EIE/ENE 334 Microprocessors

Lecture 01:

Introduction to Digital Computer System

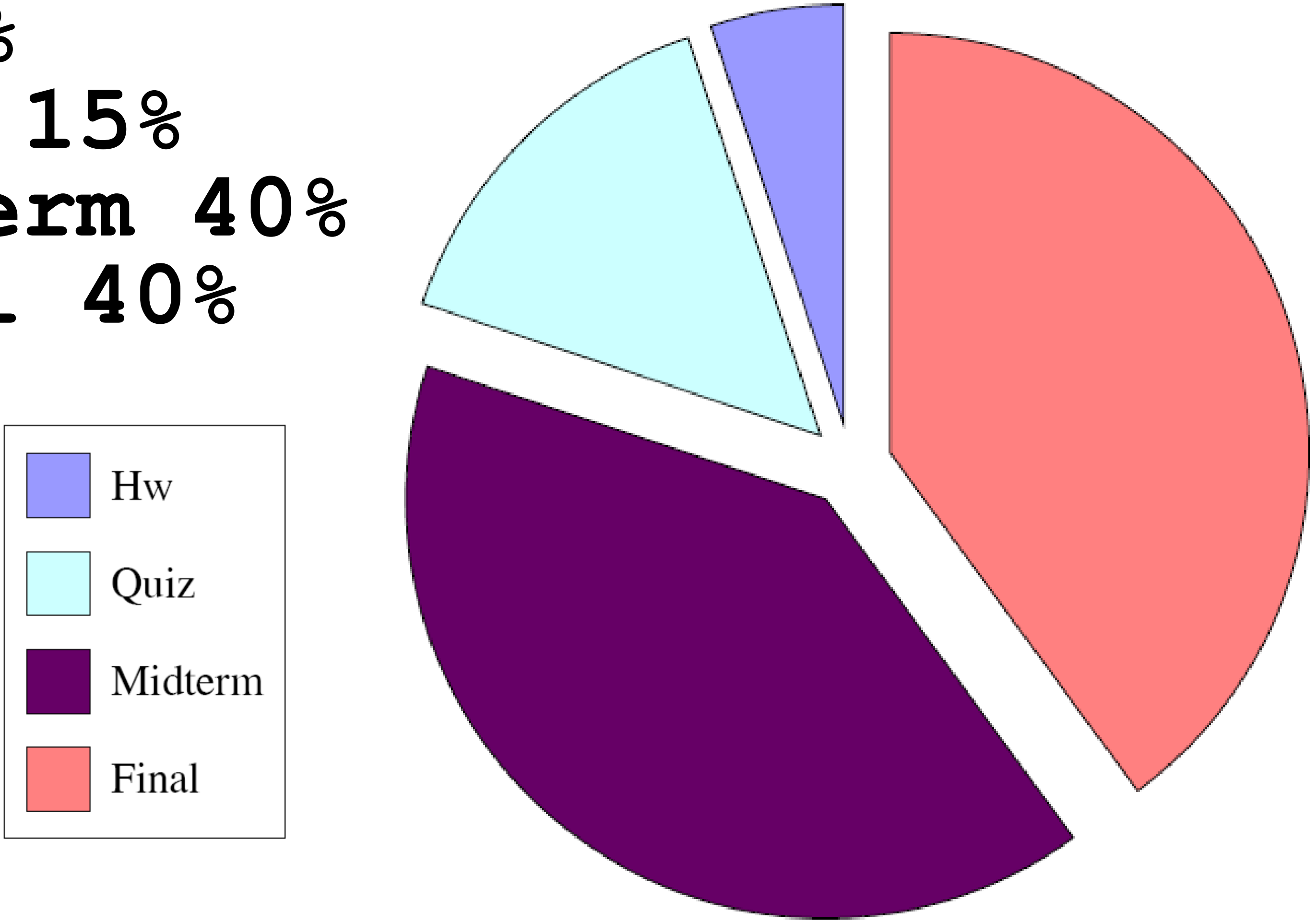
Week #01: Dejwoot KHAWPARISUTH

Adapted from *Computer Organization and Design, 4th Edition*, Patterson & Hennessy, © 2009, Elsevier (MK)

http://webstaff.kmutt.ac.th/~dejwoot.kha/

Evaluations:

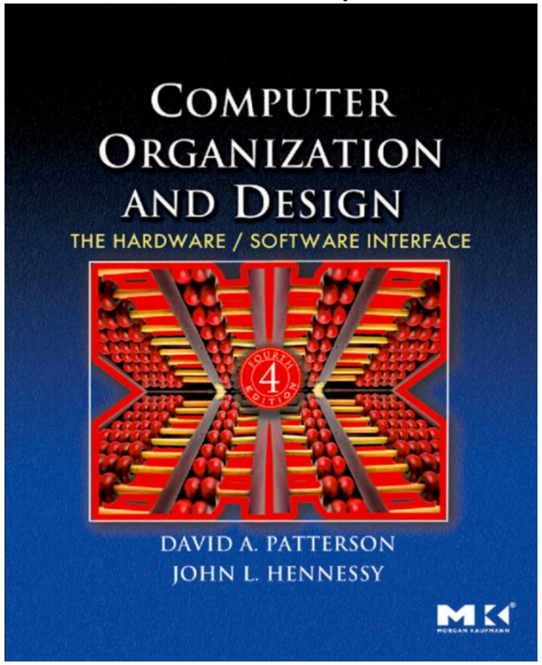
- Hw 5%
- Quiz 15%
- Midterm 40%
- Final 40%



Note: 14 weeks of classes (only 2 absent weeks allowed, or 'Fa')

Textbooks:

- David A. Patterson, John L. Hennessy, "Computer Organization and Design: The Hardware/Software Interface", Morgan Kaufmann Publishers, 4th edition.
  - Chapter 1-4
  - Appendix B,E


COMPUTER ORGANIZATION AND DESIGN
THE HARDWARE / SOFTWARE INTERFACE
4 FOURTH EDITION
DAVID A. PATTERSON
JOHN L. HENNESSY
MK

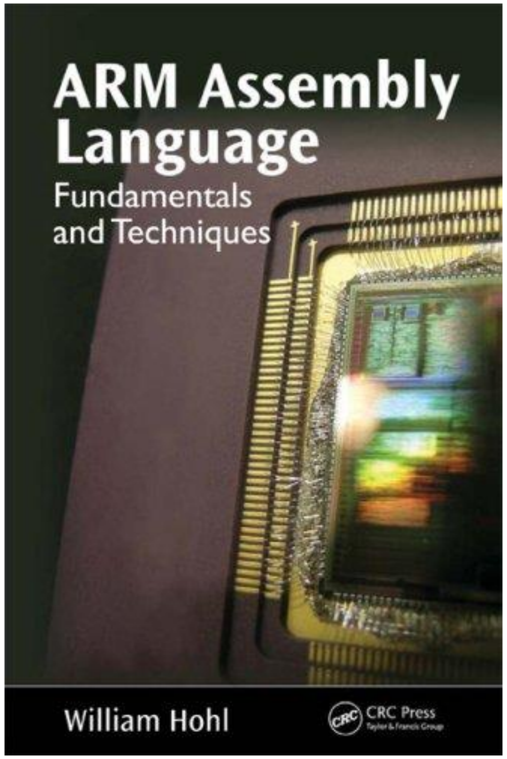
http://www.elsevierdirect.com/v2/companion.jsp?ISBN=9780123747501
https://opac.lib.kmutt.ac.th/vufind/Record/1228134

Textbooks:

- **William Hohl, "ARM Assembly Language: Fundamentals and Techniques", CRC, 2009**

ARM7TDMI

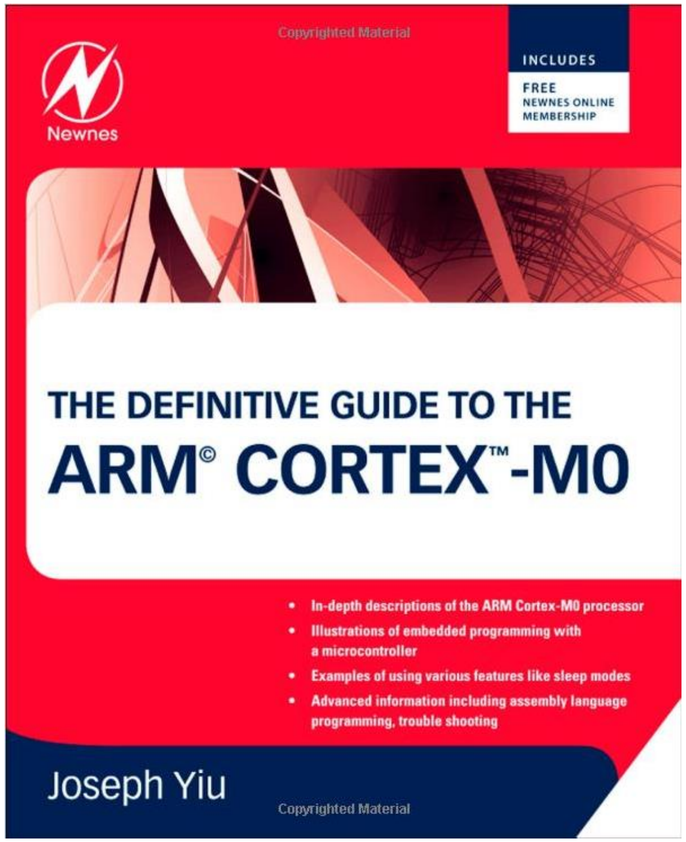
(ARM Version 4T)



