CS61C - Machine Structures

Lecture 21 - Introduction to Pipelined Execution

November 8, 2000

David Patterson

http://www-inst.eecs.berkeley.edu/~cs61c/

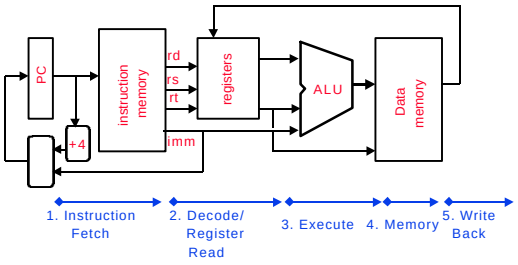
Review (1/3)

- 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 (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.
- Datapath designed in hardware.

Review Datapath

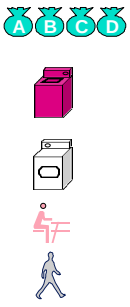


Outline

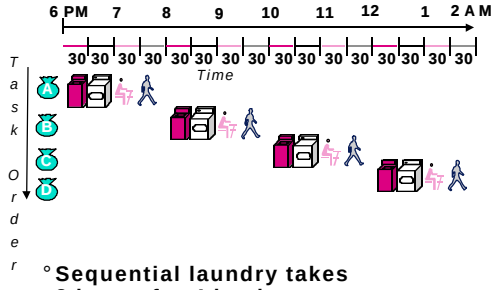
- Pipelining Analogy
- Pipelining Instruction Execution
- Hazards
- Advanced Pipelining Concepts by Analogy

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

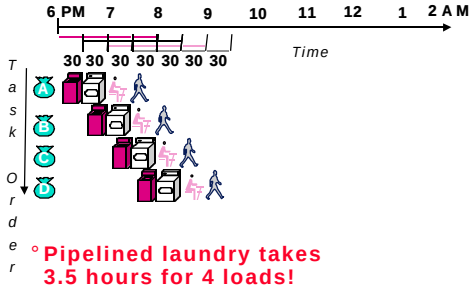


Sequential Laundry



° Sequential laundry takes 8 hours for 4 loads

Pipelined Laundry



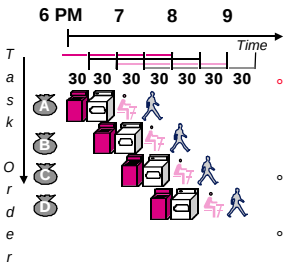
° Pipelined laundry takes 3.5 hours for 4 loads!

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)



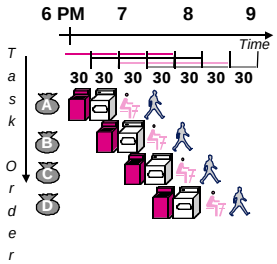
° Pipelining doesn't help latency of single task, it helps throughput of entire workload

° 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

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

Steps in Executing MIPS

1) IFetch: Fetch Instruction, Increment PC

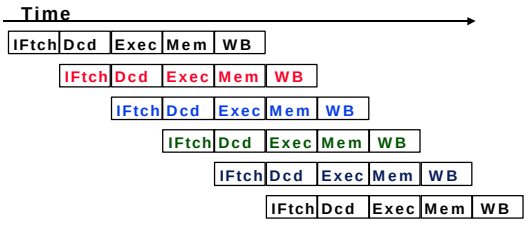
2) Decode Instruction, Read Registers

3) Execute:
   Mem-ref: Calculate Address
   Arith-log: Perform Operation

4) Memory:
   Load: Read Data from Memory
   Store: Write Data to Memory

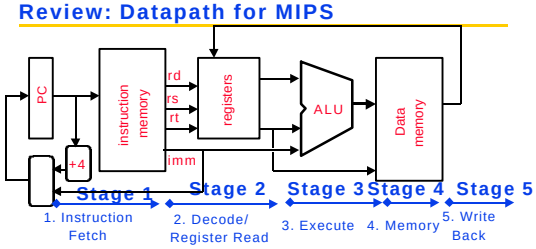
5) Write Back: Write Data to Register

Pipelined Execution Representation

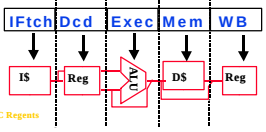


° Every instruction must take same number of steps, also called pipeline "stages", so some will go idle sometimes

