Umass Lowell 16.671
Advanced Computer Architecture

Lecture 1
Fundamentals of Computer Design

Instructor: Prof. Yan Luo

Instructor Information

Prof. Yan Luo
Office: Ball Bldg Room 413
E-mail: yan_luo@uml.edu
Tel: (978) 934-2592
Office Hours: Wed, Fri 9:30-11am

Course Syllabus

- **Pipeline and Hazards - Appendix A**
- **Instruction level parallelism, Dynamic scheduling, Branch Prediction and Speculation – Ch 3 Text**
- **ILP with Software Approaches – Ch 4**
- **Memory Hierarchy – Ch 5**
- **VLIW, Multithreading, CMP and Network processor architectures – From papers**

Text: Hennessy and Patterson, Computer Architecture: A Quantitative Approach, Morgan Kaufman Publisher, 3rd Ed. ISBN 1558605967

Prerequisite: 16.561 Computer Architecture

Course Details

Grading

Quiz 1: 15%

Quiz 2: 15%

Class Participation: 20%

Project: 50%

What is *Computer Architecture*

Computer Architecture =

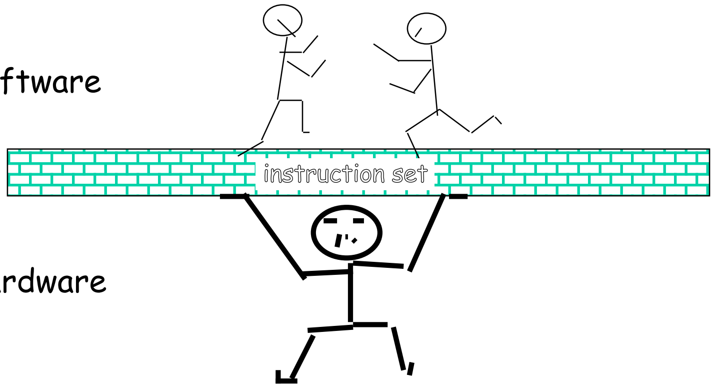
Instruction Set Architecture +

Organization +

Hardware + …

The Instruction Set: a Critical Interface

The actual programmer visible instruction set



software

hardware

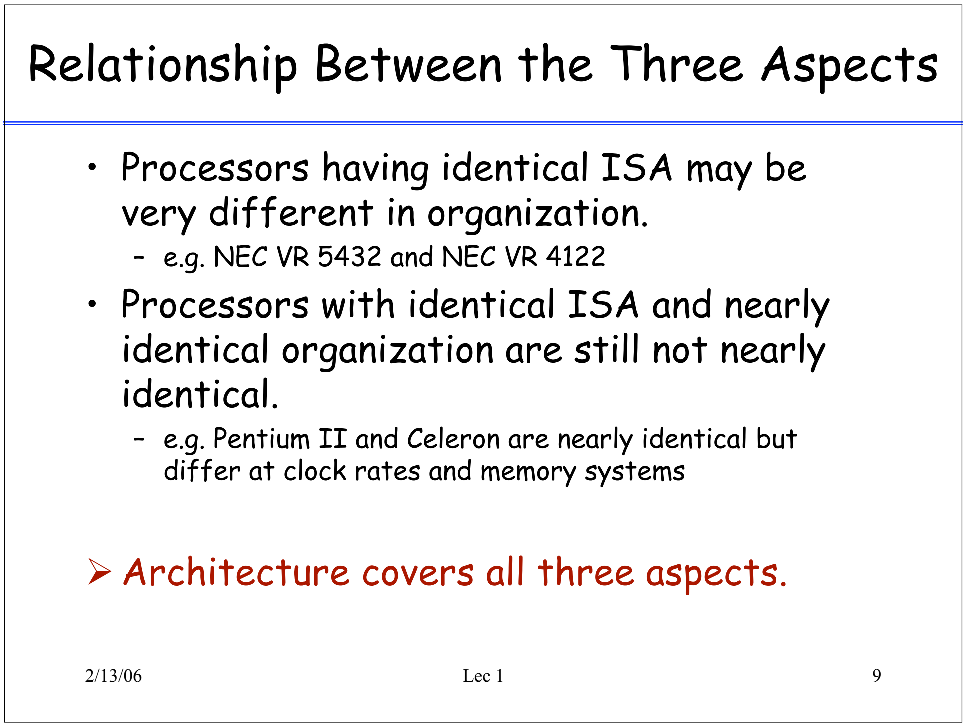
Instruction-Set Processor Design

- Architecture (ISA) programmer/compiler view
  - "functional appearance to its immediate user/system programmer"
  - **Opcodes, addressing modes, architected registers, IEEE floating point**
- Implementation (µarchitecture) processor designer/view
  - "logical structure or organization that performs the architecture"
  - **Pipelining, functional units, caches, physical registers**
- Realization (chip) chip/system designer view
  - "physical structure that embodies the implementation"
  - **Gates, cells, transistors, wires**

