EE 3613: Computer Organization
Chapter 4: Pipelining - 1

Avinash Karanth
Department of Electrical Engineering & Computer Science
Ohio University, Athens, Ohio 45701
E-mail: karanth@ohio.edu
Website: http://oucsace.cs.ohiou.edu/~avinashk/ee461a.htm
Acknowledgement: Mary J. Irwin, PSU; Srinivasan Ramasubramanian, UofA,

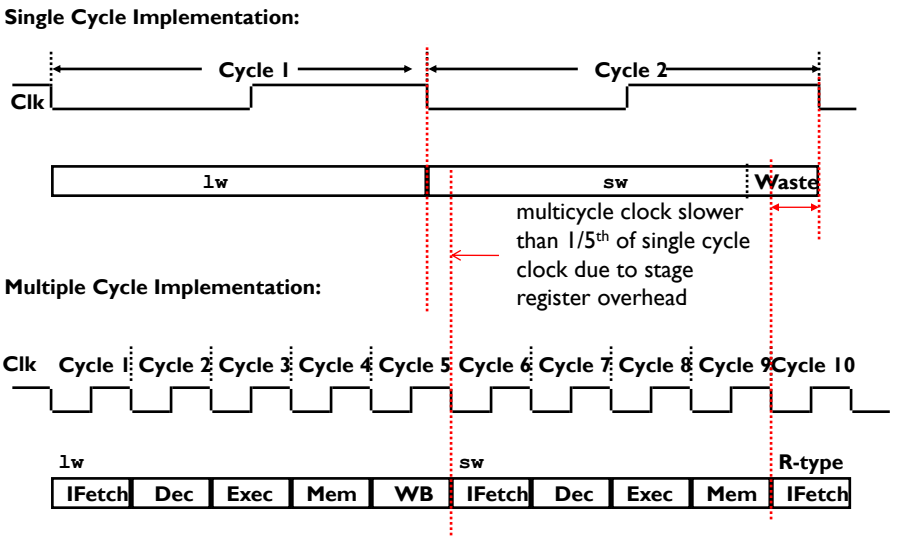
1

Course Administration

- Homework 3B due on Monday Oct 26

2

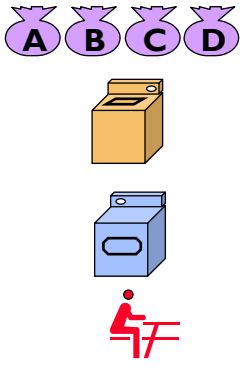
Review: Single Cycle vs. Multiple Cycle Timing



3

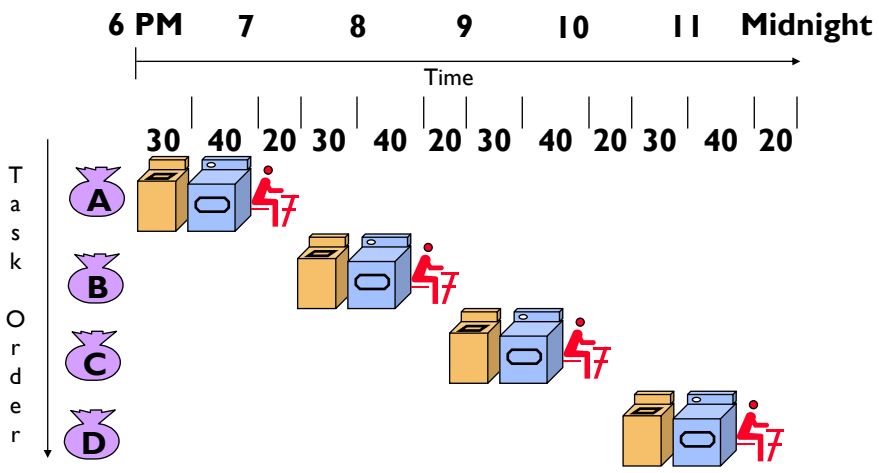
Pipelining: It's Natural!