Call Number: QA76.73.A8 HOL 2009

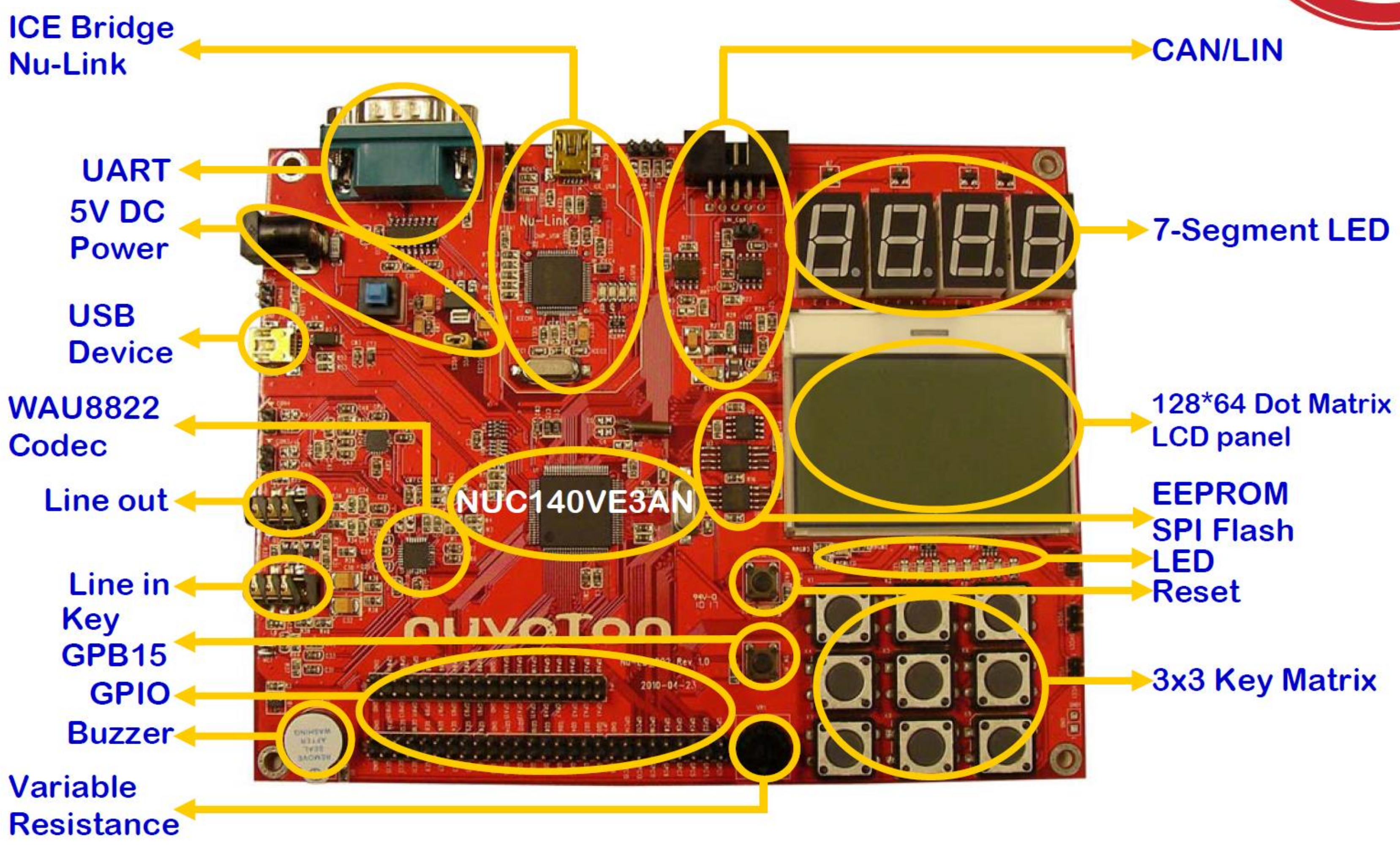
Textbooks:

- Joseph Yiu, "**The Definitive Guide to the ARM Cortex-M0**", Newnes, 2011


Copyrighted Material
Newnes
INCLUDES
FREE NEWNES ONLINE MEMBERSHIP
THE DEFINITIVE GUIDE TO THE
ARM© CORTEX™-M0
In-depth descriptions of the ARM Cortex-M0 processor
Illustrations of embedded programming with a microcontroller
Examples of using various features like sleep modes
Advanced information including assembly language programming, trouble shooting
Joseph Yiu
Copyrighted Material

http://www.arm.com/support/university/academic-resources.php

EIE/ENE 335 : Lab T2/56



Objectives:

- Digital Computer System
- To understand both hardware and software
- Hardware/software interface
- MIPS Architecture
- ARM Cortex-M0
- Subroutines

Chapter 1

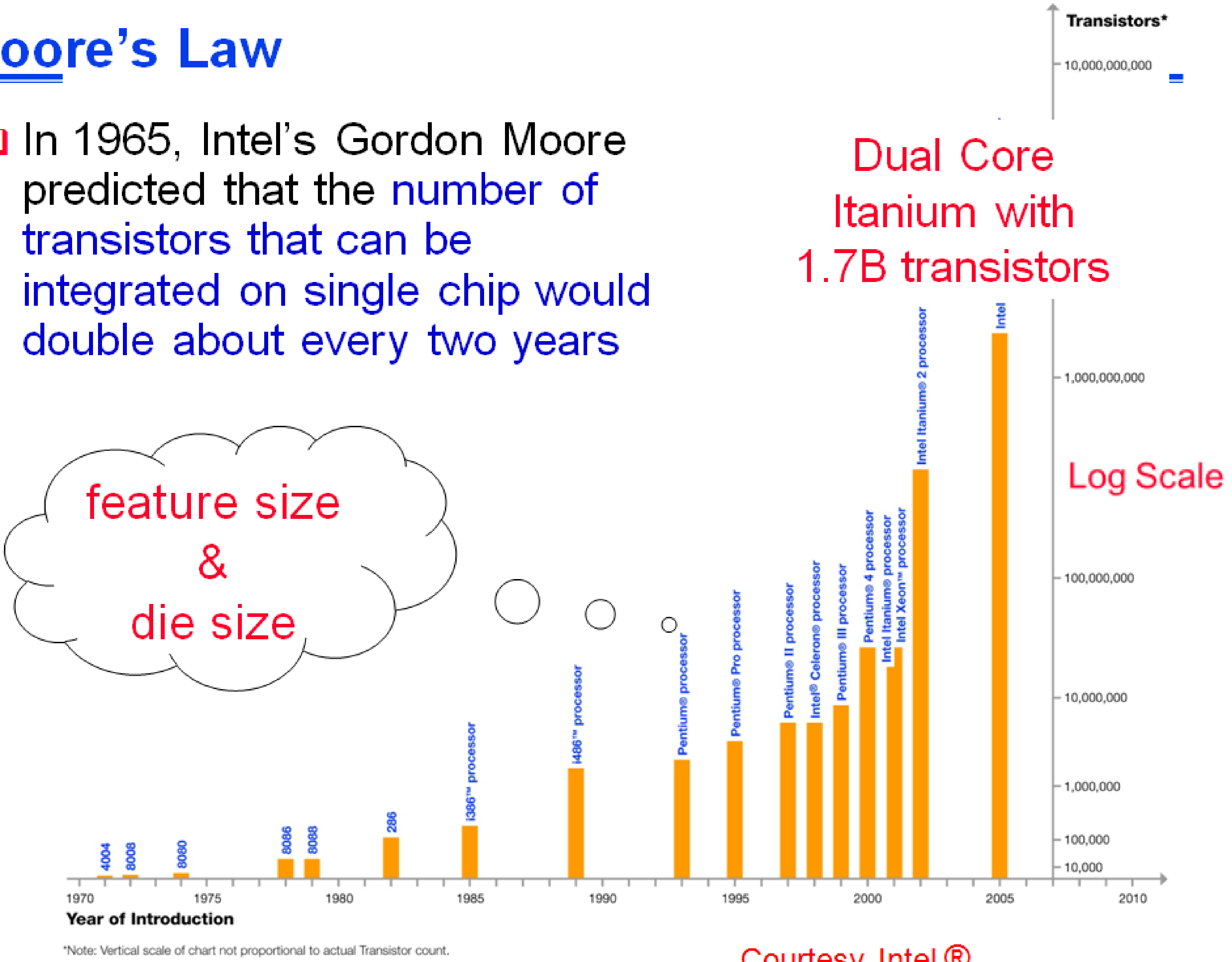
Computer Abstractions and Technology

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

Moore's Law

- In 1965, Intel's Gordon Moore predicted that the number of transistors that can be integrated on single chip would double about every two years



Classes of Computers

- Desktop 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
- Embedded computers
  - Hidden as components of systems
  - Stringent power/performance/cost constraints

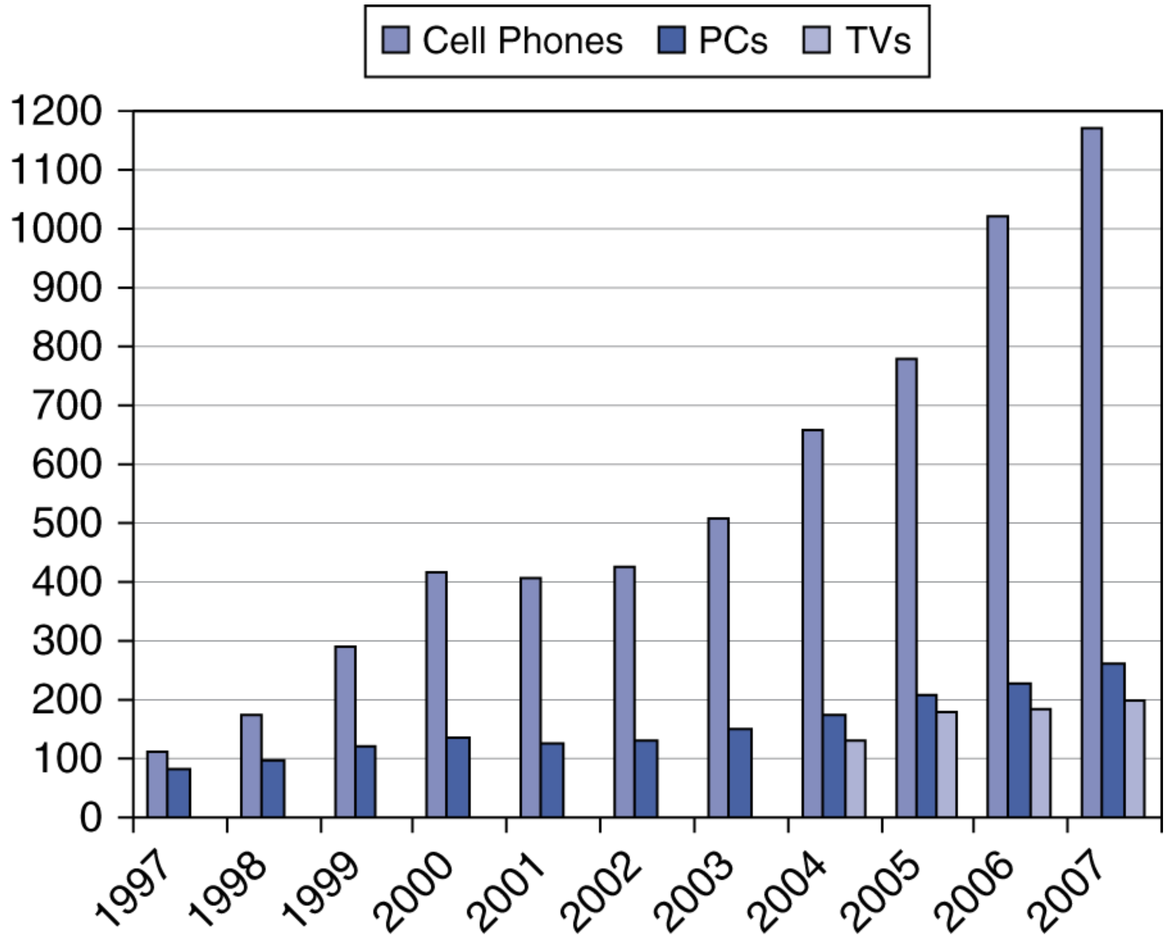
Classes of Computers:

