EECS 252 Graduate Computer Architecture
Lecture 1

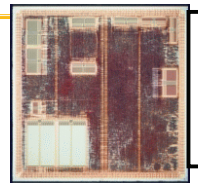
Introduction
January 20th 2010

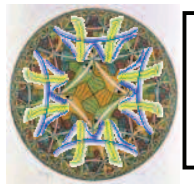
John Kubiatowicz
Electrical Engineering and Computer Sciences
University of California, Berkeley

http://www.eecs.berkeley.edu/~kubitron/cs252

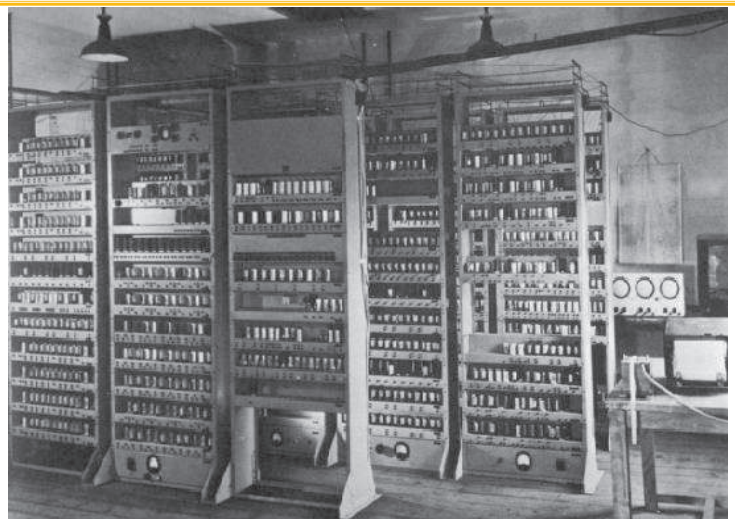
Who am I?

- Professor John Kubiatowicz (Prof "Kubi")
  - Background in Hardware Design
    - » Alewife project at MIT
    - » Designed CMMU, Modified SPAR C processor
    - » Helped to write operating system
  - Background in Operating Systems
    - » Worked for Project Athena (MIT)
    - » OS Developer (device drivers, network file systems)
    - » Worked on Clustered High-Availability systems (CLAM Associates)
    - » OS lead researcher for the new Berkeley PARLab (Tessellation OS). More later.
  - Peer-to-Peer
    - » OceanStore project – Store your data for 1000 years
    - » Tapestry and Bamboo – Find you data around globe
  - Quantum Computing
    - » Well, this is just cool, but probably not apropos





Computing Devices Then…

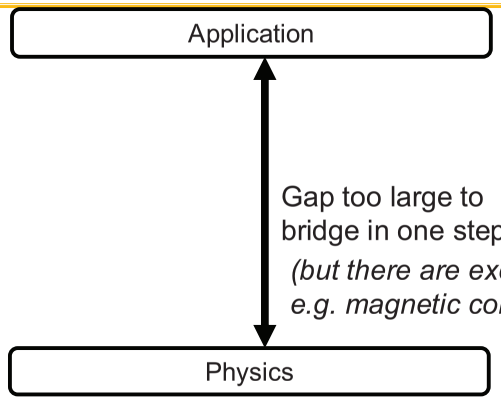


EDSAC, University of Cambridge, UK, 1949

Computing Systems Today

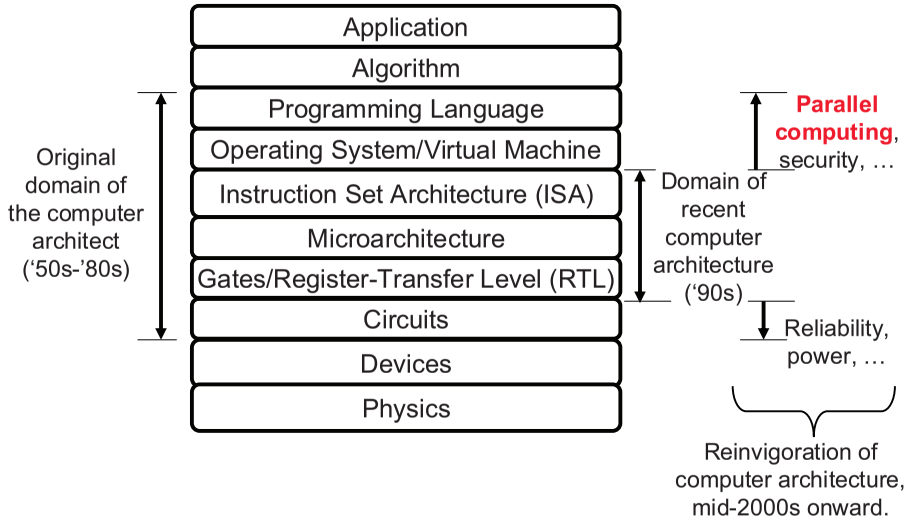
- The world is a large parallel system
  - Microprocessors in everything
  - Vast infrastructure behind them

What is Computer Architecture?



In its broadest definition, computer architecture is the *design of the abstraction layers* that allow us to implement information processing applications efficiently using available manufacturing technologies.

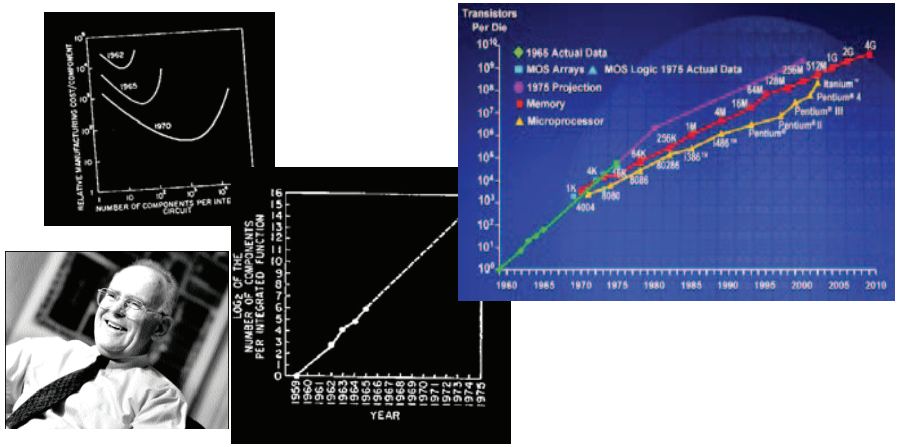
Abstraction Layers in Modern Systems



Computer Architecture's Changing Definition

- **1950s to 1960s: Computer Architecture Course: Computer Arithmetic**
- **1970s to mid 1980s: Computer Architecture Course: Instruction Set Design, especially ISA appropriate for compilers**
- **1990s: Computer Architecture Course: Design of CPU, memory system, I/O system, Multiprocessors, Networks**
- **2000s: Multi-core design, on-chip networking, parallel programming paradigms, power reduction**
- **2010s: Computer Architecture Course: Self adapting systems? Self organizing structures? DNA Systems/Quantum Computing?**