- Laundry Example
- Ann, Brian, Cathy, Dave each have one load of clothes to wash, dry, and fold
- Washer takes 30 minutes
- Dryer takes 40 minutes
- "Folder" takes 20 minutes



4

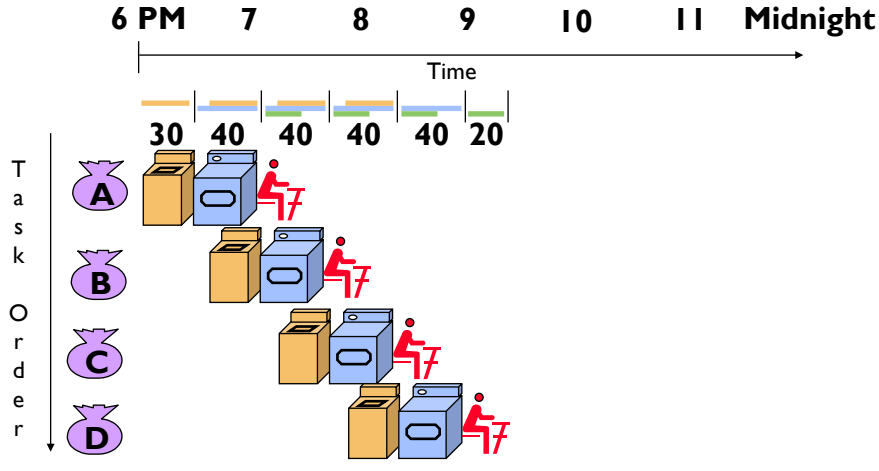
Sequential Laundry



- Sequential laundry takes 6 hours for 4 loads
- If they learned pipelining, how long would laundry take?

5

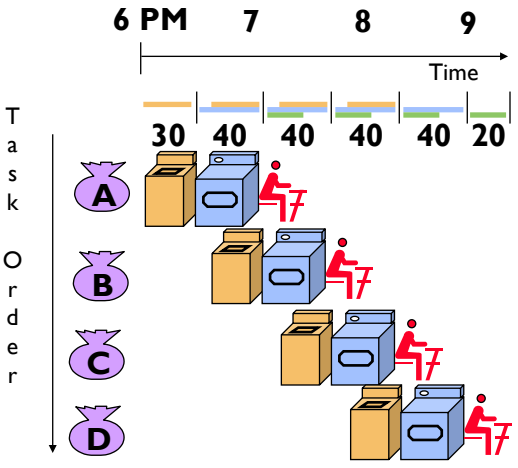
Pipelined Laundry: Start work ASAP



- Pipelined laundry takes 3.5 hours for 4 loads

6

Pipelining Lessons



- Pipelining doesn't help latency of single task, it helps throughput of entire workload
- Pipeline rate limited by slowest pipeline stage
- Multiple tasks operating simultaneously
- Potential speedup = Number pipe stages
- Unbalanced lengths of pipe stages reduces speedup
- Time to "fill" pipeline and time to "drain" it reduces speedup

What is the speedup of a pipeline of n stages?

7

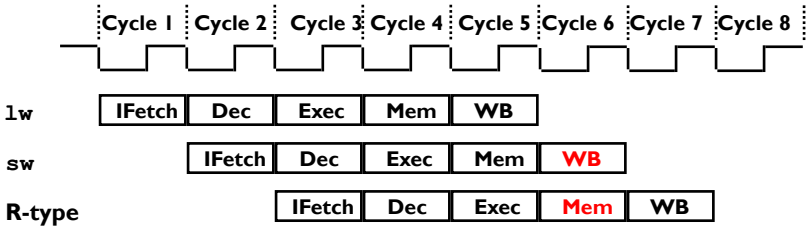
How can we make it Even Faster?

