CIS 371
~~Digital Systems~~ Organization and Design
Computer

Unit 0: Introduction

Based on slides by Prof. Amir Roth & Prof. Milo Martin

Warmup Exercise

- Consider a binary tree
  - Left & right pointers
  - Integer value keys
  - Initialized to be fully balanced

```
while (node != NULL) {
   if (node->m_data == elem) {
      return node;
   } else if (node->m_data < elem) {
     node = node->m_right;
   } else {
     node = node->m_left;
   }
 }
```

- Question#1:
  - The average lookup time for tree of size 1024 (1K) is 50ns
  - What about for a a tree of size 1,048,576 (1M)?
- Question #2:
  - Pick an element in the tree and look it up repeatedly
  - What is the expected distribution of lookup times?
    - For a tree with height *h*
    - That is, what does the histogram of lookup times look like?

Today’s Agenda

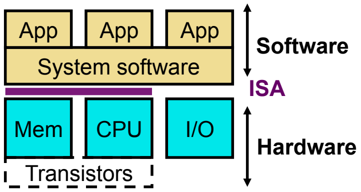
- Course overview and administrivia
- Motivational experiments
- What is computer architecture anyway?
  - …and the forces that drive it

Overview & Administration

Pervasive Idea: Abstraction and Layering

- **Abstraction**: only way of dealing with complex systems
  - Divide world into objects, each with an…
    - **Interface**: knobs, behaviors, knobs → behaviors
    - **Implementation**: "black box" (ignorance+apathy)
  - Only specialists deal with implementation, rest of us with interface
  - Example: car, only mechanics know how implementation works
- **Layering**: abstraction discipline makes life even simpler
  - Divide objects in system into layers, layer X objects…
    - Implemented using interfaces of layer X–1
    - Don't need to know interfaces of layer X–2 (sometimes helps)
- **Inertia**: a dark side of layering
  - Layer interfaces become entrenched over time ("standards")
  - – Very difficult to change even if benefit is clear (e.g., Feb 17, 2009)
- **Opacity**: hard to reason about performance across layers

Abstraction, Layering, and Computers



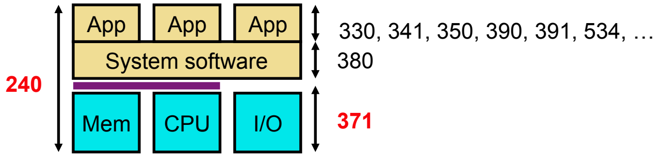
- Computers are complex, built in layers
  - Several **software** layers: assembler, compiler, OS, applications
  - **Instruction set architecture (ISA)**
  - Several **hardware** layers: transistors, gates, CPU/Memory/IO
- 99% of users don't know hardware layers implementation
- 90% of users don't know implementation of any layer
  - That's OK, world still works just fine
  - Someone needs to understand what's "under the hood"

CIS 240: Abstraction and Layering

- Build computer bottom up by raising level of abstraction

- Solid-state semi-conductor materials → transistors
- Transistors → gates
- Gates → digital logic elements: FFs, muxes, PLAs, **adders**
  - Key insight: number representation
- Logic elements → datapath + **control** = processor
  - Key insight: stored program (instructions just another form of data)
  - Another one: few insns can be combined to do anything (software)
- Assembly language → high-level language
- Code → graphical user interface

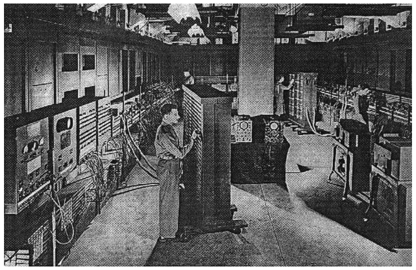
Beyond CIS 240



- CIS 240: Introduction to Computer Systems
  - Bottom-up overview of the entire hardware/software stack
  - Follow on courses look at individual pieces in more detail
- CIS 380: Operating Systems
  - A closer look at system level software
- CIS 277, 330, 341, 350, 390, 391, 455, 460, 461, 462…
  - A closer look at different important application domains
- **CIS 371: Computer Organization and Design**
  - **A closer look at hardware layers**

Why Study Hardware?