- Desktop computers
  - Designed to deliver good performance to a single user at low cost usually executing 3rd party software, usually incorporating a graphics display, a keyboard, and a mouse
- Servers
  - Used to run larger programs for multiple, simultaneous users typically accessed only via a network and that places a greater emphasis on dependability and (often) security
- Supercomputers
  - A high performance, high cost class of servers with hundreds to thousands of processors, terabytes of memory and petabytes of storage that are used for high-end scientific and engineering applications
- Embedded computers (processors)
  - A computer inside another device used for running one predetermined application

Some Basic Definitions:

- Kilobyte – 2^{10} or 1,024 bytes
- Megabyte– 2^{20} or 1,048,576 bytes
  - sometimes “rounded” to 10^{6} or 1,000,000 bytes
- Gigabyte – 2^{30} or 1,073,741,824 bytes
  - sometimes rounded to 10^{9} or 1,000,000,000 bytes
- Terabyte – 2^{40} or 1,099,511,627,776 bytes
  - sometimes rounded to 10^{12} or 1,000,000,000,000 bytes
- Petabyte – 2^{50} or 1024 terabytes
  - sometimes rounded to 10^{15} or 1,000,000,000,000,000 bytes
- Exabyte – 2^{60} or 1024 petabytes
  - Sometimes rounded to 10^{18} or 1,000,000,000,000,000,000 bytes

The Processor Market



Embedded Processor Characteristics

The largest class of computers spanning the widest range of applications and performance

- Often have minimum performance requirements. Example?
- Often have stringent limitations on cost. Example?
- Often have stringent limitations on power consumption. Example?
- Often have low tolerance for failure. Example?

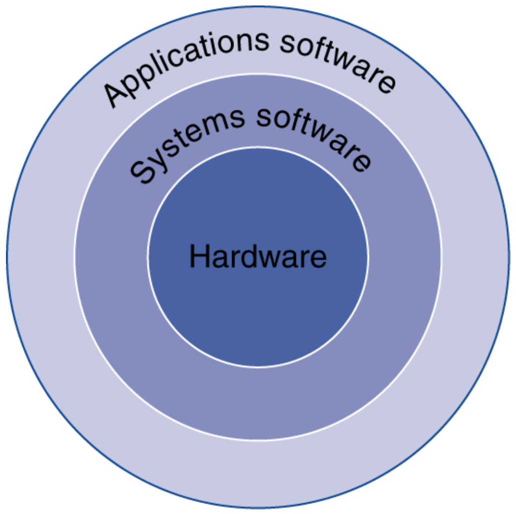
What You Will Learn

- 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

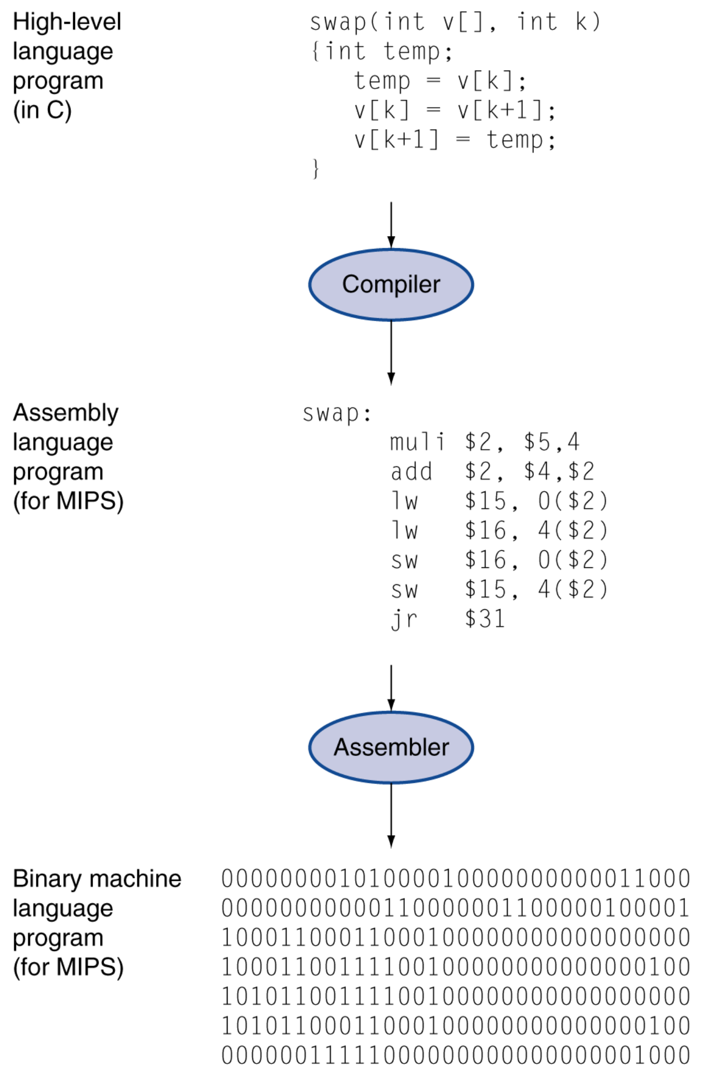
Below Your Program



- 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)
Compiler
Assembly language program (for MIPS)
Assembler
Binary machine language program (for MIPS)

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

```
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
```

```
00000000101000010000000000011000
00000000000110000001100000100001
10001100011000100000000000000000
10001100111100100000000000000100
10101100111100100000000000000000
10101100011000100000000000000100
00000011111000000000000000001000
```

Level of Program Code:

- High-level language program (in C)

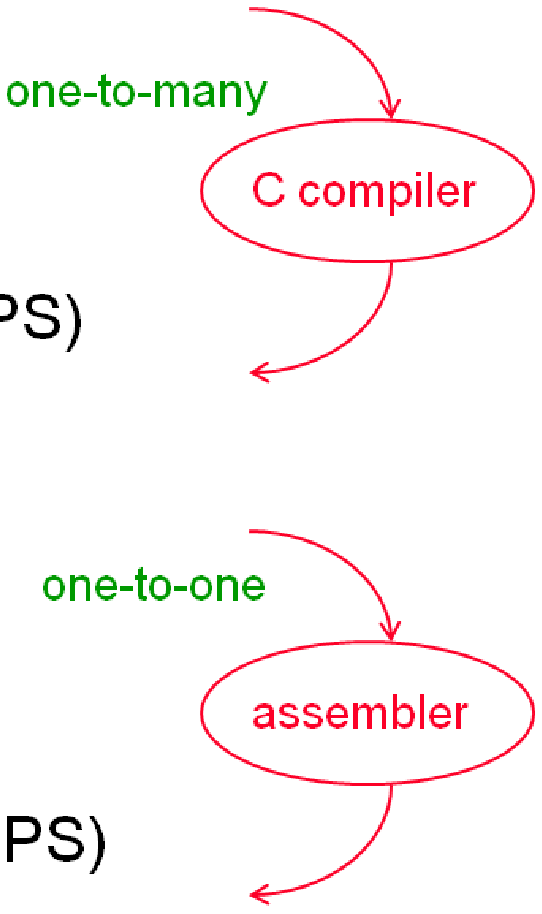
```
swap (int v[], int k)
(int temp;
        temp = v[k];
        v[k] = v[k+1];
        v[k+1] = temp;
)
```

- Assembly language program (for MIPS)

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

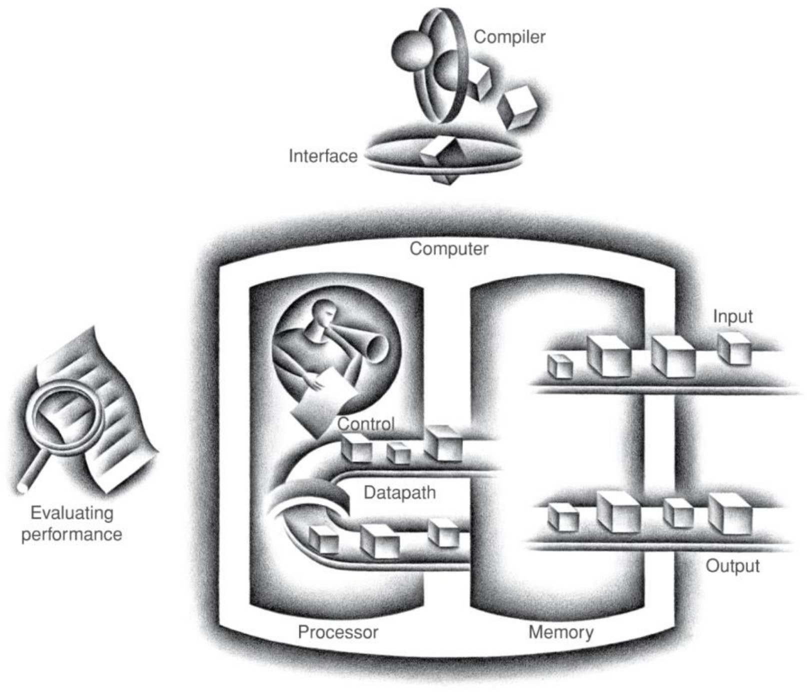
- Machine (object, binary) code (for MIPS)

```
000000 00000 00101 0001000010000000
000000 00100 00010 0001000000100000
  . . .
```



