Chapter 1

Computer Abstractions & Technology

Computer Architecture (Spring 2016)

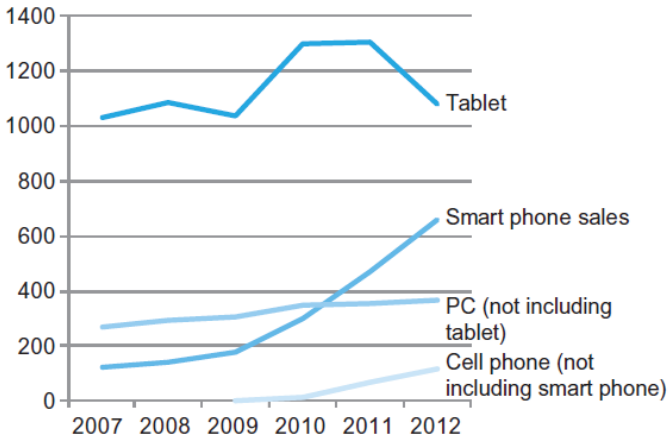
The Computer Revolution

- Progress in computer technology
  - Underpinned by Moore's Law
  - *Moore's Law*: the number of transistors in an IC has doubled approximately every two year
- Makes novel applications feasible
  - Computers in automobiles
  - Cell phones
  - Human genome project
  - World Wide Web
  - Search Engines
- Computers are pervasive

Classes of Computers

- Personal computers
  - General purpose, variety of software
  - Subject to cost/performance tradeoff
- Server computers
  - Network based
  - High capacity, performance, reliability
  - Range from small servers to building sized
- Supercomputers
  - High-end scientific and engineering calculations
  - Highest capability but represent a small fraction of the overall computer market
- Embedded computers
  - Hidden as components of systems
  - Stringent power/performance/cost constraints

The Post-PC Era



The Post-PC Era

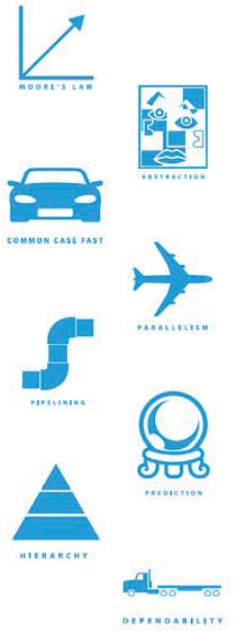
- Personal Mobile Device (PMD)
  - Battery operated
  - Connects to the Internet
  - Hundreds of dollars
  - Smart phones, tablets, electronic glasses
- Cloud computing
  - Warehouse Scale Computers (WSC)
  - Software as a Service (SaaS)
  - A portion of software run on a PMD and a portion run in the Cloud
  - Amazon and Google

What Affects the Performance

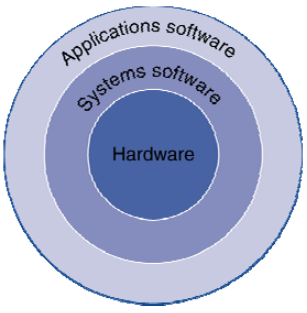
- **Algorithm**
  - Determines number of *operations* executed
- **Programming language, compiler, architecture**
  - Determine number of *machine instructions* executed per operation
- **Processor and memory system**
  - Determine how fast instructions are executed
- **I/O system** (including OS)
  - Determines how fast I/O operations are executed

Eight Great Ideas in Computer Architecture

- Design for *Moore's Law*
- Use *abstraction* to simplify design
- Make the *common case fast*
- Performance *via parallelism*
- Performance *via pipelining*
- Performance *via prediction*
- *Hierarchy* of memories
- *Dependability* via redundancy