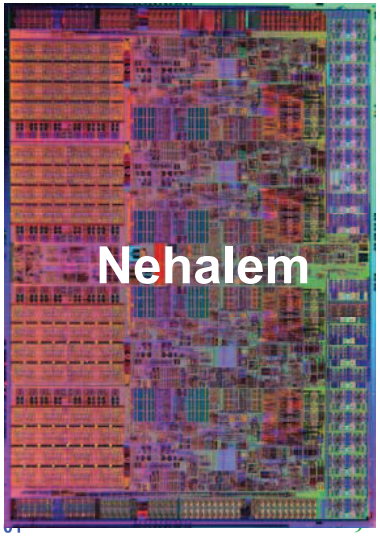
Moore's Law



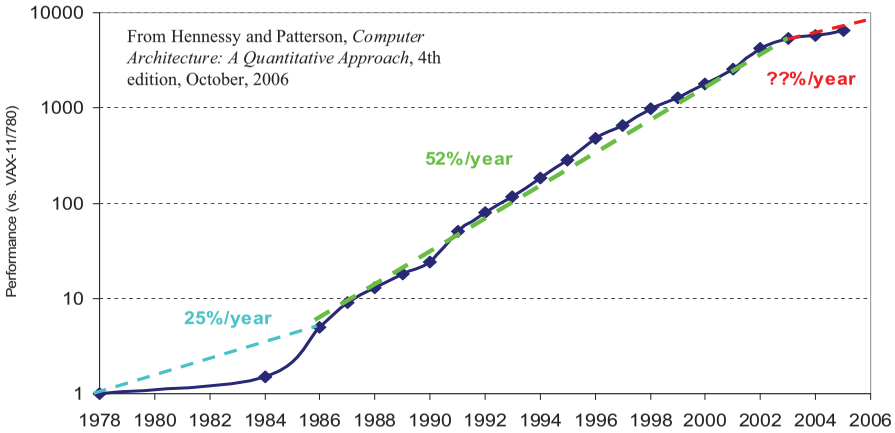
- **"Cramming More Components onto Integrated Circuits"**
  - **Gordon Moore, Electronics, 1965**
- **# on transistors on cost-effective integrated circuit double every 18 months**

Technology constantly on the move!

- Num of transistors not limiting factor
  - Currently ~ 1 billion transistors/chip
  - Problems:
    - » Too much Power, Heat, Latency
    - » Not enough Parallelism
- 3-dimensional chip technology?
  - Sandwiches of silicon
  - "Through-Vias" for communication
- On-chip optical connections?
  - Power savings for large packets
- The Intel® Core™ i7 microprocessor ("Nehalem")
  - 4 cores/chip
  - 45 nm, Hafnium hi-k dielectric
  - 731M Transistors
  - Shared L3 Cache - 8MB
  - L2 Cache - 1MB (256K x 4)



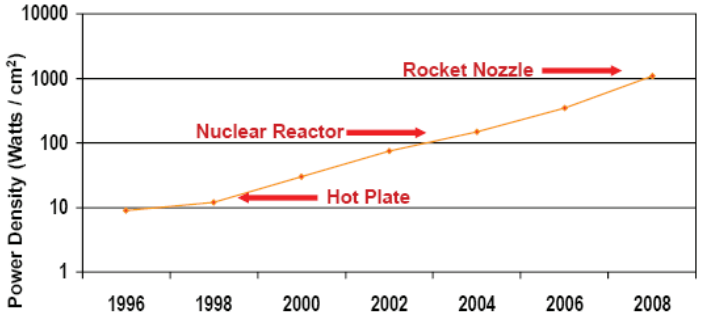
Crossroads: Uniprocessor Performance



- VAX : 25%/year 1978 to 1986
- RISC + x86: 52%/year 1986 to 2002
- RISC + x86: ??%/year 2002 to present

Limiting Force: Power Density

Moore's Law Extrapolation:
Power Density for Leading Edge Microprocessors



Power Density Becomes Too High to Cool Chips Inexpensively

Source: Shekhar Borkar, Intel Corp

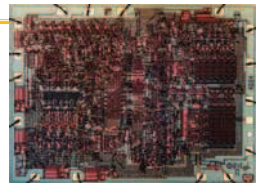
Crossroads: Conventional Wisdom in Comp. Arch

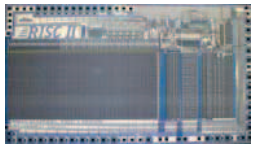
- Old Conventional Wisdom: Power is free, Transistors expensive
- New Conventional Wisdom: "Power wall" Power expensive, Xtors free (Can put more on chip than can afford to turn on)
- Old CW: Sufficiently increasing Instruction Level Parallelism via compilers, innovation (Out-of-order, speculation, VLIW, …)
- New CW: "ILP wall" law of diminishing returns on more HW for ILP
- Old CW: Multiplies are slow, Memory access is fast
- New CW: "Memory wall" Memory slow, multiplies fast (200 clock cycles to DRAM memory, 4 clocks for multiply)
- Old CW: Uniprocessor performance 2X / 1.5 yrs
- New CW: Power Wall + ILP Wall + Memory Wall = Brick Wall
  - Uniprocessor performance now 2X / 5(?) yrs

⇒ Sea change in chip design: multiple "cores" (2X processors per chip / ~ 2 years)

- » More power efficient to use a large number of simpler processors tather than a small number of complex processors

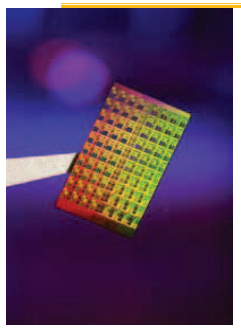
Sea Change in Chip Design





- Intel 4004 (1971):
  - 4-bit processor,
  - 2312 transistors, 0.4 MHz,
  - 10 μm PMOS, 11 mm² chip
- RISC II (1983):
  - 32-bit, 5 stage
  - pipeline, 40,760 transistors, 3 MHz,
  - 3 μm NMOS, 60 mm² chip
- 125 mm² chip, 65 nm CMOS = 2312 RISC II+FPU+Icache+Dcache
  - RISC II shrinks to ~ 0.02 mm² at 65 nm
  - Caches via DRAM or 1 transistor SRAM (www.t-ram.com) ?
  - Proximity Communication via capacitive coupling at > 1 TB/s ? (Ivan Sutherland @ Sun / Berkeley)
- **Processor is the new transistor?**

ManyCore Chips: The future is here!