Review: Datapath for MIPS

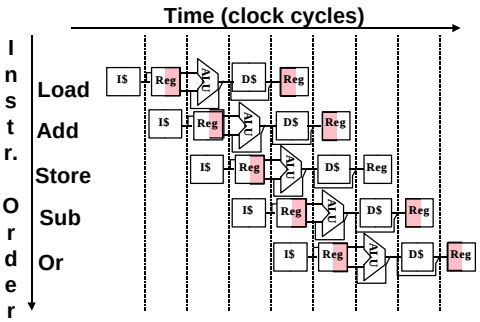


° Use datapath figure to represent pipeline



Graphical Pipeline Representation

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



Example

° Suppose 2 ns for memory access, 2 ns for ALU operation, and 1 ns for register file read or write

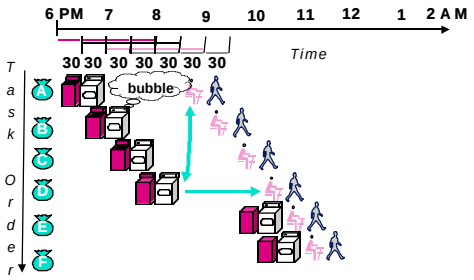
° Nonpipelined Execution:

- 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

Pipeline Hazard: Matching socks in later load



A depends on D; stall since folder tied up

Administrivia: Rest of 61C

- Rest of 61C slower pace
  - 1 project, 1 lab, no more homeworks

F 11/17 Performance; Cache Sim Project
W 11/24 X86, PC buzzwords and 61C

W 11/29 Review: Pipelines; RAID Lab
F 12/1 Review: Caches/TLB/VM; Section 7.5

M 12/4 Deadline to correct your grade record

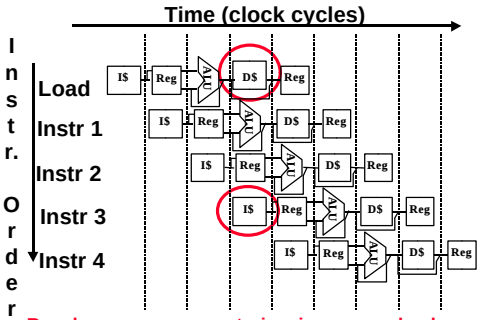
W 12/6 Review: Interrupts (A.7); Feedback lab
F 12/8 61C Summary / Your Cal heritage / HKN Course Evaluation

Sun 12/10 Final Review, 2PM (155 Dwinelle)
Tues 12/12 Final (5PM 1 Pimintel)

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)

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

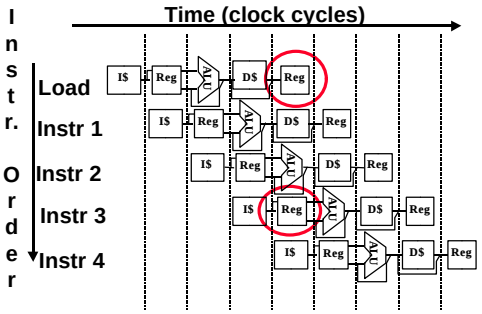


Read same memory twice in same clock cycle

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

- Solution:
  - infeasible and inefficient to create second memory
  - so simulate this by having two Level 1 Caches
  - 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't read and write to registers simultaneously

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

Control Hazard: Branching (1/6)

- Suppose we put branch decision-making hardware in ALU stage
  - then two more instructions after the branch will *always* be fetched, whether or not the branch is taken
- Desired functionality of a branch
  - if we do not take the branch, don't waste any time and continue executing normally
  - if we take the branch, don't execute any instructions after the branch, just go to the desired label

Control Hazard: Branching (2/6)

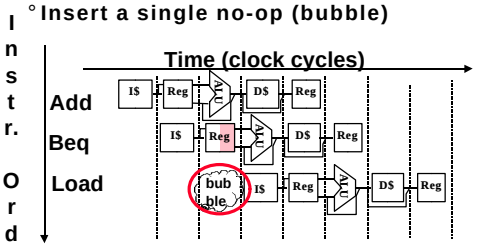
- Initial Solution: Stall until decision is made
  - insert "no-op" instructions: those that accomplish nothing, just take time
  - Drawback: branches take 3 clock cycles each (assuming comparator is put in ALU stage)

Control Hazard: Branching (3/6)

- Optimization #1:
  - move comparator up to Stage 2
  - as soon as instruction is decoded (Opcode identifies is as a branch), immediately make a decision and set the value of the PC (if necessary)
  - Benefit: since branch is complete in Stage 2, only one unnecessary instruction is fetched, so only one no-op is needed
  - Side Note: This means that branches are idle in Stages 3, 4 and 5.

