Introduction

- Rapidly changing field:
  - vacuum tube → transistor → IC → VLSI (see section 1.4)
  - memory capacity and processor speed is doubling every 1.5 years:
- Things you’ll be learning:
  - how computers work, a basic foundation
  - how to analyze their performance (or how not to!)
  - issues affecting design of modern processors (caches, pipelines)
- Why learn this stuff?
  - You want to design state-of-art system
  - you want to call yourself a “computer scientist or engineer”
  - you want to build software people use (need performance)
  - you need to make a purchasing decision or offer “expert” advice

What is a computer?

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

Abstraction

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

What is Computer Architecture?

- A programmer’s view of machine
- What does it include?

- What is Computer Organization, Structure, and Function?
### 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

True or False: Binary compatibility is extraordinarily important?

- Modern instruction set architectures:
  - 80x86/Pentium/IA-64, PowerPC, DEC Alpha, MIPS, SPARC, HP
- Historical Perspective

### A View of Hardware/Software Hierarchy

- Hardware and software are layered
- Some functions can be moved back and forth
- System software is a collection of programs
  - OS, compilers are some examples
  - It makes job of individuals user easy
- Application software programs
  - Used by many users

### View of Software

- Software means different things to different people

### Internal Structure of a Processor Chip

- Major Components
  - Instruction cache
  - Instruction Fetch
  - Instruction Decode
  - Control/Microcode
  - Register File
  - Data path
  - Data Cache
  - I/O Unit
  - Memory Buffer
  - Advanced Units

### Chip Manufacturing Process

- Performance issues (Chapter 2) vocabulary and motivation
- A specific instruction set architecture (Chapter 3)
- Other instruction set example (From Outside)
- Arithmetic and how to build an ALU (Chapter 4)
- Constructing a processor to execute our instructions (Chapter 5)
- Pipelining to improve performance (Chapter 6)
- Memory: caches and virtual memory (Chapter 7)
- I/O (Chapter 8)

Key to a good grade: attending classes, reading the book!
**Historical Perspective**

- 1642 Pascal: Mechanical Computer
- 1671: Gottfried Leibniz ADD/SUB/MUL/DIV
- 1801: Automatic Control of Weaving Process
- 1827: The Difference Engine by Charles Babbage
- 1936: Zuse Z1: electromechanical computers
- 1941: Zuse Z2
- 1943: Zuse Z3
- 1944: ABC at Iowa State (Attanasoff-Berry Computer)
- 1946: ENIAC by von-Neumann machine, father of modern computing

**Difference Engine**

- Based on computing differences, a finite n-th order polynomial can be differentiated n times, which can be represented by a difference
- Example \( y = \sin(x) = x - \frac{x^3}{3!} \)
- To compute value of \( \sin(x) \) at \( x_0, x_1, x_2, x_3, x_4, x_5, x_6, \ldots \), such that difference in two consecutive values is small, we can calculate \( y_0, y_1, y_2, \) and \( y_3 \) by hand and call them \( \Delta y_0, \Delta y_1, \Delta y_2, \) and \( \Delta y_3 \)
- Then first order difference is \( \Delta y_0 = y_1-y_0; \Delta y_1 = y_2-y_1; \Delta y_2 = y_3-y_2; \)
- Second order difference is \( \Delta^2 y_0 = \Delta y_1 - \Delta y_0 = y_2-2y_1+y_0; \) and so on
- Third order difference is \( \Delta^3 y_0 = \Delta^2 y_1 - \Delta^2 y_0 = y_3-3y_2+3y_1-y_0 \)
- Using this we can recursively compute \( \Delta^3 y_1, \Delta^2 y_1, \) and \( \Delta y_1, \) and \( \Delta y_0\)
- And so on....

**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?**

<table>
<thead>
<tr>
<th>Airplane</th>
<th>Passengers</th>
<th>Range (mi)</th>
<th>Speed (mph)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Boeing 737-100</td>
<td>101</td>
<td>630</td>
<td>598</td>
</tr>
<tr>
<td>Boeing 747</td>
<td>450</td>
<td>4150</td>
<td>610</td>
</tr>
<tr>
<td>BAC/Sud Concorde</td>
<td>132</td>
<td>4000</td>
<td>1350</td>
</tr>
<tr>
<td>Douglas DC-8-50</td>
<td>146</td>
<td>8720</td>
<td>544</td>
</tr>
</tbody>
</table>

*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 being 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
For some program running on machine X, 
\[ \text{Performance}_X = \frac{1}{\text{Execution time}_X} \]

"X is n times faster than Y" 
\[ \frac{\text{Performance}_X}{\text{Performance}_Y} = n \]

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

**Book's Definition of Performance**

- Instead of reporting execution time in seconds, we often use cycles 
- 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 200 Mhz. clock has a cycle time 
\[ \frac{1}{200 \times 10^6} \times 10^9 = 5 \text{ nanoseconds} \]

So, to improve performance (everything else being equal) you can either ______ the # of required cycles for a program, or ______ the clock cycle time or, said another way, ______ the clock rate.

**How to Improve Performance**

<table>
<thead>
<tr>
<th>seconds</th>
<th>cycles</th>
<th>seconds</th>
</tr>
</thead>
<tbody>
<tr>
<td>program</td>
<td>program</td>
<td>cycle</td>
</tr>
</tbody>
</table>

So, to improve performance (everything else being equal) you can either ______ 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 # of cycles = # 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 400 Mhz. 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