# 182.092 Computer Architecture

# Chapter 1: Abstractions and Technology

Adapted from

**Computer Organization and Design, 4th Edition,**

Patterson & Hennessy, © 2008, Morgan Kaufmann Publishers

and

Mary Jane Irwin (www.cse.psu.edu/research/mdl/mji)

# Course Administration

- Instructor: Herbert Grünbacher
  Herbert.Gruenbacher@tuwien.ac.at
  Treitlstraße 3/ 4. Stock
- TA: Wolfgang Puffitsch
  Wolfgang.Puffitsch@tuwien.ac.at
- Labs: http://www.tilab.tuwien.ac.at Treitlstraße 3/HP
- URL: http://ti.tuwien.ac.at/rts/teaching/courses/cavo
  http://ti.tuwien.ac.at/rts/teaching/courses/calu
- Text: *Computer Organization and Design*, 4th Ed.,
  D. A. Patterson, J. L. Hennessy
  Morgan Kaufmann Publishers, 2009
- Slides: Home page

# Grading, Course Structure & Schedule

- Oral exams, registration via TUWIS++
  - 18 June 2010
- Lecture: 09:00 to 11:00 Thursdays
  - 4 March to 27 May
- Lab:
  - Introduction 4 March, 10:15 to 11:00
- Lectures:
  - MIPS ISA and basic architecture
  - pipelined datapath design issues
  - superscalar/VLIW datapath design issues
  - memory hierarchies and memory design issues
  - storage and I/O design issues
  - multiprocessor design issues

# Course Content

- Memory hierarchy and design, CPU design, pipelining, multiprocessor architecture.

  "This course will introduce students to the architecture-level design issues of a computer system. They will apply their knowledge of digital logic design to explore the high-level interaction of the individual computer system hardware components. Concepts of sequential and parallel architecture including the interaction of different memory components, their layout and placement, communication among multiple processors, effects of pipelining, and performance issues, will be covered. Students will apply these concepts by studying and evaluating the merits and demerits of selected computer system architectures."

  - To learn what determines the capabilities and performance of computer systems and to understand the interactions between the computer's architecture and its software so that future software designers (compiler writers, operating system designers, database programmers, application programmers, …) can achieve the best cost-performance trade-offs and so that future architects understand the effects of their design choices on software.

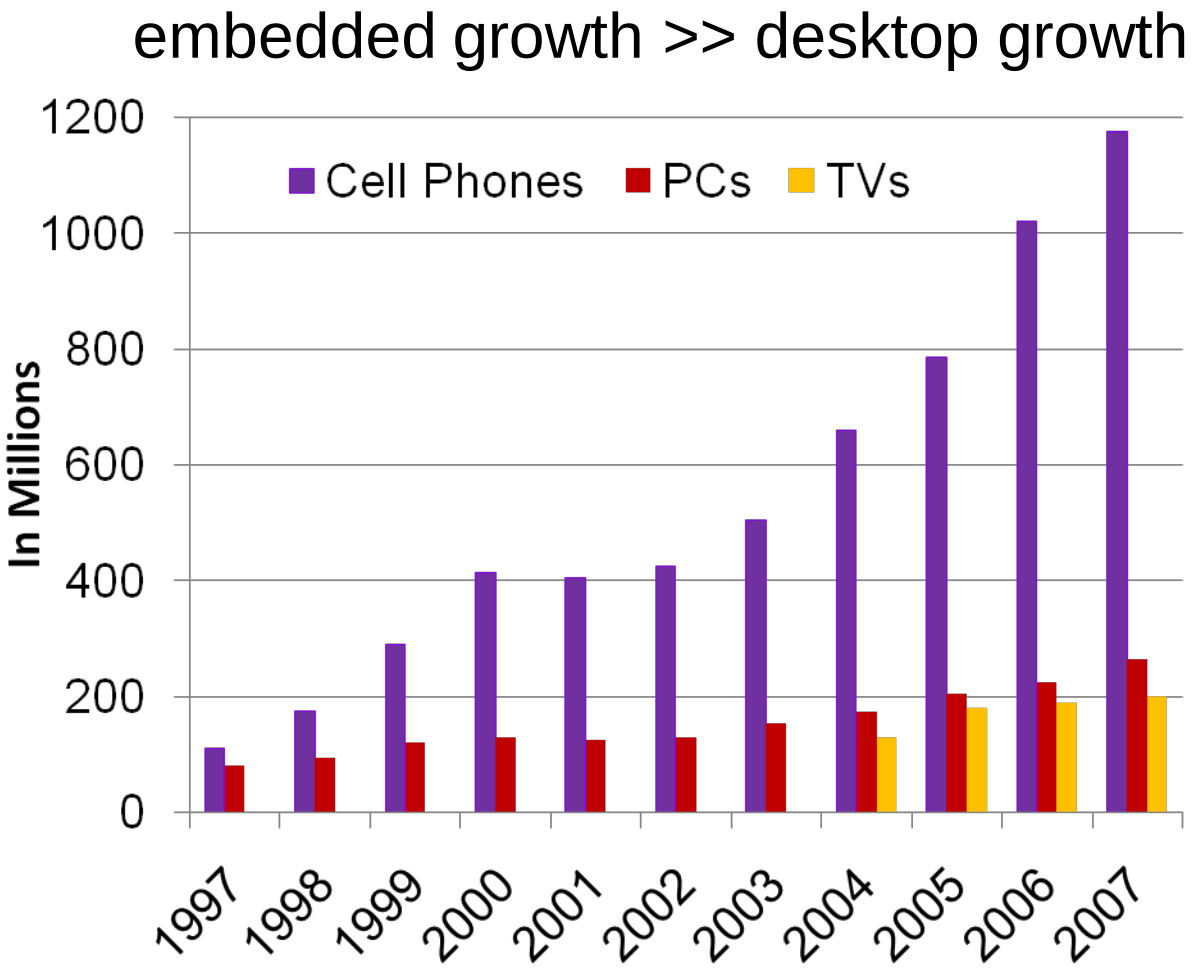
# What You Should Know

- Basic logic design & machine organization
  - logical minimization, FSMs, component design
  - processor, memory, I/O
- Create, assemble, run, debug programs in an assembly language
- Create, simulate, and debug hardware structures in a hardware description language
  - VHDL
- Create, compile, and run C (C++, Java) programs

