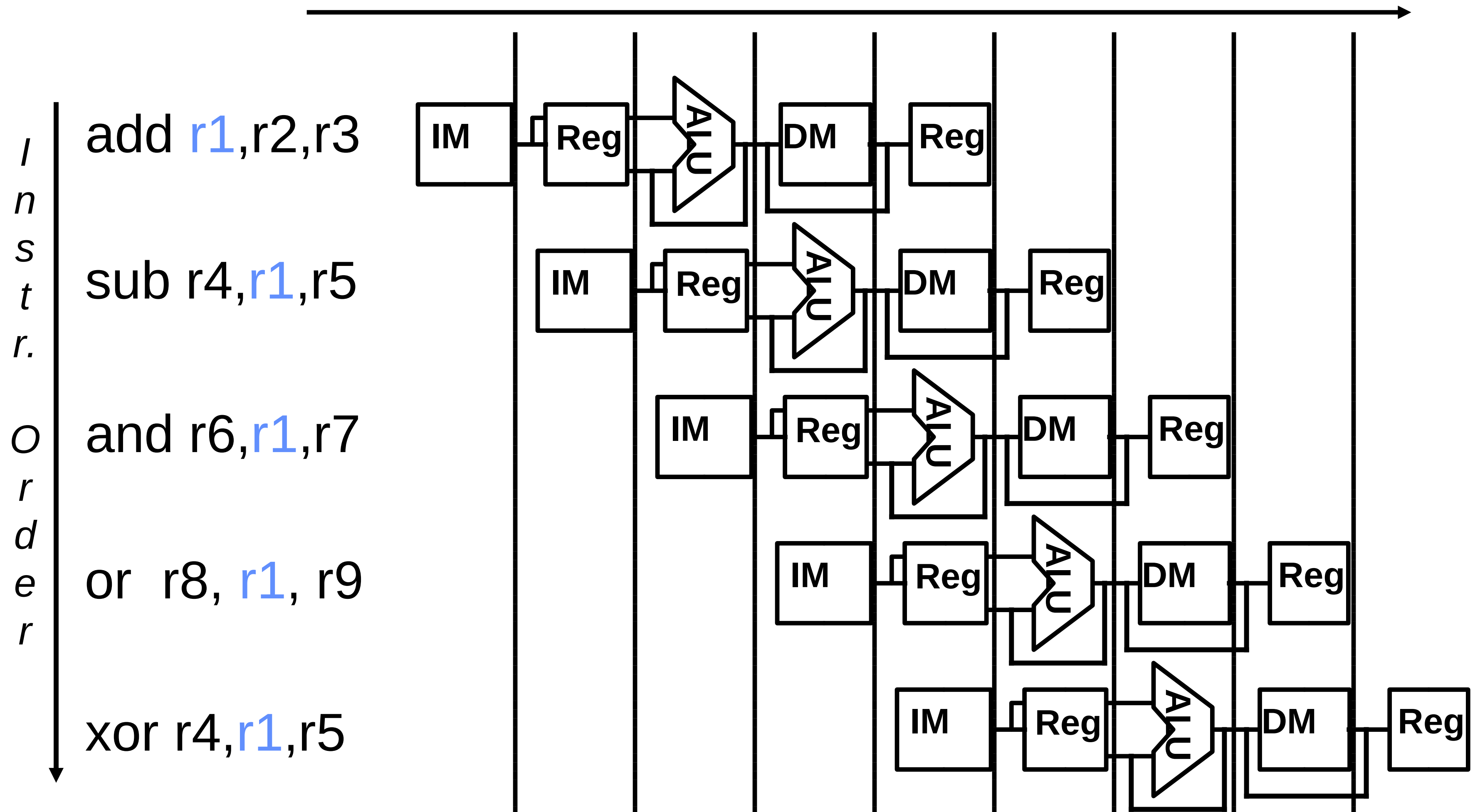
How About Register File Access?



