CSE 431
Computer Architecture
Fall 2005

Lecture 06: Basic MIPS Pipelining Review

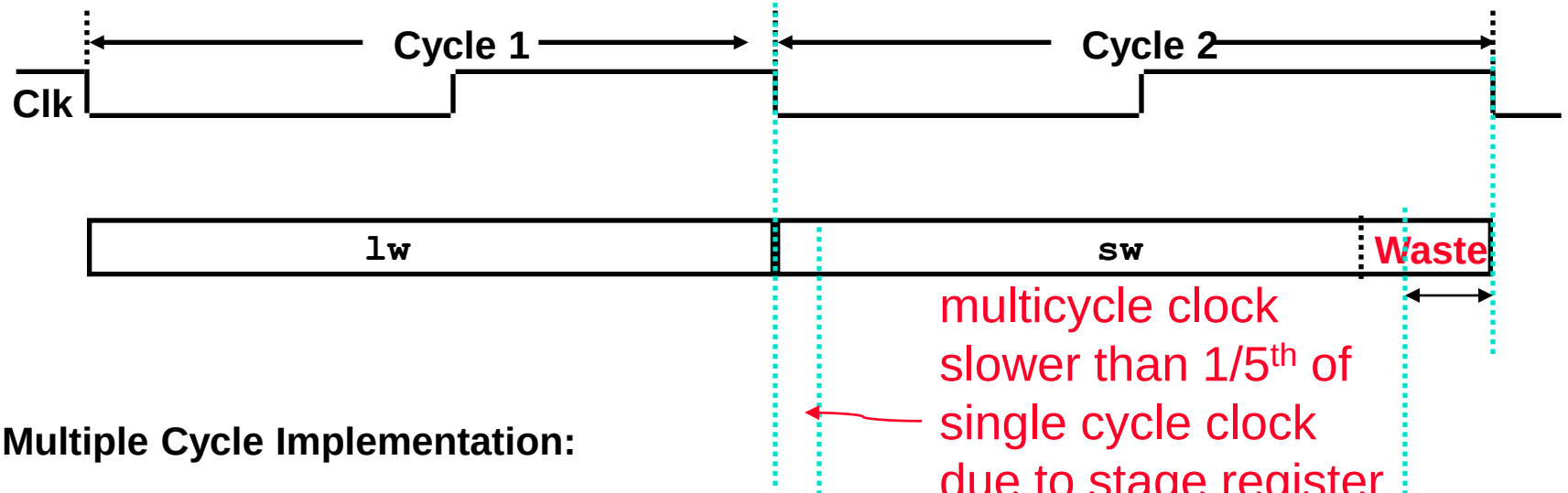
Mary Jane Irwin (www.cse.psu.edu/~mji)

www.cse.psu.edu/~cg431

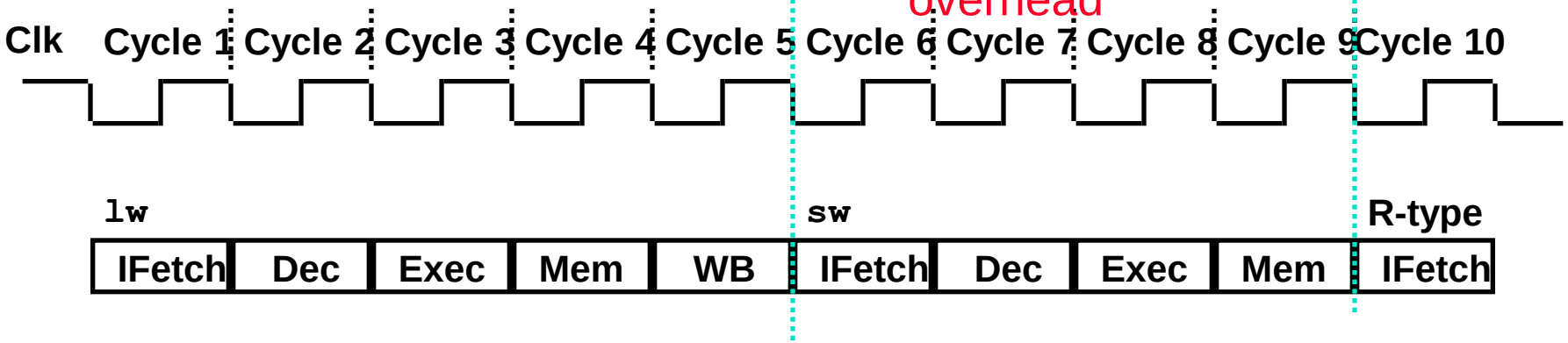
[Adapted from *Computer Organization and Design*,
Patterson & Hennessy, © 2005, UCB]

Review: Single Cycle vs. Multiple Cycle Timing

Single Cycle Implementation:



Multiple Cycle Implementation:



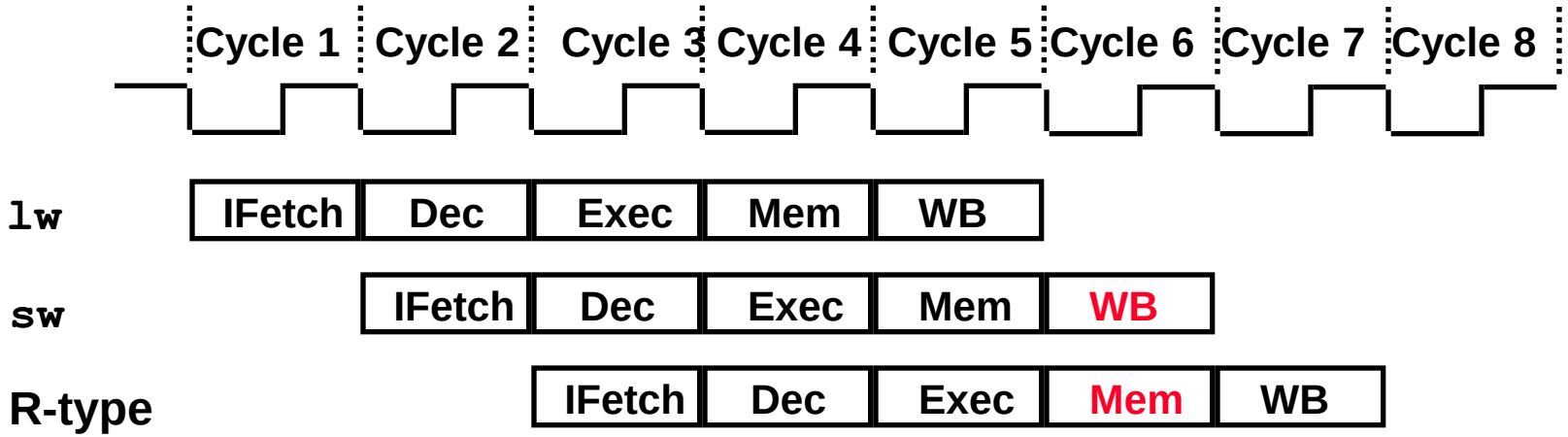
How Can We Make It Even Faster?

- Split the multiple instruction cycle into smaller and smaller steps
  - There is a point of diminishing returns where as much time is spent loading the state registers as doing the work
- Start fetching and executing the next instruction before the current one has completed
  - Pipelining – (all?) modern processors are pipelined for performance
  - Remember *the* performance equation:

    CPU time = CPI * CC * IC
- Fetch (and execute) more than one instruction at a time
  - Superscalar processing – stay tuned

