

# Chapter 1



Computer Abstractions and  
Technology

# The Computer Revolution

- Progress in computer technology
  - Underpinned by Moore's Law
- Makes novel applications feasible
  - Computers in automobiles
  - Cell phones
  - Human genome project
  - World Wide Web
  - Search Engines
- Computers are pervasive

# Bill Gates “joke”

- At a computer expo (COMDEX), Bill Gates reportedly compared the computer industry with the auto industry and stated that:
- "If GM had kept up with technology like the computer industry has, we would all be driving twenty-five dollar cars that got 1000 miles to the gallon."

# GM's "joke" response!

- In response to Gates' comments, General Motors issued a press release stating:  
If GM had developed technology like Microsoft, we would all be driving cars with the following characteristics:
  1. For no reason whatsoever your car would crash twice a day.
  2. Every time they repainted the lines on the road you would have to buy a new car.
  3. Occasionally your car would die on the freeway for no reason, and you would just accept this, restart and drive on.
  4. Occasionally, executing a maneuver such as a left turn, would cause your car to shut down and refuse to restart, in which case you would have to reinstall the engine.
  5. Only one person at a time could use the car, unless you bought "Car95" or "CarNT". But then you would have to buy more seats.
  6. Macintosh/Apple would make a car that was powered by the sun, reliable, five times as fast, and twice as easy to drive, but would only work on five percent of the roads.

# GM “joke” continues...

- 7. The oil, water temperature and alternator warning lights would be replaced by a single "general car default" warning light.
- 8. New seats would force everyone to have the same size butt.
- 9. The airbag system would say "Are you sure?" before going off.
- 10. Occasionally for no reason whatsoever, your car would lock you out and refuse to let you in until you simultaneously lifted the door handle, turned the key, and grab hold of the radio antenna.
- 11. GM would require all car buyers to also purchase a deluxe set of Rand McNally road maps (now a GM subsidiary), even though they neither need them nor want them. Attempting to delete this option would immediately cause the car's performance to diminish by 50% or more. Moreover, GM would become a target for investigation by the Justice Department.
- 12. Every time GM introduced a new model car buyers would have to learn to drive all over again because none of the controls would operate in the same manner as the old car.
- 13. You'd press the "start" button to shut off the engine.

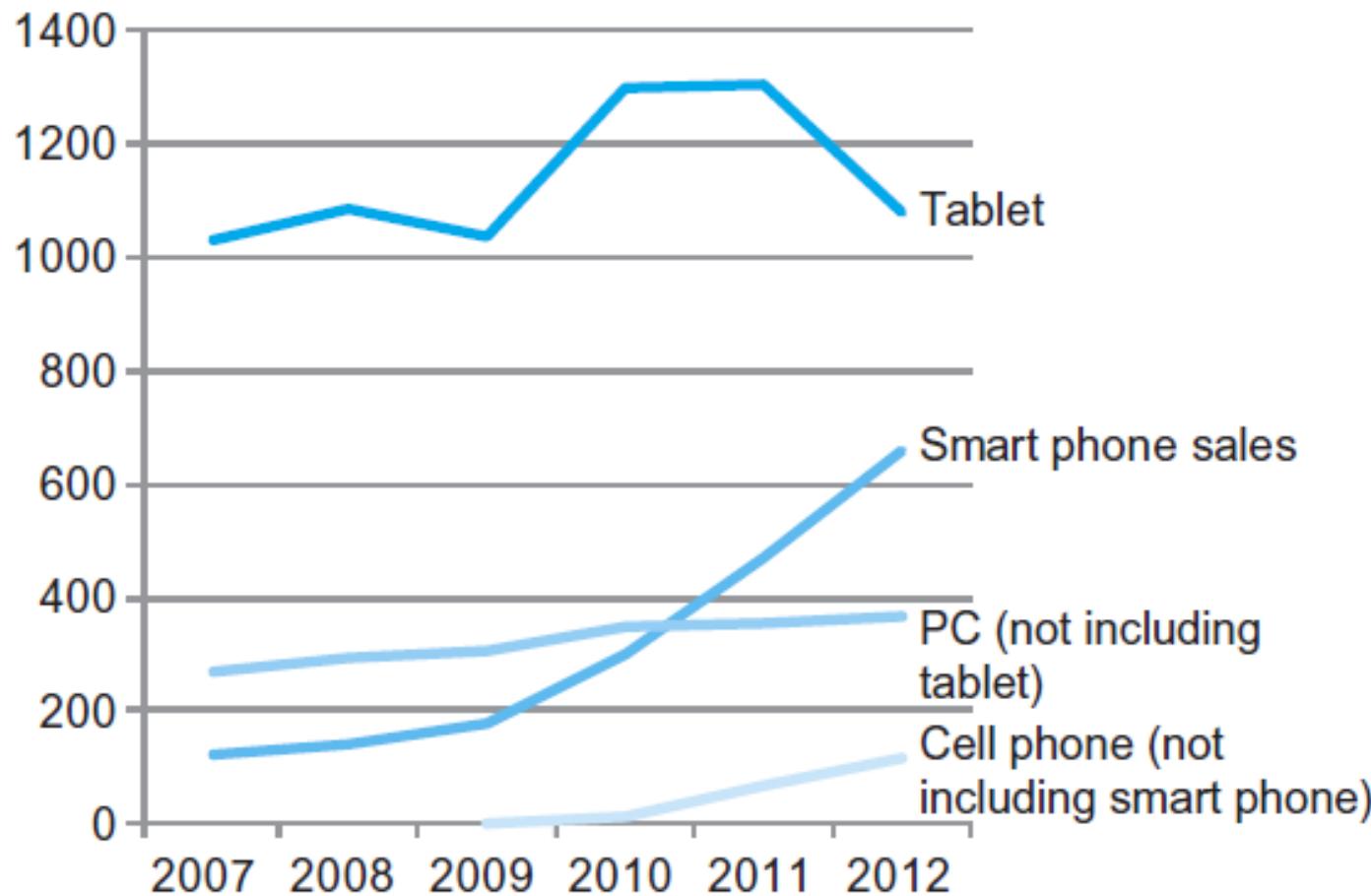
# Classes of Computers

- 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

# Classes of Computers

- 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 PostPC Era



# The PostPC 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)
  - Portion of software run on a PMD and a portion run in the Cloud
  - Amazon and Google

# What You Will Learn

- How programs are translated into the machine language
  - And how the hardware executes them
- The hardware/software interface
- What determines program performance
  - And how it can be improved
- How hardware designers improve performance
- What is parallel processing

# Understanding 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

# Detour

- One of the TAs floated the idea some of you were having pointer difficulty
- Let's revisit some pointer basics

# Pointers - Review

- How do we implement this function?

```
void swap(int x, int y);
```

# Pointers - Review

- How do we implement this function?