# 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

# Growth in Cell Phone Sales (Embedded)

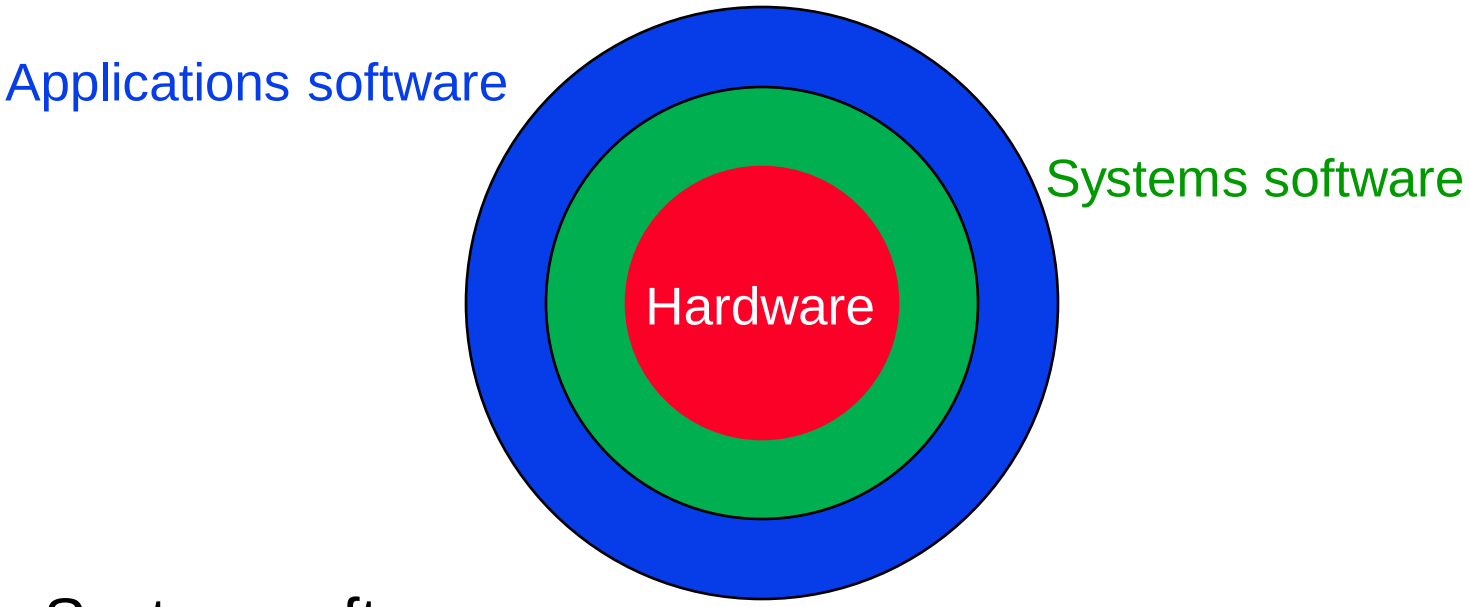


# Embedded Processor Characteristics

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

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

# Below the Program

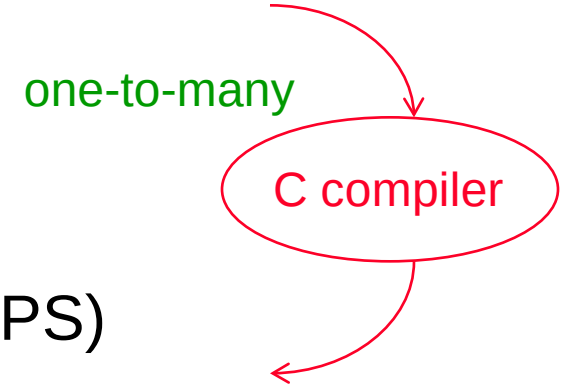



- System software
  - Operating system – supervising program that interfaces the user's program with the hardware (e.g., Linux, MacOS, Windows)
    - Handles basic input and output operations
    - Allocates storage and memory
    - Provides for protected sharing among multiple applications
  - Compiler – translate programs written in a high-level language (e.g., C, Java) into instructions that the hardware can execute

# Below the Program, Con't

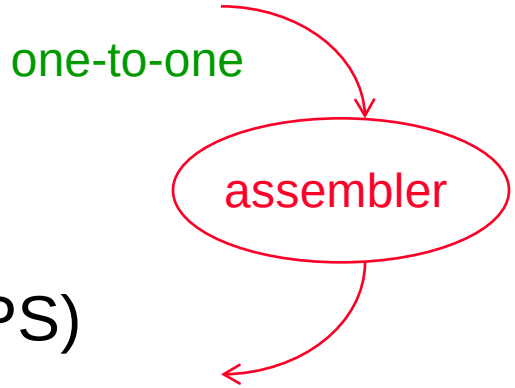
- High-level language program (in C)

```
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 00010000010000000
000000 00100 00010 00010000000100000
 . . .
```

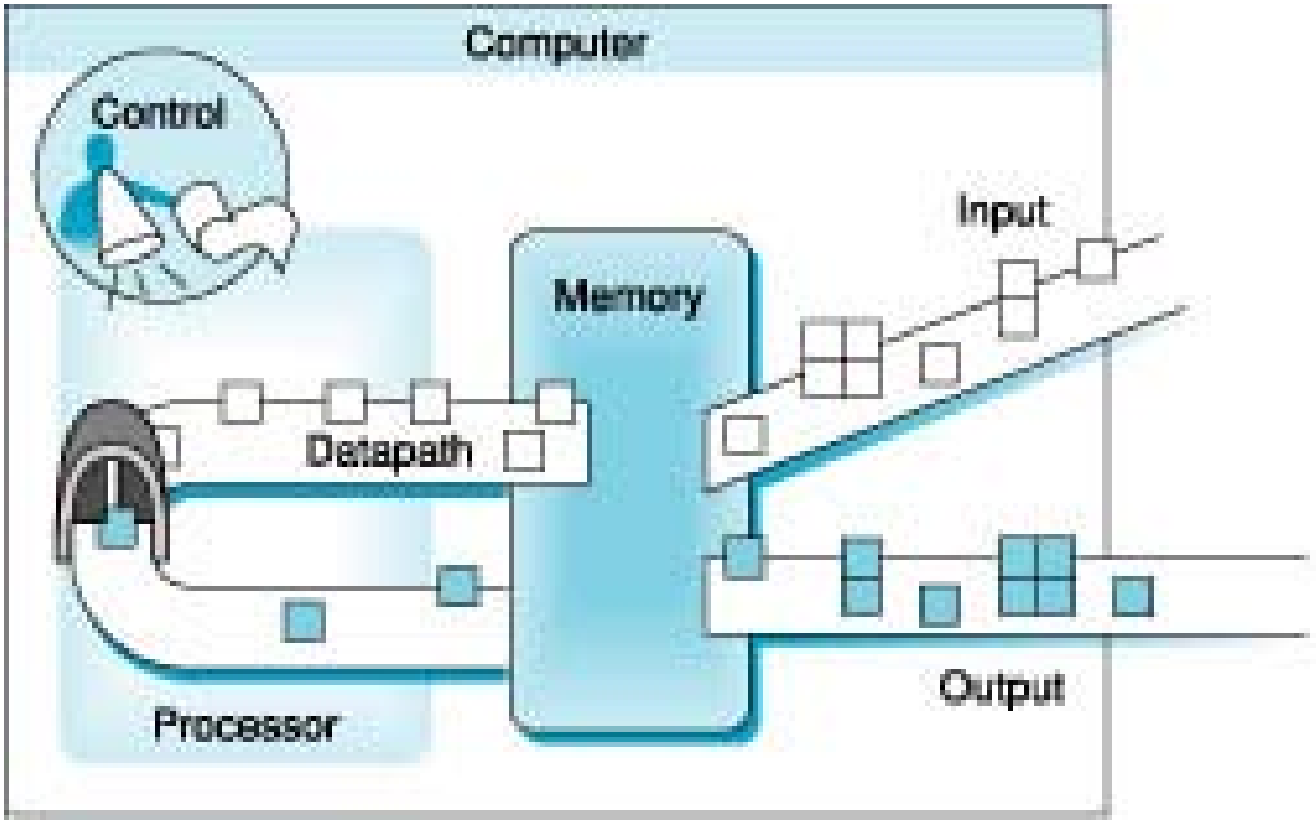
# Advantages of Higher-Level Languages ?

