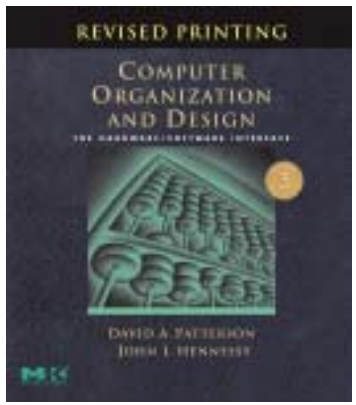
ECE369: Fundamentals of Computer Architecture

ECE 369 MWF **10:00 AM - 10:50 AM** in HARV-302		
	Instructor	**Teaching Assistant**
Name:	Ali Akoglu	Chad Rossmeisl
Office:	ECE 356-B	
Phone:	(520) 626-5149	TBA
Email:	akoglu@ece.arizona.edu	rossmeis@email.arizona.edu
Office Hours:	Mondays 11:00 AM – 12:00 Wednesdays 3:30- 4:30 or by appointment	Tuesdays 9:00 AM - 11:00 AM, Thursdays 2:00PM- 3:30 PM Fridays 10:00 AM-11:30AM, or by appointment

General Policies, Tips

- **Computer Organization and Design:**
  The Hardware/Software Interface, Third Edition, Revised Printing
- **Prerequisite:** ECE274 & Programming in C
- 8 to 12 assignments, 2 mid-terms, final project, final comprehensive exam
- NO LATE ASSIGNMENTS
- Make-ups *may* be arranged prior to the scheduled activity.
- Inquiries about graded material => within 3 days of receiving a grade.
- You are encouraged to discuss the assignment specifications with your instructor, your teaching assistant, and your fellow students. However, anything you submit for grading must be unique and should NOT be a duplicate of another source.
- Read before the class
- Participate and ask questions
- Manage your time
- Start working on assignments early



Fun Stuff…

Distribution of Components		Grades Scale	
Component	**Percentage**	**Percentage**	**Grade**
Assignments	15	90-100%	A
Midterm-I	20	80-89%	B
Midterm-II	20	70-79%	C
Project	15	60-69%	D
Final exam	30	Below 60%	E

Announcements

- **ECE250 LAB**
  - **TA lab hours, office hours, etc.**
- **Assignments & Project**
  - **Pairs only !!!**
  - **Who is my partner?**
- **Assignment-0 Due August 29th**
- **Lecture notes on the web**
  - **http://ece.arizona.edu/~ece369/**

How to survive in ece369 ?

- **Readings**
  - **General principles and historical perspectives**
- **Assignments**
  - **Apply the knowledge**
- **Project**
  - **Hands on practice**
  - **Xilinx ISE Design Tool**
  - **Verilog**

Objective

- **Understand “how computer works”**
- **Appreciate the design tradeoffs**
- **Talk/Discuss/Express/Explain/Articulate/….. at ALL LEVELS**

Chapter 1

Introduction

- This course is all about how computers work
- But what do we mean by a computer?
  - Different types: desktop, servers, embedded devices
  - Different uses: automobiles, graphics, finance, genomics…
  - Different manufacturers: Intel, Apple, IBM, Microsoft, Sun…
  - Different underlying technologies and different costs!
- Analogy: Consider a course on "automotive vehicles"
  - Many similarities from vehicle to vehicle (e.g., wheels)
  - Huge differences from vehicle to vehicle (e.g., gas vs. electric)

Why learn this stuff?

- You want to call yourself a “computer engineer”
- You want to build software people use (need performance)
- You need to make a purchasing decision or offer “expert” advice

- Both Hardware and Software affect performance:
  - Algorithm determines number of source-level statements
  - Language/Compiler/Architecture determine machine instructions (Chapter 2 and 3)
  - Processor/Memory determine how fast instructions are executed (Chapter 5, 6, and 7)

