Computer System Architecture

Introduction

Chalermek Intanagonwiwat

Slides courtesy of Peiyi Tang, David Culler, Graham Kirby, and Zoltan Somogyi

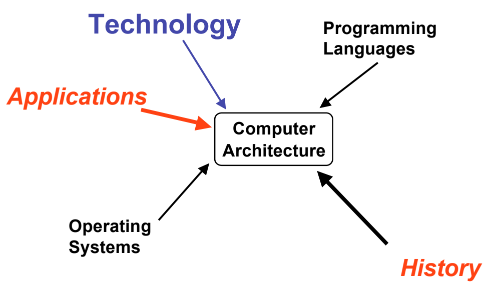
Why take this class?

- To design the next great instruction set?...well...
  - Instruction Set Architecture (ISA) has largely converged
  - Especially in the desktop / server / laptop space
  - Dictated by powerful market forces
- Tremendous organizational innovation relative to established ISA abstractions

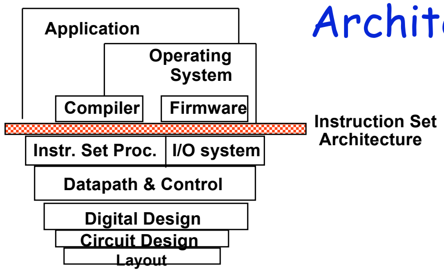
Why take this class? (cont.)

- Many New instruction sets or equivalent
  - embedded space, controllers, and specialized devices
- Design, analysis, implementation concepts vital to all aspects of CE & CS
- Equip you with an intellectual toolbox for dealing with a host of systems design challenges

Forces on Computer Architecture



What is "Computer Architecture"?



- Coordination of many *levels of abstraction*
- Under a rapidly changing set of forces
- Design, Measurement, *and* Evaluation

Computer Design

- What are the principal goals?
  - performance, performance, performance...
  - but not at any cost
- Trade-offs:
  - need to understand cost and performance issues
  - need models and measures of cost and performance

Tasks of Computer Designers (Architects)

- Designing a computer involves:
  - instruction set architecture (ISA) – programmer visible
  - computer organization – CPU internals, memory, buses, ...
  - computer hardware – logic design, packaging, ...
- Architects must meet:
  - functional requirements
    - »market & application driven
  - performance goals
  - cost constraints

Functional Requirements

- Application area
  - general purpose, scientific, commercial
- Operating system requirements
  - address space, memory management, protection
  - context switching, interrupts
- Standards
  - floating-point, I/O interconnect, operating systems, networks, programming languages

Functional Requirements (cont.)

- Given these requirements, optimize cost/performance trade-off
  - e.g., hardware or software implementation of a feature
- Design complexity
  - time to market is critical

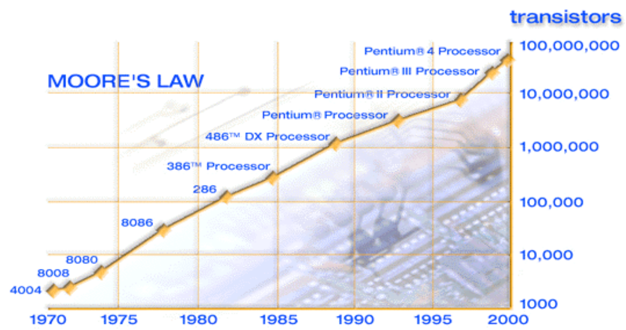
Technology Trends

- Software trends
  - increasing memory usage (from increasing functionality?)
    - » 1.5x to 2x per year - up to one address bit/year
  - use of high-level languages - use of compilers
    - » ISA designed for the compiler, not the programmer
  - improved compiler technology – optimization, scheduling

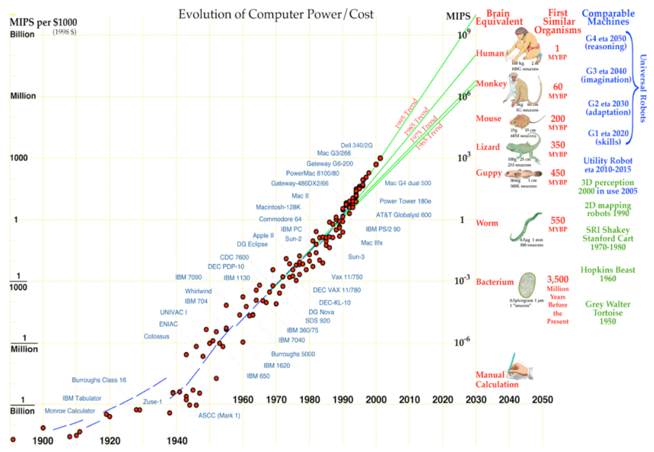
Technology Trends (cont.)