Potential read before write data hazard

Register Usage Can Cause Data Hazards

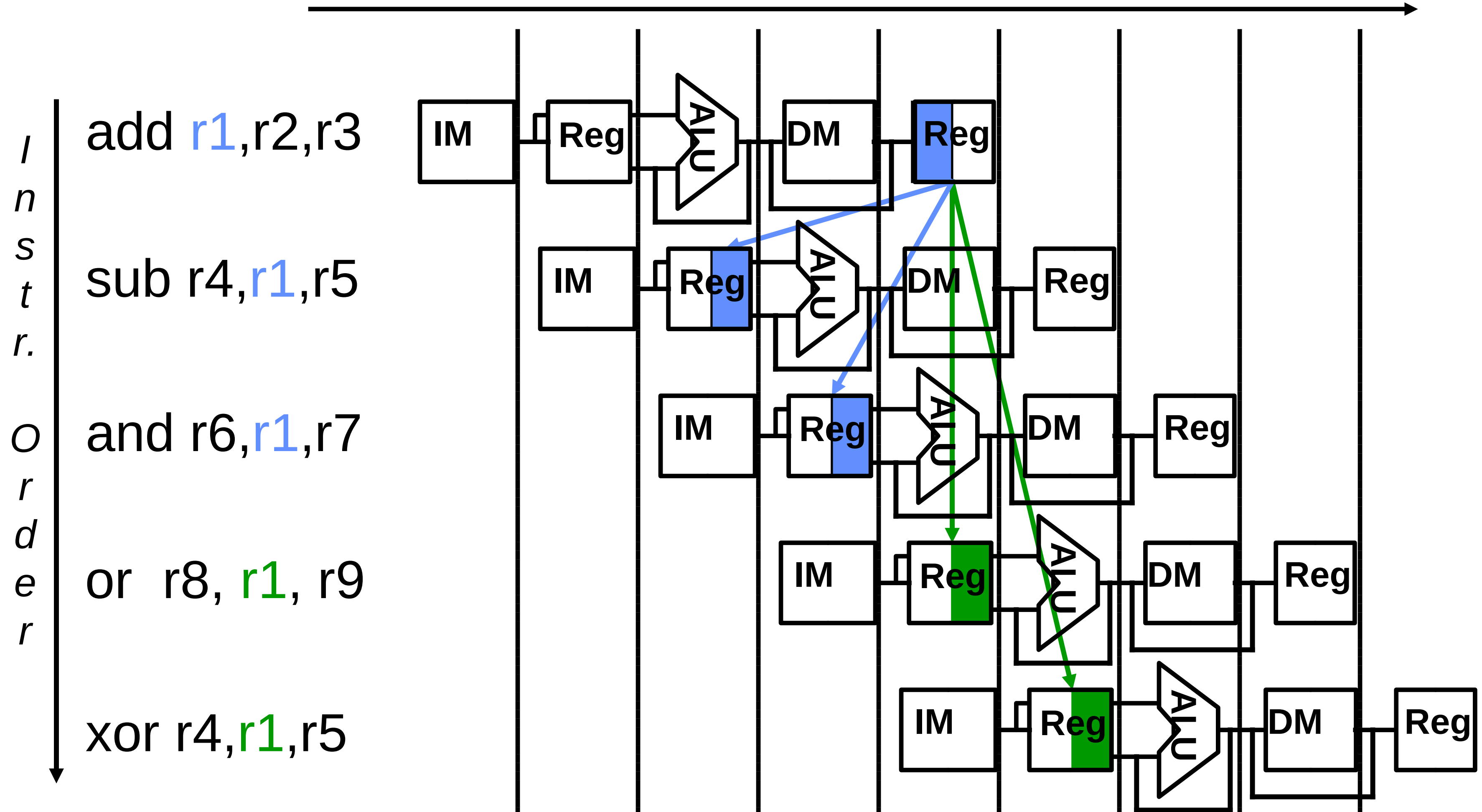
- Dependencies backward in time cause hazards



Which are read before write data hazards?

Register Usage Can Cause Data Hazards

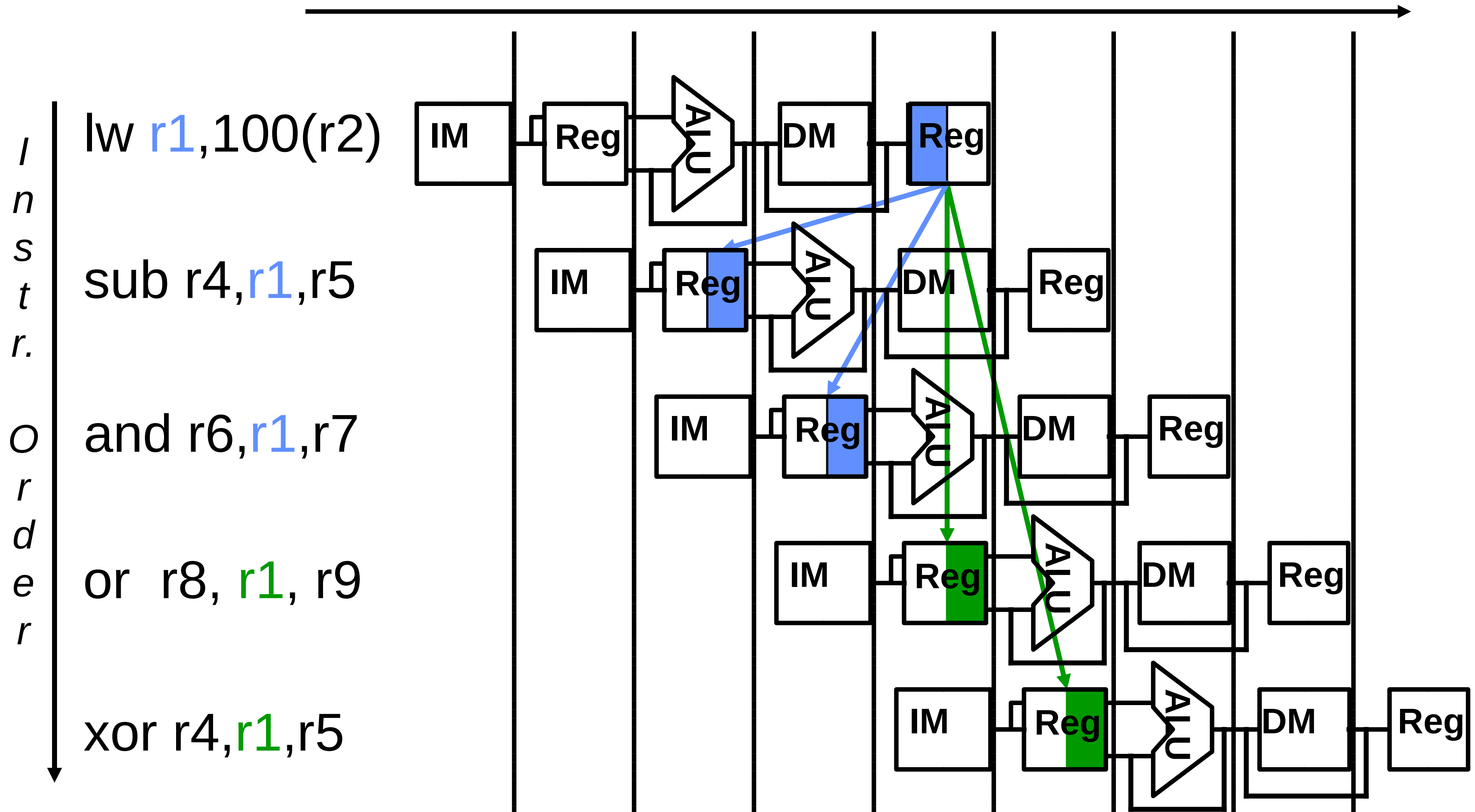
- Dependencies backward in time cause hazards



