CS152 – Computer Architecture and Engineering

Lecture 10 – Introduction to Pipelining

2003-09-24

Dave Patterson

(www.cs.berkeley.edu/~patterson)

www-inst.eecs.berkeley.edu/~cs152/

Review (1 of 4)

- Disadvantages of the Single Cycle Processor
  - Long cycle time
  - Cycle time is too long for all instructions except the Load
  - No reuse of hardware
- Multiple Cycle Processor:
  - Divide the instructions into smaller steps
  - Execute each step (instead of the entire instruction) in one cycle
- Partition datapath into equal size chunks to minimize cycle time
  - ~10 levels of logic between latches
- Follow same 5-step method for designing “real” processor

Review (2 of 4)

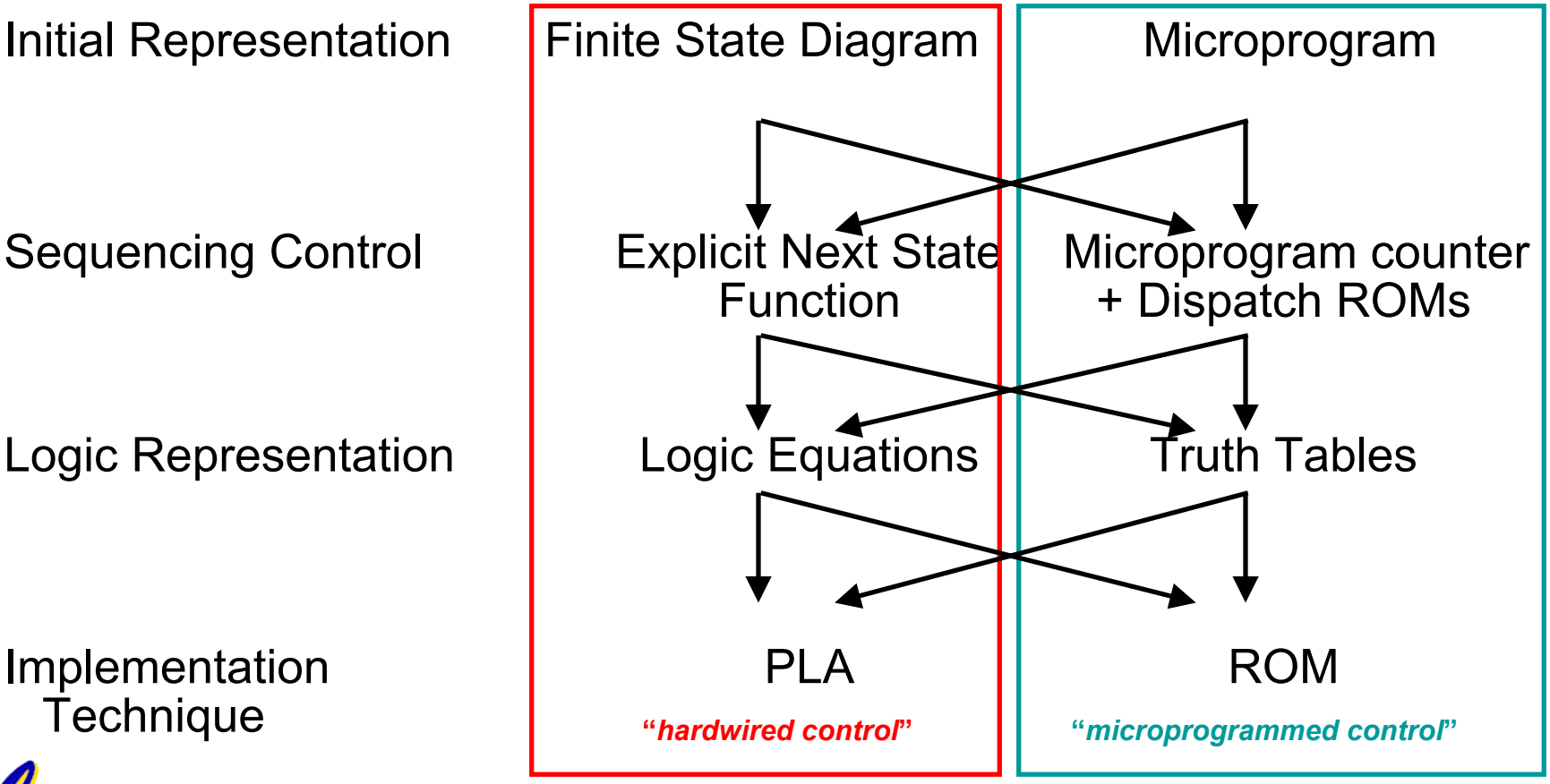
- Control is specified by finite state diagram
- Specialized state-diagrams easily captured by microsequencer
  - simple increment & “branch” fields
  - datapath control fields
- Control is more complicated with:
  - complex instruction sets
  - restricted datapaths (see the book)
- Control design can become Microprogramming

Summary (3 of 4)

- Microprogramming is a fundamental concept
  - implement an instruction set by building a very simple processor and interpreting the instructions
  - essential for very complex instructions and when few register transfers are possible
  - *Control design reduces to Microprogramming*
- Design of a Microprogramming language
  - Start with list of control signals
  - Group signals together that make sense (vs. random): called “fields”
  - Place fields in some logical order (e.g., ALU operation & ALU operands first and microinstruction sequencing last)
  - To minimize the width, encode operations that will never be used at the same time
  - Create a symbolic legend for the microinstruction format, showing name of field values and how they set the control signals

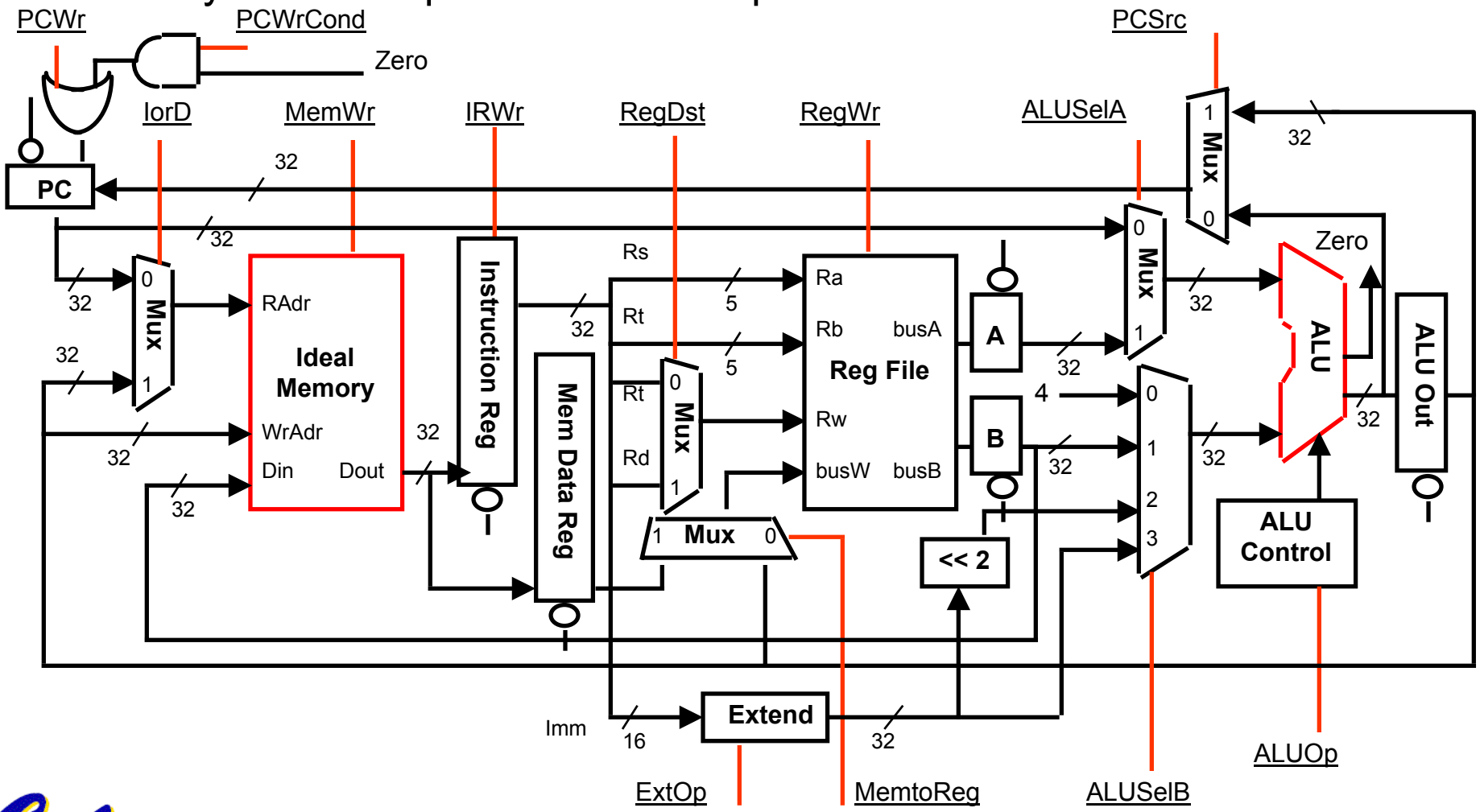
Review: Overview of Control

- Control may be designed using one of several initial representations. The choice of sequence control, and how logic is represented, can then be determined independently; the control can then be implemented with one of several methods using a structured logic technique.



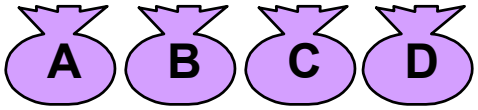
Can we get CPI < 4?

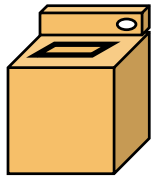
- Seems to be lots of "idle" hardware
  - Why not overlap instructions? Pipeline

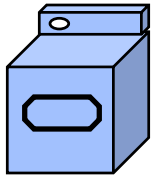


Pipelining is Natural!

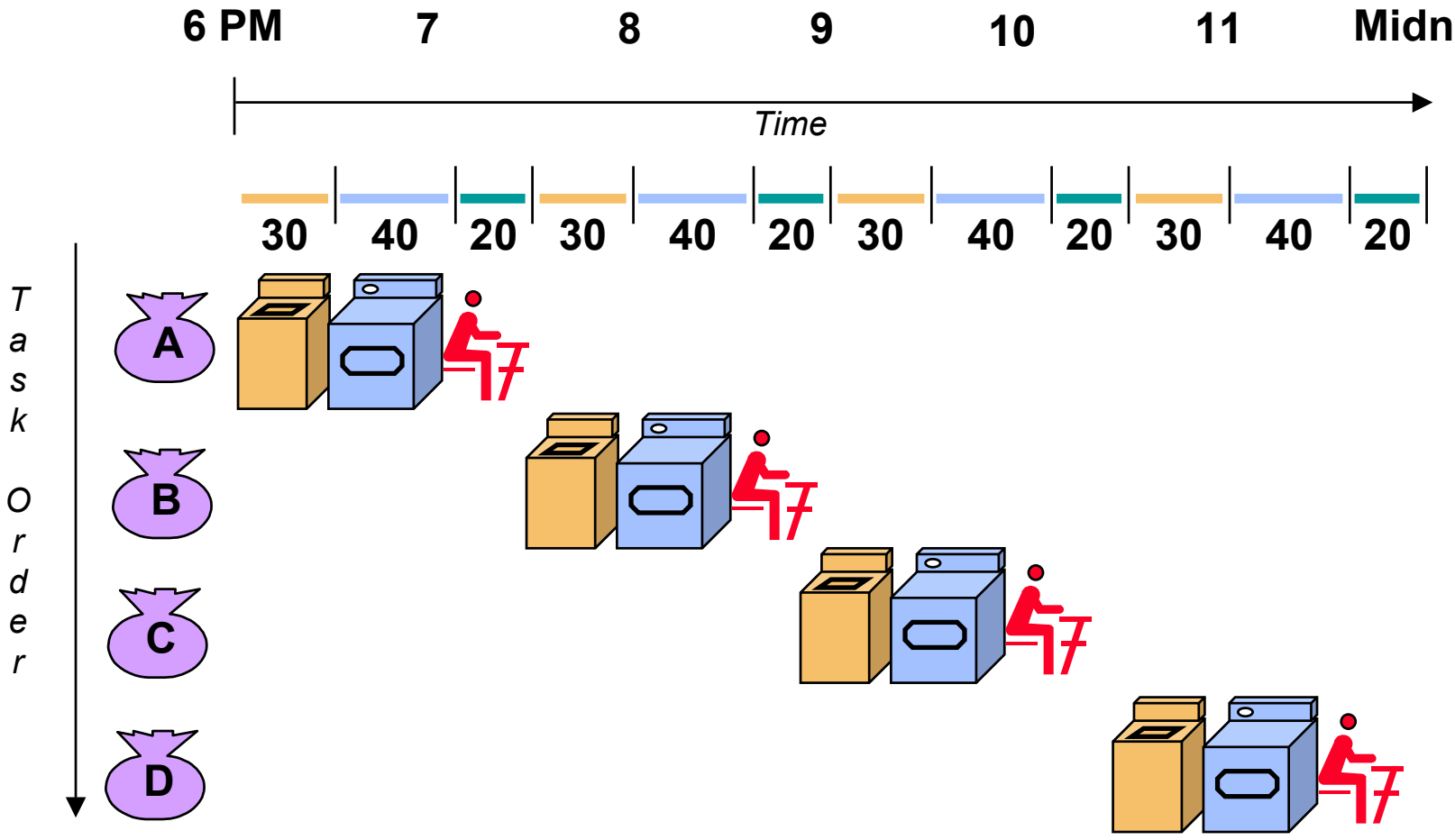
- Laundry Example
- Ann, Brian, Cathy, Dave each have one load of clothes to wash, dry, and fold
- Washer takes 30 minutes
- Dryer takes 40 minutes
- "Folder" takes 20 minutes