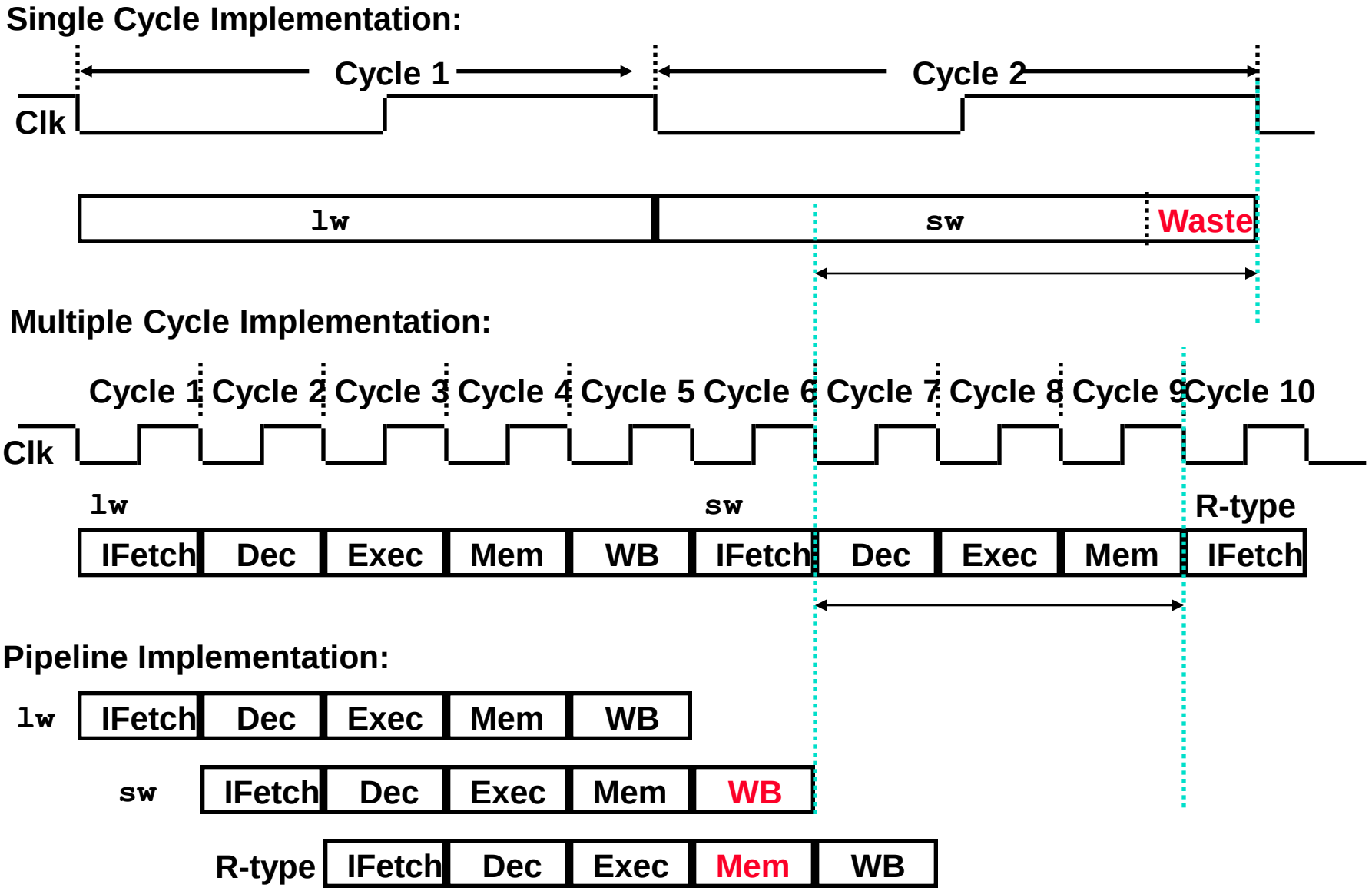
A Pipelined MIPS Processor

- 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



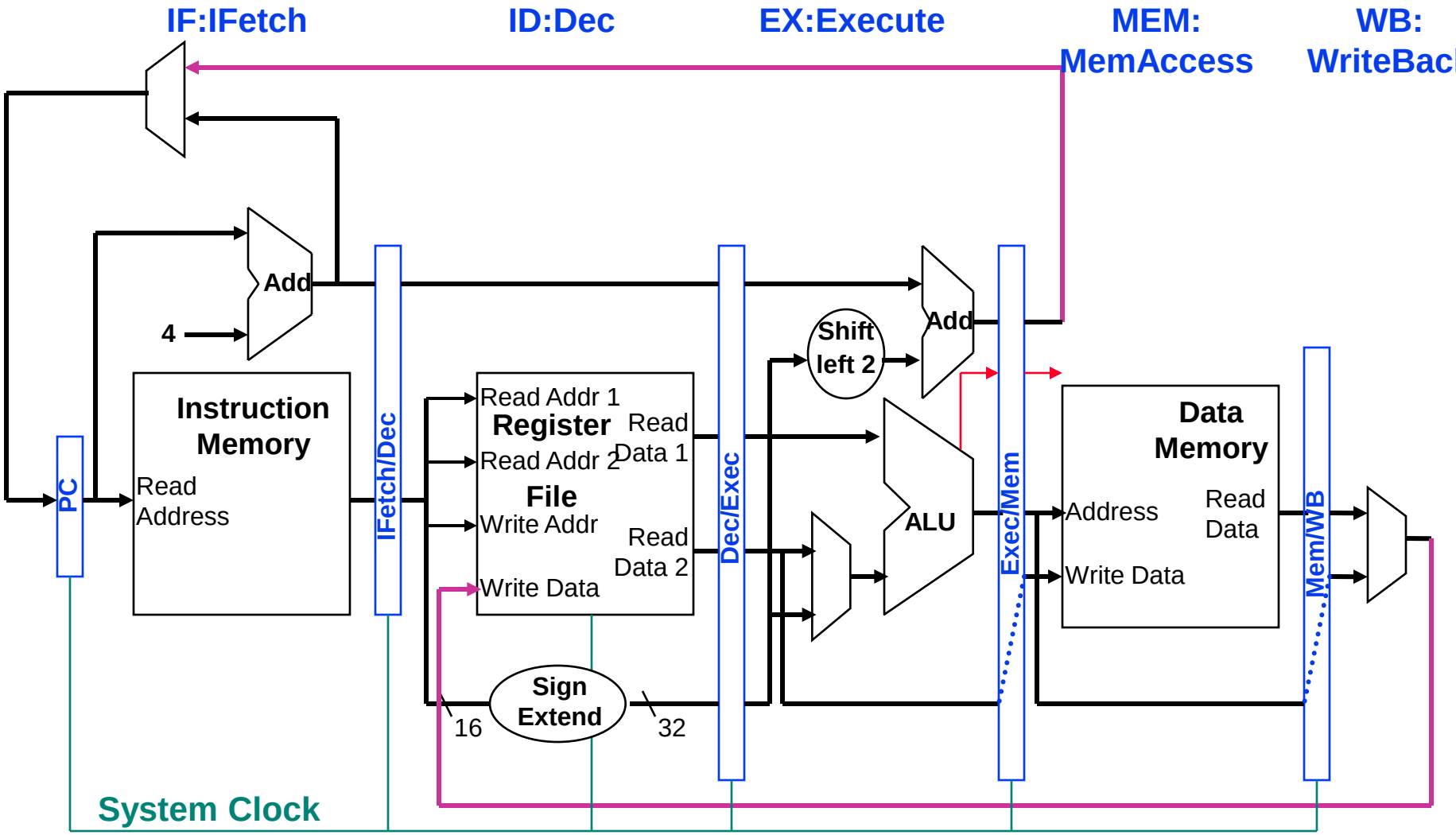
  - clock cycle (pipeline stage time) is limited by the slowest stage
  - for some instructions, some stages are wasted cycles

Single Cycle, Multiple Cycle, vs. Pipeline



MIPS Pipeline Datapath Modifications

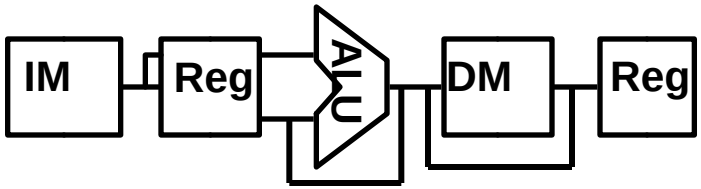
- What do we need to add/modify in our MIPS datapath?
  - State registers between each pipeline stage to isolate them