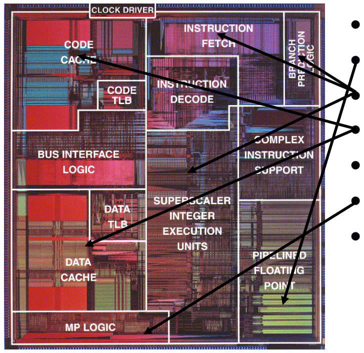
- It's required (translation: "it's good for you", we think)
- Real world impact
  - Without computer architecture there would be no computers
- Penn legacy
  - First "computer" (ENIAC) was built here
  - "computer" = general-purpose stored-program computer
- Get a hardware job
  - Intel, AMD/ATI, IBM, Sun/Oracle, NVIDIA, ARM, HP, TI, Samsung, Microsoft…
- Be better at a software job
  - Apple, Google, Microsoft, etc.
- Go to grad school



Hardware Aspect of CIS 240 vs. CIS 371

- Hardware aspect of CIS 240
  - Focus on one toy ISA: LC4
  - Focus on functionality: "just get something that works"
  - Instructive, learn to crawl before you can walk
  - Not representative of real machines: 240 hardware is circa 1975
- CIS 371
  - De-focus from any particular ISA
  - Focus on quantitative aspects: **performance**, cost, power, etc.
  - 371 hardware is circa 1995 (Pentium)

CIS 371 Topics



- Review of CIS 240 level hardware
  - Instruction set architecture
  - Single-cycle datapath and control
- New
  - Performance, cost and technology basis
  - Fast integer and floating-point
  - Pipelining and superscalar issue
  - Memory hierarchy and virtual memory
  - I/O
  - Multiprocessors and multithreading
  - Power, energy, reliability

The CIS371 Lab

- Lab
  - "Build your own processor" (pipelined 16-bit CPU for LC4)
  - Use Verilog HDL (hardware description language)
    - Programming language compiles to gates/wires not insns
  - Implement and test on FPGA (field-programmable gate array)

  + Instructive: learn by doing

  + Satisfying: "look, I built my own processor"
- Course evolution
  - Until last year "lab" was a 0.5 course unit additional course
  - Last year we rolled them into one course (1.5 -> 1.0 CUs)

Course Goals

- Three primary goals
  - Understand key hardware concepts
    - Pipelining, parallelism, caching, etc.
  - Hands-on design lab
  - A bit of scientific/experimental exposure and/or analysis
    - Not found too many other places in the major

- My role:
  - Trick you into learning something

CIS371 Administrivia

- Instructor
  - **Prof. Milo Martin** (milom@cis), Levine 606
- "Lecture" TAs
  - **Jim Anderson & Cem Karan**
- "Lab" TAs
  - TBD
- Contact e-mail:
  - **cis371@cis.upenn.edu** (goes to me and lecture TAs)
- Lectures
  - Please do not be disruptive (I'm easily distracted as it is)
- Information on assignments, labs, exams, grading
  - Forthcoming

Resources

- Textbook
  - *P+H, "Computer Organization and Design"*
    - **4th edition?** ($80 or $90!)
  - Penn Bookstore?
  - Course will largely be lecture note driven

- WWW: http://www.cis.upenn.edu/~cis371/
  - Lecture slides, homeworks, solutions, etc.
  - Keep an eye on http://…/schedule.html

- Newsgroup/forum?
  - Blackboard?

CIS 371 in Context

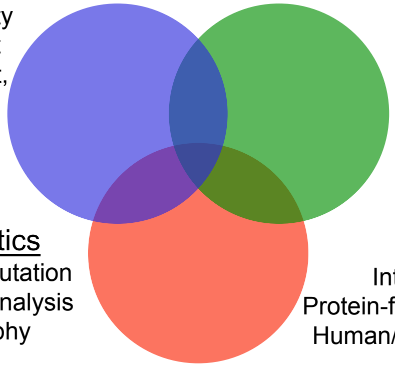
- Prerequisite: CIS 240
  - **Absolutely required as prerequisite**
  - Focused on "**function**"
  - Exposure to logic gates and assembly language programming

- The "lecture" component of the course:
  - Lecture material focuses on "**performance**"
  - Assignments will have some "**experimental evaluation**"

- The "lab" component of the course:
  - Focuses on "**design**"
  - Design a working processor

Computer Science as an Estuary

Where does CIS371/372 fit into computer science?
Engineering, some science



Engineering
Design
Handling complexity
Real-world impact
Examples: Internet,
microprocessor