- Intel 80-core multicore chip (Feb 2007)
  - 80 simple cores
  - Two floating point engines /core
  - Mesh-like "network-on-a-chip"
  - 100 million transistors
  - 65nm feature size

Frequency	Voltage	Power	Bandwidth	Performance
3.16 GHz	0.95 V	62W	1.62 Terabits/s	1.01 Teraflops
5.1 GHz	1.2 V	175W	2.61 Terabits/s	1.63 Teraflops
5.7 GHz	1.35 V	265W	2.92 Terabits/s	1.81 Teraflops

- "ManyCore" refers to many processors/chip
  - 64? 128? Hard to say exact boundary
- How to program these?
  - Use 2 CPUs for video/audio
  - Use 1 for word processor, 1 for browser
  - 76 for virus checking???
- Something new is clearly needed here…

The End of the Uniprocessor Era

Single biggest change in the history of computing systems

Déjà vu all over again?

- Multiprocessors imminent in 1970s, '80s, '90s, …
- "… today's processors … are nearing an impasse as technologies approach the speed of light.."

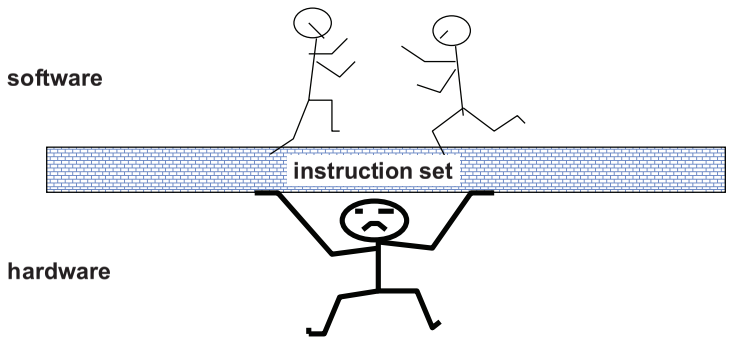
  David Mitchell, *The Transputer: The Time Is Now* (1989)
- Transputer was premature
  ⇒ Custom multiprocessors strove to lead uniprocessors
  ⇒ Procrastination rewarded: 2X seq. perf. / 1.5 years
- "We are dedicating all of our future product development to multicore designs. … This is a sea change in computing"

  Paul Otellini, President, Intel (2004)
- Difference is all microprocessor companies switch to multicore (AMD, Intel, IBM, Sun; all new Apples 2-4 CPUs)
  ⇒ Procrastination penalized: 2X sequential perf. / 5 yrs
  ⇒ Biggest programming challenge: 1 to 2 CPUs

Problems with Sea Change

- Algorithms, Programming Languages, Compilers, Operating Systems, Architectures, Libraries, … not ready to supply Thread Level Parallelism or Data Level Parallelism for 1000 CPUs / chip
  - Need whole new approach
  - People have been working on parallelism for over 50 years without general success
- Architectures not ready for 1000 CPUs / chip
  - Unlike Instruction Level Parallelism, cannot be solved by just by computer architects and compiler writers alone, but also cannot be solved *without* participation of computer architects
- PARLab: Berkeley researchers from many backgrounds meeting since 2005 to discuss parallelism
  - Krste Asanovic, Ras Bodik, Jim Demmel, Kurt Keutzer, John Kubiatowicz, Edward Lee, George Necula, Dave Patterson, Koushik Sen, John Shalf, John Wawrzynek, Kathy Yelick, …
  - Circuit design, computer architecture, massively parallel computing, computer-aided design, embedded hardware and software, programming languages, compilers, scientific programming, and numerical analysis

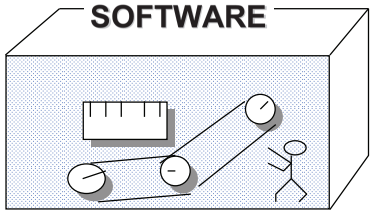
The Instruction Set: a Critical Interface



- 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

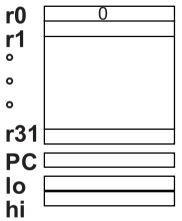
Instruction Set Architecture

... 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, Blaaw, 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

Example: MIPS R3000



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

Data types ?

Format ?

Addressing Modes?

Arithmetic logical

Add, AddU, Sub, SubU, And, Or, Xor, Nor, SLT, SLTU,
Addl, AddlU, SLTI, SLTIU, Andl, Orl, Xorl, *LUI*
SLL, SRL, SRA, SLLV, SRLV, SRAV

Memory Access

LB, LBU, LH, LHU, LW, LWL,LWR
SB, SH, SW, SWL, SWR

Control

32-bit instructions on word boundary

J, JAL, JR, JALR
BEq, BNE, BLEZ,BGTZ,BLTZ,BGEZ,BLTZAL,BGEZAL

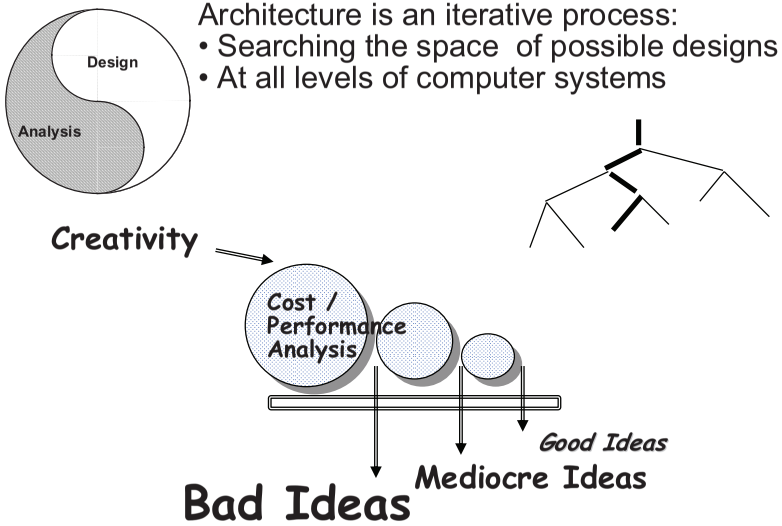
ISA vs. Computer Architecture

- Old definition of computer architecture = instruction set design
  - Other aspects of computer design called implementation
  - Insinuates implementation is uninteresting or less challenging
- Our 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
  - We disagree on conclusion
  - Agree that ISA not where action is (ISA in CA:AQA 4/e appendix)

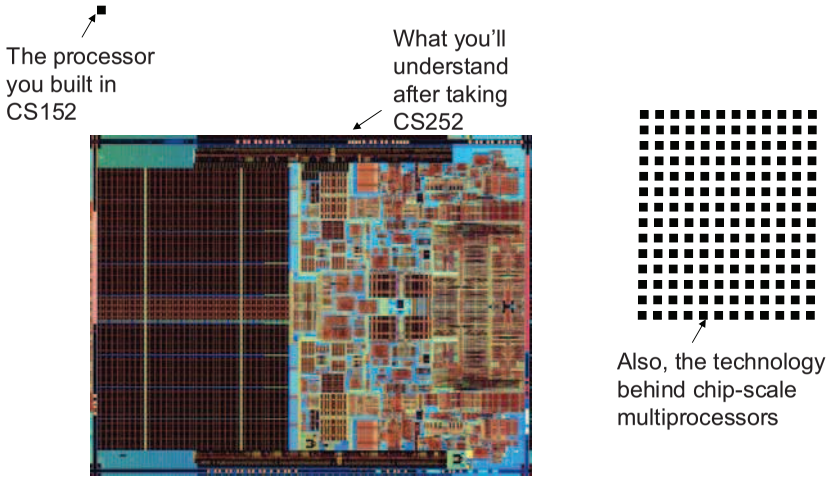
Computer Architecture is an Integrated Approach

