UCB CS61C : Machine Structures

Lecture 28 – CPU Design : Pipelining to Improve Performance
2010-04-05

Lecturer SOE
Dan Garcia

CATCHING HARDWARE BUGS FASTER

Stanford Researchers have invented a monitoring technique called "Instruction Footprint Recording and Analysis" (IFRA) that collects info about the hardware when it's actually running (as opposed to very slow simulations) to help pinpoint hardware errors. When errors are detected, it takes a "snapshot" of the current state to help reproduce it. It can locate 96% of bugs, 80% w/time & location.

www.technologyreview.com/computing/24933

Review: Single cycle datapath

- **5 steps to design a processor**
  1. Analyze instruction set ⇒ datapath **requirements**
  2. **Select** set of datapath components & establish clock methodology
  3. **Assemble** datapath meeting the requirements
  4. **Analyze** implementation of each instruction to determine setting of control points that effects the register transfer.
  5. **Assemble** the control logic

- **Control is the hard part**

- **MIPS makes that easier**
  - Instructions same size
  - Source registers always in same place
  - Immediates same size, location
  - Operations always on registers/immediates

```
Processor                    Input
  Control
                   Memory
  Datapath                   Output
```
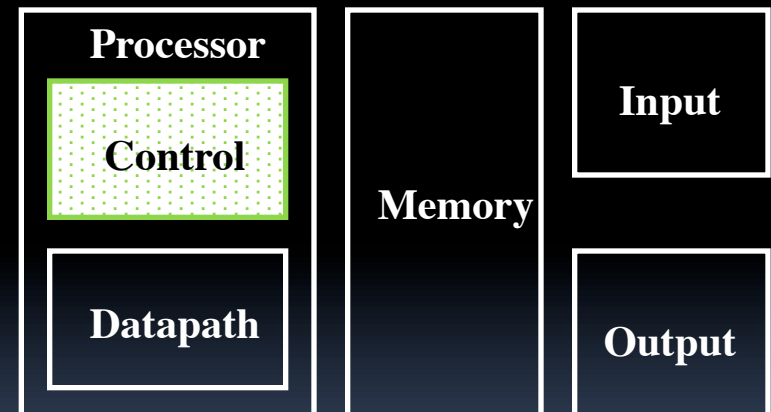
How We Build The Controller

opcode func

RegDst   = add + sub
ALUSrc   = ori + lw + sw
MemtoReg = lw
RegWrite   = add + sub + ori + lw
MemWrite = sw
nPCsel   = beq
Jump   = jump
ExtOp   = lw + sw
ALUctr[0] = sub + beq   (assume ALUctr is 0 ADD, 01: SUB, 10: OR)
ALUctr[1] = or
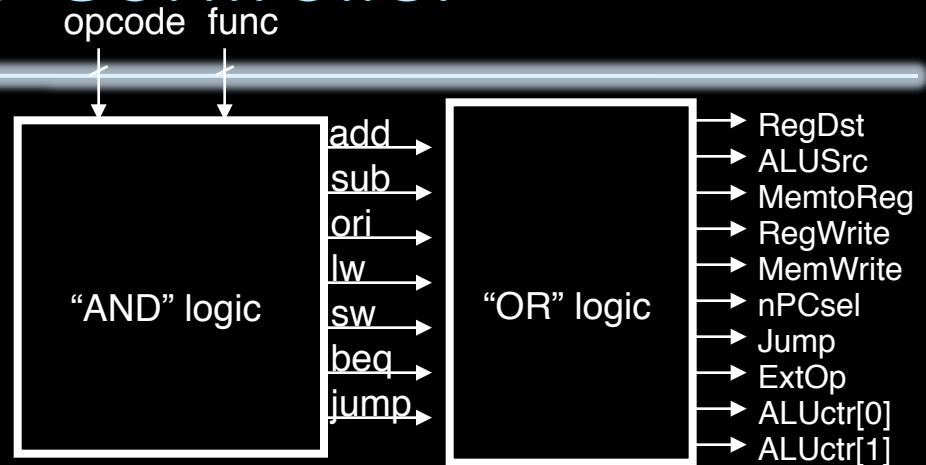
where,

$rtype = \sim op_5 \bullet \sim op_4 \bullet \sim op_3 \bullet \sim op_2 \bullet \sim op_1 \bullet \sim op_0,$
$ori \quad = \sim op_5 \bullet \sim op_4 \bullet op_3 \bullet op_2 \bullet \sim op_1 \bullet op_0$
$lw \quad = op_5 \bullet \sim op_4 \bullet \sim op_3 \bullet \sim op_2 \bullet op_1 \bullet op_0$
$sw \quad = op_5 \bullet \sim op_4 \bullet op_3 \bullet \sim op_2 \bullet op_1 \bullet op_0$
$beq \quad = \sim op_5 \bullet \sim op_4 \bullet \sim op_3 \bullet op_2 \bullet \sim op_1 \bullet \sim op_0$
$jump = \sim op_5 \bullet \sim op_4 \bullet \sim op_3 \bullet \sim op_2 \bullet op_1 \bullet \sim op_0$

$add = rtype \bullet func_5 \bullet \sim func_4 \bullet \sim func_3 \bullet \sim func_2 \bullet \sim func_1 \bullet \sim func_0$
$sub = rtype \bullet func_5 \bullet \sim func_4 \bullet \sim func_3 \bullet \sim func_2 \bullet func_1 \bullet \sim func_0$

"AND" logic

add
sub
ori
lw
sw
beq
jump

"OR" logic

RegDst
ALUSrc
MemtoReg
RegWrite
MemWrite
nPCsel
Jump
ExtOp
ALUctr[0]
ALUctr[1]

Omigosh omigosh,
do you know what
this means?

Call home, we've made HW/SW contact!