- Hardware trends
  - IC technology – density & size - transistor count; cycle time
  - DRAM – capacity 4x per 3 years, but slow cycle time change
  - disk – capacity was 2x per 3 years before 1990, now 4x per 3 years,
    - » slow change in access time
- Need to be aware of trends when designing computers
  - design for requirements and technology at time of shipping

Moore's Law



http://www.intel.com/research/silicon/mooreslaw.htm

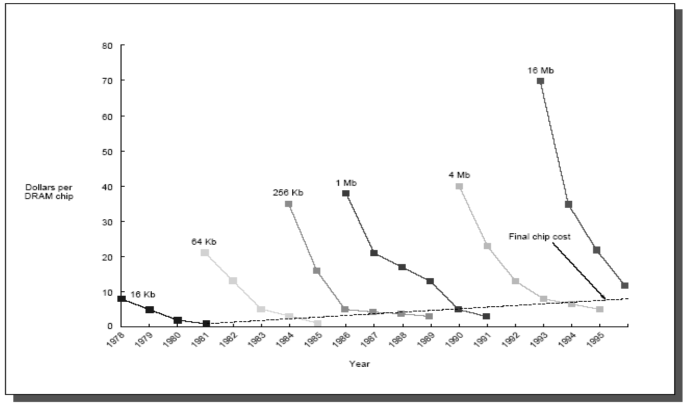


http://www.frc.ri.cmu.edu/~hpm/talks/revo.slides/power.aug.curve/power.aug.html

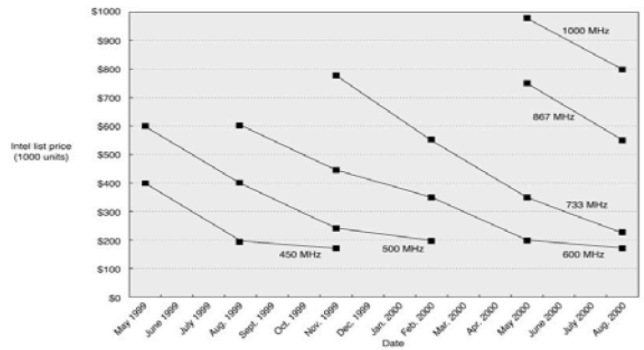
Cost and Trends in Cost

- Learning curve brings manufacturing cost down
  - DRAM cost drops 40% per year
- Large volume increases purchasing and manufacturing efficiency
  - bringing both cost and selling price down
- Commodization brings both cost and price down

Memory Price



Pentium III Cost



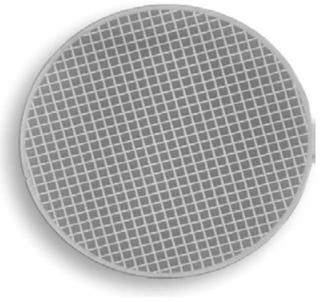
IC Cost

- Manufacture of an IC involves
  - making the wafer
  - testing dies on the wafer
  - chopping wafer into dies
  - packaging
  - final testing

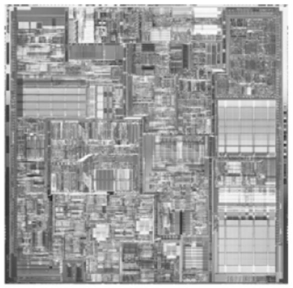
$$\text{Cost}_{\text{IC}} = \frac{\text{Cost}_{\text{die}} + \text{Cost}_{\text{testing}} + \text{Cost}_{\text{packaging}}}{\text{Final Test Yield}}$$