- 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
- It is very important to think across all hardware/software boundaries
  - New technology ⇒ New Capabilities ⇒ New Architectures ⇒ New Tradeoffs
  - Delicate balance between backward compatibility and efficiency

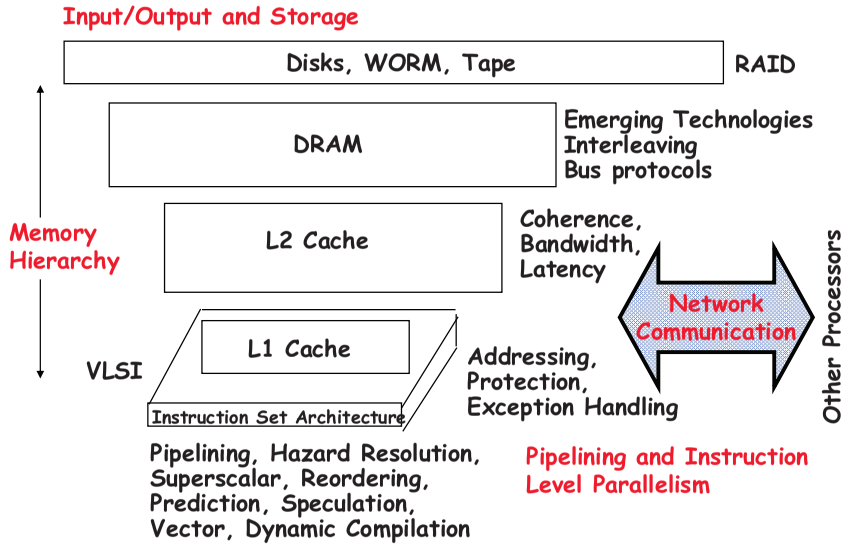
Computer Architecture is Design and Analysis



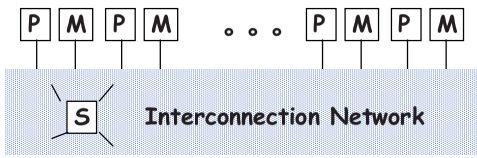
CS252 Executive Summary



Computer Architecture Topics



Computer Architecture Topics



Processor-Memory-Switch

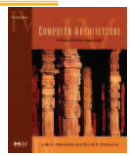
Multiprocessors
Networks and Interconnections

Shared Memory,
Message Passing,
Data Parallelism

Network Interfaces

Topologies,
Routing,
Bandwidth,
Latency,
Reliability

Tentative Topics Coverage



Textbook: Hennessy and Patterson, *Computer Architecture: A Quantitative Approach*, 4th Ed., 2006

Research Papers -- Handed out in class

- 1.5 weeks Review: Fundamentals of Computer Architecture, Instruction Set Architecture, Pipelining
- 2.5 weeks: Pipelining, Interrupts, and Instructional Level Parallelism, Vector Processors
- 1 week: Memory Hierarchy
- 1.5 weeks: Networks and Interconnection Technology
- 1 week: Parallel Models of Computation
- 1 week: Message-Passing Interfaces
- 1 week: Shared Memory Hardware
- 1.5 weeks: Multithreading, Latency Tolerance, GPU
- 1.5 weeks: Fault Tolerance, Input/Output and Storage
- 0.5 weeks: Quantum Computing, DNA Computing

CS252: Information

Instructor: Prof John D. Kubiatowicz
Office: 673 Soda Hall, 643-6817 kubitron@cs
Office Hours: Mon 2:30-4:00 or by appt.

T. A: No TA this term!

Class: Mon/Wed,1:00-2:30pm, 310 Soda Hall

Text: Computer Architecture: A Quantitative Approach, Fourth Edition (2004)

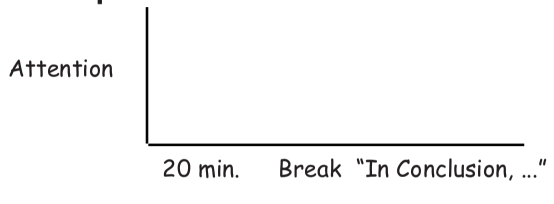
Web page: http://www.cs/~kubitron/cs252/
Lectures available online <11:30AM day of lecture

Newsgroup: ucb.class.cs252

Email: cs252@kubi.cs.berkeley.edu

Lecture style

- 1-Minute Review
- 20-Minute Lecture/Discussion
- 5- Minute Administrative Matters
- 25-Minute Lecture/Discussion
- 5-Minute Break (water, stretch)
- 25-Minute Lecture/Discussion
- Instructor will come to class early & stay after to answer questions



Research Paper Reading

- As graduate students, you are now researchers.
  - Most information of importance to you will be in research papers
  - Ability to scan and understand research papers is key to success
- So: you will read lots of papers in this course!
  - Quick 1 paragraph summaries will be due in class
  - Important supplement to book
  - Will discuss some of the papers in class
- Papers will be scanned and on web page
  - Will be available (hopefully) > 1 week in advance

Quizzes

- Reduce the pressure of taking quizes
  - Two (maybe one) Graded Quizes:
    Tentative: Wed March 17th and Wed May 5th
  - Our goal: test knowledge vs. speed writing
  - 3 hrs to take 1.5-hr test (5:30-8:30 PM, TBA location)
  - Both mid-term quizzes can bring summary sheet
    - » Transfer ideas from book to paper
  - Last chance Q&A: during class time day of exam
- Students/Staff meet over free pizza/drinks at La Vals: Wed March 17th (8:30 PM) and Wed May 5th (8:30 PM)

