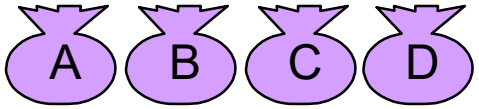
CSE 3421
Computer Architecture

Chapter 4: The Processor
- Pipelining -

Xiaodong Zhang

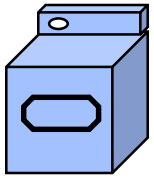
Pipelining: Its Natural!

- Dave has four loads of clothes to wash, dry, and fold



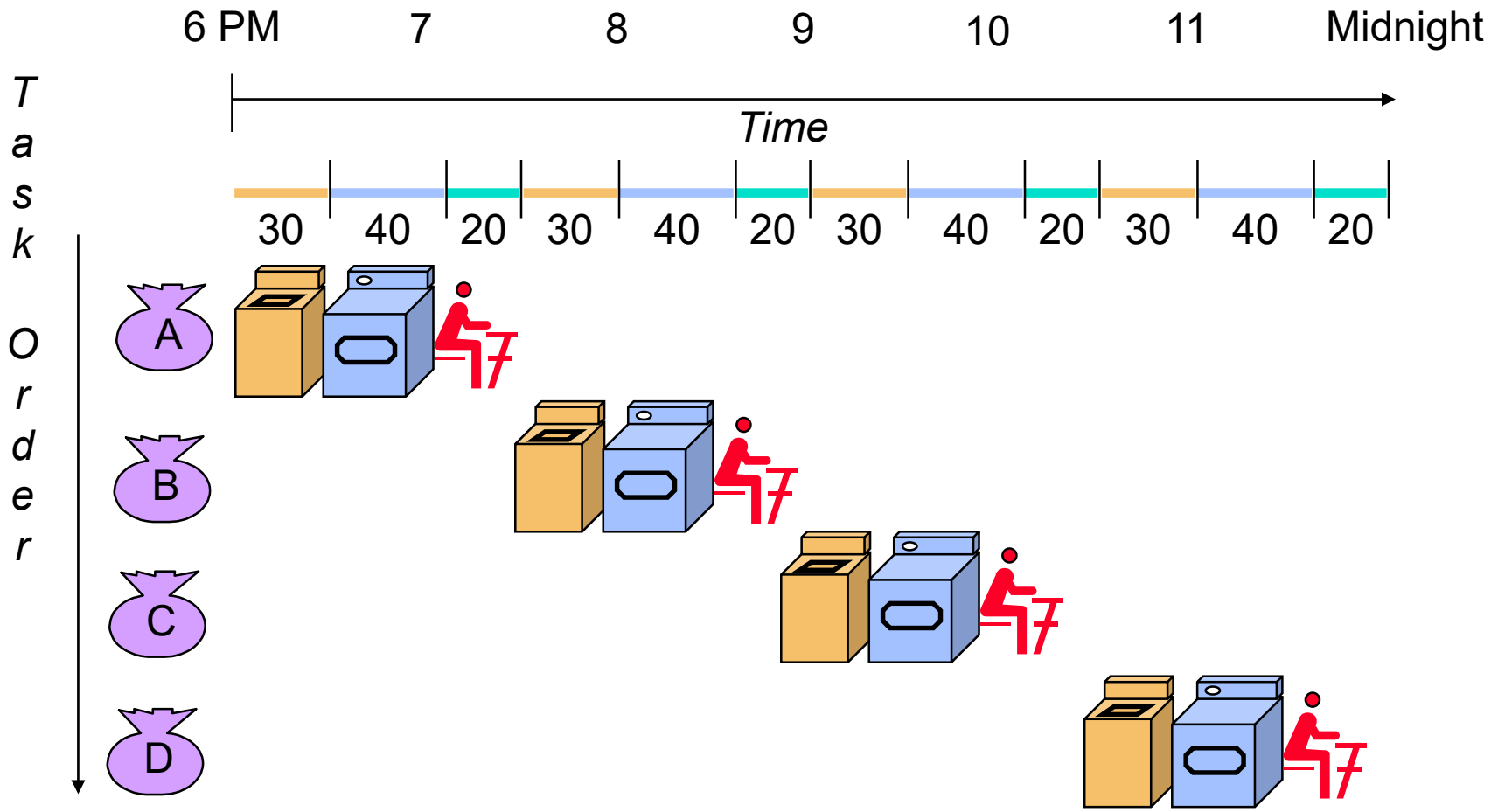
- Washer takes 30 minutes

- Dryer takes 40 minutes



- “Folder” takes 20 minutes

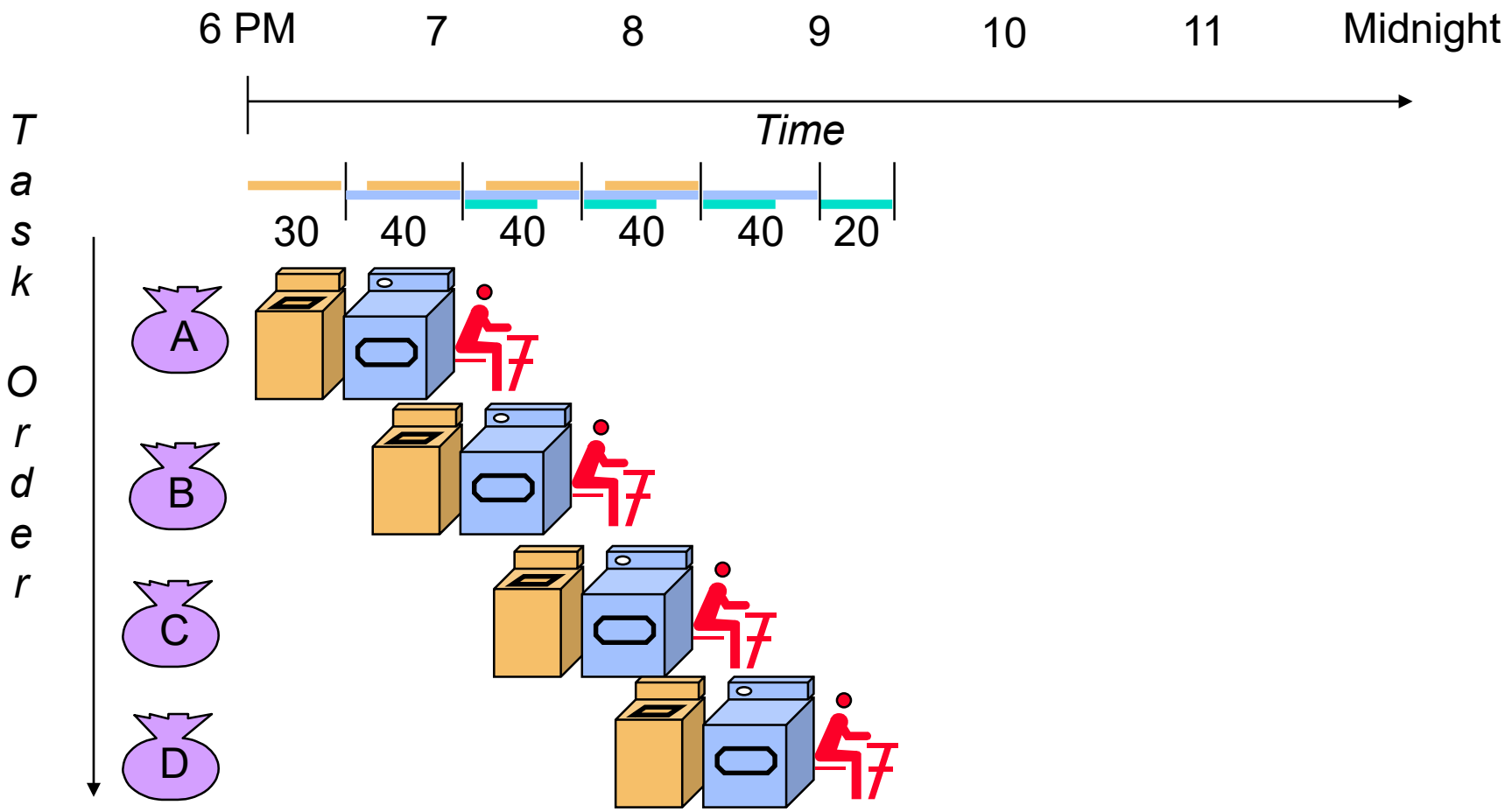
Sequential Laundry



Sequential laundry takes 6 hours for 4 loads;

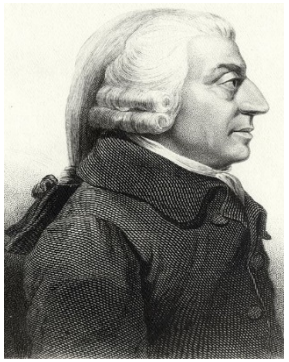
If Dave learned pipelining, how long would laundry take?

Pipelined Laundry



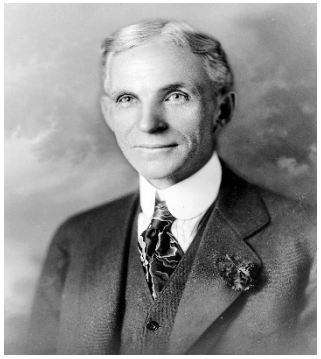
Pipelined laundry takes 3.5 hours for 4 loads;

Division of Labor vs Computing Specialization



Economic growth is rooted in the increasing division of labor, which is primarily related to the specialization of the labor force, essentially the breaking down of large jobs into many tiny components. Each worker becomes an expert in one isolated area of production, increasing his efficiency.

Adam Smith, *The Wealth of Nations*, 1776



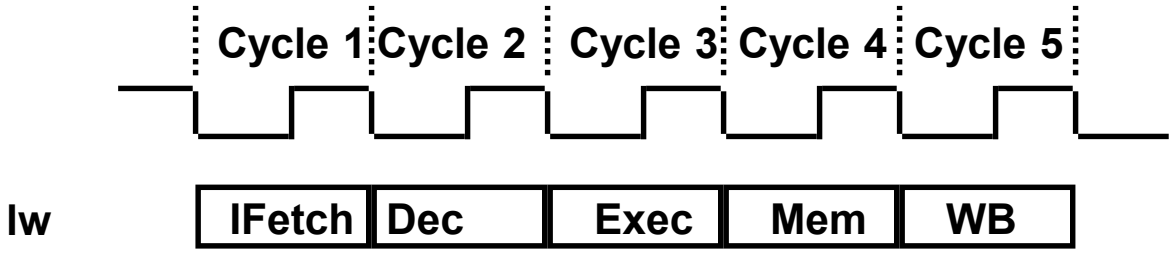
In1913, **Henry Ford** divided t-model car into **8,772** units of specialized work with high productivity and low price