Pipelining the MIPS ISA

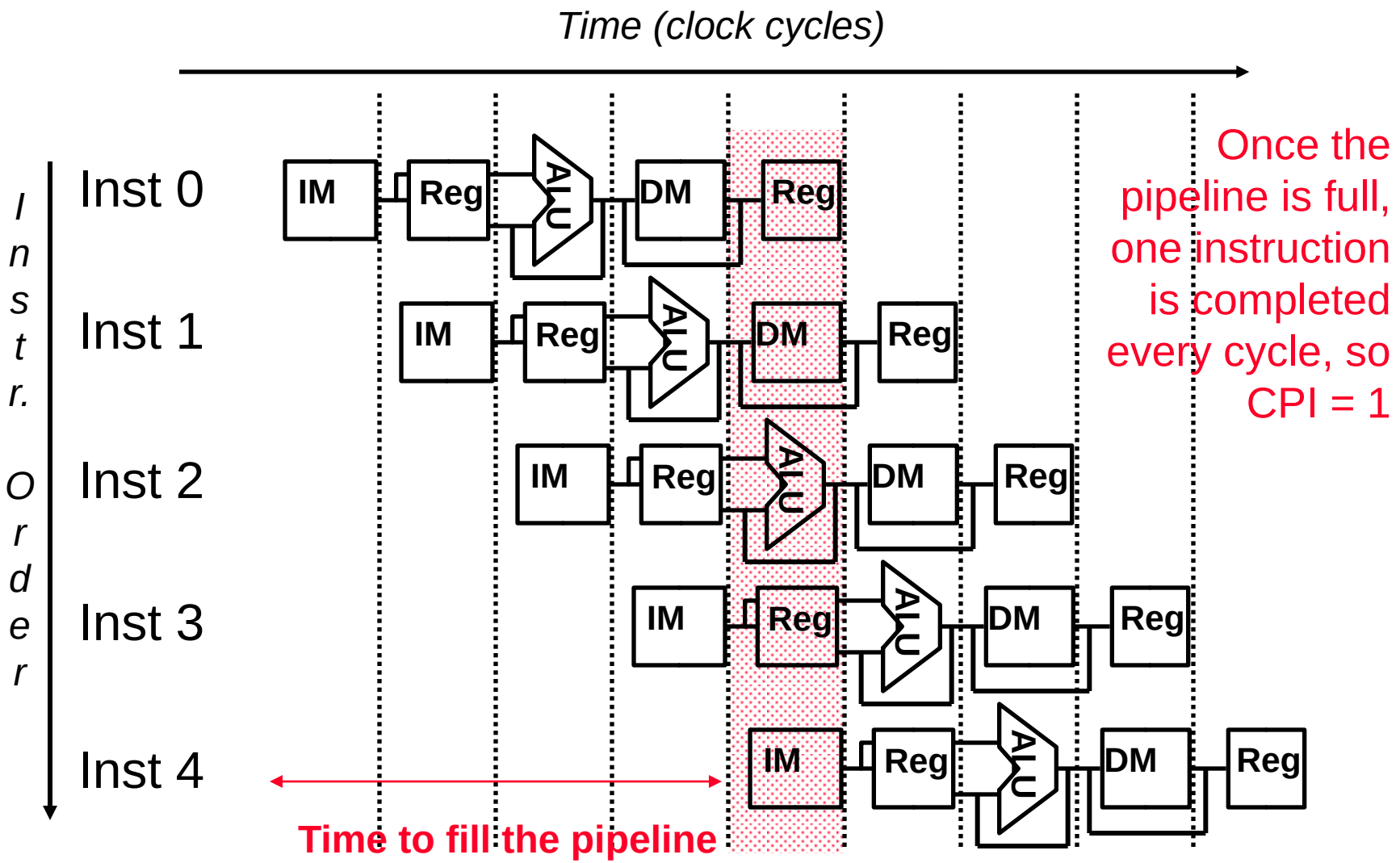
- What makes it easy
  - all instructions are the same length (32 bits)
    - can fetch in the 1st stage and decode in the 2nd stage
  - few instruction formats (three) with symmetry across formats
    - can begin reading register file in 2nd stage
  - memory operations can occur only in loads and stores
    - can use the execute stage to calculate memory addresses
  - each MIPS instruction writes at most one result (i.e., changes the machine state) and does so near the end of the pipeline (MEM and WB)
- What makes it hard
  - structural hazards: what if we had only one memory?
  - control hazards: what about branches?
  - data hazards: what if an instruction's input operands depend on the output of a previous instruction?

Graphically Representing MIPS Pipeline



- 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!

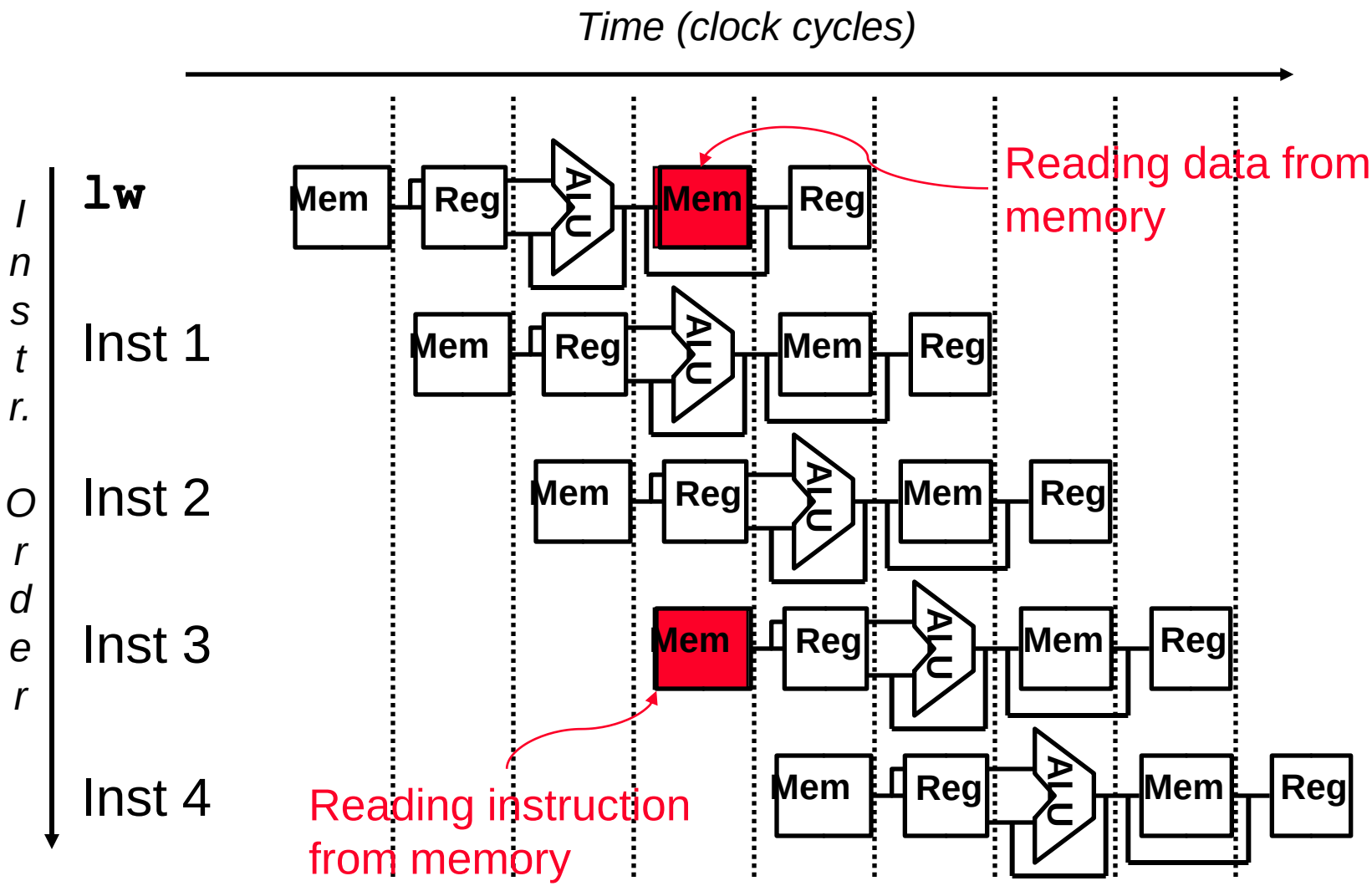


Can Pipelining Get Us Into Trouble?