Research Project

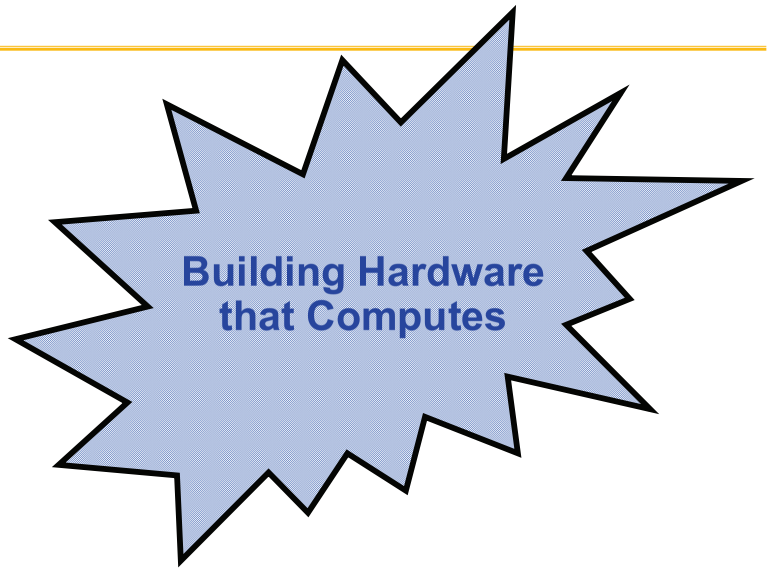
- Research-oriented course
  - Project provides opportunity to do "research in the small" to help make transition from good student to research colleague
  - Assumption is that you will advance the state of the art in some way
  - Projects done in groups of 2 or 3 students
- Topic?
  - Should be topical to CS252
  - Exciting possibilities related to the ParLAB research agenda
- Details:
  - meet 3 times with faculty/TA to see progress
  - give oral presentation
  - give poster session (possibly)
  - written report like conference paper
- Can you share a project with other systems projects?
  - Under most circumstances, the answer is "yes"
  - Need to ok with me, however

More Course Info

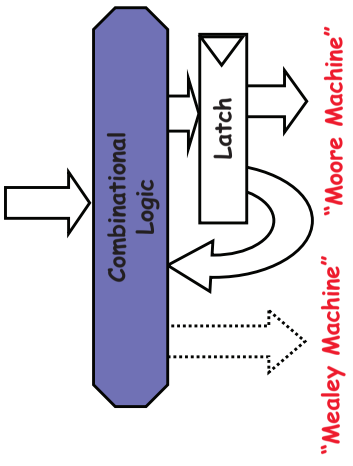
- Grading:
  - 10% Class Participation
  - 10% Reading Writups
  - 40% Examinations (2 Midterms)
  - 40% Research Project (work in pairs)
- Schedule:
  - 2 Graded Quizes: Wed March 17th and Wed May 5th
  - President's Day: February 15th
  - Spring Break: Monday March 22nd to March 26th
  - 252 Last lecture: Monday, April 28th
  - Oral Presentations: Wednesday May 10th?
  - 252 Poster Session: ???
  - Project Papers/URLs due: Monday May 13th
- Project Suggestions: TBA

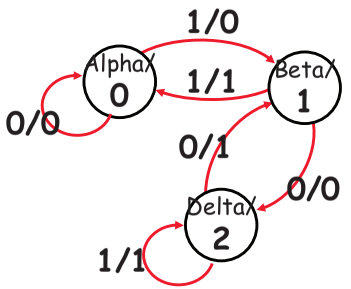
Coping with CS 252

- Undergrads must have taken CS152
- Grad Students with too varied background?
  - In past, CS grad students took written prelim exams on undergraduate material in hardware, software, and theory
  - 1st 5 weeks reviewed background, helped 252, 262, 270
  - Prelims were dropped => some unprepared for CS 252?
- Grads without CS152 equivalent may have to work hard; Review: Appendix A, B, C; CS 152 home page, maybe *Computer Organization and Design (COD) 3/e*
  - Chapters 1 to 8 of COD if never took prerequisite
  - If took a class, be sure COD Chapters 2, 6, 7 are familiar
  - I can loan you a copy
- Will spend 2 lectures on review of Pipelining and Memory Hierarchy



Finite State Machines: Implementation as Comb logic + Latch

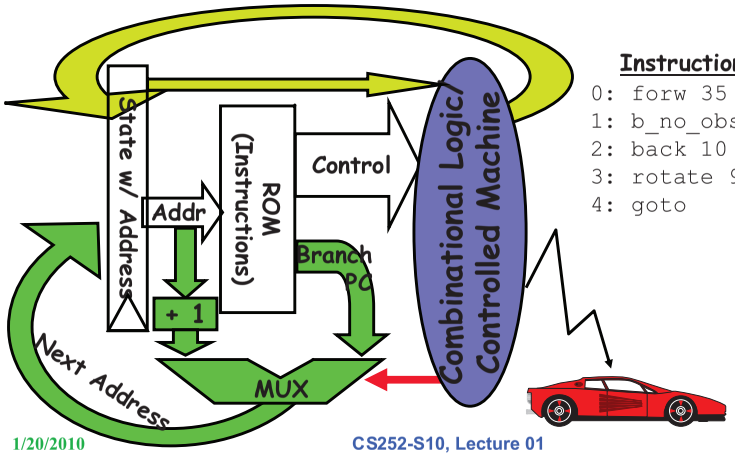




Input	$State_{old}$	$State_{new}$	Div
0	00	00	0
0	01	10	0
0	10	01	1
1	00	01	0
1	01	00	1
1	10	10	1

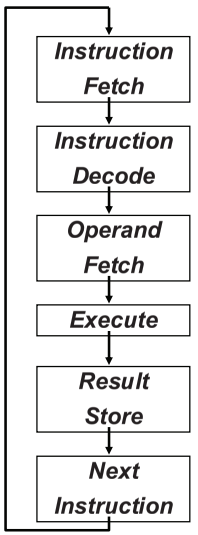
Microprogrammed Controllers

- State machine in which part of state is a "micro-pc".
  - Explicit circuitry for incrementing or changing PC
- Includes a ROM with "microinstructions".
  - Controlled logic implements at least branches and jumps

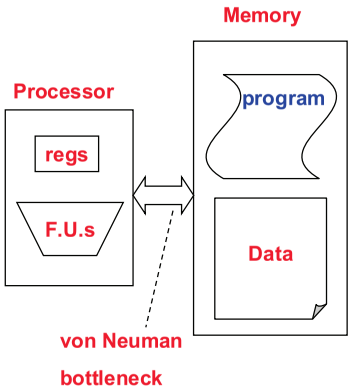


Instruction	Branch
0: forw 35	xxx
1: b_no_obstacles	000
2: back 10	xxx
3: rotate 90	xxx
4: goto	001

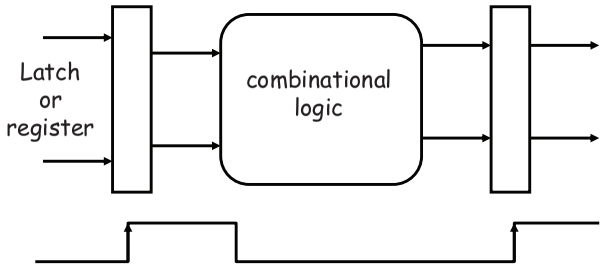
Fundamental Execution Cycle



- Obtain instruction from program storage
- Determine required actions and instruction size
- Locate and obtain operand data
- Compute result value or status
- Deposit results in storage for later use
- Determine successor instruction