- 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

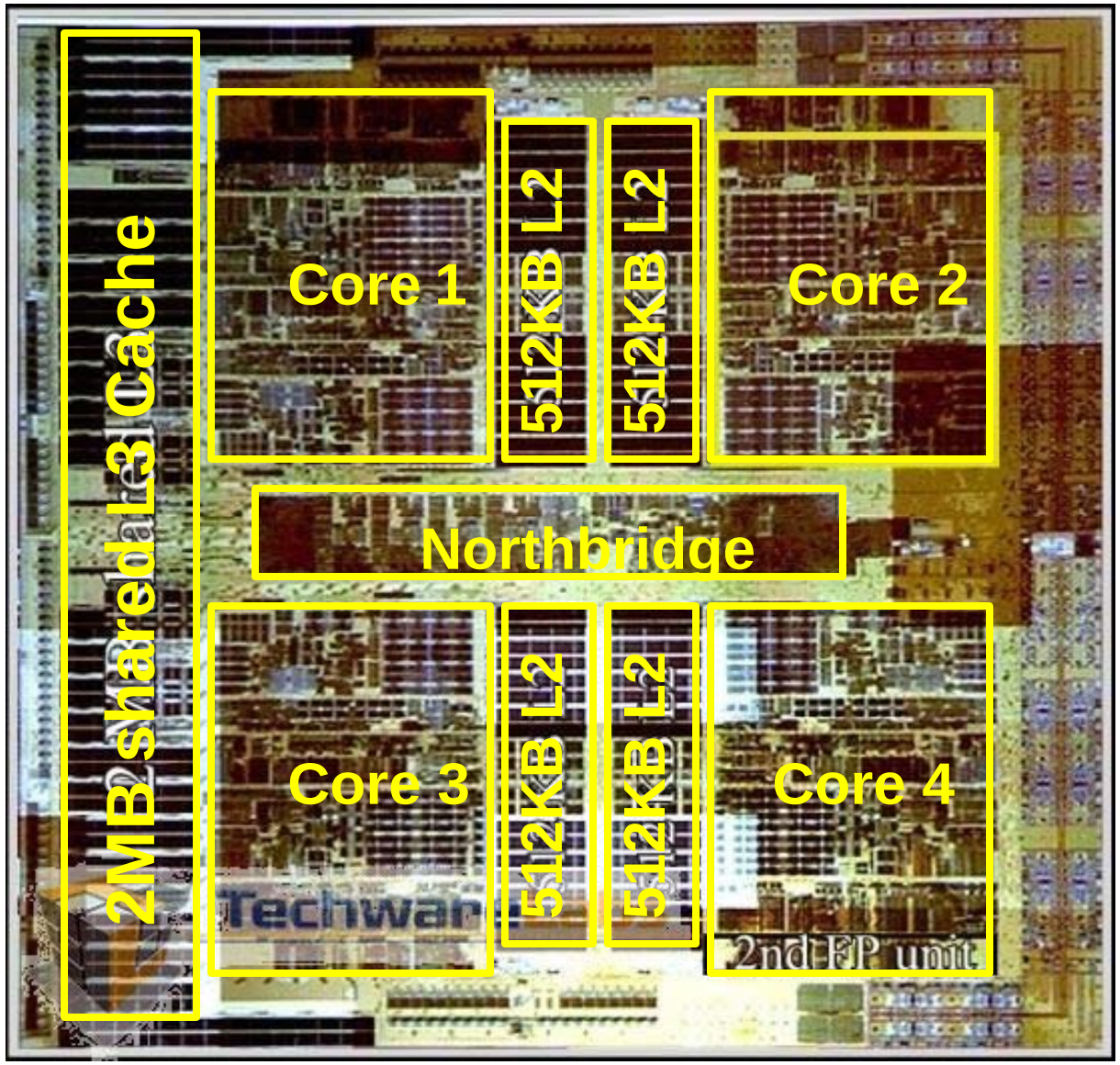
# Under the Covers

- Five classic components of a computer – input, output, memory, datapath, and control

- datapath + control = processor (CPU)



# AMD's Barcelona Multicore Chip




- Four out-of-order cores on one chip
- 1.9 GHz clock rate
- 65nm technology
- Three levels of caches (L1, L2, L3) on chip
- Integrated Northbridge

http://www.techwarelabs.com/reviews/processors/barcelona/

# Instruction Set Architecture (ISA)

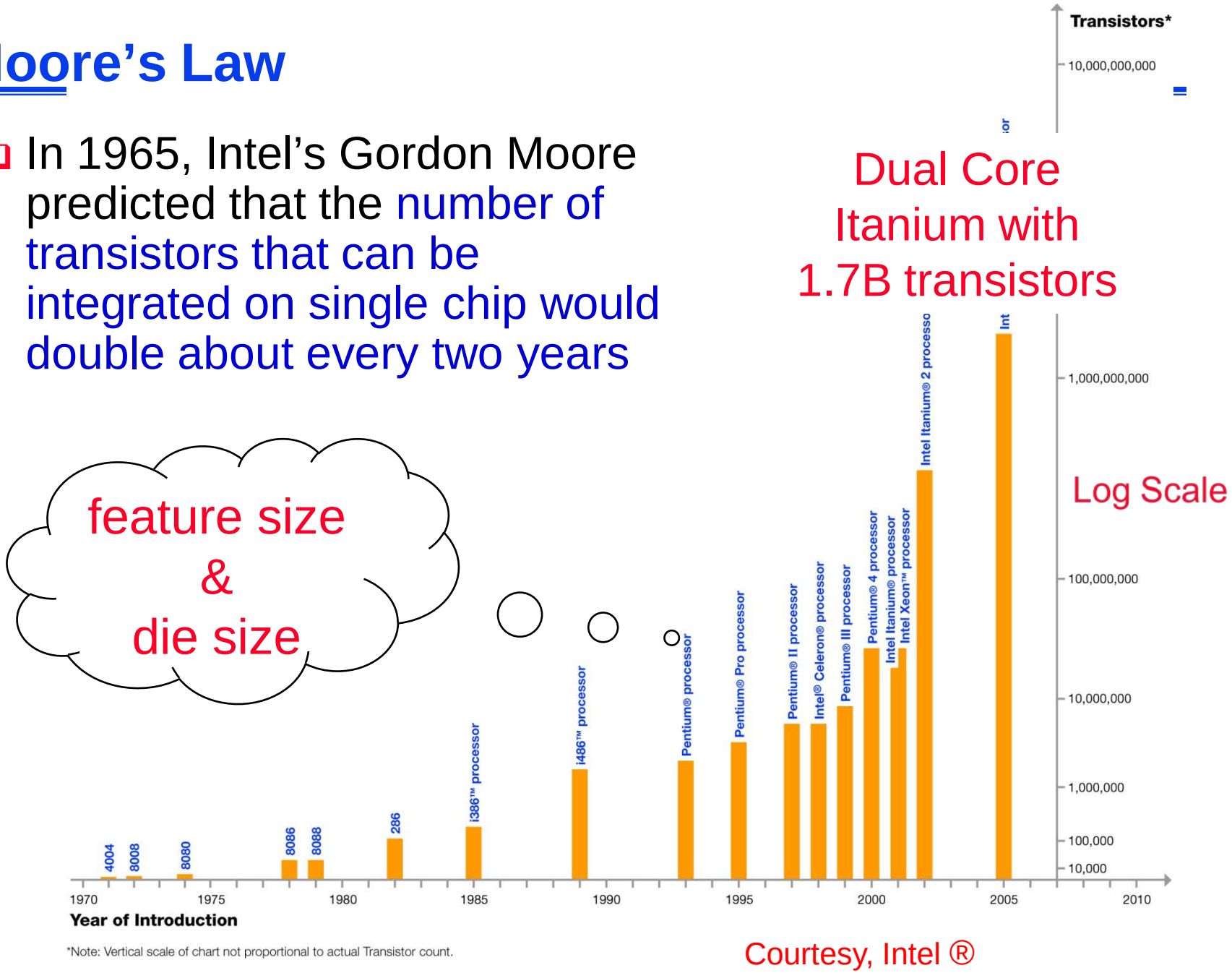
- ISA, or simply architecture – the abstract interface between the hardware and the lowest level software that encompasses all the information necessary to write a machine language program, including instructions, registers, memory access, I/O, …
  - Enables implementations of varying cost and performance to run identical software