- Split the multiple instruction cycle into smaller steps
  - There is a point of diminishing returns where as much time is spent reading the state registers as doing the work
- Start fetching and executing the next instructions before the current one has completed
  - Pipelining – All modern processors are pipelined for performance
  - Remember the performance equation: CPU Time = IC x CPI x CC
- Fetch and execute more than one instruction at a time
  - Superscalar processing
  - VLIW processing

8

A Pipelined MIPS Processor

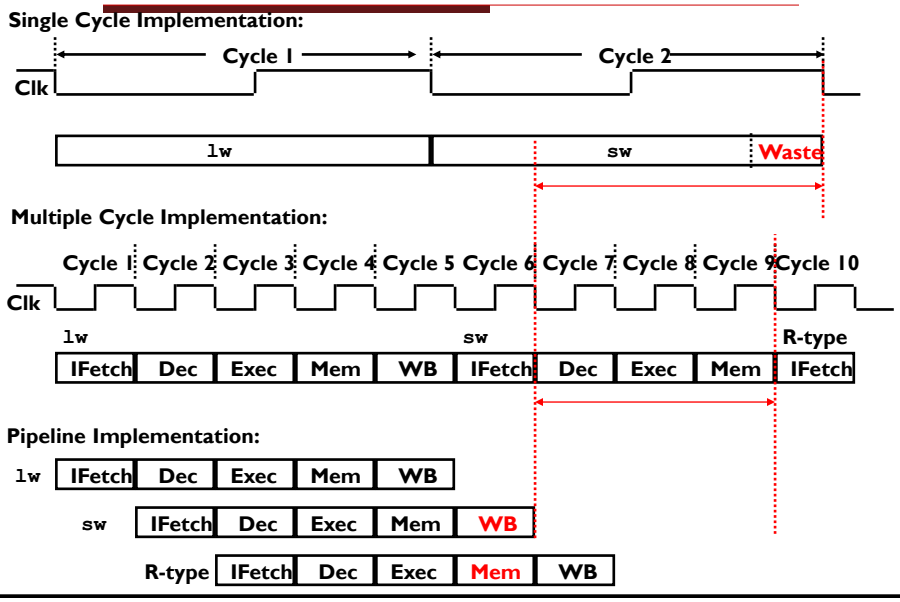
- Start the next instruction before the current one has completed
  - improves throughput - total amount of work done in a given time
  - instruction latency (execution time, delay time, response time - time from the start of an instruction to its completion) is *not* reduced



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

9

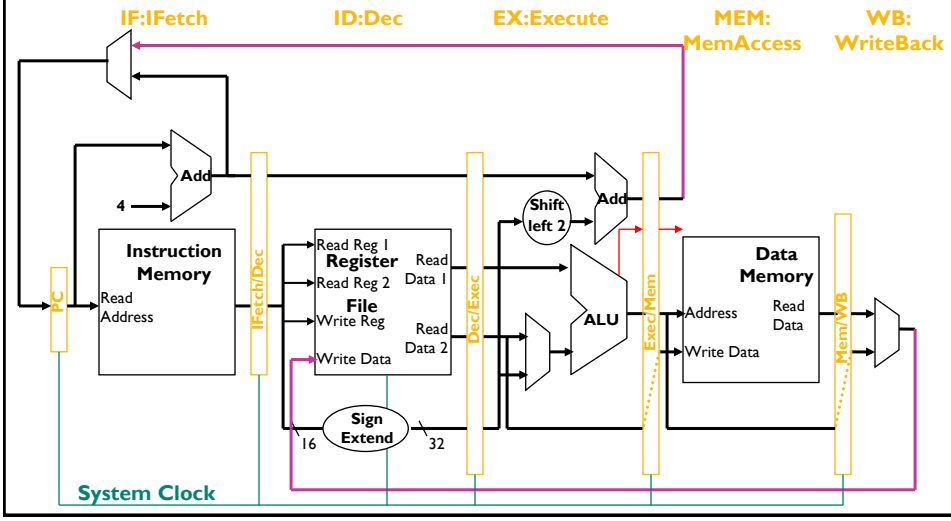
Single Cycle, Multiple Cycle, vs. Pipeline



10

MIPS Pipeline Datapath Modifications

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



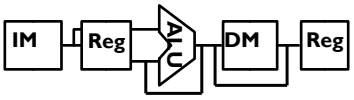
11

Pipelining the MIPS ISA

- What makes pipelining 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 pipelining 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?