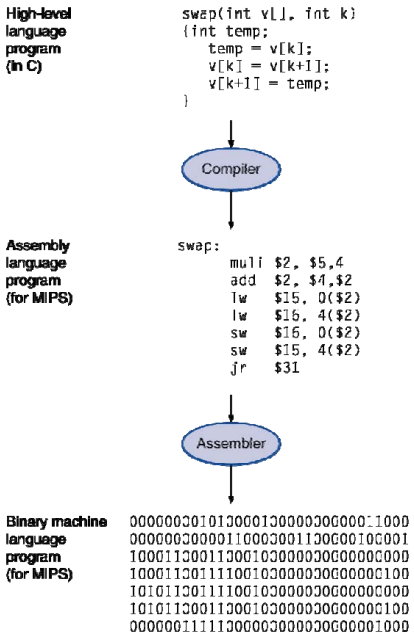
Below Your Program



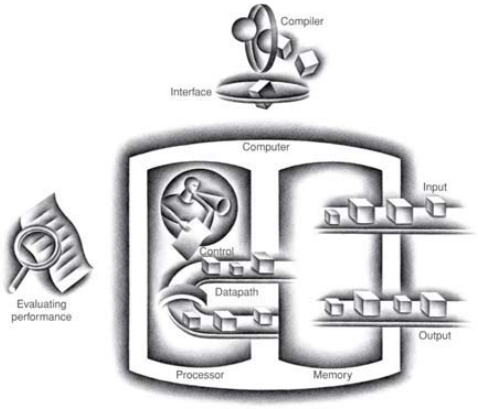
- Application software
  - Written in high-level language
- System software
  - Compiler: translates HLL code to machine code
  - Operating System: service code
    - Handling input/output
    - Managing memory and storage
    - Scheduling tasks & sharing resources
- Hardware
  - Processor, memory, I/O controllers

Levels of Program Code

- High-level language
  - Level of abstraction closer to problem domain
  - Provides for productivity and portability
- Assembly language
  - Textual representation of instructions
- Hardware representation
  - Binary digits (bits)
  - Encoded instructions and data