- Assessing and Understanding Performance in Chapter 4

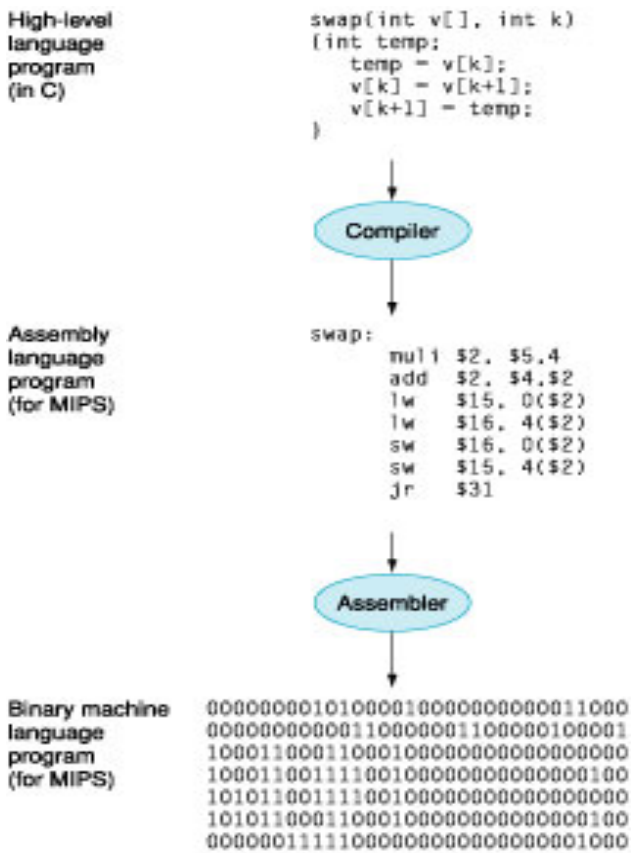
What is a computer?

- **Components:**
  - **input (mouse, keyboard)**
  - **output (display, printer)**
  - **memory (disk drives, DRAM, SRAM, CD)**
  - **network**
- **Our primary focus: the processor (datapath and control)**
  - **implemented using millions of transistors**
  - **Impossible to understand by looking at each transistor**
  - **We need...**

Abstraction

- **Delving into the depths reveals more information**
- **An abstraction omits unneeded detail, helps us cope with complexity**

What are some of the details that appear in these familiar abstractions?