Ford's **assembly line** is inclusive to all specialized jobs

Ford put the wheels on the globe

The "Division of Labor" is well used as "pipeline" in processor design

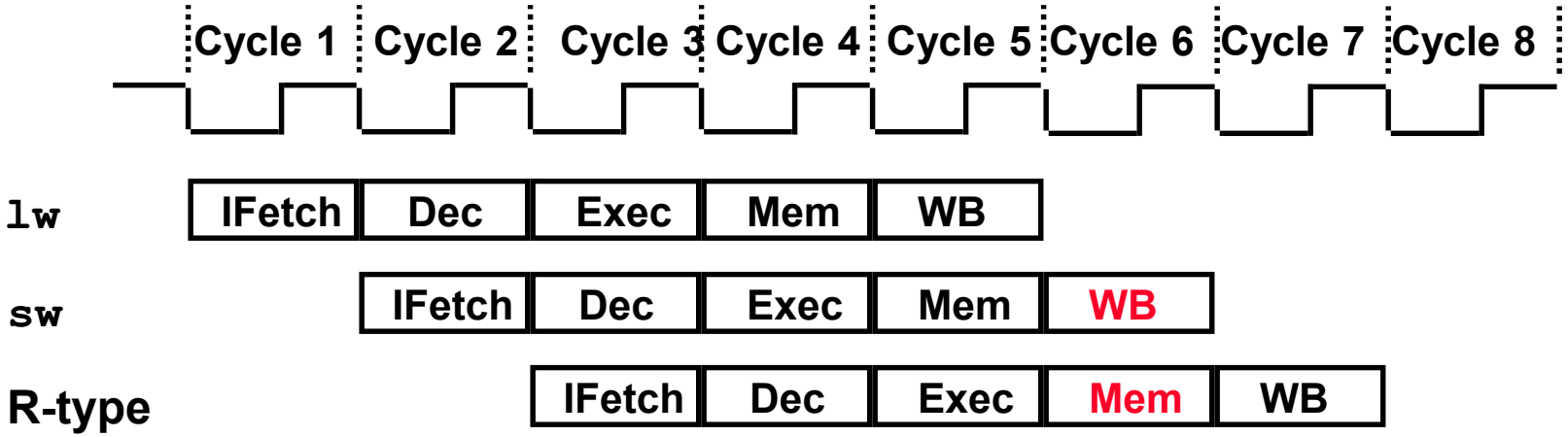
The Five Stages of Load Instruction



- IFetch: Instruction Fetch and Update PC
- Dec: Registers Fetch and Instruction Decode
- Exec: Execute R-type; calculate memory address
- Mem: Read/write the data from/to the Data Memory
- WB: Write the result data into the register file

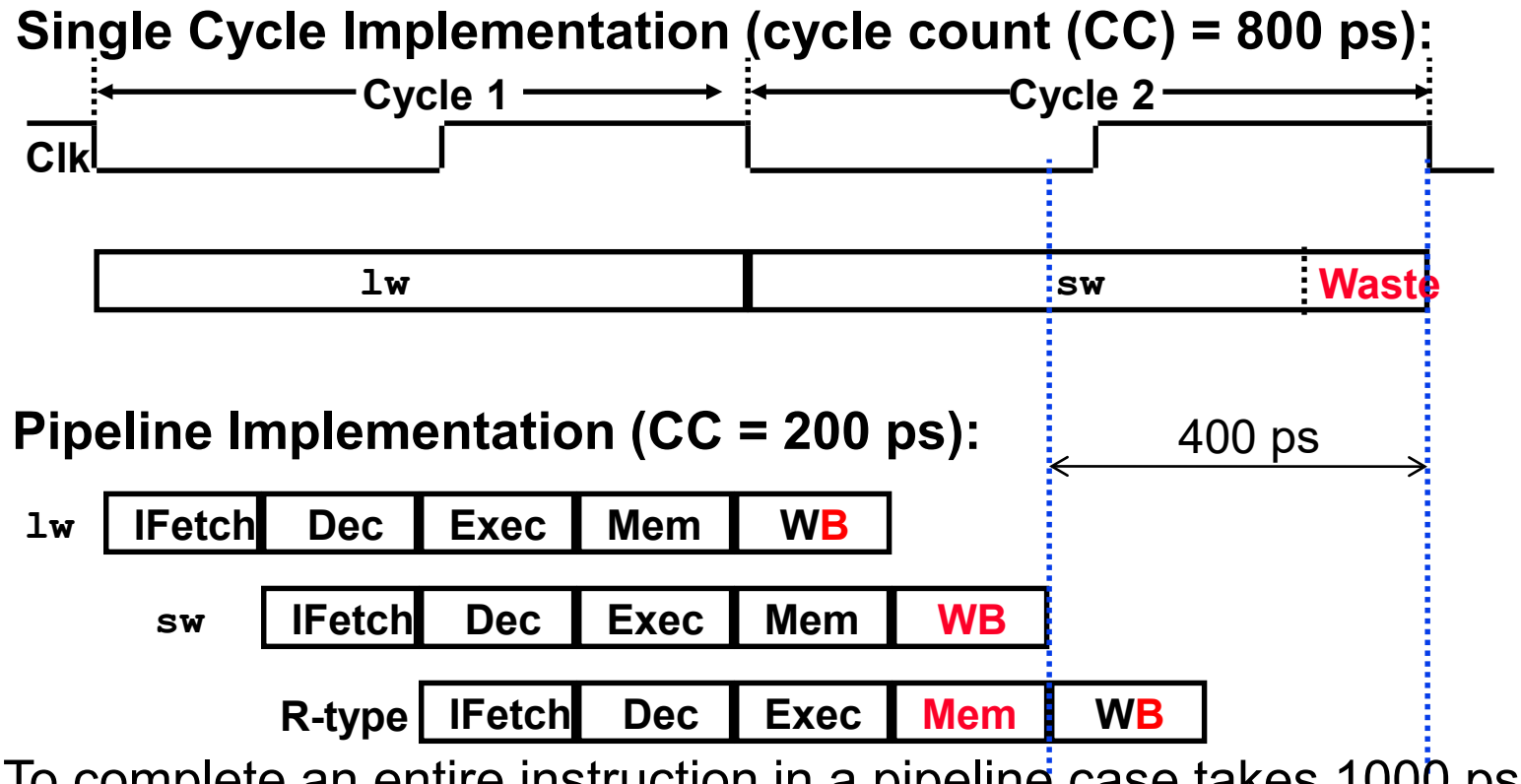
A Pipelined MIPS Processor

- Start the next instruction before the current one has completed
  - improves throughput - total amount of work done in a given time
  - instruction latency (execution time, delay time, response time - time from the start of an instruction to its completion) is *not* reduced



- clock cycle (pipeline stage time) is limited by the **slowest** stage
  - for some stages don't need the whole clock cycle (e.g., write reg)
  - for some instructions, some stages are wasted cycles (i.e., nothing is done during that cycle for that instruction)

Single Cycle versus Pipeline



- To complete an entire instruction in a pipeline case takes 1000 ps (compared to 800 ps for single cycle case). Why ?
  - Single cycle design takes advantage of unequal lengths in each instructions
  - In pipelining design, the time in each stage must be the maximum. Compared with single cycle, the execution time is longer for a single instruction
  - The advantage of pipelining is High Performance on **long streams of instructions**.

Long sequence of instructions benefit pipelining

- How long does each take to complete 1,000,000 adds ?
  - Single cycle: 800ps * 1 M = 800 M ps
- Pipeline: 1 add takes 1*1000 ps
  - 6 adds take 2*1000 ps
  - 11 adds take 3*1000 ps
  - … an arithmetic progression (equal difference series)
  - $A_n = A_1 + (n-1) * d$, where d is the difference between any pair
  - [1 + (n-1) * 5] adds take n* 1000 ps, and d=5
  - For 1 M adds, n=[(1M -1) / 5] +1 ≈ 1 M /5 = 0.2 M
  - 1 M adds take 0.2 M * 1000 = 200 M ps

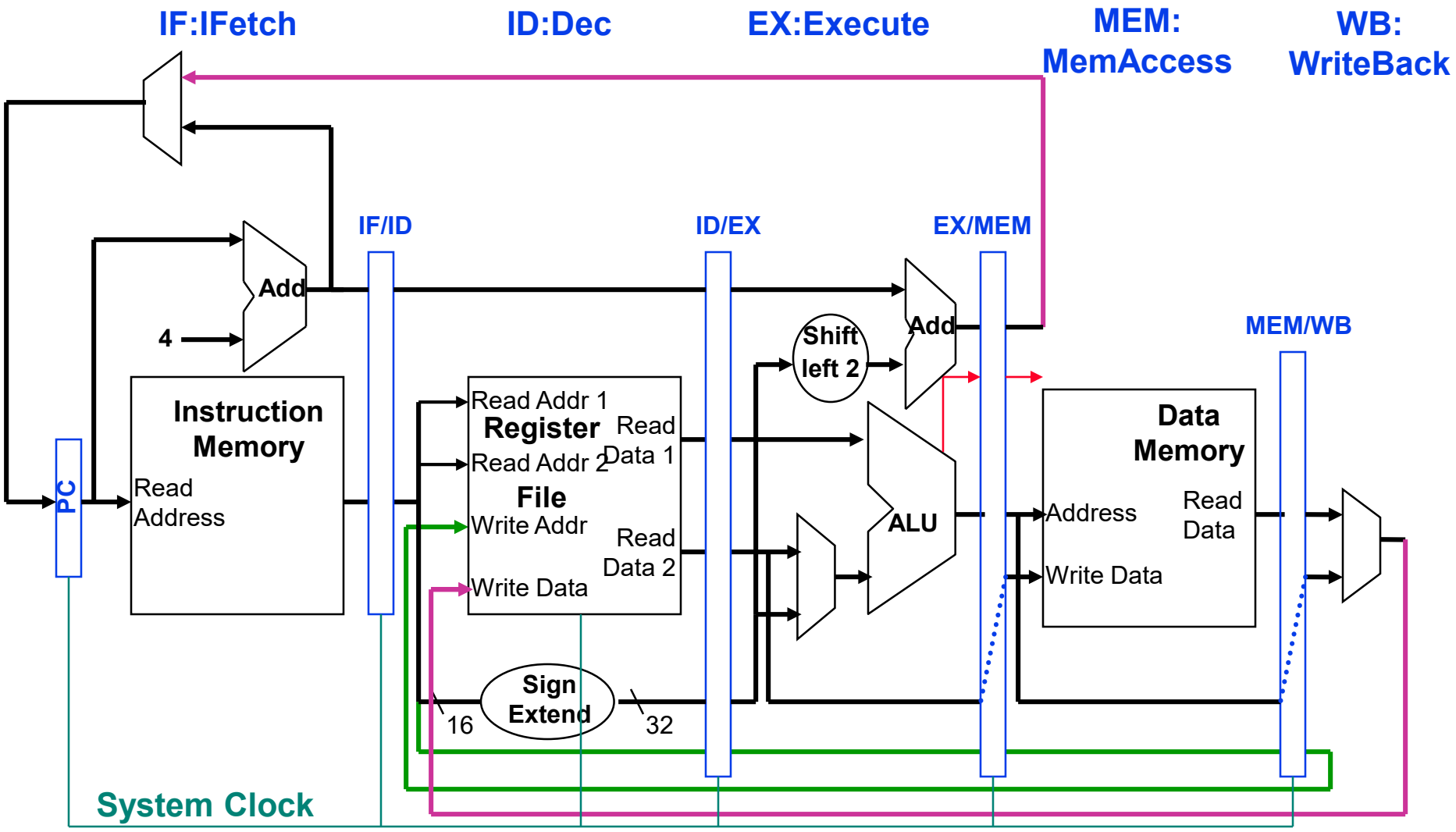
- Pipelined execution time =
- single instruction time * # of instructions / # of stages in an instruction + overhead