High Level Language Program (e.g., C)

temp = v[k];
v[k] = v[k+1];
v[k+1] = temp;

Compiler

Assembly Language Program (e.g., MIPS)

lw $t0, 0($2)
lw $t1, 4($2)
sw $t1, 0($2)
sw $t0, 4($2)

Assembler

Machine Language Program (MIPS)

0000 1001 1100 0110 1010 1111 0101 1000
1010 1111 0101 1000 0000 1001 1100 0110
1100 0110 1010 1111 0101 1000 0000 1001
0101 1000 0000 1001 1100 0110 1010 1111

Machine Interpretation

Hardware Architecture Description (e.g., block diagrams)

Register File

ALU

Architecture Implementation

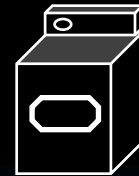**Logic Circuit Description (Circuit Schematic Diagrams)**
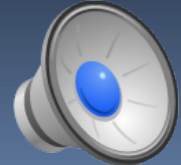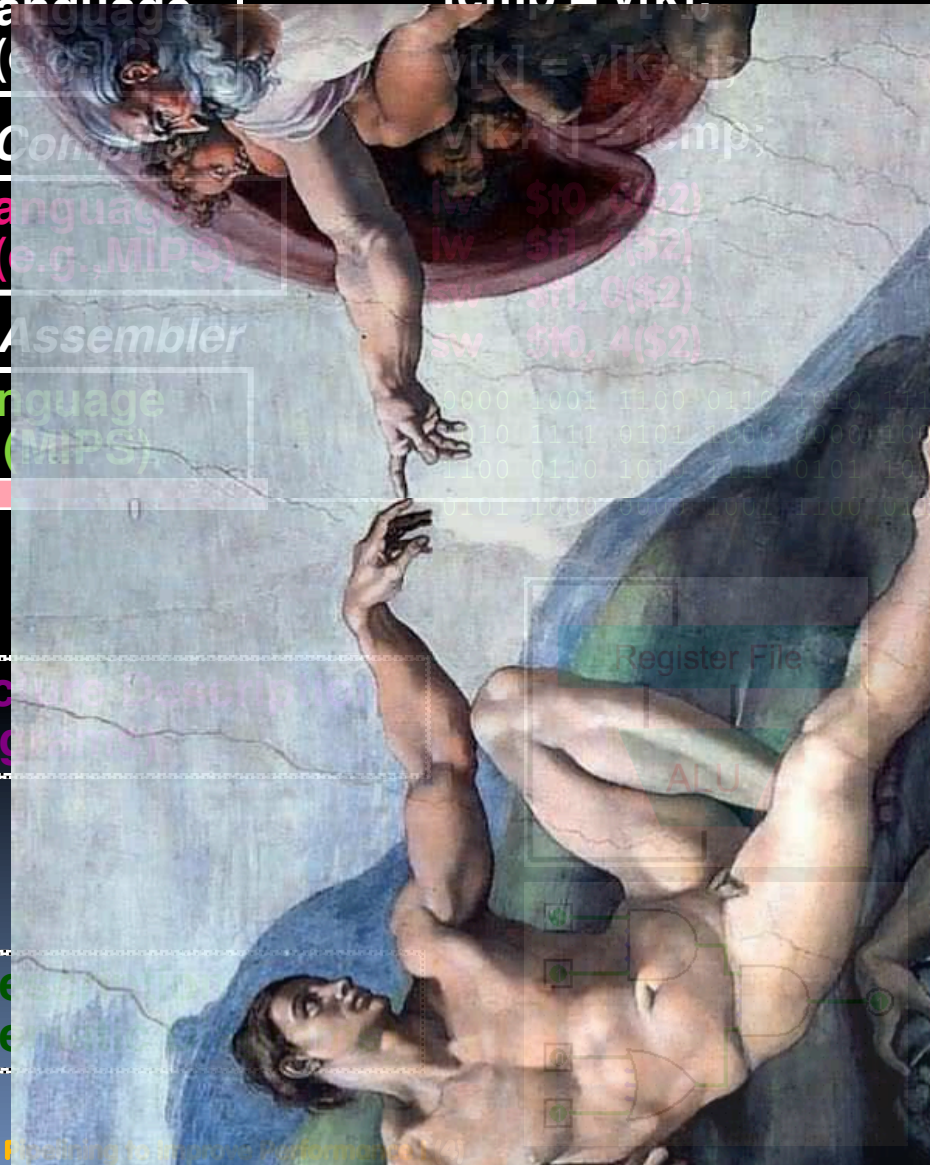
Processor Performance

- **Can we estimate the clock rate (frequency) of our single-cycle processor? We know:**
  - **1 cycle per instruction**
  - **`lw` is the most demanding instruction.**
  - **Assume these delays for major pieces of the datapath:**
    - **Instr. Mem, ALU, Data Mem : 2ns each, regfile 1ns**
    - **Instruction execution requires: 2 + 1 + 2 + 2 + 1 = 8ns**
    - **⇒ 125 MHz**