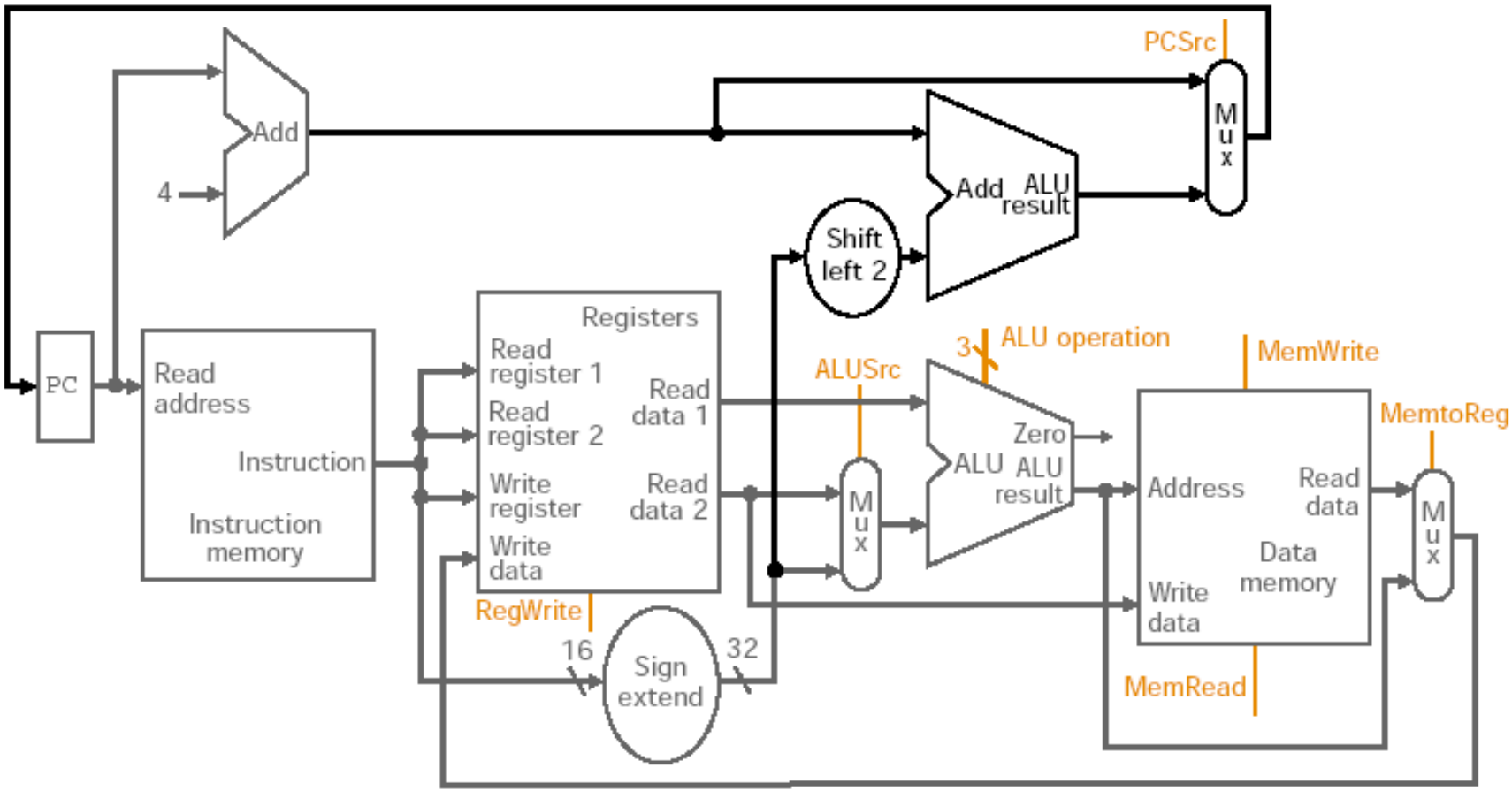


Instruction Set Architecture

- **A very important abstraction**
  - **interface between hardware and low-level software**
  - **standardizes instructions, machine language bit patterns, etc.**
  - **advantage: *different implementations of the same architecture***
  - **disadvantage: *sometimes prevents using new innovations***

- **Modern instruction set architectures:**
  - **IA-32, PowerPC, MIPS, SPARC, ARM, and others**

Sneak Peak (Lecture 18)



Chapter 4

Performance

- **Measure, Report, and Summarize**
- **Make intelligent choices**
- **See through the marketing hype**
- **Key to understanding underlying organizational motivation**

Why is some hardware better than others for different programs?

What factors of system performance are hardware related?
(e.g., Do we need a new machine, or a new operating system?)

How does the machine's instruction set affect performance?

Which of these airplanes has the best performance?

Airplane	Passengers	Range (mi)	Speed (mph)
Boeing 737-100	101	630	598
Boeing 747	470	4150	610
BAC/Sud Concorde	132	4000	1350
Douglas DC-8-50	146	8720	544

- **How much faster is the Concorde compared to the 747?**
- **How much bigger is the 747 than the Douglas DC-8?**

Computer Performance: TIME, TIME, TIME

- **Response Time (latency)**
  - **— How long does it take for my job to run?**
  - **— How long does it take to execute a job?**
  - **— How long must I wait for the database query?**
- **Throughput**
  - **— How many jobs can the machine run at once?**
  - **— What is the average execution rate?**
  - **— How much work is getting done?**

- ***If we upgrade a machine with a new processor what do we increase?***
- ***If we add a new machine to the lab what do we increase?***

Execution Time

- **Elapsed Time**
  - **counts everything *(disk and memory accesses, I/O , etc.)***
  - **a useful number, but often not good for comparison purposes**
- **CPU time**
  - **doesn't count I/O or time spent running other programs**
  - **can be broken up into system time, and user time**

- **Our focus: user CPU time**
  - **time spent executing the lines of code that are "in" our program**

Book's Definition of Performance

- **For some program running on machine X,**

  $$\text{Performance}_X = 1 / \text{Execution time}_X$$

- **"X is n times faster than Y"**

  $$\text{Performance}_X \ / \ \text{Performance}_Y = n$$

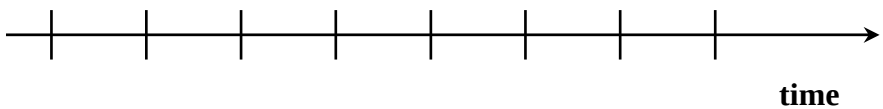
- **Problem:**
  - **machine A runs a program in 20 seconds**
  - **machine B runs the same program in 25 seconds**

Clock Cycles

- **Instead of reporting execution time in seconds, we often use cycles**

$$\frac{\text{seconds}}{\text{program}} = \frac{\text{cycles}}{\text{program}} \times \frac{\text{seconds}}{\text{cycle}}$$

- **Clock "ticks" indicate when to start activities (one abstraction):**



- **cycle time = time between ticks = seconds per cycle**
- **clock rate (frequency) = cycles per second (1 Hz. = 1 cycle/sec)**

A 4 Ghz. clock has a $\frac{1}{4\times10^{9}}\times10^{12} = 250\ \text{picoseconds (ps)}$ **cycle time**

How to Improve Performance

$$\frac{\text{seconds}}{\text{program}} = \frac{\text{cycles}}{\text{program}} \times \frac{\text{seconds}}{\text{cycle}}$$

So, to improve performance (everything else being equal) you can either (increase or decrease?)