Components of a Computer

The BIG Picture

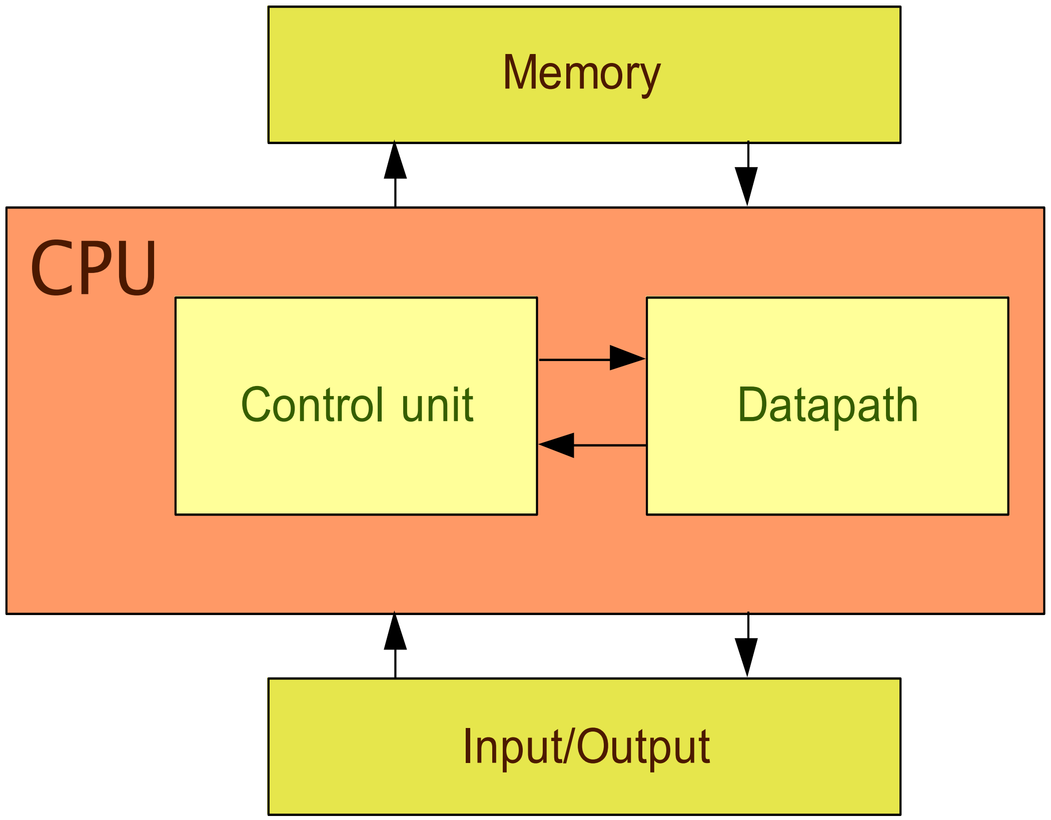


- Same components for all kinds of computer
  - Desktop, server, embedded
- Input/output includes
  - User-interface devices
    - Display, keyboard, mouse
  - Storage devices
    - Hard disk, CD/DVD, flash
  - Network adapters
    - For communicating with other computers

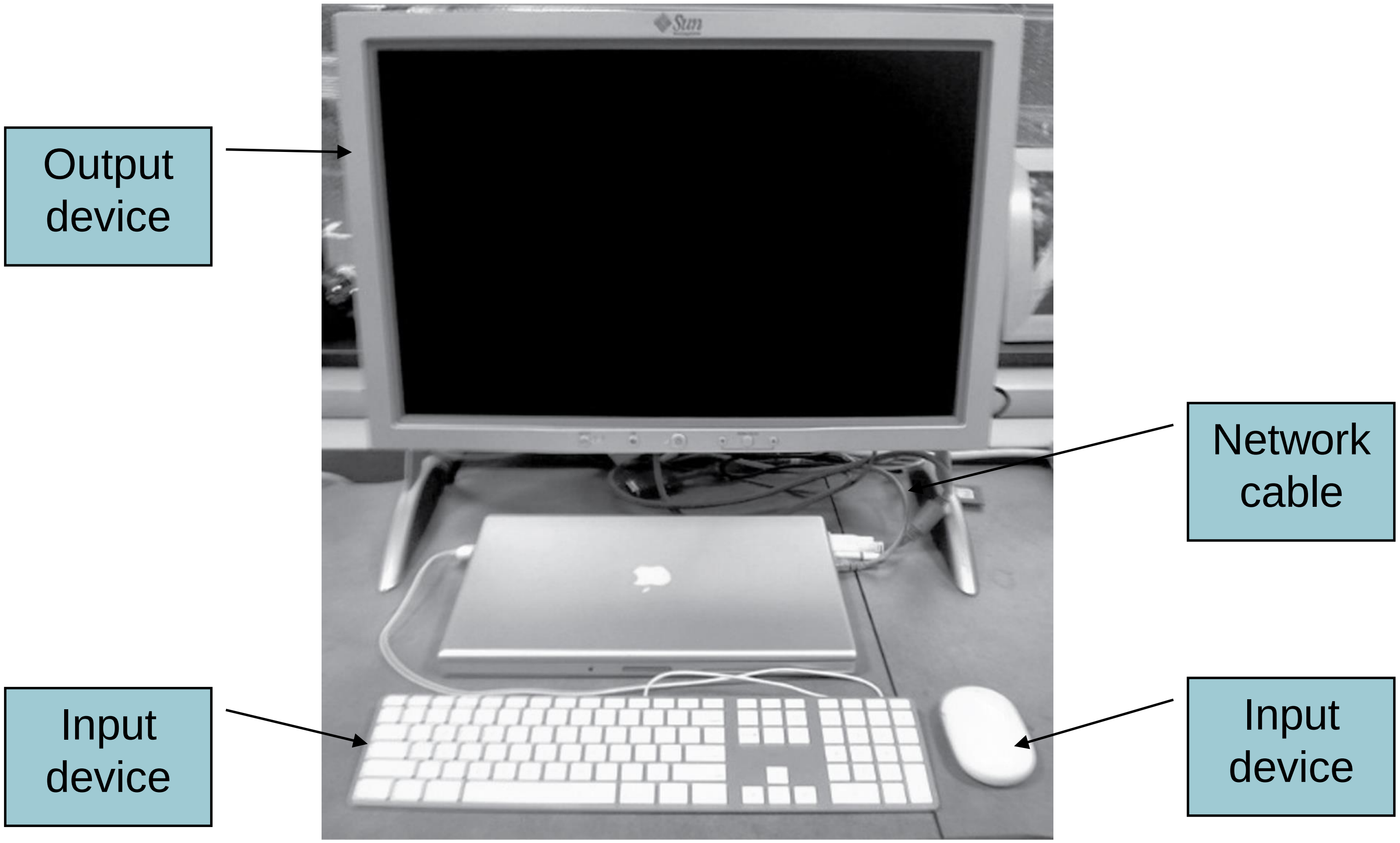
Advantages of Higher-Level Languages ?

- Higher-level languages
  - Allow the programmer to think in a more natural language and for their intended use (Fortran for scientific computation, Cobol for business programming, Lisp for symbol manipulation, Java for web programming, …)
  - Improve programmer productivity – more understandable code that is easier to debug and validate
  - Improve program maintainability
  - Allow programs to be independent of the computer on which they are developed (compilers and assemblers can translate high-level language programs to the binary instructions of any machine)
  - Emergence of optimizing compilers that produce very efficient assembly code optimized for the target machine
- As a result, very little programming is done today at the assembler level

The five classic components:

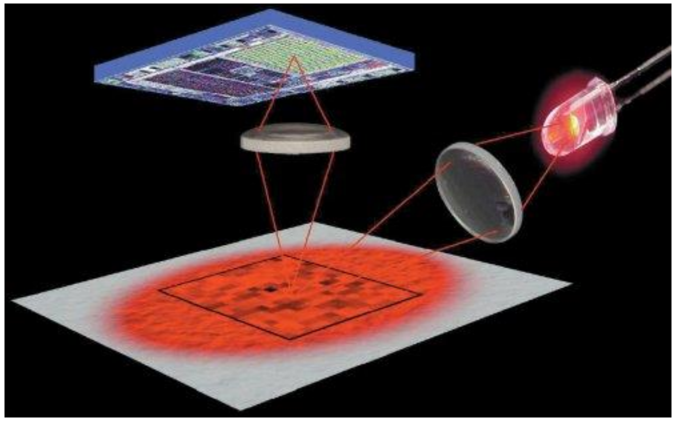


Anatomy of a Computer



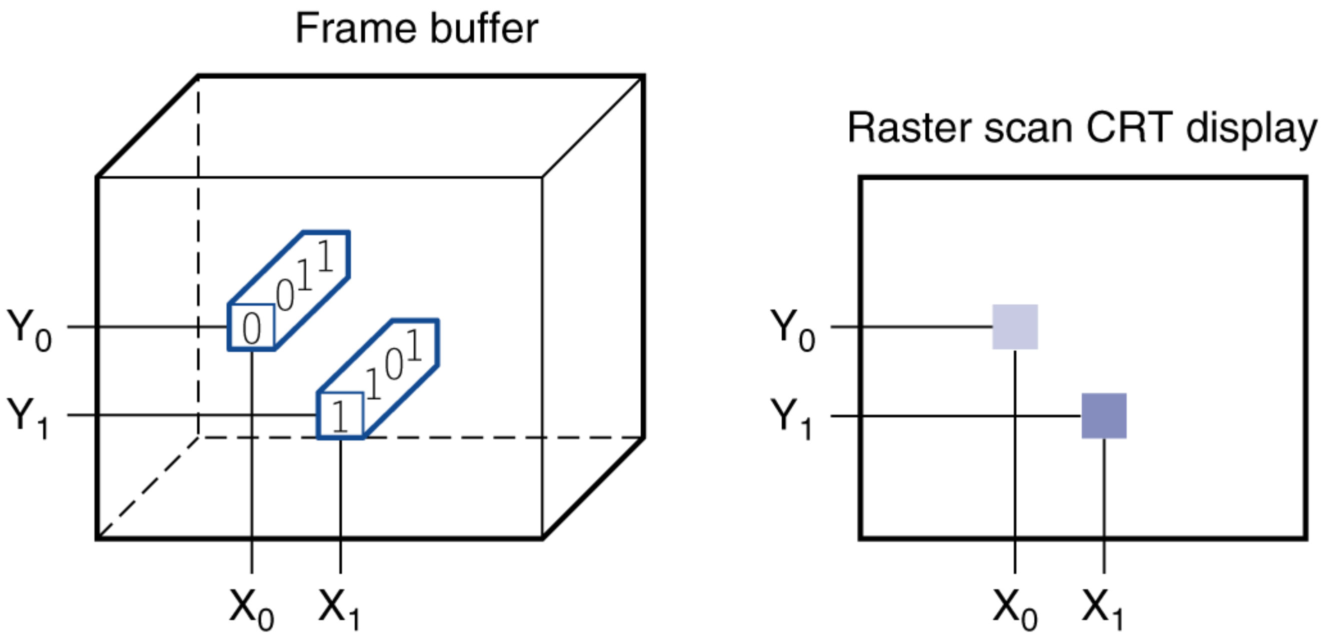
Anatomy of a Mouse

- Optical mouse
  - LED illuminates desktop
  - Small low-res camera
  - Basic image processor
    - Looks for x, y movement
  - Buttons & wheel
- Supersedes roller-ball mechanical mouse