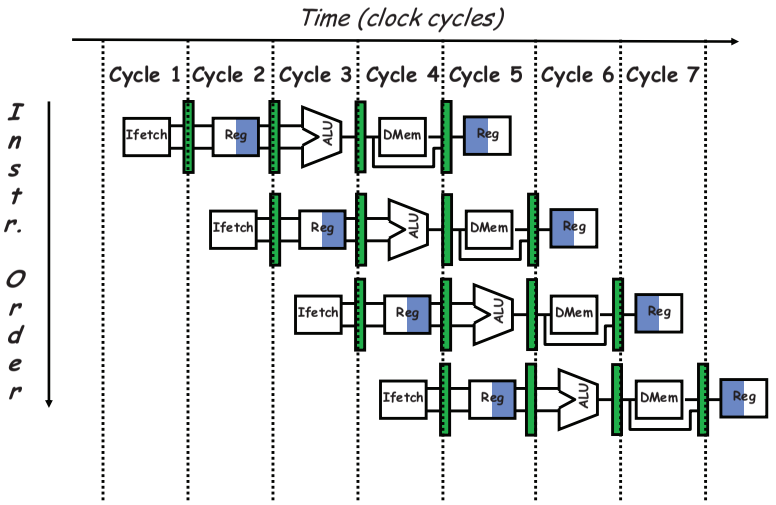


What's a Clock Cycle?



- Old days: 10 levels of gates
- Today: determined by numerous time-of-flight issues + gate delays
  - clock propagation, wire lengths, drivers

Pipelined Instruction Execution



Limits to pipelining

- Maintain the von Neumann "illusion" of one instruction at a time execution
- Hazards prevent next instruction from executing during its designated clock cycle
  - Structural hazards: attempt to use the same hardware to do two different things at once
  - Data hazards: Instruction depends on result of prior instruction still in the pipeline
  - Control hazards: Caused by delay between the fetching of instructions and decisions about changes in control flow (branches and jumps).
- Power: Too many thing happening at once ⇒ Melt your chip!
  - Must disable parts of the system that are not being used
  - Clock Gating, Asynchronous Design, Low Voltage Swings, …

Progression of ILP

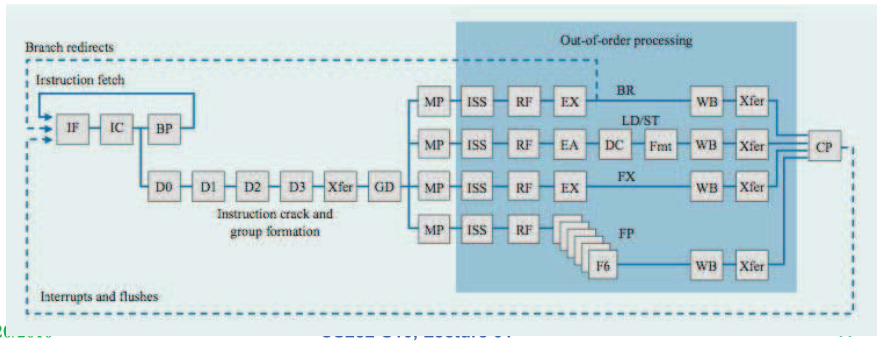
- 1st generation RISC - pipelined
  - Full 32-bit processor fit on a chip => issue almost 1 IPC
    - » Need to access memory 1+x times per cycle
  - Floating-Point unit on another chip
  - Cache controller a third, off-chip cache
  - 1 board per processor ➔ multiprocessor systems
- 2nd generation: superscalar
  - Processor and floating point unit on chip (and some cache)
  - Issuing only one instruction per cycle uses at most half
  - Fetch multiple instructions, issue couple
    - » Grows from 2 to 4 to 8 …
  - How to manage dependencies among all these instructions?
  - Where does the parallelism come from?
- VLIW
  - Expose some of the ILP to compiler, allow it to schedule instructions to reduce dependences

Modern ILP

- Dynamically scheduled, out-of-order execution
  - Current microprocessor 6-8 of instructions per cycle
  - Pipelines are 10s of cycles deep
    ⇒ many simultaneous instructions in execution at once
  - Unfortunately, hazards cause discarding of much work
- What happens:
  - Grab a bunch of instructions, determine all their dependences, eliminate dep's wherever possible, throw them all into the execution unit, let each one move forward as its dependences are resolved
  - Appears as if executed sequentially
  - On a trap or interrupt, capture the state of the machine between instructions perfectly
- Huge complexity
  - Complexity of many components scales as n^2 (issue width)
  - Power consumption big problem

IBM Power 4

- Combines: Superscalar and OOO
- Properties:
  - 8 execution units in out-of-order engine, each may issue an instruction each cycle.
  - In-order Instruction Fetch, Decode (compute dependencies)
  - Reordering for in-order commit



When all else fails - guess

- Programs make decisions as they go
  - Conditionals, loops, calls
  - Translate into branches and jumps (1 of 5 instructions)
- How do you determine what instructions for fetch when the ones before it haven't executed?
  - Branch prediction
  - Lot's of clever machine structures to predict future based on history
  - Machinery to back out of mis-predictions
- Execute all the possible branches
  - Likely to hit additional branches, perform stores

  ⇒speculative threads

  ⇒What can hardware do to make programming (with performance) easier?

Have we reached the end of ILP?

- Multiple processor easily fit on a chip
- Every major microprocessor vendor has gone to multithreaded cores
  - Thread: loci of control, execution context
  - Fetch instructions from multiple threads at once, throw them all into the execution unit
  - Intel: hyperthreading, Sun:
  - Concept has existed in high performance computing for 20 years (or is it 40? CDC6600)
- Vector processing
  - Each instruction processes many distinct data
  - Ex: MMX
- Raise the level of architecture – many processors per chip

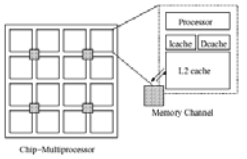
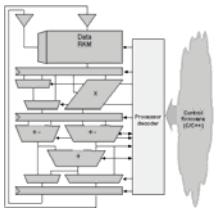


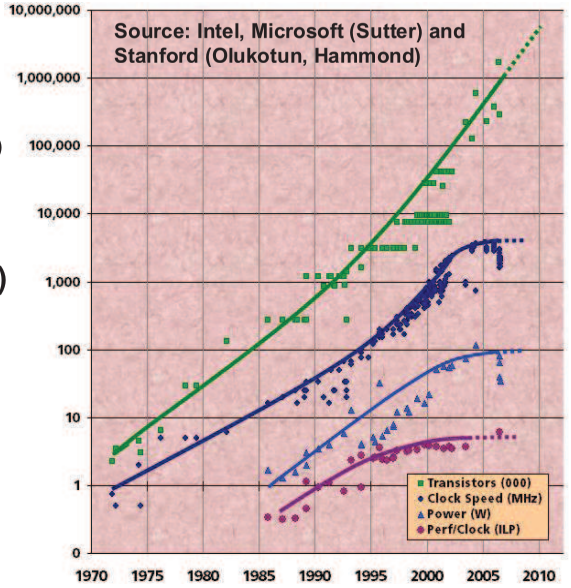
Figure 1. Chip-multiprocessor model.