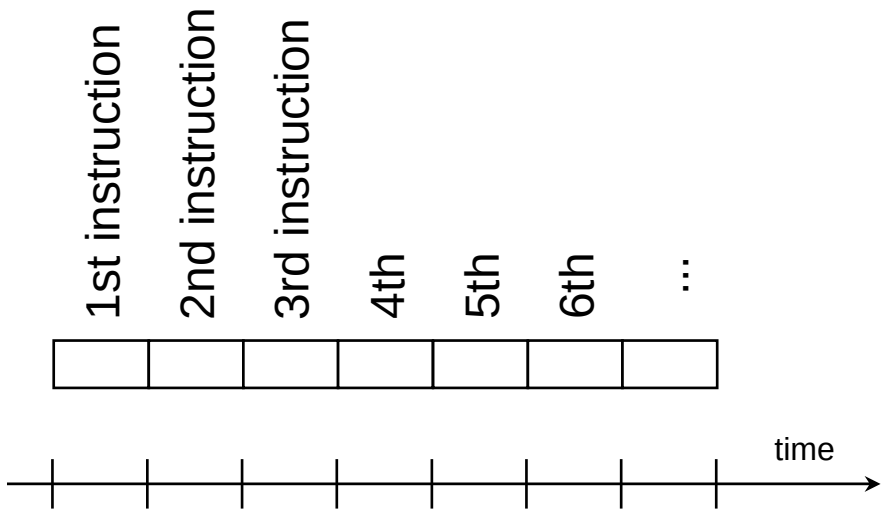
_____ the # of required cycles for a program, or

_____ the clock cycle time or, said another way,

_____ the clock rate.

How many cycles are required for a program?

- **Could assume that number of cycles equals number of instructions**

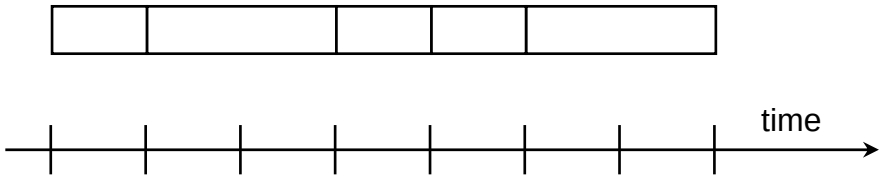


This assumption is incorrect,

different instructions take different amounts of time on different machines.

Why? hint: remember that these are machine instructions, not lines of C code

Different numbers of cycles for different instructions



- **Multiplication takes more time than addition**
- **Floating point operations take longer than integer ones**
- **Accessing memory takes more time than accessing registers**

- ***Important point: changing the cycle time often changes the number of cycles required for various instructions (more later)***

Example

- **Our favorite program runs in 10 seconds on computer A, which has a 4 GHz. clock. We are trying to help a computer designer build a new machine B, that will run this program in 6 seconds. The designer can use new (or perhaps more expensive) technology to substantially increase the clock rate, but has informed us that this increase will affect the rest of the CPU design, causing machine B to require 1.2 times as many clock cycles as machine A for the same program. What clock rate should we tell the designer to target?"**

- **Don't Panic, can easily work this out from basic principles**

Example

$$\frac{\text{seconds}}{\text{program}} = \frac{\text{cycles}}{\text{program}} \times \frac{\text{seconds}}{\text{cycle}}$$

- **Our favorite program runs in 10 seconds on computer A, which has a 4 GHz. clock. We are trying to help a computer designer build a new machine B, that will run this program in 6 seconds. The designer can use new (or perhaps more expensive) technology to substantially increase the clock rate, but has informed us that this increase will affect the rest of the CPU design, causing machine B to require 1.2 times as many clock cycles as machine A for the same program. What clock rate should we tell the designer to target?"**

$$10 \text{ seconds} = \frac{\text{CPU clock cycles}_A}{4 \times 10^9 \frac{\text{cycles}}{\text{second}}}$$

$$\text{CPU clock cycles}_A = 10 \text{ seconds} \times 4 \times 10^9 \frac{\text{cycles}}{\text{second}} = 40 \times 10^9 \text{ cycles}$$

$$\text{CPU time}_B = \frac{1.2 \times \text{CPU clock cycles}_A}{\text{Clock rate}_B}$$

$$6 \text{ seconds} = \frac{1.2 \times 40 \times 10^9 \text{ cycles}}{\text{Clock rate}_B}$$

$$\text{Clock rate}_B = \frac{1.2 \times 40 \times 10^9 \text{ cycles}}{6 \text{ seconds}} = \frac{8 \times 10^9 \text{ cycles}}{\text{second}} = 8 \text{ GHz}$$