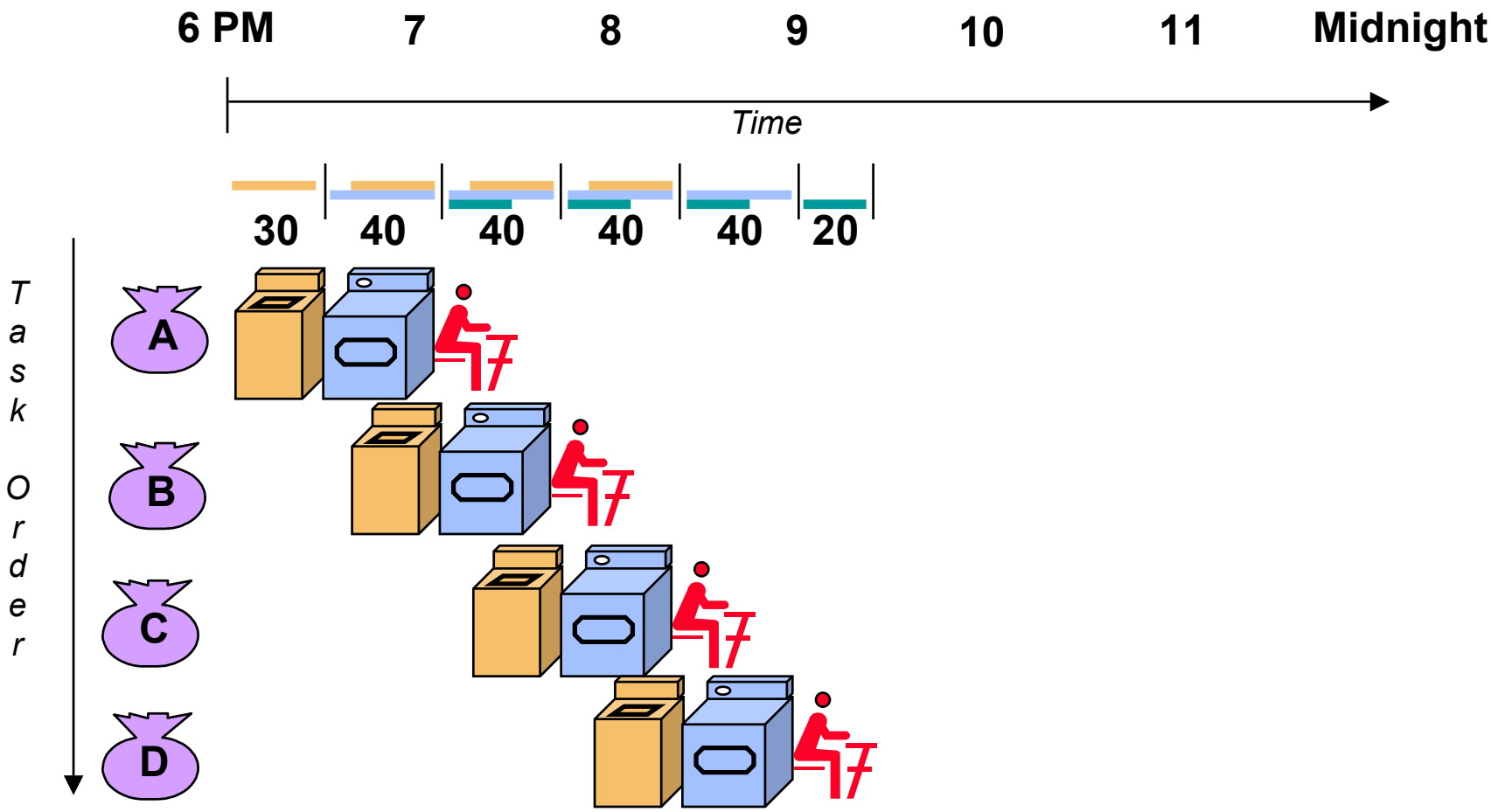


Sequential Laundry



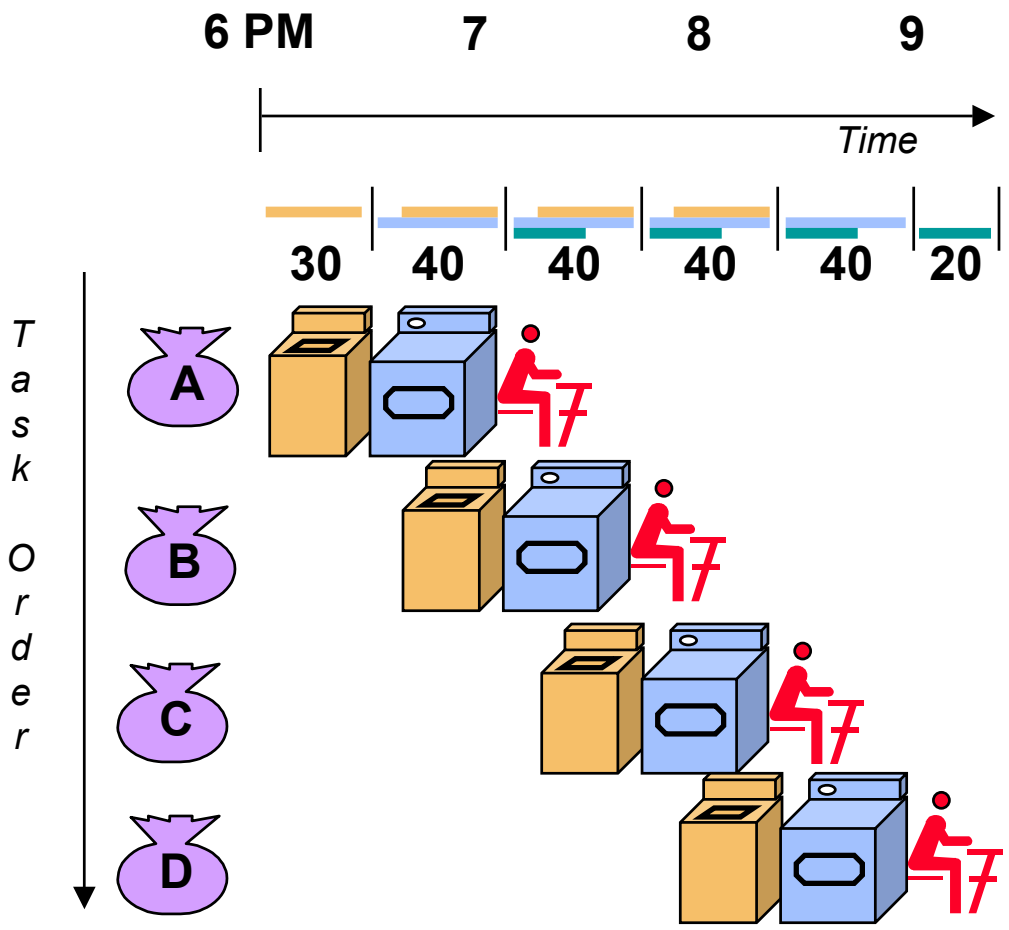
- Sequential laundry takes 6 hours for 4 loads
- If they learned pipelining, how long would laundry take?

Pipelined Laundry: Start work ASAP



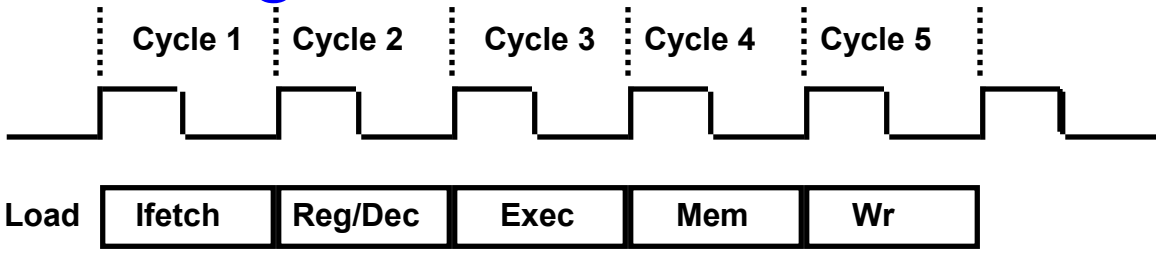
- Pipelined laundry takes 3.5 hours for 4 loads

Pipelining Lessons



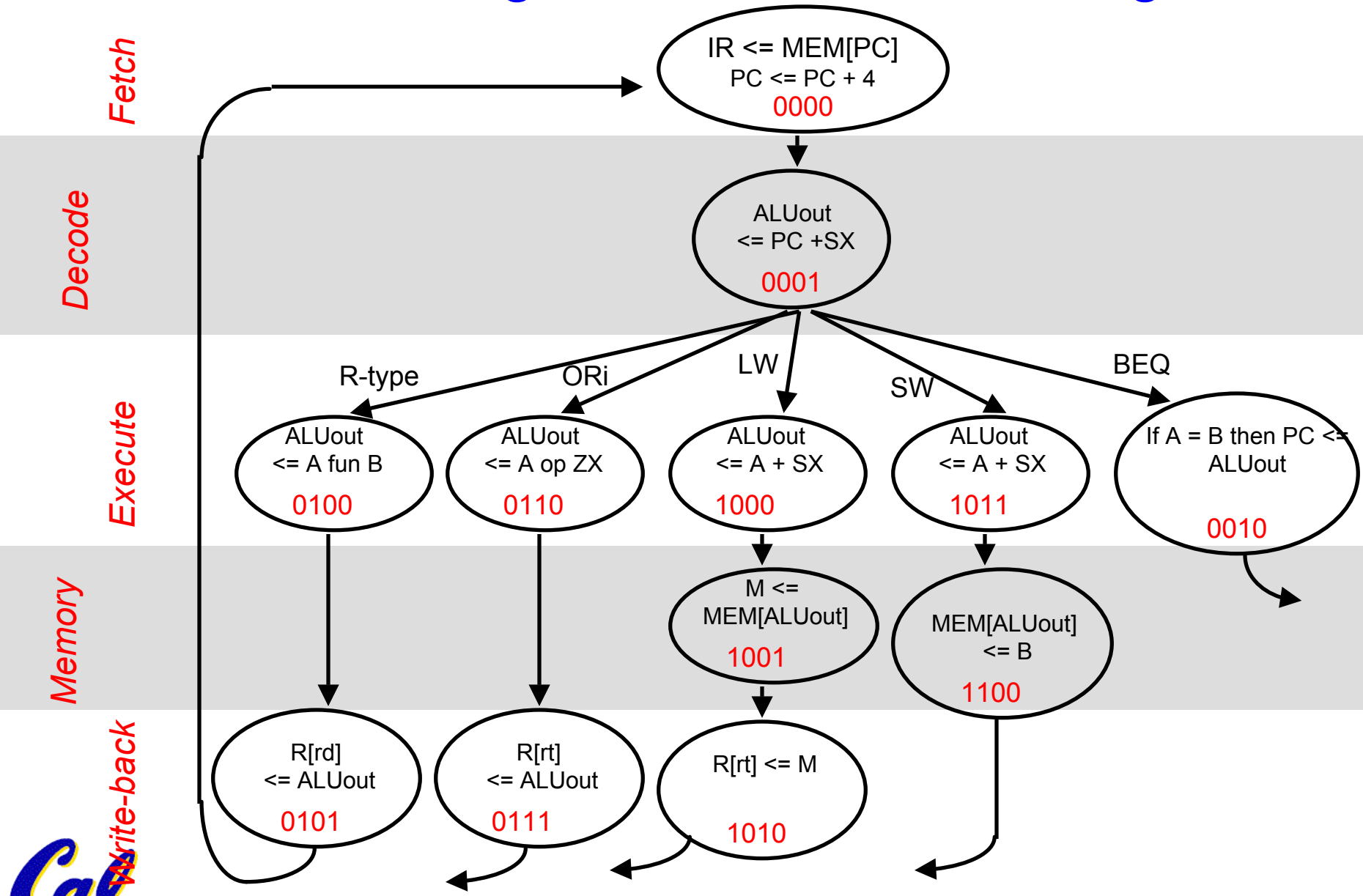
- Pipelining doesn't help latency of single task, it helps throughput of entire workload
- Pipeline rate limited by slowest pipeline stage
- Multiple tasks operating simultaneously using different resources
- Potential speedup = Number pipe stages
- Unbalanced lengths of pipe stages reduces speedup
-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

The Five Stages of Load



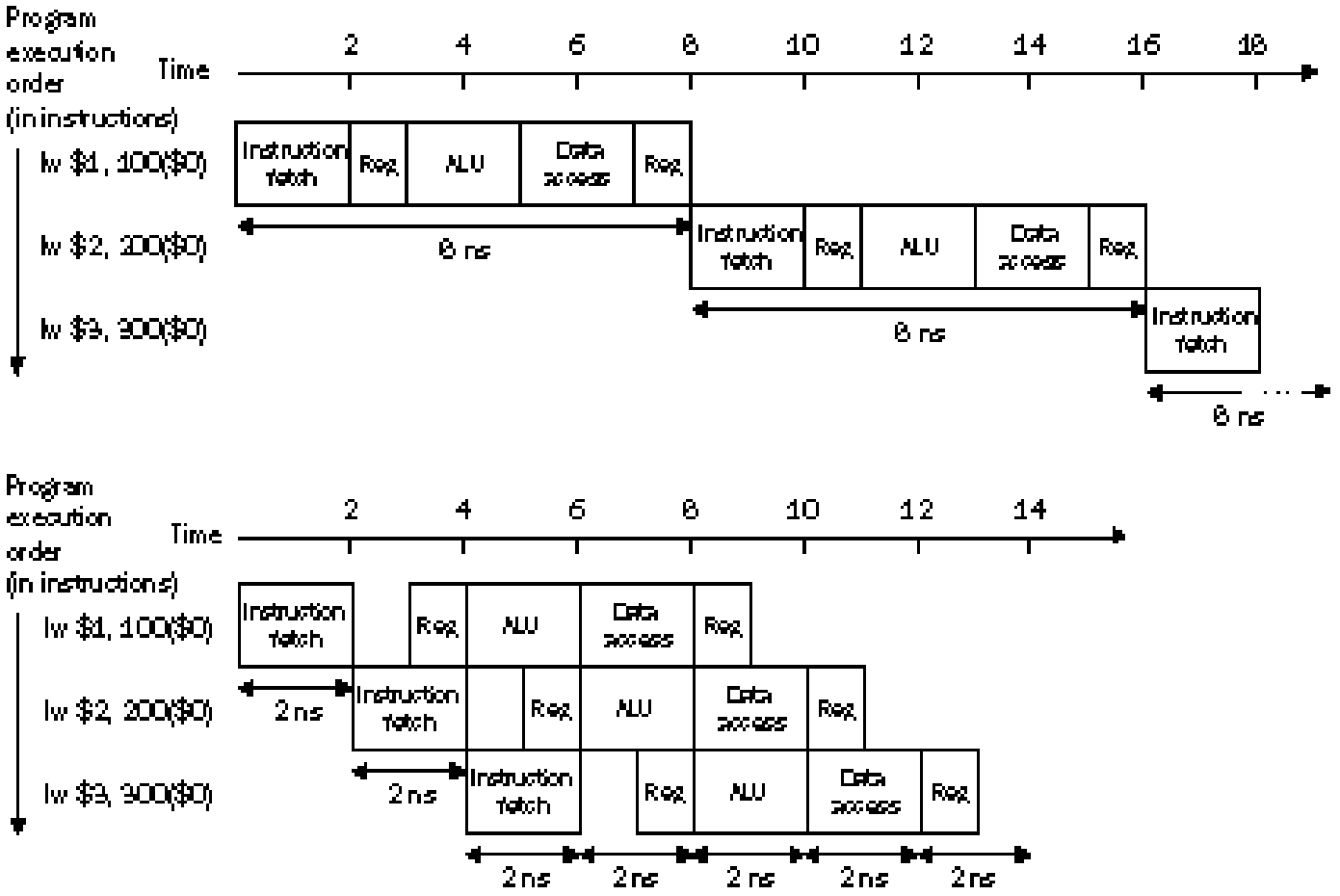
- Ifetch: Instruction Fetch
  - Fetch the instruction from the Instruction Memory
- Reg/Dec: Registers Fetch and Instruction Decode
- Exec: Calculate the memory address
- Mem: Read the data from the Data Memory
- Wr: Write the data back to the register file

Note: These 5 stages were there all along!



