Computer Performance

Performance

- Purchasing perspective
  - given a collection of machines, which has the
    - best performance ?
    - least cost ?
    - best performance / cost ?
- Design perspective
  - faced with design options, which has the
    - best performance improvement ?
    - least cost ?
    - best performance / cost ?
- Both require
  - basis for comparison
  - metric for evaluation
- Our goal is to understand cost & performance implications of architectural choices

Two notions of “performance”

Plane	DC to Paris	Speed	Passengers	Throughput (pmph)
Boeing 747	6.5 hours	610 mph	470	286,700
BAD/Sud Concodre	3 hours	1350 mph	132	178,200

Which has higher performance?

- **Time to do the task (Execution Time)**
  - execution time, response time, latency
- **Tasks per day, hour, week, sec, ns. .. (Performance)**
  - throughput, bandwidth

Response time and throughput often are in opposition

Definitions

- Performance is in units of things-per-second
  - bigger is better
- If we are primarily concerned with response time
  - performance(x) = $\frac{1}{\text{execution\_time}(x)}$

" X is n times faster than Y" means

$$n = \frac{\text{Performance}(X)}{\text{Performance}(Y)}$$

1 Hz = 1 cycle/sec 1 KHz = 103 cycles/sec
1 MHz = 106 cycles/sec 1 GHz = 109 cycles/sec
2 GHz clock has a cycle time = 1/(2×109) = 0.5 nanosecond (ns)

Execution.time x Clock.Rate = Instruction.time x CPI
MIPS x CPI = Clock.Rate

Example

A program runs in 10 seconds on computer *X* with 2 GHz clock
What is the number of CPU cycles on computer *X* ?
We want to design computer *Y* to run same program in 6 seconds
But computer *Y* requires 10% more cycles to execute program
What is the clock rate for computer *Y* ?

Solution:
CPU cycles on computer *X* = 10 sec × 2 × 10^9 cycles/s = 20 × 109
CPU cycles on computer *Y* = 1.1 × 20 × 10^9 = 22 × 10^9 cycles
Clock rate for computer *Y* = 22 × 10^9 cycles / 6 sec = 3.67 GHz

Aspects of CPU Performance

$$\text{CPU time} = \frac{\text{Seconds}}{\text{Program}} = \frac{\text{Instructions}}{\text{Program}} \times \frac{\text{Cycles}}{\text{Instruction}} \times \frac{\text{Seconds}}{\text{Cycle}}$$

	instr. count	CPI	clock rate
Program			
Compiler			
Instr. Set Arch.			
Organization			
Technology			

Aspects of CPU Performance

$$\text{CPU time} = \frac{\text{Seconds}}{\text{Program}} = \frac{\text{Instructions}}{\text{Program}} \times \frac{\text{Cycles}}{\text{Instruction}} \times \frac{\text{Seconds}}{\text{Cycle}}$$

	instr count	CPI	clock rate
Program	X		
Compiler	X	X	
Instr. Set	X	X	
Organization		X	X
Technology			X

CPI

“Average cycles per instruction”

CPI = (CPU Time * Clock Rate) / Instruction Count
= Clock Cycles / Instruction Count

$$\text{CPU time} = \text{ClockCycleTime} * \sum_{i=1}^{n} \text{CPI}_i * I_i$$

$$\text{CPI} = \sum_{i=1}^{n} \text{CPI}_i * F_i \quad \text{where } F_i = \frac{I_i}{\text{Instruction Count}}$$

"instruction frequency"

Invest Resources where time is Spent!

Problem 1
Machine A has a clock cycle time of 250 ps and a CPI of 2.0
Machine B has a clock cycle time of 500 ps and a CPI of 1.2
Which machine is faster for this program, and by how much?

Solution:
Both computer execute same count of instructions = I
CPU execution time (A) = I × 2.0 × 250 ps = 500 × I ps
CPU execution time (B) = I × 1.2 × 500 ps = 600 × I ps
Computer A is faster than B by a factor = 1.2

Problem 2
A compiler designer is trying to decide between two code sequences for a particular machine. Based on the hardware implementation, there are three different classes of instructions: class A, class B, and class C, and they require one, two, and three cycles per instruction, respectively.
The first code sequence has 5 instructions: 2 of A, 1 of B, and 2 of C
The second sequence has 6 instructions: 4 of A, 1 of B, and 1 of C
Compute the CPU cycles for each sequence. Which sequence is faster?
What is the CPI for each sequence?