- The combination of the basic instruction set (the ISA) and the operating system interface is called the application binary interface (ABI)
  - ABI – The user portion of the instruction set plus the operating system interfaces used by application programmers. Defines a standard for binary portability across computers.

# 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

Dual Core Itanium with 1.7B transistors




feature size
&
die size

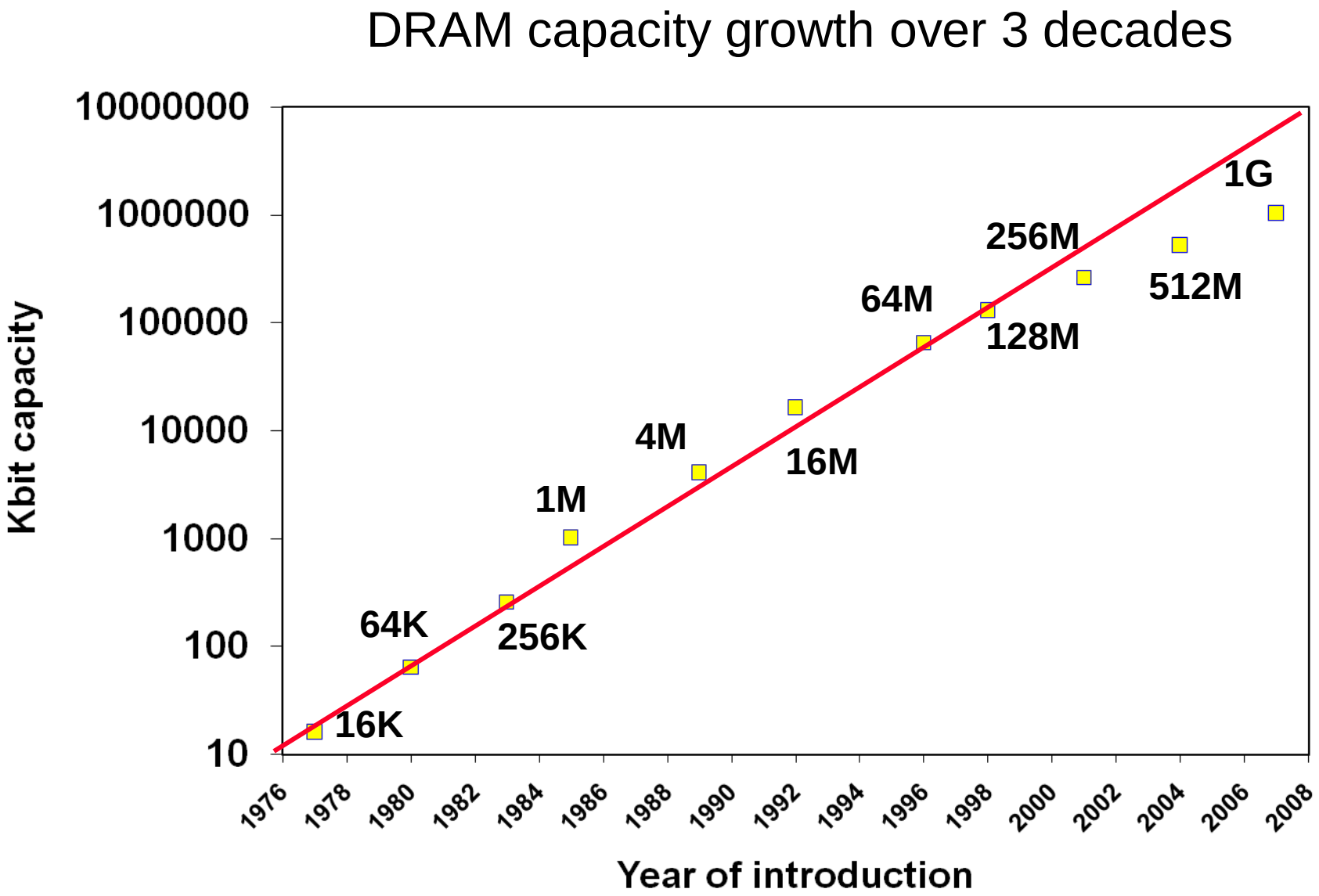
Courtesy, Intel ®

# Technology Scaling Road Map (ITRS)

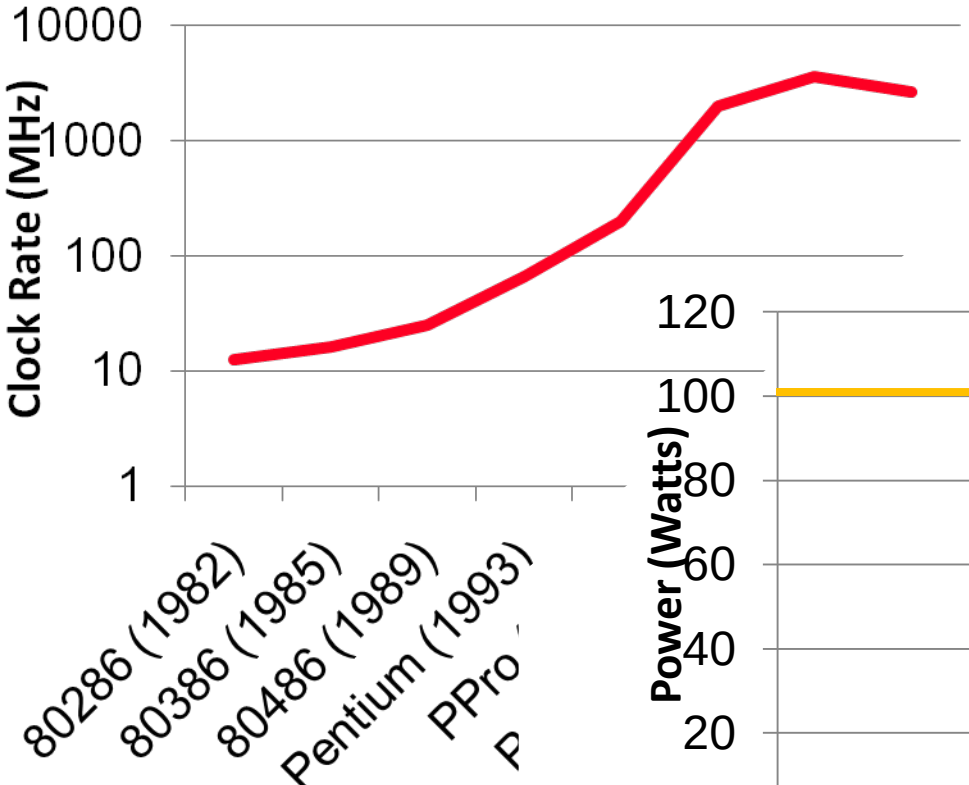
| Year | 2004 | 2006 | 2008 | 2010 | 2012 |
|---|---|---|---|---|---|
| Feature size (nm) | 90 | 65 | 45 | 32 | 22 |
| Intg. Capacity (BT) | 2 | 4 | 6 | 16 | 32 |