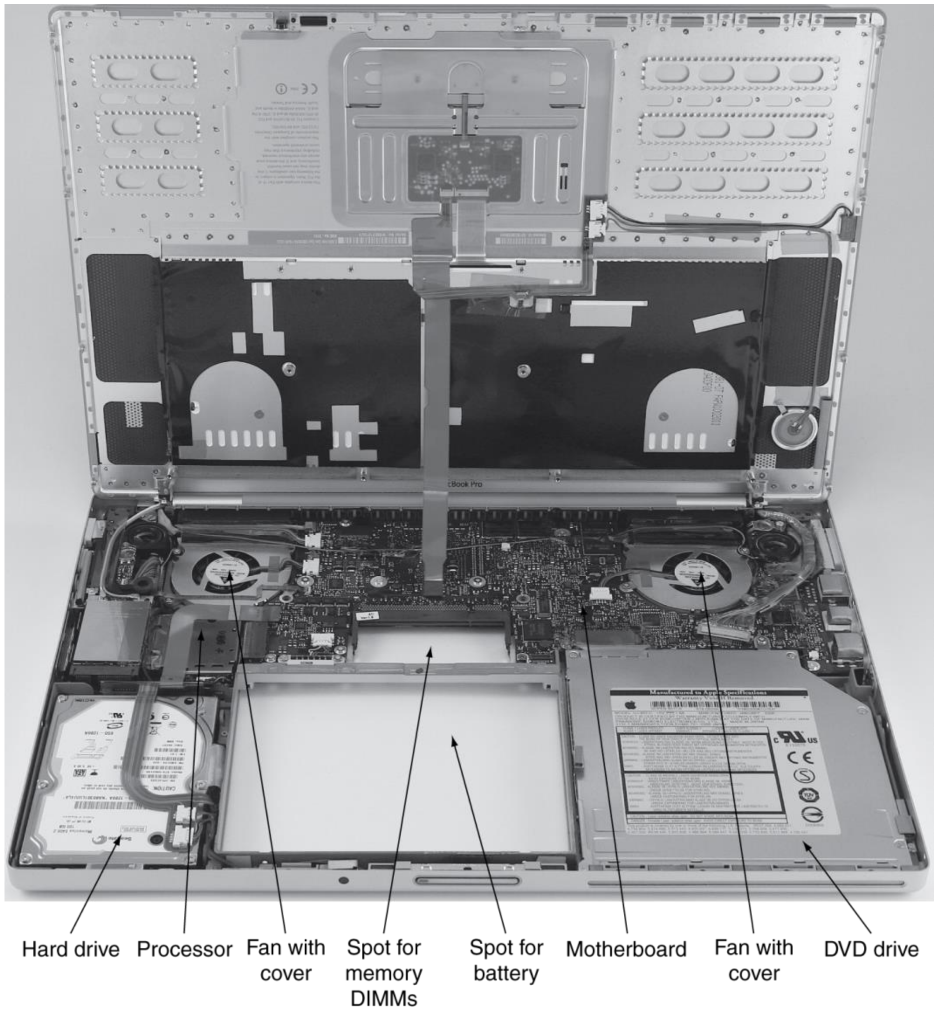


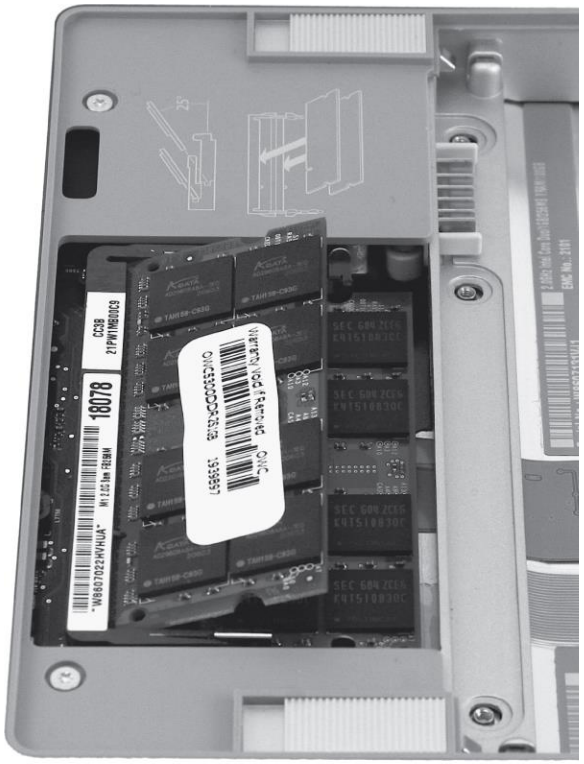
Through the Looking Glass

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



Opening the Box



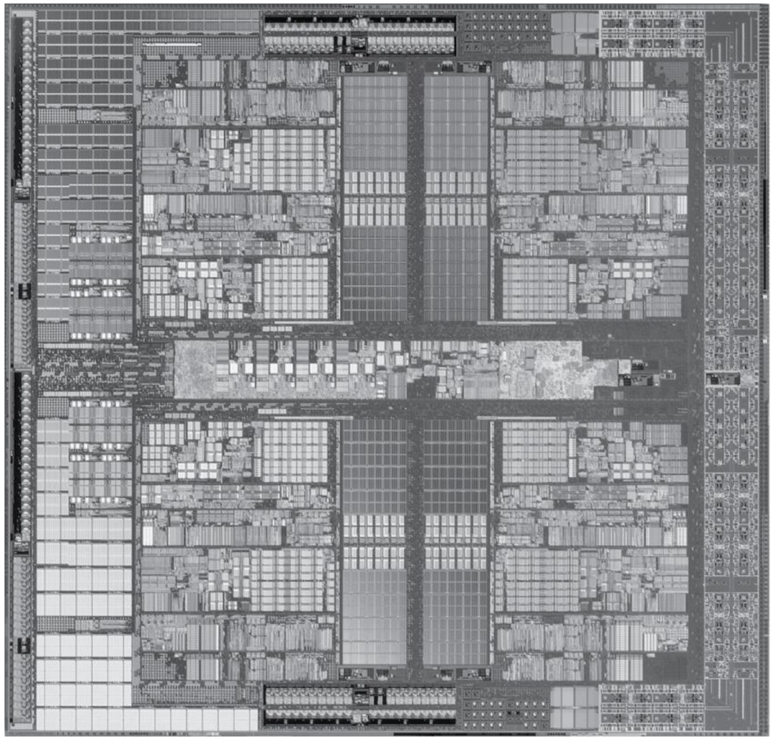


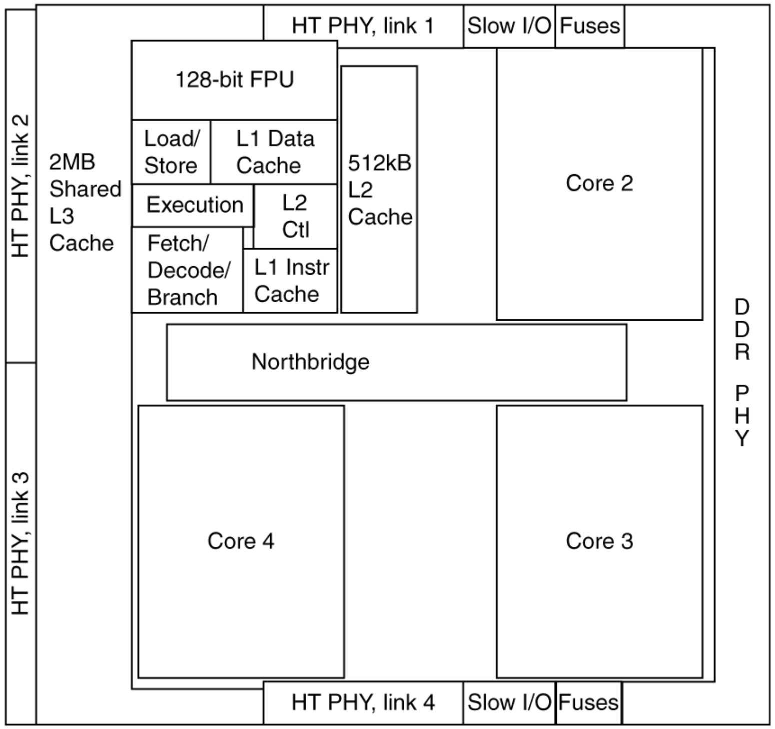
Inside the Processor (CPU)

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

Inside the Processor

- AMD Barcelona: 4 processor cores





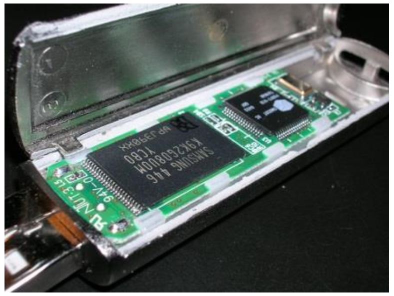
Abstractions

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

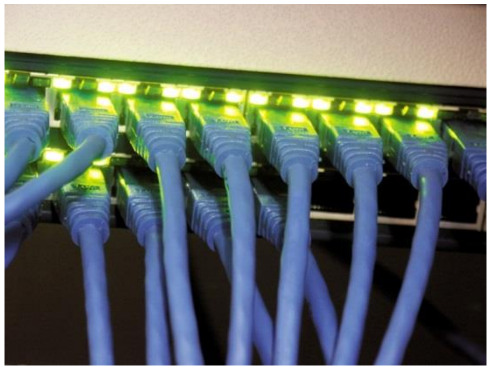
A Safe Place for Data

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



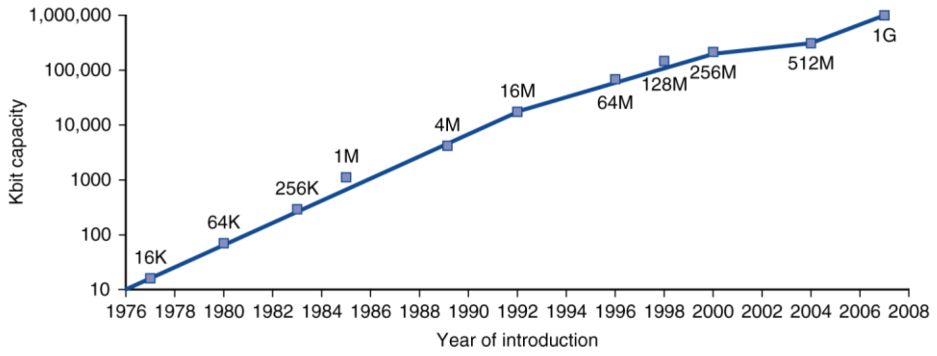
Networks

- Communication and resource sharing
- Local area network (LAN): Ethernet
  - Within a building
- Wide area network (WAN: the Internet
- Wireless network: WiFi, Bluetooth