12

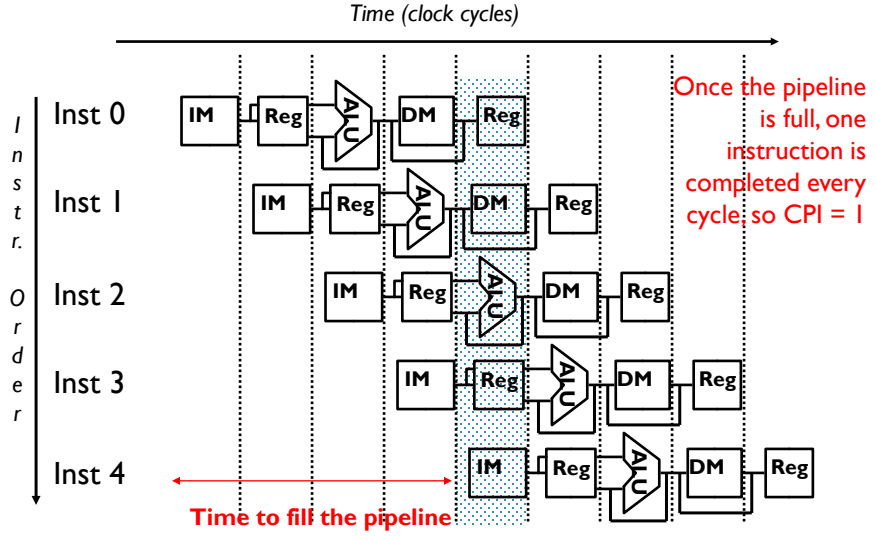
Graphically Representing MIPS Pipeline



- 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?

13

Why Pipeline? For Performance!



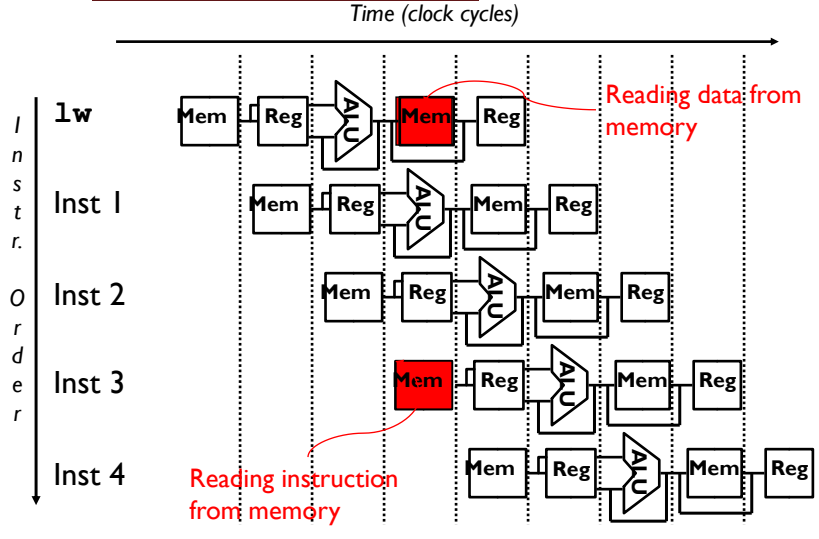
14

Can Pipelining Get Us Into Trouble?

- Yes – Pipelining 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 hazard: 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
  - Pipelining control must detect hazards and take action to resolve hazard

15

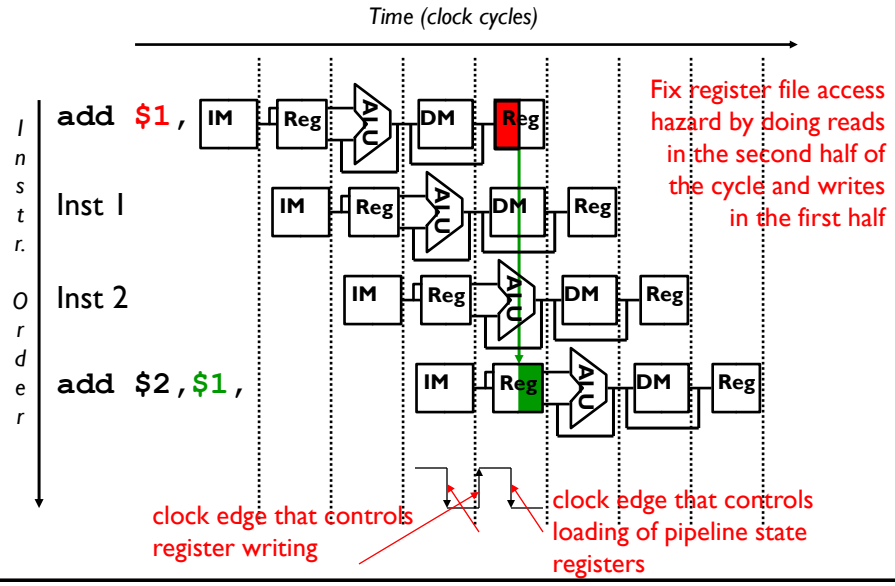
A Single Memory Would Be a Structural Hazard



