

**Lecture 29 –
 Introduction to Pipelined Execution**



Lecturer PSOE Dan Garcia

www.cs.berkeley.edu/~ddgarcia

Bionic Eyes let blind see! ⇒

Johns Hopkins researchers have announced they have invented a “bionic eye” with a computer chip on the back of the eye and a small wireless video camera in a pair of glasses. Face recognition? Not yet, but soon!



Review: Single cycle datapath

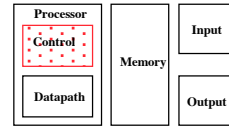
◦ **5 steps to design a processor**

- 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



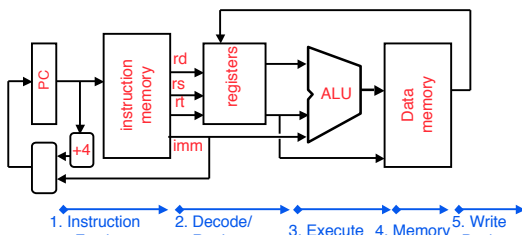
Review Datapath (1/3)

- Datapath is the hardware that performs operations necessary to execute programs.
- Control instructs datapath on what to do next.
- Datapath needs:
 - access to storage (general purpose registers and memory)
 - computational ability (ALU)
 - helper hardware (local registers and PC)

Review Datapath (2/3)

- Five stages of datapath (executing an instruction):
 1. Instruction Fetch (Increment PC)
 2. Instruction Decode (Read Registers)
 3. ALU (Computation)
 4. Memory Access
 5. Write to Registers
- **ALL instructions must go through ALL five stages.**

Review Datapath (3/3)



Gotta Do Laundry

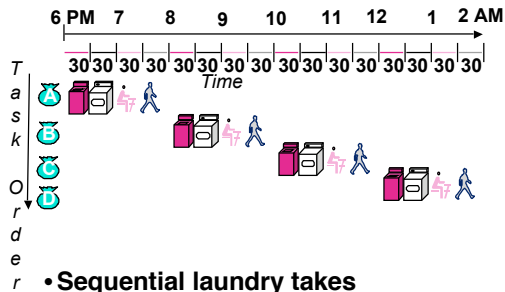
◦ Ann, Brian, Cathy, Dave each have one load of clothes to wash, dry, fold, and put away



- Washer takes 30 minutes
- Dryer takes 30 minutes
- “Folder” takes 30 minutes
- “Stasher” takes 30 minutes to put clothes into drawers



Sequential Laundry



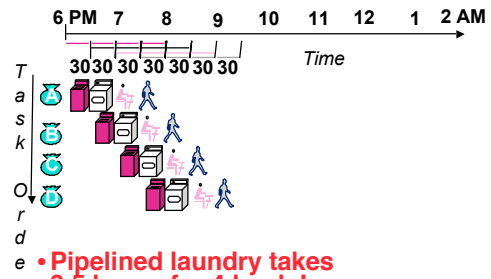
- Sequential laundry takes 8 hours for 4 loads



CS 61C L30 Introduction to Pipelined Execution (7)

Garcia, Fall 2004 © UCB

Pipelined Laundry



- Pipelined laundry takes 3.5 hours for 4 loads!



CS 61C L30 Introduction to Pipelined Execution (8)

Garcia, Fall 2004 © UCB

General Definitions

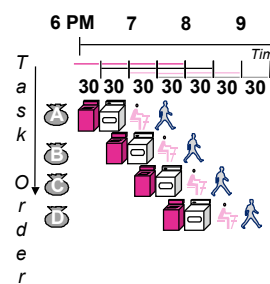
- **Latency**: time to completely execute a certain task
 - for example, time to read a sector from disk is disk access time or disk latency
- **Throughput**: amount of work that can be done over a period of time



CS 61C L30 Introduction to Pipelined Execution (9)

Garcia, Fall 2004 © UCB

Pipelining Lessons (1/2)



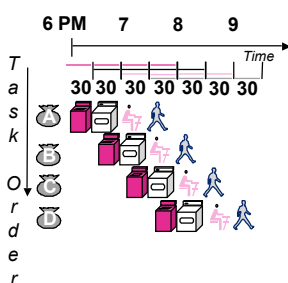
- Pipelining doesn't help **latency** of single task, it helps **throughput** of entire workload
- **Multiple** tasks operating simultaneously using different resources
- Potential speedup = **Number pipe stages**
- Time to "**fill**" pipeline and time to "**drain**" it reduces speedup: 2.3X v. 4X in this example



CS 61C L30 Introduction to Pipelined Execution (10)

Garcia, Fall 2004 © UCB

Pipelining Lessons (2/2)



- Suppose new Washer takes 20 minutes, new Stasher takes 20 minutes. How much faster is pipeline?
- Pipeline rate limited by **slowest** pipeline stage
- Unbalanced lengths of pipe stages also reduces speedup



CS 61C L30 Introduction to Pipelined Execution (11)

Garcia, Fall 2004 © UCB

Steps in Executing MIPS

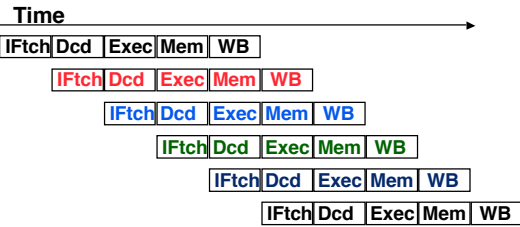
- 1) **IFetch**: Fetch Instruction, Increment PC
- 2) **Decode** Instruction, Read Registers
- 3) **Execute**:
Mem-ref: Calculate Address
Arith-log: Perform Operation
- 4) **Memory**:
Load: Read Data from Memory
Store: Write Data to Memory
- 5) **Write Back**: Write Data to Register