Components of a Computer

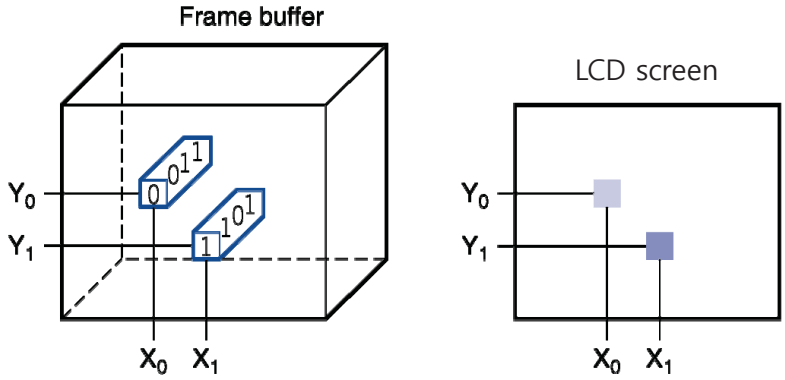


Organization of a computer

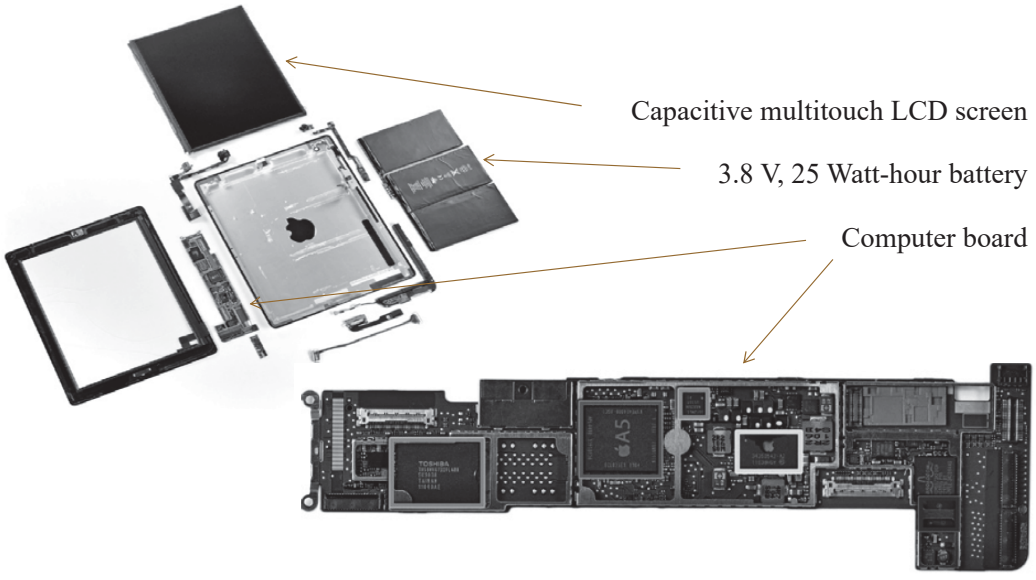
- Same components
  - Desktop
  - Server
  - Embedded
- Input/output includes
  - User-interface devices
    - Display, keyboard, mouse
  - Storage devices
    - Hard disk, CD/DVD, flash
  - Network adapters
    - For communicating with other computers

Through the Looking Glass

- LCD screen: picture elements (pixels)
  - Mirrors content of frame buffer memory



Opening the Box: Case of Apple IPad 2

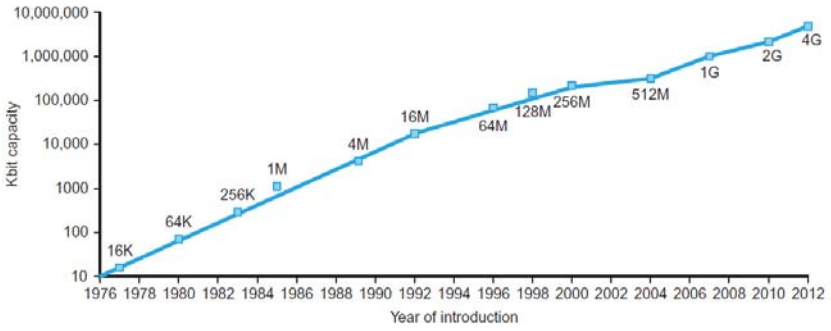


Abstractions

- Abstraction helps us deal with complexity
  - Hide lower-level detail
- *Instruction set architecture* (*ISA*)
  - The hardware/software interface
- *Application binary interface* (*ABI*)
  - The ISA plus system software interface
- Implementation
  - The details underlying and interface

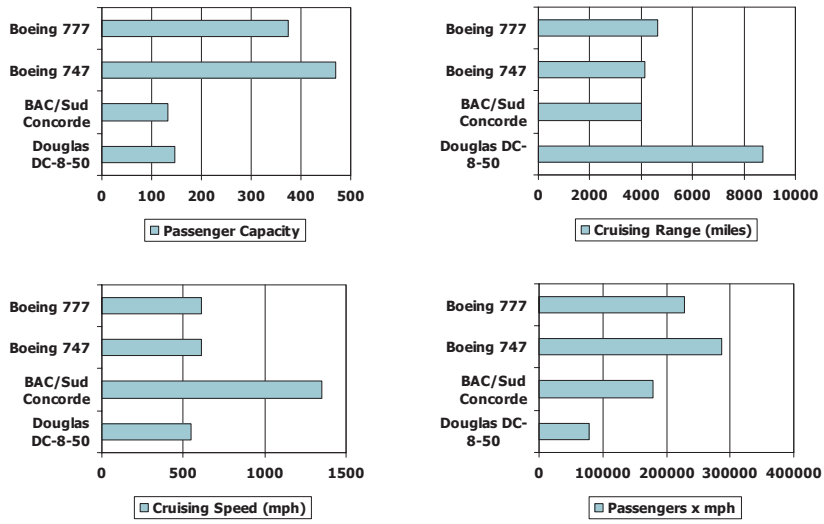
Technology Trends

- Electronics technology continues to evolve
  - Increased capacity and performance
  - Reduced cost
- Growth of capacity of DRAM chip over time



Defining Performance

- Which airplane has the best performance?