Now we understand cycles

- **A given program will require**
  - **some number of instructions (machine instructions)**
  - **some number of cycles**
  - **some number of seconds**
- **We have a vocabulary that relates these quantities:**
  - **cycle time (seconds per cycle)**
  - **clock rate (cycles per second)**
  - **CPI (cycles per instruction)**

    a floating point intensive application might have a higher CPI
  - **MIPS (millions of instructions per second)**

    this would be higher for a program using simple instructions

Back to the Same Formula, CPI (Cycles/Instruction)

$$\frac{\text{seconds}}{\text{program}} = \frac{\text{cycles}}{\text{program}} \times \frac{\text{seconds}}{\text{cycle}}$$

$$\text{Time} = \frac{\text{Seconds}}{\text{Program}} = \frac{\text{Instructions}}{\text{Program}} \times \frac{\text{Clock cycles}}{\text{Instruction}} \times \frac{\text{Seconds}}{\text{Clock cycle}}$$

Always bear in mind that the only complete and reliable measure of computer performance is time. For example, changing the instruction set to lower the instruction count may lead to an organization with a slower clock cycle time that offsets the improvement in instruction count. Similarly, because CPI depends on type of instructions executed, the code that executes the fewest number of instructions may not be the fastest.

Components of performance	Units of measure
CPU execution time for a program	Seconds for the program
Instruction count	Instructions executed for the program
Clock cycles per instruction (CPI)	Average number of clock cycles per instruction
Clock cycle time	Seconds per clock cycle

CPI Example

- **Suppose we have two implementations of the same instruction set architecture (ISA).**

  For some program,

  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

  What machine is faster for this program, and by how much?

$$\frac{\text{seconds}}{\text{program}} = \frac{\text{cycles}}{\text{program}} \times \frac{\text{seconds}}{\text{cycle}}$$

$$\text{CPU clock cycles}_A = I \times 2.0$$

$$\text{CPU clock cycles}_B = I \times 1.2$$

$$\begin{aligned}\text{CPU time}_A &= \text{CPU clock cycles}_A \times \text{Clock cycle time}_A \\ &= I \times 2.0 \times 250 \text{ ps} = 500 \times I \text{ ps}\end{aligned}$$

$$\text{CPU time}_B = I \times 1.2 \times 500 \text{ ps} = 600 \times I \text{ ps}$$

$$\frac{\text{CPU performance}_A}{\text{CPU performance}_B} = \frac{\text{Execution time}_B}{\text{Execution time}_A} = \frac{600 \times I \text{ ps}}{500 \times I \text{ ps}} = 1.2$$

Let's Complicate Things A Little bit…

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 (respectively).

Which sequence will be faster? How much?

$$\frac{\text{seconds}}{\text{program}} = \frac{\text{cycles}}{\text{program}} \times \frac{\text{seconds}}{\text{cycle}}$$

$$\text{CPU clock cycles} = \sum_{i=1}^{n} (\text{CPI}_i \times \text{C}_i)$$

$$\text{CPU clock cycles}_1 = (2 \times 1) + (1 \times 2) + (2 \times 3) = 2 + 2 + 6 = 10 \text{ cycles}$$

$$\text{CPU clock cycles}_2 = (4 \times 1) + (1 \times 2) + (1 \times 3) = 4 + 2 + 3 = 9 \text{ cycles}$$

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.

What is the CPI for each sequence?

$$\text{CPI} = \frac{\text{CPU clock cycles}}{\text{Instruction count}}$$

$$\text{CPI}_1 = \frac{\text{CPU clock cycles}_1}{\text{Instruction count}_1} = \frac{10}{5} = 2$$

$$\text{CPI}_2 = \frac{\text{CPU clock cycles}_2}{\text{Instruction count}_2} = \frac{9}{6} = 1.5$$

Scary Stuff

Op	Frequency	Cycle Count
ALU	43%	1
Load	21%	1
Store	12%	2
Branch	24%	2

Let’s say we were able to reduce the cycle count for “Store” operations to 1 with a cost of slowing our clock by15%. Is this new design feasible?

Example(Contd.)

$$CPI_{original} = \frac{\left(\sum_{i=1}^{n} CPI_i \times IC_i\right)}{Instruction\_Count} = \sum_{i=1}^{n} CPI_i \times \left(\frac{IC_i}{Instruction\_Count}\right)$$

Old CPI = 0.43 + 0.21 + 0.12x2 + 0.24x2 = 1.38

New CPI = 0.43 + 0.21 + 0.12 + 0.24x2 = 1.24

Speed up = old time/new time
= (ICx oldCPI x T)/(IC x newCPI x 1.15T)
=0.97

so, don't make this change.