- **What can we do to improve clock rate?**

- **Will this improve performance as well?**
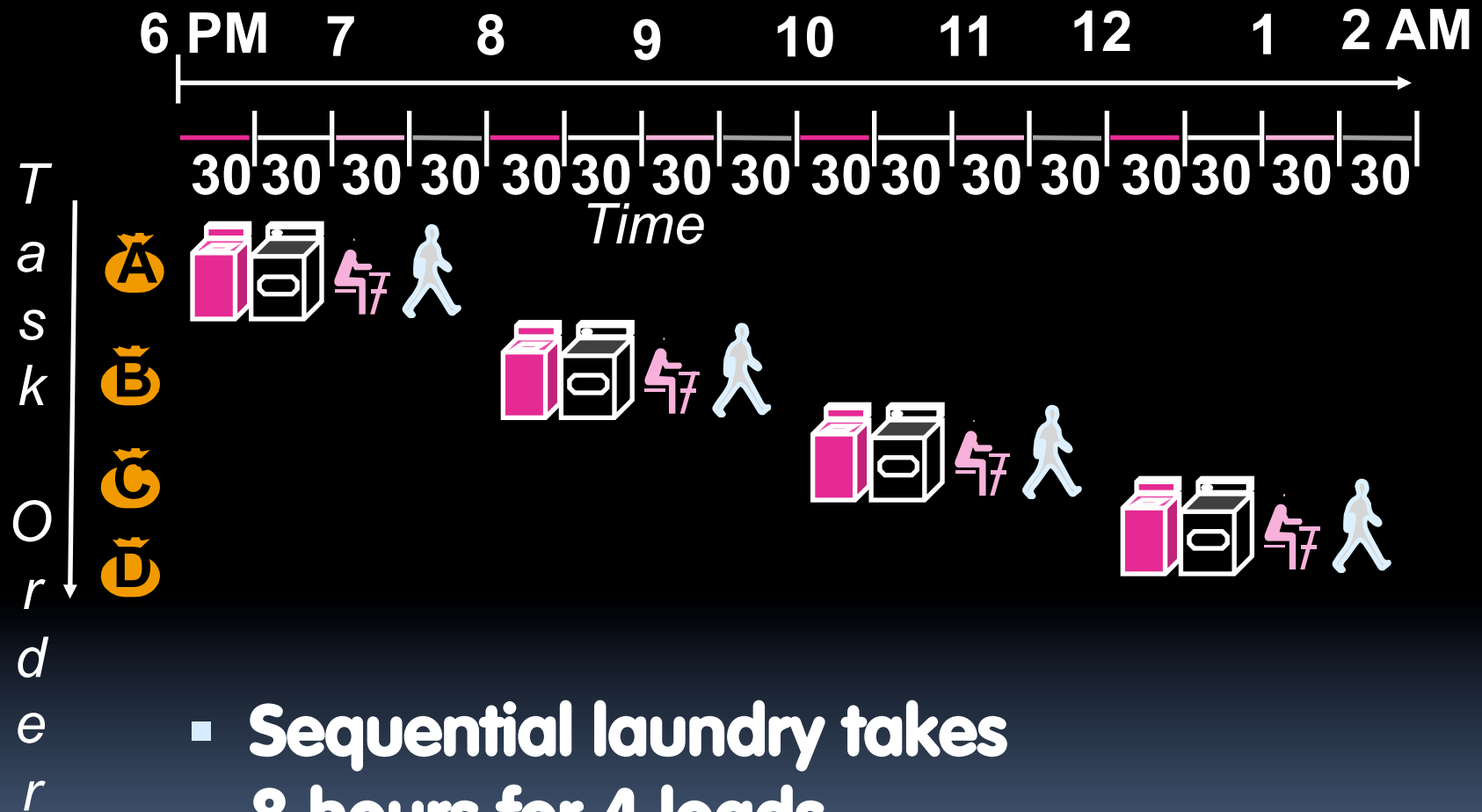 - **We want increases in clock rate to result in programs executing quicker.**
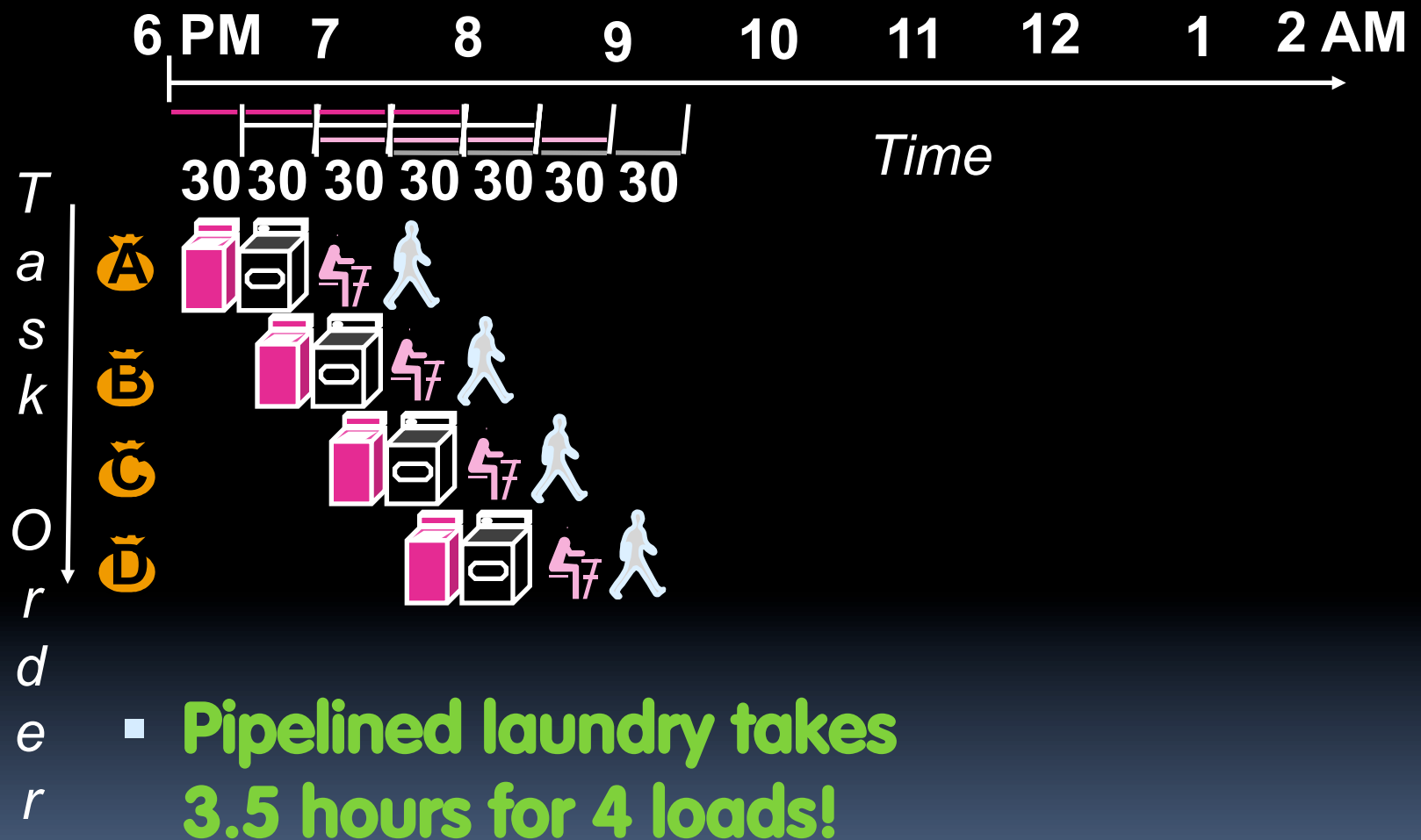
Gotta Do Laundry

- **Ann, Brian, Cathy, Dave each have one load of clothes to wash, dry, fold, and put away**