Read before write data hazards

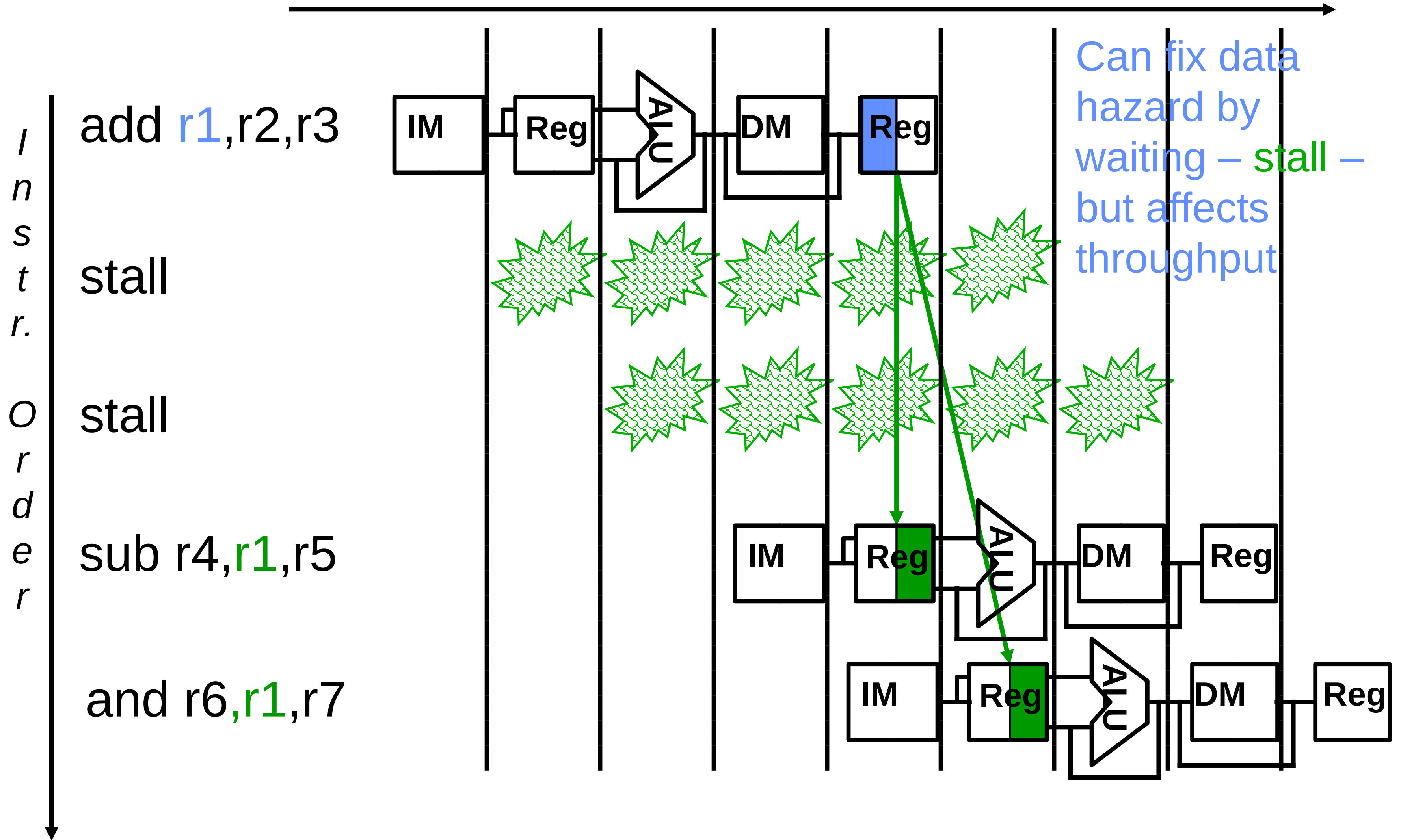
Loads Can Cause Data Hazards

- Dependencies backward in time cause hazards