Response Time and Throughput

- *Response time*
  - How long it takes to do a task
- *Throughput*
  - Total work done per unit time
  - Example: tasks/transactions/… per hour
- How are response time and throughput affected by
  - Replacing the processor with a faster version?
  - Adding more processors?

Relative Performance

- Define Performance = 1/Execution Time
- "X is *n* time faster than Y"

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

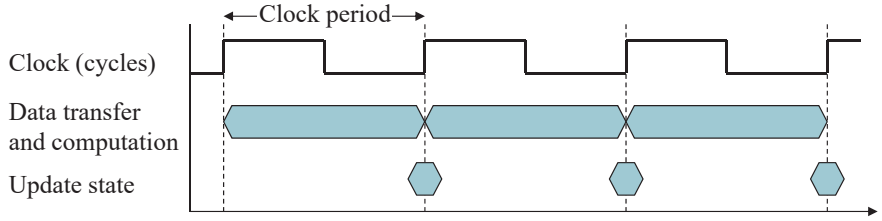
- Example: time taken to run a program
  - 10s on A, 15s on B
  - Execution Time_B / Execution Time_A =
  - So A is times faster than B

Measuring Execution Time

- *Elapsed time*
  - Total response time, including all aspects
    - Processing, I/O, OS overhead, idle time
  - Determines system performance
- *CPU time*
  - Time spent processing a given job
    - Discounts I/O time, other jobs' shares
  - Comprises *user CPU time* and *system CPU time*
  - Different programs are affected differently by CPU and system performance

CPU Clocking

- Operation of digital hardware governed by a constant-rate clock



- *Clock period*: duration of a clock cycle
  - Example: 250 ps = 0.25 ns = s
- *Clock frequency (rate)*: cycles per second
  - Example: 4.0 GHz = 4000 MHz = Hz

CPU Performance Equation

$$\text{CPU Time} = \text{CPU Clock Cycles} \times \text{Clock Cycle Time}$$

$$= \frac{\text{CPU Clock Cycles}}{\text{Clock Rate}}$$

- Performance improved by
  - Reducing number of clock cycles
  - Increasing clock rate
  - Hardware designer must often trade off clock rate against cycle count

CPU Performance Equation

$$\text{CPU Time} = \text{CPU Clock Cycles} \times \text{Clock Cycle Time}$$

$$\text{Clock Cycles} = \sum_{i=1}^{n} (\text{CPI}_i \times \text{Instruction Count}_i)$$

$$\text{CPU Time} = \text{Instruction Count} \times \text{CPI} \times \text{Clock Cycle Time}$$

$$= \frac{\text{Instruction Count} \times \text{CPI}}{\text{Clock Rate}}$$

$$\text{CPI} = \frac{\text{Clock Cycles}}{\text{Instruction Count}} = \sum_{i=1}^{n} \left(\text{CPI}_i \times \frac{\text{Instruction Count}_i}{\text{Instruction Count}} \right)$$

- *CPI* (*Cycles per Instruction*)
  - Each instruction has different CPI
  - Average CPI is affected by instruction mix

Example Problem on Clock Rate

- A program runs **in 10 seconds** on computer A with **2 GHz. Clock**
- A new machine B will run this program **in 6 seconds** with new technology to substantially increase the clock rate
- The increase will cause the 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?

$$\text{Clock Rate}_B = \frac{\text{Clock Cycles}_B}{\text{CPU Time}_B} = \frac{1.2 \times \text{Clock Cycles}_A}{6\text{s}}$$

$$\text{Clock Cycles}_A = \text{CPU Time}_A \times \text{Clock Rate}_A$$

$$= 10\text{s} \times 2\text{GHz} = 20 \times 10^9$$

$$\text{Clock Rate}_B = \frac{1.2 \times 20 \times 10^9}{6\text{s}} = \frac{24 \times 10^9}{6\text{s}} = 4\text{GHz}$$