  - **Washer takes 30 minutes**

  - **Dryer takes 30 minutes**

  - **"Folder" takes 30 minutes**

  - **"Stasher" takes 30 minutes to put clothes into drawers**

Sequential Laundry

- **Sequential laundry takes 8 hours for 4 loads**

Pipelined Laundry

6 PM   7   8   9   10   11   12   1   2 AM

Time

30 30 30 30 30 30 30

T
a
s
k

A

B

O
r
d
e
r

C

D

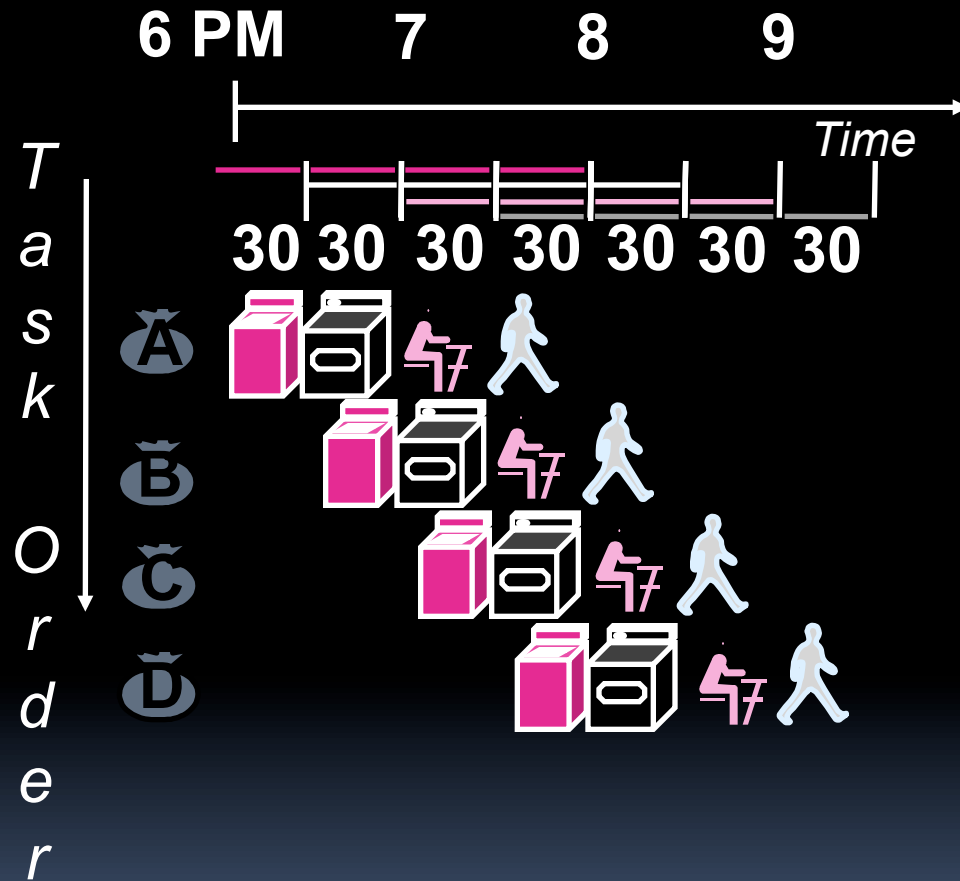- **Pipelined laundry takes 3.5 hours for 4 loads!**

General Definitions

- **Latency**: time to completely execute a certain task
  - for example, time to read a sector from disk is disk access time or disk latency
- **Throughput**: amount of work that can be done over a period of time

Pipelining Lessons (1/2)

6 PM   7   8   9

Time

T
a
s
k

O
r
d
e
r

30 30 30 30 30 30 30

A

B

C

D

- **Pipelining doesn't help <u>latency</u> of single task, it helps <u>throughput</u> of entire workload**

- **<u>Multiple</u> tasks operating simultaneously using different resources**

- **Potential speedup = <u>Number pipe stages</u>**

- **Time to "<u>fill</u>" pipeline and time to "<u>drain</u>" it reduces speedup: 2.3X v. 4X in this example**

Pipelining Lessons (2/2)
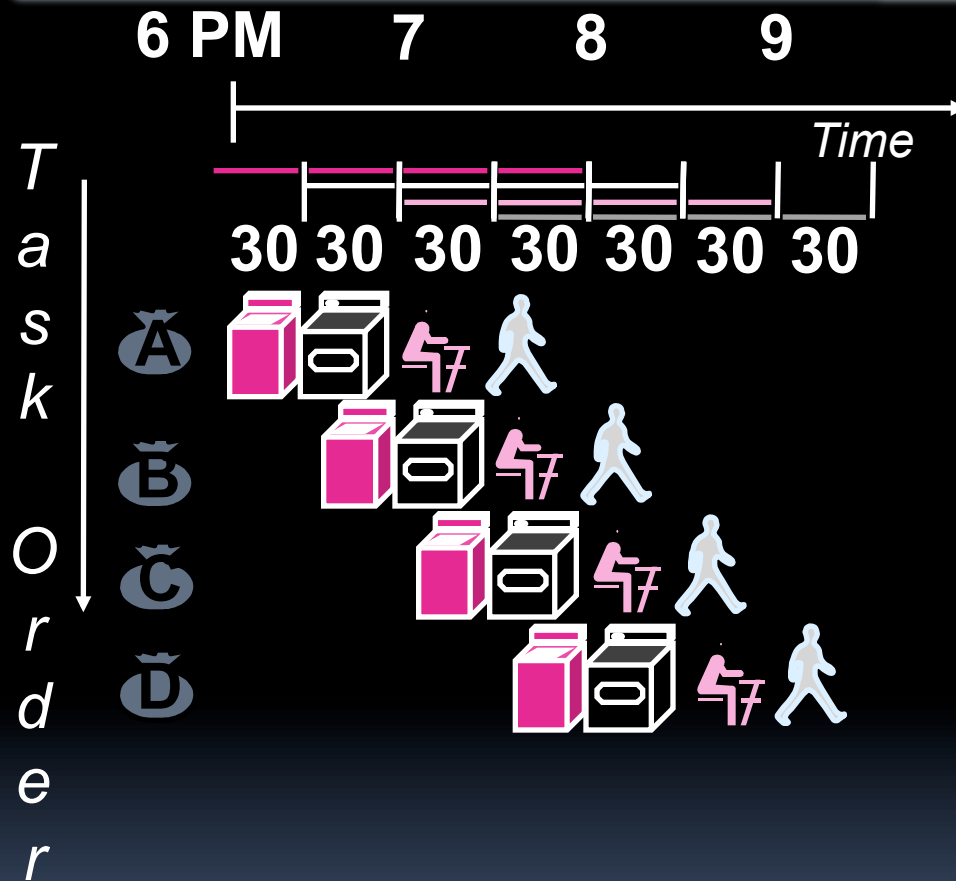
- **Suppose new Washer takes 20 minutes, new Stasher takes 20 minutes. How much faster is pipeline?**