Hardware

- Machine specifics:
  - Feature size (10 microns in 1971 to 0.18 microns in 2001, 90nm in 2003)
    - Minimum size of a transistor or a wire in either the x or y dimension
  - Logic designs
  - Packaging technology
  - Clock rate
  - Supply voltage

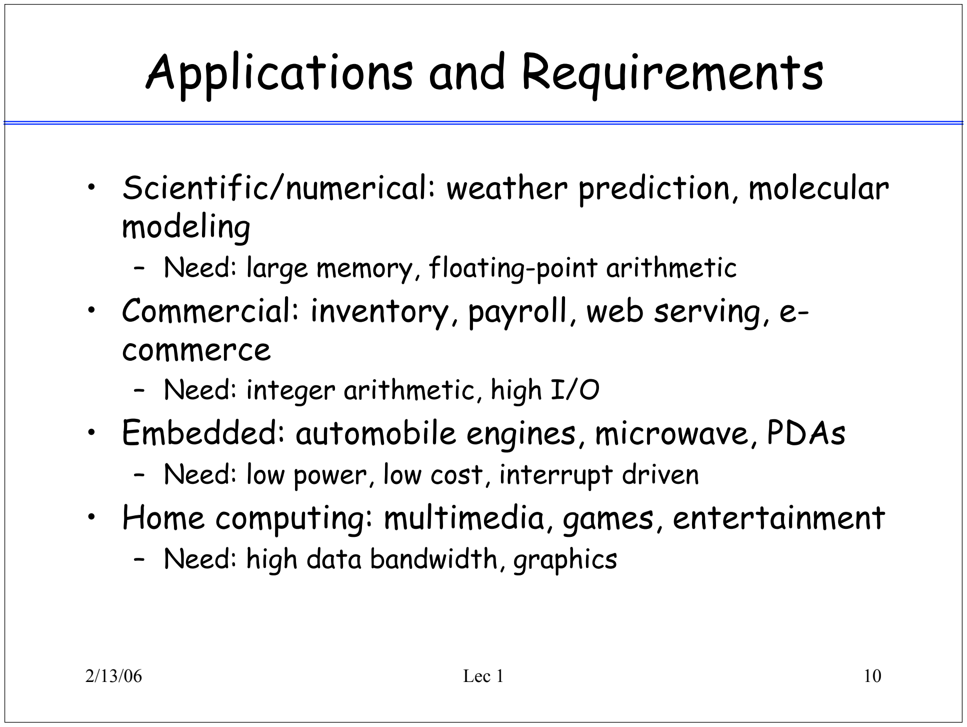
  …



Relationship Between the Three Aspects

- Processors having identical ISA may be very different in organization.
  - e.g. NEC VR 5432 and NEC VR 4122
- Processors with identical ISA and nearly identical organization are still not nearly identical.
  - e.g. Pentium II and Celeron are nearly identical but differ at clock rates and memory systems

➢ Architecture covers all three aspects.



Applications and Requirements

- Scientific/numerical: weather prediction, molecular modeling
  - Need: large memory, floating-point arithmetic
- Commercial: inventory, payroll, web serving, e-commerce
  - Need: integer arithmetic, high I/O
- Embedded: automobile engines, microwave, PDAs
  - Need: low power, low cost, interrupt driven
- Home computing: multimedia, games, entertainment
  - Need: high data bandwidth, graphics