Control Hazard: Branching (4/6)

- Insert a single no-op (bubble)



- Impact: 2 clock cycles per branch instruction ⇒ slow

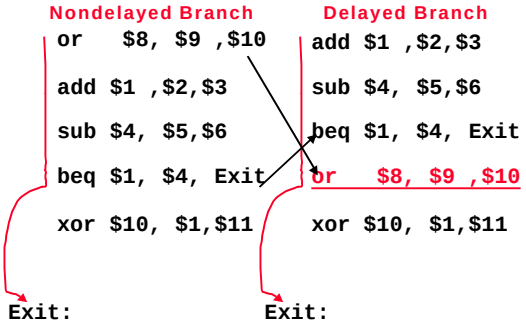
Control Hazard: Branching (5/6)

- Optimization #2: Redefine branches
  - Old definition: if we take the branch, none of the instructions after the branch get executed by accident
  - New definition: whether or not we take the branch, the single instruction immediately following the branch gets executed (called the branch-delay slot)

Control Hazard: Branching (6/6)

- Notes on Branch-Delay Slot
  - Worst-Case Scenario: can always put a no-op in the branch-delay slot
  - Better Case: can find an instruction preceding the branch which can be placed in the branch-delay slot without affecting flow of the program
    - re-ordering instructions is a common method of speeding up programs
    - compiler must be very smart in order to find instructions to do this
    - usually can find such an instruction at least 50% of the time

Example: Nondelayed vs. Delayed Branch



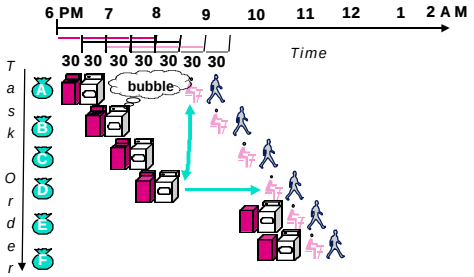
Things to Remember (1/2)

- 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.

Advanced Pipelining Concepts (if time)

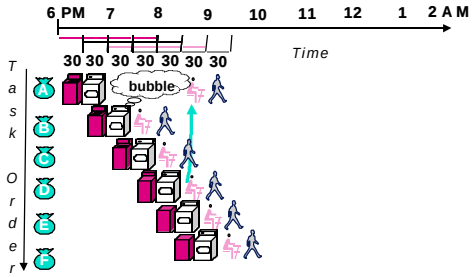
- "Out-of-order" Execution
- "Superscalar" execution
- State-of-the-Art Microprocessor

Review Pipeline Hazard: Stall is dependency



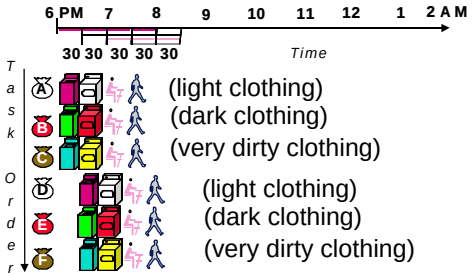
A depends on D; stall since folder tied up

Out-of-Order Laundry: Don't Wait



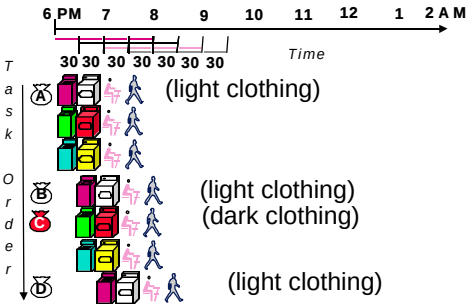
A depends on D; rest continue; need more resources to allow out-of-order

Superscalar Laundry: Parallel per stage



More resources, HW to match mix of parallel tasks?

Superscalar Laundry: Mismatch Mix



Task mix underutilizes extra resources

State of the Art: Compaq Alpha 21264

- Very similar instruction set to MIPS
- 1 64KB Instruction cache, 1 64 KB Data cache on chip; 16MB L2 cache off chip
- Clock cycle = 1.5 nanoseconds, or 667 MHz clock rate
- Superscalar: fetch up to 6 instructions /clock cycle, retires up to 4 instruction/clock cycle
- Execution out-of-order
- 15 million transistors, 90 watts!

Things to Remember (1/2)

- 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.

Things to Remember (2/2)

- Pipelining a Big Idea: widely used concept
- What makes it less than perfect?
  - Structural hazards: suppose we had only one cache?
    ⇒ Need more HW resources
  - Control hazards: need to worry about branch instructions?
    ⇒ Delayed branch
  - Data hazards: an instruction depends on a previous instruction?