Pipelining the MIPS ISA

- What makes it easy
  - all instructions are the same length (32 bits)
    - can fetch in the 1st stage and decode in the 2nd stage
  - Half cycle is needed only to write result to register
    - can begin reading register file in 2nd stage
  - memory operations occur only in loads and stores
    - can use the execution stage to calculate memory addresses
  - each instruction writes at most one result (i.e., changes the machine state) and does it in the last pipeline stages (MEM or WB)
  - operands must be aligned in memory, so a single data transfer takes only one data memory access (1 byte or 4 bytes)

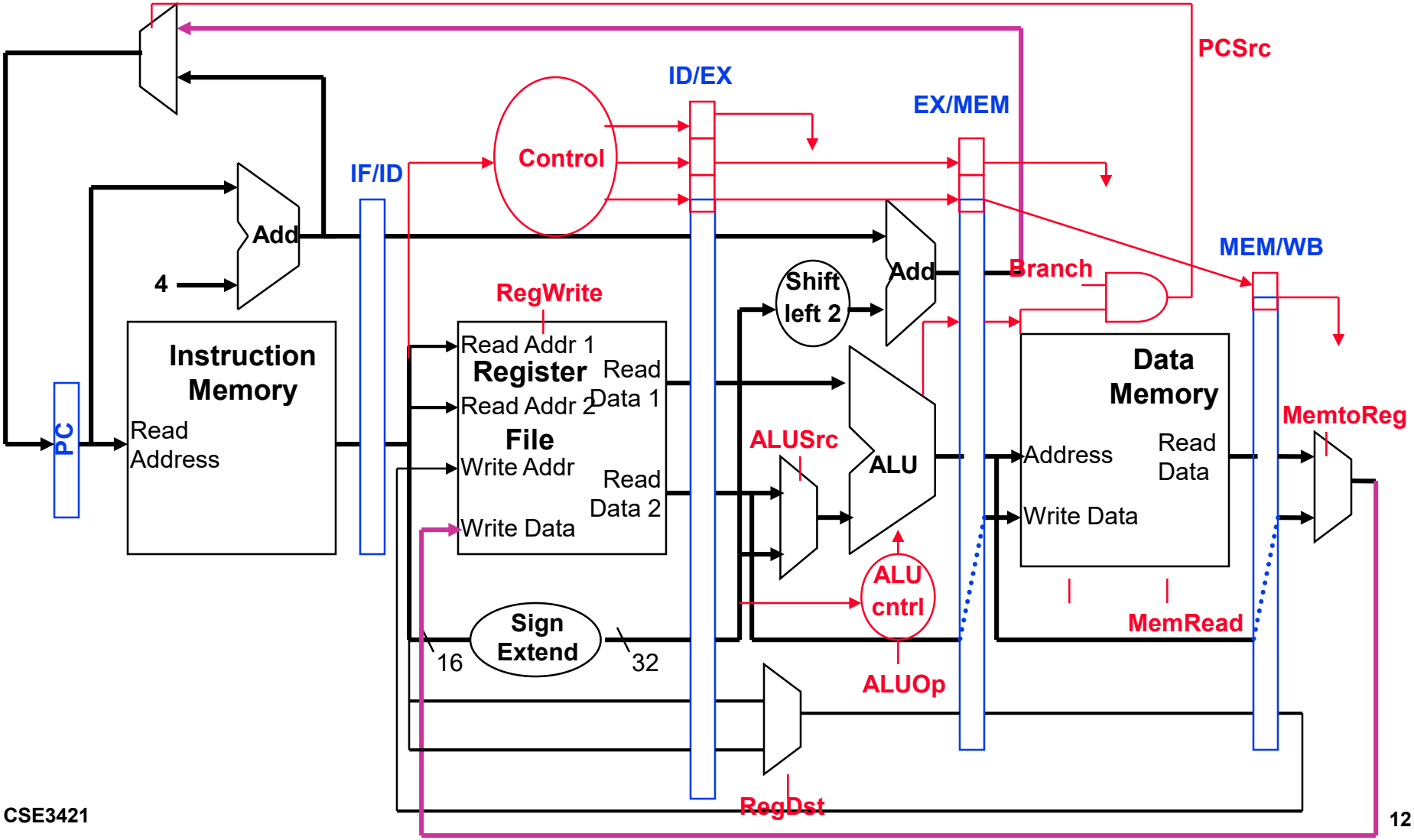
The 5 Stages of the MIPS Pipeline Datapath

- State registers between each stage are isolated, only updated in the last stage. The system **clock ticks in each stage concurrently**

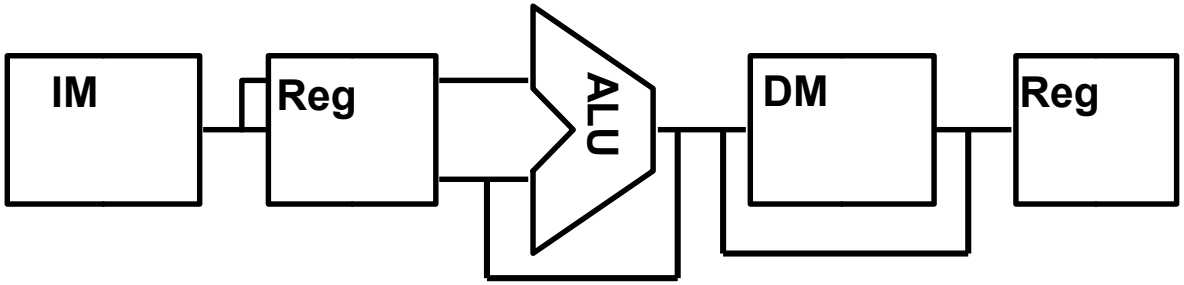


MIPS Pipeline Control Path Modifications

- All control signals can be determined during Decode
  - and held in the state registers between pipeline stages (except branch)



Building Blocks (Toy Bricks) in MIPS Pipeline

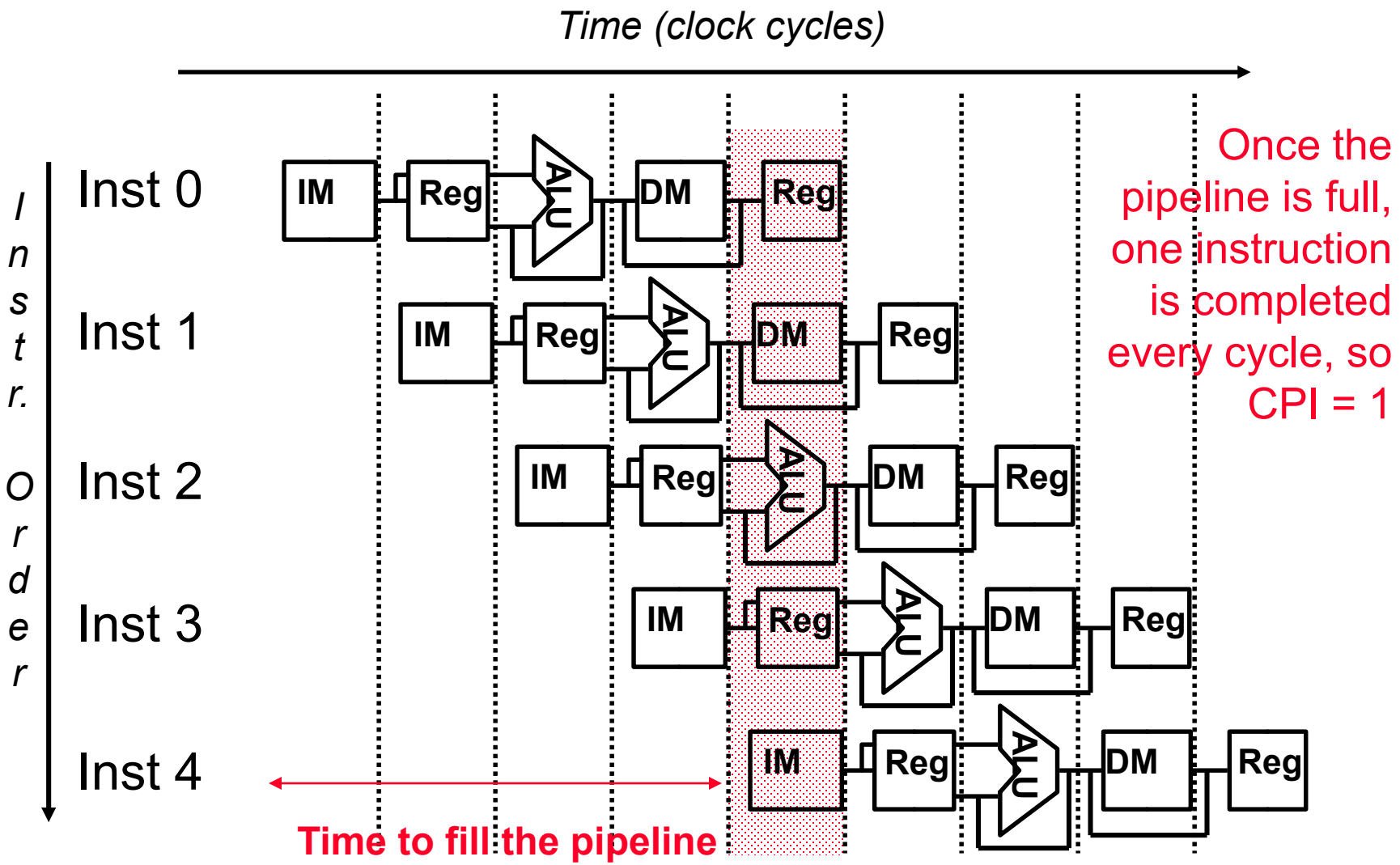


IM: instruction memory, **Reg:** registers in decoding stage, **ALU:** calculation, **DM:** memory access,

Last Reg: write to memory or opposite direction

- Can help with answering questions like:
  - How many cycles does it take to execute this code?
  - What is the ALU doing during cycle 4?
  - Is there a hazard, why does it occur, and how can it be fixed?

Why Pipeline? For Performance!