- 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 instructions

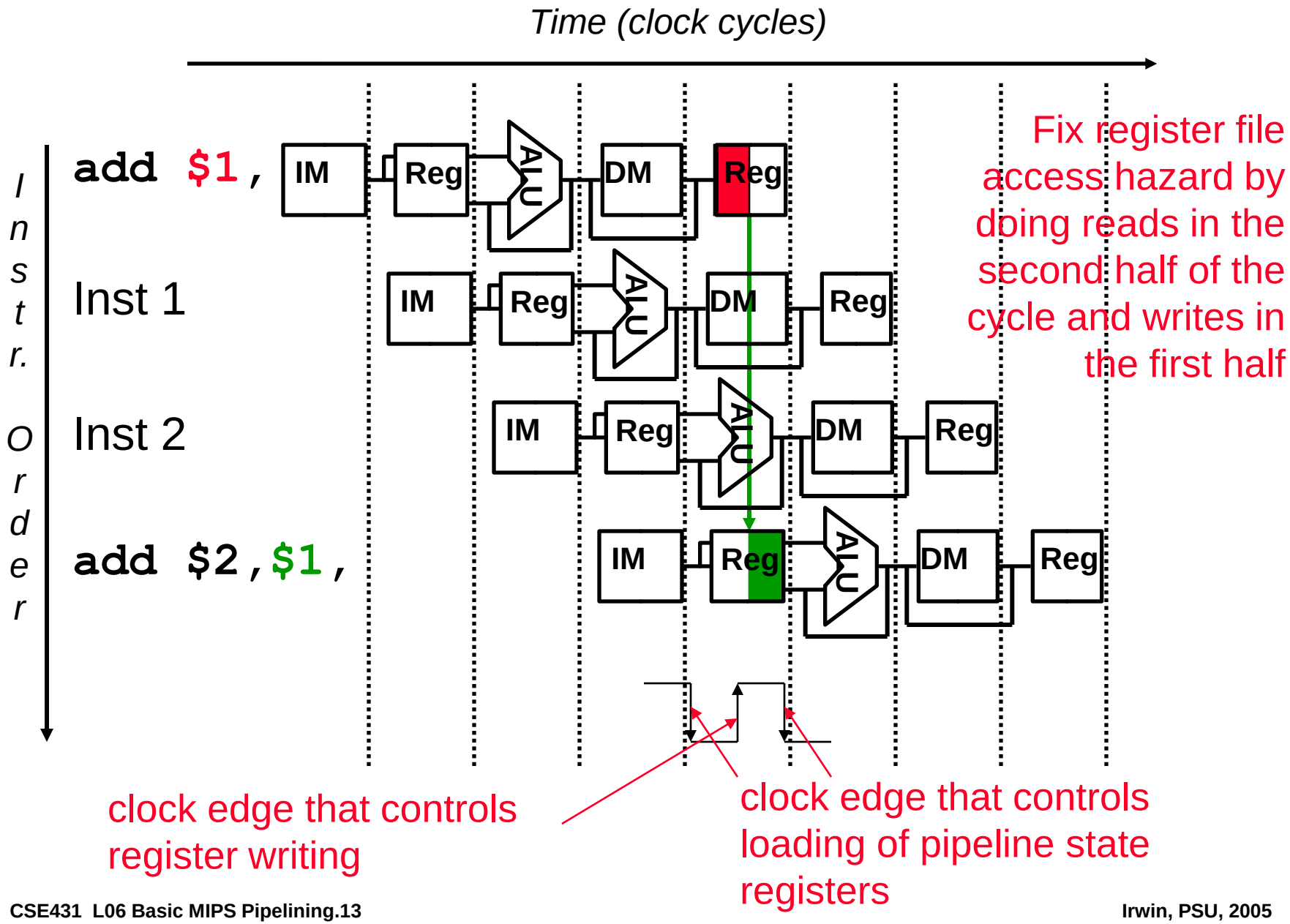
- Can always resolve hazards by waiting
  - pipeline control must detect the hazard
  - and take action to resolve hazards

A Single Memory Would Be a Structural Hazard



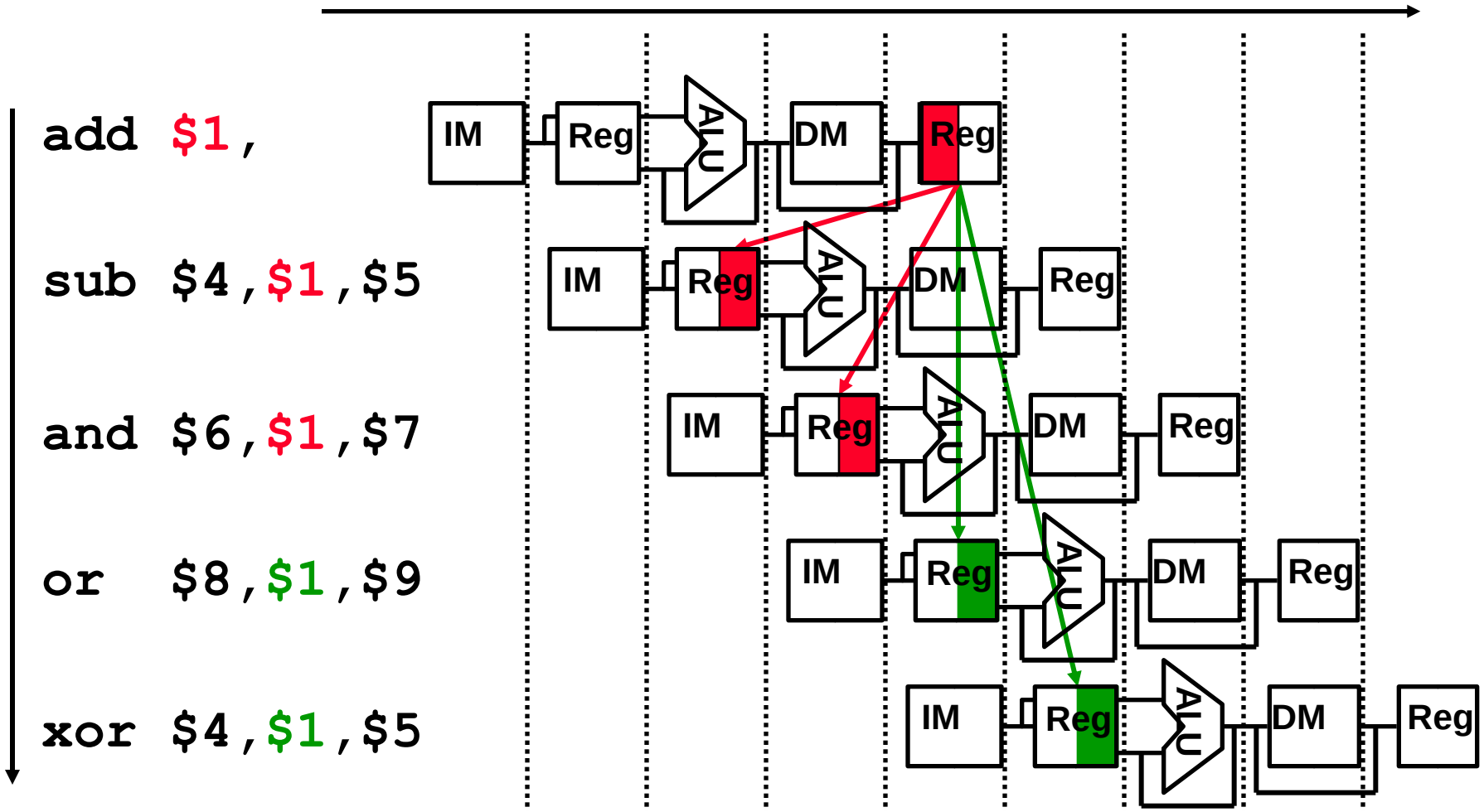
- Fix with separate instr and data memories (I$ and D$)

How About Register File Access?