~~void swap(int x, int y);~~

(We can't but this can)

`void swap(int *x, int *y);`

# Why the first function can't

- C function arguments are passed by *value*
- This means a *copy* is created and passed to the function
- Anything done to arguments only affect those copies

# How do pointers work?

- Passing something by *address* allows you to modify data *at that address*, ex.

```
foo(int *x) {  
    *x = 17;  
}
```

```
int main() {  
    int value;  
    foo(&value);  
    printf("%d\n", value); // 17!  
    return 0;  
}
```

# What about pointers to arrays?

- In the homework, we had a function to allocate the right amount of memory:

```
void alloc(int **ptr, int count) {  
    // What do we do???  
}
```

# What about pointers to arrays?

- In the homework, we had a function to allocate the right amount of memory:

```
void alloc_square_matrix(int n) {  
    *ptr = (int*)malloc(sizeof(int)*n);  
}
```

```
int main() {  
    int *int_pointer;  
    alloc_square_matrix(5);
```

# Multi-dimensional pointers

- Pointers can also be multi-dimensional as we saw in the quiz

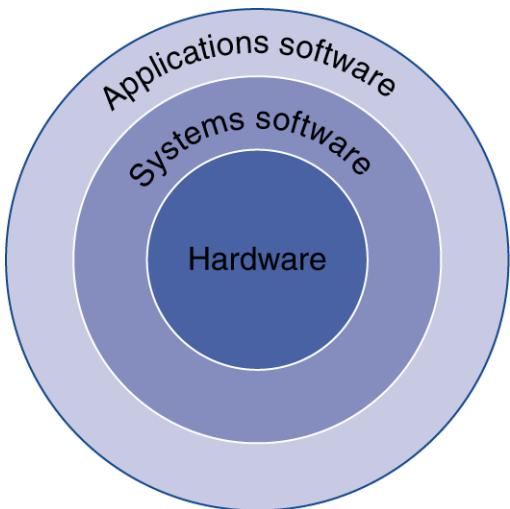
```
int **matrix;  
// Above does almost nothing  
// Just make space for SINGLE POINTER  
void alloc_square_matrix(int n) {  
    matrix = (int**)calloc(n, sizeof(int*));  
  
    for (int i = 0; i < n; i++) {  
        // What goes here??  
        // Pay attention, iClicker question  
    }  
}
```

# Eight Great Ideas

- 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

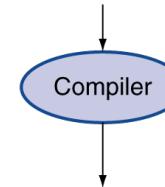
- Textual representation of instructions

## ■ Hardware representation

- Binary digits (bits)
  - Encoded instructions and data

High-level  
language  
program  
(in C)

```
swap(int v[], int k)
{int temp;
  temp = v[k];
  v[k] = v[k+1];
  v[k+1] = temp;
}
```

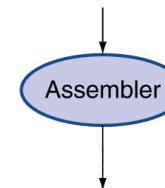


## Assembly language program (for MIPS)

```

swap:
    muli $2, $5,4
    add  $2, $4,$2
    lw   $15, 0($2)
    lw   $16, 4($2)
    sw   $16, 0($2)
    sw   $15, 4($2)
    jr   $31

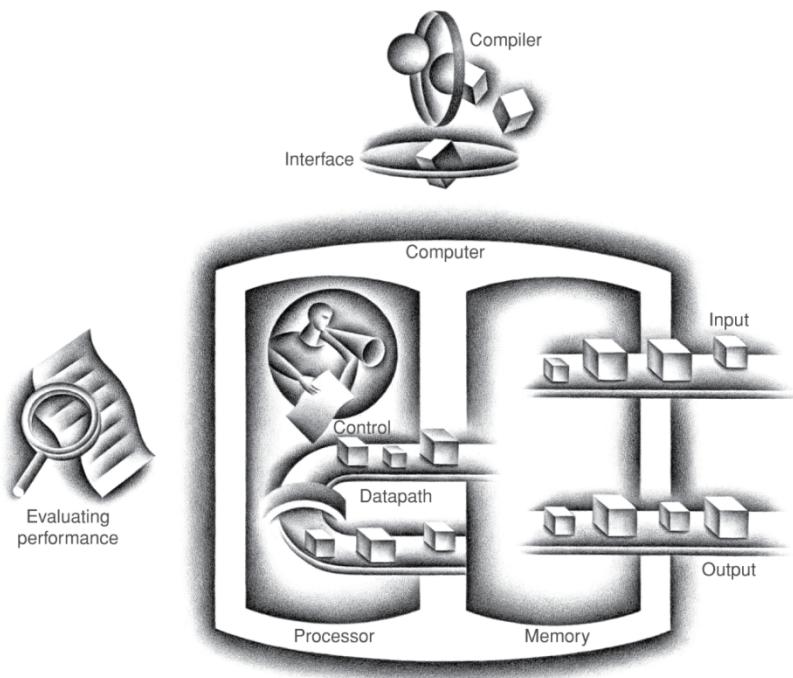
```



Binary machine  
language  
program  
(for MIPS)

# Components of a Computer

## The BIG Picture



- Same components for all kinds of computer
  - 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