Wafer

- 8 inch diameter
- 564 MIPS processors
- 0.18μ process



Pentium 4 Die



Cost of Die

- Manufacturing process determines
  - cost of wafer, wafer yield, defect rate
- IC designer controls die area
- Area determined by both circuit elements and I/O pads
  - lots of pins increases die cost
- Cost of die $\propto$ Area^n
  - where n between about 2.0 and 4.0
- Also fixed costs (e.g., mask costs, setting up fabrication)

Cost of Die (cont.)

$$\text{Cost of die} = \frac{\text{Cost of wafer}}{\text{Dies per water} \times \text{Die yield}}$$

$$\text{Dies per water} = \frac{\pi \times (\text{Wafer Diameter}/2)^2}{\text{Dies area}} - \frac{\pi \times \text{Wafer Diameter}}{\sqrt{2 \times \text{Die area}}}$$

$$\text{Die yield} = \text{Wafer yield} \times (1 + \frac{\text{Defects per unit area} \times \text{Die area}}{\alpha})^{-\alpha}$$

where α is the manufacturing complexity factor, which is 3.0 for the multilevel metal CMOS in 1995.

Cost of Components

- Example: component costs in a workstation:

– Cabinet & packaging	4%	6%
– Circuit board – processor	6%	22%
– DRAM (64/128MB)	36%	5%
– video system	14%	5%
– PCB & I/O system	4%	5%
– I/O devices – keyboard/mouse	1%	3%
– monitor	22%	19%
– disk (1/20GB)	7%	9%
– CD/DVD drive	6%	6%

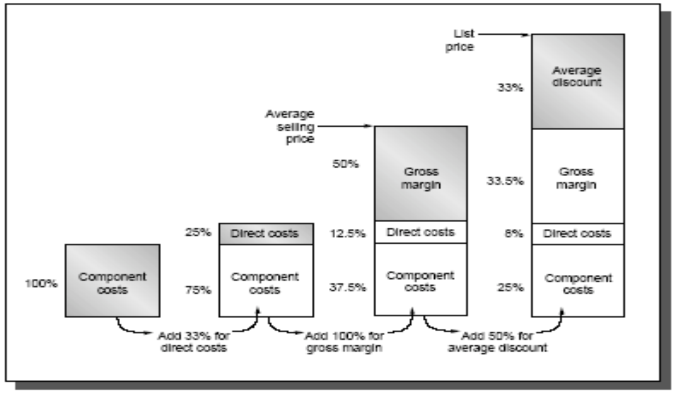
Cost of Components (cont.)

- Although IC cost is a differentiator
  - it is not a major cost component
- Cost reductions over time offset by increased resources required
  - E.g., more DRAM & disk,...

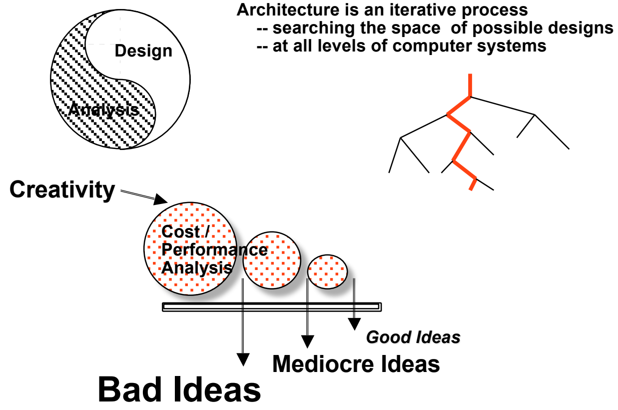
From Component Costs to Product Prices

- Direct Cost:
  - 20-40% of component cost for labor, warranty, etc.
- Gross Margin:
  - 20-55% of the average selling price for research and development, marketing, etc.
- Average Discount:
  - 40-50% of the list price for retailers' margin

Price Components



Measurement and Evaluation



Performance

- Many performance metrics are context dependent
  - response time: time from start to completion of a job
  - throughput: rate of job completion
- Usual question: how much faster is X than Y?
  - depends on execution time

Performance (cont.)

- “X is n times faster than Y” means:

$$n = \frac{\text{Performance}_X}{\text{Performance}_Y} = \frac{\text{Execution Time}_Y}{\text{Execution Time}_X}$$

Measuring Performance

- Difficulties
  - what to measure
  - interference
  - reproducibility
  - comparability
- Only consistent and reliable measure:
  - the **time** taken to run **real programs**

Measuring Performance (cont.)