Register Usage Can Cause Data Hazards

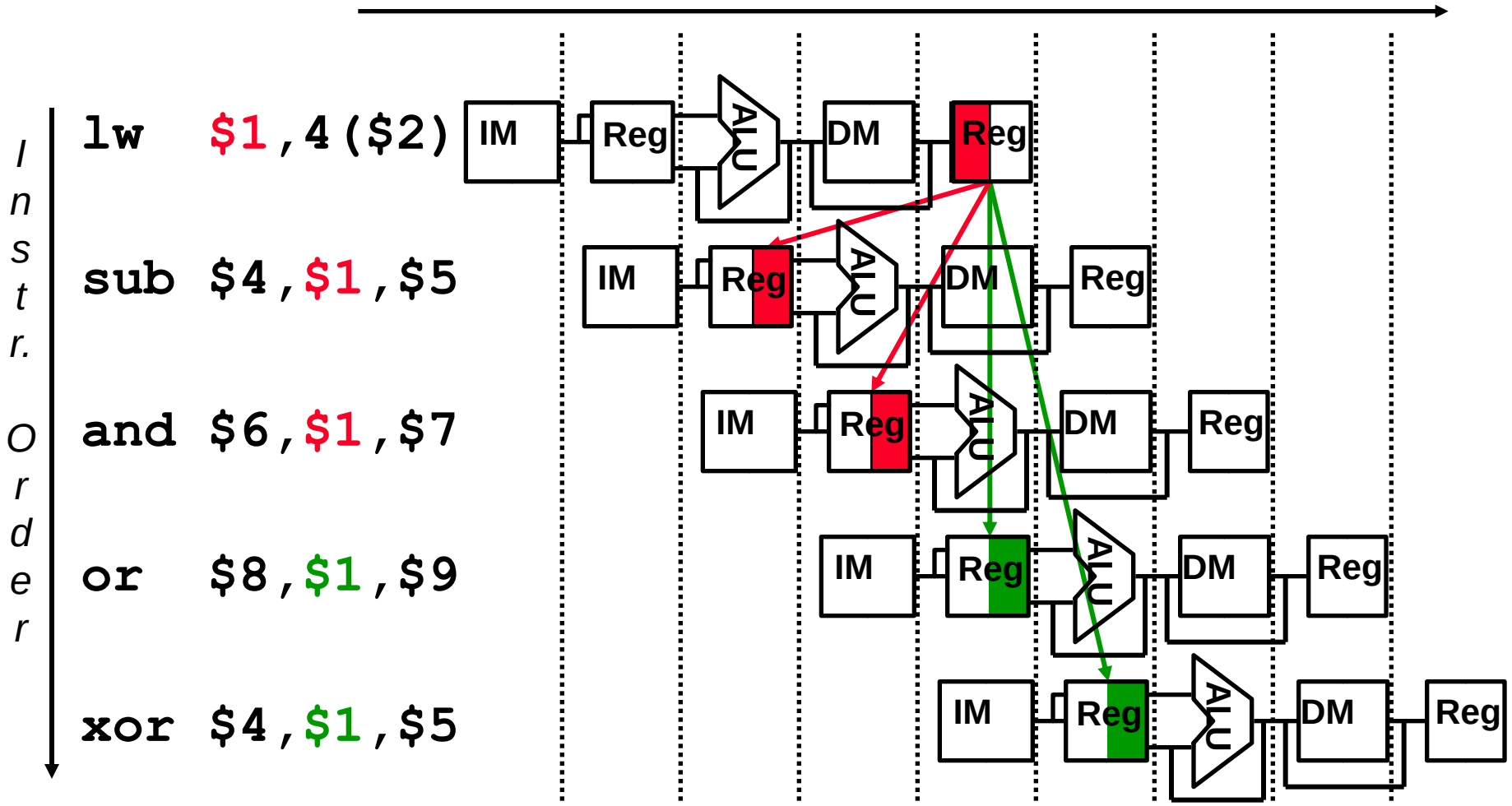
- Dependencies backward in time cause hazards