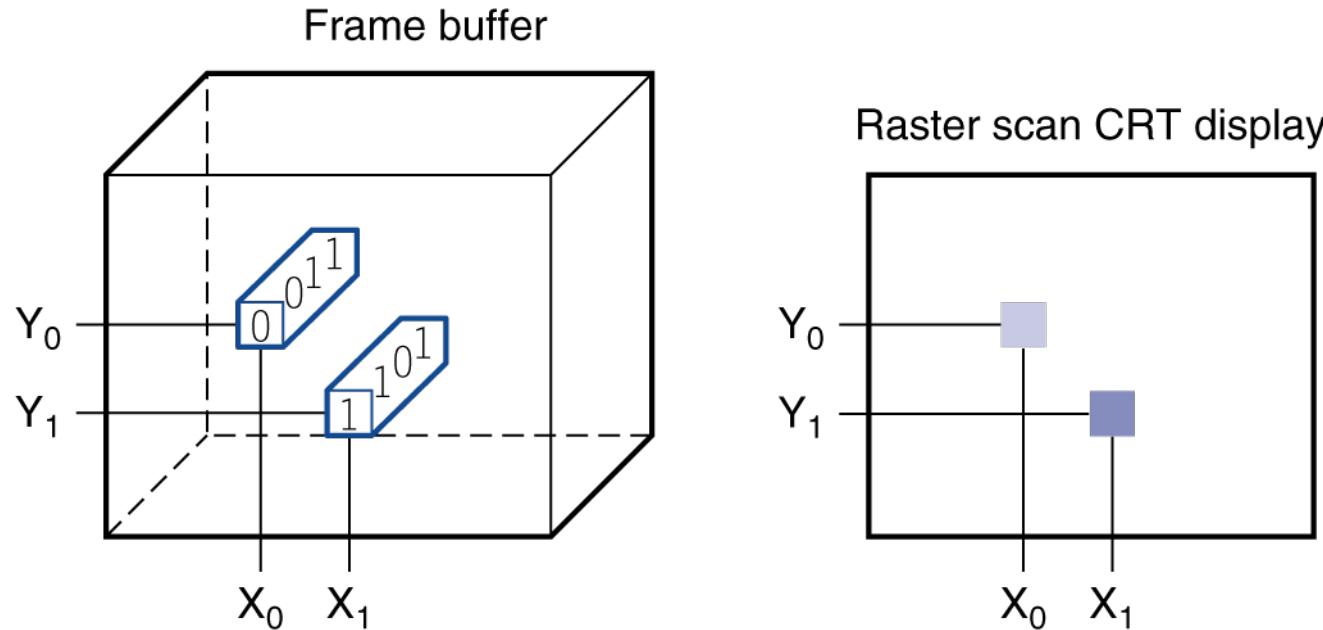
# Touchscreen

- PostPC device
- Supersedes keyboard and mouse
- Resistive and Capacitive types
  - Most tablets, smart phones use capacitive
  - Capacitive allows multiple touches simultaneously

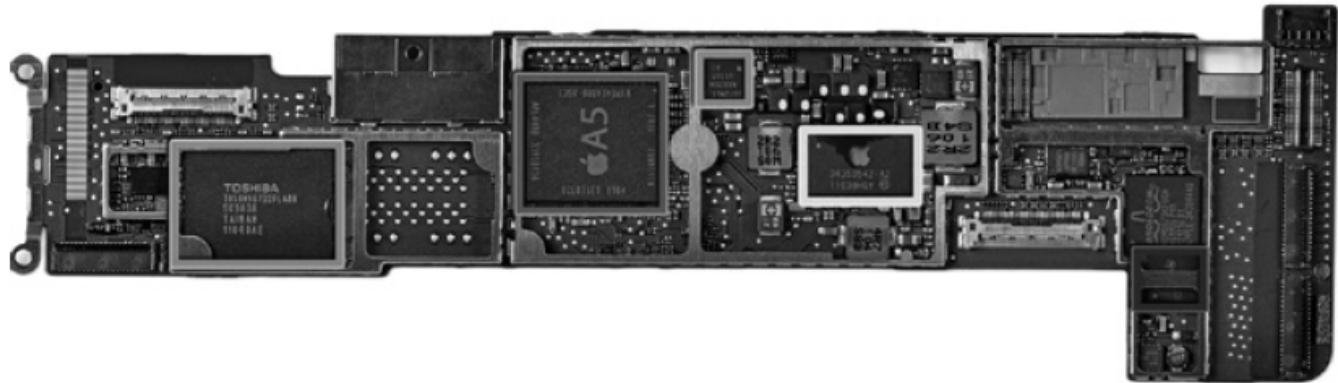
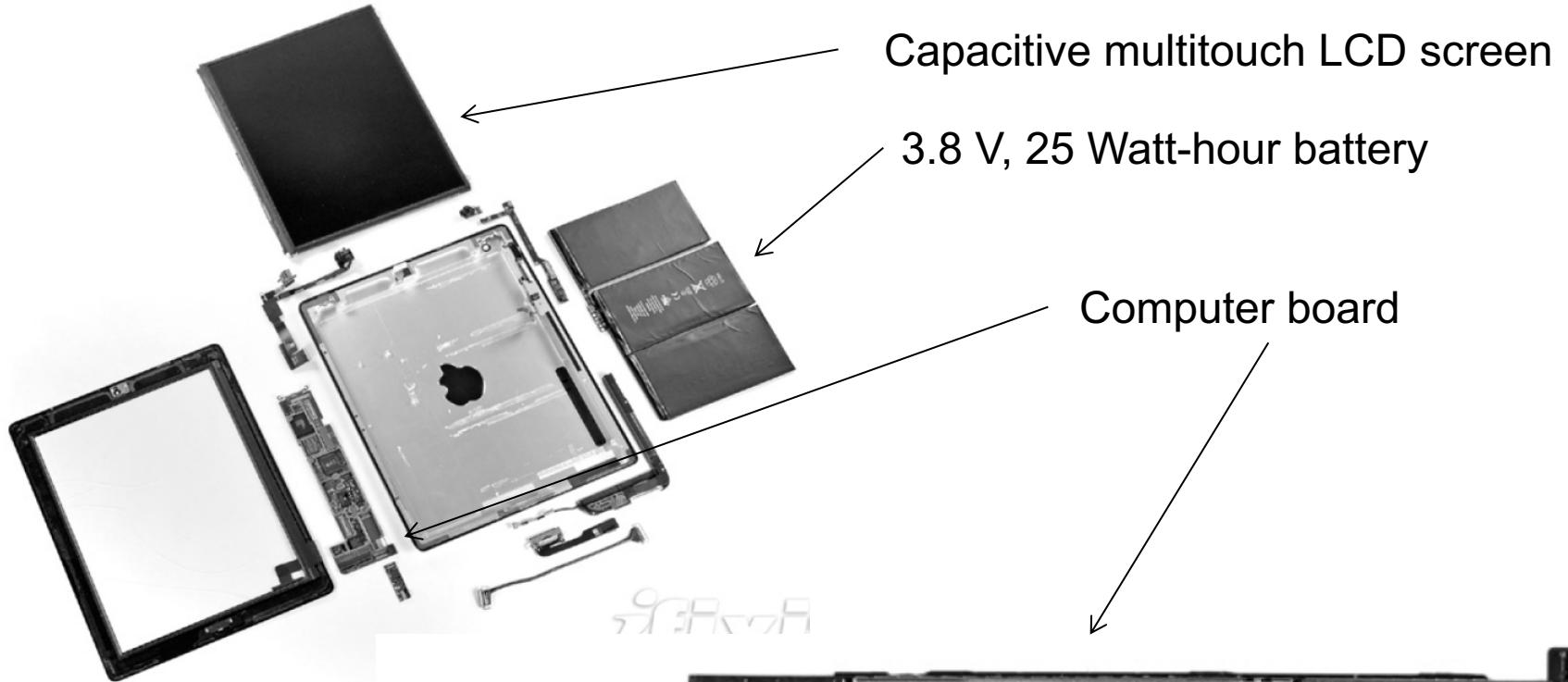