Solution
CPU cycles (1st sequence) = (2×1) + (1×2) + (2×3) = 2+2+6 = 10 cycles
CPU cycles (2nd sequence) = (4×1) + (1×2) + (1×3) = 4+2+3 = 9 cycles
Second sequence is faster, even though it executes one extra instruction
CPI (1st sequence) = 10/5 = 2 CPI (2nd sequence) = 9/6 = 1.5

Example (RISC processor)

Base Machine (Reg / Reg)

Op	Freq	Cycles	CPI(i)	% Time
ALU	50%	1	.5	23%
Load	20%	5	1.0	45%
Store	10%	3	.3	14%
Branch	20%	2	.4	18%
			2.2	

Typical Mix

How much faster would the machine be is a better data cache reduced the average load time to 2 cycles?

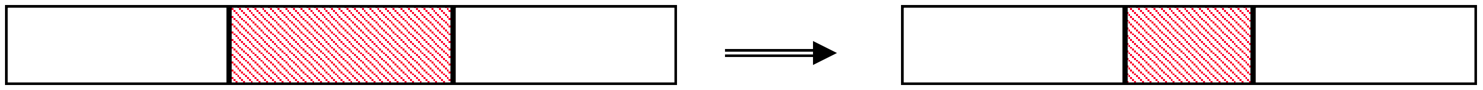
How does this compare with using branch prediction to shave a cycle off the branch time?

What if two ALU instructions could be executed at once?

Amdahl's Law

Speedup due to enhancement E:

$$\text{Speedup(E)} = \frac{\text{ExTime w/o E}}{\text{ExTime w/ E}} = \frac{\text{Performance w/ E}}{\text{Performance w/o E}}$$



Suppose that enhancement E accelerates a fraction F of the task

by a factor S and the remainder of the task is unaffected then,

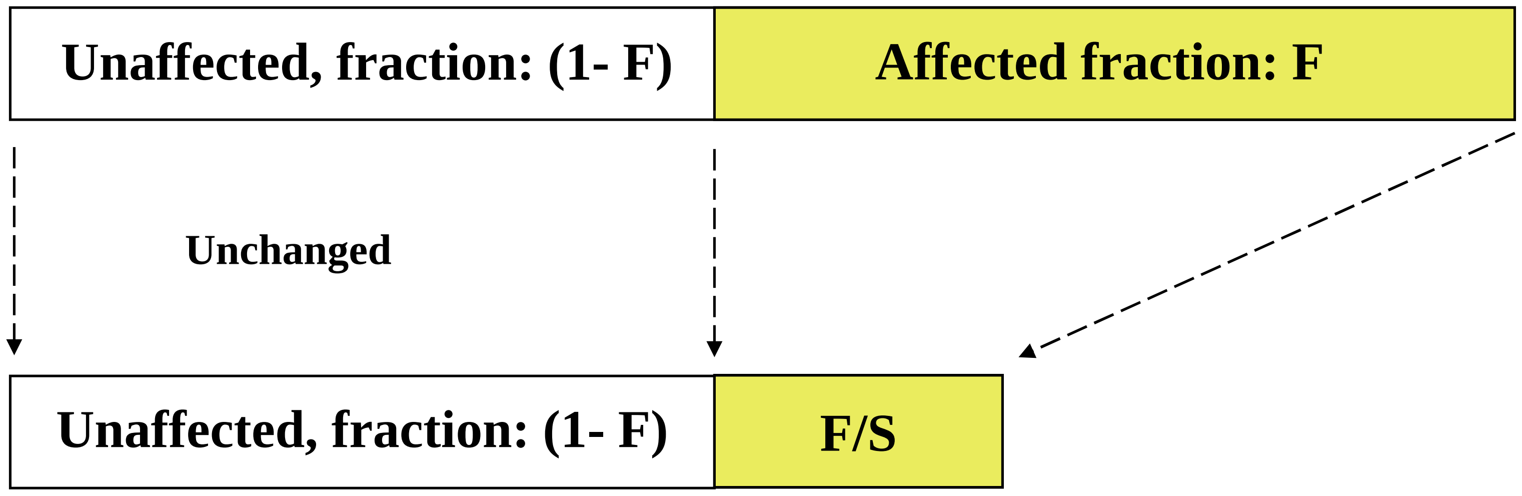
ExTime(with E) = ((1-F) + F/S) X ExTime(without E)

$$\text{Speedup(with E)} = \frac{1}{(1\text{-}F) + F/S}$$

Pictorial Depiction of Amdahl's Law

Enhancement E accelerates fraction F of execution time by a factor of S

Before:
Execution Time without enhancement E:



After:
Execution Time with enhancement E:

$$\text{Speedup}(E) = \frac{\text{Execution Time without enhancement E}}{\text{Execution Time with enhancement E}} = \frac{1}{(1 - F) + F/S}$$

(From 550)

Performance Enhancement Example

° **For the RISC machine with the following instruction mix given earlier:**

Op	Freq	Cycles	CPI(i)	% Time
ALU	50%	1	.5	23%
Load	20%	5	1.0	45%
Store	10%	3	.3	14%
Branch	**20%**	**2**	**.4**	**18%**

CPI = 2.2

° **If a CPU design enhancement improves the CPI of load instructions from 5 to 2, what is the resulting performance improvement from this enhancement:**

Fraction enhanced = F = 45% or .45

Unaffected fraction = 100% - 45% = 55% or .55

Factor of enhancement = 5/2 = 2.5

Average CPI = 0.5+1.0+0.3+0.4 = 2.2 clocks/instructions

Using Amdahl's Law:

$$\text{Speedup(E)} = \frac{1}{(1 - F) + F/S} = \frac{1}{.55 + .45/2.5} = 1.37$$

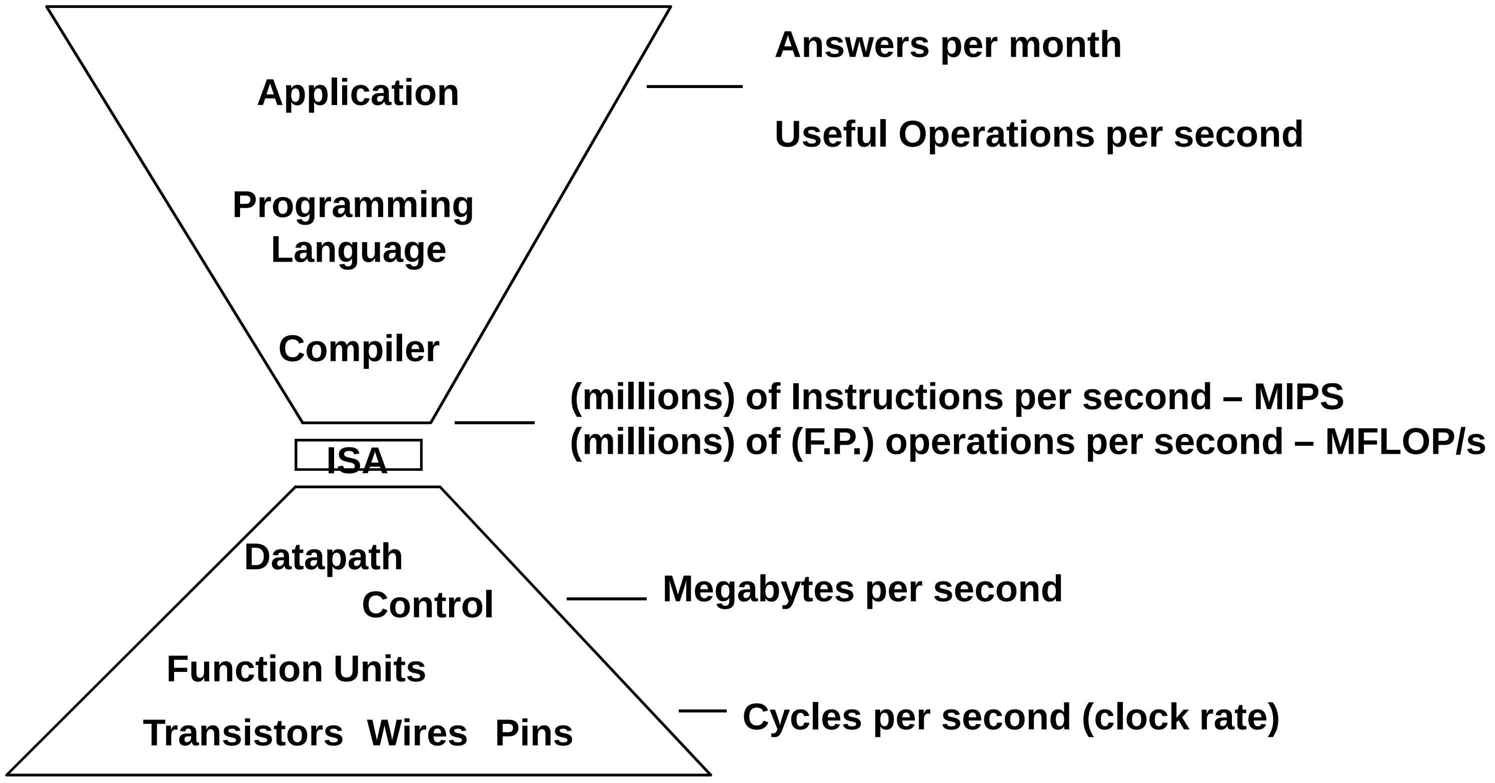
Basis of Evaluation

Pros		Cons
• representative	Actual Target Workload	• very specific • non-portable • difficult to run, or measure • hard to identify cause
• portable • widely used • improvements useful in reality	Full Application Benchmarks	•less representative
• easy to run, early in design cycle	Small “Kernel” Benchmarks	• easy to “fool”
• identify peak capability and potential bottlenecks	Microbenchmarks	• “peak” may be a long way from application performance