Example Problem on CPI (1)

- Suppose we have two implementations of the same instruction set architecture (ISA)
- 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 is faster, and by how much?

$$\text{CPU Time}_A = \text{Instruction Count} \times \text{CPI}_A \times \text{Cycle Time}_A$$
$$= \text{IC} \times 2.0 \times 250\text{ps} = \text{IC} \times 500\text{ps}$$
$$\text{CPU Time}_B = \text{Instruction Count} \times \text{CPI}_B \times \text{Cycle Time}_B$$
$$= \text{IC} \times 1.2 \times 500\text{ps} = \text{IC} \times 600\text{ps}$$
$$\frac{\text{CPU Time}_B}{\text{CPU Time}_A} = \frac{\text{IC} \times 600\text{ps}}{\text{IC} \times 500\text{ps}} = 1.2$$

Example Problem on CPI (2)

- Compiled code sequences using instructions in classes A, B, C

Instruction class	A	B	C
CPI for class	1	2	3
IC in sequence 1	2	1	2
IC in sequence 2	4	1	1

■ Which will be faster, and what is the average CPI for each?

Sequence 1
- IC = 5
- Clock cycles = 2×1 + 1×2 + 2×3 = 10
- Average CPI = 10 / 5 = 2

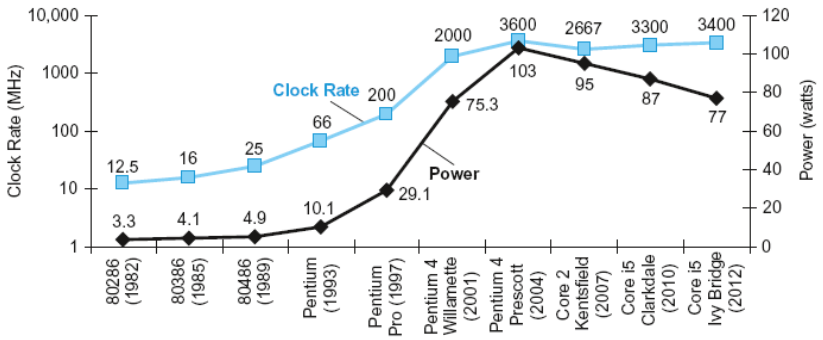
Sequence 2
- IC = 6
- Clock cycles = 4×1 + 1×2 + 1×3 = 9
- Average CPI = 9 / 6 = 1.5

Performance Summary

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

- Performance depends on
  - Algorithm: affects IC, possibly CPI
  - Programming language: affects IC, CPI
  - Compiler: affects IC, CPI
  - Instruction set architecture: affects IC, CPI, CCT

Power Trends



- In CMOS IC technology

$$\text{Power} = \text{Capacitive load} \times \text{Voltage}^2 \times \text{Frequency}$$

×30 5V → 1V ×1000

Example Problem on Reducing Power

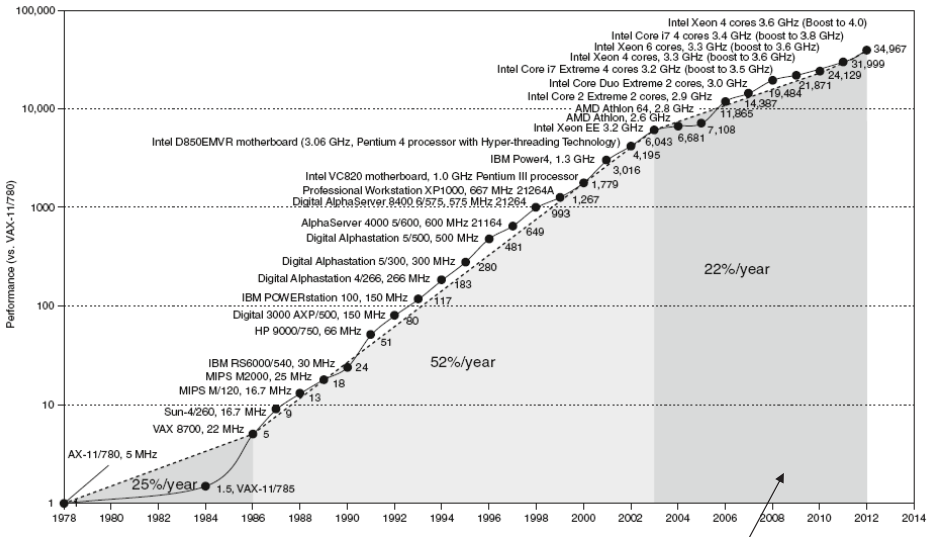
- Suppose a new CPU has 85% of capacitive load of old CPU
- It can run on 15% voltage and 15% frequency reduction