# Through the Looking Glass

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



# Opening the Box



# Inside the Processor (CPU)

- Datapath: performs operations on data
- Control: sequences datapath, memory, ...
- Cache memory
  - Small fast SRAM memory for immediate access to data

# Inside the Processor

## Apple A5



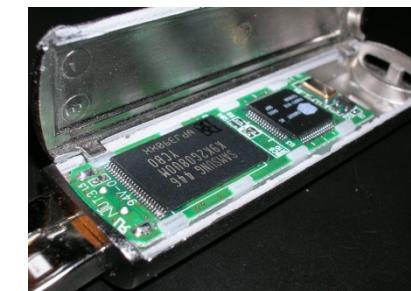
# Abstractions

## The BIG Picture

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

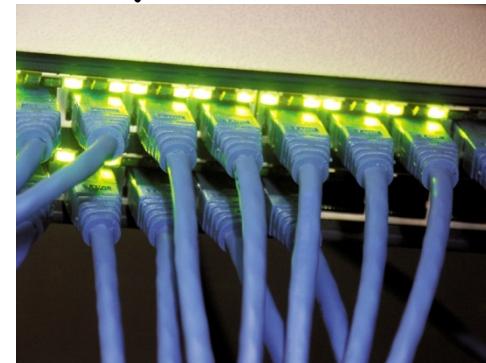
# A Safe Place for Data

- Volatile main memory
  - Loses instructions and data when power off
- Non-volatile secondary memory
  - Magnetic disk
  - Flash memory
  - Optical disk (CDROM, DVD)



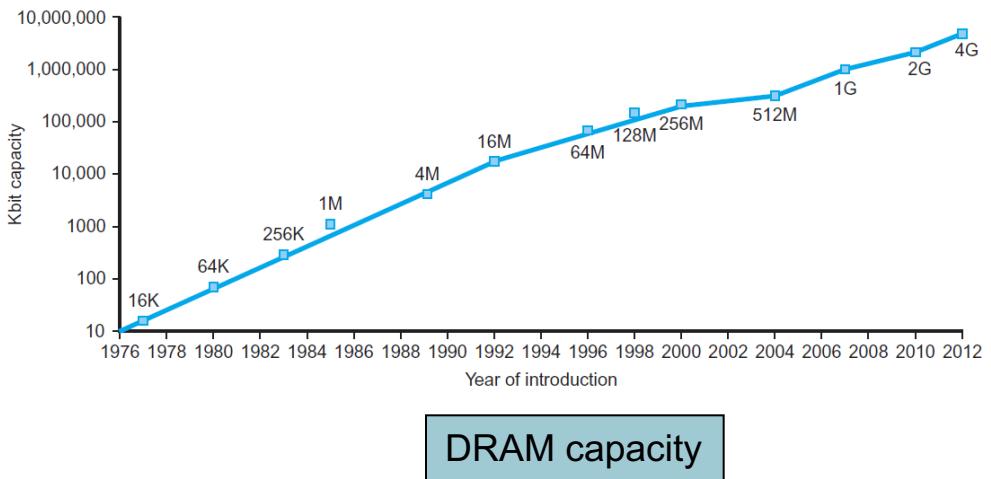
# Networks

- Communication, resource sharing, nonlocal access
- Local area network (LAN): Ethernet
- Wide area network (WAN): the Internet
- Wireless network: WiFi, Bluetooth



# Technology Trends

- Electronics technology continues to evolve
  - Increased capacity and performance
  - Reduced cost

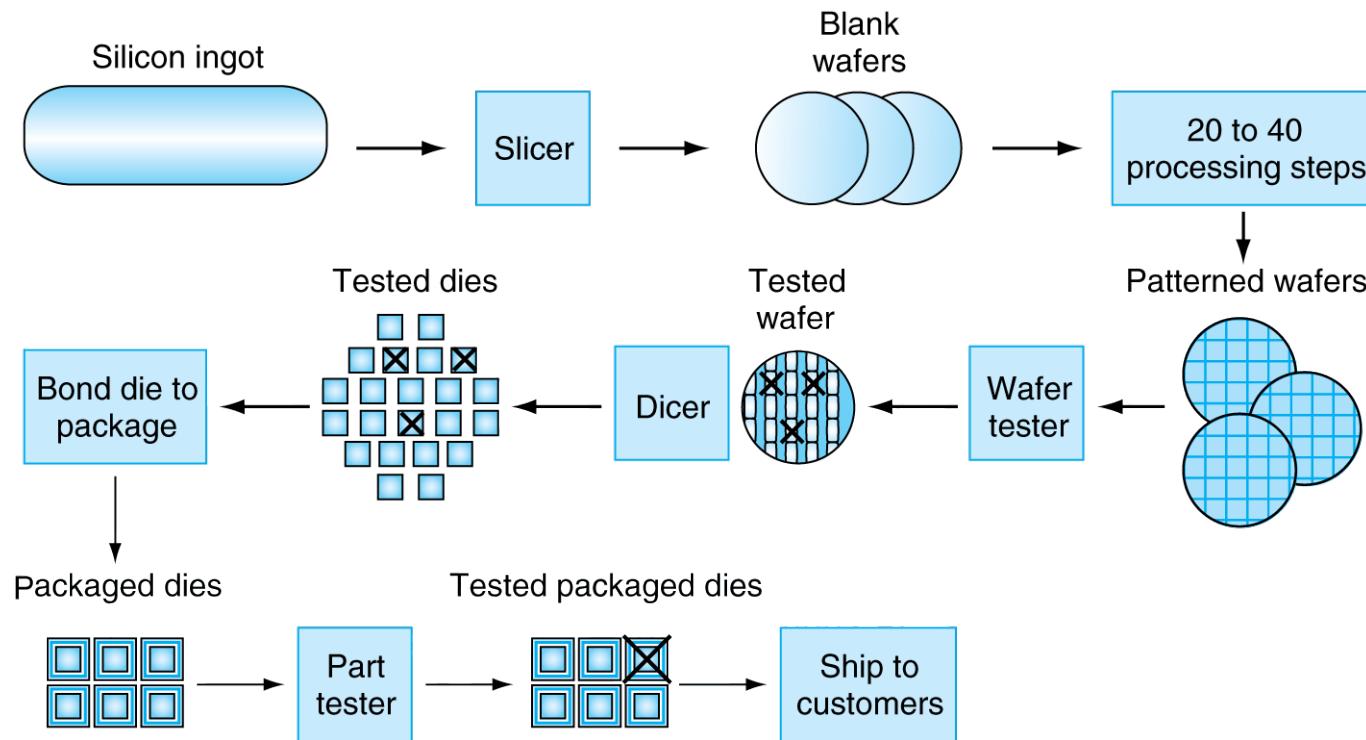


Year	Technology	Relative performance/cost
1951	Vacuum tube	1
1965	Transistor	35
1975	Integrated circuit (IC)	900
1995	Very large scale IC (VLSI)	2,400,000
2013	Ultra large scale IC	250,000,000,000