Science
Experiments
Hypothesis
Examples:
Internet behavior,
Protein-folding supercomputer
Human/computer interaction

Mathematics
Limits of computation
Algorithms & analysis
Cryptography
Logic
Proofs of correctness

Other Issues
Public policy, ethics,
law, security

Experimental Motivation

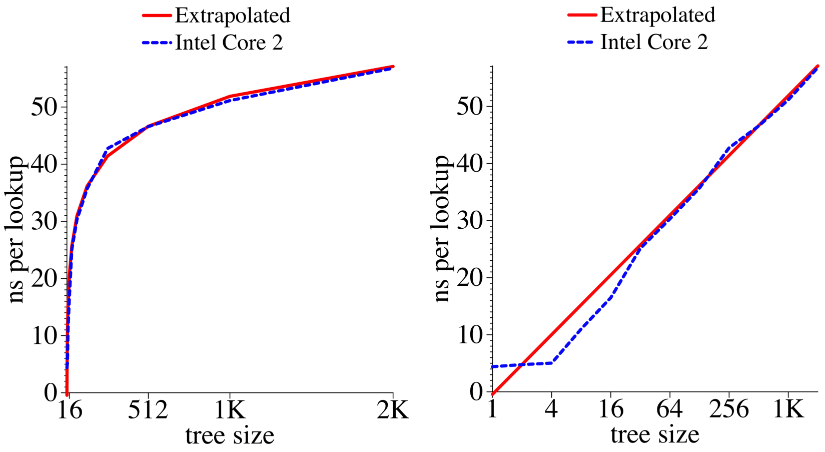
Limits of Abstraction: Question #1

- Question#1:
  - The average lookup time for tree of size 1024 (1K) is 50ns
  - What is the expected lookup time for a tree of size 1048576 (1M)?

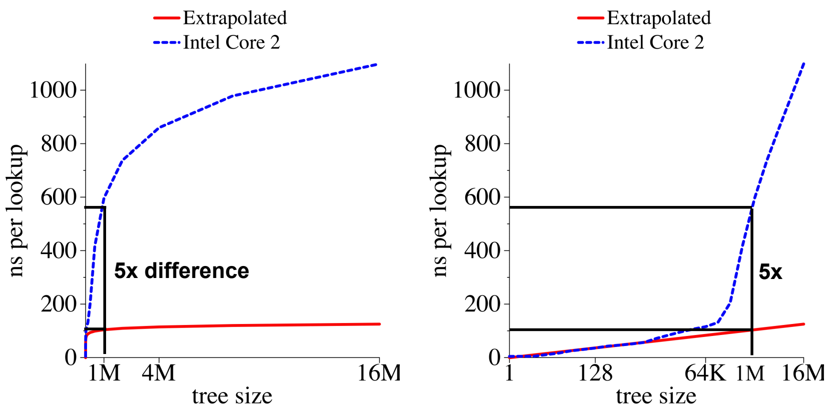
- Analysis (from what you know from 121, 240, 320):
  - 1024 is $2^{10,}$ 1048576 is 2^{20}
  - Binary search is O(log *n*)
  - Based on that, it will take roughly twice as long to lookup in a 2^{20} tree than a 2^{10} tree
  - Expected time: 100ns

- Let's evaluate this ***experimentally***
  - Experiment: create a balanced tree of size *n*, lookup a random node 100 million times, find the average lookup time, repeat