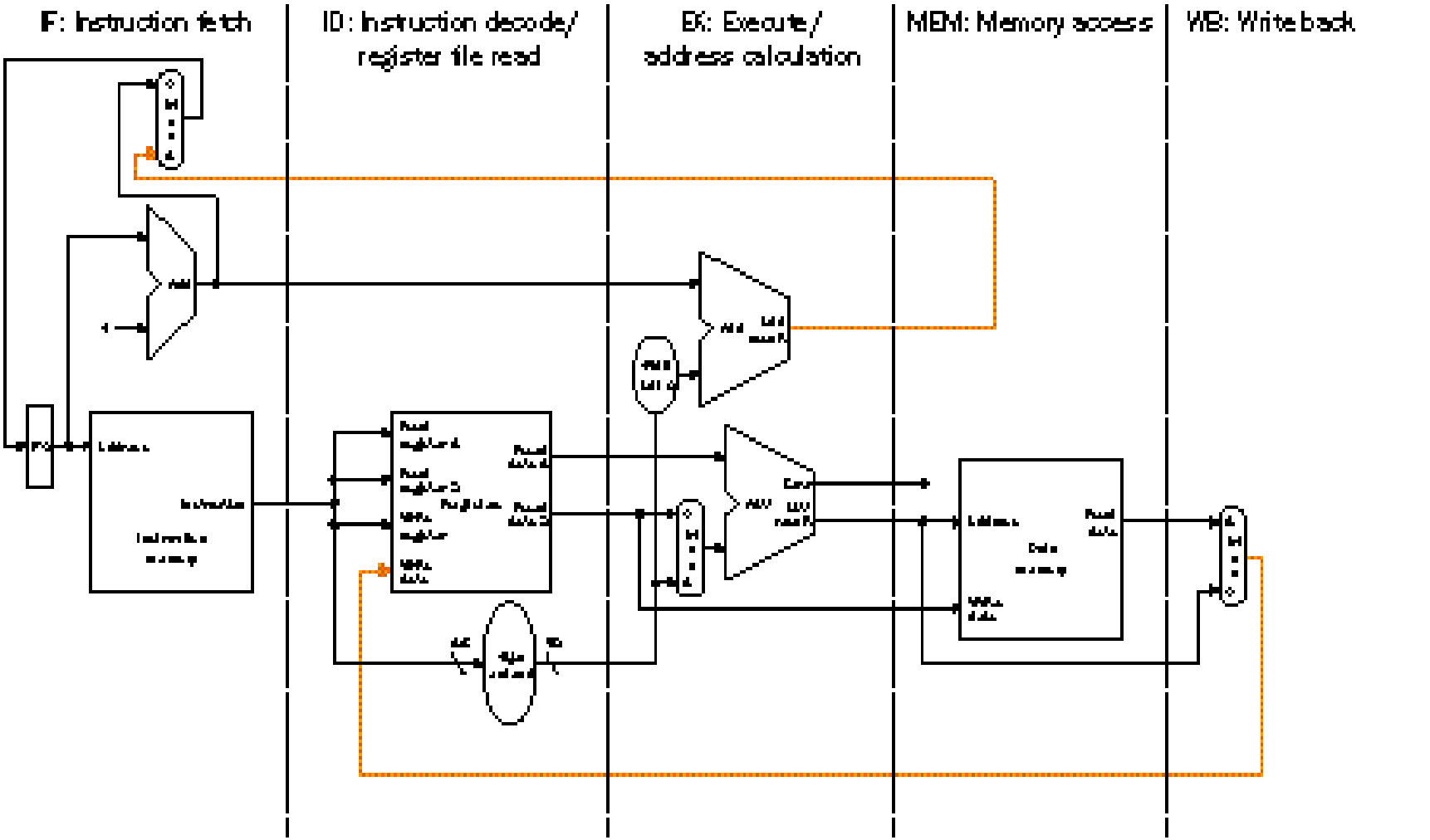
Pipelining

- Improve performance by increasing throughput

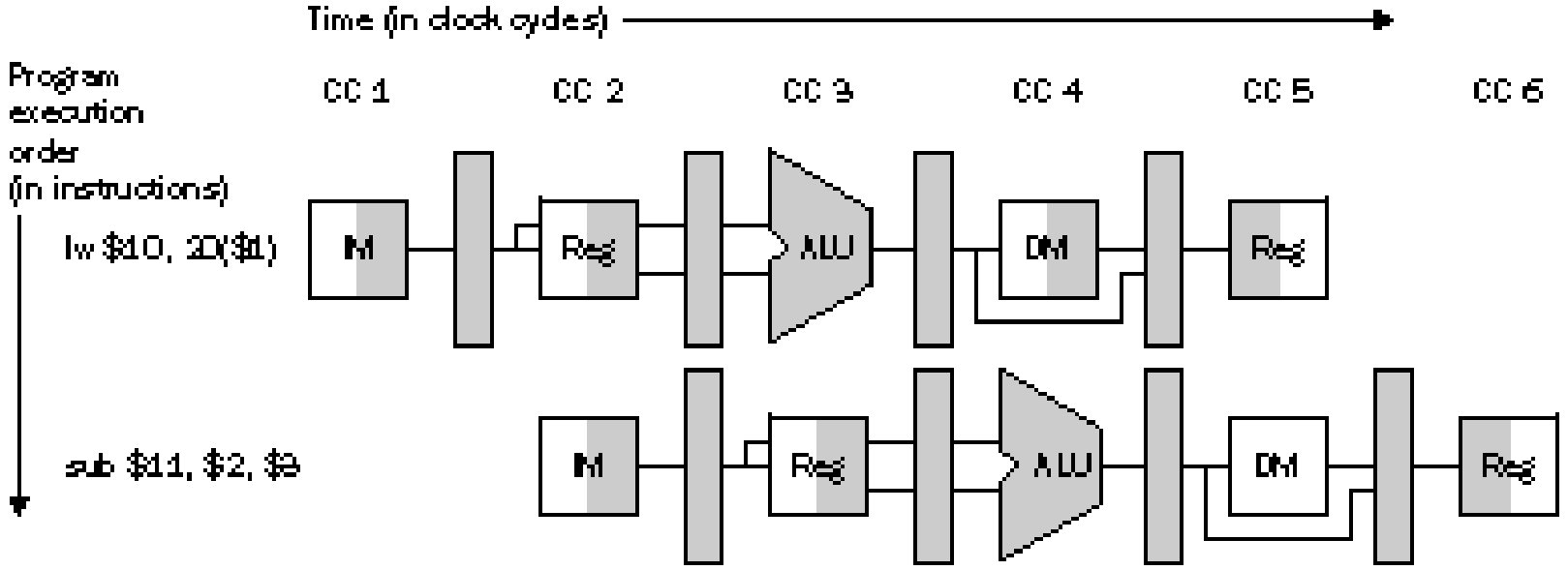


Ideal speedup is number of stages in the pipeline. Do we achieve this?

Basic Idea

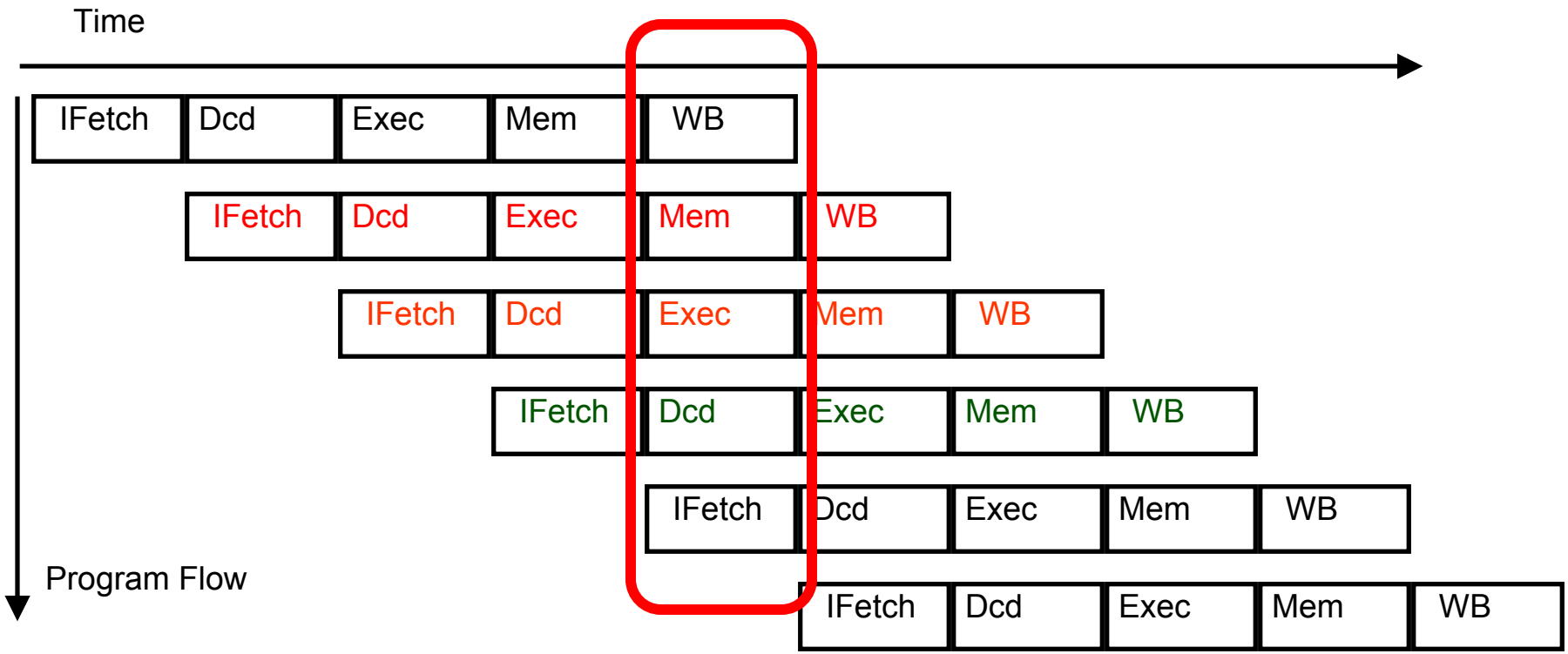


Graphically Representing Pipelines



- Can help with answering questions like:
  - how many cycles does it take to execute this code?
  - what is the ALU doing during cycle 4?
  - use this representation to help understand datapaths

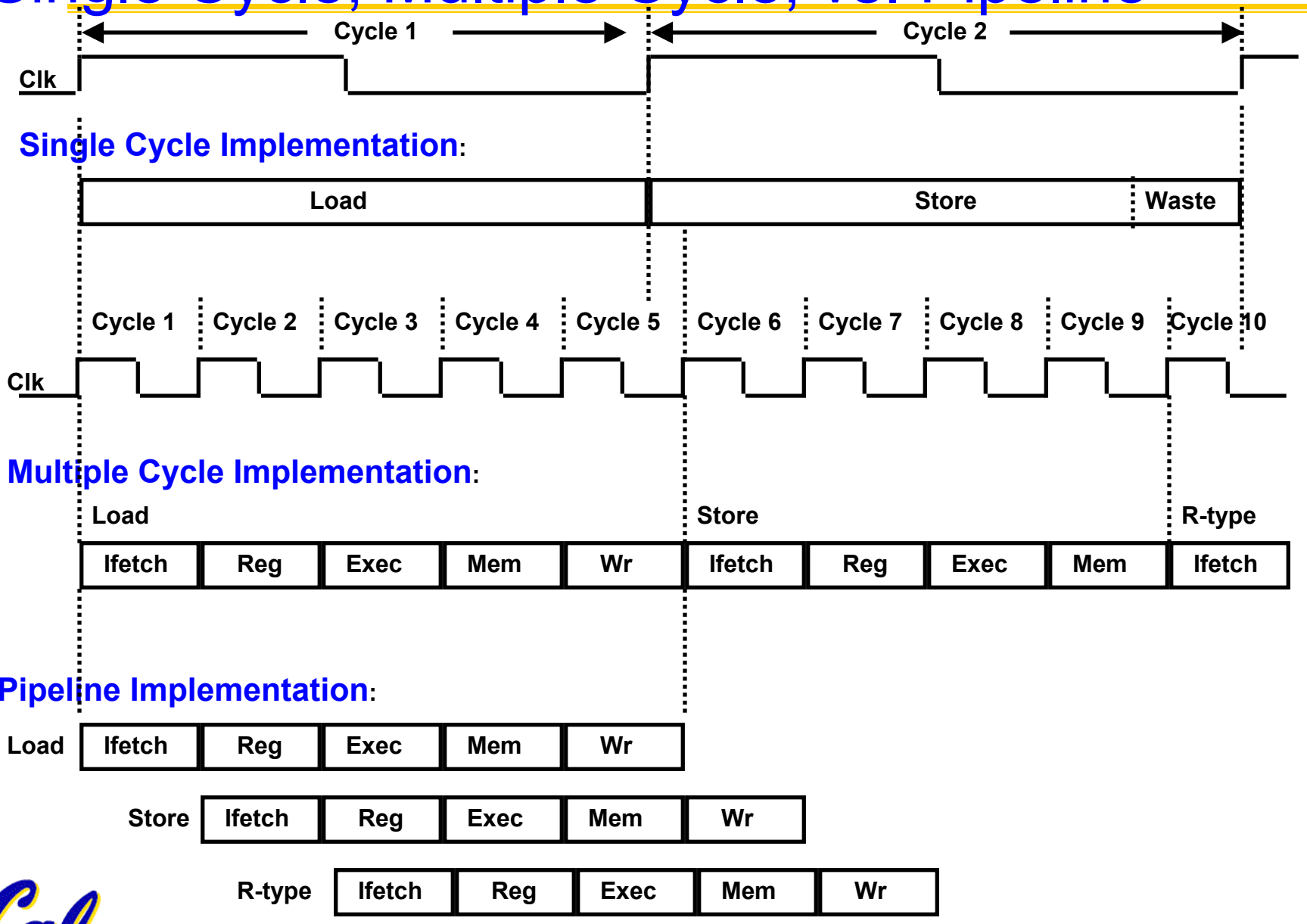
Conventional Pipelined Execution Representation