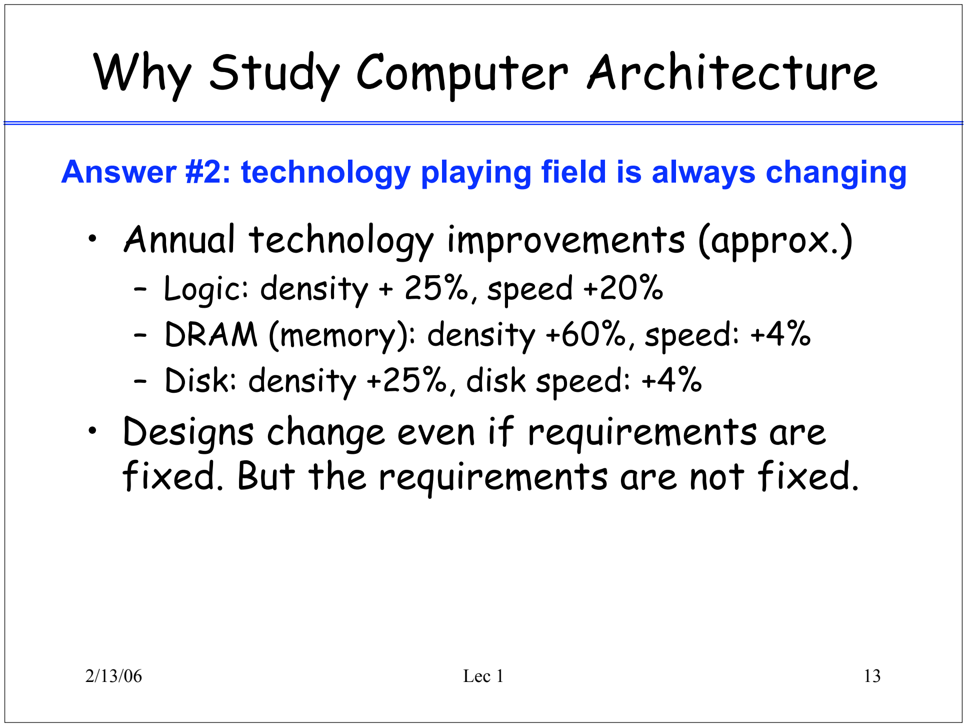
Classes of Computers

- High performance (supercomputers)
  - Supercomputers – Cray T-90
  - Massively parallel computers – Cray T3E
- Balanced cost/performance
  - Workstations – SPARCstations
  - Servers – SGI Origin, UltraSPARC
  - High-end PCs – Pentium quads
- Low cost/power
  - Low-end PCs, laptops, PDAs – mobile Pentiums

Why Study Computer Architecture

- Aren't they fast enough already?
  - Are they?
  - Fast enough to do everything we will EVER want?
    - AI, protein sequencing, graphics
  - Is speed the only goal?
    - Power: heat dissipation + battery life
    - Cost
    - Reliability
    - Etc.

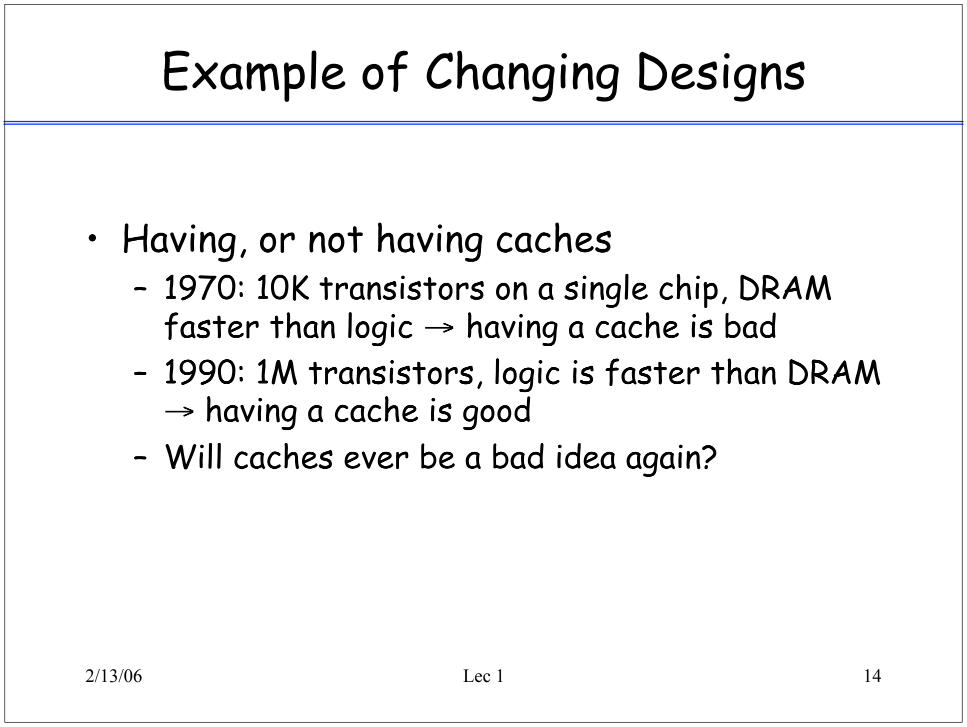
Answer #1: requirements are always changing