- **Pipeline rate limited by <u>slowest</u> pipeline stage**

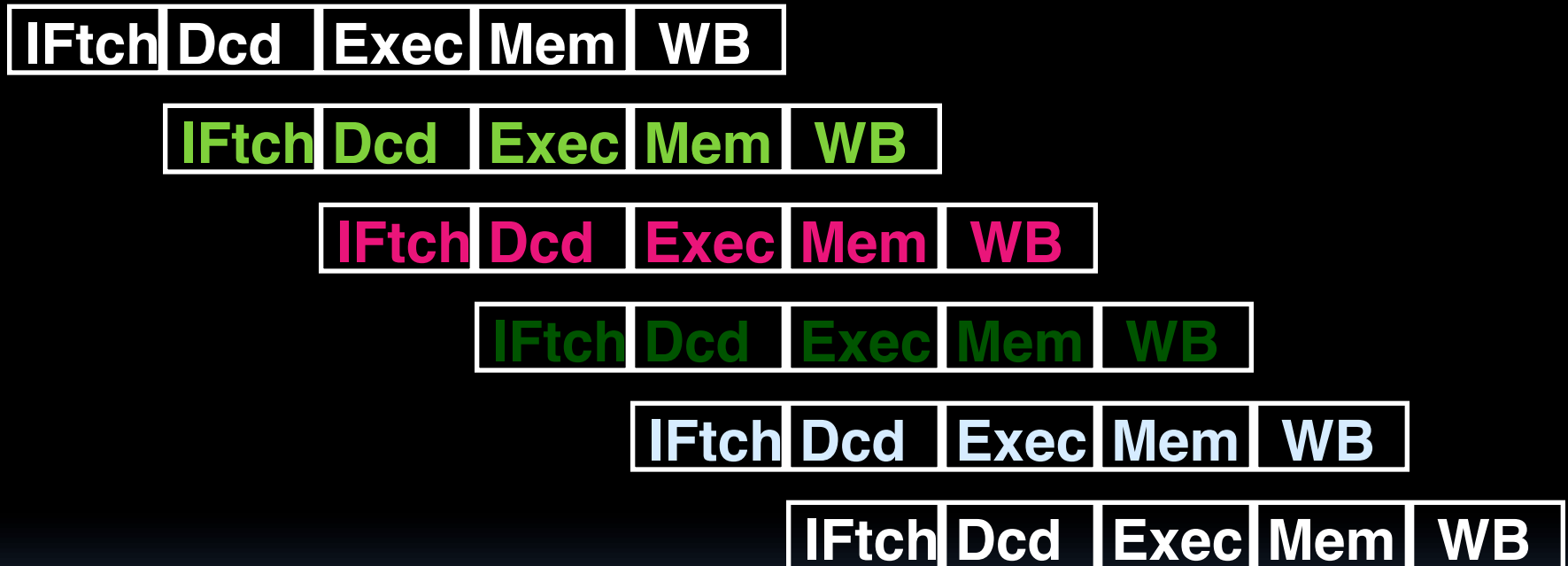- **Unbalanced lengths of pipe stages reduces speedup**

Steps in Executing MIPS

1) **IFtch**: **I**nstruction **F**e**tch**, Increment PC

2) **Dcd**: **I**nstruction **Dec**o**d**e, Read Registers

3) **Exec**:

   Mem-ref: Calculate Address

   Arith-log: Perform Operation

4) **Mem**:

   Load: Read Data from Memory

   Store: Write Data to Memory

5) **WB**: **W**rite Data **B**ack to Register

Pipelined Execution Representation

Time

IFtch	Dcd	Exec	Mem	WB

	IFtch	Dcd	Exec	Mem	WB

		IFtch	Dcd	Exec	Mem	WB

			IFtch	Dcd	Exec	Mem	WB

				IFtch	Dcd	Exec	Mem	WB

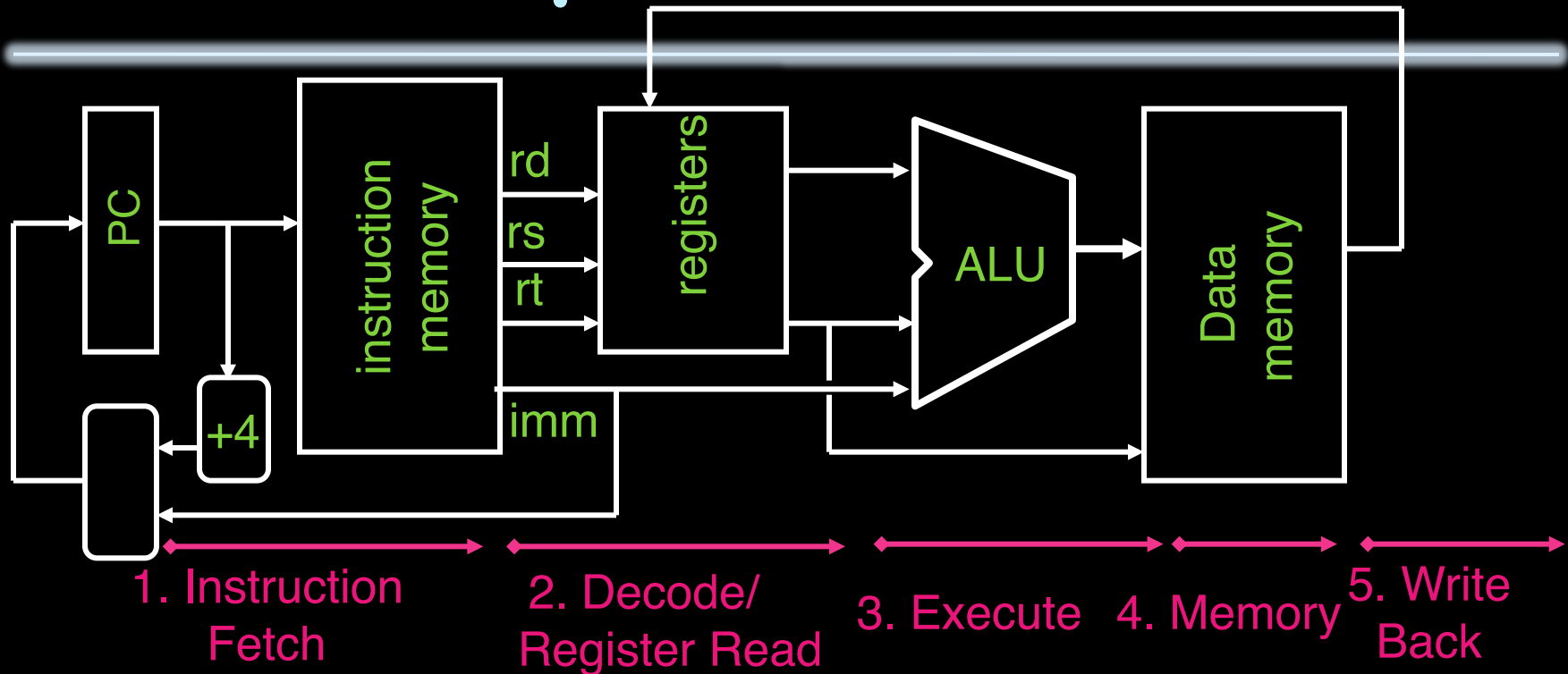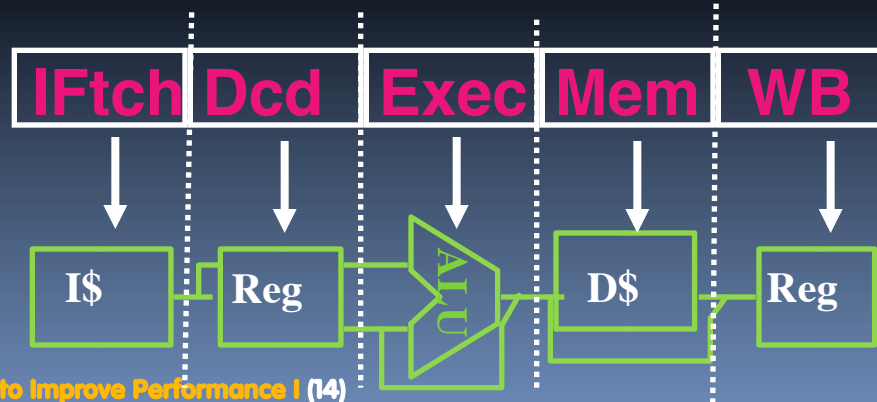|					IFtch	Dcd	Exec	Mem	WB