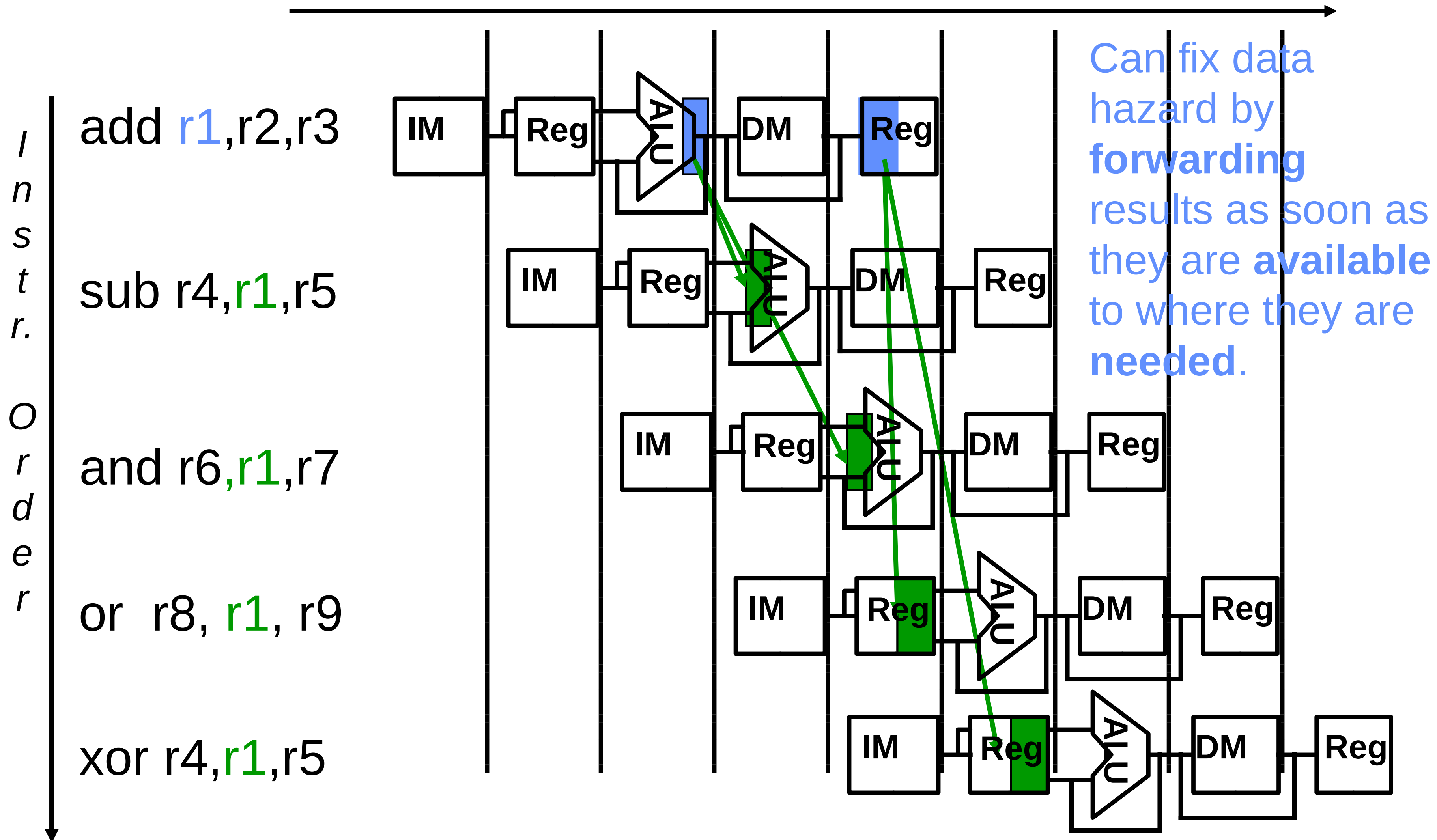


Load-use data hazard

One Way to "Fix" a Data Hazard