# Semiconductor Technology

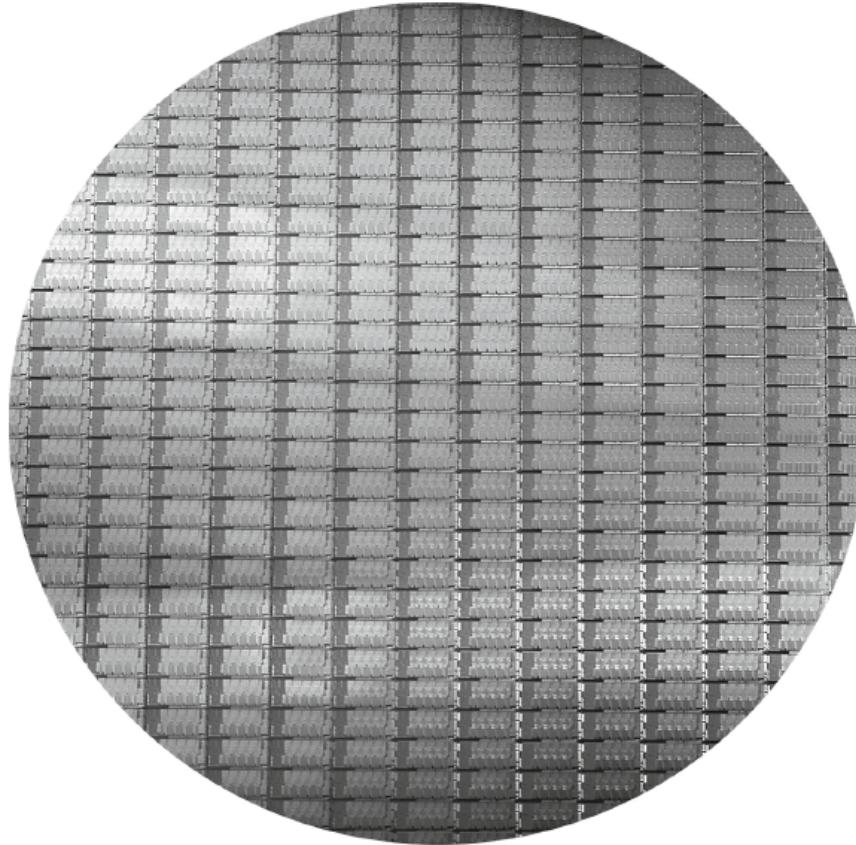
- Silicon: semiconductor
- Add materials to transform properties:
  - Conductors
  - Insulators
  - Switch

# Manufacturing ICs



- Yield: proportion of working dies per wafer

# Intel Core i7 Wafer



- 300mm wafer, 280 chips, 32nm technology
- Each chip is  $20.7 \times 10.5$  mm

# Integrated Circuit Cost

$$\text{Cost per die} = \frac{\text{Cost per wafer}}{\text{Dies per wafer} \times \text{Yield}}$$

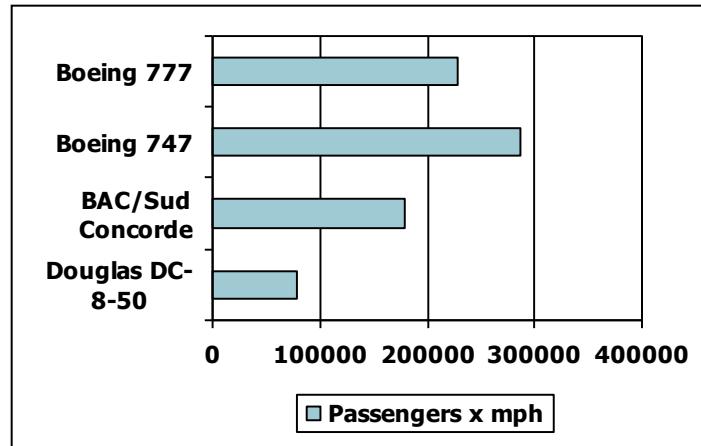
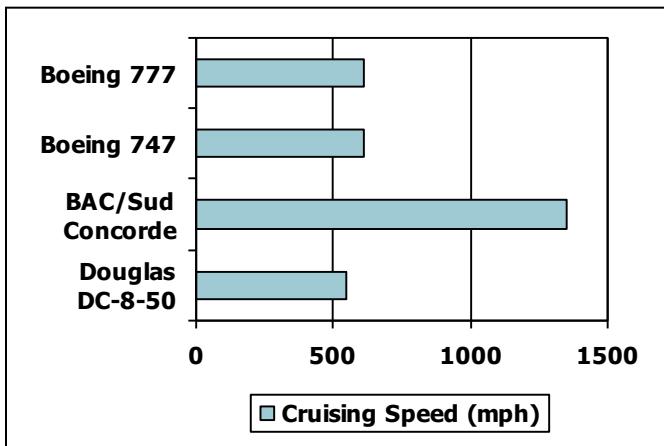
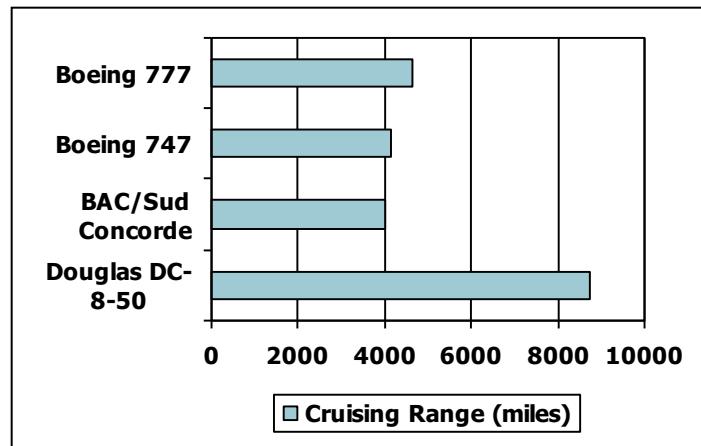
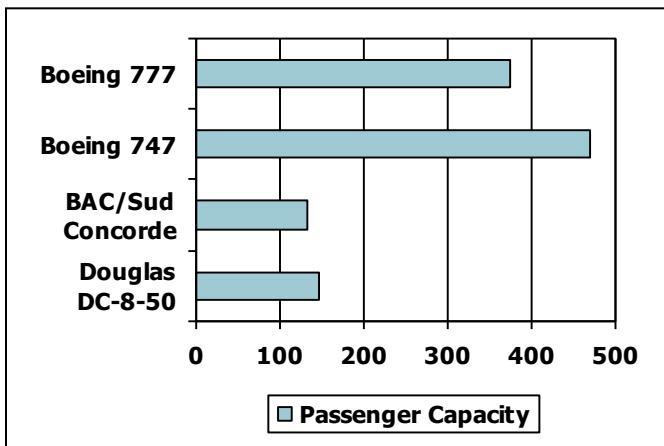
$$\text{Dies per wafer} \approx \text{Wafer area} / \text{Die area}$$

$$\text{Yield} = \frac{1}{(1 + (\text{Defects per area} \times \text{Die area}/2))^2}$$