SPEC95

- Eighteen application benchmarks (with inputs) reflecting a technical computing workload
- Eight integer
  - go, m88ksim, gcc, compress, li, ijpeg, perl, vortex
- Ten floating-point intensive
  - tomcatv, swim, su2cor, hydro2d, mgrid, applu, turb3d, apsi, fppp, wave5
- Must run with standard compiler flags
  - eliminate special undocumented incantations that may not even generate working code for real programs

Metrics of performance


Application
Programming Language
Compiler
ISA
Datapath
Control
Function Units
Transistors Wires Pins
Answers per month
Useful Operations per second
(millions) of Instructions per second – MIPS
(millions) of (F.P.) operations per second – MFLOP/s
Megabytes per second
Cycles per second (clock rate)

Each metric has a place and a purpose, and each can be misused

Summary: Salient features of MIPS I

- **32-bit fixed format inst** (3 formats)
- **32 32-bit GPR** (R0 contains zero) and 32 FP registers (and HI LO)
  - partitioned by software convention
- **3-address, reg-reg arithmetic instr.**
- **Single address mode for load/store:** base+displacement
  - –no indirection, scaled

–**16-bit immediate plus LUI**

- **Simple branch conditions**
  - compare against zero or two registers for =,°
  - no integer condition codes
- **Delayed branch**
  - execute instruction after the branch (or jump) even if
    the branch is taken (Compiler can fill a delayed branch with
    useful work about 50% of the time)

Summary: Instruction set design (MIPS)

- Use general purpose registers with a load-store architecture: YES
- Provide at least 16 general purpose registers plus separate floating-point registers: 31 GPR & 32 FPR
- Support basic addressing modes: displacement (with an address offset size of 12 to 16 bits), immediate (size 8 to 16 bits), and register deferred; : YES: 16 bits for immediate, displacement (disp=0 => register deferred)
- All addressing modes apply to all data transfer instructions : YES
- Use fixed instruction encoding if interested in performance and use variable instruction encoding if interested in code size : Fixed
- Support these data sizes and types: 8-bit, 16-bit, 32-bit integers and 32-bit and 64-bit IEEE 754 floating point numbers: YES
- Support these simple instructions, since they will dominate the number of instructions executed: load, store, add, subtract, move register-register, and, shift, compare equal, compare not equal, branch (with a PC-relative address at least 8-bits long), jump, call, and return: YES, 16b
- Aim for a minimalist instruction set: YES

Summary: Evaluating Instruction Sets?

Design-time metrics:

° Can it be implemented, in how long, at what cost?

° Can it be programmed? Ease of compilation?

Static Metrics:

° How many bytes does the program occupy in memory?

Dynamic Metrics:

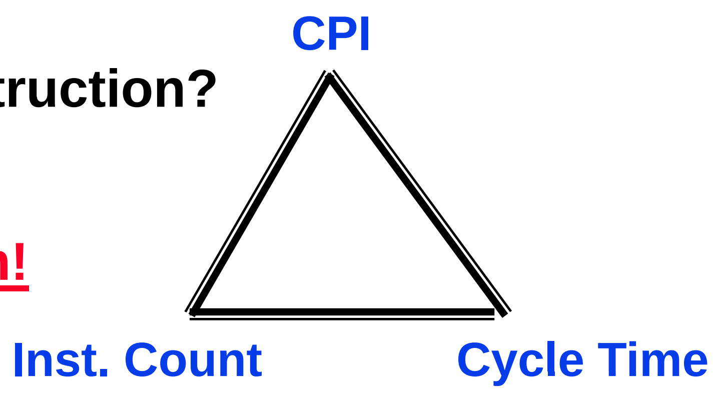
° How many instructions are executed?

° How many bytes does the processor fetch to execute the program?

° How many clocks are required per instruction?

° How "lean" a clock is practical?

Best Metric: Time to execute the program!



NOTE: this depends on instructions set, processor organization, and compilation techniques.