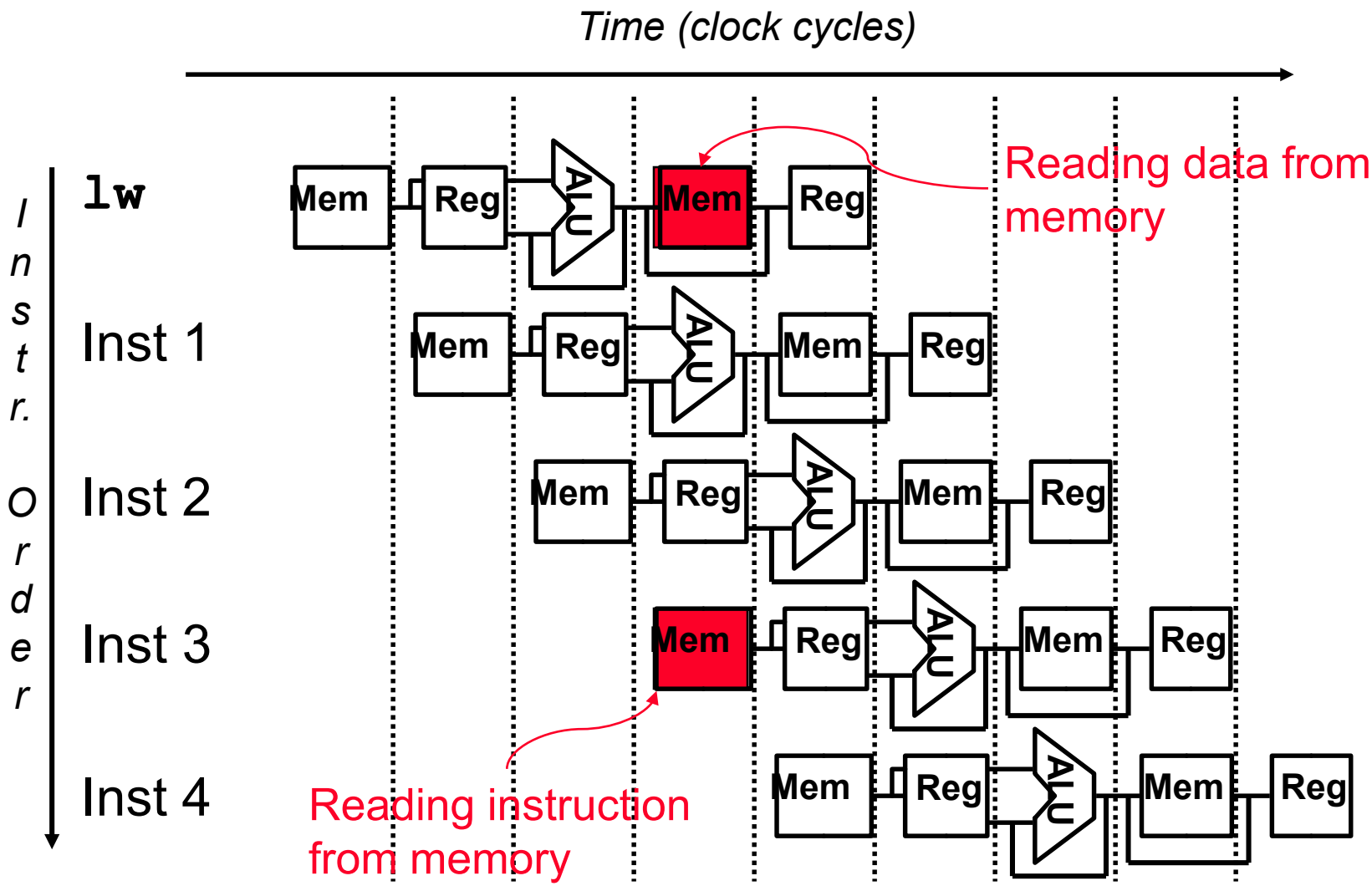


Can Pipelining Get Us Into Trouble?

- 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
    - An instruction's source operand(s) are produced by a prior instruction still in the pipeline
  - control hazards: attempt to make a decision about program control flow before the condition has been evaluated and the new PC target address calculated
    - branch and jump instructions, exceptions

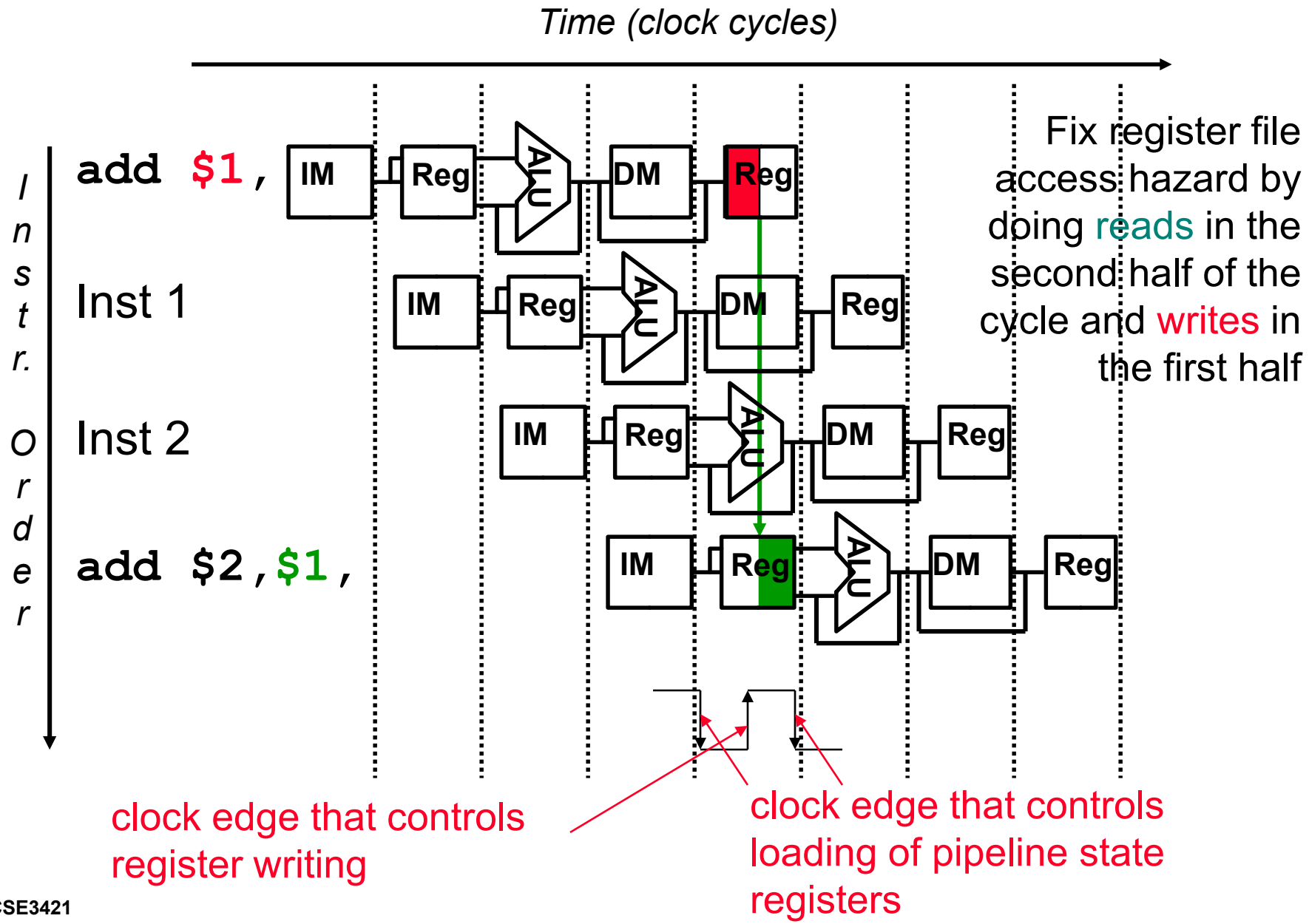
- Can usually resolve hazards by waiting
  - pipeline control must detect the hazard
  - and take action to resolve hazards

A Single Memory Would Be a Structural Hazard



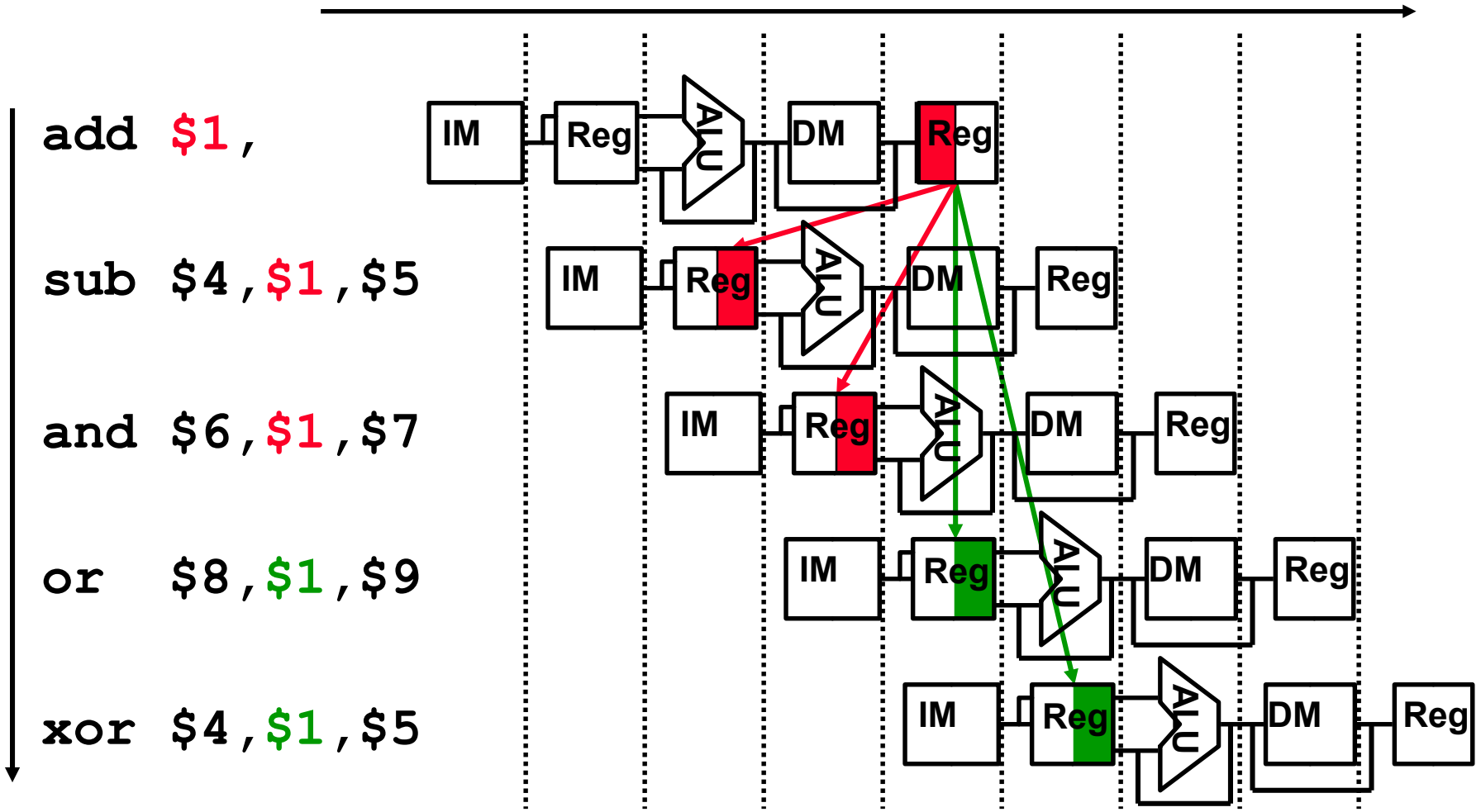
- Fix with separate instr and data memories (I$ and D$)

How About Register File Access?