Why Study Computer Architecture

Answer #2: technology playing field is always changing

- Annual technology improvements (approx.)
  - Logic: density + 25%, speed +20%
  - DRAM (memory): density +60%, speed: +4%
  - Disk: density +25%, disk speed: +4%
- Designs change even if requirements are fixed. But the requirements are not fixed.



Example of Changing Designs

- Having, or not having caches
  - 1970: 10K transistors on a single chip, DRAM faster than logic → having a cache is bad
  - 1990: 1M transistors, logic is faster than DRAM → having a cache is good
  - Will caches ever be a bad idea again?

Performance Growth in Perspective

- Same absolute increase in computing power
  - Big Bang – 2001
  - 2001 – 2003
- 1971 – 2001: performance improved 35,000X!!!
  - What if cars improved at this rate?

Measuring Performance

- Latency (response time, execution time)
  - Minimize time to wait for a computation
- Throughput (tasks completed per unit time, bandwidth)
  - Maximize work done in a given interval
  - = 1/latency when there is no overlap among tasks
  - > 1/latency when there is
    - In real processors there is always overlap (pipelining)
- Both are important

Performance Terminology

"X is *n* times faster than Y" means:

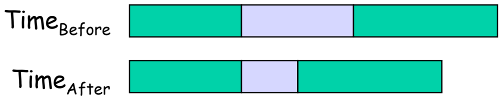
$$\frac{\text{Execution time}_Y}{\text{Execution time}_X} = n$$

"X is *m*% faster than Y" means:

$$\frac{\text{Execution time}_Y - \text{Execution time}_X}{\text{Execution time}_X} \text{ X } 100\% = m$$

Compute Speedup – Amdahl's Law

Speedup is due to enhancement(E):



$$\text{Speedup (E)} = \frac{\text{Execution time w/o E (Before)}}{\text{Execution time w E (After)}}$$

Suppose that enhancement E accelerates a fraction F of the task by a factor S, and the remainder of the task is unaffected, what is the *Execution time$_{after}$* and *Speedup(E)*?

Amdahl's Law

$$\textit{Execution time}_{after} = \text{ExTime}_{before} \times \left[(1-F) + \frac{F}{S}\right]$$

$$\textit{Speedup(E)} = \frac{\text{ExTime}_{before}}{\text{ExTime}_{after}} = \frac{1}{\left[(1-F) + \frac{F}{S}\right]}$$

Amdahl's Law – An Example

Q: Floating point instructions improved to run 2X; but only 10% of execution time are FP ops. What is the execution time and speedup after improvement?

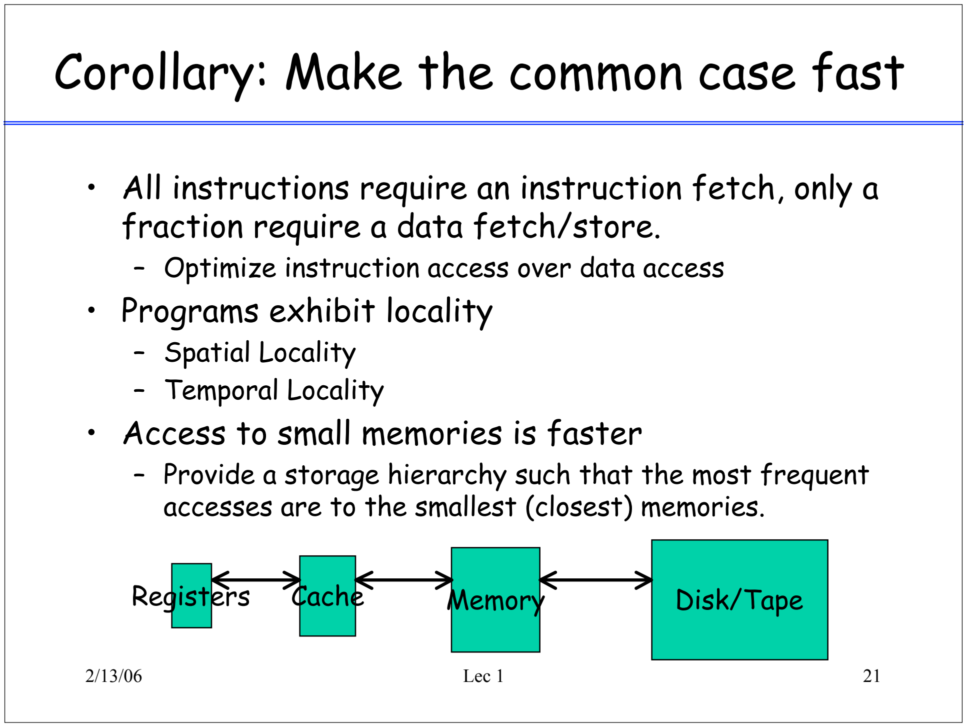
Ans:

$F = 0.1$, $S = 2$

$\text{ExTime}_{after} = \text{ExTime}_{before} \times [\ (1-0.1) + 0.1/2\ ] = 0.95\ \text{ExTime}_{before}$

$$\text{Speedup} = \frac{\text{ExTime}_{before}}{\text{ExTime}_{after}} = \frac{1}{0.95} = 1.053$$

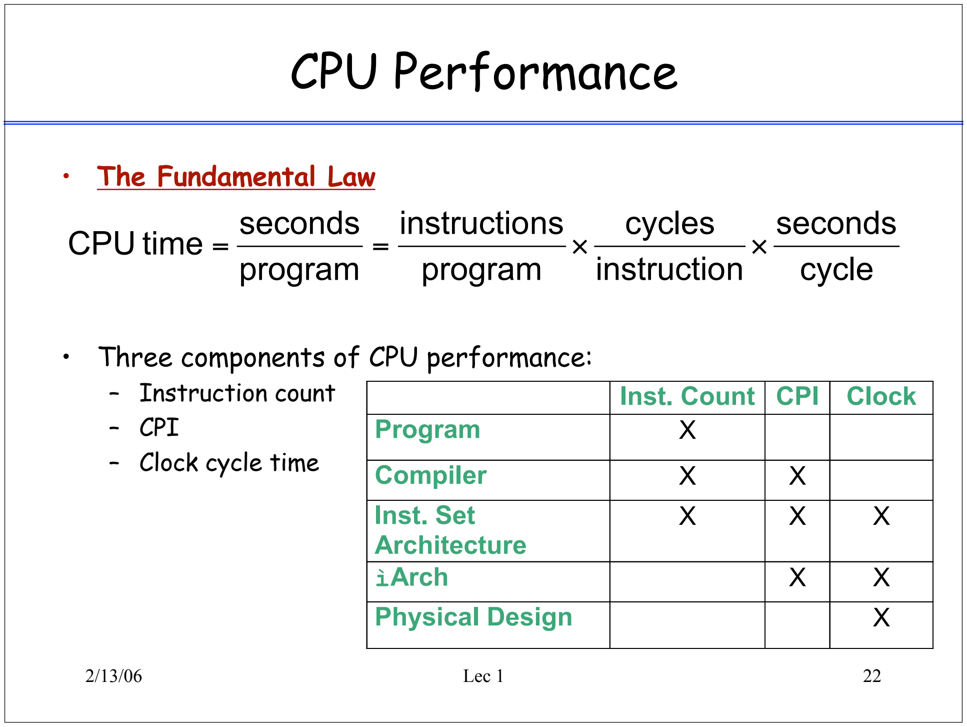
Read examples in the book!



Corollary: Make the common case fast

- All instructions require an instruction fetch, only a fraction require a data fetch/store.
  - Optimize instruction access over data access
- Programs exhibit locality
  - Spatial Locality
  - Temporal Locality
- Access to small memories is faster
  - Provide a storage hierarchy such that the most frequent accesses are to the smallest (closest) memories.

