CMSC 411
Computer Systems Architecture
Lecture 2
Computer Design and Evaluation

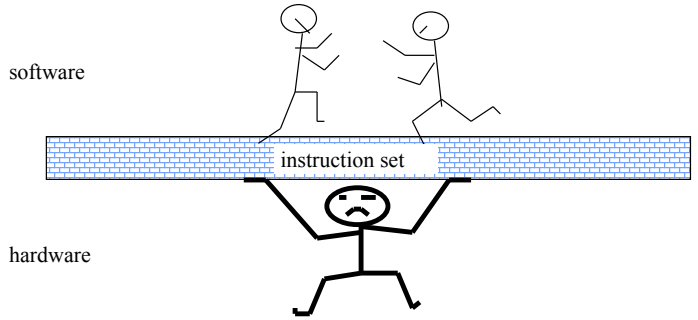
Alan Sussman
als@cs.umd.edu

Administrivia

- **Unit 1 Homework posted on homework/project web page Tuesday**
  - **as stated on syllabus, 2 problems will be graded**
- **Continue reading Ch. 1 of H&P**

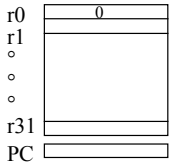
COMPUTER ARCHITECTURE VS. INSTRUCTION SET ARCHITECTURE

Instruction Set Architecture: Critical Interface



- **Properties of a good abstraction**
  - **Lasts through many generations (portability)**
  - **Used in many different ways (generality)**
  - **Provides convenient functionality to higher levels**
  - **Permits an efficient implementation at lower levels**

Example: MIPS



Programmable storage
- 2^{32} bytes
- 31 x 32-bit GPRs (R0=0)
- 32 x 32-bit FP regs (paired DP)
- PC

Data types ?

Format ?

Addressing Modes?

Arithmetic logical
Add, AddU, Sub, SubU, And, Or, Xor, SLT, SLTU, AddI, AddIU, SLTI, SLTIU, AndI, OrI, XorI, SLL, SRL, SRA

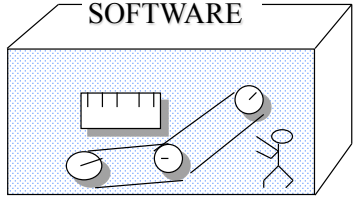
Memory Access
LB, LBU, LH, LHU, LW, SB, SH, SW

Control
J, JAL, JR, JALR
BEq, BNE, BLEZ,BGTZ,BLTZ,BGEZ

32-bit instructions on word boundary

Instruction Set Architecture

"... the attributes of a [computing] system as seen by the programmer, *i.e.* the conceptual structure and functional behavior, as distinct from the organization of the data flows and controls the logic design, and the physical implementation."
– Amdahl, Blaauw, and Brooks, 1964



-- Organization of Programmable Storage

-- Data Types & Data Structures: Encodings & Representations

-- Instruction Formats

-- Instruction (or Operation Code) Set

-- Modes of Addressing and Accessing Data Items and Instructions

-- Exceptional Conditions

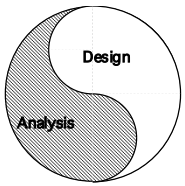
ISA vs. Computer Architecture

- **Old definition of computer architecture = instruction set design**
  - **Other aspects of computer design called implementation**
  - **Insinuates implementation is uninteresting or less challenging**
- **H&P's view is computer architecture >> ISA**
- **Architect's job much more than instruction set design; technical hurdles today *more* challenging than those in instruction set design**
- **Since instruction set design not where action is, some conclude computer architecture (using old definition) is not where action is**
  - **H&P disagree on conclusion**
  - **Agree that ISA not where action is (ISA in H&P 5/e appendix)**

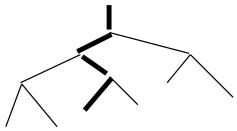
Comp. Arch. is an Integrated Approach

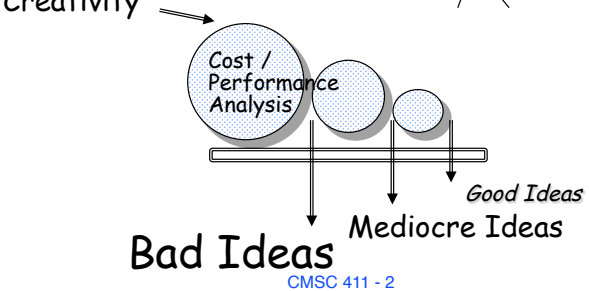
- **What really matters is the functioning of the complete system**
  - **hardware, runtime system, compiler, operating system, and application**
  - **In networking, this is called the "End to End argument"**
- **Computer architecture is not just about transistors, individual instructions, or particular implementations**
  - **E.g., Original RISC projects replaced complex instructions with a compiler + simple instructions**

Computer Architecture is Design and Analysis



Architecture is an iterative process:
- Searching the space of possible designs
- At all levels of computer systems