Fix with separate instruction and data memories (I$ and D$)

16

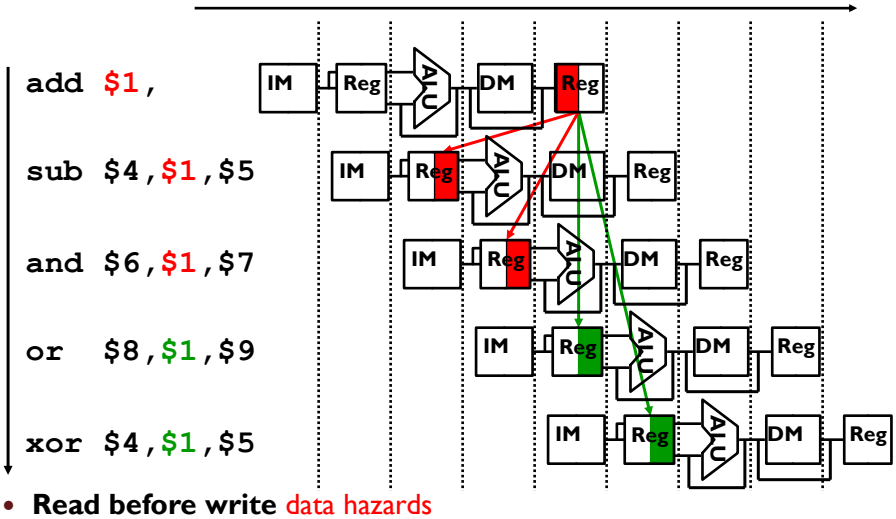
How About Register File Access?