Component Analysis

Hardware or software component	Affects what?	How?
Algorithm	Instruction count, possibly CPI	The algorithm determines the number of source program instructions executed and hence the number of processor instructions executed. The algorithm may also affect the CPI, by favoring slower or faster instructions. For example, if the algorithm uses more floating-point operations, it will tend to have a higher CPI.
Programming language	Instruction count, CPI	The programming language certainly affects the instruction count, since statements in the language are translated to processor instructions, which determine instruction count. The language may also affect the CPI because of its features; for example, a language with heavy support for data abstraction (e.g., Java) will require indirect calls, which will use higher-CPI instructions.
Compiler	Instruction count, CPI	The efficiency of the compiler affects both the instruction count and average cycles per instruction, since the compiler determines the translation of the source language instructions into computer instructions. The compiler's role can be very complex and affect the CPI in complex ways.
Instruction set architecture	Instruction count, clock rate, CPI	The instruction set architecture affects all three aspects of CPU performance, since it affects the instructions needed for a function, the cost in cycles of each instruction, and the overall clock rate of the processor.

What is MIPS?

million instructions per second (MIPS) A measurement of program execution speed based on the number of millions of instructions. MIPS is computed as the instruction count divided by the product of the execution time and 10^6.

$$\text{MIPS} = \frac{\text{Instruction count}}{\text{Execution time} \times 10^6}$$

- **Instruction execution rate => higher is better**
- **Issues:**
  - **Can not compare processors with different instruction sets**
  - **Varies between programs on the same processor**
  - **Can vary inversely with the performance… ?**

MIPS Example

	CPI for this instruction class		
	A	B	C
CPI	1	2	3

Code from	Instruction counts (in billions) for each instruction class		
	A	B	C
Compiler 1	5	1	1
Compiler 2	10	1	1

Assume that the computer's clock rate is 4 GHz. Which code sequence will execute faster according to MIPS? According to execution time?

$$\text{MIPS} = \frac{\text{Instruction count}}{\text{Execution time} \times 10^6}$$

$$\text{MIPS}_1 = \frac{(5+1+1)\times 10^9}{2.5\times 10^6} = 2800$$

$$\text{MIPS}_2 = \frac{(10+1+1)\times 10^9}{3.75(30)\times 10^6} = 3200$$

$$\text{Execution time} = \frac{\text{CPU clock cycles}}{\text{Clock rate}}$$

$$\text{CPU clock cycles} = \sum_{i=1}^{n} (\text{CPI}_i \times \text{C}_i)$$

$$\text{Execution time}_1 = \frac{10\times 10^9}{4\times 10^9} = 2.5 \text{ seconds}$$

$$\text{Execution time}_2 = \frac{15\times 10^9}{4\times 10^9} = 3.75 \text{ seconds}$$

$$\text{CPU clock cycles}_1 = (5\times 1 + 1\times 2 + 1\times 3)\times 10^9 = 10\times 10^9$$

$$\text{CPU clock cycles}_2 = (10\times 1 + 1\times 2 + 1\times 3)\times 10^9 = 15\times 10^9$$

Benchmarks

- Performance best determined by running a real application
  - Use programs typical of expected workload
  - Or, typical of expected class of applications
    e.g., compilers/editors, scientific applications, graphics, etc.
- Small benchmarks
  - nice for architects and designers
  - easy to standardize
  - can be abused
- SPEC (System Performance Evaluation Cooperative)
  - companies have agreed on a set of real program and inputs
  - valuable indicator of performance (and compiler technology)
  - can still be abused

Benchmark Games

- ***An embarrassed Intel Corp. acknowledged Friday that a bug in a software program known as a compiler had led the company to overstate the speed of its microprocessor chips on an industry benchmark by 10 percent. However, industry analysts said the coding error…was a sad commentary on a common industry practice of "cheating" on standardized performance tests…The error was pointed out to Intel two days ago by a competitor, Motorola …came in a test known as SPECint92…Intel acknowledged that it had "optimized" its compiler to improve its test scores. The company had also said that it did not like the practice but felt to compelled to make the optimizations because its competitors were doing the same thing…At the heart of Intel's problem is the practice of "tuning" compiler programs to recognize certain computing problems in the test and then substituting special handwritten pieces of code…***