- **Every instruction must take same number of steps, also called pipeline "stages", so some will go idle sometimes**

Review: Datapath for MIPS

PC | instruction memory | rd / rs / rt / imm | registers | ALU | Data memory

1. Instruction Fetch
2. Decode/ Register Read
3. Execute
4. Memory
5. Write Back

- **Use datapath figure to represent pipeline**

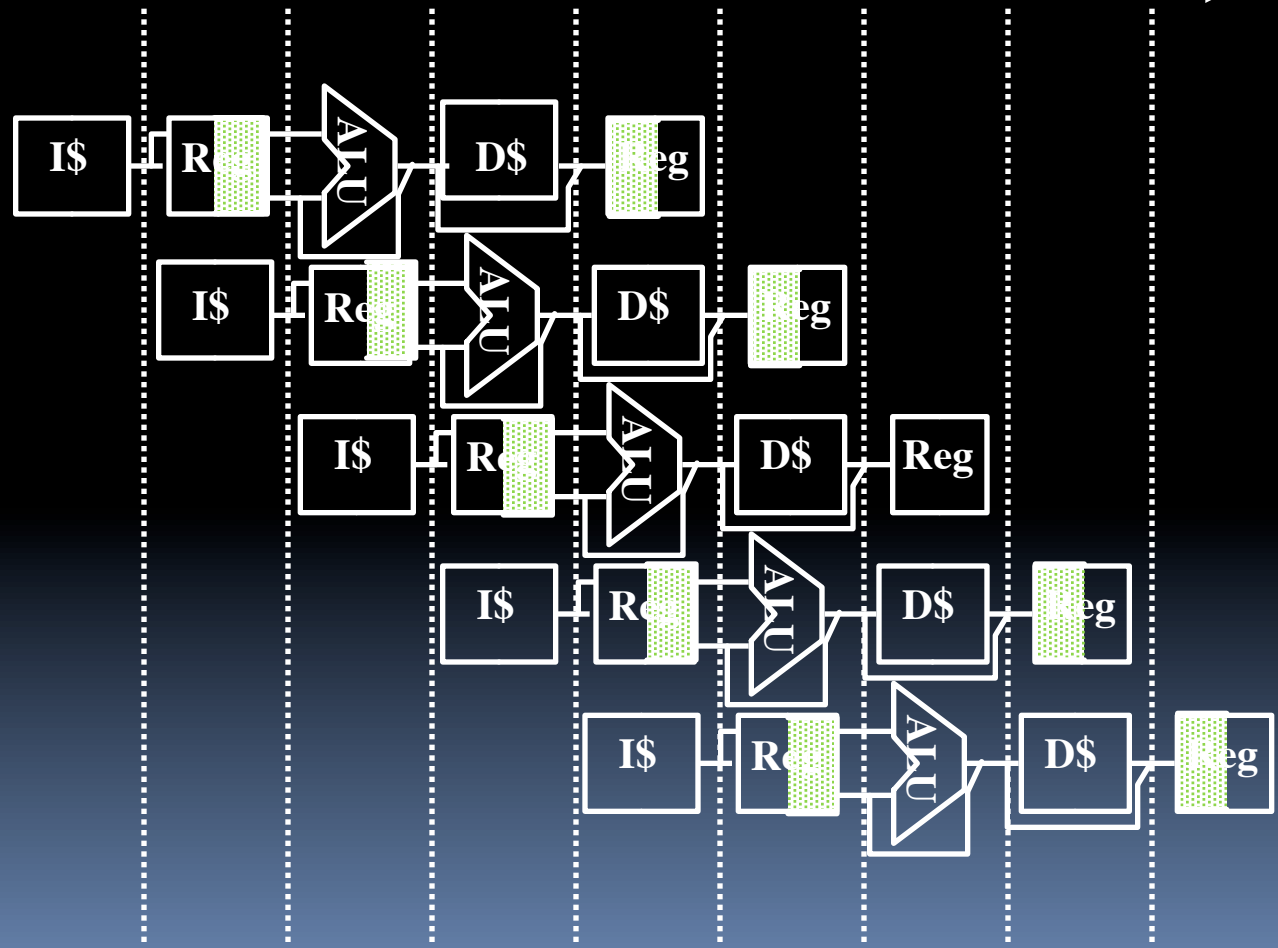| **IFtch** | **Dcd** | **Exec** | **Mem** | **WB** |

I$ | Reg | ALU | D$ | Reg

Graphical Pipeline Representation

(In Reg, right half highlight read, left half write)

Time (clock cycles)

Instr. Order

- Load
- Add
- Store
- Sub
- Or

Example

- **Suppose 2 ns for memory access, 2 ns for ALU operation, and 1 ns for register file read or write; compute instruction rate**

- **Nonpipelined Execution:**

  - `lw` : **IF + Read Reg + ALU + Memory + Write Reg = 2 + 1 + 2 + 2 + 1 = 8 ns**

  - `add`: **IF + Read Reg + ALU + Write Reg = 2 + 1 + 2 + 1 = 6 ns**
    (recall 8ns for single-cycle processor)