- Fun facts about 45nm transistors
  - 30 million can fit on the head of a pin
  - You could fit more than 2,000 across the width of a human hair
  - If car prices had fallen at the same rate as the price of a single transistor has since 1968, a new car today would cost about 1 cent

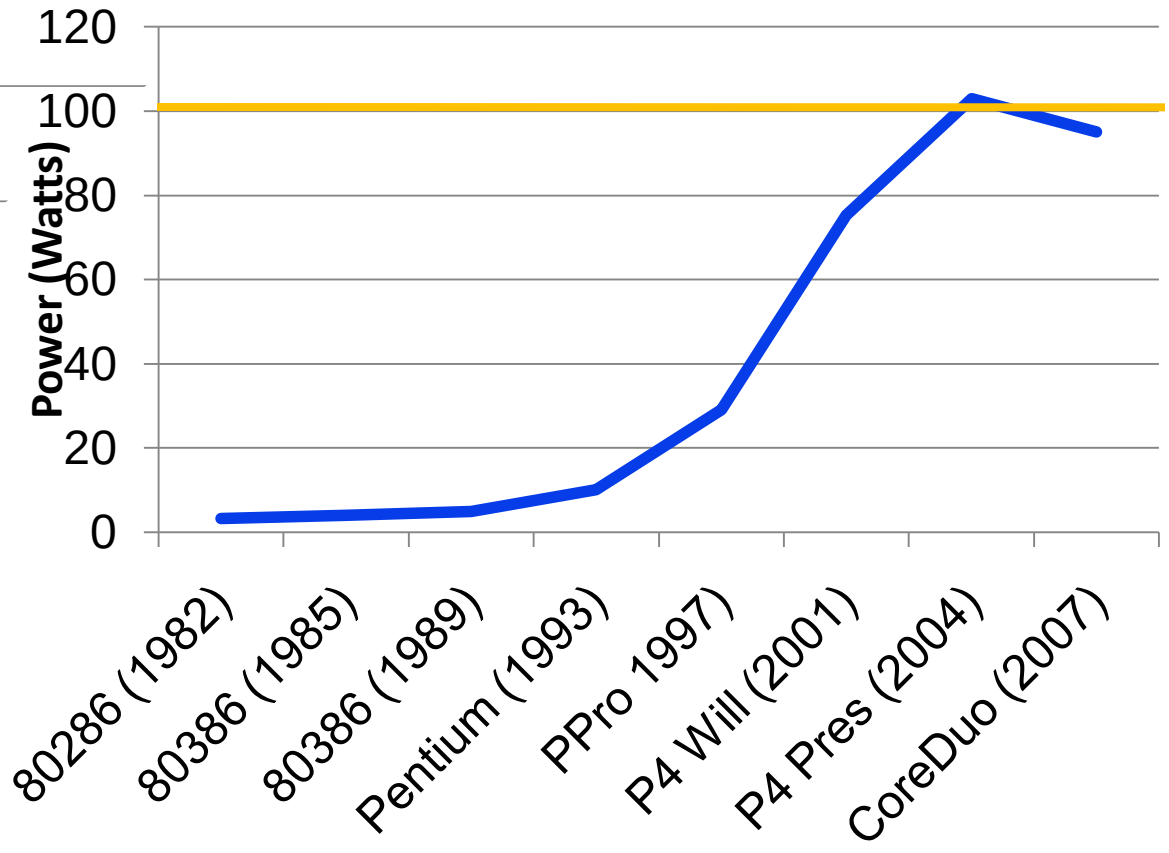
# Another Example of Moore's Law Impact



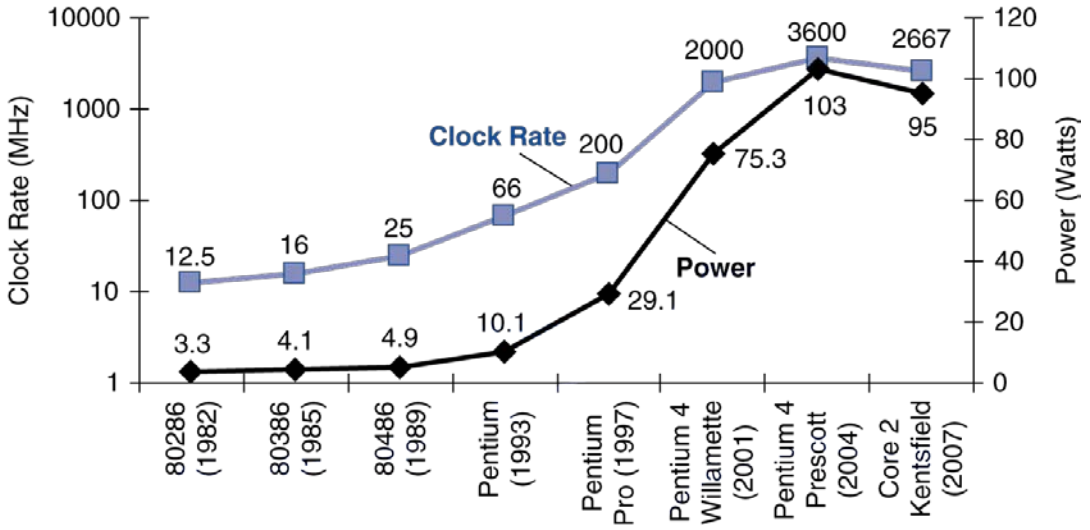
# But What Happened to Clock Rates and Why?




- Clock rates hit a "power wall"