Register Usage Can Cause Data Hazards

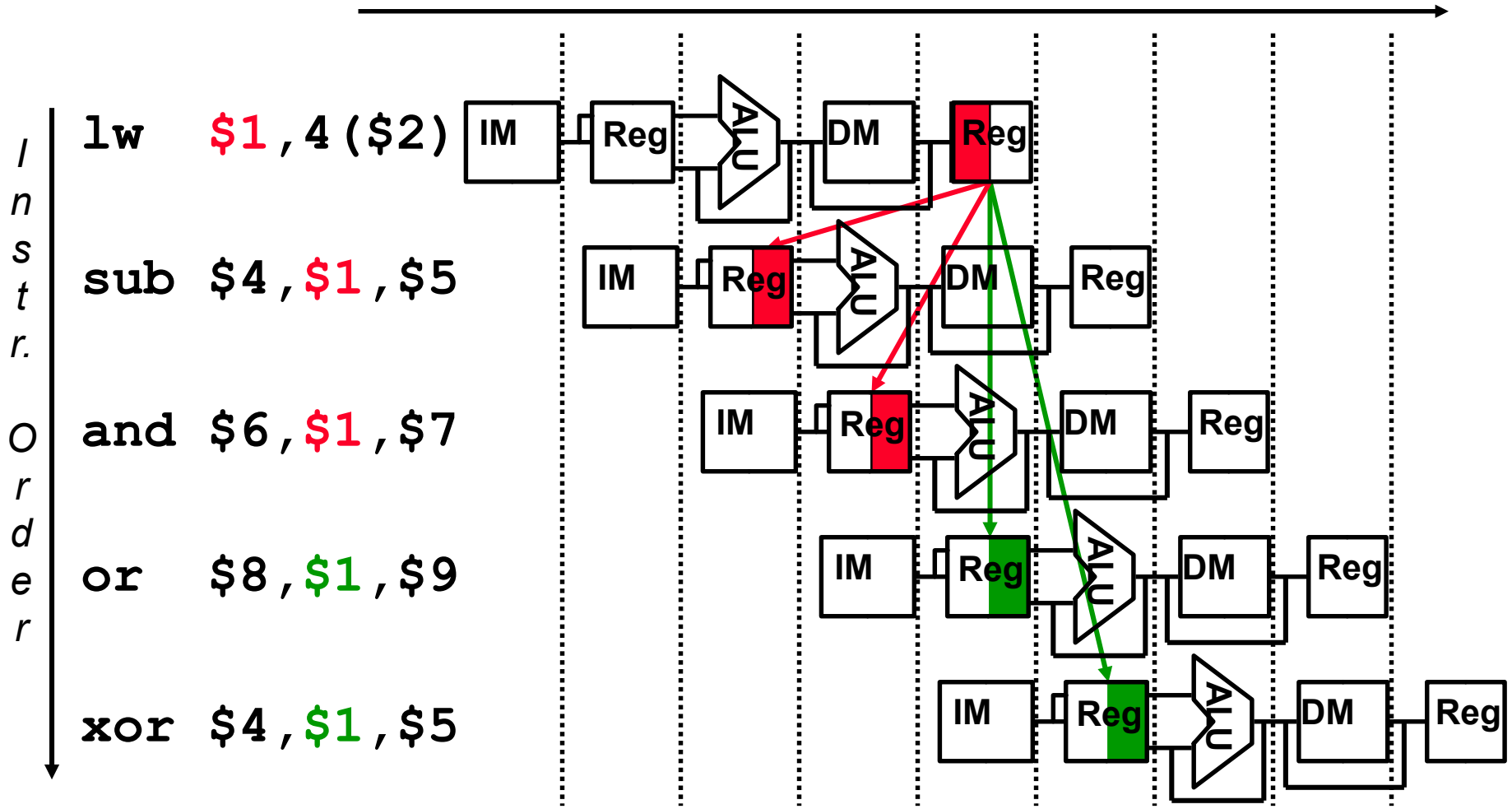
- Dependencies backward in time cause hazards



- Read before write data hazard

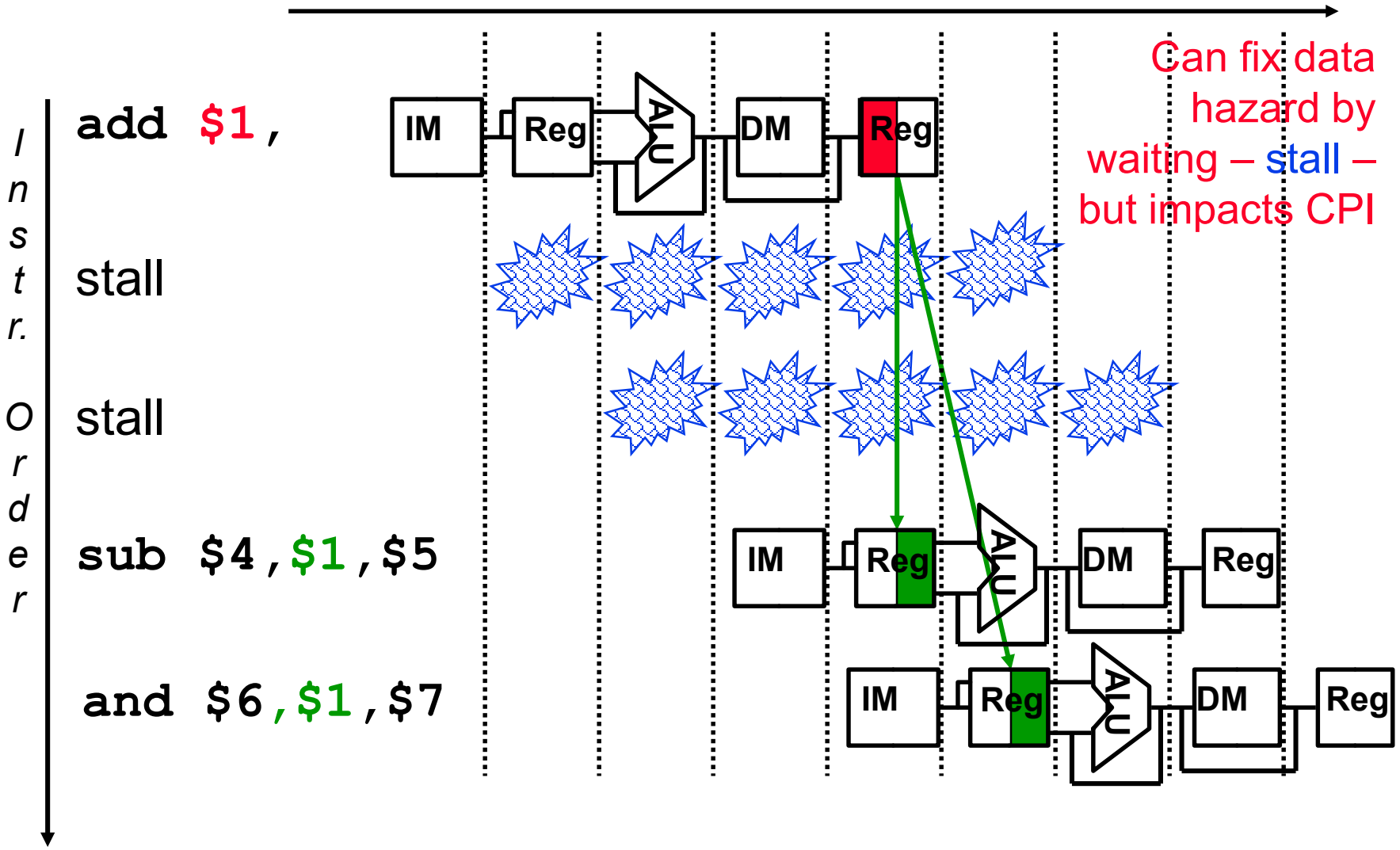
Loads Can Cause Data Hazards

- Dependencies backward in time cause hazards

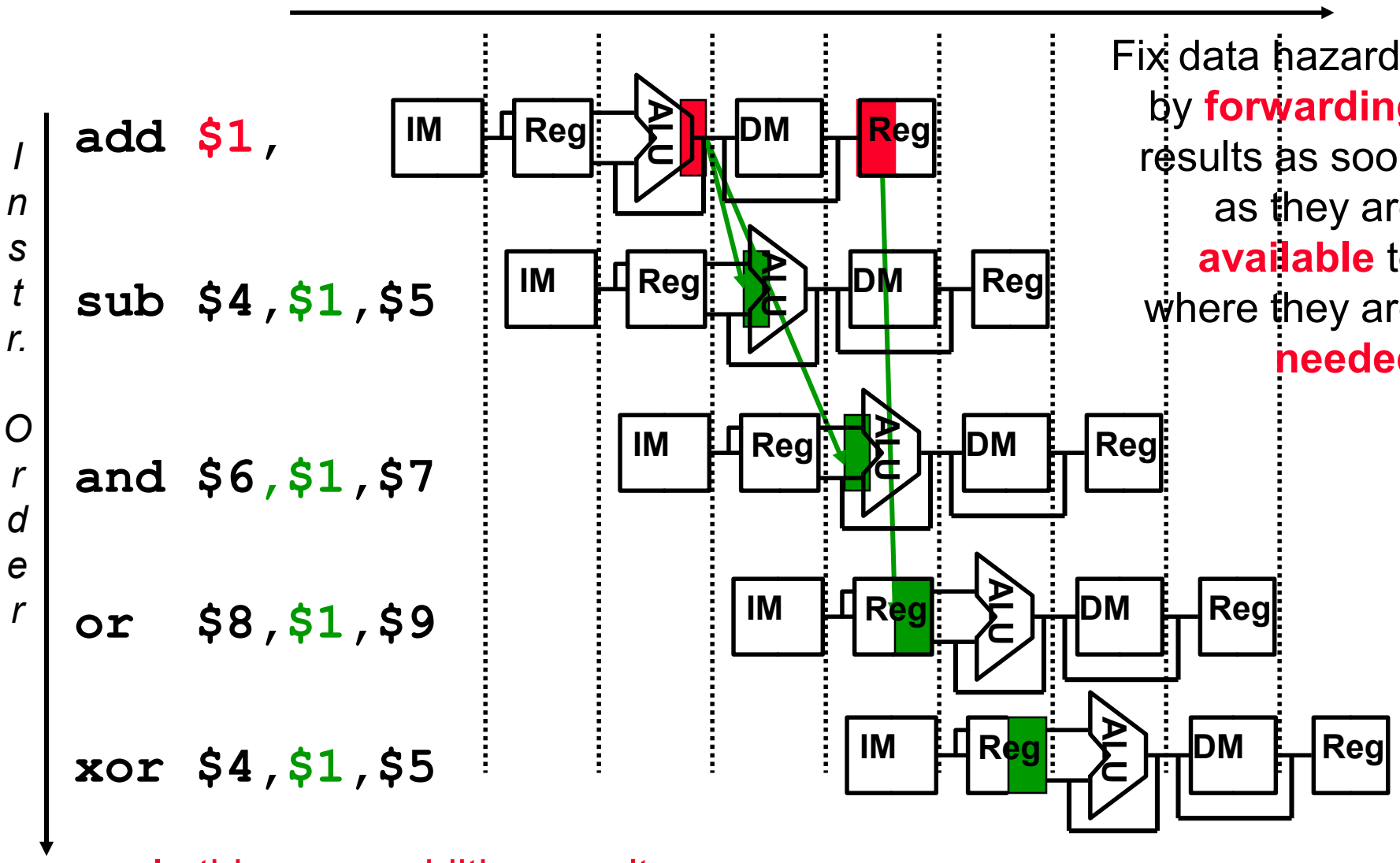


- Load-use data hazard

One Way to "Fix" a Data Hazard



Another Way to "Fix" a Data Hazard



Fix data hazards by **forwarding** results as soon as they are **available** to where they are **needed**