Administrivia

- Office hours in Lab
  - Mon 4 – 5:30 Jack, Tue 3:30-5 Kurt,
    Wed 3 – 4:30 John, Thu 3:30-5 Ben
- Dave's office hours Tue 3:30 – 5
- Lab 3 demo Friday, due Monday
- Reading Chapter 6, sections 6.1 to 6.4
- Midterm Wed Oct 8 5:30 - 8:30 in 1 LeConte
  - Midterm review Sunday Oct 4, 5 PM in 306 Soda
  - Bring 1 page, handwritten notes, both sides
  - Meet at LaVal's Northside afterwards for Pizza

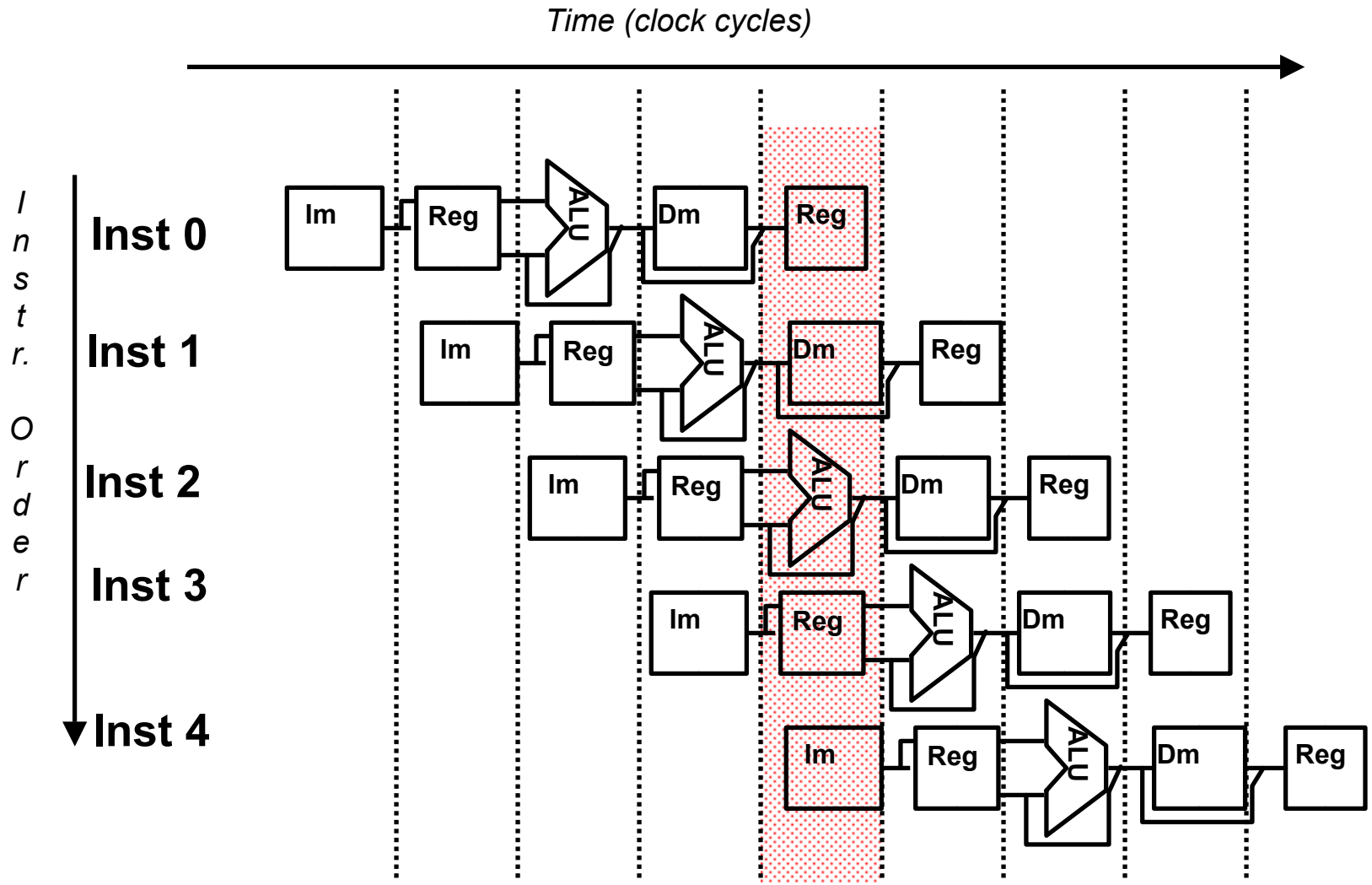
Single Cycle, Multiple Cycle, vs. Pipeline



Why Pipeline?

- Suppose we execute 100 instructions
- Single Cycle Machine
  - 4.5 ns/cycle x 1 CPI x 100 inst = 450 ns
- Multicycle Machine
  - 1.0 ns/cycle x 4.1 CPI (due to inst mix) x 100 inst = 410 ns
- Ideal pipelined machine
  - 1.0 ns/cycle x (1 CPI x 100 inst + 4 cycle fill) = 104 ns

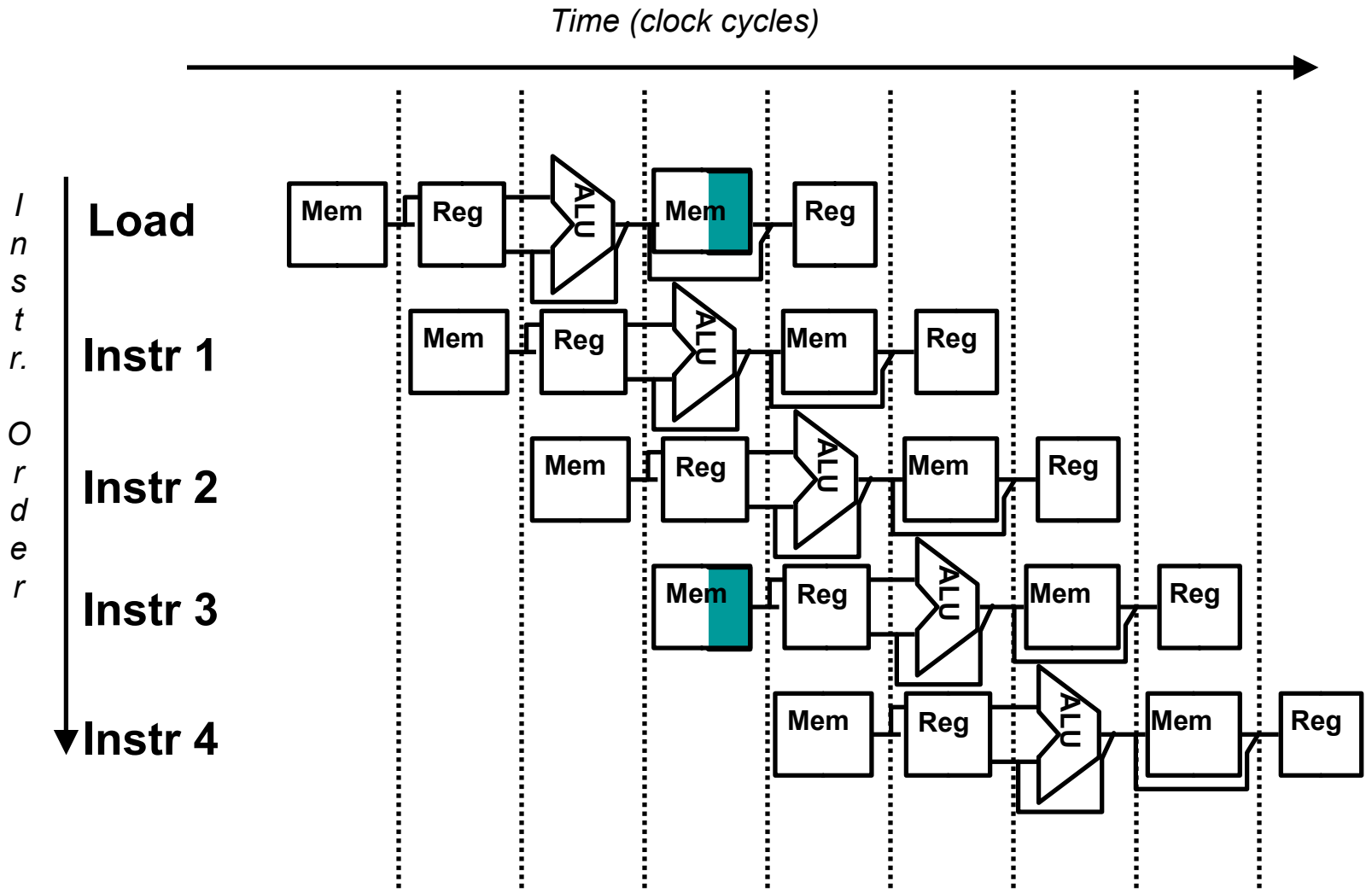
Why Pipeline? Because we can!



Can pipelining get us into trouble?

- Yes: Pipeline Hazards
  - structural hazards: attempt to use the same resource two different ways at the same time
    - E.g., combined washer/dryer would be a structural hazard or folder busy watching TV
  - control hazards: attempt to make a decision before condition is evaluated
    - E.g., washing football uniforms and need to get proper detergent level; need to see after dryer before next load in
    - branch instructions
  - data hazards: attempt to use item before it is ready
    - E.g., one sock of pair in dryer and one in washer; can't fold until get sock from washer through dryer
    - instruction depends on result of prior instruction still in the pipeline
- Can always resolve hazards by waiting
  - pipeline control must detect the hazard
  - take action (or delay action) to resolve hazards

Single Memory is a Structural Hazard

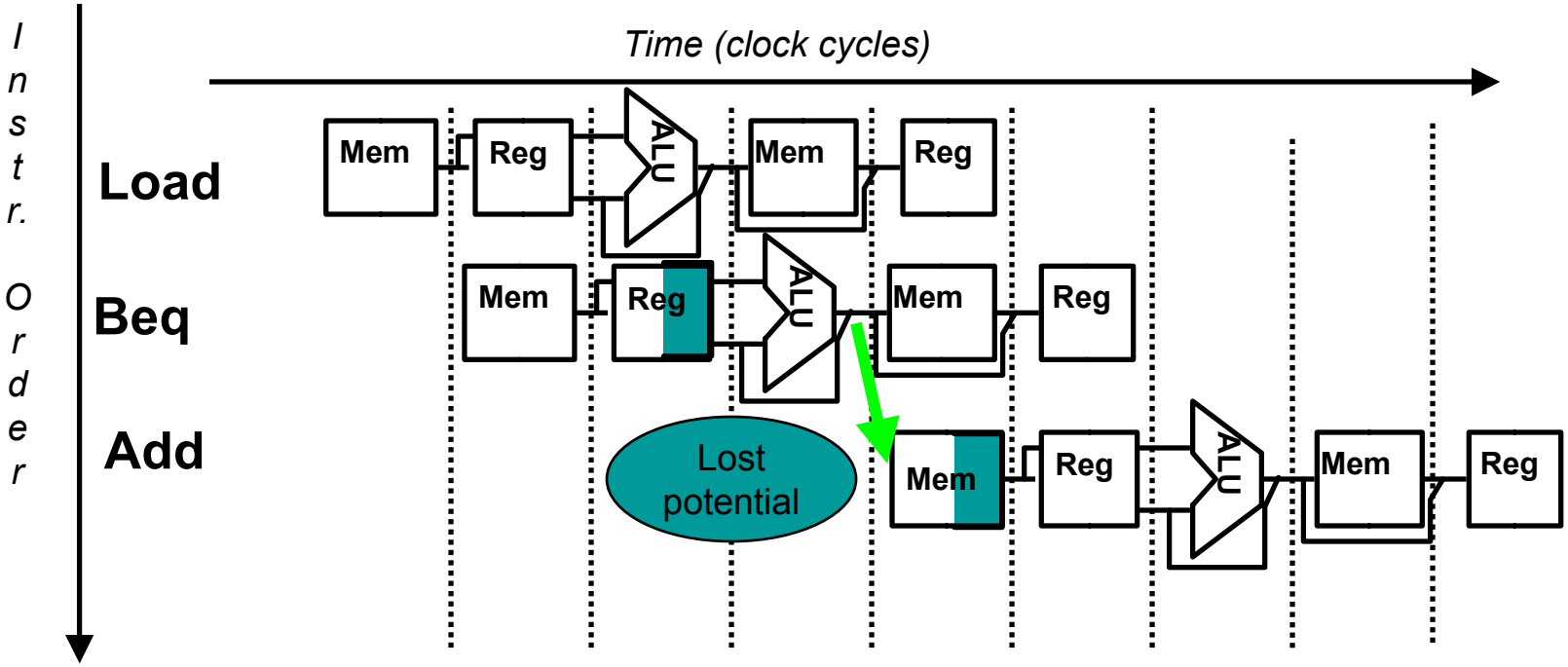


Detection is easy in this case! (right half highlight means read, left half write)