# Power Trends




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

×30

5V → 1V

×1000

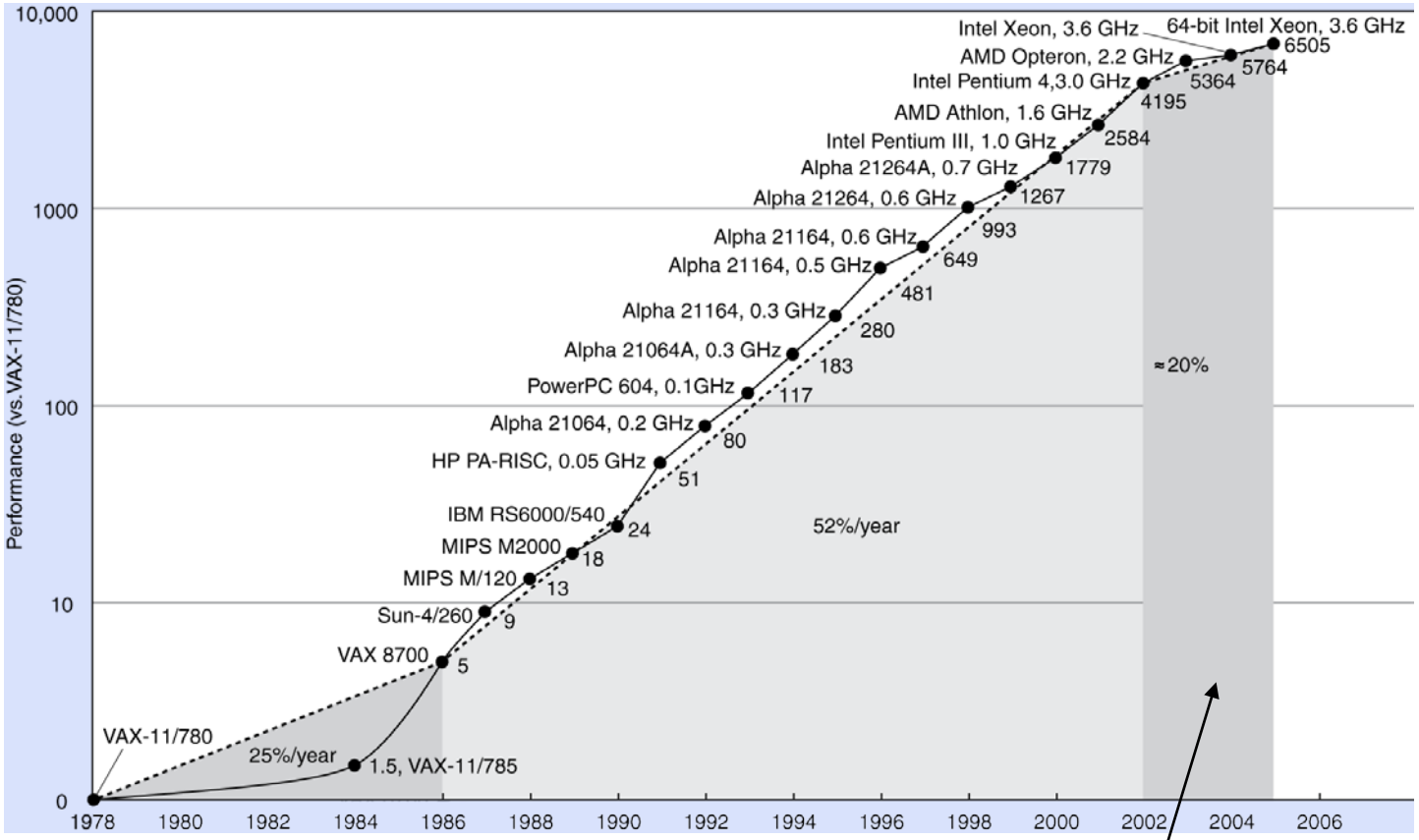
# Reducing Power

- 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

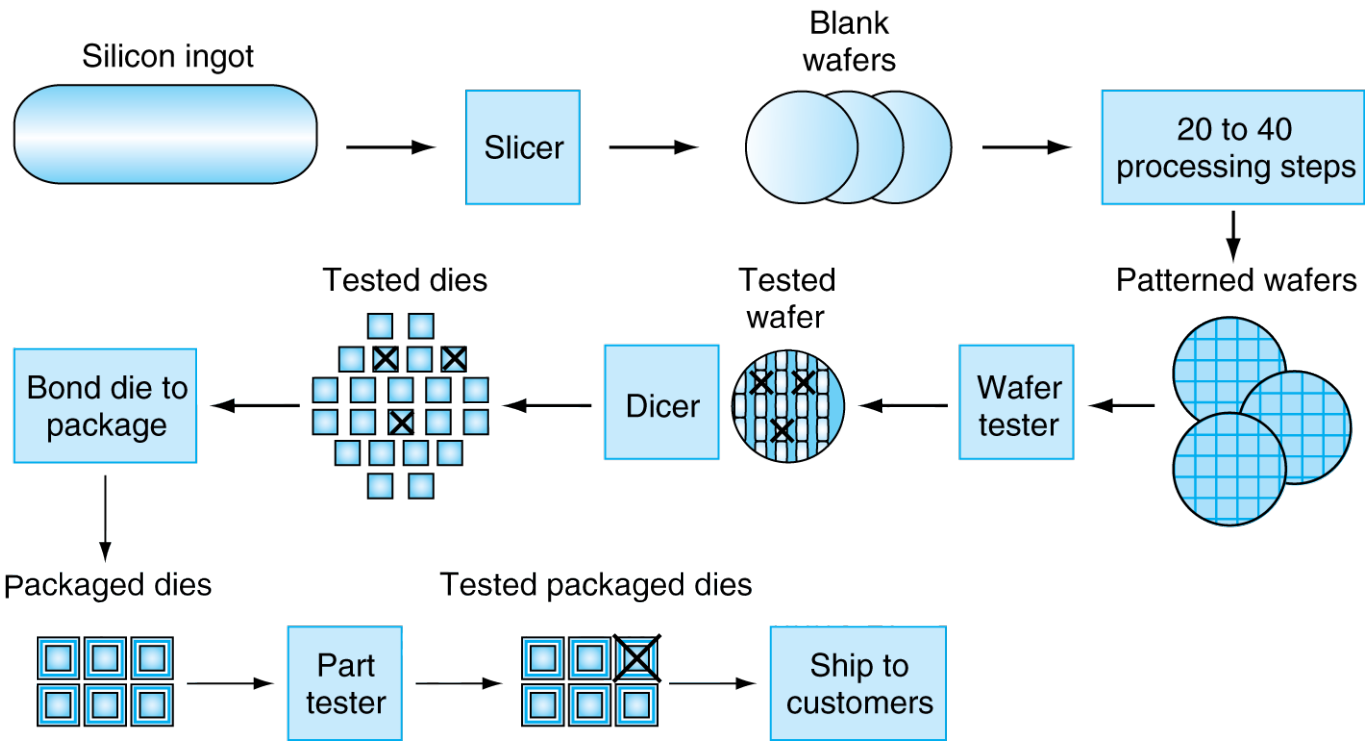
# A Sea Change is at Hand

- The power challenge has forced a change in the design of microprocessors
  - Since 2002 the rate of improvement in the response time of programs on desktop computers has slowed from a factor of 1.5 per year to less than a factor of 1.2 per year
- As of 2006 all desktop and server companies are shipping microprocessors with multiple processors – cores – per chip

| Product | AMD Barcelona | Intel Nehalem | IBM Power 6 | Sun Niagara 2 |
|---|---|---|---|---|
| Cores per chip | 4 | 4 | 2 | 8 |
| Clock rate | 2.5 GHz | ~2.5 GHz? | 4.7 GHz | 1.4 GHz |
| Power | 120 W | ~100 W? | ~100 W? | 94 W |