Average Time per Lookup

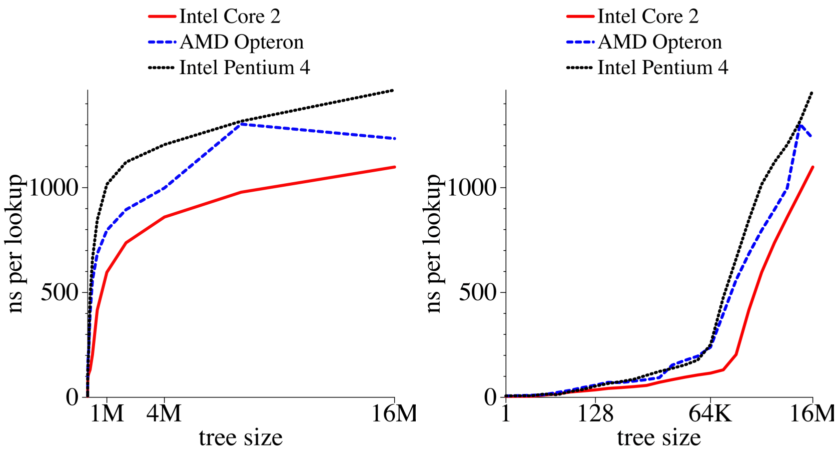


Average Time per Lookup

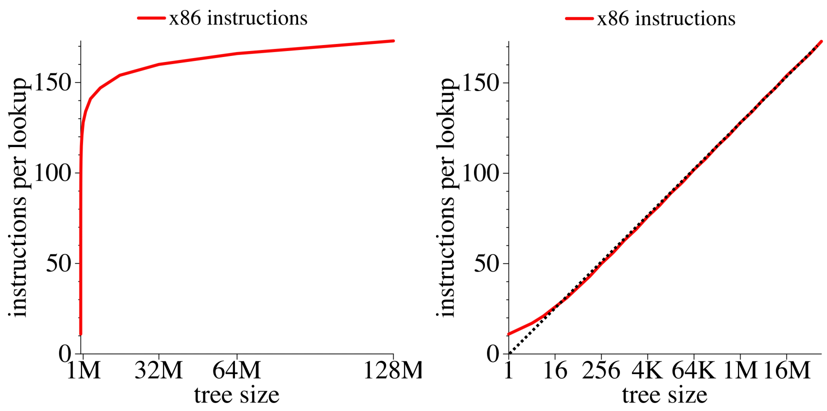


What is going on here?

Average Time per Lookup



Average *Instructions* per Lookup



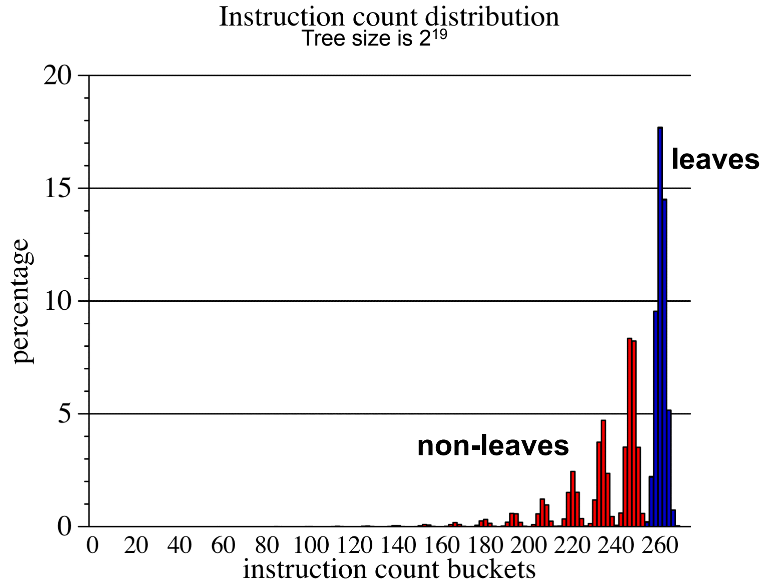
So number of instructions isn't the problem

Question #1 Discussion

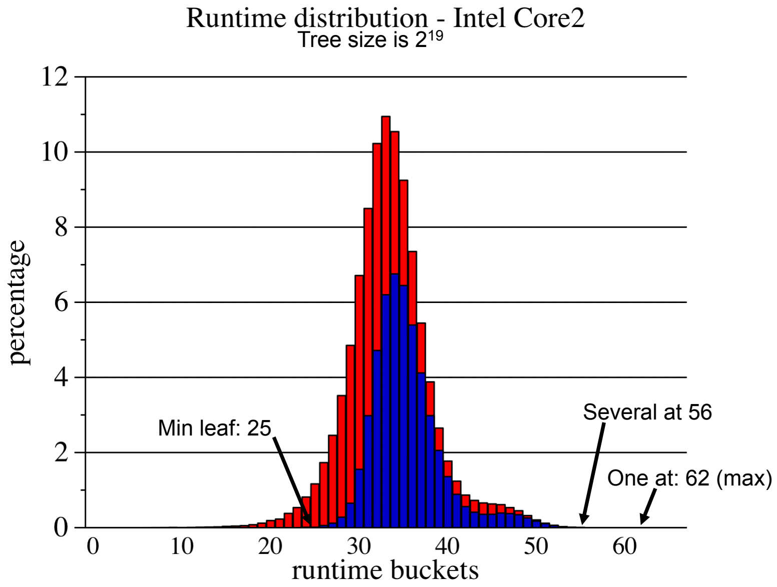
- Analytical answer assuming O(log n)
  - 2^{10} to 2^{20} will have 2x slowdown
- Experimental result
  - 2^{10} to 2^{20} has a 10x slowdown
- 5x gap in expected from experimental!
- What is going on?
  - Modern processor have "fast" and "slow" memories
    - Fast memory is called a "cache"
  - As tree gets bigger, it doesn't fit in fast memory anymore
  - Result: average memory access latency becomes slower