- What is the impact on dynamic power?

$$\frac{P_{new}}{P_{old}} = \frac{C_{old} \times 0.85 \times (V_{old} \times 0.85)^2 \times F_{old} \times 0.85}{C_{old} \times V_{old}^{\ 2} \times F_{old}} = 0.85^4 = 0.52$$

The new processor uses about half the power of the old processor

Uniprocessor Performance



Constrained by power, instruction-level parallelism, memory latency

Multiprocessors

- Multicore microprocessors
  - More than one processor per chip
- Requires explicitly parallel programming
  - Compare with instruction level parallelism
    - Hardware executes multiple instructions at once
    - Hidden from the programmer
  - Hard to do
    - Programming for performance
    - Load balancing
    - Optimizing communication and synchronization

SPEC CPU Benchmark

- Programs used to measure performance
  - Supposedly typical of actual workload
- Standard Performance Evaluation Corp (SPEC)
  - Develops benchmarks for CPU, I/O, Web, …
- SPEC CPU2006
  - Elapsed time to execute a selection of programs
    - Negligible I/O, so focuses on CPU performance
  - Normalize relative to reference machine

$$\sqrt[n]{\prod_{i=1}^{n} \text{Execution time ratio}_i}$$

  - Summarize as geometric mean of performance ratios
    - CINT2006 (integer) and CFP2006 (floating-point)

CINT2006 for Intel Core i7 920 (2.66 GHz)

Description	Name	Instruction Count x 10^9	CPI	Clock cycle time (seconds x 10^{-9})	Execution Time (seconds)	Reference Time (seconds)	SPECratio
Interpreted string processing	perl	2252	0.60	0.376	508	9770	19.2
Block-sorting compression	bzip2	2390	0.70	0.376	629	9650	15.4
GNU C compiler	gcc	794	1.20	0.376	358	8050	22.5
Combinatorial optimization	mcf	221	2.66	0.376	221	9120	41.2
Go game (AI)	go	1274	1.10	0.376	527	10490	19.9
Search gene sequence	hmmer	2616	0.60	0.376	590	9330	15.8
Chess game (AI)	sjeng	1948	0.80	0.376	586	12100	20.7
Quantum computer simulation	libquantum	659	0.44	0.376	109	20720	190.0
Video compression	h264avc	3793	0.50	0.376	713	22130	31.0
Discrete event simulation library	omnetpp	367	2.10	0.376	290	6250	21.5
Games/path finding	astar	1250	1.00	0.376	470	7020	14.9
XML parsing	xalancbmk	1045	0.70	0.376	275	6900	25.1
Geometric mean	–	–	–	–	–	–	25.7

Amdahl's Law

- Pitfall: Improving an aspect of a computer and expecting a proportional improvement in overall performance

$$T_{improved} = \frac{T_{affected}}{\text{improvement factor}} + T_{unaffected}$$

- Example
  - Multiply accounts for 80s/100s
  - How much improvement in multiply performance to get 5× overall?

$$20 = \frac{80}{n} + 20 \qquad \therefore \text{ Impossible}$$

- Corollary: make the common case fast