Tensilica Configurable Proc

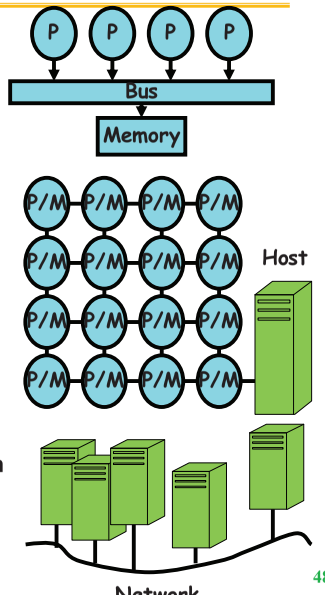
Limiting Forces: Clock Speed and ILP

- Chip density is continuing increase ~2x every 2 years
  - Clock speed is not
  - # processors/chip (cores) may double instead
- There is little or no more Instruction Level Parallelism (ILP) to be found
  - Can no longer allow programmer to think in terms of a serial programming model
- Conclusion: **Parallelism must be exposed to software!**

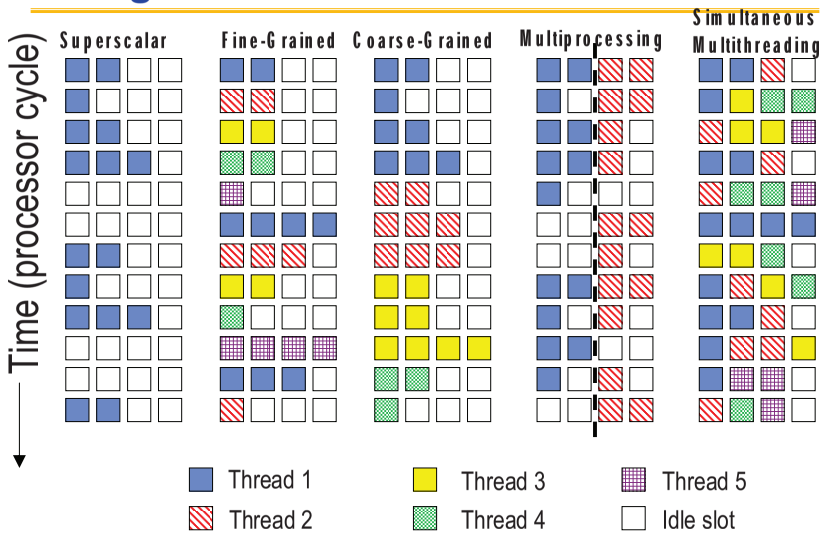


Examples of MIMD Machines

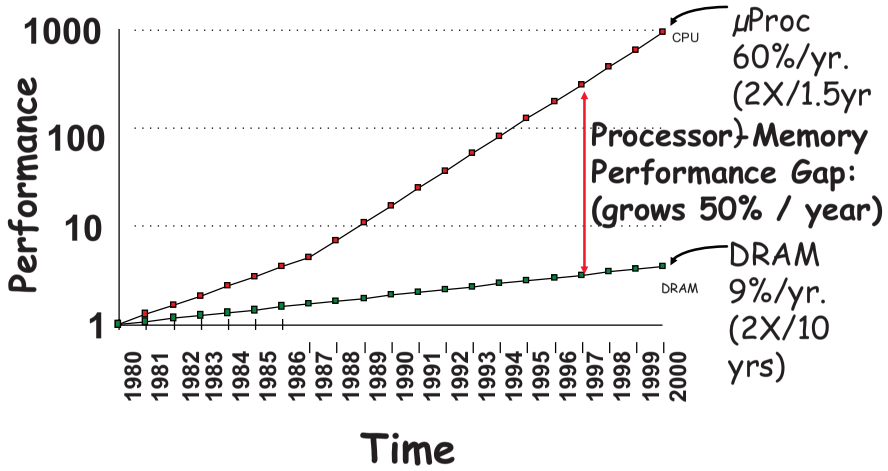
- Symmetric Multiprocessor
  - Multiple processors in box with shared memory communication
  - Current MultiCore chips like this
  - Every processor runs copy of OS
- Non-uniform shared-memory with separate I/O through host
  - Multiple processors
    - » Each with local memory
    - » general scalable network
  - Extremely light "OS" on node provides simple services
    - » Scheduling/synchronization
  - Network-accessible host for I/O
- Cluster
  - Many independent machine connected with general network
  - Communication through messages



Categories of Thread Execution

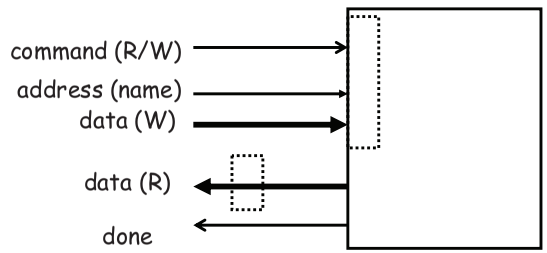


Processor-DRAM Memory Gap (latency)



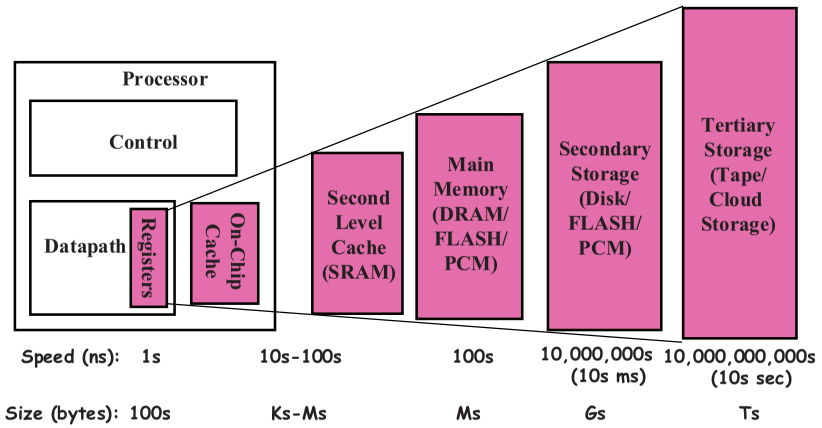
The Memory Abstraction

- Association of <name, value> pairs
  - typically named as byte addresses
  - often values aligned on multiples of size
- Sequence of Reads and Writes
- Write binds a value to an address
- Read of addr returns most recently written value bound to that address



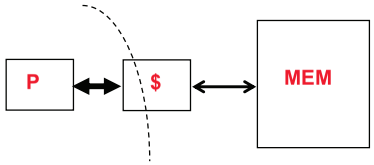
Memory Hierarchy

- Take advantage of the principle of locality to:
  - Present as much memory as in the cheapest technology
  - Provide access at speed offered by the fastest technology