Technology Trends

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

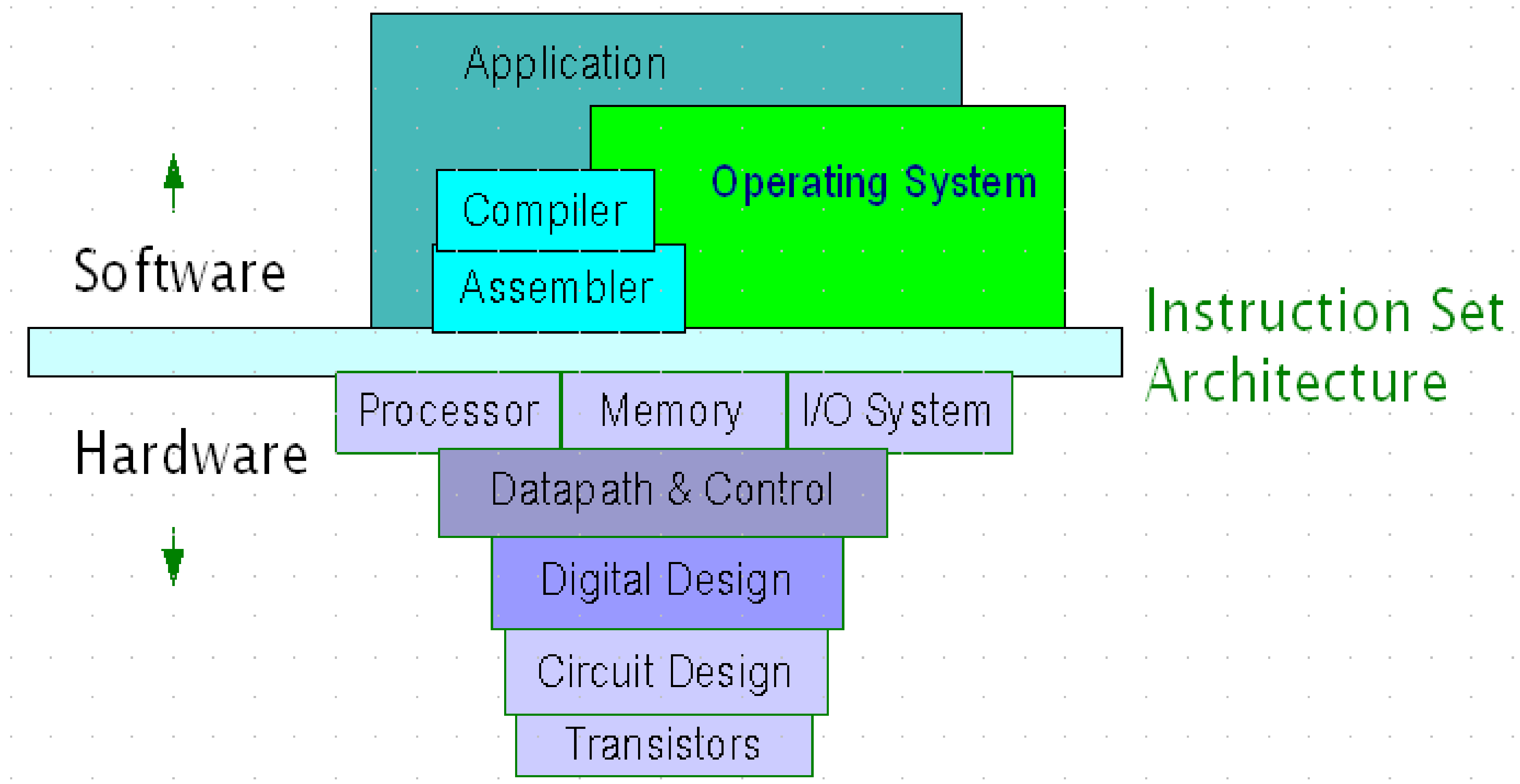


DRAM capacity

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
2005	Ultra large scale IC	6,200,000,000

Hardware and Software:

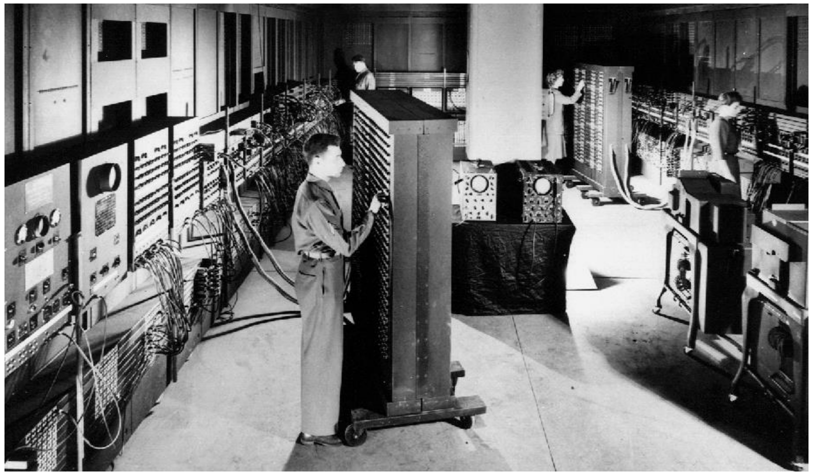
Coordinate of many levels(layers) of abstraction



Historical Perspective: 1/4

➔ **During World War II:**

ENIAC (Electronic Numerical Integrator and Calculator): the world's first operational electronic, general-purpose computer by J. Presper Eckert and John Mauchly at the Moore School of the University of Pennsylvania. (limited primarily by a small amount of storage and tedious programming.)



Historical Perspective: 2/4

- ➔ **In 1944:**

EDVAC (Electronic Discrete Variable Automatic Computer): a stored-program computer

- ➔ **In 1951:**

UNIVAC I (Universal Automatic Computer), designed to be sold as a general-purpose computer

- ➔ **In 1963:**

CDC 6600, the first supercomputer: the Control Data Corporation

Historical Perspective: 3/4

- ➔ **In 1964:**

  the System/360 by IBM

- ➔ **In 1971:**

  the Intel 4004 , Intel inventing the first microprocessor

- ➔ **In 1977:**

  the Apple II

Historical Perspective: 4/4

Year	Name	Size (cu. ft.)	Power (watts)	Performance (adds/sec)	Memory (KB)	Adjusted price (2003 $)
1951	UNIVAC I	1,000	125,000	2,000	48	$6,107,600
1964	IBM S/360 model 50	60	10,000	500,000	64	$4,792,300
1965	PDP-8	8	500	330,000	4	$75,390
1976	Cray-1	58	60,000	166,000,000	32,000	$10,756,800
1981	IBM PC	1	150	240,000	256	$5,461
1991	HP 9000/ model 750	2	500	50,000,000	16,384	$9,401
1996	Intel PPro PC (200 MHz)	2	500	400,000,000	16,384	$4,945
2003	Intel Pentium 4 PC (3.0 GHz)	2	500	6,000,000,000	262,144	$1,600

Instruction set architraves: 1/7

> Accumulator Architectures:
the earliest stored-program computers had a single register for arithmetic instructions. it was called the accumulator

```
      add 200
```

The next step in the evolution of instruction sets was the addition of registers dedicated to specific operations.(Ex. Intel 8086)

> Extended Accumulator, dedicated register, or special-purpose register

Instruction set architraves: 2/7

> General-Purpose Register Architectures:
(Ex. MIPS)
a register-memory architecture (Ex. IBM 360)
a load-store or a register-register machine
memory-memory

> Compact Code and Stack Architectures:

Instruction set architraves: 3/7

To keep programs small, so machines like the Intel IA-32, IBM 360, and VAX had variable-length instructions, both to match the varying operand specifications and to minimize code size. Intel IA-32 instructions are from 1 to 17 bytes long; IBM 360 instructions are 2, 4, or 6 bytes long; and VAX instruction lengths are anywhere from 1 to 54 bytes.

In embedded applications:
compact instruction encoding is desirable (for Java bytecode.)

Instruction set architraves: 4/7

> High-Level-Language Computer Architectures:
More efficient programming languages and compilers, plus expanding memory, doomed this movement to a historical footnote. The Burroughs B5000 was the commercial fountainhead of this philosophy, but today there is no significant commercial descendent of this 1960s radical.

Instruction set architraves: 5/7

> Reduced Instruction Set Computer Architectures:
Improvements in programming languages, compiler technology, and memory cost meant that less programming was being done at the assembly level, so instruction sets could be measured by how well compilers used them as opposed to how well assembly language programmers used them.

Instruction set architraves: 6/7

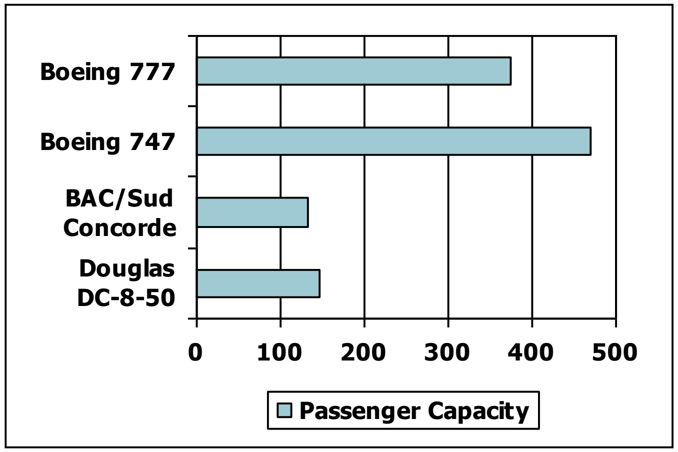
Virtually all new instruction sets since 1982 have followed this RISC philosophy of fixed instruction lengths, load-store instruction sets, limited addressing modes, and limited operations. ARM, Hitachi SH, IBM PowerPC, MIPS, and Sun SPARC are all examples of RISC architectures.