- Execution time best measured using **elapsed time**
  - e.g. from the clock on the wall
  - includes all aspects of execution — what the user sees
- Can use a tool such as Unix time command to make measurements:

```
graham% time ls
2003-09-30.xbk week_01.pdf week_01_handout.ppt
misc week_01.ppt
0.000u 0.010s 0:00.00 0.0%
```

Measuring Performance (cont.)

- On a multi-programmed system, some time spent on other jobs
  - use an otherwise unloaded system to make measurements

Benchmarks

- Real applications
  - the kind of programs run in real life, with real I/O, options, ...
    - » e.g., compiler, text processor
- Scripted applications
  - to reproduce interactive or multi-user behavior
- Kernels
  - key parts of real programs used to evaluate aspects of performance

Benchmarks (cont.)

- Toy benchmarks - small programs with known results
  - » e.g., Quicksort
- Synthetic benchmarks
  - constructed to match typical behavior of real programs
    - » e.g., Whetstone, Dhrystone

SPEC Benchmarks

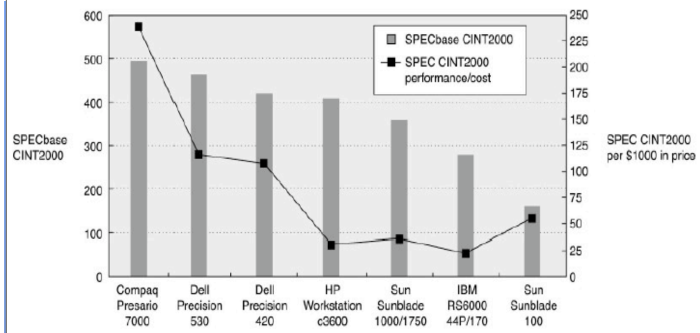
- Benchmark suite
  - better indication of overall performance?
- Standard Performance Evaluation Corporation (SPEC)
  - formed in response to lack of believable benchmarks
  - SPEC92, SPEC95, SPEC2000 — mix of integer & floating-point benchmarks, including kernels, small programs and real programs

SPEC Benchmarks (cont.)

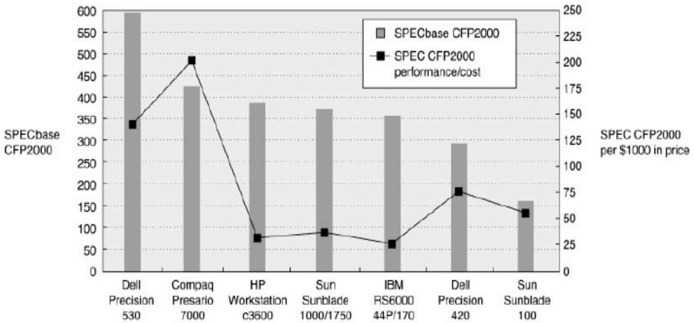
- SPEC reports
  - detailed machine configuration and compiler options, and includes measured data
    - » aim for reproducibility
    - » unlike figures often reported in magazines!
  - also compare baseline with optimized performance
- Result summarized as SPECmarks
  - relative to reference machine: VAX-11/780 = 1

http://www.spec.org/

Integer SPEC Results



Floating Point SPEC Results



Reporting Performance

- Want repeatable results
  - experimental science
  - predict running time for X on Y
- How do we compare machines based on collections of execution times for each?

Reporting Performance: Example

	Computer A	Computer B	Computer C
Program P1	1s	10s	20s
Program P2	1000s	100s	20s
Total	1001s	110s	40s

Combining Performance Measures

- Arithmetic mean tracks total execution time in this case

$$\text{Time}_{\text{ave}} = \frac{1}{n}\sum_{i=1}^{n} \text{Time}_i$$

- Performance is often expressed as a rate
  - e.g. millions of instructions per second
  - inverse of time
- Use harmonic mean — inverse of (average of inverses)

$$\text{Rate}_{\text{mean}} = \frac{n}{\sum_{i=1}^{n} \frac{1}{\text{Rate}_i}}$$

Weighted Means

- If different programs run with different frequencies
  - weight each component with its relative frequency
- Weighted arithmetic mean

$$\text{Time}_{\text{ave}} = \sum_{i=1}^{n} (\text{Weight}_i \times \text{Time}_i)$$