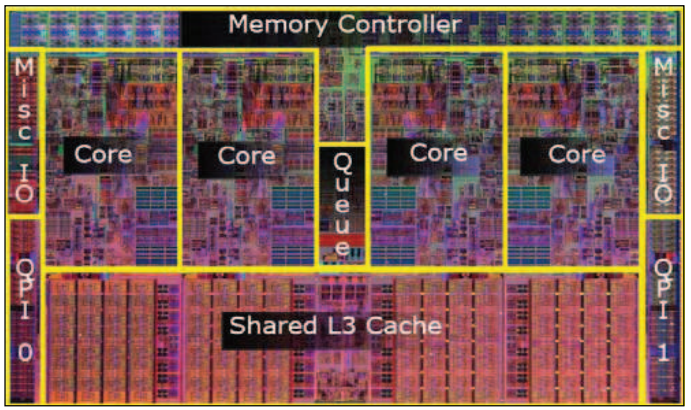


The Principle of Locality

- The Principle of Locality:
  - Program access a relatively small portion of the address space at any instant of time.
- Two Different Types of Locality:
  - Temporal Locality (Locality in Time): If an item is referenced, it will tend to be referenced again soon (e.g., loops, reuse)
  - Spatial Locality (Locality in Space): If an item is referenced, items whose addresses are close by tend to be referenced soon (e.g., straightline code, array access)
- Last 30 years, HW relied on locality for speed



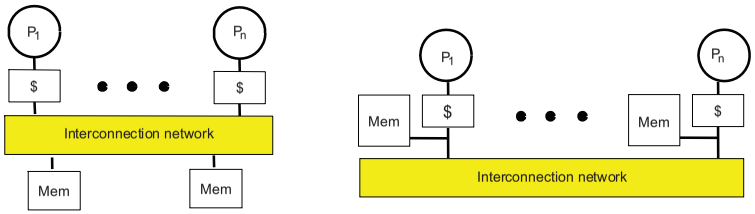
Example of modern core: Nehalem



- ON-chip cache resources:
  - For each core: L1: 32K instruction and 32K data cache, L2: 1MB
  - L3: 8MB shared among all 4 cores
- Integrated, on-chip memory controller (DDR3)

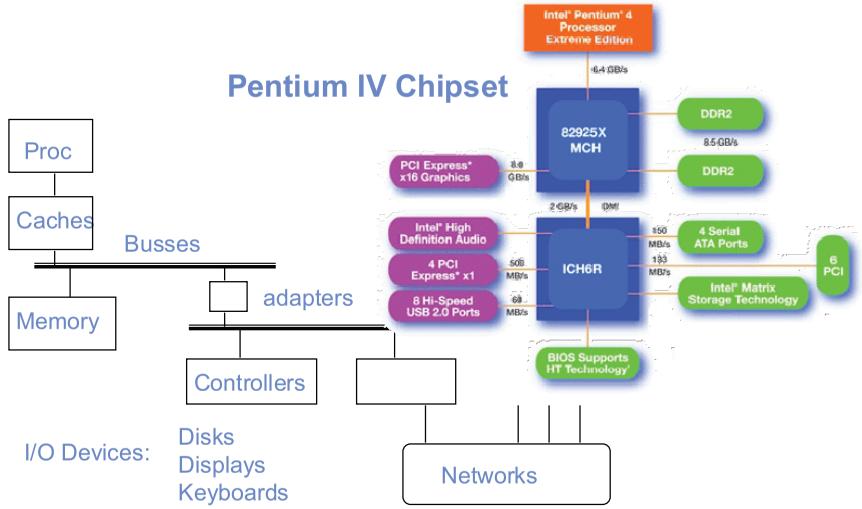
Memory Abstraction and Parallelism

- Maintaining the illusion of sequential access to memory across distributed system
- What happens when multiple processors access the same memory at once?
  - Do they see a consistent picture?



- Processing and processors embedded in the memory?

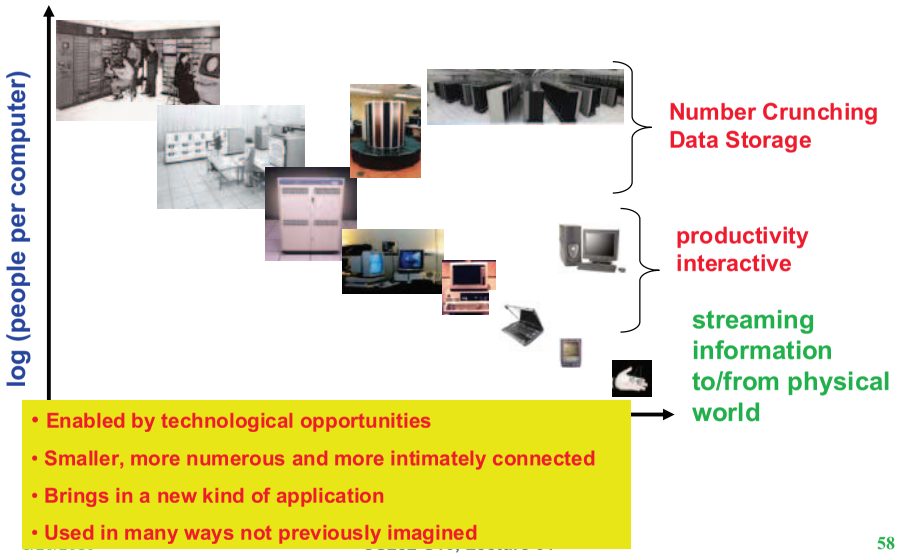
Is it all about communication?



Breaking the HW/Software Boundary

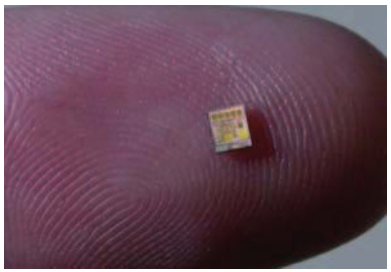
- Moore's law (more and more trans) is all about volume and regularity
- What if you could pour nano-acres of unspecific digital logic "stuff" onto silicon
  - Do anything with it. Very regular, large volume
- Field Programmable Gate Arrays
  - Chip is covered with logic blocks w/ FFs, RAM blocks, and interconnect
  - All three are "programmable" by setting configuration bits
  - These are huge?
- Can each program have its own instruction set?
- Do we compile the program entirely into hardware?

"Bell's Law" – new class per decade



It's not just about bigger and faster!

- Complete computing systems can be tiny and cheap
- System on a chip
- Resource efficiency
  - Real-estate, power, pins, …



And in conclusion …

- Computer Architecture >> instruction sets
- Computer Architecture skill sets are different
  - Quantitative approach to design
  - Solid interfaces that really work
  - Technology tracking and anticipation
- CS 252 to learn new skills, transition to research
- Computer Science at the crossroads from sequential to parallel computing
  - Salvation requires innovation in many fields, including computer architecture
- Read Appendix A, B, C of your book
- Next time: quick summary of everything you need to know to take this class