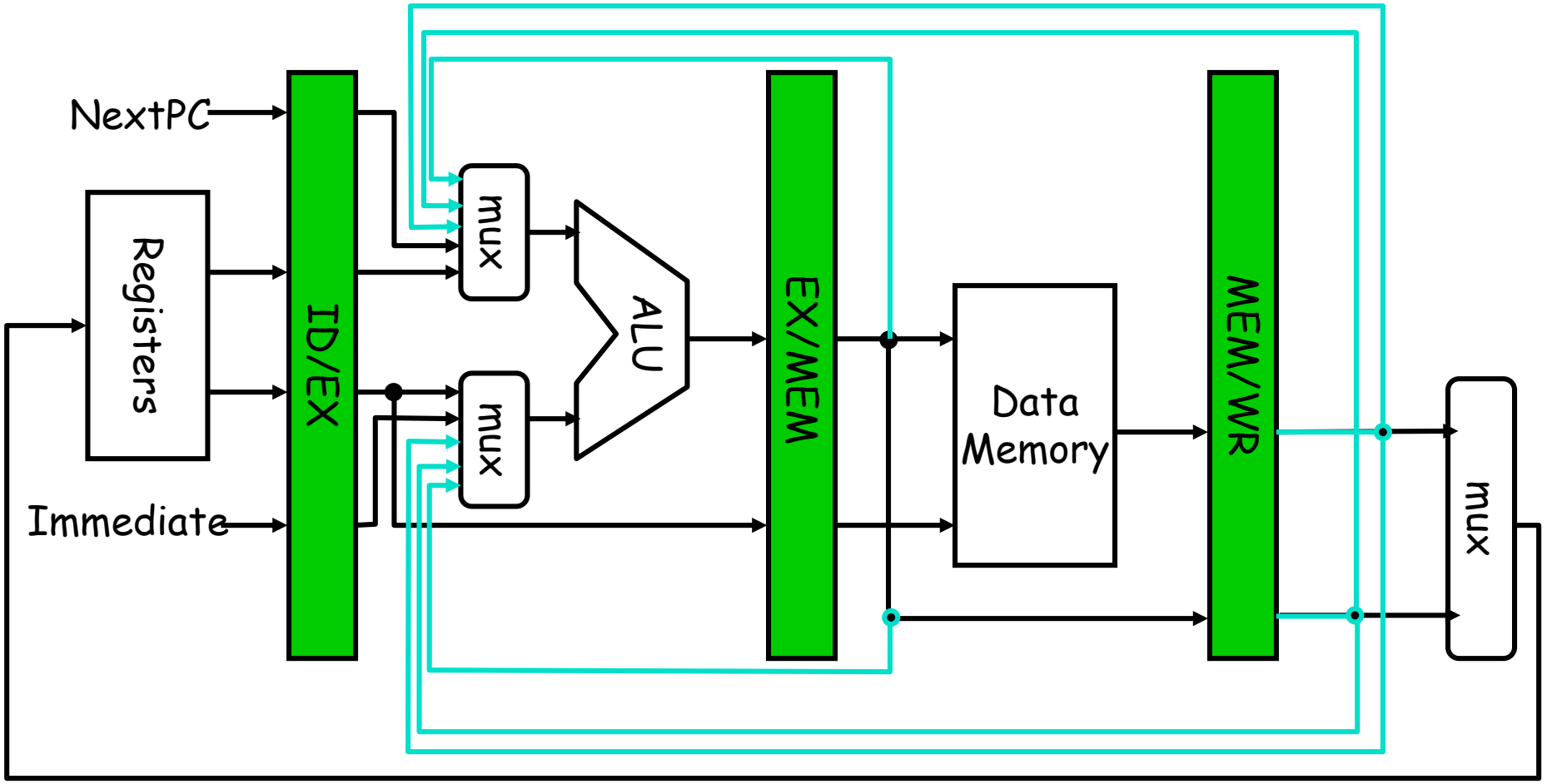
In this case, addition result can obtain in 3rd stage

Data Forwarding (aka Bypassing)

- Take the result from the earliest point that it exists in any of the pipeline state registers and forward it to the functional units (e.g., the ALU) that need it that cycle
- For ALU functional unit: the inputs can come from any pipeline register rather than just from ID/EX by
  - adding multiplexors to the inputs of the ALU
  - connecting the Rd write data in EX/MEM or MEM/WB to either (or both) of the EX's stage Rs and Rt ALU mux inputs
  - adding the proper control hardware to control the new muxes
- Other functional units may need similar forwarding logic (e.g., the DM)
- With forwarding can achieve a CPI of 1 even in the presence of data dependencies

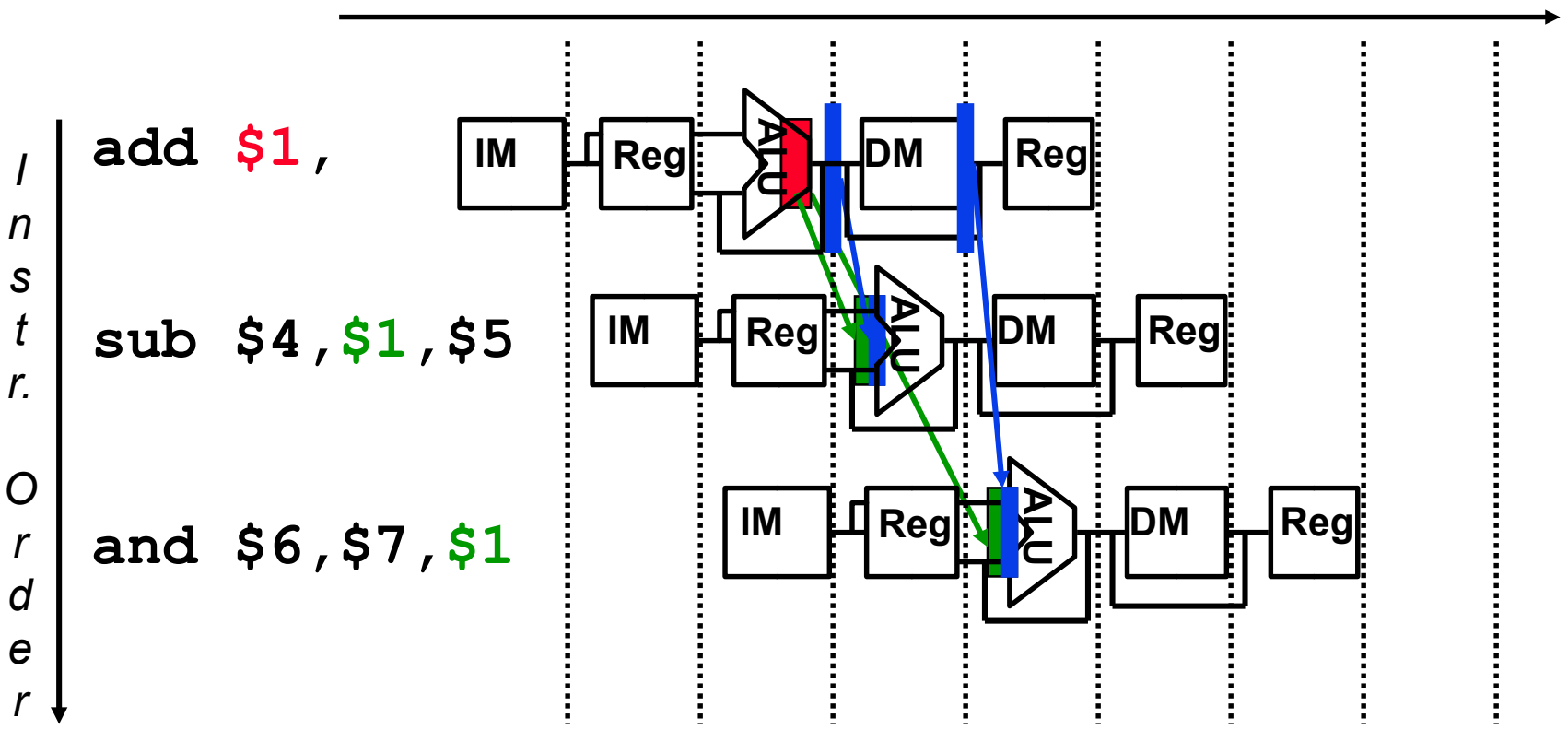
HW Change for Forwarding

Figure A.23, Page A-37



What circuit detects and resolves this hazard?
The multiplexor selects result quickly as input

Forwarding Illustration

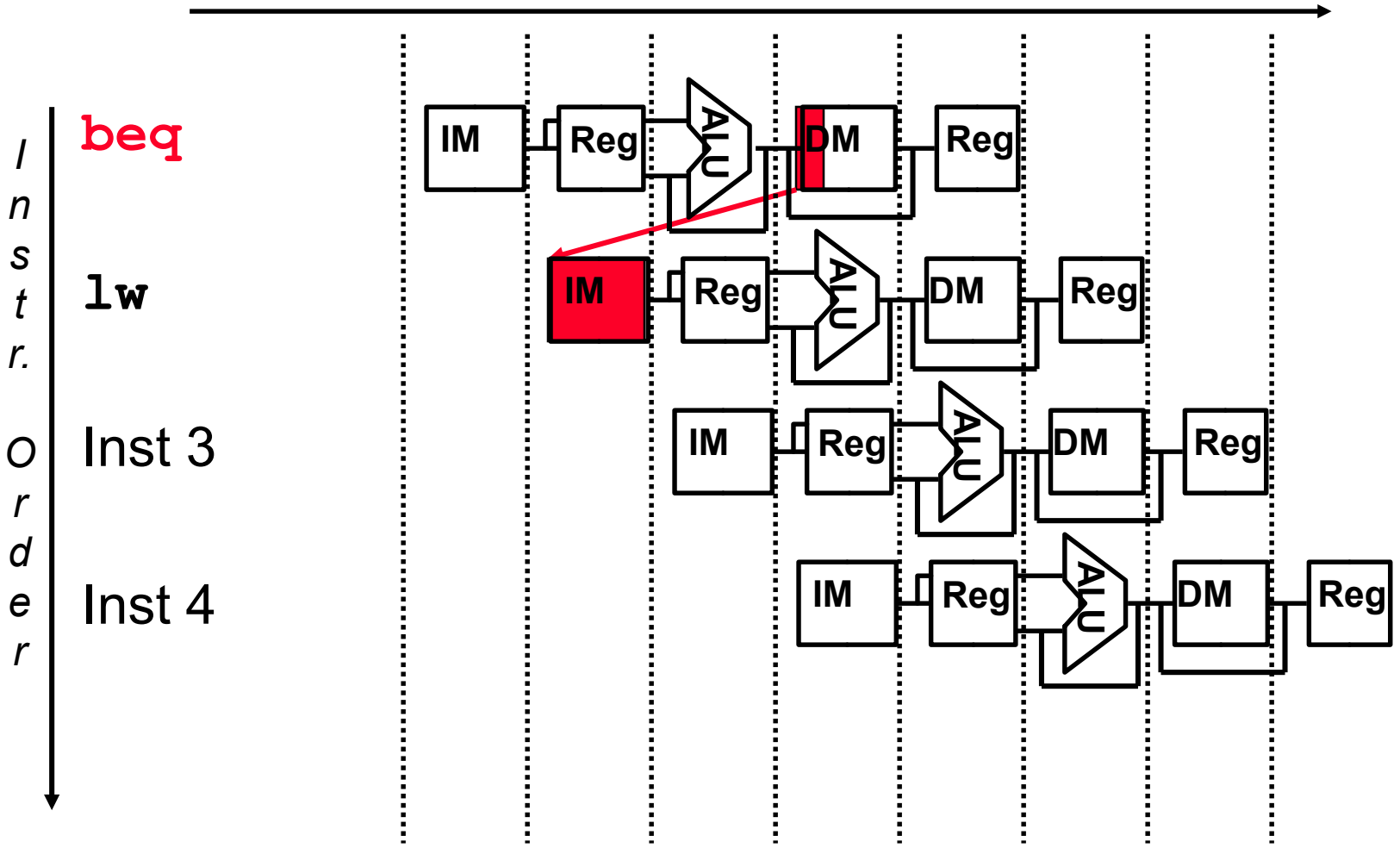


Control Hazards

- When the flow of instruction addresses is not sequential (i.e., PC = PC + 4); incurred by change of flow instructions
  - Unconditional branches (`j, jal, jr`)
  - Conditional branches (`beq`, `bne`)
  - Exceptions
- Possible approaches
  - Stall (impacts CPI)
  - Move decision point as early in the pipeline as possible, thereby reducing the number of stall cycles
  - Delay decision (requires compiler support)
  - Predict and hope for the luck !
- Control hazards occur less frequently than data hazards, but there is *nothing* as effective against control hazards as forwarding is for data hazards