- **Pipelined Execution:**

  - **Max(IF,Read Reg,ALU,Memory,Write Reg) = 2 ns**

Pipeline Hazard: Matching socks in later load
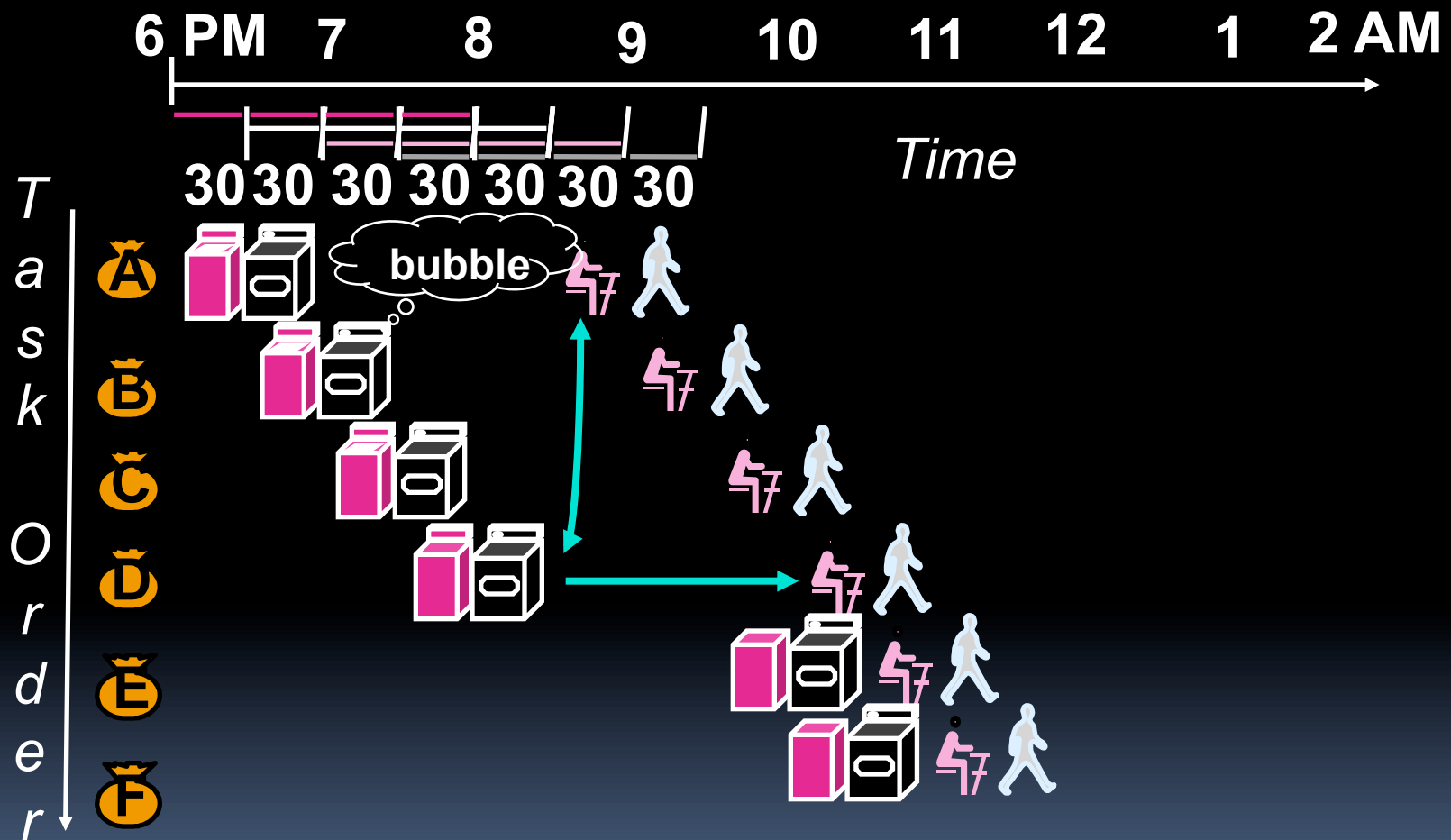
A depends on D; stall since folder tied up

# Administrivia

- **Administrivia?**

Problems for Pipelining CPUs

- **Limits to pipelining: <u>Hazards</u> prevent next instruction from executing during its designated clock cycle**
  - <u>**Structural hazards**</u>**: HW cannot support some combination of instructions (single person to fold and put clothes away)**
  - <u>**Control hazards**</u>**: Pipelining of branches causes later instruction fetches to wait for the result of the branch**
  - <u>**Data hazards**</u>**: Instruction depends on result of prior instruction still in the pipeline (missing sock)**
- **These might result in pipeline stalls or "bubbles" in the pipeline.**

Structural Hazard #1: Single Memory (1/2)

Time (clock cycles)

Instr. Order

Load

Instr 1

Instr 2

Instr 3

Instr 4
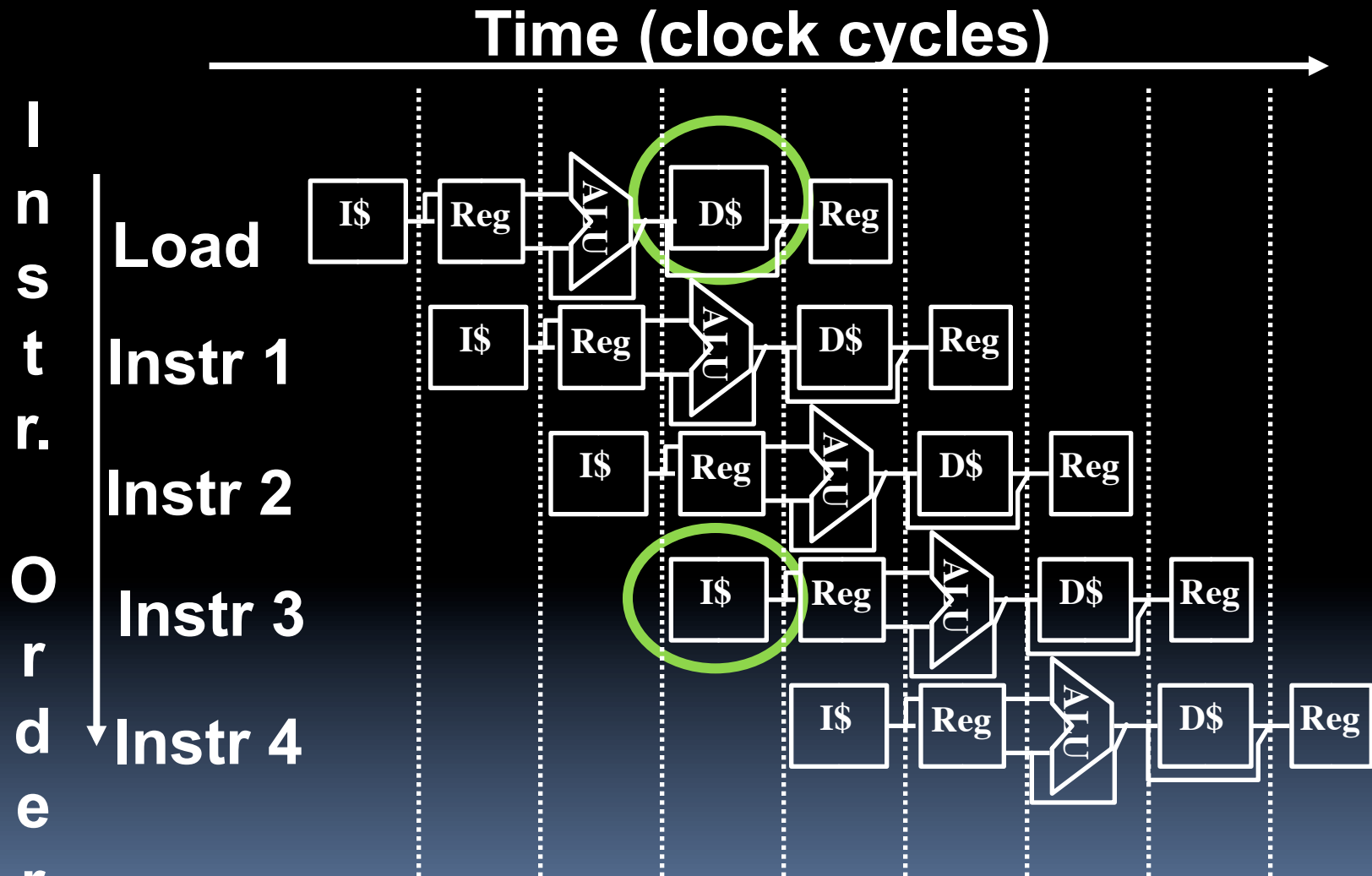
Read same memory twice in same clock cycle