- Read before write data hazard

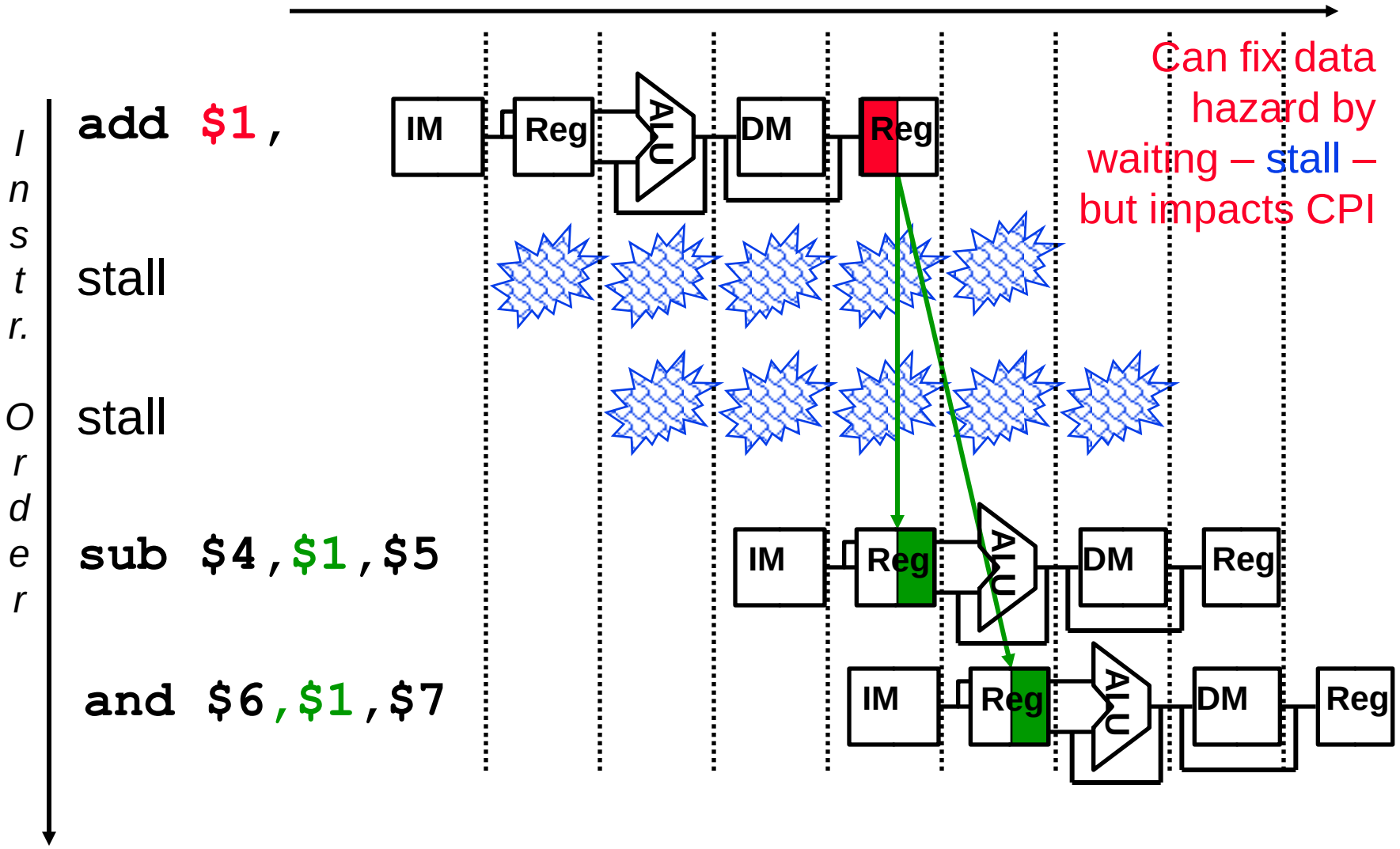
Loads Can Cause Data Hazards

- Dependencies backward in time cause hazards

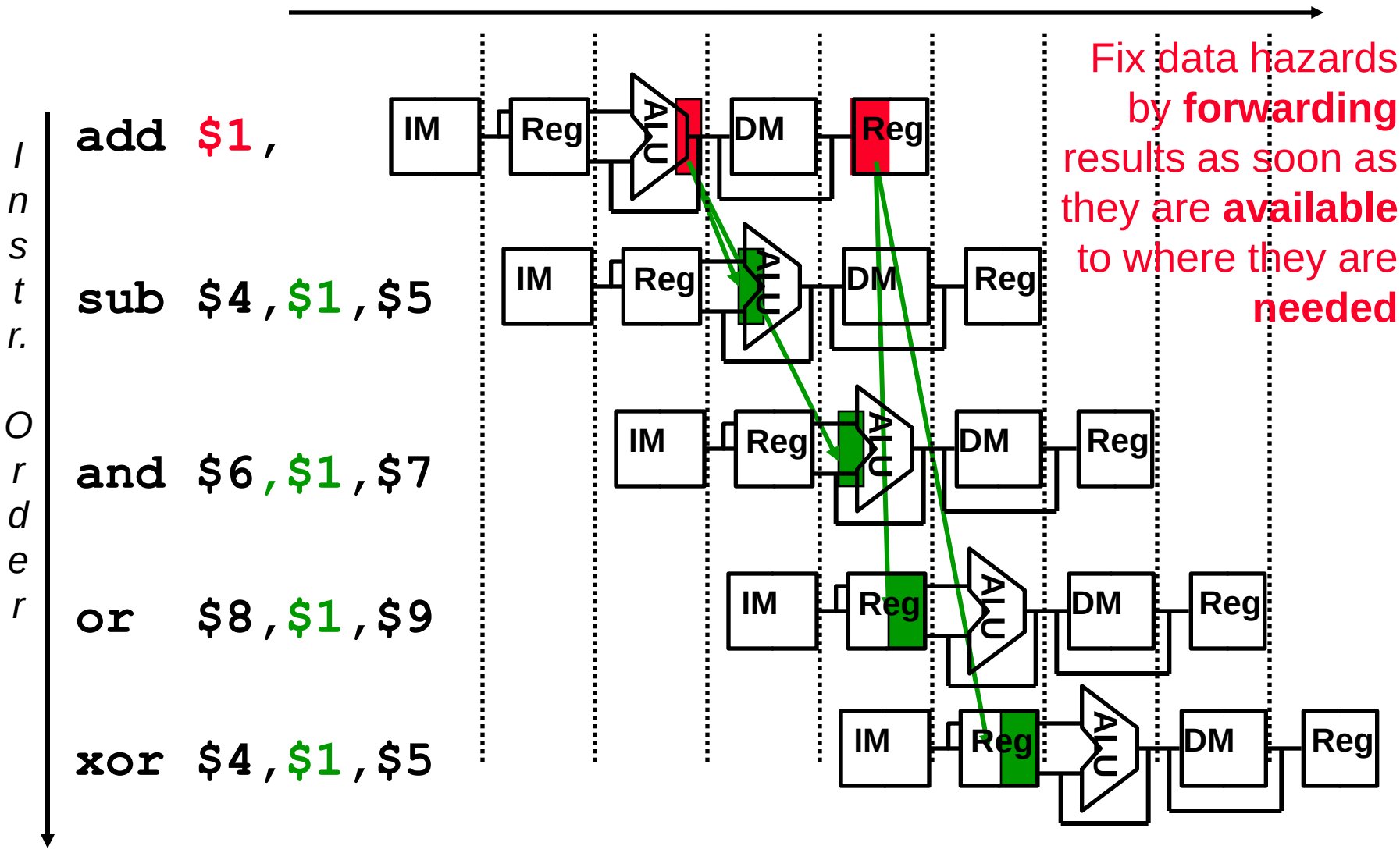


- Load-use data hazard

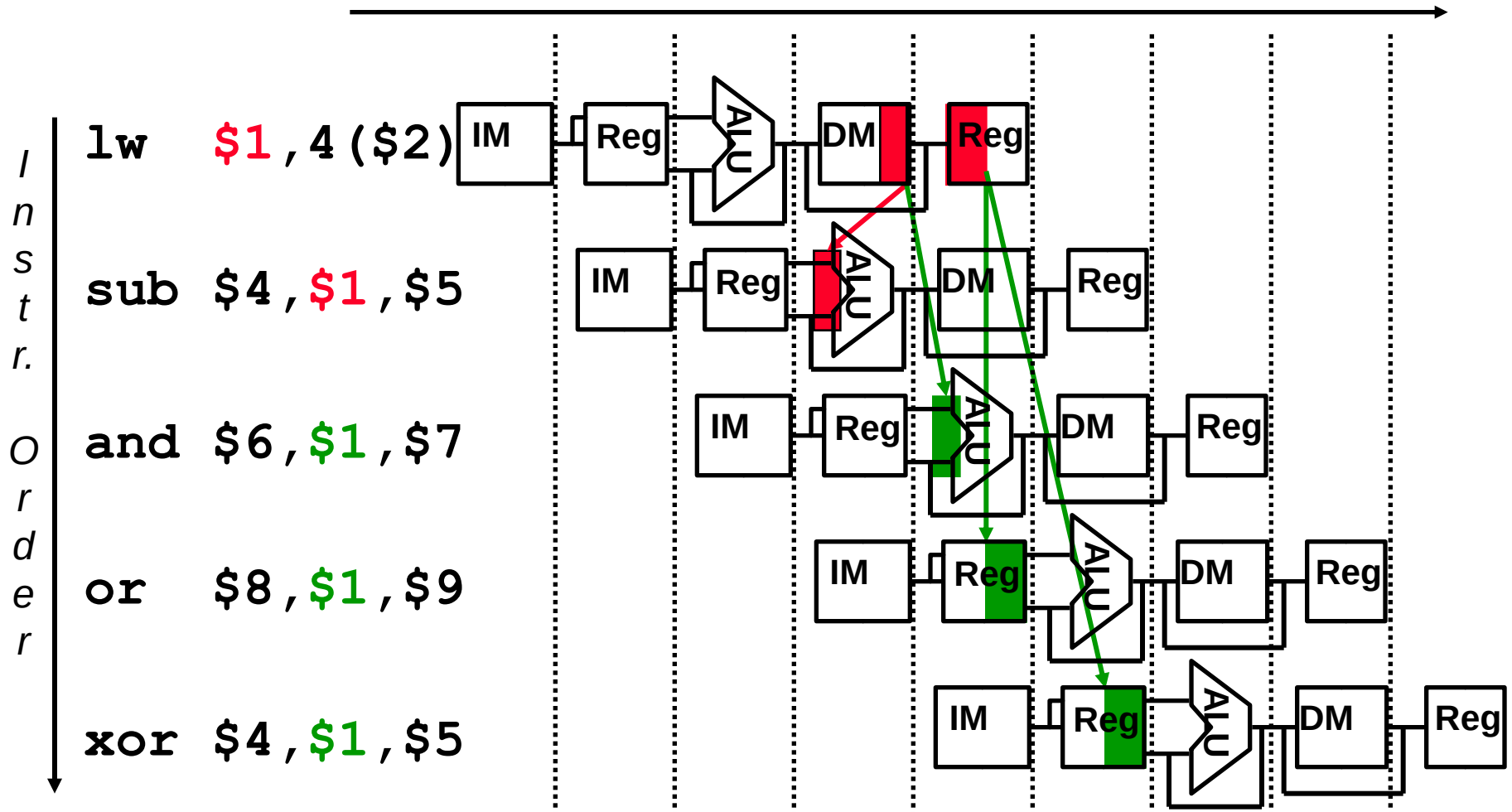