Limits of Abstraction: Question #2

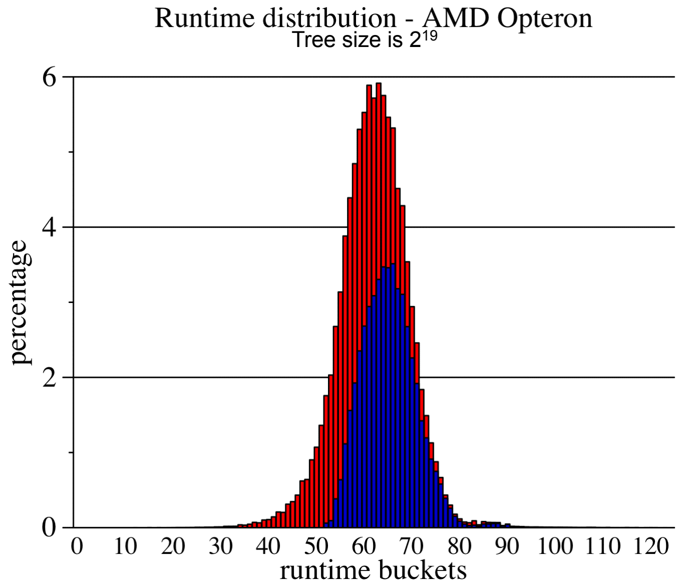
- Question #2:
  - What is the expected distribution of lookup times?
  - That is, for a tree with height *h*, what is the histogram of repeatedly looking up a random value in the tree?
- Analysis:
  - 50% of nodes are at level *n* (leaves), slowest
  - 25% of nodes are at level *n-1,* a bit faster
  - 12.5% of nodes are at level *n-2*, a bit faster yet
  - 6.25%, 3%, 1.5%…
- Let’s evaluate this experimentally
  - Experiment: create a balanced tree of size 2^{19}, for each node, lookup it up 100 million times (consecutively), calculate lookup time for each node, create a histogram

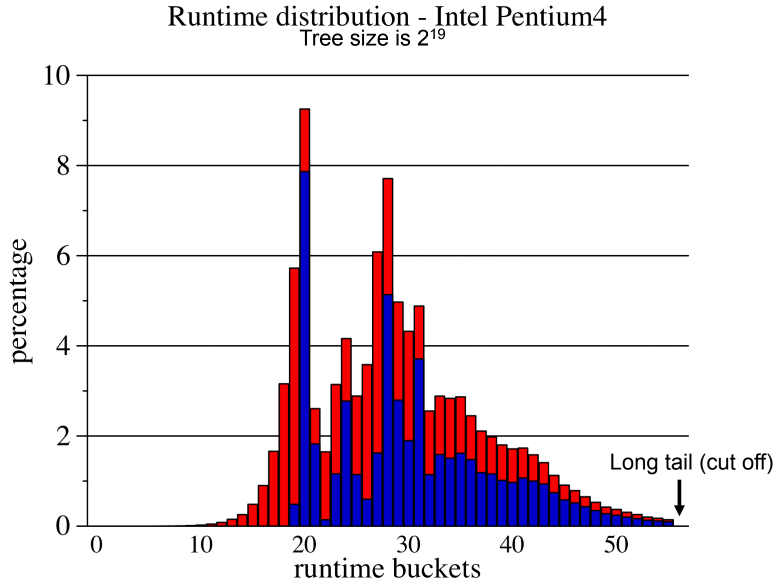


What about runtime? (not instructions)



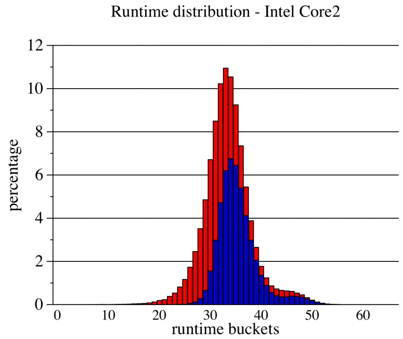
What is going on here?