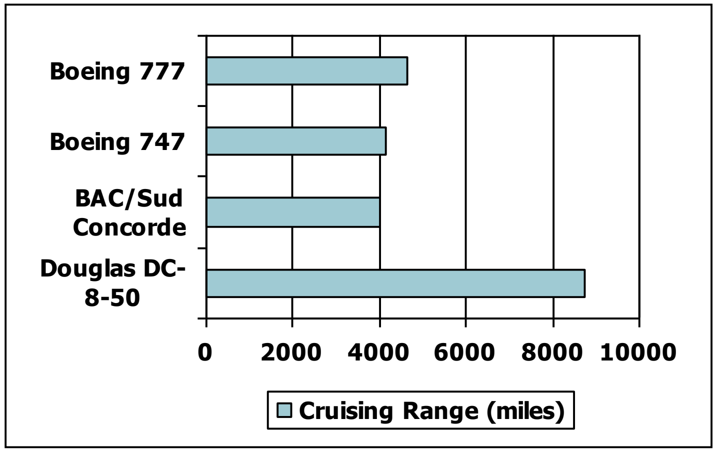
Instruction set architraves: 7/7

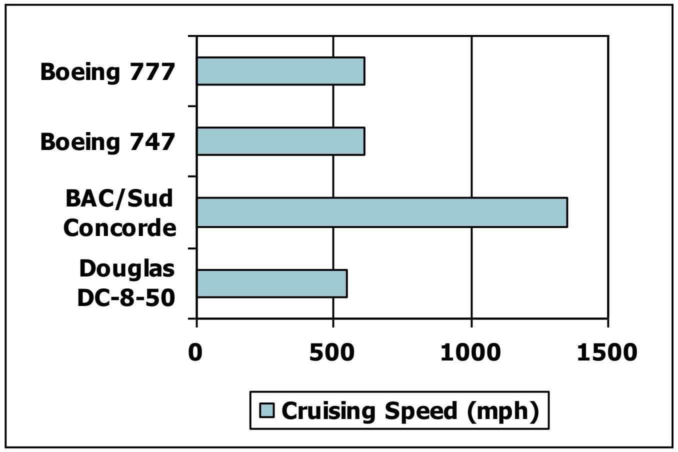
Machine	Number of general-purpose registers	Architectural style	Year
EDSAC	1	accumulator	1949
IBM 701	1	accumulator	1953
CDC 6600	8	load-store	1963
IBM 360	16	register-memory	1964
DEC PDP-8	1	accumulator	1965
DEC PDP-11	8	register-memory	1970
Intel 8008	1	accumulator	1972
Motorola 6800	2	accumulator	1974
DEC VAX	16	register-memory, memory-memory	1977
Intel 8086	1	extended accumulator	1978
Motorola 68000	16	register-memory	1980
Intel 80386	8	register-memory	1985
MIPS	32	load-store	1985
HP PA-RISC	32	load-store	1986
SPARC	32	load-store	1987
PowerPC	32	load-store	1992
DEC Alpha	32	load-store	1992
HP/Intel IA-64	128	load-store	2001
AMD64 (EMT64)	16	register-memory	2003

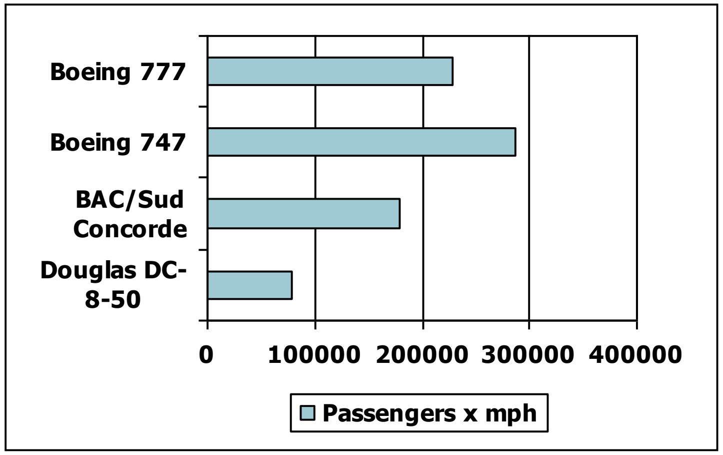
Defining Performance

- 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"

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

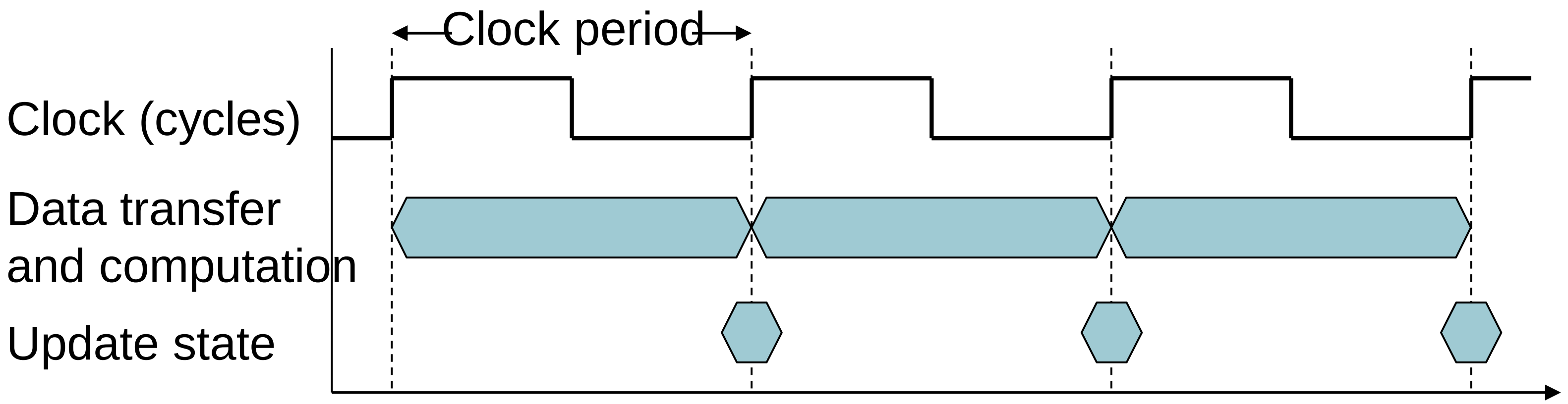
- Example: time taken to run a program
  - 10s on A, 15s on B
  - Execution Time$_B$ / Execution Time$_A$ = 15s / 10s = 1.5
  - So A is 1.5 times faster than B

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

CPU Time

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

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

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

CPU 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 × clock cycles
- How fast must Computer B clock be?

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

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

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

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

Instruction Count and CPI

$$\text{Clock Cycles} = \text{Instruction Count} \times \text{Cycles per Instruction}$$

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

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

- 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?

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

$$= I \times 2.0 \times 250\text{ps} = I \times 500\text{ps} \longleftarrow \boxed{\text{A is faster…}}$$

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

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

$$\frac{\text{CPU Time}_B}{\text{CPU Time}_A} = \frac{I \times 600\text{ps}}{I \times 500\text{ps}} = 1.2 \longleftarrow \boxed{\text{…by this much}}$$