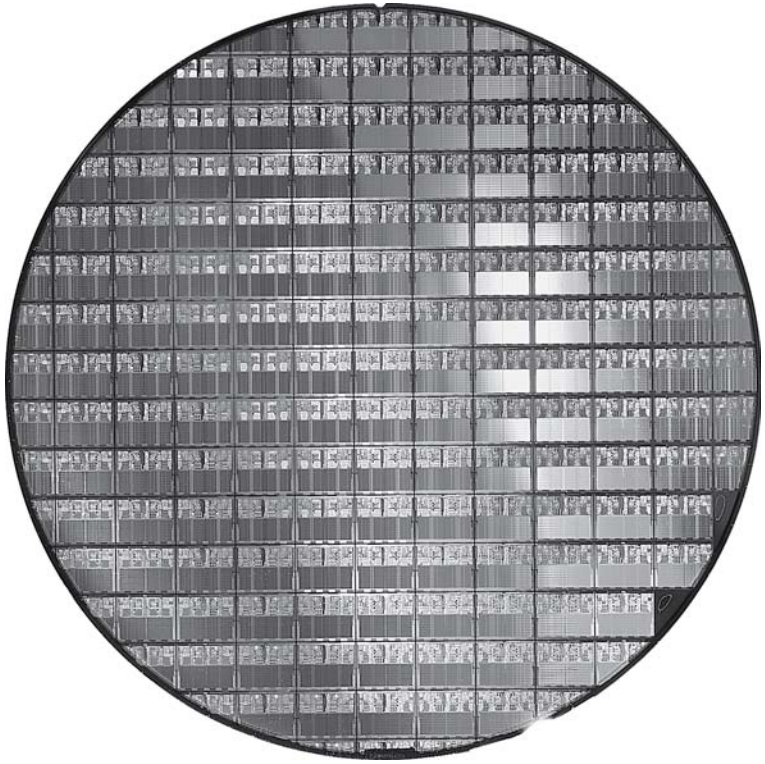
- Plan of record is to double the number of cores per chip per generation (about every two years)

# Manufacturing ICs




- Yield: proportion of working dies per wafer

# AMD Opteron X2 Wafer



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

# Integrated Cicuit 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

# Performance Metrics

- Purchasing perspective
  - given a collection of machines, which has the
    - best performance ?
    - least cost ?
    - best cost/performance?
- Design perspective
  - faced with design options, which has the
    - best performance improvement ?
    - least cost ?
    - best cost/performance?
- Both require
  - basis for comparison
  - metric for evaluation
- Our goal is to understand what factors in the architecture contribute to overall system performance and the relative importance (and cost) of these factors

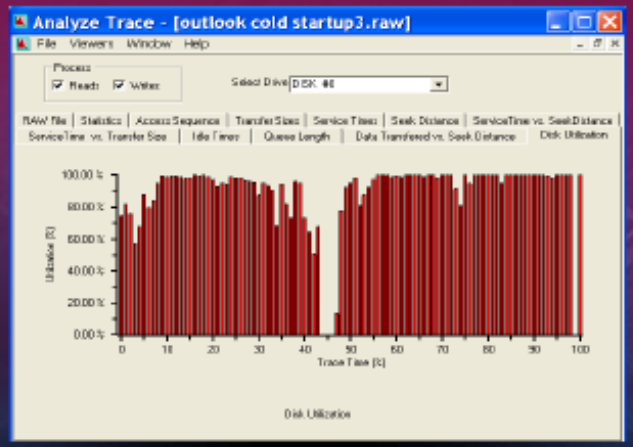
# Throughput versus Response Time

- Response time (execution time) – the time between the start and the completion of a task
  - Important to individual users
- Throughput (bandwidth) – the total amount of work done in a given time
  - Important to data center managers
- Will need different performance metrics as well as a different set of applications to benchmark embedded and desktop computers, which are more focused on response time, versus servers, which are more focused on throughput

# Response Time Matters

## It's the Hard Disk, Stupid!

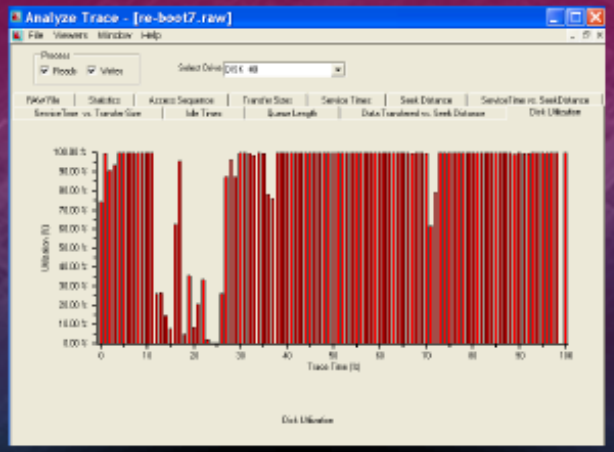
### Re-Boot/Startup on Home PC




| | |
|---|---|
| Elapsed Time | 105.213536, s |
| Disk Busy Time | 91.368480, s |
| Average Data Rate | 6.60669, MB/s |

**86% BUSY**

### Starting Outlook




| | |
|---|---|
| Elapsed Time | 45.700667, s |
| Disk Busy Time | 41.056997, s |
| Average Data Rate | 1.37389, MB/s |

**89% BUSY**

intel



Justin Rattner's ISCA'08 Keynote (VP and CTO of Intel)

# Defining (Speed) Performance

- To maximize performance, need to minimize execution time

$$\text{performance}_X = 1 / \text{execution\_time}_X$$

If X is n times faster than Y, then

$$\frac{\text{performance}_X}{\text{performance}_Y} = \frac{\text{execution\_time}_Y}{\text{execution\_time}_X} = n$$

- Decreasing response time almost always improves throughput

# Relative Performance Example

- If computer A runs a program in 10 seconds and computer B runs the same program in 15 seconds, how much faster is A than B?

We know that A is n times faster than B if

$$\frac{\text{performance}_A}{\text{performance}_B} = \frac{\text{execution\_time}_B}{\text{execution\_time}_A} = n$$

The performance ratio is $\frac{15}{10} = 1.5$

So A is 1.5 times faster than B

# Performance Factors

- CPU execution time (CPU time) – time the CPU spends working on a task
  - Does not include time waiting for I/O or running other programs

$$\text{CPU execution time for a program} = \text{\# CPU clock cycles for a program} \times \text{clock cycle time}$$

or

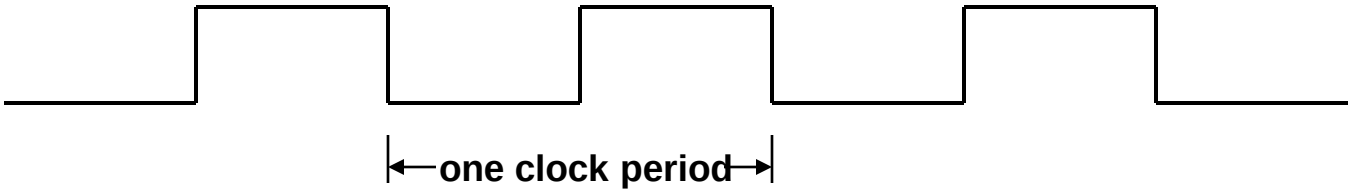
$$\text{CPU execution time for a program} = \frac{\text{\# CPU clock cycles for a program}}{\text{clock rate}}$$

- Can improve performance by reducing either the length of the clock cycle or the number of clock cycles required for a program

# Review: Machine Clock Rate

- Clock rate (clock cycles per second in MHz or GHz) is inverse of clock cycle time (clock period)

$$CC = 1 / CR$$




10 nsec clock cycle => 100 MHz clock rate

5 nsec clock cycle => 200 MHz clock rate

2 nsec clock cycle => 500 MHz clock rate

1 nsec ($10^{-9}$) clock cycle => 1 GHz ($10^{9}$) clock rate

500 psec clock cycle => 2 GHz clock rate

250 psec clock cycle => 4 GHz clock rate

200 psec clock cycle => 5 GHz clock rate

# Improving Performance Example

- A program runs on computer A with a 2 GHz clock in 10 seconds. What clock rate must computer B run at to run this program in 6 seconds? Unfortunately, to accomplish this, computer B will require 1.2 times as many clock cycles as computer A to run the program.