Structural Hazard #1: Single Memory (2/2)

- ## Solution:
  - **infeasible and inefficient to create second memory**
  - (We'll learn about this more next week)
  - **so simulate this by having two Level 1 Caches** (a temporary smaller [of usually most recently used] copy of memory)
  - **have both an L1 Instruction Cache and an L1 Data Cache**
  - **need more complex hardware to control when both caches miss**

Structural Hazard #2: Registers (1/2)
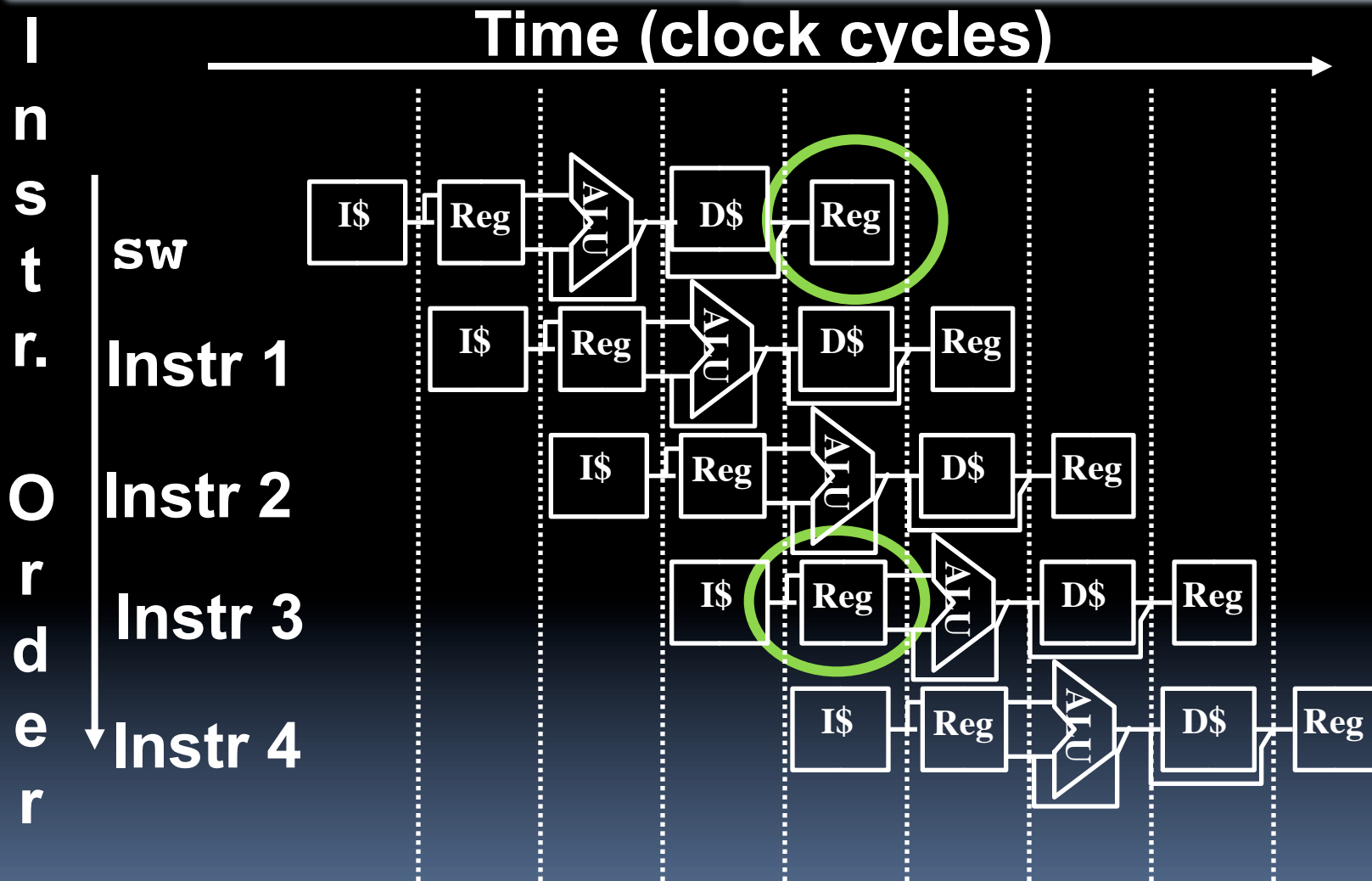
Can we read and write to registers simultaneously?

Structural Hazard #2: Registers (2/2)

- **Two different solutions have been used:**

  1) RegFile access is *VERY* fast: takes less than half the time of ALU stage

  - **Write to Registers during first half of each clock cycle**
  - **Read from Registers during second half of each clock cycle**

  2) Build RegFile with independent read and write ports

- **Result: can perform Read and Write during same clock cycle**

Peer Instruction

1) Thanks to pipelining, I have <u>reduced the time</u> it took me to wash my one shirt.

2) Longer pipelines are <u>always a win</u> (since less work per stage & a faster clock).

```
        12
a)   FF
b)   FT
c)   TF
d)   TT
```

Peer Instruction Answer

1) **Throughput** better, not execution time

2) **"…longer pipelines do usually mean faster clock, but branches cause problems!"**

1) Thanks to pipelining, I have reduced the time it took me to wash my one shirt.

2) Longer pipelines are always a win (since less work per stage & a faster clock).

FALSE
FALSE

	12
a)	FF
b)	FT
c)	TF
d)	TT

Things to Remember

- **Optimal Pipeline**
  - **Each stage is executing part of an instruction each clock cycle.**
  - **One instruction finishes during each clock cycle.**
  - **On average, execute far more quickly.**
- **What makes this work?**
  - **Similarities between instructions allow us to use same stages for all instructions (generally).**
  - **Each stage takes about the same amount of time as all others: little wasted time.**