Registers ↔ Cache ↔ Memory ↔ Disk/Tape



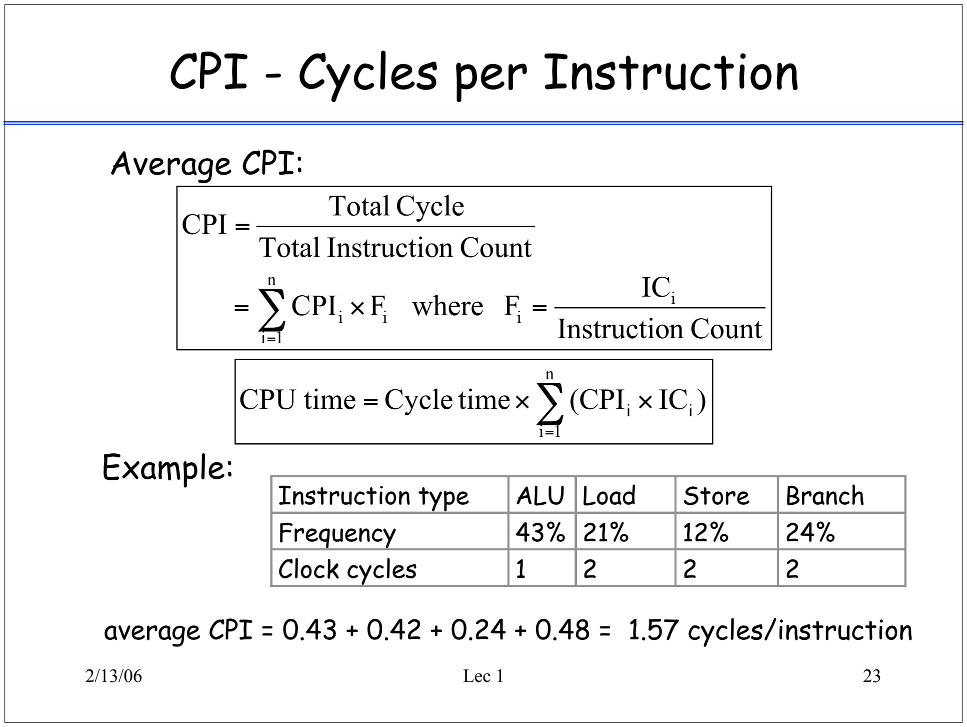
CPU Performance

- **The Fundamental Law**

$$\text{CPU time} = \frac{\text{seconds}}{\text{program}} = \frac{\text{instructions}}{\text{program}} \times \frac{\text{cycles}}{\text{instruction}} \times \frac{\text{seconds}}{\text{cycle}}$$

- Three components of CPU performance:
  - Instruction count
  - CPI
  - Clock cycle time

	Inst. Count	CPI	Clock
Program	X		
Compiler	X	X	
Inst. Set Architecture	X	X	X
ìArch		X	X
Physical Design			X



CPI - Cycles per Instruction

Average CPI:

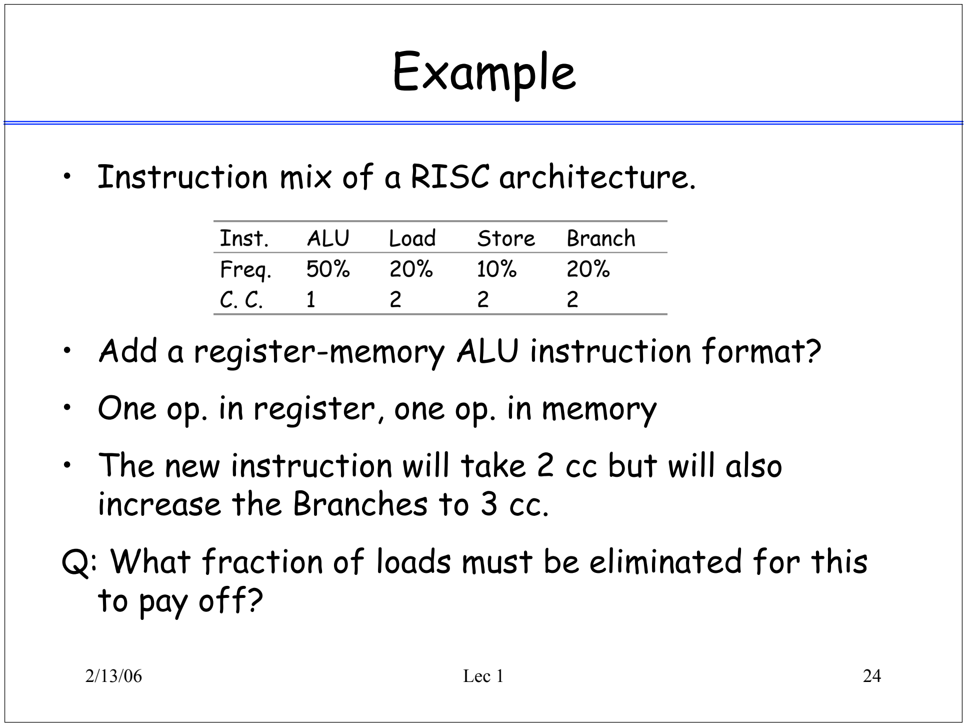
$$\mathrm{CPI} = \frac{\text{Total Cycle}}{\text{Total Instruction Count}} = \sum_{i=1}^{n} \mathrm{CPI}_i \times F_i \quad \text{where} \quad F_i = \frac{\mathrm{IC}_i}{\text{Instruction Count}}$$