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

$$\text{CPU clock cycles}_A = 10 \text{ sec} \times 2 \times 10^9 \text{ cycles/sec} = 20 \times 10^9 \text{ cycles}$$

$$\text{CPU time}_B = \frac{1.2 \times 20 \times 10^9 \text{ cycles}}{\text{clock rate}_B}$$

$$\text{clock rate}_B = \frac{1.2 \times 20 \times 10^9 \text{ cycles}}{6 \text{ seconds}} = 4 \text{ GHz}$$

# Clock Cycles per Instruction

- Not all instructions take the same amount of time to execute
  - One way to think about execution time is that it equals the number of instructions executed multiplied by the average time per instruction

$$\text{\# CPU clock cycles for a program} = \text{\# Instructions for a program} \times \text{Average clock cycles per instruction}$$

- Clock cycles per instruction (CPI) – the average number of clock cycles each instruction takes to execute
  - A way to compare two different implementations of the same ISA

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

# Using the Performance Equation

- Computers A and B implement the same ISA. Computer A has a clock cycle time of 250 ps and an effective CPI of 2.0 for some program and computer B has a clock cycle time of 500 ps and an effective CPI of 1.2 for the same program. Which computer is faster and by how much?

Each computer executes the same number of instructions, *I*, so

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

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

Clearly, A is faster … by the ratio of execution times

$$\frac{\text{performance}_A}{\text{performance}_B} = \frac{\text{execution\_time}_B}{\text{execution\_time}_A} = \frac{600 \times I \text{ ps}}{500 \times I \text{ ps}} = 1.2$$

# Effective (Average) CPI

- Computing the overall effective CPI is done by looking at the different types of instructions and their individual cycle counts and averaging

$$\text{Overall effective CPI} = \sum_{i=1}^{n} (CPI_i \times IC_i)$$

  - Where $IC_i$ is the count (percentage) of the number of instructions of class i executed
  - $CPI_i$ is the (average) number of clock cycles per instruction for that instruction class
  - n is the number of instruction classes

- The overall effective CPI varies by instruction mix – a measure of the dynamic frequency of instructions across one or many programs

# THE Performance Equation

- Our basic performance equation is then

$$\text{CPU time} = \text{Instruction\_count} \times \text{CPI} \times \text{clock\_cycle}$$

or

$$\text{CPU time} = \frac{\text{Instruction\_count} \times \text{CPI}}{\text{clock\_rate}}$$

- These equations separate the three key factors that affect performance
  - Can measure the CPU execution time by running the program
  - The clock rate is usually given
  - Can measure overall instruction count by using profilers/ simulators without knowing all of the implementation details
  - CPI varies by instruction type and ISA implementation for which we must know the implementation details

# Determinates of CPU Performance

CPU time = Instruction_count x CPI x clock_cycle

| | Instruction_ count | CPI | clock_cycle |
|---|---|---|---|
| Algorithm | X | X | |
| Programming language | X | X | |
| Compiler | X | X | |
| ISA | X | X | X |
| Core organization | | X | X |
| Technology | | | X |

# A Simple Example

| Op | Freq | $CPI_i$ | Freq x $CPI_i$ | | | |
|---|---|---|---|---|---|---|
| ALU | 50% | 1 | .5 | .5 | .5 | .25 |
| Load | 20% | 5 | 1.0 | .4 | 1.0 | 1.0 |
| Store | 10% | 3 | .3 | .3 | .3 | .3 |
| Branch | 20% | 2 | .4 | .4 | .2 | .4 |
| | | | $\Sigma =$ 2.2 | 1.6 | 2.0 | 1.95 |

- How much faster would the machine be if a better data cache reduced the average load time to 2 cycles?

  CPU time new = 1.6 x IC x CC  so  2.2/1.6  means 37.5% faster

- How does this compare with using branch prediction to shave a cycle off the branch time?

  CPU time new = 2.0 x IC x CC  so  2.2/2.0  means 10% faster

- What if two ALU instructions could be executed at once?

  CPU time new = 1.95 x IC x CC  so  2.2/1.95  means 12.8% faster

# Workloads and Benchmarks

- Benchmarks – a set of programs that form a “workload” specifically chosen to measure performance
- SPEC (System Performance Evaluation Cooperative) creates standard sets of benchmarks starting with SPEC89. The latest is SPEC CPU2006 which consists of 12 integer benchmarks (CINT2006) and 17 floating-point benchmarks (CFP2006).

www.spec.org

- There are also benchmark collections for power workloads (SPECpower_ssj2008), for mail workloads (SPECmail2008), for multimedia workloads (mediabench), …

# SPEC CINT2006 on Barcelona (CC = 0.4 x $10^9$)

| Name | ICx$10^9$ | CPI | ExTime | RefTime | SPEC ratio |
|---|---|---|---|---|---|
| perl | 2,1118 | 0.75 | 637 | 9,770 | 15.3 |
| bzip2 | 2,389 | 0.85 | 817 | 9,650 | 11.8 |
| gcc | 1,050 | 1.72 | 724 | 8,050 | 11.1 |
| mcf | 336 | 10.00 | 1,345 | 9,120 | 6.8 |
| go | 1,658 | 1.09 | 721 | 10,490 | 14.6 |
| hmmer | 2,783 | 0.80 | 890 | 9,330 | 10.5 |
| sjeng | 2,176 | 0.96 | 837 | 12,100 | 14.5 |
| libquantum | 1,623 | 1.61 | 1,047 | 20,720 | 19.8 |
| h264avc | 3,102 | 0.80 | 993 | 22,130 | 22.3 |
| omnetpp | 587 | 2.94 | 690 | 6,250 | 9.1 |
| astar | 1,082 | 1.79 | 773 | 7,020 | 9.1 |
| xalancbmk | 1,058 | 2.70 | 1,143 | 6,900 | 6.0 |
| Geometric Mean | | | | | **11.7** |

# Comparing and Summarizing Performance

- How do we summarize the performance for benchmark set with a single number?
  - First the execution times are normalized giving the “SPEC ratio” (bigger is faster, i.e., SPEC ratio is the inverse of execution time)
  - The SPEC ratios are then “averaged” using the geometric mean (GM)

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

- Guiding principle in reporting performance measurements is reproducibility – list everything another experimenter would need to duplicate the experiment (version of the operating system, compiler settings, input set used, specific computer configuration (clock rate, cache sizes and speed, memory size and speed, etc.))

# Summary: Evaluating ISAs

- Design-time metrics:
  - Can it be implemented, in how long, at what cost?
  - Can it be programmed? Ease of compilation?
- Static Metrics:
  - How many bytes does the program occupy in memory?
- Dynamic Metrics:
  - How many instructions are executed? How many bytes does the processor fetch to execute the program?
  - How many clocks are required per instruction?
  - How "lean" a clock is practical?

*Best Metric*: Time to execute the program!

depends on the instructions set, the processor organization, and compilation techniques.