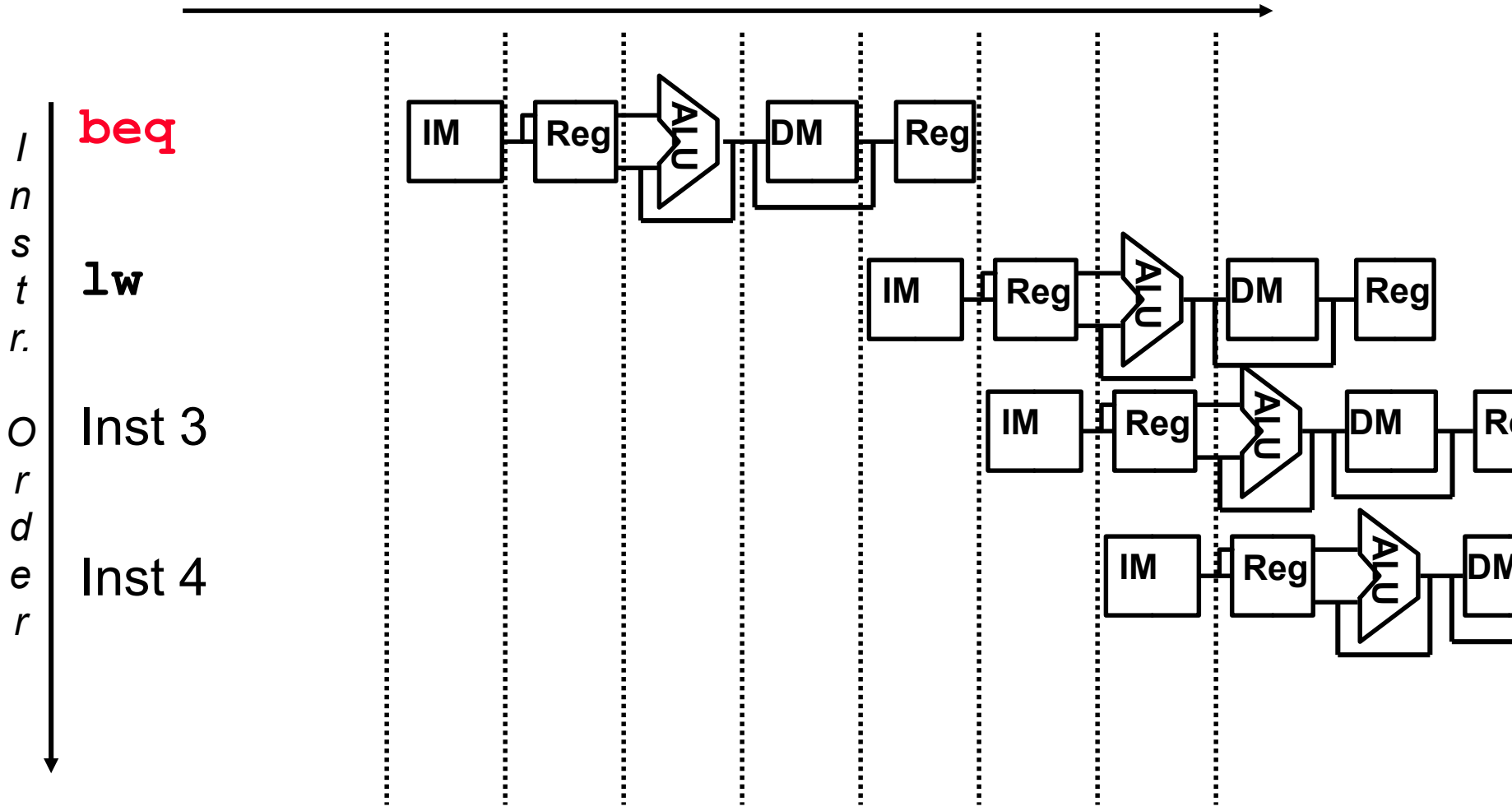
Branch Instructions Cause Control Hazards

- Dependencies backward in time cause hazards



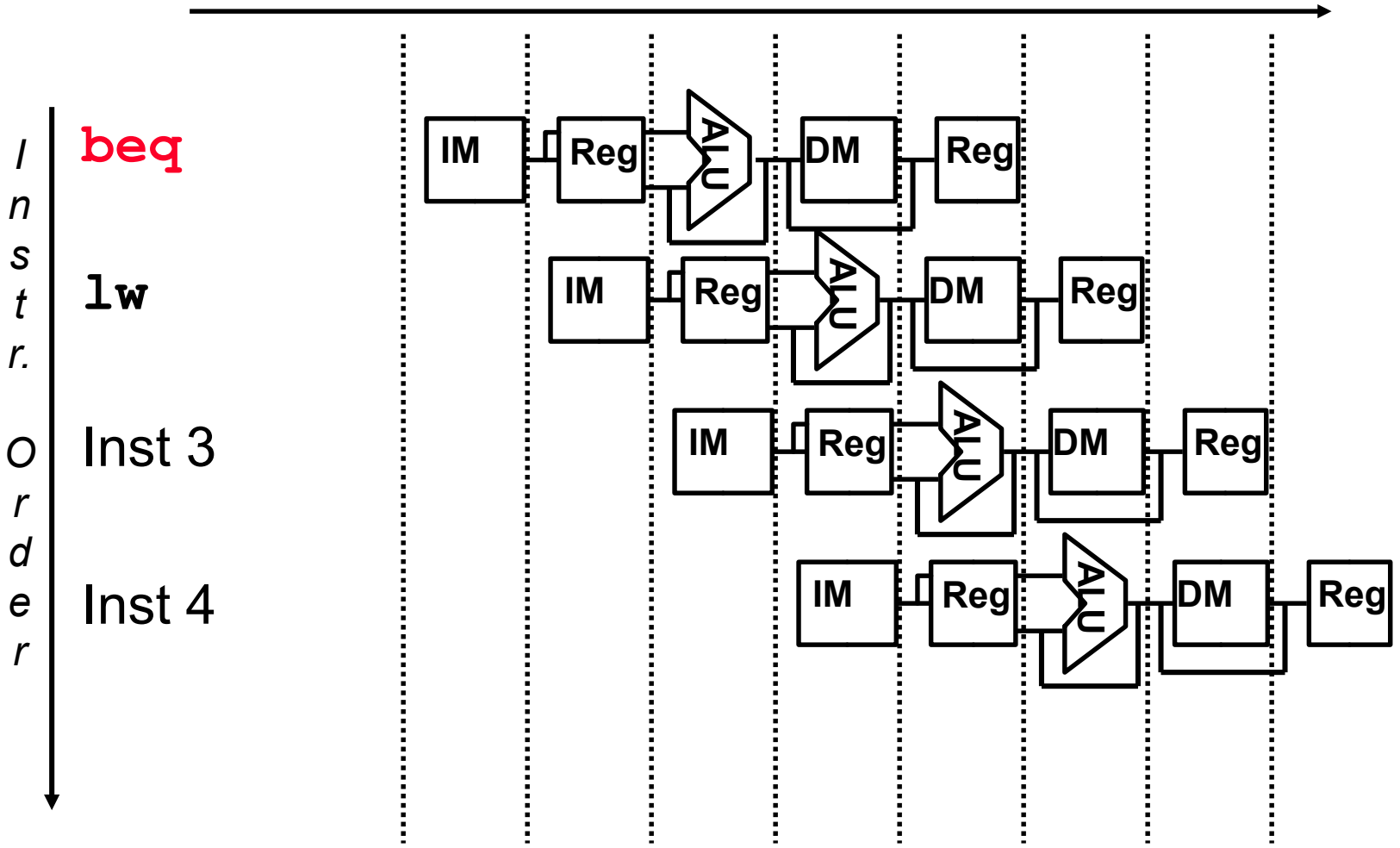
Solution 1: next instruction stalls for three cycles