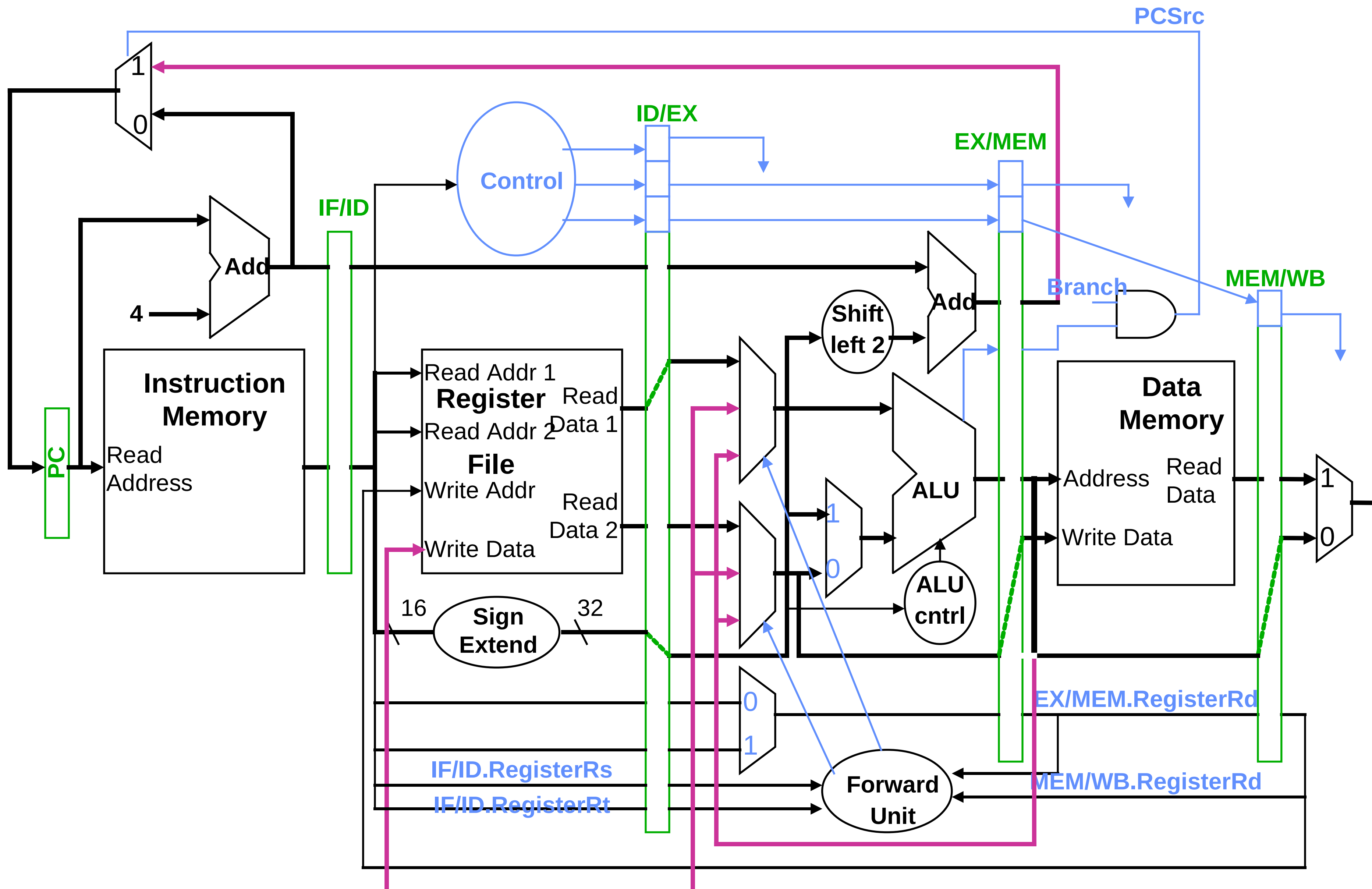
Another Way to “Fix” a Data Hazard



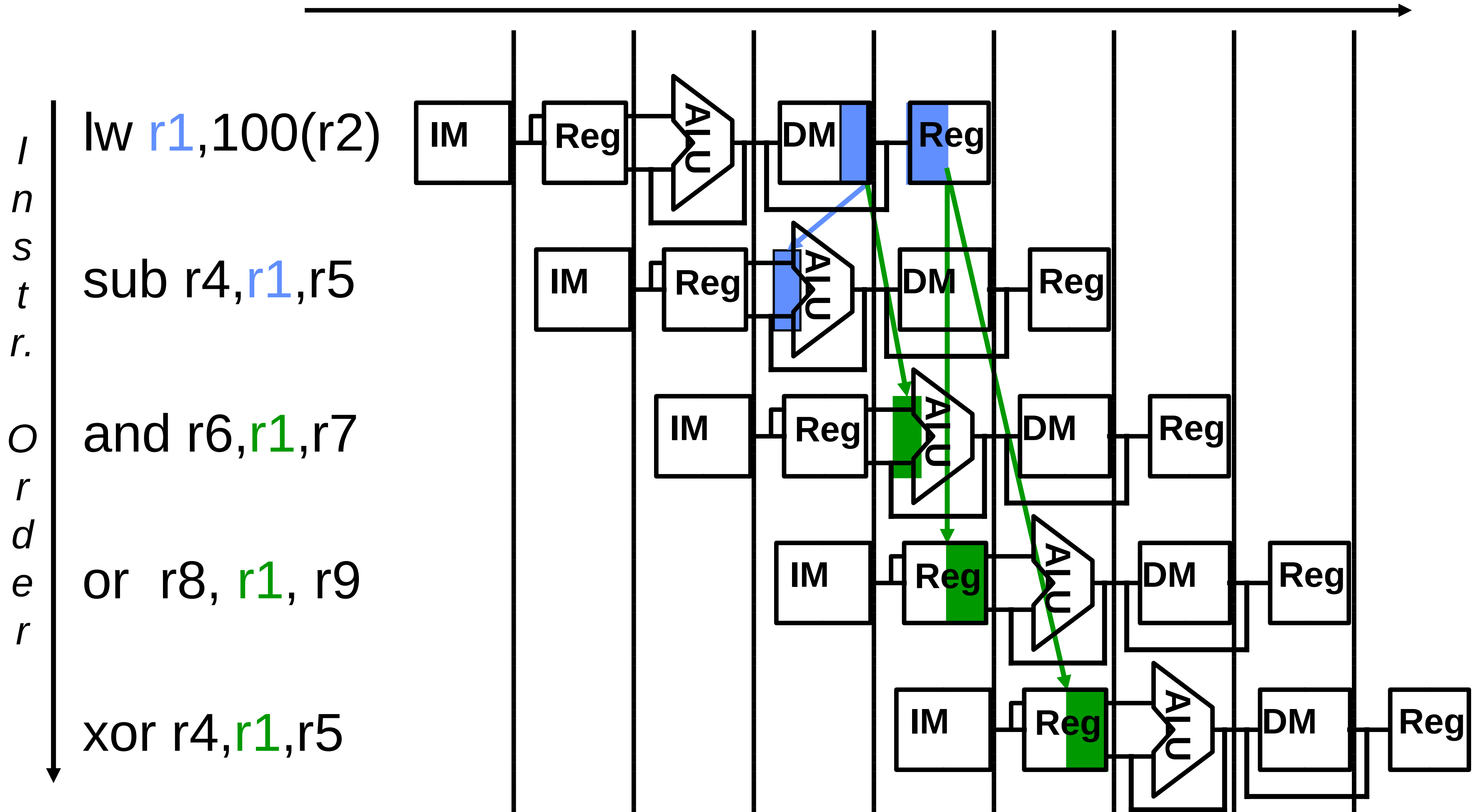
Data Forwarding (aka Bypassing)