Fastest and Slowest Leaf Nodes (Core2)

- Expectation:
  - Let's just consider the leaves
  - Same depth, similar instruction count -> similar runtime
- Some of the fastest leaves (all ~24): L = Left, R = Right
  - `LLLLLLLLLLLLLLLLLL`
  - `LLLLLLLLLLLLLLLLLR` (or any with one "R")
  - `LLRRLLRRLLRRLLRRLL`
  - `LLRRLRLRLRLRLRLRLR`
  - `LLRRRLRLLRRRLRLLRR`
- `RRRRRRRRRRRRRRRRRR`
  - was worst than average (~41)
- Some of the slowest leaves:
  - `RRRRLRRRRLRLRRLLLL` (~62)
  - `RRRRLRRRRRRLLLRRRL` (~56)
  - `RRRRRLRRRLRRLRLRLL` (~56)

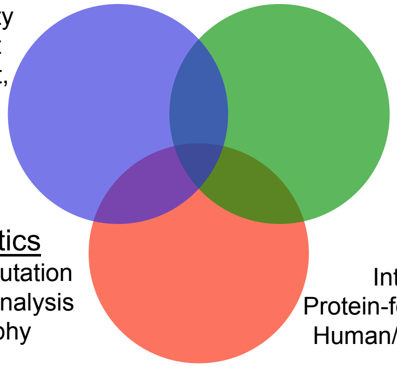


Question #2 Discussion

- Analytical expectation
  - 50%, 25%, 12.5%, 6.25%, 3%, 1.5%…
  - All leaf nodes with similar runtime
- Experimental result
  - Significant variation, position in tree matters
  - All "left" is fastest, all "right" is slow, but not the slowest
  - Pattern of left/right seems to matter significantly
- What is going on?
  - "Taken" branches are slower than "non-taken" branches
  - Modern processors learn and *predict* branch directions over time
    - Can detect simple patterns, but not complicated ones
  - Result: exact branching behavior matters

Computer Science as an Estuary

Where does CIS371 fit into computer science?
Engineering, some science



Engineering
Design
Handling complexity
Real-world impact
Examples: Internet,
microprocessor

Science
Experiments
Hypothesis
Examples:
Internet behavior,
Protein-folding supercomputer
Human/computer interaction

Mathematics
Limits of computation
Algorithms & analysis
Cryptography
Logic
Proofs of correctness

Other Issues
Public policy, ethics,
law, security

What is Computer Architecture?

"Computer Organization"

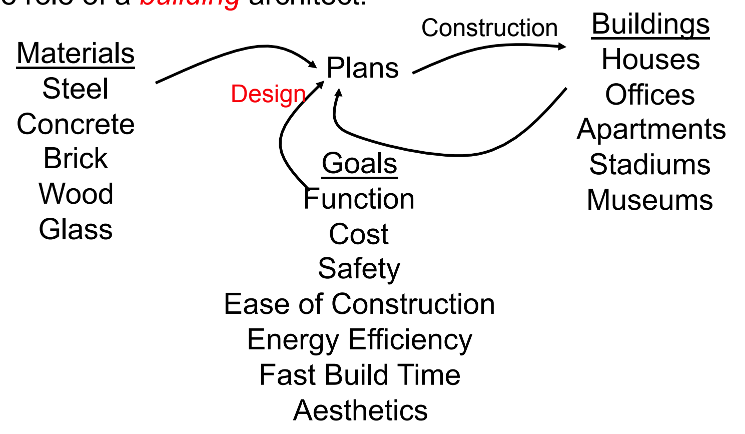
- "Digital Systems Organization and Design"
  - Don't really care about "digital systems" in general
  - **"Computer Organization and Design"**
- **Computer architecture**
  - Definition of ISA to facilitate implementation of software layers
  - The hardware/software interface
- **Computer micro-architecture**
  - Design processor, memory, I/O to implement ISA
  - Efficiently implementing the interface
- CIS 371 is mostly about processor micro-architecture
  - Confusing: architecture also means micro-architecture

Computer Architecture

- *"Computer Architecture* is the science and art of selecting and interconnecting hardware components to create computers that meet functional, performance and cost goals." - WWW Computer Architecture Page
- An analogy to architecture of buildings…

What is Building Architecture?