One Way to "Fix" a Data Hazard



Another Way to "Fix" a Data Hazard



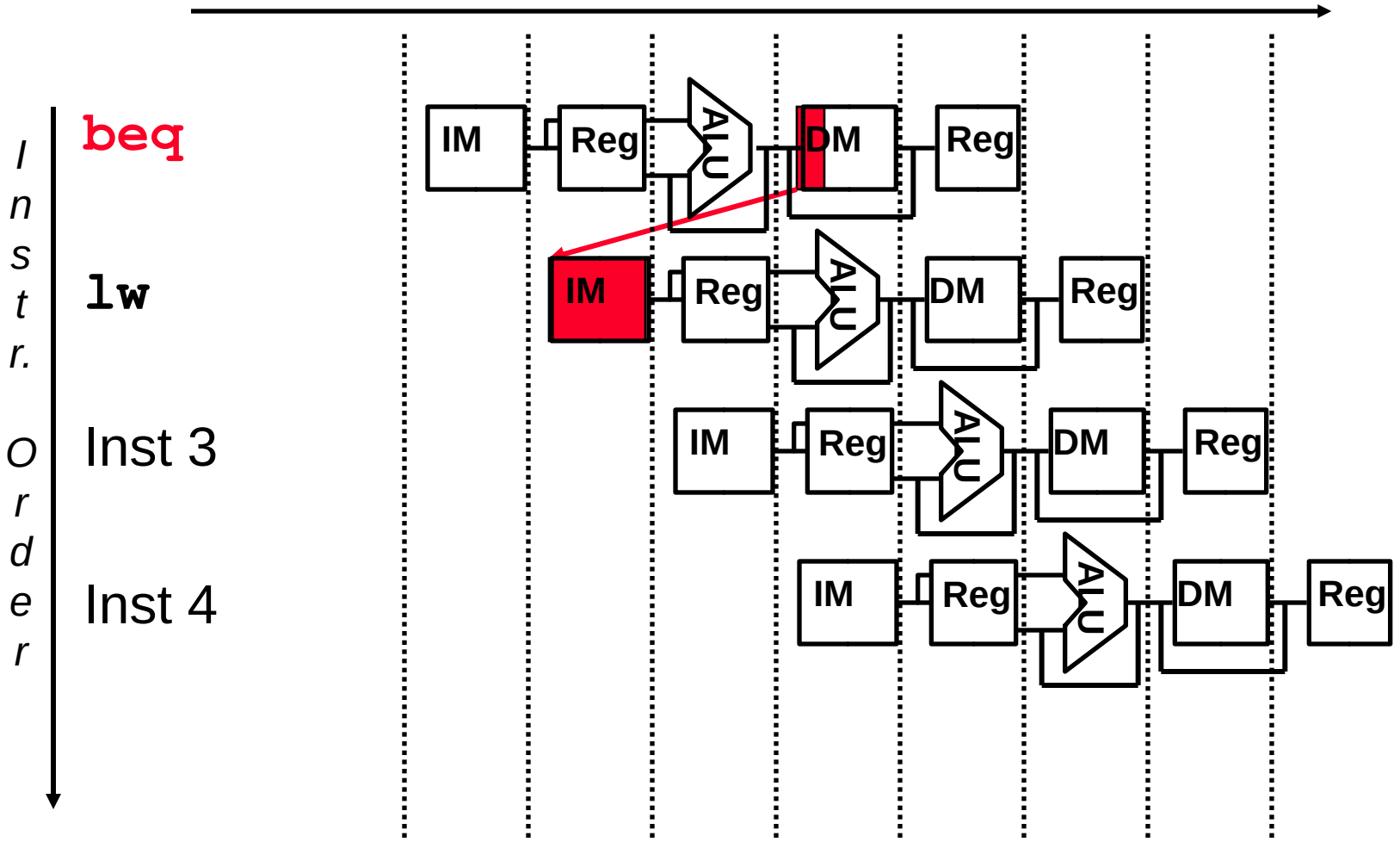
Forwarding with Load-use Data Hazards



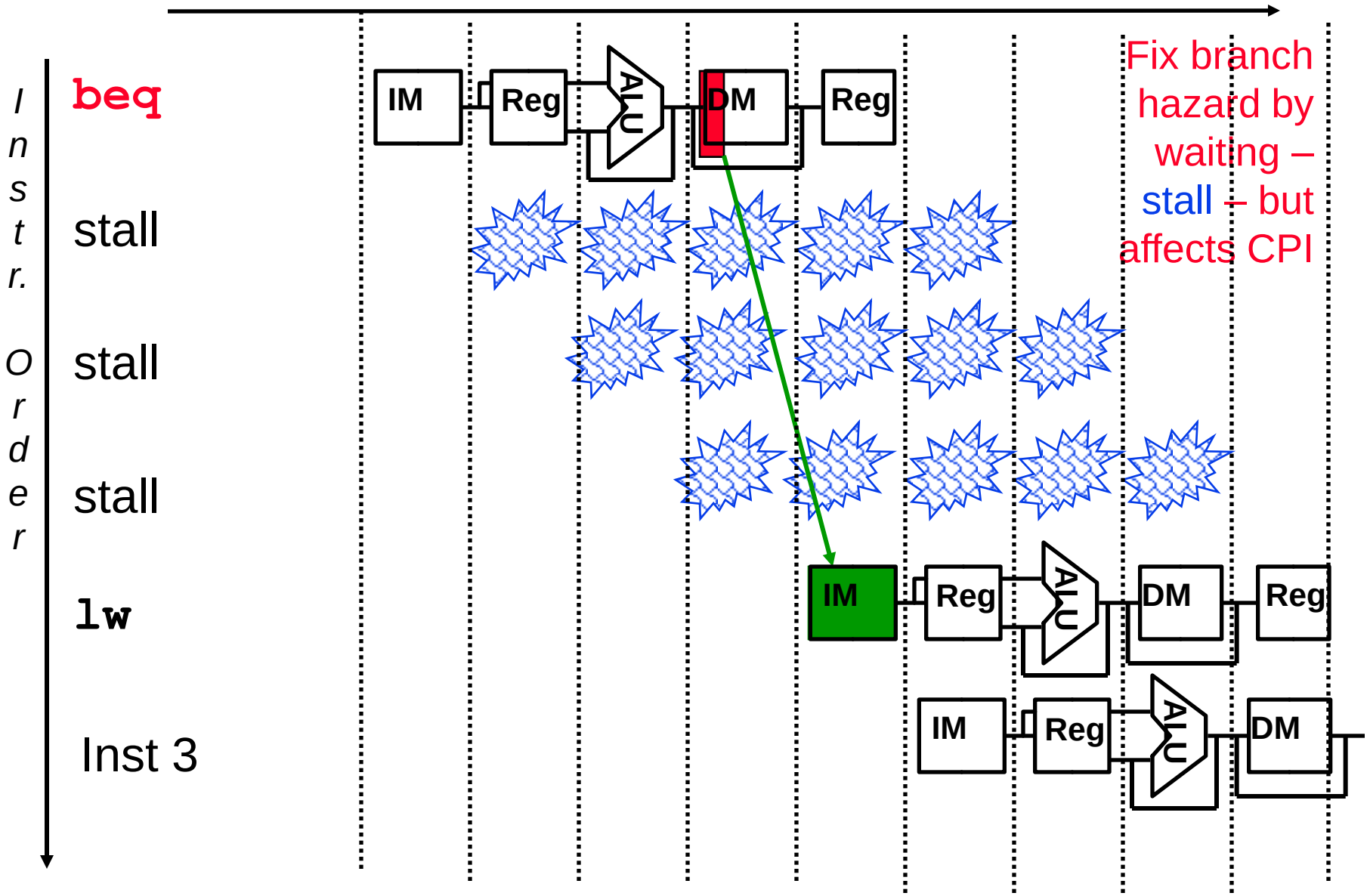
- Will still need one stall cycle even with forwarding