$$\text{CPU time} = \text{Cycle time} \times \sum_{i=1}^{n} (\mathrm{CPI}_i \times \mathrm{IC}_i)$$

Example:

Instruction type	ALU	Load	Store	Branch
Frequency	43%	21%	12%	24%
Clock cycles	1	2	2	2

average CPI = 0.43 + 0.42 + 0.24 + 0.48 = 1.57 cycles/instruction



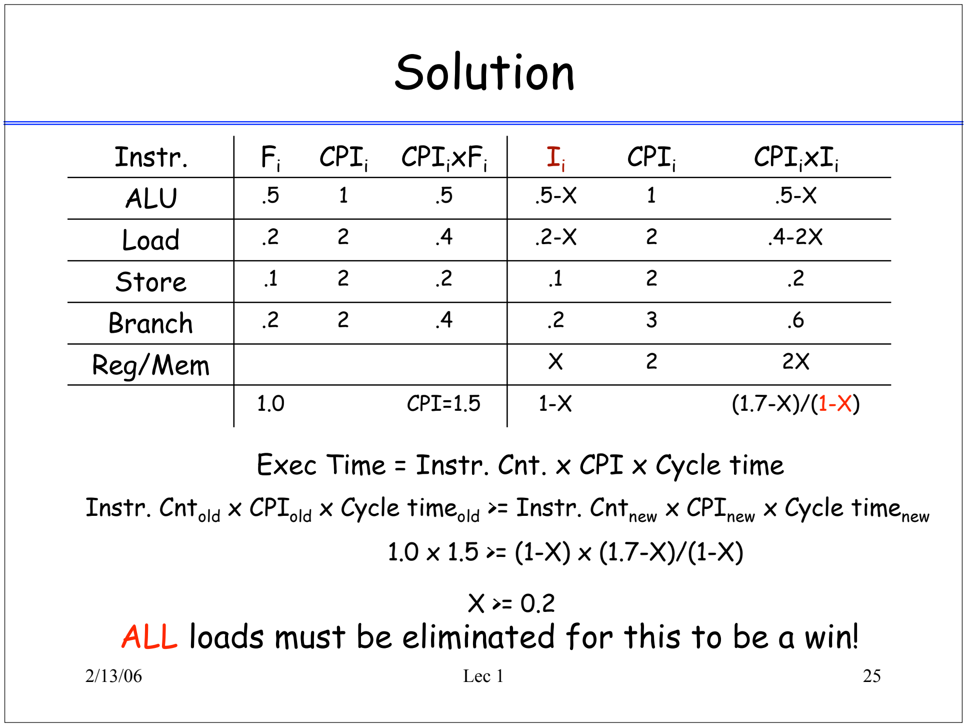
Example

- Instruction mix of a RISC architecture.

Inst.	ALU	Load	Store	Branch
Freq.	50%	20%	10%	20%
C. C.	1	2	2	2

- Add a register-memory ALU instruction format?
- One op. in register, one op. in memory
- The new instruction will take 2 cc but will also increase the Branches to 3 cc.

Q: What fraction of loads must be eliminated for this to pay off?