- Nonlinear relation to area and defect rate
  - Wafer cost and area are fixed
  - Defect rate determined by manufacturing process
  - Die area determined by architecture and circuit design

# 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
    - e.g., tasks/transactions/... per hour
- How are response time and throughput affected by
  - Replacing the processor with a faster version?
  - Adding more processors?
- We'll focus on response time for now...

# Relative Performance

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

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

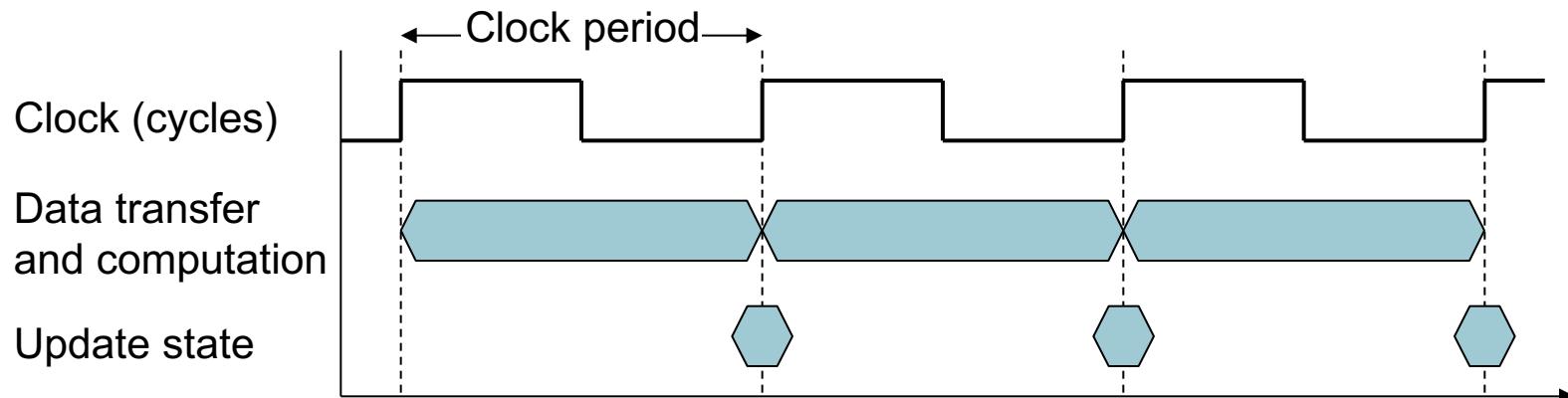
- Example: time taken to run a program
  - 10s on A, 15s on B
  - $\text{Execution Time}_B / \text{Execution Time}_A$   
 $= 15s / 10s = 1.5$
  - So A is 1.5 times faster than B

# 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
  - e.g.,  $250\text{ps} = 0.25\text{ns} = 250 \times 10^{-12}\text{s}$
- Clock frequency (rate): cycles per second
  - e.g.,  $4.0\text{GHz} = 4000\text{MHz} = 4.0 \times 10^9\text{Hz}$

# CPU Time

CPU Time = CPU Clock Cycles  $\times$  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

# Performance

---

- CPU time is the only metric that really matters!
- Other factors may change and **\*impact\*** CPU time

# CPU Time Example

- Computer A: 2GHz clock, 10s CPU time
- Designing Computer B
  - Aim for 6s CPU time
  - Can do faster clock, but causes  $1.2 \times$  clock cycles
- How fast must Computer B clock be?

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

$$\begin{aligned}\text{Clock Cycles}_A &= \text{CPU Time}_A \times \text{Clock Rate}_A \\ &= 10s \times 2\text{GHz} = 20 \times 10^9\end{aligned}$$

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

# Instruction Count and CPI

Clock Cycles = Instruction Count  $\times$  Cycles per Instruction

CPU Time = Instruction Count  $\times$  CPI  $\times$  Clock Cycle Time

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

- Instruction Count for a program
  - Determined by program, ISA and compiler
- Average cycles per instruction
  - Determined by CPU hardware
  - If different instructions have different CPI
    - Average CPI affected by instruction mix

# CPI Example

- Computer A: Cycle Time = 250ps, CPI = 2.0
- Computer B: Cycle Time = 500ps, CPI = 1.2
- Same ISA
- Which is faster, and by how much?

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

$$= I \times 2.0 \times 250\text{ps} = I \times 500\text{ps}$$

A is faster...

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

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

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

...by this much

# CPI in More Detail

- If different instruction classes take different numbers of cycles

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

- Weighted average CPI

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

  
Relative frequency

# CPI Example

- Alternative compiled code sequences using instructions in classes A, B, C

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

- Sequence 1: IC = 5
  - Clock Cycles  
 $= 2 \times 1 + 1 \times 2 + 2 \times 3$   
 $= 10$
  - Avg. CPI =  $10/5 = 2.0$
- Sequence 2: IC = 6
  - Clock Cycles  
 $= 4 \times 1 + 1 \times 2 + 1 \times 3$   
 $= 9$
  - Avg. CPI =  $9/6 = 1.5$