CS 61C L30 Introduction to Pipelined Execution (12)

Garcia, Fall 2004 © UCB

Pipelined Execution Representation



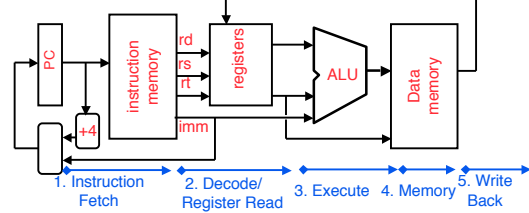
- Every instruction must take same number of steps, also called pipeline “**stages**”, so some will go idle sometimes



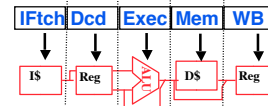
CS 61C L30 Introduction to Pipelined Execution (13)

Garcia, Fall 2004 © UCB

Review: Datapath for MIPS



- Use datapath figure to represent pipeline

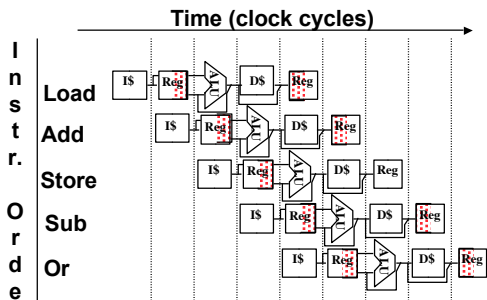


CS 61C L30 Introduction to Pipelined Execution (14)

Garcia, Fall 2004 © UCB

Graphical Pipeline Representation

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



CS 61C L30 Introduction to Pipelined Execution (15)

Garcia, Fall 2004 © UCB

Example

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

- Pipelined Execution:

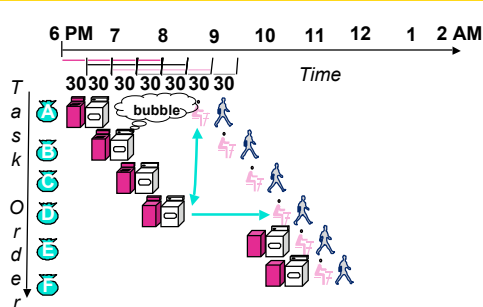
- Max(IF, Read Reg, ALU, Memory, Write Reg) = 2 ns



CS 61C L30 Introduction to Pipelined Execution (16)

Garcia, Fall 2004 © UCB

Pipeline Hazard: Matching socks in later load



A depends on D; **stall** since folder tied up



CS 61C L30 Introduction to Pipelined Execution (17)

Garcia, Fall 2004 © UCB

Problems for Computers

- Limits to pipelining: **Hazards** prevent next instruction from executing during its designated clock cycle

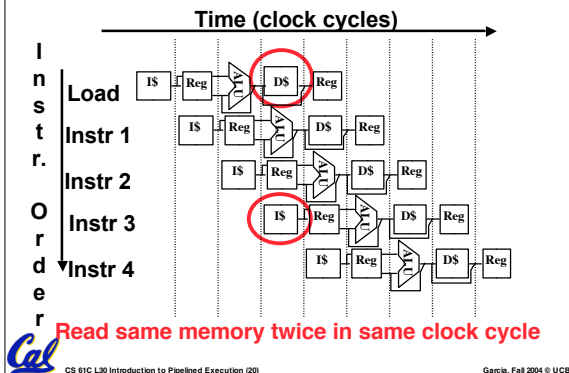
- **Structural hazards**: HW cannot support this combination of instructions (single person to fold and put clothes away)
- **Control hazards**: Pipelining of branches & other instructions **stall** the pipeline until the hazard; “**bubbles**” in the pipeline
- **Data hazards**: Instruction depends on result of prior instruction still in the pipeline (missing sock)



CS 61C L30 Introduction to Pipelined Execution (18)

Garcia, Fall 2004 © UCB

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

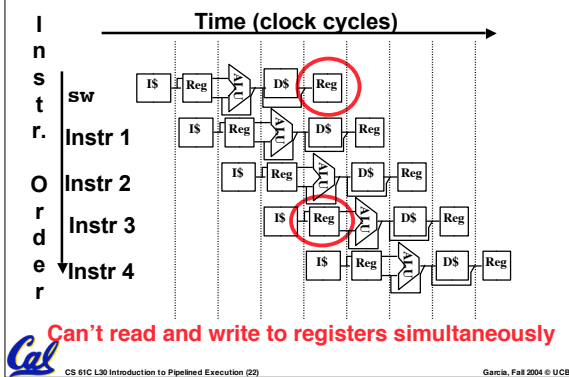


Structural Hazard #1: Single Memory (2/2)

• 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)



Structural Hazard #2: Registers (2/2)

• **Fact:** Register access is *VERY* fast: takes less than half the time of ALU stage

• Solution: introduce convention

- always Write to Registers during first half of each clock cycle
- always Read from Registers during second half of each clock cycle
- **Result:** can perform Read and Write during same clock cycle

Peer Instruction

- Thanks to pipelining, I have **reduced the time** it took me to wash my shirt.
- Longer pipelines are **always a win** (since less work per stage & a faster clock).
- We can **rely on compilers** to help us avoid data hazards by reordering instrs.

	ABC
1:	FFF
2:	FFT
3:	FTF
4:	FTT
5:	TFF
6:	TFT
7:	FTT
8:	TTT

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.