  Saturday, January 6, 1996 New York Times

SPEC CPU2000

Integer benchmarks		FP benchmarks	
Name	Description	Name	Type
gzip	Compression	wupwise	Quantum chromodynamics
vpr	FPGA circuit placement and routing	swim	Shallow water model
gcc	The Gnu C compiler	mgrid	Multigrid solver in 3-D potential field
mcf	Combinatorial optimization	applu	Parabolic/elliptic partial differential equation
crafty	Chess program	mesa	Three-dimensional graphics library
parser	Word processing program	galgel	Computational fluid dynamics
eon	Computer visualization	art	Image recognition using neural networks
perlbmk	perl application	equake	Seismic wave propagation simulation
gap	Group theory, interpreter	facerec	Image recognition of faces
vortex	Object-oriented database	ammp	Computational chemistry
bzip2	Compression	lucas	Primality testing
twolf	Place and rote simulator	fma3d	Crash simulation using finite-element method
		sixtrack	High-energy nuclear physics accelerator design
		apsi	Meteorology: pollutant distribution

FIGURE 4.5 The SPEC CPU2000 benchmarks. The 12 integer benchmarks in the left half of the table are written in C and C++, while the floating-point benchmarks in the right half are written in Fortran (77 or 90) and C. For more information on SPEC and on the SPEC benchmarks, see www.spec.org. The SPEC CPU benchmarks use wall clock time as the metric, but because there is little I/O, they measure CPU performance.

Remember

- **Performance is specific to a particular program**
  - **Total execution time is a consistent summary of performance**

- **For a given architecture performance increases come from:**
  - **increases in clock rate (without adverse CPI affects)**
  - **improvements in processor organization that lower CPI**
  - **compiler enhancements that lower CPI and/or instruction count**
  - **Algorithm/Language choices that affect instruction count**

- **Pitfall: expecting improvement in one aspect of a machine's performance to affect the total performance**

Problems with Benchmarking

- **Hard to evaluate real benchmarks:**
  - **Machine not built yet, simulators too slow**
  - **Benchmarks not ported**
  - **Compilers not ready**

Benchmark performance is composition of hardware and software (program, input, compiler, OS) performance, which must all be specified

Performance Measurement Overview

$$CPUtime = CPUclock\_cycles\_for\_the\_pogram \times Clock\_Cycle\_Time$$

$$CPUtime = \frac{CPUclock\_cycles\_for\_the\_pogram}{Clock\_Rate}$$

$$CPI = \frac{CPUclock\_cycles\_for\_the\_pogram}{IC}$$

$$CPUtime = IC \times CPI \times Clock\_Cycle\_Time$$

$$CPUtime = \frac{IC \times CPI}{Clock\_Rate}$$

$$CPUtime = \frac{Seconds}{\text{Pr}ogram} = \frac{Instructions}{\text{Pr}ogram} \times \frac{ClcokCycles}{Instruction} \times \frac{Seconds}{ClockCycle}$$

Performance Measurement Overview

$$CPU_{clock\_cycles\_for\_the\_program} = \sum_{i=1}^{n} CPI_i \times IC_i$$

$$CPUtime = \left(\sum_{i=1}^{n} CPI_i \times IC_i \right) \times Clock\_Cycle\_Time$$

$$overall\_CPI = \frac{\left(\sum_{i=1}^{n} CPI_i \times IC_i \right)}{Instruction\_Count} = \sum_{i=1}^{n} CPI_i \times \left(\frac{IC_i}{Instruction\_Count} \right)$$

Exercise

Suppose we have made the following measurements:

- **Frequency of FP operations = 25%**
- **Average CPI of FP operations = 4.0**
- **Average CPI of other instructions = 1.33**
- **Frequency of FPSQR = 2%**
- **CPI of FPSQR = 20**

Assume that the two design alternatives are:

a) to reduce the CPI of FPSQR to 2 or

b) to reduce the average CPI of all FP operations to 2.

Compare these alternatives.

Solution

Suppose we have made the following measurements

- Frequency of FP operations = 25%
- Average CPI of FP operations = 4.0
- Average CPI of other instructions = 1.33
- Frequency of FPSQR = 2%
- CPI of FPSQR = 20

a) to reduce the CPI of FPSQR to 2 or

b) to reduce the average CPI of all FP operations to 2

$$CPI_{original} = \frac{\left(\sum_{i=1}^{n} CPI_i \times IC_i\right)}{Instruction\_Count} = \sum_{i=1}^{n} CPI_i \times \left(\frac{IC_i}{Instruction\_Count}\right)$$

$$= 4 \times 25\% + 1.33 \times 75\% = 2.0$$

$$CPI_{Saved\_on\_FPSQR} = 2\% \times (CPI_{oldFPSQR} - CPI_{newFPSQR}) = 2\% \times (20 - 2) = 0.36$$

$$CPI_{overall\_for\_new\_FPSQR} = CPI_{original} - CPI_{Saved\_on\_FPSQR} = 2 - 0.36 = 1.64$$

$$CPI_{overall\_for\_new\_FP} = 75\% \times 1.33 + 25\% \times 2.0 = 1.5$$

$$SpeedupFP = \frac{CPUTime_{original}}{CPUTime_{new}} = \frac{IC \times ClockCycle \times CPI_{original}}{IC \times ClockCycle \times CPI_{new}} = \frac{CPI_{original}}{CPI_{new}} = \frac{2.00}{1.5} = 1.33$$

Where are we now?