The role of a *building* architect:



Materials
Steel
Concrete
Brick
Wood
Glass

Design

Plans

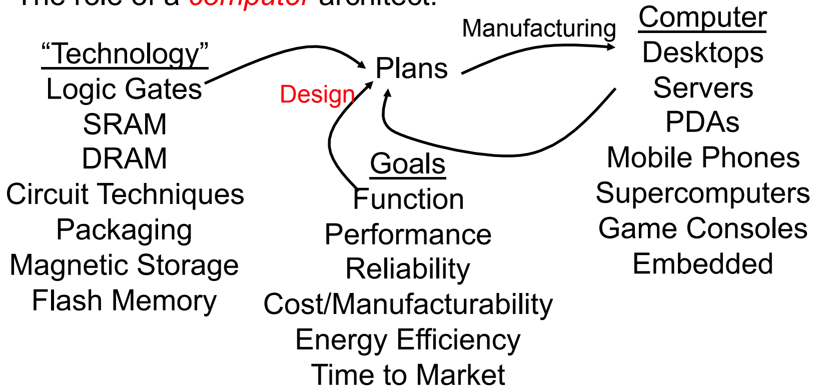
Construction

Buildings
Houses
Offices
Apartments
Stadiums
Museums

Goals
Function
Cost
Safety
Ease of Construction
Energy Efficiency
Fast Build Time
Aesthetics

What is Computer Architecture?

The role of a *computer* architect:



“Technology”
Logic Gates
SRAM
DRAM
Circuit Techniques
Packaging
Magnetic Storage
Flash Memory

Design

Plans

Manufacturing

Goals
Function
Performance
Reliability
Cost/Manufacturability
Energy Efficiency
Time to Market

Computer
Desktops
Servers
PDAs
Mobile Phones
Supercomputers
Game Consoles
Embedded

Important differences: age (~60 years vs thousands), rate of change, automated mass production (magnifies design)

Computer Architecture Is Different…

- Age of discipline
  - 60 years (vs. five thousand years)
- Automated mass production
  - Design advances magnified over millions of chips
- Boot-strapping effect
  - Better computers help design next generation
- Rate of change
  - All three factors (technology, applications, goals) are changing
  - Quickly

Design Constraints

- **Functional**
  - Needs to be correct
    - And unlike software, difficult to update once deployed
  - What functions should it support (Turing completeness aside)
- **Reliable**
  - Does it ***continue*** to perform correctly?
  - Hard fault vs transient fault
  - Google story - memory errors and sun spots
  - Space satellites vs desktop vs server reliability
- **High performance**
  - “Fast” is only meaningful in the context of a set of important tasks
  - Not just “Gigahertz” – truck vs sports car analogy
  - Impossible goal: fastest possible design for all programs

Design Goals

- **Low cost**
  - Per unit manufacturing cost (wafer cost)
  - Cost of making first chip after design (mask cost)
  - Design cost (huge design teams, why? Two reasons…)
  - (Dime/dollar joke)
- **Low power/energy**
  - Energy in (battery life, cost of electricity)
  - Energy out (cooling and related costs)
  - Cyclic problem, very much a problem today
- Challenge: balancing the relative importance of these goals
  - And the balance is constantly changing
    - No goal is absolutely important at expense of all others
  - Our focus: *performance,* only touch on cost, power, reliability

Shaping Force: Applications/Domains

- Another shaping force: **applications** (usage and context)
  - Applications and application domains have different requirements
    - Domain: group with similar character
  - Lead to different designs
- **Scientific**: weather prediction, genome sequencing
  - First computing application domain: naval ballistics firing tables
  - Need: large memory, heavy-duty floating point
  - Examples: CRAY T3E, IBM BlueGene
- **Commercial**: database/web serving, e-commerce, Google
  - Need: data movement, high memory + I/O bandwidth
  - Examples: Sun Enterprise Server, AMD Opteron, Intel Xeon