MANUFACTURING COSTS

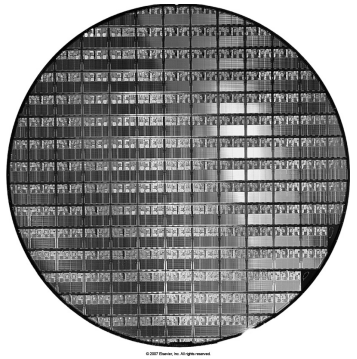
Costs

- **From Figure 1.9 in H&P 3/e**
- **The cost of components in a $1000 PC in 2001 are:**
  - **CPU – 22%**
  - **Monitor – 19%**
  - **Hard drive – only 9%**
  - **DRAM – only 5% (for 128MB)**
  - **Software – 20% (OS & basic office suite)**

Manufacture of DRAM and other chips

- **Chips are manufactured on wafers - circular disks containing many dies (chips).**
- **The wafer is tested and chopped into dies.**

117 AMD Opterons



Wafers and dies

- To find the cost of a die:
  - Number of dies per wafer is *at most* the area of the wafer divided by the area of the die.
  - The cost of the wafer divided by the number of working dies per wafer is the cost of each die.
- The fraction of working dies is called the *die yield*, which decreases as the area of the die increases.
- Rule of thumb (p. 32): Cost of die is proportional to the square of the die area

PERFORMANCE MEASUREMENT

Comparing performance of two machines

- Definition: Performance is equal to 1 divided by execution time
- Problem: How to measure execution time?

What is time?

- Unix **time** command example:
  - 90.7u 12.9s 2:39 65%
  - The user used the CPU for 90.7 seconds (user CPU time)
  - The system used it for 12.9 seconds (system CPU time)
  - Elapsed time from the user's request to completion of the task was 2 minutes, 39 seconds (159 seconds)
  - And (90.7 + 12.9)/159 = 65%
    - » the rest of the time was spent waiting for I/O or running other programs

Time (cont.)

- Usual measurements of time:
  - system performance measures the elapsed time on unloaded (single user) system
  - CPU performance measures user CPU time on unloaded system

How to measure CPU performance

- Benchmark: a program used to measure performance
  - real programs - what is reality?
  - kernels - loops in which most of time is spent in a real program
  - toy programs
  - synthetic programs
- Fact: Computer manufacturers tune their product to the popular benchmarks
  - ``Your results may vary," unless you run benchmark programs and nothing else
  - See Figure 1.16 listing programs in the SPEC CPU2006 benchmark suite

Reproducibility

- Benchmarking is a laboratory experiment, and needs to be documented as fully as a well-run chemistry experiment
  - Identify each variable hardware component
  - Identify compiler flags and measure variability
  - Verify reproducibility and provide data for others to reproduce the benchmarking results

Reporting results

- Example of performance for SPEC CFP2000 benchmark is in H&P Figure 1.17
  - for Sun Ultra5, AMD Opteron, Intel Itanium 2
  - need to look on SPEC web site for all the parameters of the machines, including
    - » data on hardware - CPU (how many, primary and secondary caches), memory, disk
    - » data on software – OS, compilers, file system type, and compiler flags used (very important!)
- How to summarize results?

Sample results

	Computer A	Computer B	Computer C
Program P1 (sec)	1	10	20
Program P2 (sec)	1000	100	20
Program P3 (sec)	1001	110	40

Statistical reporting

- "It is rare indeed when advertisers, politicians, pop economists, and drumbeaters for medical programs offer a statistical argument that is not either misleading or downright deceptive." - Martin Gardner
- "... in mathematics you don't understand things, you just get used to them." - John von Neumann
- "... if you torture your data long enough, they will tell you what you want to hear." - James L. Mills, M.D.

Statistical reporting (cont.)

- The average execution time is the arithmetic mean:
  - sum the execution times for each program and divide by the number of programs *n*
- The harmonic mean is *n* divided by the sum of some function of each execution time
- Often, both of these means are modified to give more weight to more important programs

Statistical reporting (cont.)

- Arithmetic mean is good if the weights represent your own workload - then you can compare how each machine will perform for you
  - *Don't* set the weights based on performance on any particular machine; can get confusing results by doing that
- Harmonic mean gives same sort of information, but is used when rates (e.g., instructions per second) are measured instead of times

Geometric mean

- To measure relative performance, use the geometric mean
  - Choose a *reference machine*, divide all execution times by the corresponding times on the reference machine, multiply those ratios together, and take the *n*th root of the product
- Geometric means have the nice property that you get consistent results (relative performance) regardless of which machine is used to normalize (Figure 1.17)

How to make computers faster

- Make the common case faster!
- Example: Put more effort and funds into optimizing the hardware for addition than to optimize square root
- Amdahl's law quantifies this principle:
  - Define speedup as the time the task took originally divided by the time the task takes after improvement

Amdahl's Law

- Then Amdahl's Law tells us what the speedup of a particular task is, given
  - fraction *f* of the original execution time that the task could use the improvement
  - the speedup *s* of a task that always uses the improvement
- Then what is the speedup of the task?

Amdahl's Law (cont.)

- Suppose that the original task runs for 1 second so it takes *f* seconds in the critical piece, and *1-f* in other things
- Then the task on the improved machine will take only *f/s* seconds in the critical piece, but will still take *1-f* seconds in other things
- speedup = $\frac{old\_time}{new\_time} = \frac{1}{(1-f) + \frac{f}{s}}$