- **Chapter 4 : Performance Issues**
- **Chapter 1: For you to read**
- **Instruction Set Architecture is coming up**
  - **But, first cover basics**
    - ALU and ISA relationship
    - Some details in ALU
- **Chapter 3**
- **Chapter 2**

Amdahl's Law

- ***The performance enhancement of an improvement is limited by how much the improved feature is used. In other words: Don't expect an enhancement proportional to how much you enhanced something.***

$$\text{Amdahl's law: Execution time after improvement} = \left(\frac{\text{Execution time affected by improvement}}{\text{Amount of improvement}} + \text{Execution time unaffected} \right)$$

- **Example:**

"Suppose a program runs in 100 seconds on a machine, with multiply operations responsible for 80 seconds of this time. How much do we have to improve the speed of multiplication if we want the program to run 4 times faster?"

How about making it 5 times faster?

Amdahl's Law

1. Speed up = 4
2. Old execution time = 100
3. New execution time = 100/4 = 25
4. If 80 seconds is used by the affected part =>
5. Unaffected part = 100-80 = 20 sec
6. Execution time new = Execution time unaffected + Execution time affected / Improvement
7. 25= 20 + 80/Improvement
8. Improvement = 16

Example: Speed up using parallel processors

Suppose an application is “almost all” parallel: 90%.
What is the speedup using 10, 100, and 1000 processors?

Amdahl’s law: Execution time after improvement

$$= \left(\frac{\text{Execution time affected by improvement}}{\text{Amount of improvement}} + \text{Execution time unaffected} \right)$$

new time = old time * 10% + (old time * 90%) / 10

Speed up (P=10) = old time / new time

Speedup (P=10) = 5.3

Speedup (P = 100) = 9.1

Speedup (P = 1000) = 9.9

Amdahl's Law Overview

$$\text{Speedup} = \frac{\text{Execution time before improvement}}{\text{Execution time after improvement}}$$

$$\text{Execution time after improvement} = \frac{\text{Execution time affected by improvement}}{\text{Amount of improvement}} + \text{Execution time unaffected}$$

$$= \frac{\text{Execution time affected} + \text{Amount of improvement} \times \text{Execution time unaffected}}{\text{Amount of improvement}}$$

f is the fraction affected

$$= \frac{\text{Execution time before improvement} \times f}{\text{Amount of improvement}} + \text{Execution time before improvement} \times (1 - f)$$

$$= \left(\frac{f}{\text{Amount of improvement}} + (1 - f) \right) \times \text{Execution time before improvement}$$

$$\text{Speedup} = \frac{\text{Execution time before improvement}}{\left(\frac{f}{\text{Amount of improvement}} + (1 - f) \right) \times \text{Execution time before improvement}}$$

$$\text{Speedup} = \frac{1}{\left(\frac{f}{\text{Amount of improvement}} + (1 - f) \right)}$$

Example

- **Suppose we are considering an enhancement that runs 10times faster than the original machine but is only usable 40% of the time. What is the overall speedup gained by incorporating the enhancement?**

$$\text{Speedup} = \frac{1}{\left(\frac{f}{\text{Amount of improvement}} + (1 - f)\right)}$$

$$Speedup = \frac{1}{\frac{0.4}{10} + 0.6} \cong 1.56$$

Example

- **Implementations of floating point square root vary significantly in performance. Suppose FP square root (FPSQR) is responsible for 20% of the execution time of a critical benchmark on am machine, One proposal is to add FPSQR hardware that will speed up this operation by a factor of 10. The other alternative is just to try to make all FP instructions run faster; FP instructions are responsible for a total of 50% of the execution time. The design team believes that they can make all FP instructions run two times faster with the same effort as required for the fast square root. Compare those two design alternatives.**

$$Speedup_{FPSQR} = \frac{1}{\frac{0.2}{10} + (1 - 0.2)} \cong 1.22$$

$$SpeedupFP = \frac{1}{\frac{0.5}{10} + (1 - 0.5)} \cong 1.33$$

Hardware/Software Tradeoffs

- Which operations are directly supported in “hardware” and which are synthesized in software?

- How much hardware should you employ to speed up certain functions?

	Hardware	Software
Advantages	Speed,	Flexibility,
	Consistency	easier/faster design
		lower cost of errors
Disadvantages	Cost	
	No flexibility	Slowness