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

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

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

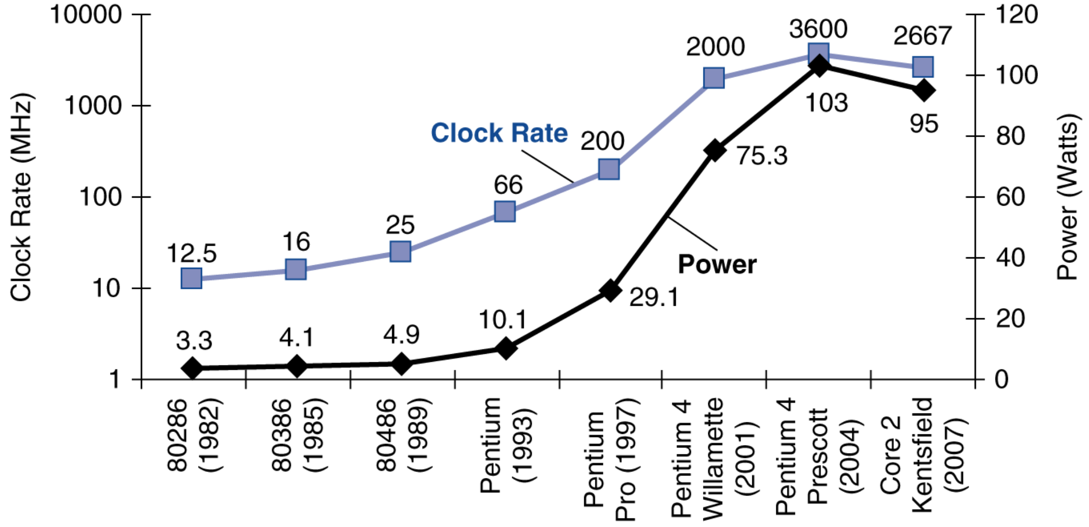
Performance Summary

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

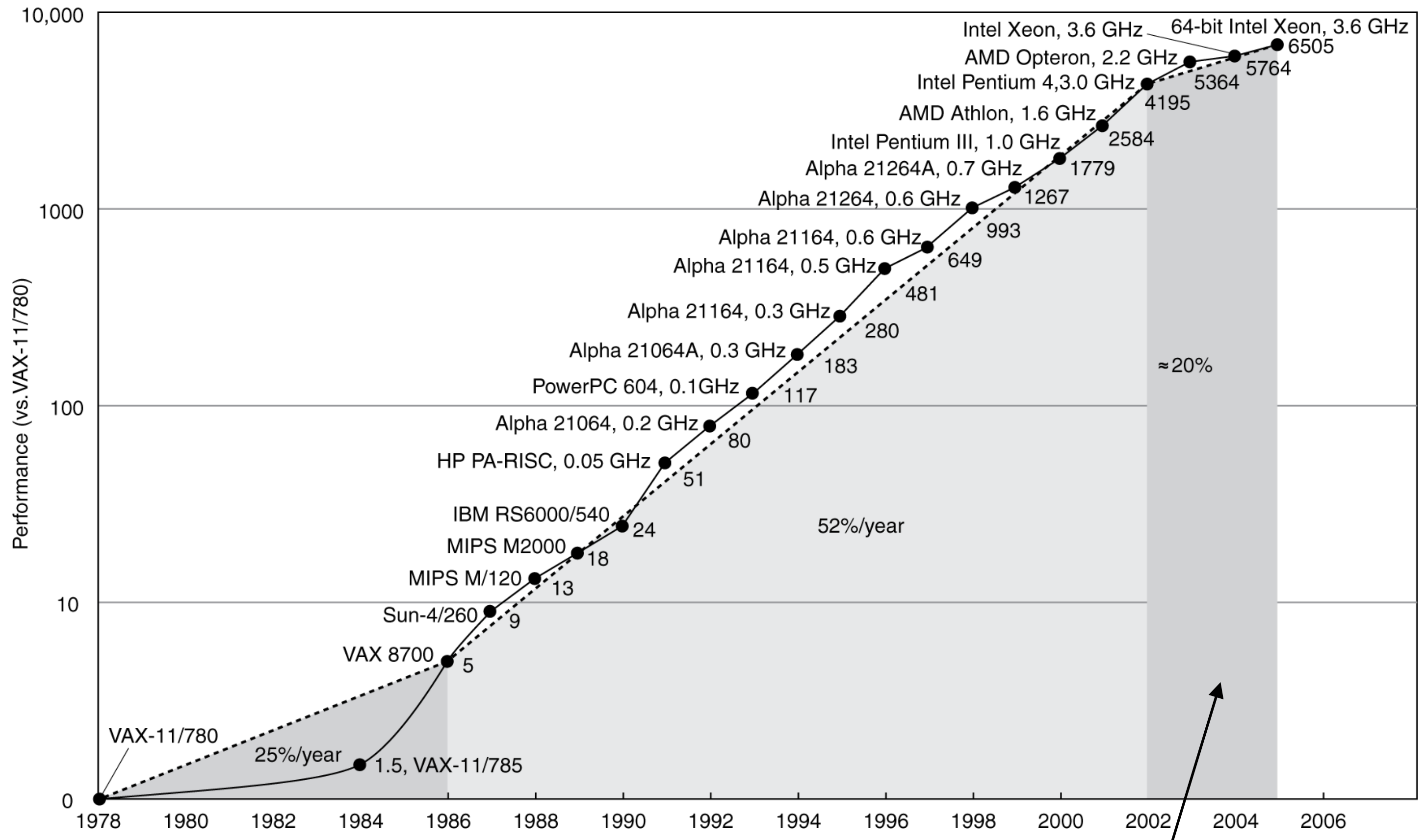
Reducing Power

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

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

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

Uniprocessor Performance

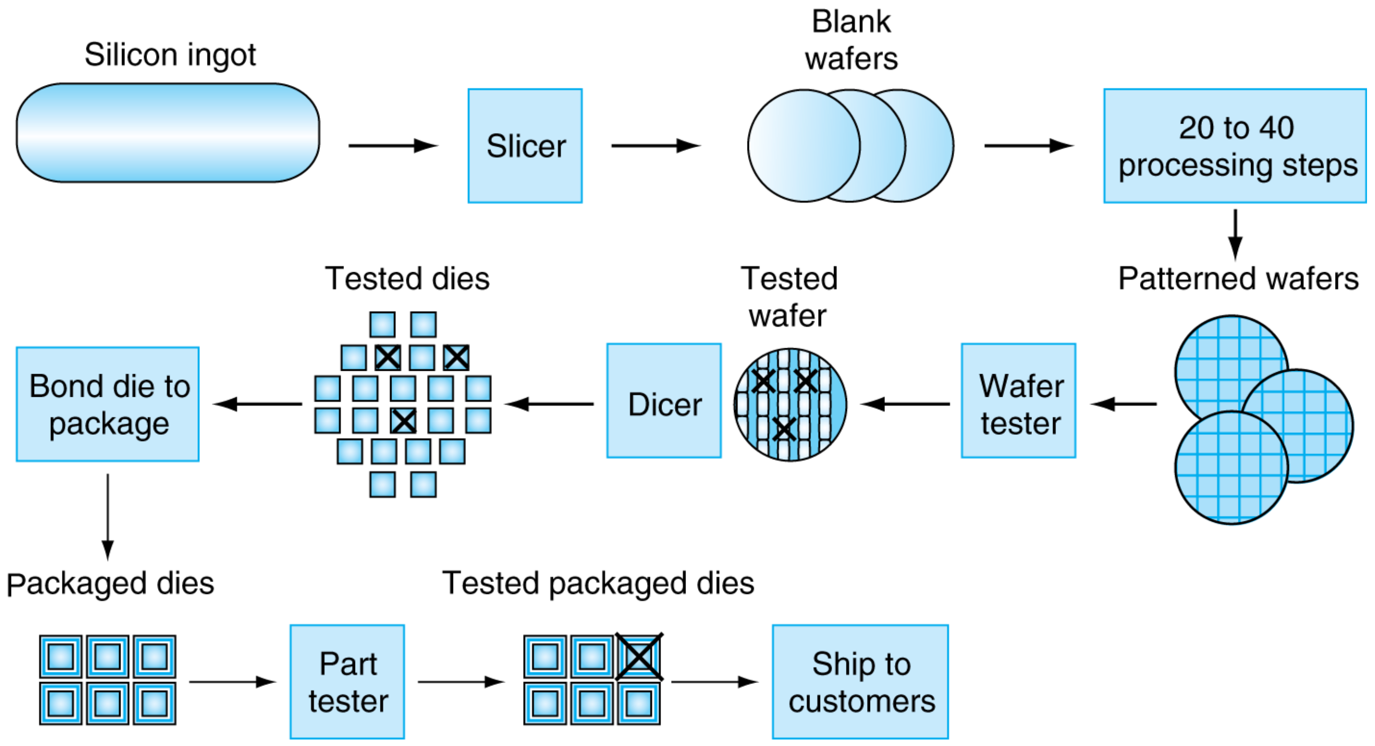


Constrained by power, instruction-level parallelism, memory latency

Multiprocessors

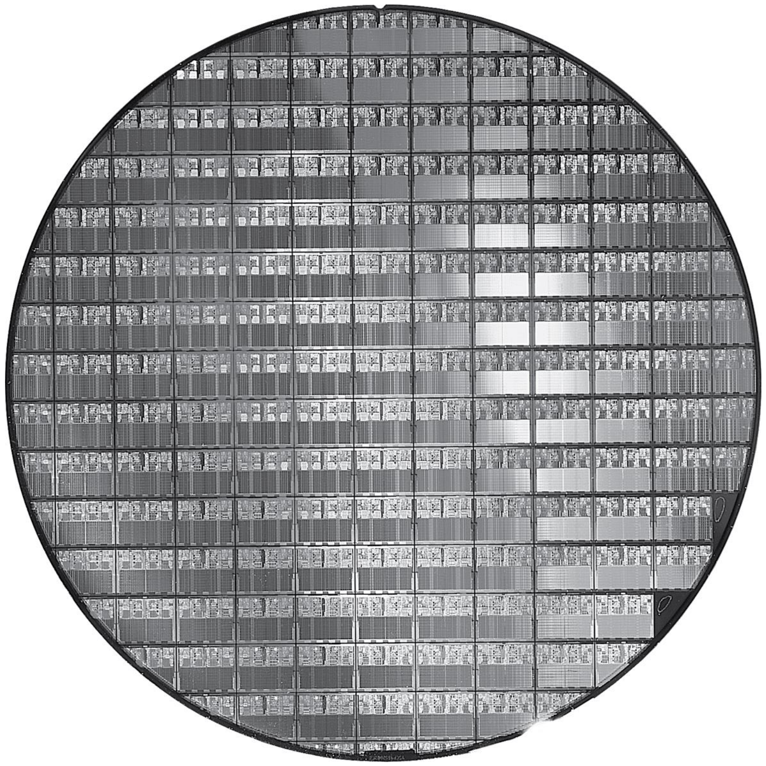
- Multicore microprocessors
  - More than one processor per chip
- Requires explicitly parallel programming
  - Compare with instruction level parallelism
    - Hardware executes multiple instructions at once
    - Hidden from the programmer
  - Hard to do
    - Programming for performance
    - Load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

Manufacturing ICs



- Yield: proportion of working dies per wafer

AMD Opteron X2 Wafer



- X2: 300mm wafer, 117 chips, 90nm technology
- X4: 45nm technology

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

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 Opteron X4 2356

Name	Description	IC×10^9	CPI	Tc (ns)	Exec time	Ref time	SPECratio
perl	Interpreted string processing	2,118	0.75	0.40	637	9,777	15.3
bzip2	Block-sorting compression	2,389	0.85	0.40	817	9,650	11.8
gcc	GNU C Compiler	1,050	1.72	0.47	24	8,050	11.1
mcf	Combinatorial optimization	336	10.00	0.40	1,345	9,120	6.8
go	Go game (AI)	1,658	1.09	0.40	721	10,490	14.6
hmmer	Search gene sequence	2,783	0.80	0.40	890	9,330	10.5
sjeng	Chess game (AI)	2,176	0.96	0.48	37	12,100	14.5
libquantum	Quantum computer simulation	1,623	1.61	0.40	1,047	20,720	19.8
h264avc	Video compression	3,102	0.80	0.40	993	22,130	22.3
omnetpp	Discrete event simulation	587	2.94	0.40	690	6,250	9.1
astar	Games/path finding	1,082	1.79	0.40	773	7,020	9.1
xalancbmk	XML parsing	1,058	2.70	0.40	1,143	6,900	6.0
Geometric mean							11.7

High cache miss rates

SPEC Power Benchmark

- Power consumption of server at different workload levels
  - Performance: ssj_ops/sec
  - Power: Watts (Joules/sec)

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

SPECpower_ssj2008 for X4

Target Load %	Performance (ssj_ops/sec)	Average Power (Watts)
100%	231,867	295
90%	211,282	286
80%	185,803	275
70%	163,427	265
60%	140,160	256
50%	118,324	246
40%	920,35	233
30%	70,500	222
20%	47,126	206
10%	23,066	180
0%	0	141
Overall sum	1,283,590	2,605
∑ssj_ops/ ∑power		493

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 X4 power benchmark
  - At 100% load: 295W
  - At 50% load: 246W (83%)
  - At 10% load: 180W (61%)
- 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

$$\begin{aligned}\text{MIPS} &= \frac{\text{Instruction count}}{\text{Execution time} \times 10^6} \\ &= \frac{\text{Instruction count}}{\dfrac{\text{Instruction count} \times \text{CPI}}{\text{Clock rate}} \times 10^6} = \frac{\text{Clock rate}}{\text{CPI} \times 10^6}\end{aligned}$$

  - CPI varies between programs on a given CPU

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