- Any data dependence line that goes backwards in time
  - **EX stage generating R-type ALU results or effective address calculation**
  - **MEM stage generating lw results**
- Forward by taking the inputs to the ALU from any pipeline register rather than just ID/EX by
  - **adding multiplexors to the inputs of the ALU so can pass Rd back to either (or both) of the EX's stage Rs and Rt ALU inputs**
    -: normal input (ID/EX pipeline registers)
    -: forward from previous instr (EX/MEM pipeline registers)
    -: forward from instr 2 back (MEM/WB pipeline registers)
  - **adding the proper control hardware**
- With forwarding can run at full speed even in the presence of data dependencies

Datapath with Forwarding Hardware

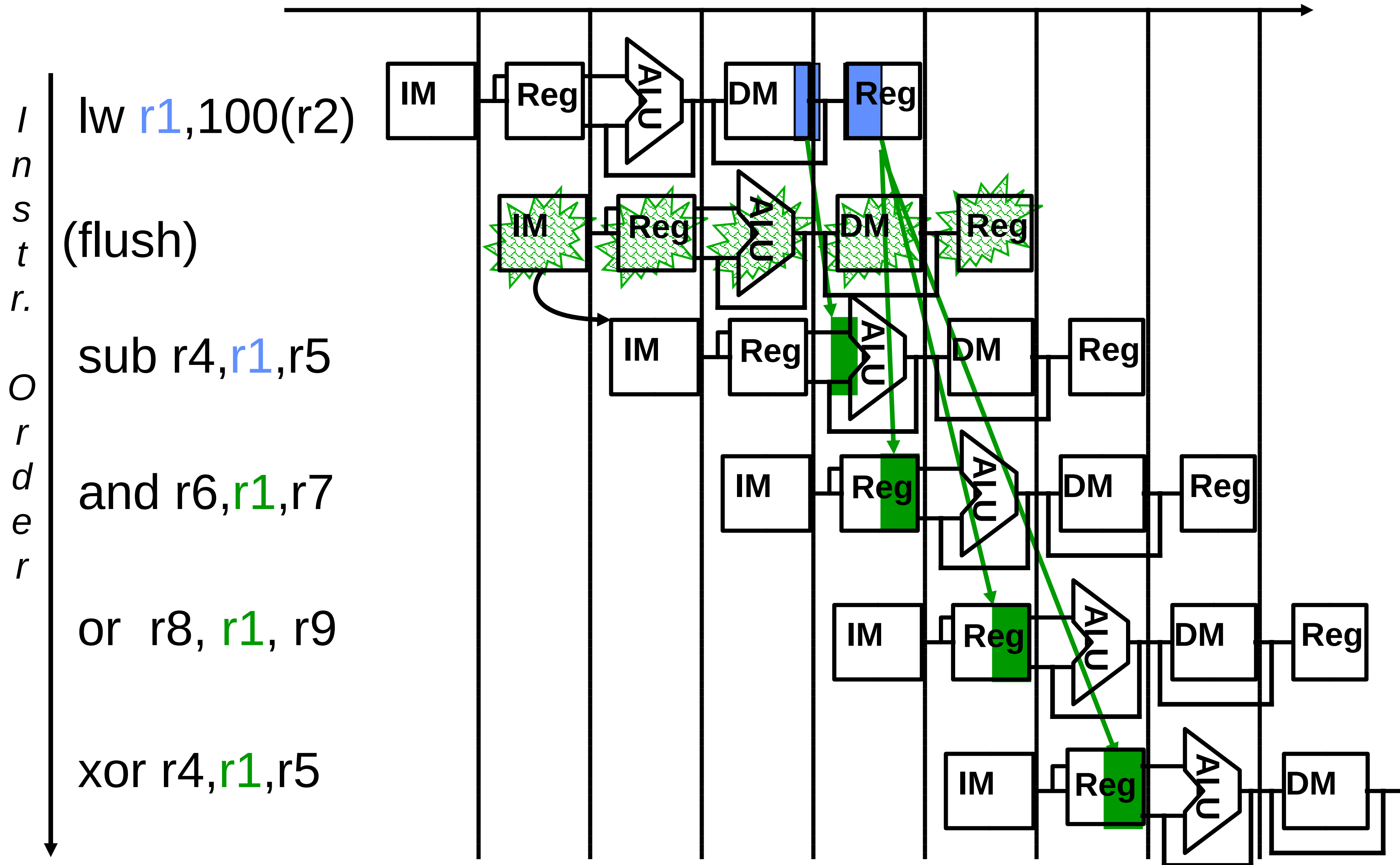


Forwarding with Load-use Data Hazards



- Will still need one stall cycle even with forwarding

Forwarding with Load-use Data Hazards



Control Hazards

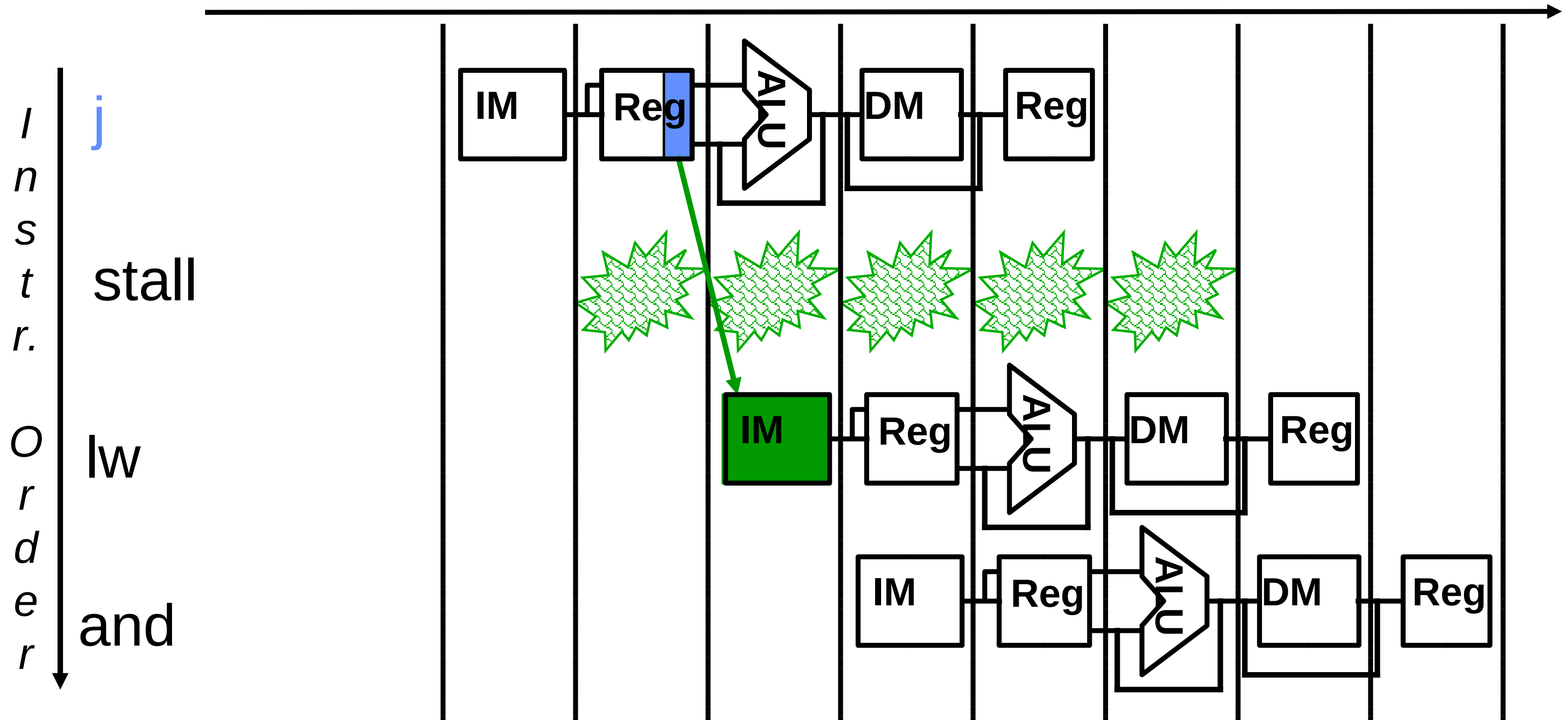
- When the flow of instruction addresses is not what the pipeline expects; incurred by change of flow instructions
  - **Conditional branches (beq, bne)**
  - **Unconditional branches (j)**