18

Register Usage Can Cause Data Hazards

- Dependencies backward in time cause hazards

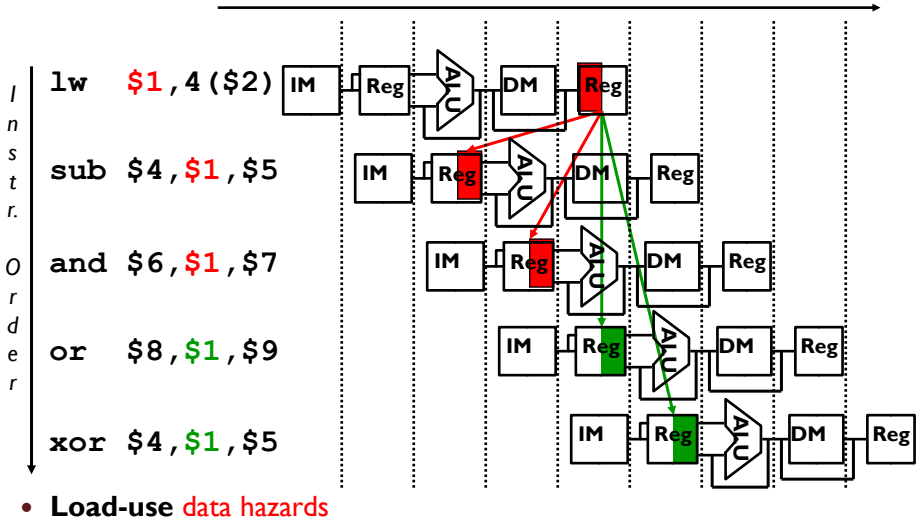


- **Read before write** data hazards

20

Loads Can Cause Data Hazards

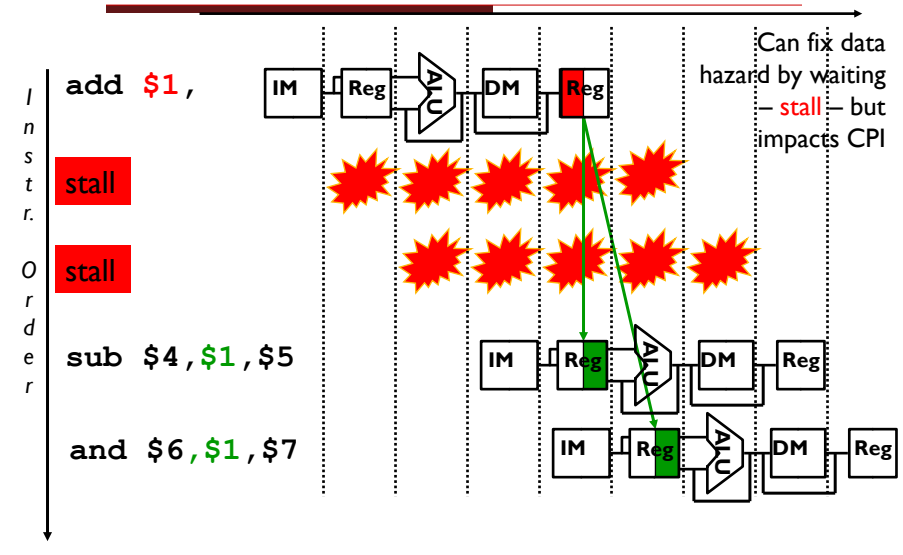
- Dependencies backward in time cause hazards