Branch Instructions Cause Control Hazards

- Dependencies backward in time cause hazards

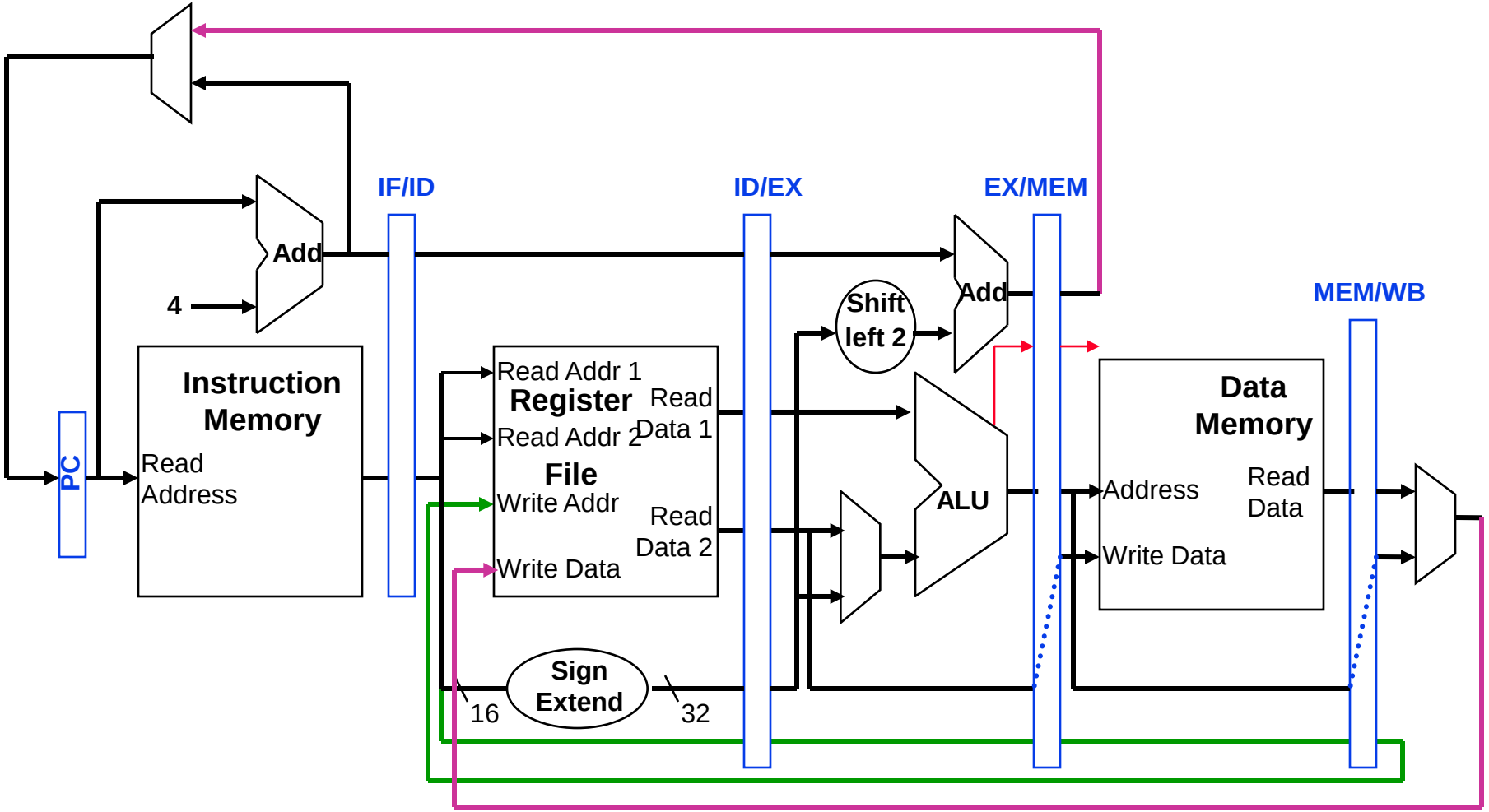


One Way to "Fix" a Control Hazard



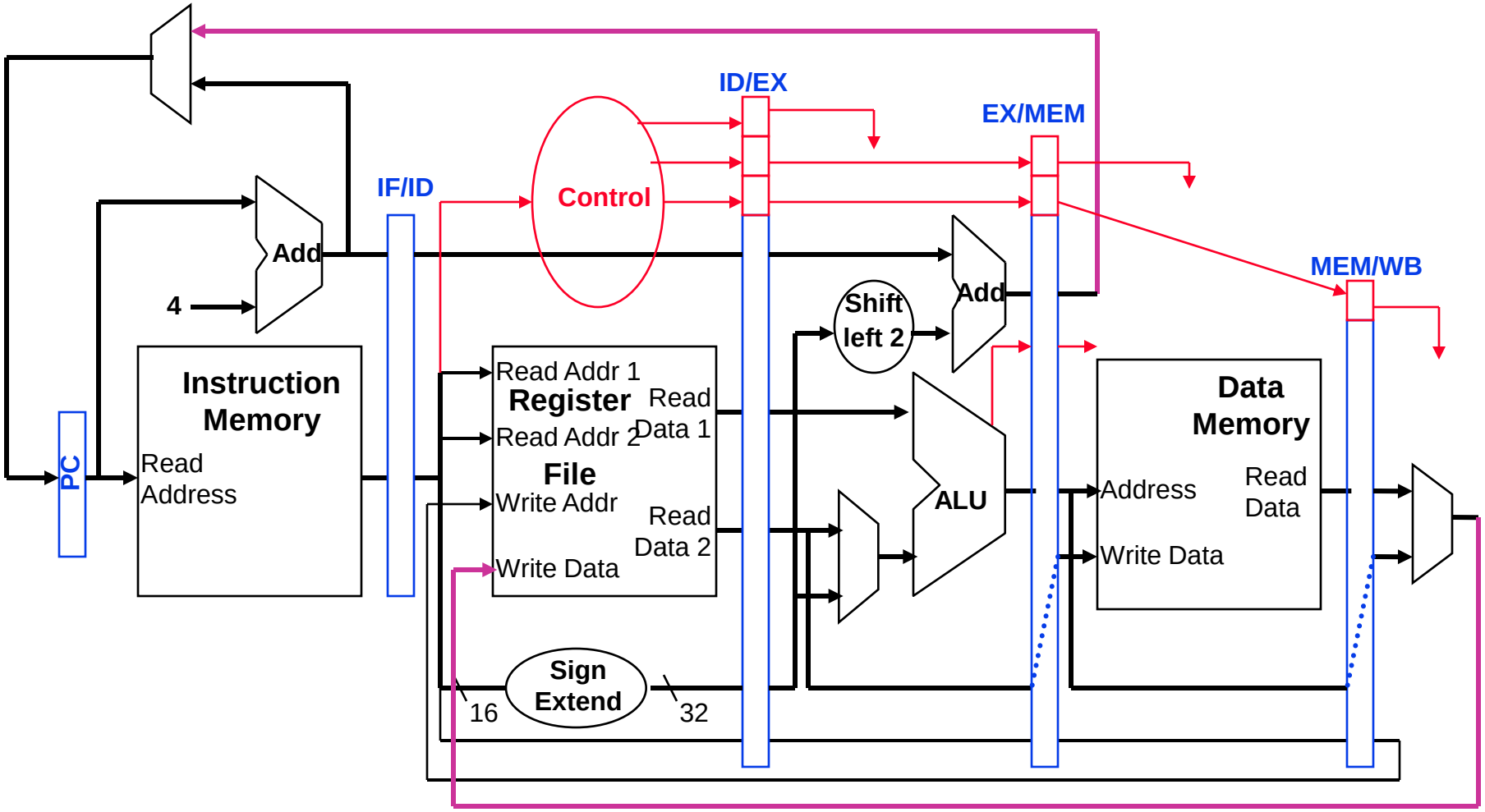
Corrected Datapath to Save RegWrite Addr

- Need to preserve the destination register address in the pipeline state registers



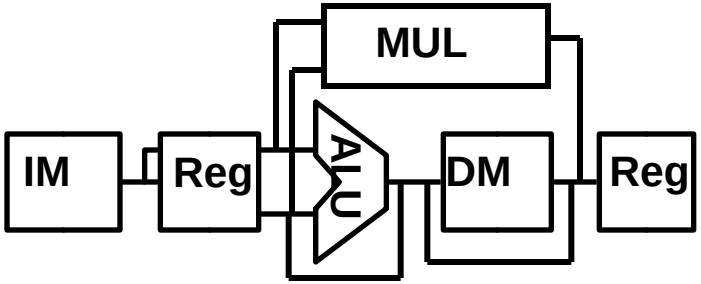
MIPS Pipeline Control Path Modifications

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

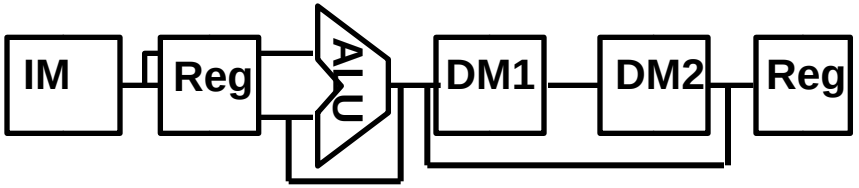


Other Pipeline Structures Are Possible

- What about the (slow) multiply operation?
  - Make the clock twice as slow or …
  - 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?
  - make the clock twice as slow or …
  - let data memory access take two cycles (and keep the same clock rate)



Sample Pipeline Alternatives

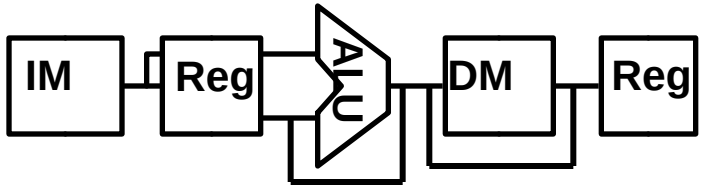
- ARM7

PC update
IM access

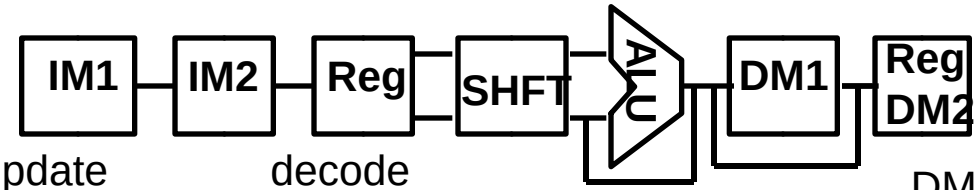
decode
reg
access

ALU op
DM access
shift/rotate
commit result
(write back)

- StrongARM-1



- XScale



PC update
BTB access
start IM access

IM access

decode
reg 1 access

shift/rotate
reg 2 access

ALU op

start DM access
exception

DM write
reg write

Summary

- All modern day processors use pipelining
- Pipelining doesn't help latency of single task, it helps throughput of entire workload
- Potential speedup: a CPI of 1 and fast a CC
- Pipeline rate limited by slowest pipeline stage
  - Unbalanced pipe stages makes for inefficiencies
  - The time to "fill" pipeline and time to "drain" it can impact speedup for deep pipelines and short code runs
- Must detect and resolve hazards
  - Stalling negatively affects CPI (makes CPI less than the ideal of 1)

Next Lecture and Reminders

- Next lecture
  - Overcoming data hazards
    - Reading assignment – PH, Chapter 6.4-6.5

- Reminders
  - HW2 due September 29th
  - SimpleScalar tutorials scheduled
    - Thursday, Sept 22, 5:30-6:30 pm in 218 IST
  - Evening midterm exam scheduled
    - Tuesday, October 18th , 20:15 to 22:15, Location 113 IST
    - You should have let me know by now if you have a conflict