Structural Hazards limit performance

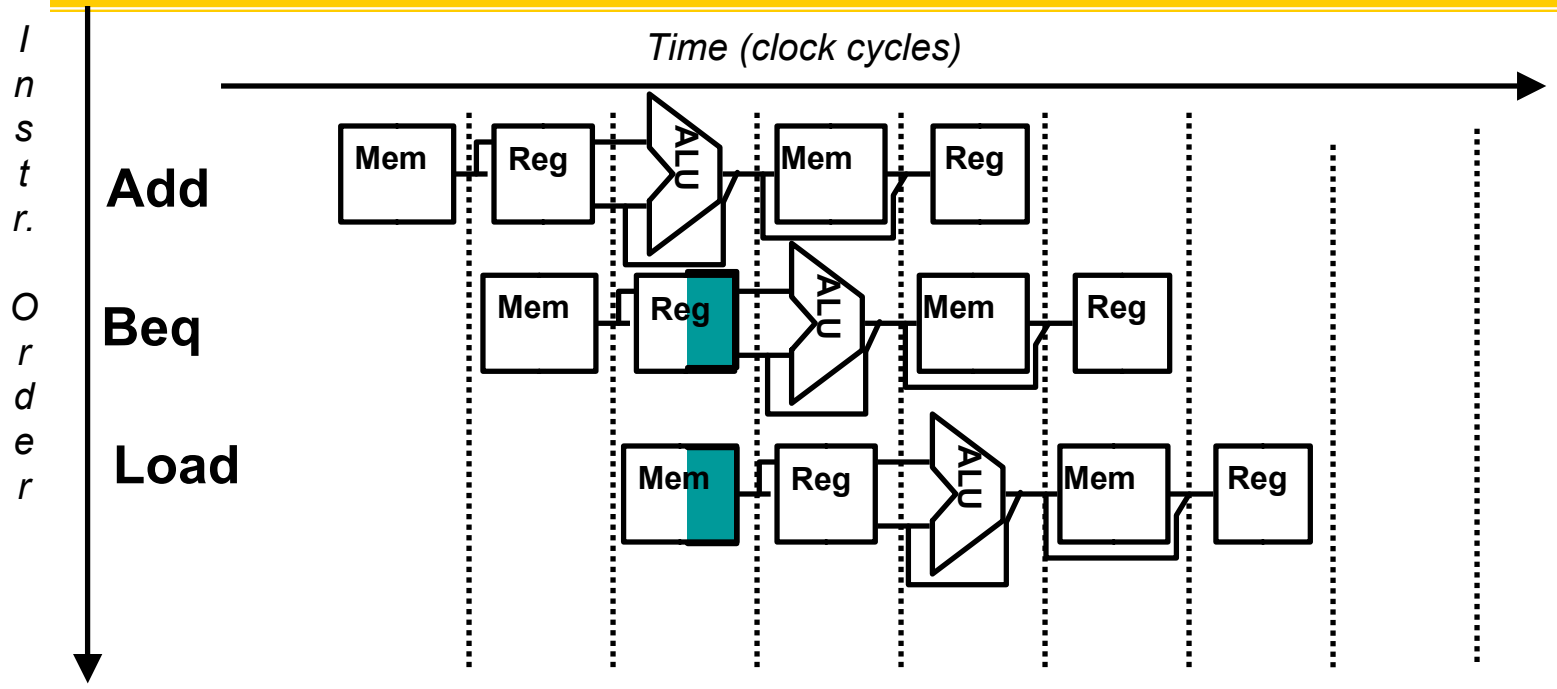
- Example: if 1.3 memory accesses per instruction and only one memory access per cycle then
  - average CPI ≥ 1.3
  - otherwise resource is more than 100% utilized
- One Structural Hazard solution: more resources
  - Instruction cache and Data cache

Control Hazard Solution #1: Stall



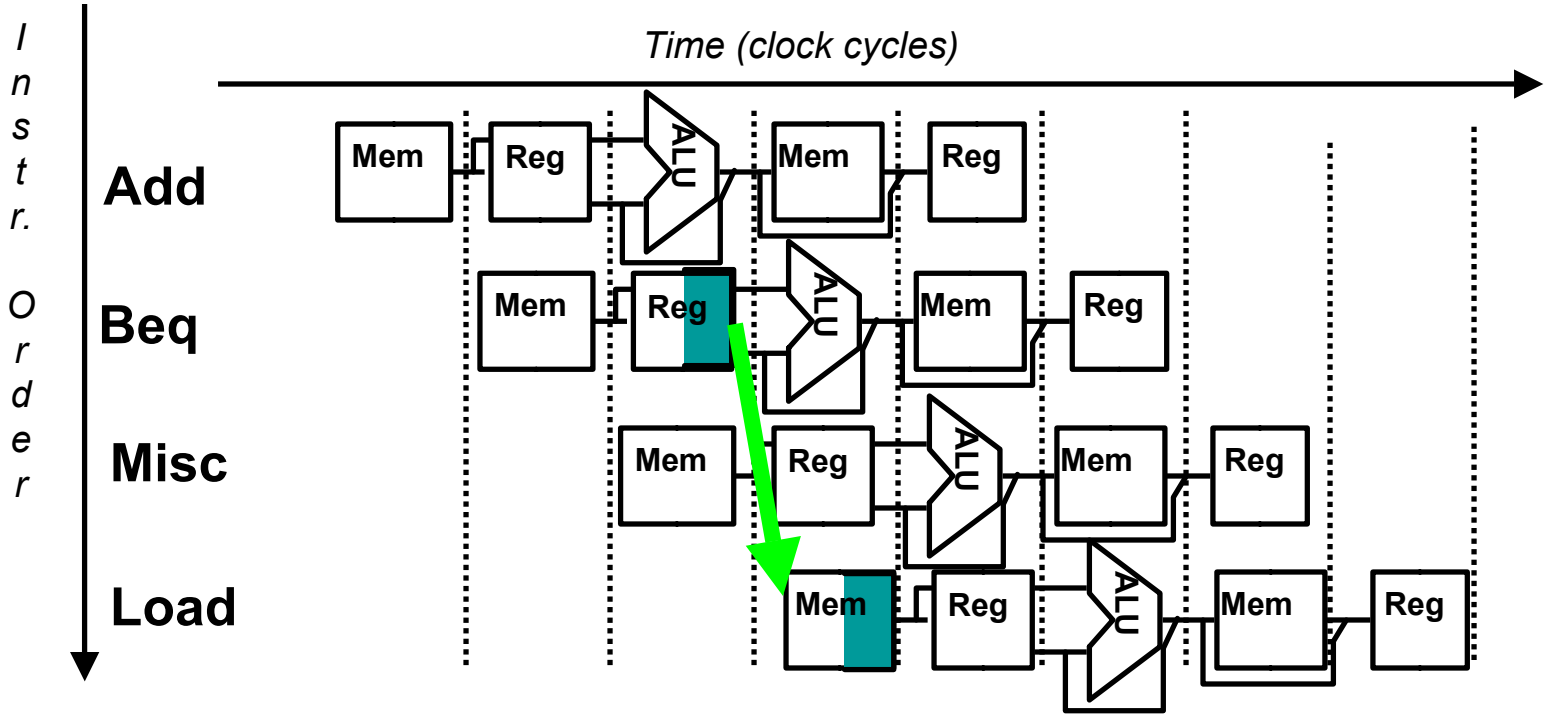
- Stall: wait until decision is clear
- Impact: 2 lost cycles (i.e. 3 clock cycles for Beq instruction above) => slow
- Move decision to end of decode
  - save 1 cycle per branch, may stretch clock cycle

Control Hazard Solution #2: Predict



- Predict: guess one direction then back up if wrong
- Impact: 0 lost cycles per branch instruction if guess right, 1 if wrong (right ~ 50% of time)
  - Need to "Squash" and restart following instruction if wrong
  - Produce CPI on branch of (1 *.5 + 2 * .5) = 1.5
  - Total CPI might then be: 1.5 * .2 + 1 * .8 = 1.1 (20% branch)
- More dynamic schemes: history of branch behavior (~90-99%)

Control Hazard Solution #3: Delayed Branch



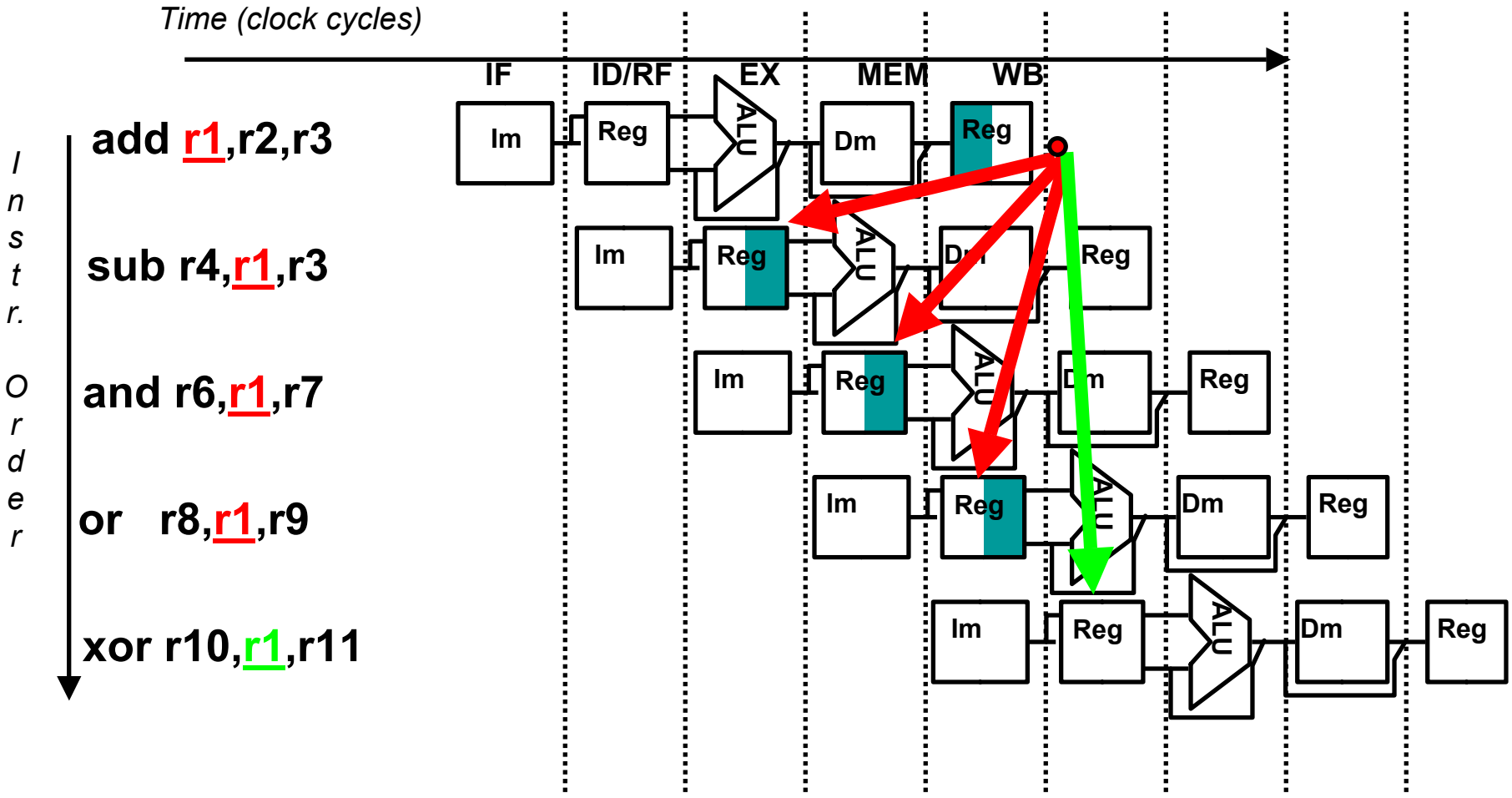
- Delayed Branch: Redefine branch behavior (takes place after next instruction)
- Impact: 0 clock cycles per branch instruction if can find instruction to put in "slot" (~50% of time)
- As launch more instruction per clock cycle, less useful

Data Hazard on r1

```
add r1,r2,r3

sub r4,r1,r3

and r6,r1,r7

or  r8,r1,r9

xor r10,r1,r11
```

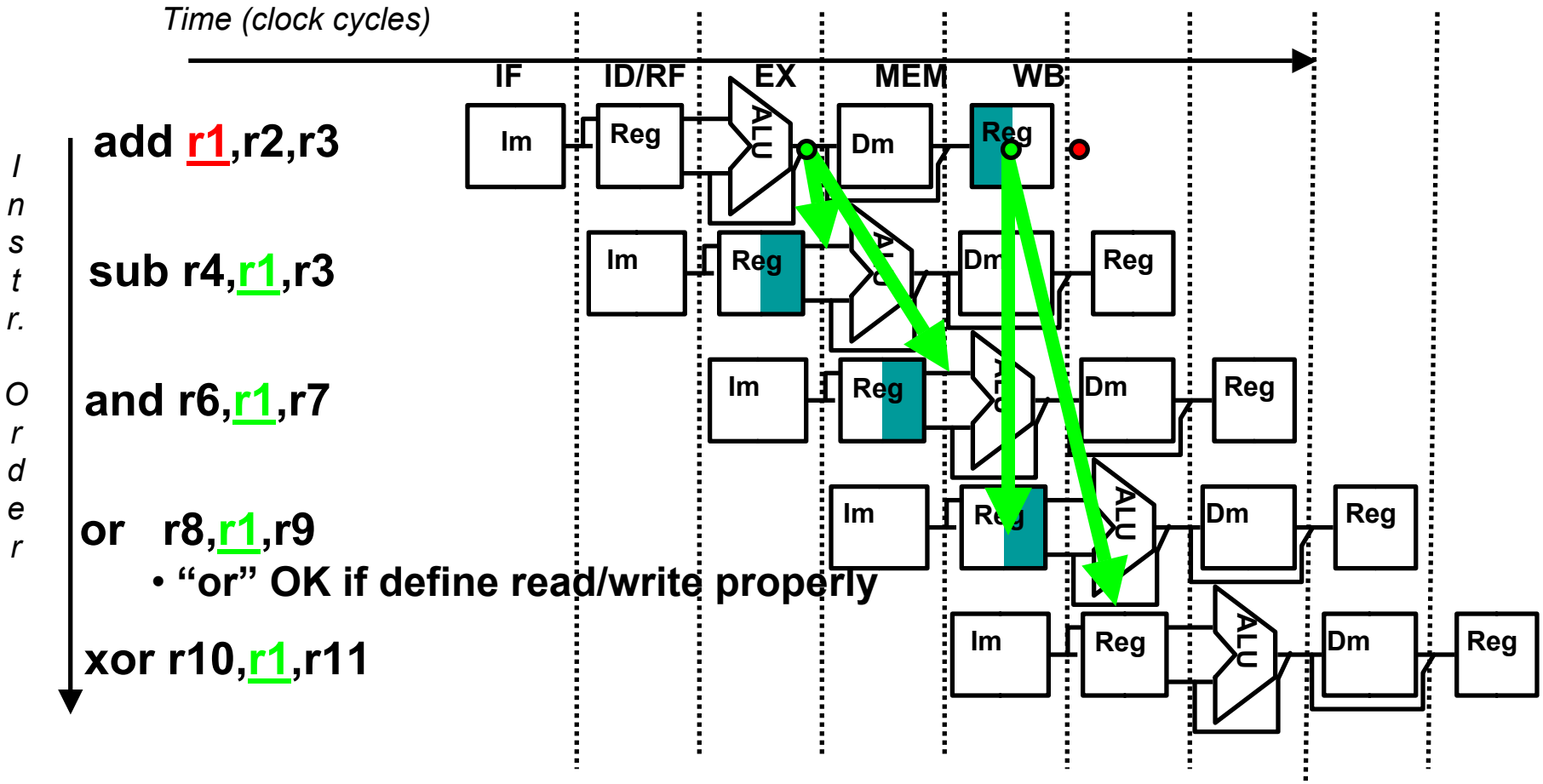
Data Hazard on r1:

- **Dependencies backwards in time are hazards**



Data Hazard Solution:

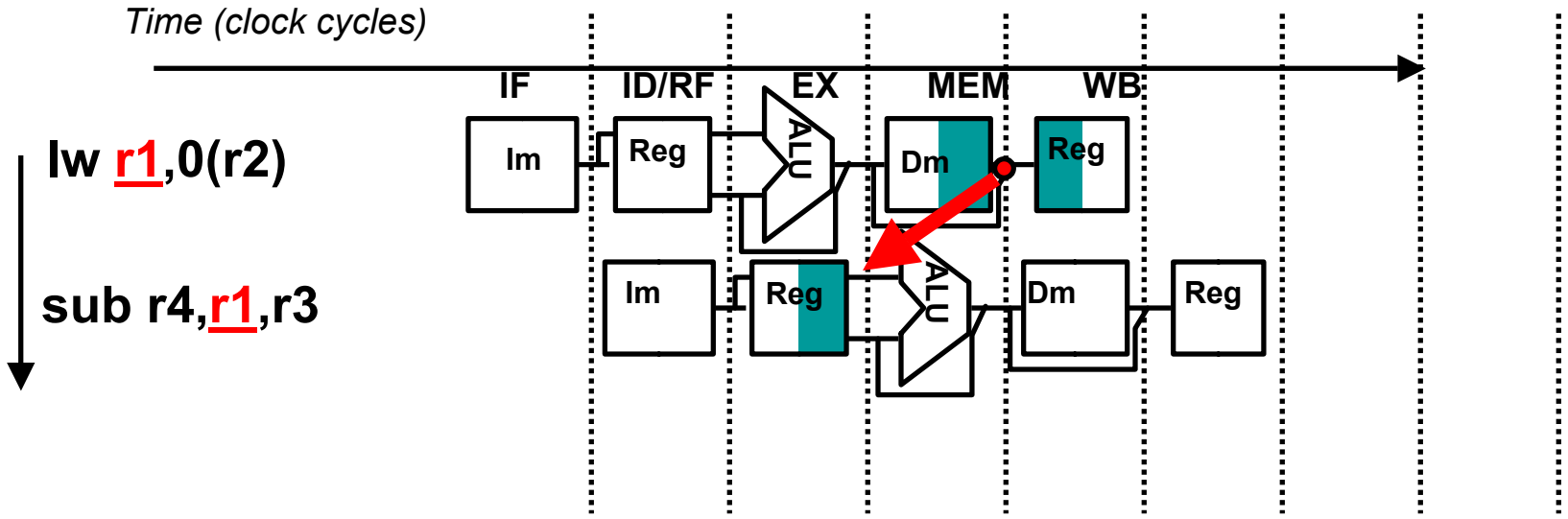
- "Forward" result from one stage to another



- "or" OK if define read/write properly

Forwarding (or Bypassing): What about Loads?

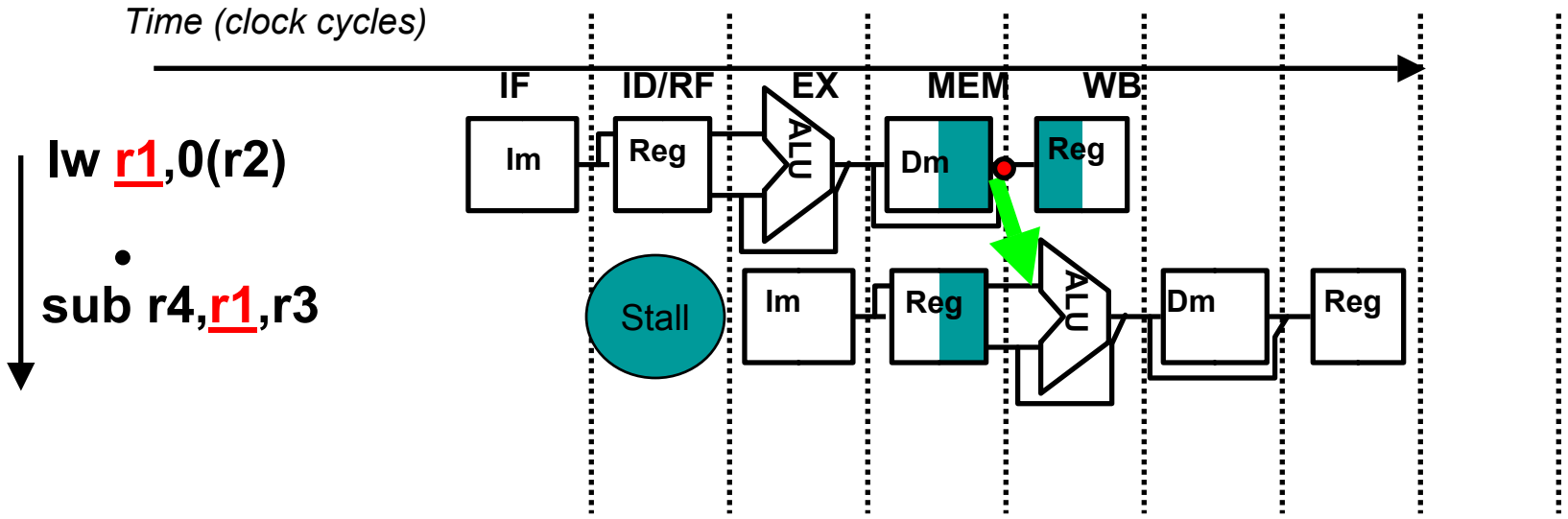
- **Dependencies backwards in time are hazards**



- **Can't solve with forwarding:**
- **Must delay/stall instruction dependent on loads**

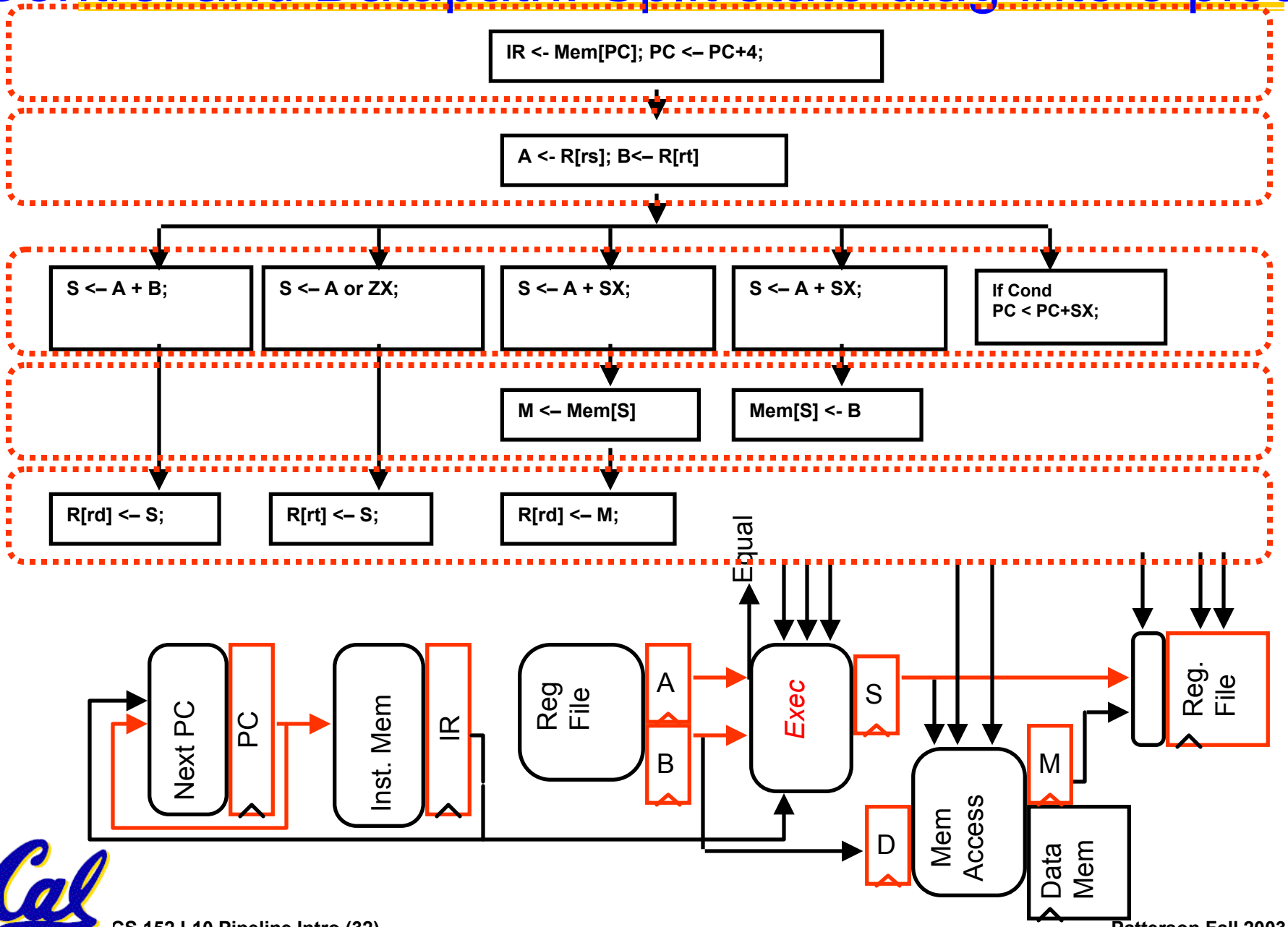
Forwarding (or Bypassing): What about Loads

- **Dependencies backwards in time are hazards**

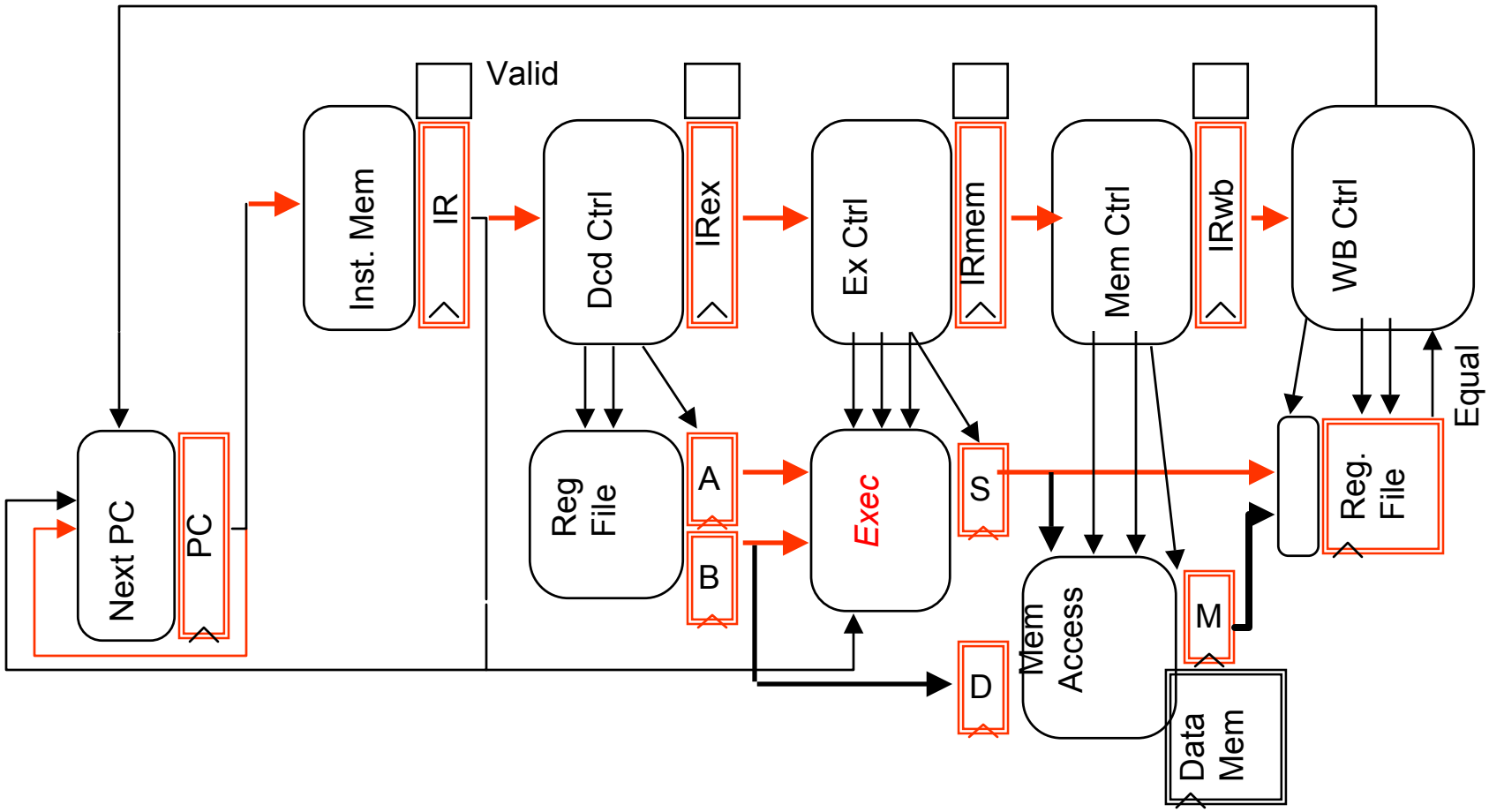


- **Can't solve with forwarding:**
- **Must delay/stall instruction dependent on loads**