- **Load-use** data hazards

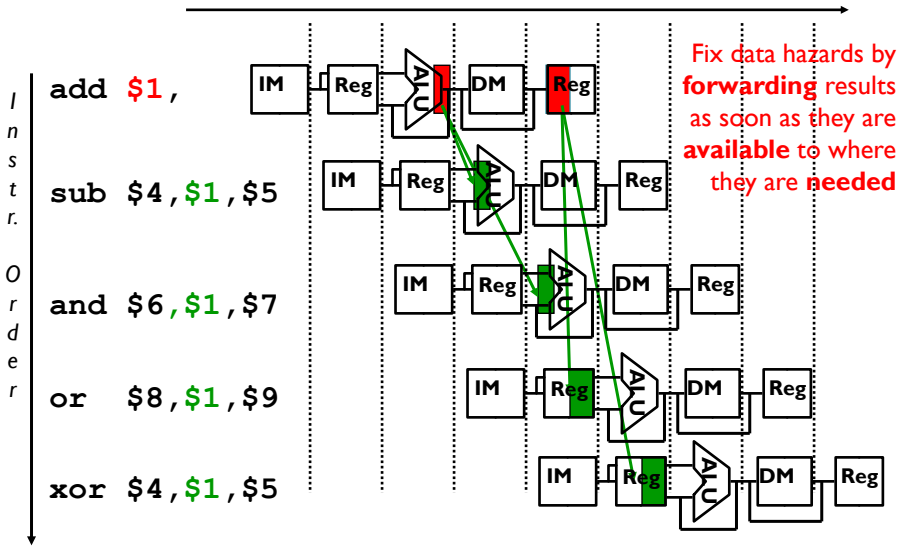
21

One Way to "Fix" a Data Hazard



22

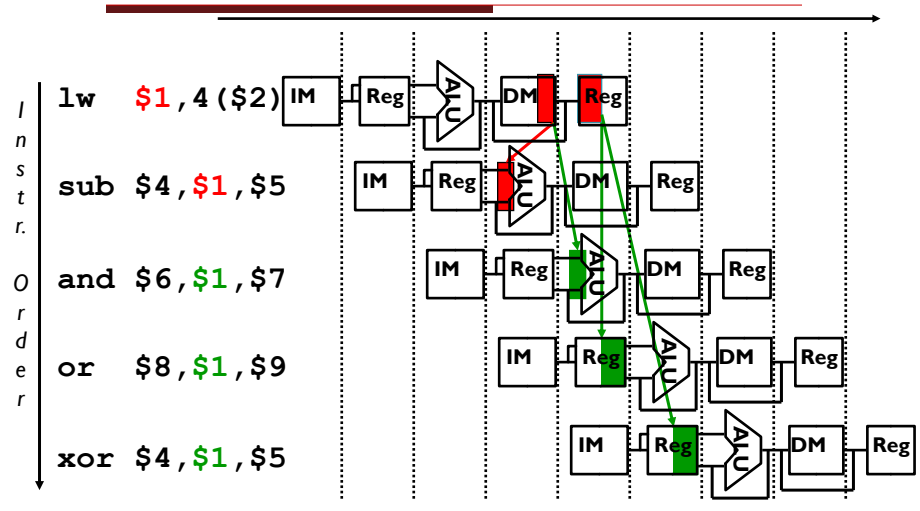
Another Way to "Fix" a Data Hazard



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

24

Forwarding with Load-use Data Hazards

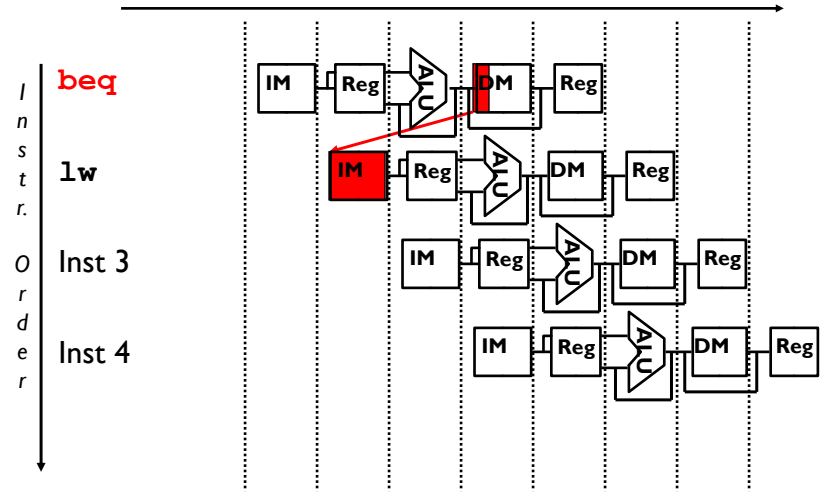


- Will still need one stall cycle even with forwarding

26

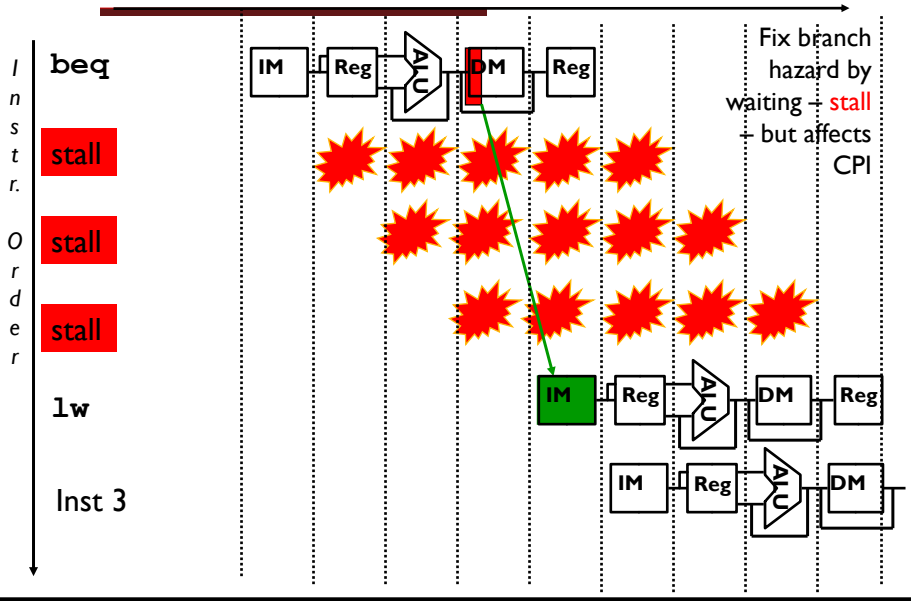
Branch Instructions Cause Control Hazards