- Weighted harmonic mean

$$\text{Rate}_{\text{mean}} = \frac{1}{\sum_{i=1}^{n} \frac{\text{Weight}_i}{\text{Rate}_i}}$$

Combining Relative Ratios

- Approach used by SPEC
  - normalised results
    - » for each program in the suite, calculate time ratio w.r.t. reference
  - use geometric mean to combine ratios

$$\text{Ratio}_{\text{mean}} = \sqrt[n]{\prod_{i=1}^{n} \text{Ratio}_i}$$

Comparison

- Equal-time Weighted arithmetic mean can be influenced
  - by the peculiarity of the machine and the size of program input
- Geometric mean of normalized time is independent of them
  - Relative to referenced machine for the same program on the same input

Comparison (cont.)

- Geometric mean rewards relative improvement regardless the size of the program
  - Improvement from 2 sec to 1 sec == improvement from 2000 sec to 1000 sec
- Geometric mean cannot predict actual performance

Quantitative Principle of Computer Design

- Make The Common Case Fast
  - Make frequent cases simpler, faster and use less resources
  - Improving frequent cases has greatest impact on overall performance
- Examples:
  - in ALU, most operations don't overflow
    - » make non-overflowing operations faster, even if overflow case slows down
  - exception handling in Java

Amdahl's Law

- Law of diminishing returns
- Overall effect of an enhancement is weighted by proportion of time that the enhancement is used

Amdahl's Law Quantified

- Speedup is ratio of execution times:

$$S_{overall} = \frac{T_{old}}{T_{enh}}$$

- Let F_{enh} be fraction of **original execution time** that enhancement is used

$$T_{enh} = T_{old} \times \left((1 - F_{enh}) + \frac{F_{enh}}{S_{enh}} \right)$$

$$S_{overall} = \frac{1}{(1 - F_{enh}) + \frac{F_{enh}}{S_{enh}}}$$

Amdahl's Law Example

- Suppose
  - we can modify branch instructions to take half as long
  - measurements show branches account for 10% of execution time
- F_{enh} = 0.1, S_{enh} = 2, so

$$S_{overall} = \frac{1}{(1-0.1)+\frac{0.1}{2}} = \frac{1}{0.9+0.05} \cong 1.05$$

- Thus improvement is only 5%
  - if enhancement costs more than 5% extra, is it worth it?

Clocks, Cycles, etc.

- What does 2GHz mean?
  - clock frequency
  - clock signal used to synchronize operation of the processor
- CPU time = number of cycles for a program x cycle time
- Instruction count =
  number of instructions executed in the program
- Average cycles per instruction (CPI) =
  cycle count / instruction count

$$CPU\ \ Time = IC \times CPI \times T_c$$

- Parameters are interrelated:
  - cycle time depends on hardware technology
  - IC depends on instruction set and compiler
  - CPI depends on CPU organisation and instruction set

CPU Performance Model

- If we have *n* instruction classes, each taking different number of cycles
  - IC_i = instruction count for class *i*
  - CPI_i = CPI for class *i*

$$CPU\ \ Time = \sum_{i=1}^{n} (IC_i \times CPI_i) \times T_c$$

$$CPI = \frac{\sum_{i=1}^{n} (IC_i \times CPI_i)}{IC} = \sum_{i=1}^{n} \left(\frac{IC_i}{IC} \times CPI_i \right)$$

Example

- CPU A
  - compare to set the condition code (20%)
  - conditional branch based on the condition code (20%)
- CPU B
  - compare is included in the conditional branch (20%)
  - Cycle time is 25% slower than in CPU A.
- The conditional branch takes 2 cycles. All other instructions take one cycle.

Example (cont.)

- NIA = # of instructions on A
- CTA = cycle time of A
- CPU time A = 0.8 * NIA * 1 * CTA +
  0.2 * NIA * 2 * CTA
  = 1.2 * NIA * CTA
- CPU time B = 0.6 * NIA * 1 * 1.25*CTA +
  0.2 * NIA * 2 * 1.25*CTA
  = 1.25 * NIA * CTA