Control and Datapath: Split state diag into 5 pieces

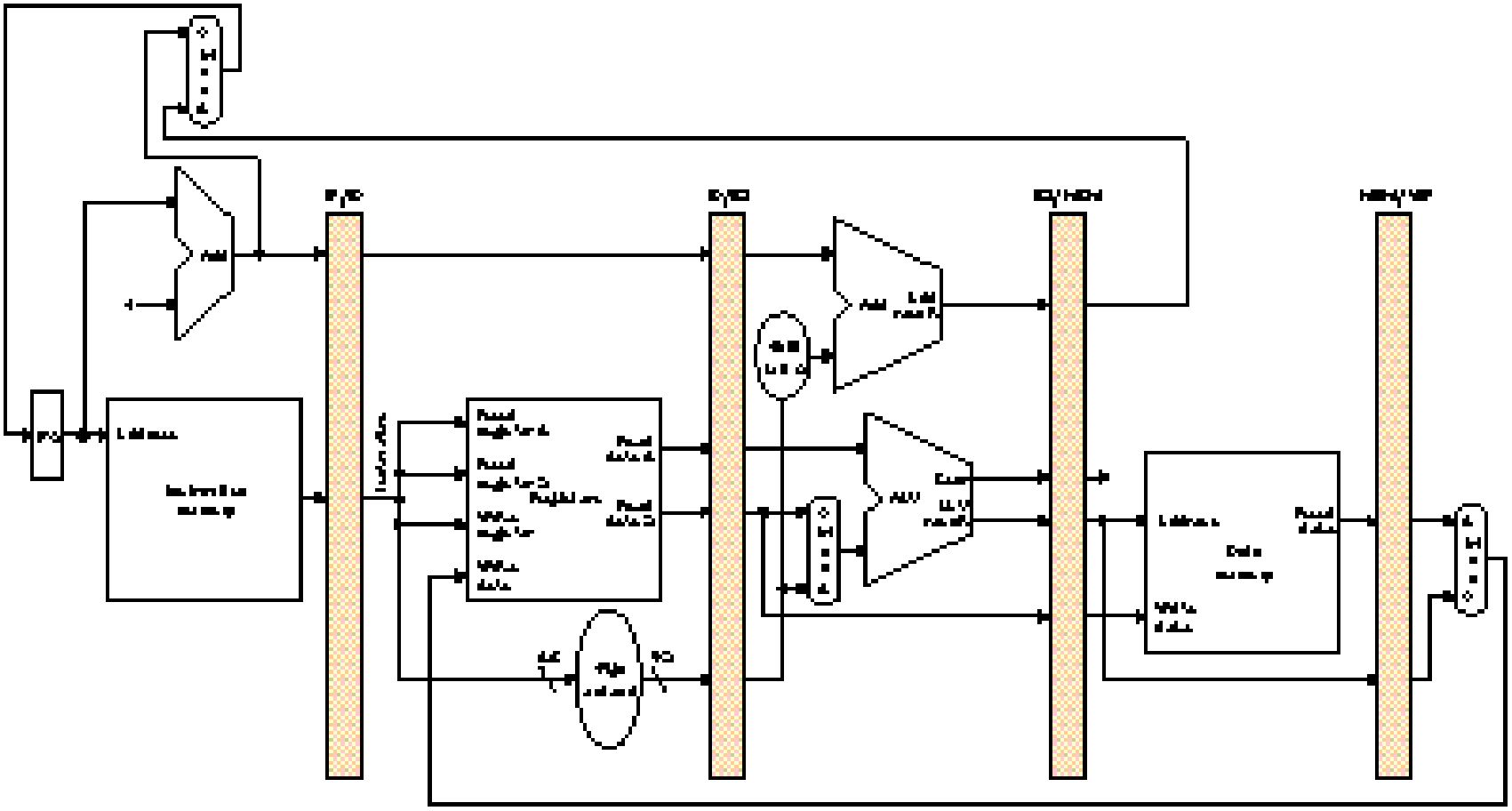


Pipelined Processor (almost) for slides

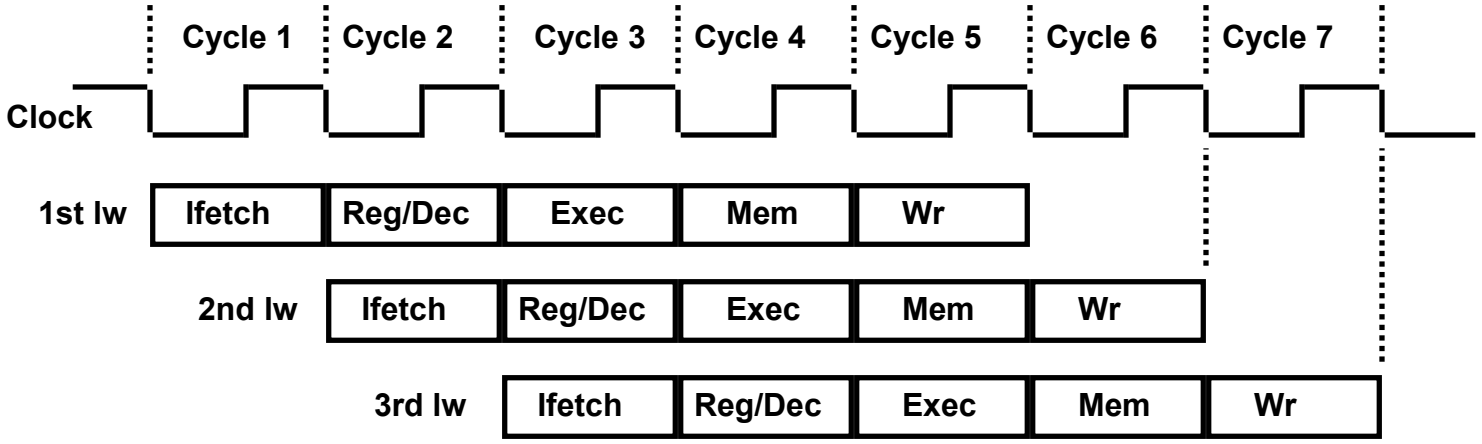


- What happens if we start a new instruction every cycle?

Pipelined Datapath (as in book); hard to read

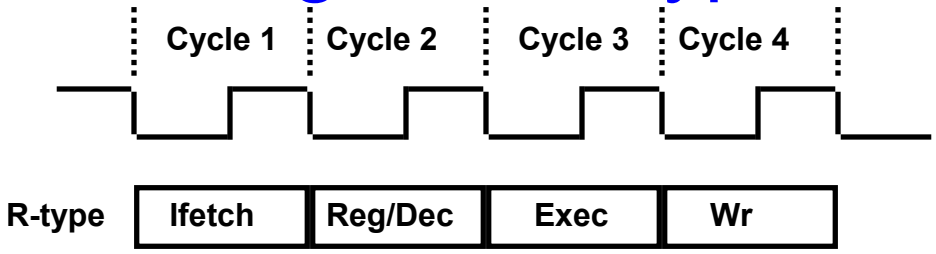


Pipelining the Load Instruction



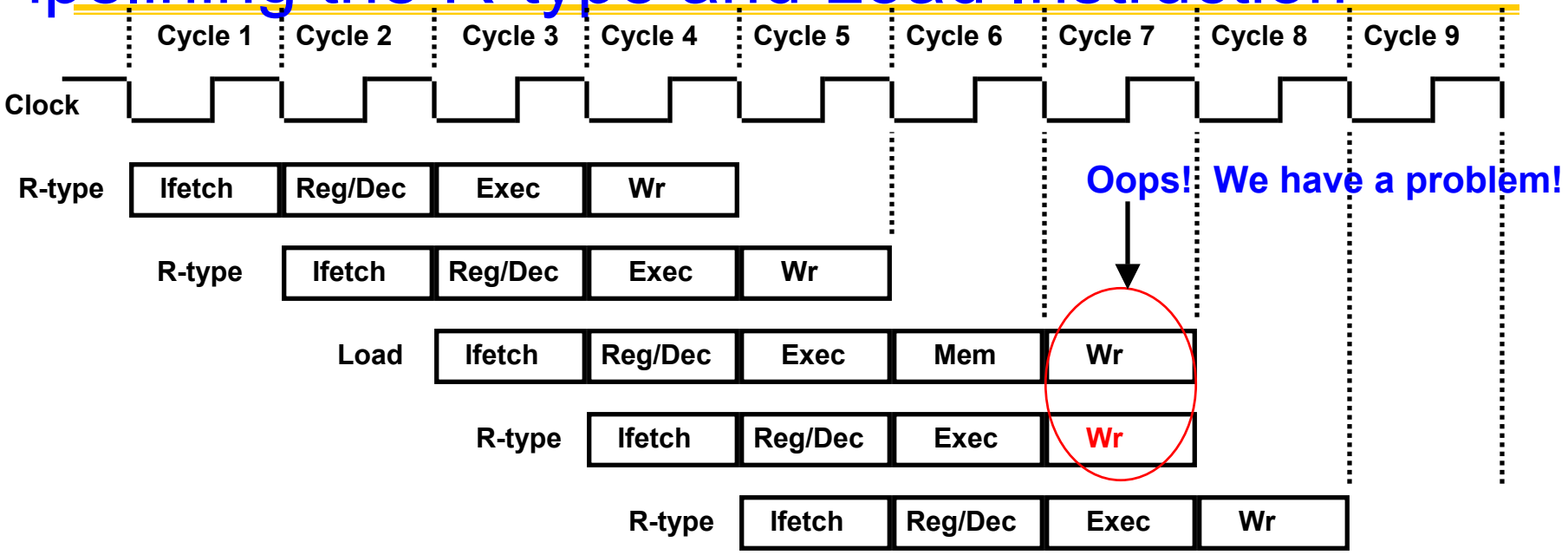
- The five independent functional units in the pipeline datapath are:
  - Instruction Memory for the Ifetch stage
  - Register File's Read ports (bus A and busB) for the Reg/Dec stage
  - ALU for the Exec stage
  - Data Memory for the Mem stage
  - Register File's Write port (bus W) for the Wr stage

The Four Stages of R-type



- Ifetch: Instruction Fetch
  - Fetch the instruction from the Instruction Memory
- Reg/Dec: Registers Fetch and Instruction Decode
- Exec:
  - ALU operates on the two register operands
  - Update PC
- Wr: Write the ALU output back to the register file

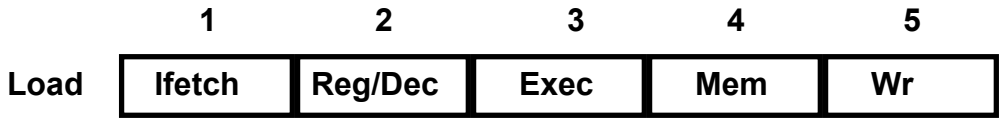
Pipelining the R-type and Load Instruction



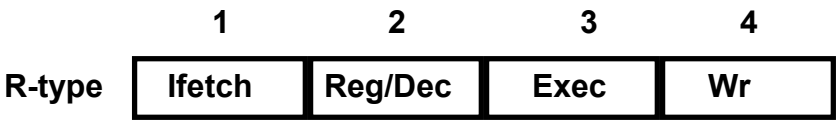
- We have pipeline conflict or structural hazard:
  - Two instructions try to write to the register file at the same time!
  - Only one write port

Important Observation

- Each functional unit can only be used once per instruction
- Each functional unit must be used at the same stage for all instructions:
  - Load uses Register File's Write Port during its 5th stage

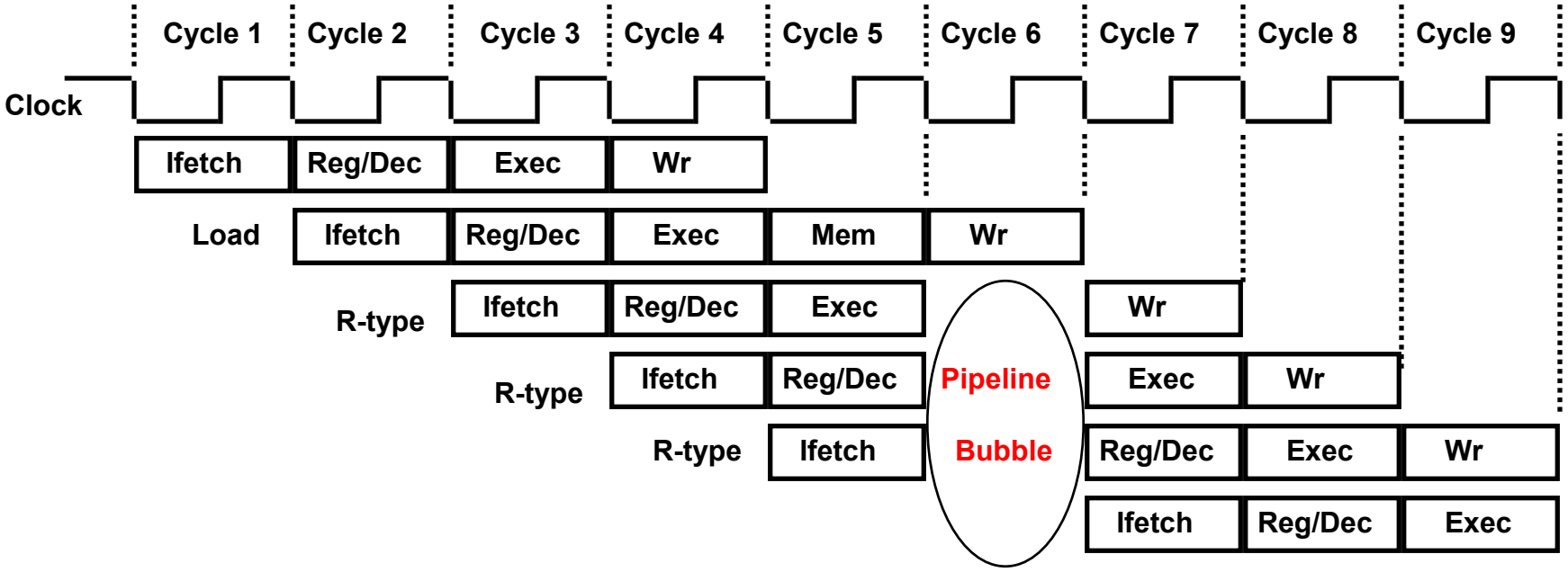


  - R-type uses Register File's Write Port during its 4th stage



° 2 ways to solve this pipeline hazard.