Solution

Instr.	F_i	CPI_i	$CPI_i x F_i$	I_i	CPI_i	$CPI_i x I_i$
ALU	.5	1	.5	.5-X	1	.5-X
Load	.2	2	.4	.2-X	2	.4-2X
Store	.1	2	.2	.1	2	.2
Branch	.2	2	.4	.2	3	.6
Reg/Mem				X	2	2X
	1.0		CPI=1.5	1-X		(1.7-X)/(1-X)

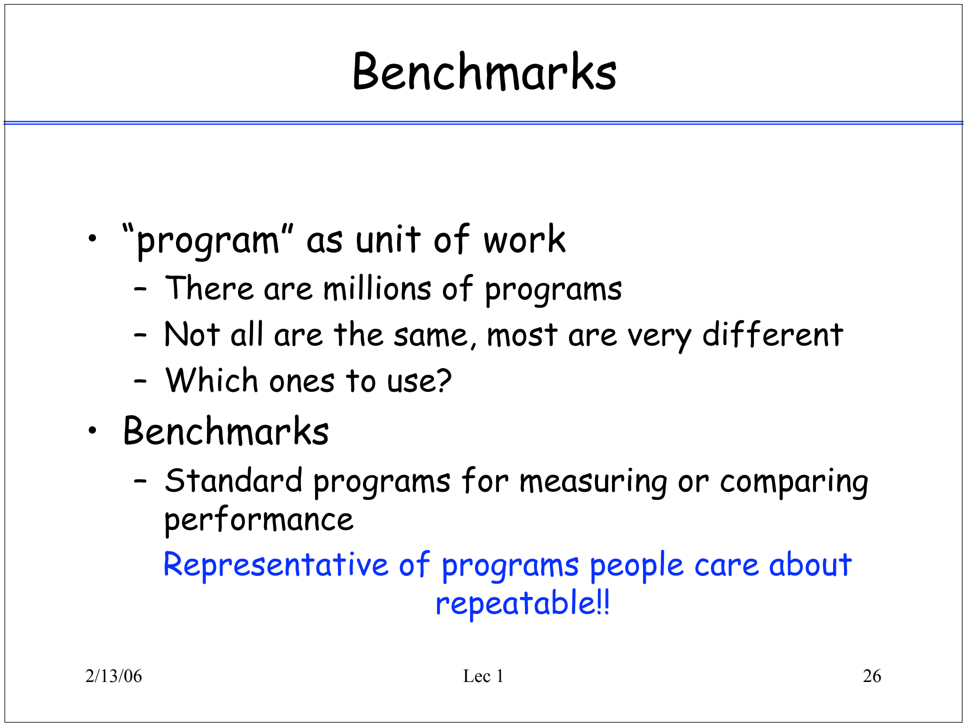
Exec Time = Instr. Cnt. x CPI x Cycle time

$\text{Instr. Cnt}_{old} \times \text{CPI}_{old} \times \text{Cycle time}_{old} >= \text{Instr. Cnt}_{new} \times \text{CPI}_{new} \times \text{Cycle time}_{new}$

$1.0 \times 1.5 >= (1-X) \times (1.7-X)/(1-X)$

$X >= 0.2$

ALL loads must be eliminated for this to be a win!



Benchmarks

- “program” as unit of work
  - There are millions of programs
  - Not all are the same, most are very different
  - Which ones to use?
- Benchmarks
  - Standard programs for measuring or comparing performance

Representative of programs people care about repeatable!!

Choosing Programs to Evaluate Perf.

- Toy benchmarks
  - e.g., quicksort, puzzle
  - No one really runs. Scary fact: used to prove the value of RISC in early 80's
- Synthetic benchmarks
  - Attempt to match average frequencies of operations and operands in real workloads.
  - e.g., Whetstone, Dhrystone
  - Often slightly more complex than kernels; But do not represent real programs
- Kernels
  - Most frequently executed pieces of real programs
  - e.g., livermore loops
  - Good for focusing on individual features not big picture
  - Tend to over-emphasize target feature
- Real programs
  - e.g., gcc, spice, SPEC89, 92, 95, SPEC2000 (standard performance evaluation corporation)

MIPS and MFLOPS

- **MIPS:** millions of instructions per second:
  - MIPS = Inst. count/ (CPU time * 10**6) = Clock rate/(CPI*10^6)
  - easy to understand and to market
  - inst. set dependent, cannot be used across machines.
  - program dependent
  - can vary inversely to performance! (why? read the book)
- **MFLOPS:** million of FP ops per second.
  - less compiler dependent than MIPS.
  - not all FP ops are implemented in h/w on all machines.
  - not all FP ops have same latencies.
  - normalized MFLOPS: uses an equivalence table to even out the various latencies of FP ops.