- Waiting for three cycles for ALU's output, …

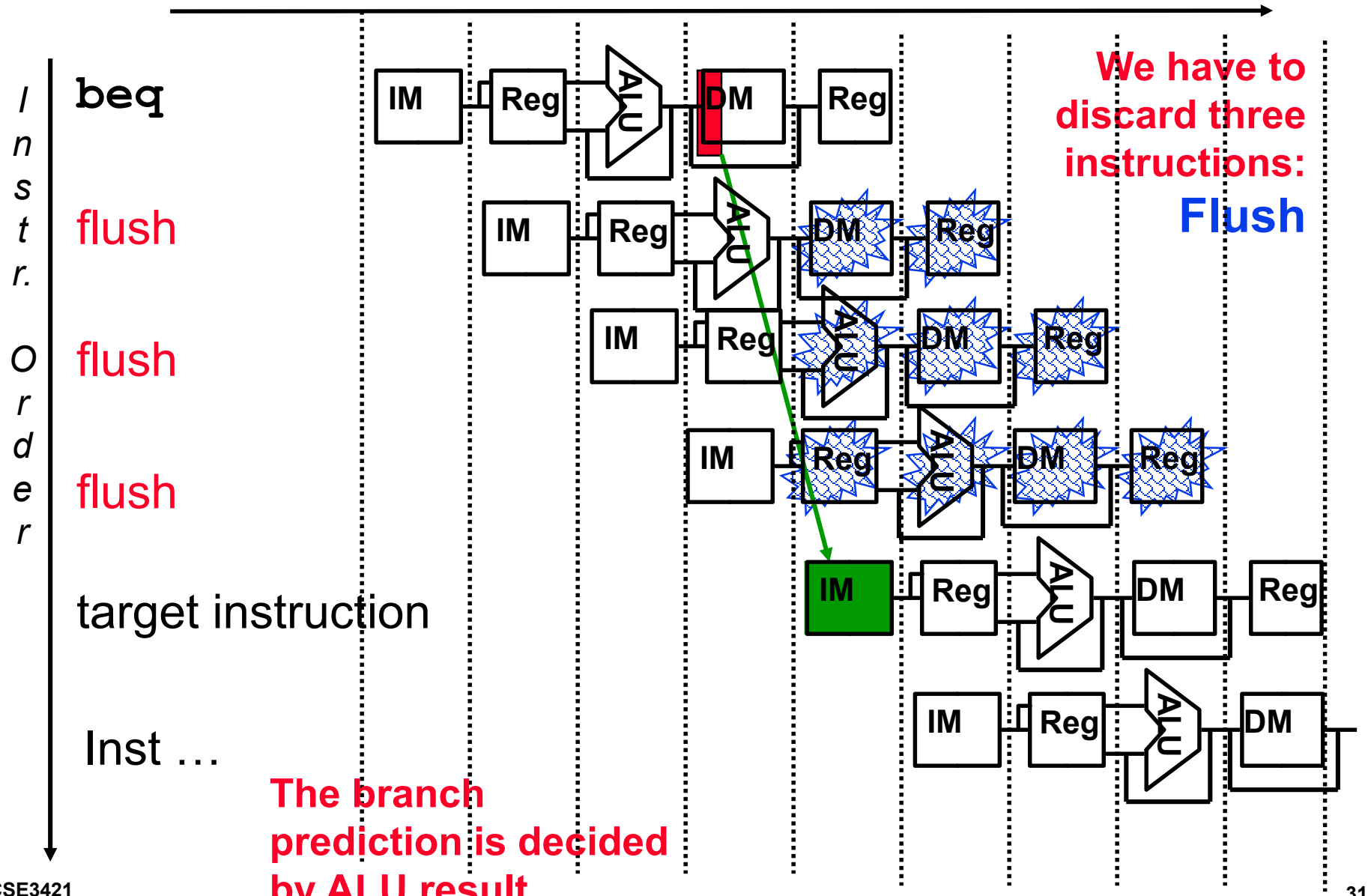


Solution 2: Branch prediction - not taken

- If the prediction is right, pipeline execution continues

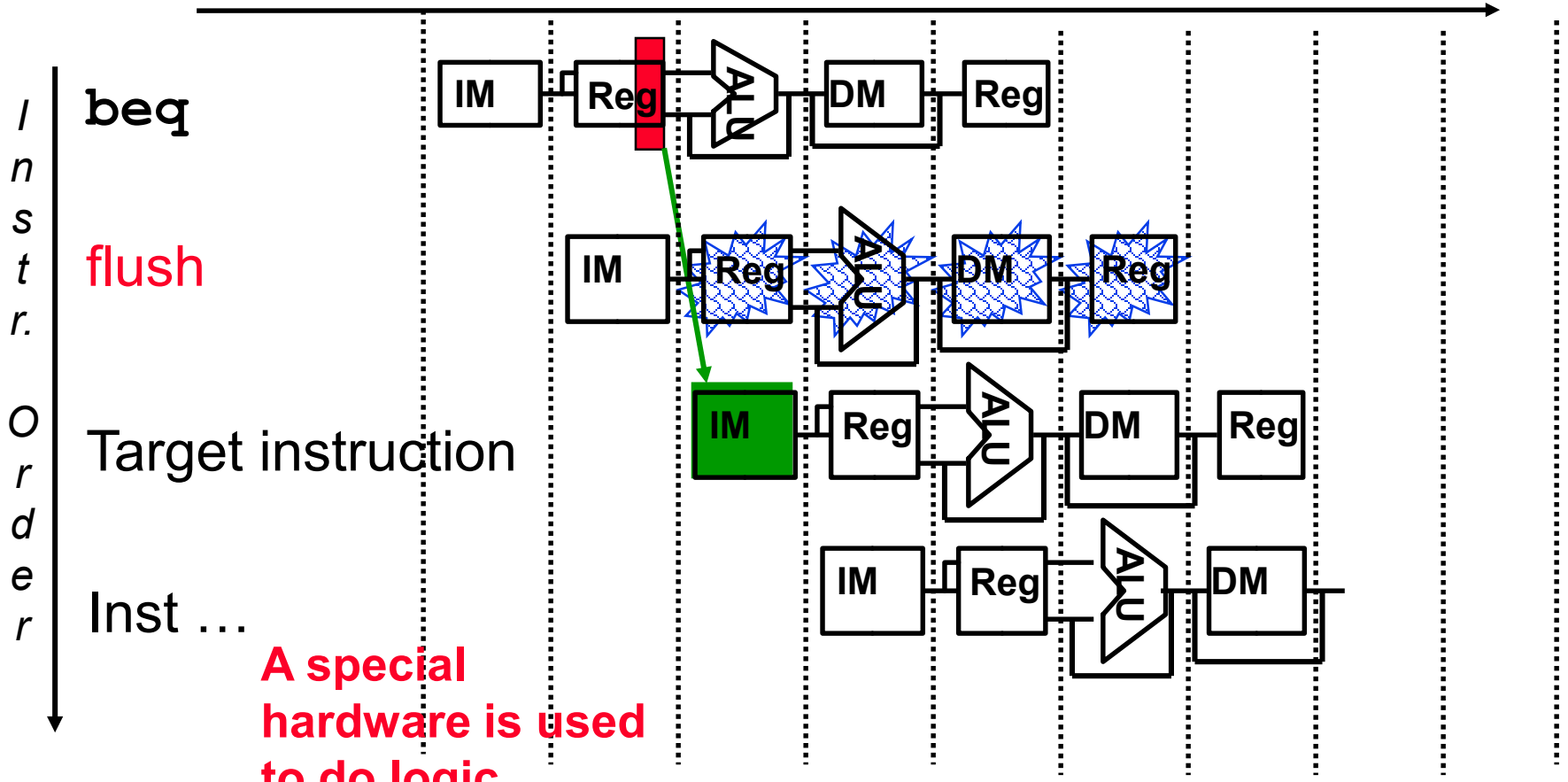


Branch misprediction



A special ALU for comparison: make flush quickly

- Move branch decision hardware back to as early in the pipeline as possible – i.e., during the decode cycle, where branch is known, quickly make a comparison to reduce the lost from misprediction

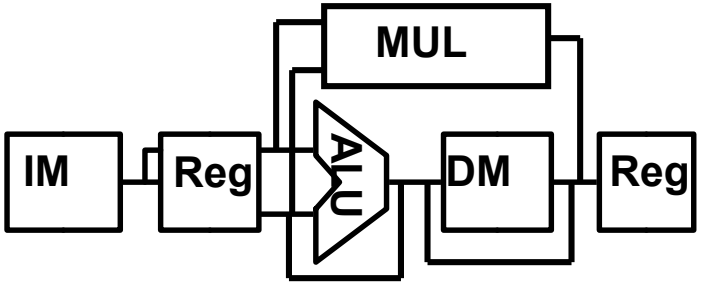


Performance gains and losses

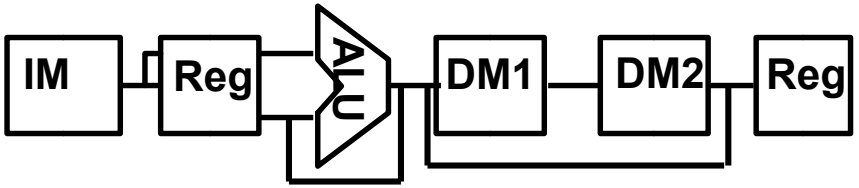
- Experiments show prediction improves performance
  - Misprediction means 2-3 cycles are lost
  - Correct prediction makes the pipeline execute in full speed
- All modern CPUs use branch prediction
  - Many dynamic methods based on research
- The pipeline structure also has big impact on branch prediction
  - Longer pipeline may require more instructions to be flushed for a misprediction

Other Pipeline Structures: Flexible Stage Length

- What about the (slow) multiply operation?
  - let it take two cycles (since it doesn't use the DM stage)

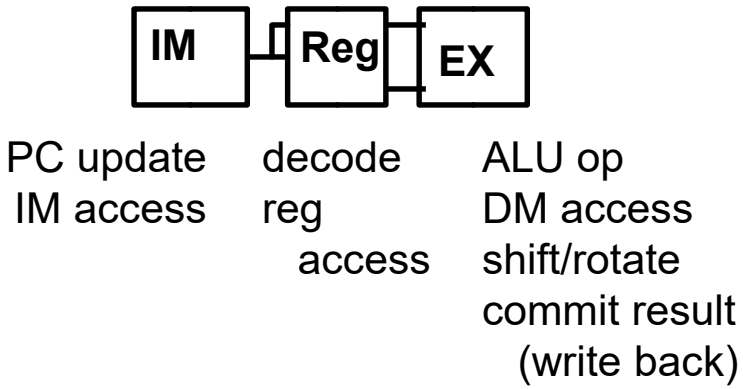


- What if the data memory access is twice as slow as the instruction memory?
  - let data memory access take two cycles (and keep the same clock rate)

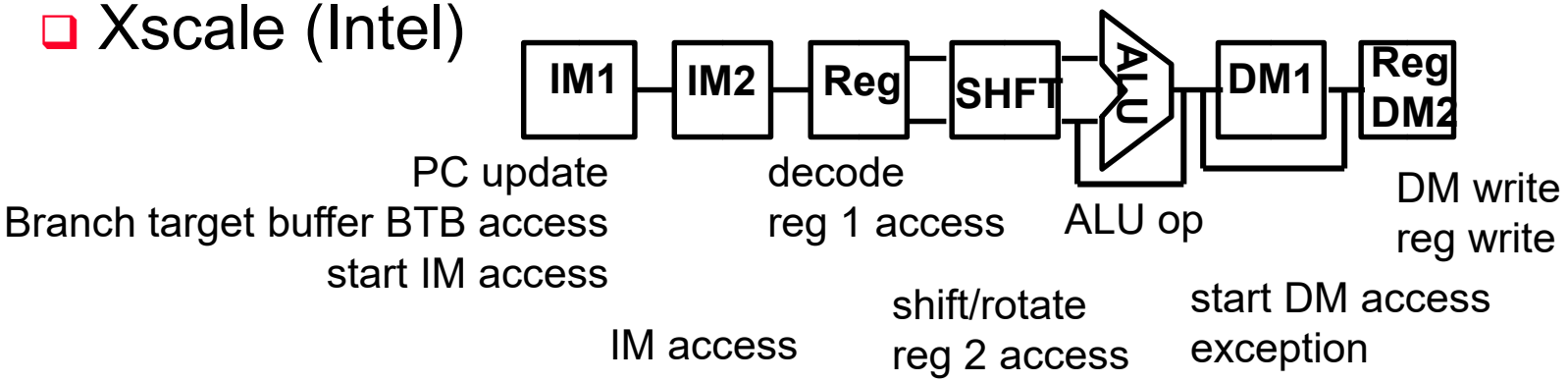


Other Sample Pipeline Alternatives

- ARM7



- Xscale (Intel)



Summary

- All modern processors use pipelining (cycle time reduced rapidly under Moore's Law until 5 years ago).
- Pipelining doesn't help latency of single task, it helps throughput of entire workload
- Potential speedup: a CPI of 1 and fast a CC (cycle time)
- Pipeline rate limited by slowest pipeline stage
  - Unbalanced pipe stages makes for inefficiencies
  - The time to "fill" pipeline and time to "drain" it as hazards are major impact on the performance of pipelines
- Must detect and resolve hazards
  - Stalling negatively affects CPI (makes CPI bigger than ideal 1)