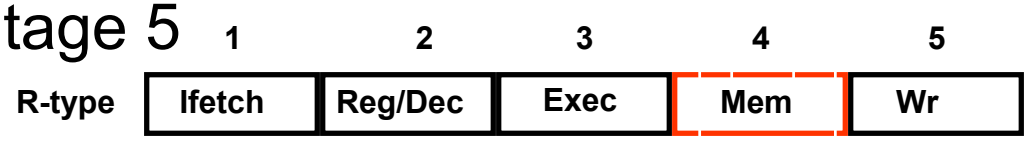
Solution 1: Insert "Bubble" into the Pipeline



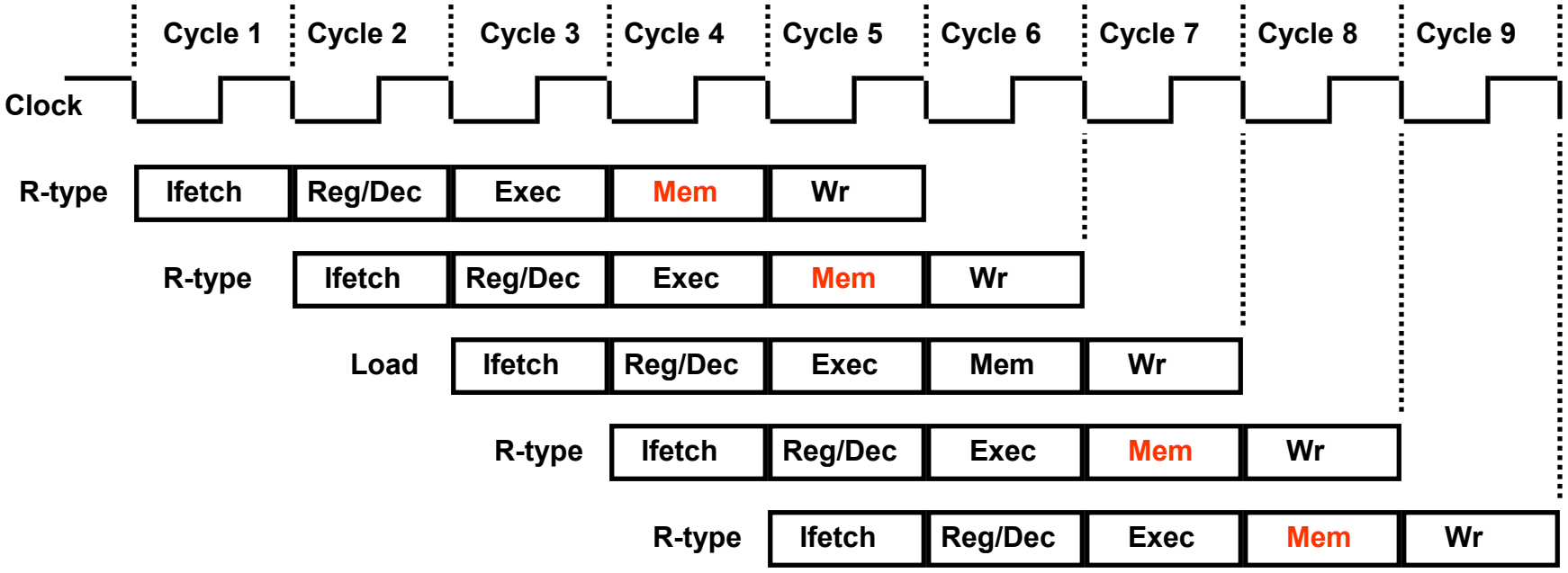
- Insert a "bubble" into the pipeline to prevent 2 writes at the same cycle
  - The control logic can be complex.
  - Lose instruction fetch and issue opportunity.
- No instruction is started in Cycle 6!

Solution 2: Delay R-type's Write by One Cycle

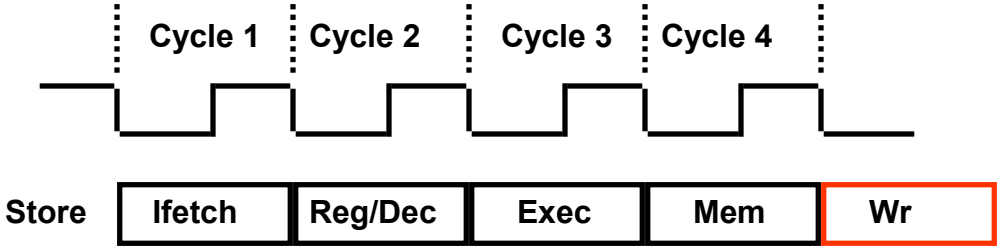
- Delay R-type's register write by one cycle:
  - Now R-type instructions also use Reg File's write port at Stage 5



  - Mem stage is a NOP stage: nothing is being done.

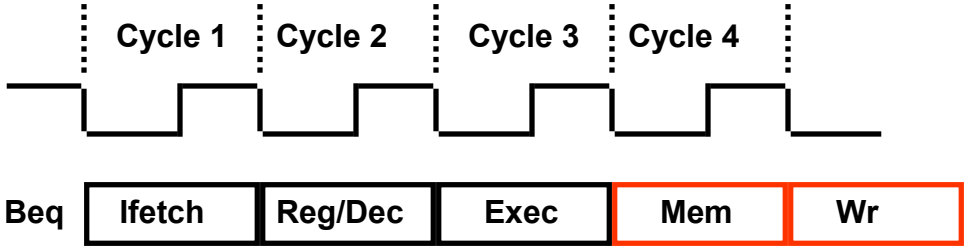


The Four Stages of Store => 5 stages



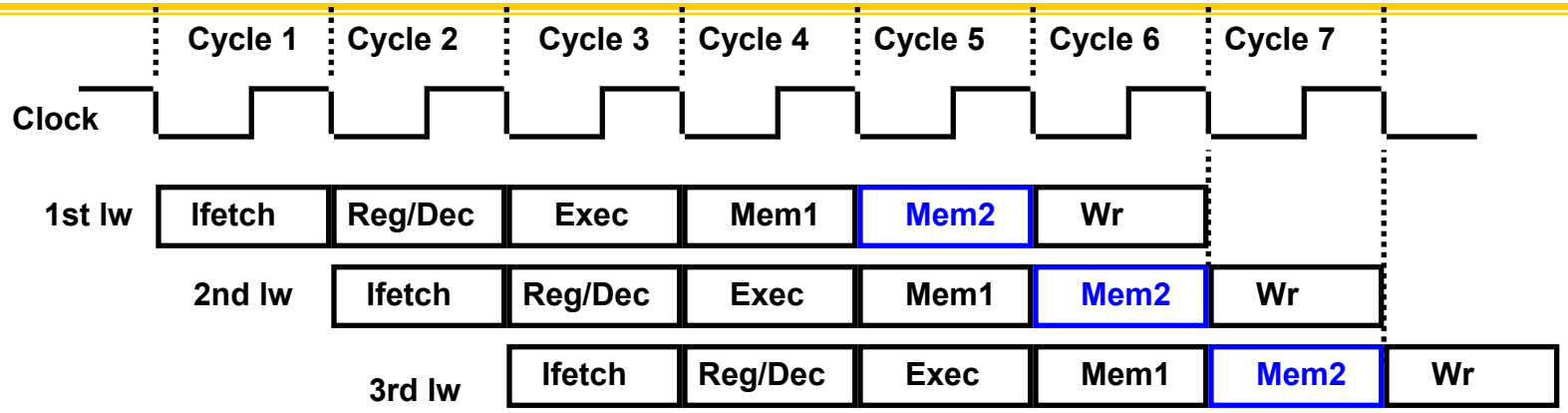
- Ifetch: Instruction Fetch
  - Fetch the instruction from the Instruction Memory
- Reg/Dec: Registers Fetch and Instruction Decode
- Exec: Calculate the memory address
- Mem: Write the data into the Data Memory

The Three Stages of Beq => 5 stages



- Ifetch: Instruction Fetch
  - Fetch the instruction from the Instruction Memory
- Reg/Dec:
  - Registers Fetch and Instruction Decode
- Exec:
  - compares the two register operand,
  - select correct branch target address
  - latch into PC

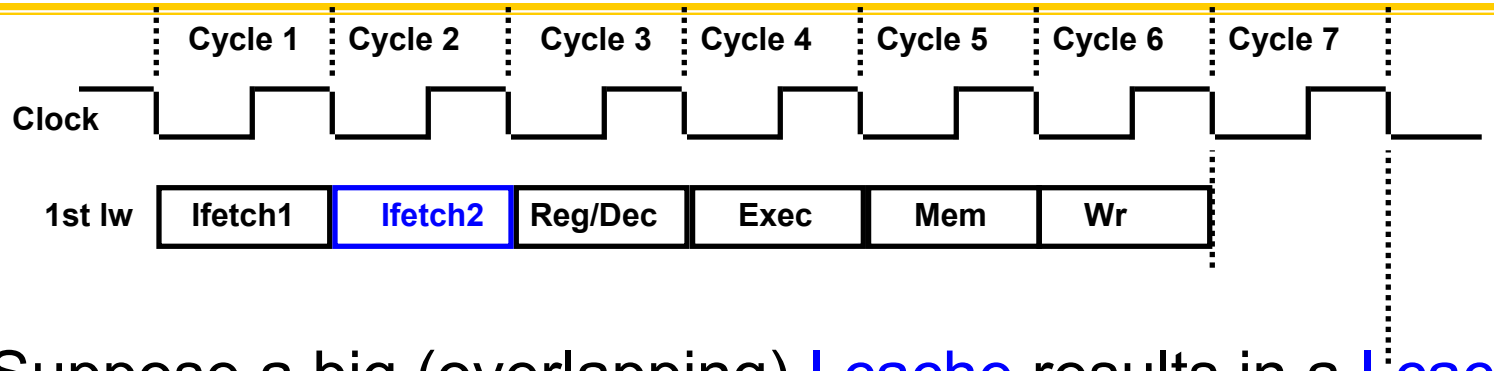
Peer Instruction



- Suppose a big (overlapping) data cache results in a data cache latency of 2 clock cycles and a 6-stage pipeline. What is the impact?

1. Instruction bandwidth is now 5/6-ths of the 5-stage pipeline
2. Instruction bandwidth is now 1/2 of the 5-stage pipeline
3. The branch delay slot is now 2 instructions
4. The load-use hazard can be with 2 instructions following load
5. Both 3 and 4: branch delay and load-use now 2 instructions
6. None of the above

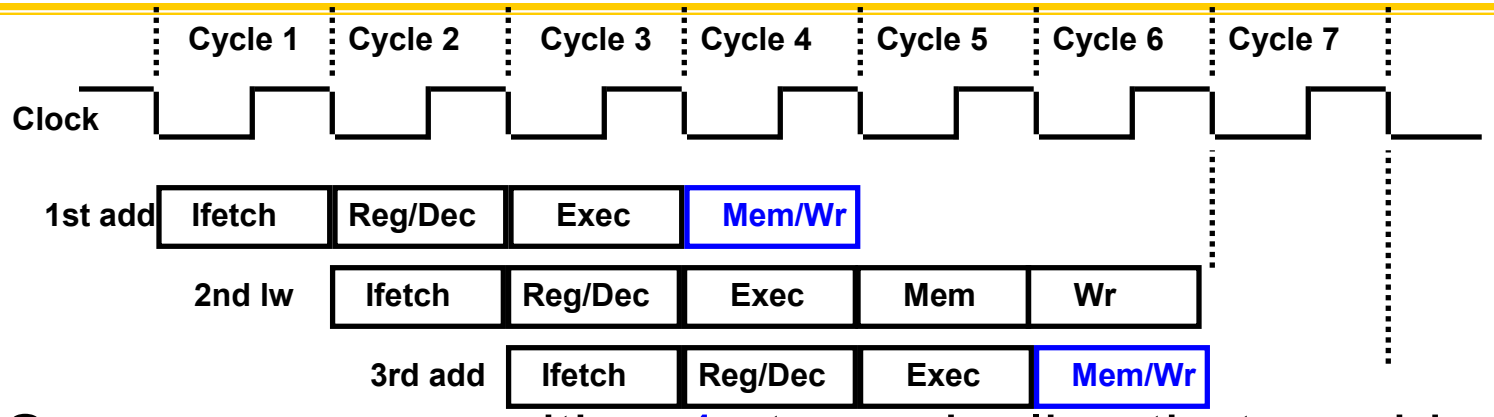
Peer Instruction



- Suppose a big (overlapping) I cache results in a I cache latency of 2 clock cycles and a 6-stage pipeline. What is the impact?

1. Instruction bandwidth is now 5/6-ths of the 5-stage pipeline
2. Instruction bandwidth is now 1/2 of the 5-stage pipeline
3. The branch delay slot is now 2 instructions
4. The load-use hazard can be with 2 instructions following load
5. Both 3 and 4: branch delay and load-use now 2 instructions
6. None of the above

Peer Instruction



- Suppose we use with a 4 stage pipeline that combines memory access and write back stages for all instructions but load, stalling when there are structural hazards. Impact?

1. The branch delay slot is now 0 instructions
2. Every load stalls since it has a structural hazard
3. Every store stalls since it has a structural hazard
4. Both 2 & 3: loads & stores stall due to structural hazards
5. Every load stalls, but there is no load-use hazard anymore
6. Both 2 & 3, but there is no load-use hazard anymore
7. None of the above

Designing a Pipelined Processor

- Go back and examine your datapath and control diagram
- Associate resources with states
- Ensure that backwards flows do not conflict, or figure out how to resolve
- Assert control in appropriate stage

Summary: Pipelining

- Reduce CPI by overlapping many instructions
  - Average throughput of approximately 1 CPI with fast clock
- Utilize capabilities of the Datapath
  - start next instruction while working on the current one
  - limited by length of longest stage (plus fill/flush)
  - detect and resolve hazards
- What makes it easy
  - all instructions are the same length
  - just a few instruction formats
  - memory operands appear only in loads and stores
- What makes it hard?
  - structural hazards: suppose we had only one memory
  - control hazards: need to worry about branch instructions
  - data hazards: an instruction depends on a previous instruction