- Possible solutions
  - **Stall**
  - **Move decision point earlier in the pipeline**
  - **Predict**
  - **Delay decision (requires compiler support)**

- Control hazards occur less frequently than data hazards; there is nothing as effective against control hazards as forwarding is for data hazards

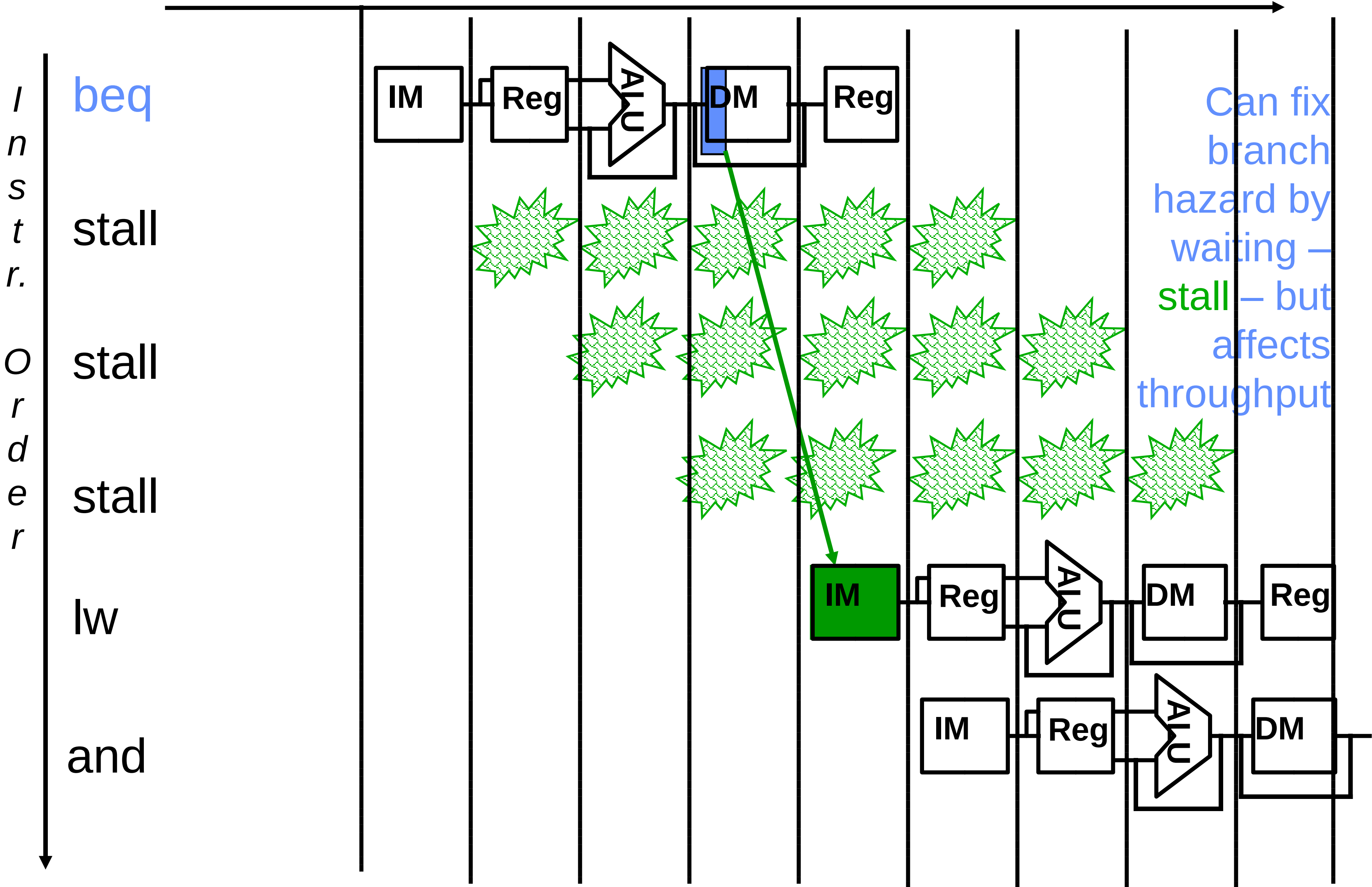
Jumps Incur One Stall

- Jumps not decoded until ID, so one stall is needed



- Fortunately, jumps are very infrequent – only 2% of the SPECint instruction mix

Review: Branches Incur Three Stalls



Branch Prediction

- Resolve branch hazards by assuming a given outcome and proceeding without waiting to see the actual branch outcome

1. Predict not taken – always predict branches will not be taken, continue to fetch from the sequential instruction stream, only when branch is taken does the pipeline stall
   - **If taken, flush instructions in the pipeline after the branch**
     - in IF, ID, and EX if branch logic in MEM – three stalls
     - in IF if branch logic in ID – one stall
   - **ensure that those flushed instructions haven't changed machine state– automatic in the MIPS pipeline since machine state changing operations are at the tail end of the pipeline (MemWrite or RegWrite)**
   - **restart the pipeline at the branch destination**