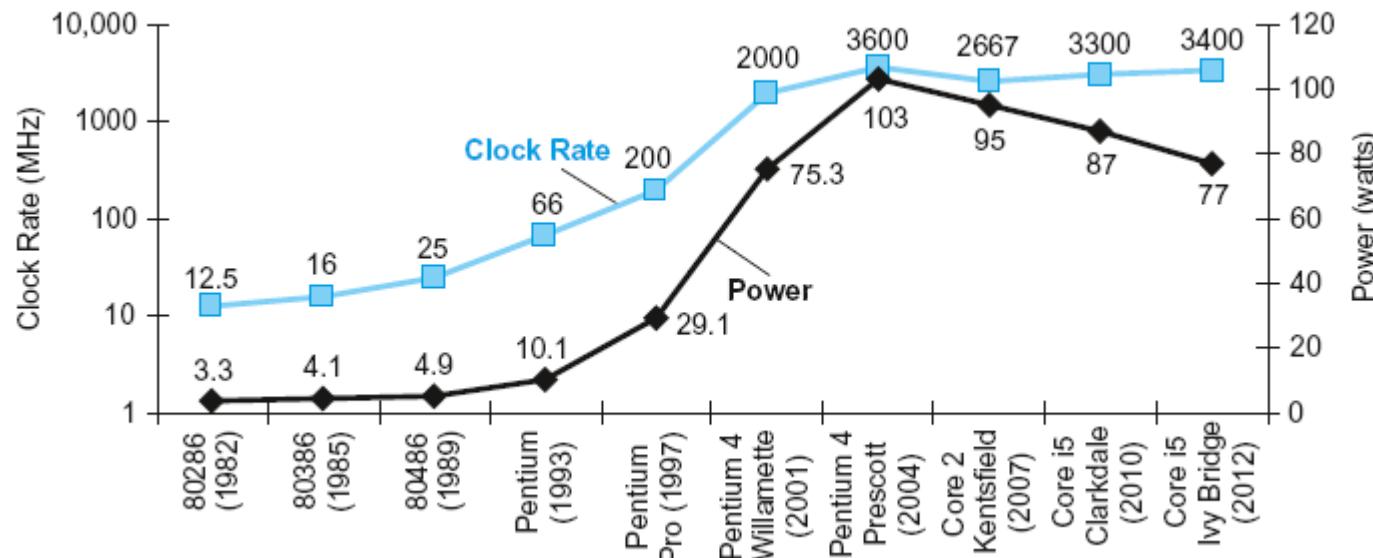
# Performance Summary

## The BIG Picture

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

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

# Power Trends



- In CMOS IC technology

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

↑  
×30

↑  
5V → 1V

↑  
×1000

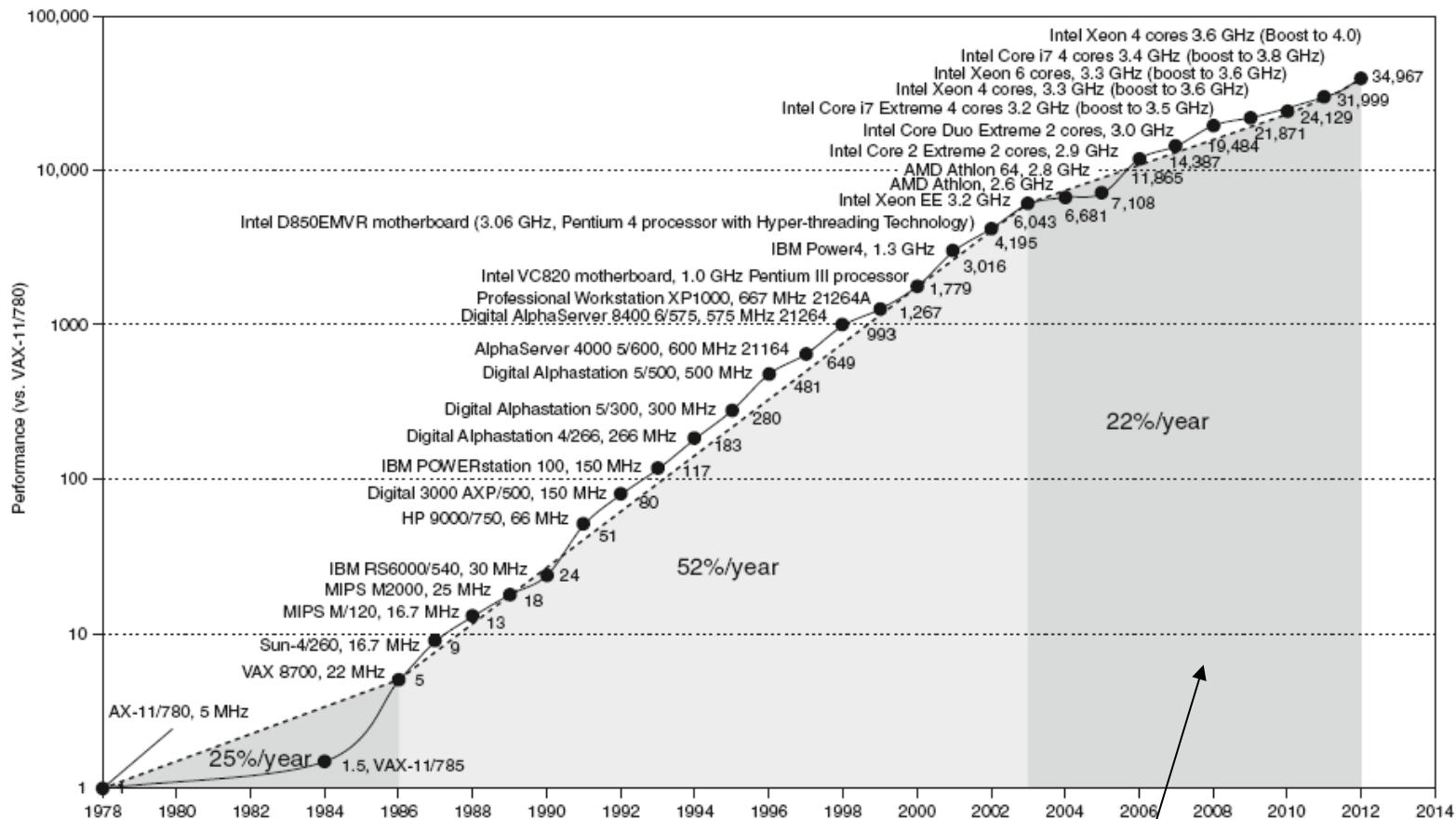
# Reducing Power

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

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

- The power wall
  - We can't reduce voltage further
  - We can't remove more heat
- How else can we improve performance?

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

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

# CINT2006 for Intel Core i7 920

Description	Name	Instruction Count x 10 <sup>9</sup>	CPI	Clock cycle time (seconds x 10 <sup>-9</sup> )	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

# SPEC Power Benchmark

- Power consumption of server at different workload levels
  - Performance:  $ssj\_ops/sec$
  - Power: Watts (Joules/sec)

$$\text{Overall } ssj\_ops \text{ per Watt} = \left( \sum_{i=0}^{10} ssj\_ops_i \right) / \left( \sum_{i=0}^{10} \text{power}_i \right)$$

# SPECpower\_ssj2008 for Xeon X5650

Target Load %	Performance (ssj_ops)	Average Power (Watts)
100%	865,618	258
90%	786,688	242
80%	698,051	224
70%	607,826	204
60%	521,391	185
50%	436,757	170
40%	345,919	157
30%	262,071	146
20%	176,061	135
10%	86,784	121
0%	0	80
Overall Sum	4,787,166	1,922
$\Sigma ssj\_ops / \Sigma power =$		2,490

# Pitfall: Amdahl's Law

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

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

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

$$20 = \frac{80}{n} + 20$$

- Can't be done!