- Dependencies backward in time cause hazards



27

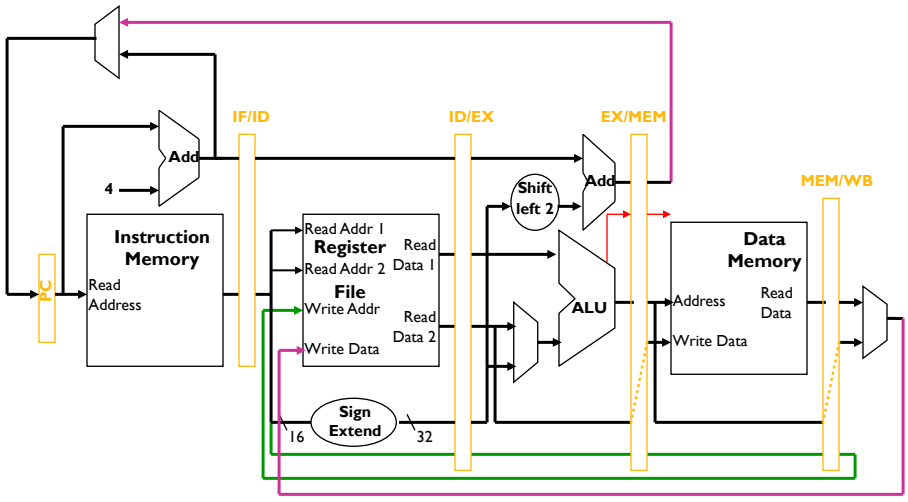
One Way to "Fix" a Control Hazard



28

Corrected Datapath to Save RegWrite Addr

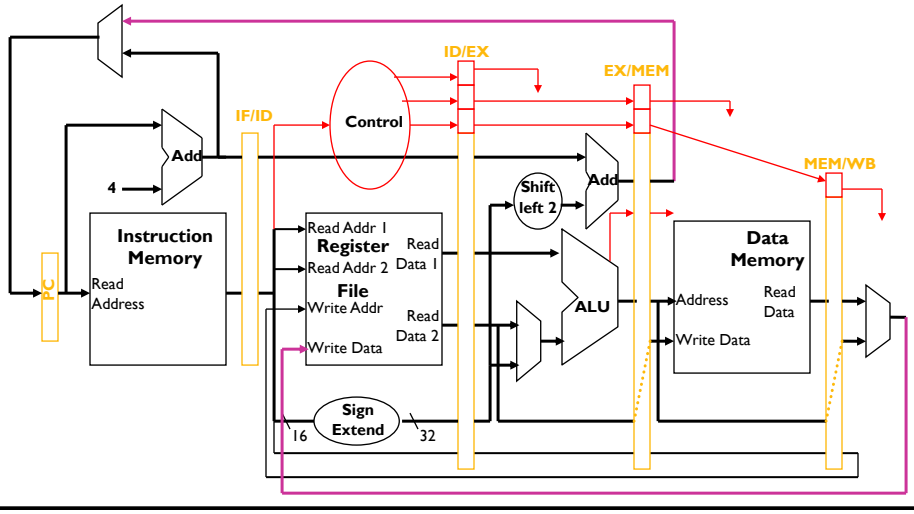
- Need to preserve the destination register address in the pipeline state registers



30

MIPS Pipeline Control Path Modifications

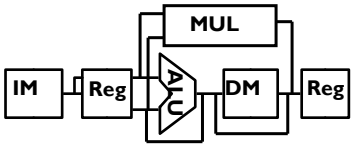
- All control signals can be determined during Decode
  - and held in the state registers between pipeline stages



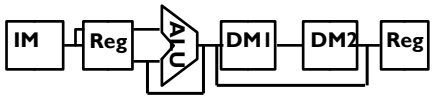
31

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 the data memory access take two clock cycles and (keep the same clock rate)



32

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)

33