- Corollary: make the common case fast

# Fallacy: Low Power at Idle

- Look back at i7 power benchmark
  - At 100% load: 258W
  - At 50% load: 170W (66%)
  - At 10% load: 121W (47%)
- Google data center
  - Mostly operates at 10% - 50% load
  - At 100% load less than 1% of the time
- Consider designing processors to make power proportional to load

# Pitfall: MIPS as a Performance Metric

- MIPS: Millions of Instructions Per Second
  - Doesn't account for
    - Differences in ISAs between computers
    - Differences in complexity between instructions

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

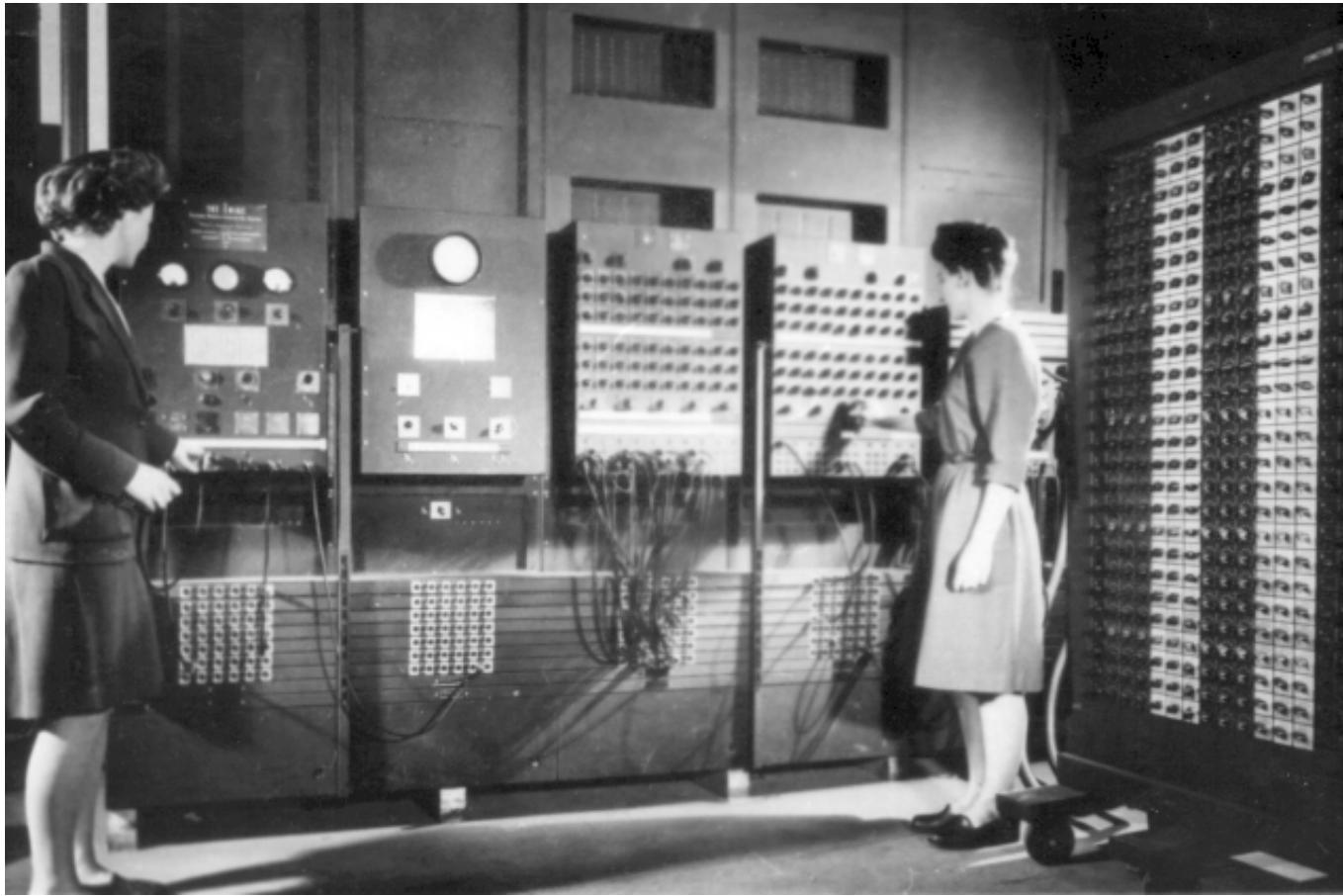
$$= \frac{\text{Instruction count}}{\text{Instruction count} \times \text{CPI}} \times 10^6 = \frac{\text{Clock rate}}{\text{CPI} \times 10^6}$$

- CPI varies between programs on a given CPU

# History: The Beginning...

- ENIAC - ~1940s, First Electronic Computer
  - Built @ UPenn by Eckert and Mauchly
  - ENIAC → Electronic Numerical Integrator and Calculator
  - HUGE! → 80 feet long, 8.5 feet high and a couple of feet wide.
  - Each of the 20, 10 bit registers where 2 ft long.
  - Had a total of 18,000 vacuum tubes
  - Used to compute artillery firing tables

# Pictures of ENIAC



Programmers/Early "Computer Geeks" Betty Jean Jennings (left) and Fran Bilas (right) operate the ENIAC's main control panel at the Moore School of Electrical Engineering. (U.S. Army photo from the archives of the ARL Technical Library)

# The von Neumann Computer

- 1944: John von Neumann proposes idea for a “stored program” computer - e.g., that a “program” can be stored in memory just like data...
- EDVAC - Electronic Discrete Variable Automatic Computer
- All modern day computer systems are built with this key concept!

# First microprocessor

- Intel 4004
- Co-invented by Dr. Ted Huff in 1971
  - Dr. Huff is an RPI alumnus! :-)
- Could add 2, 4 bit numbers
- Multiplication was done by repeated addition
- This led to first 8-bit microprocessor, Intel 8008 in 1974
  - First General CPU on a single chip!

# Concluding Remarks

- Cost/performance is improving
  - Due to underlying technology development
- Hierarchical layers of abstraction
  - In both hardware and software
- Instruction set architecture
  - The hardware/software interface
- Execution time: the best performance measure
- Power is a limiting factor
  - Use parallelism to improve performance