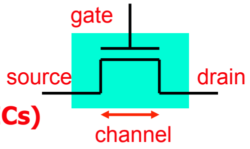
More Recent Applications/Domains

- **Desktop**: home office, multimedia, games
  - Need: integer, memory bandwidth, integrated graphics/network?
  - Examples: Intel Core 2, Core i7, AMD Athlon
- **Mobile**: laptops, mobile phones
  - Need: **low power**, integer performance, integrated wireless
  - Laptops: Intel Core 2 Mobile, Atom, AMD Turion
  - Smaller devices: ARM chips by Samsung and others, Intel Atom
- **Embedded**: microcontrollers in automobiles, door knobs
  - Need: low power, **low cost**
  - Examples: ARM chips, dedicated digital signal processors (DSPs)
  - Over 1 billion ARM cores sold in 2006 (at least one per phone)
- **Deeply Embedded**: disposable "smart dust" sensors
  - Need: extremely low power, extremely low cost

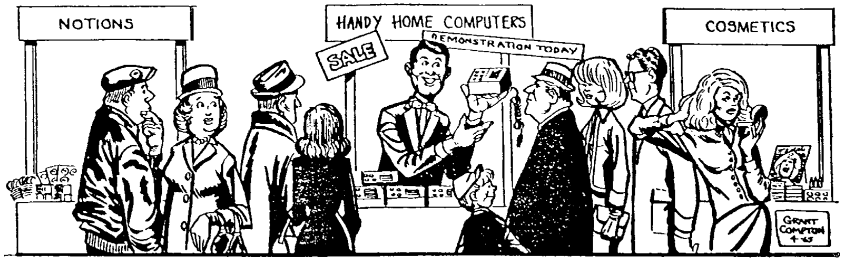
Application Specific Designs

- This class is about **general-purpose CPUs**
  - Processor that can do anything, run a full OS, etc.
  - E.g., Intel Core 2, AMD Athlon, IBM Power, ARM, Intel Itanium
- In contrast to **application-specific chips**
  - Or **ASICs** (Application specific integrated circuits)
    - Also application-domain specific processors
  - Implement critical domain-specific functionality in hardware
    - Examples: video encoding, 3D graphics
  - General rules
    - - Hardware is less flexible than software
    - + Hardware more effective (speed, power, cost) than software
    - + Domain specific more "parallel" than general purpose
      - But general mainstream processors becoming more parallel
- Trend: from specific to general (for a specific domain)

Changing Technology

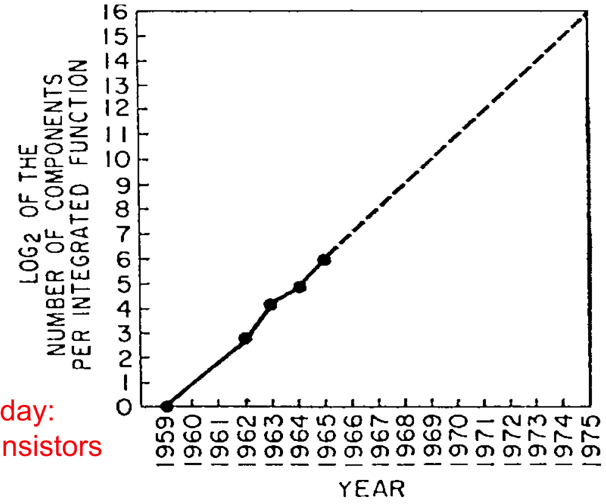


- Basic technology element: **transistor**
  - Solid-state **transistor** (i.e., electrical switch)
  - Basic building block of **integrated circuits (ICs)**
- What's so great about ICs? Everything
  - High performance, high reliability, low cost, low power
  - Lever of mass production
- Several kinds of integrated circuit families
  - **"Static" RAM/logic**: used to make processors (speed optimized)
  - **"Dynamic" RAM**: used to make main memory (cost optimized)
  - **Flash**: non-volatile memory (only important in recent years)
- Disk is also a "technology", but isn't transistor-based
- Other technologies on the near (and medium horizon)
  - 3D silicon, nano-photonics, embedded phase-change memory



Funny or Not Funny?

Moore's Law - 1965



Today:
2^{30} transistors

Changing Technology

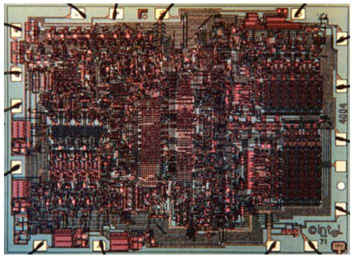
- **Moore's Law**
  - Continued (up until now, at least) transistor miniaturization
- Some technology-based ramifications
  - Absolute improvements in density, speed, power, costs
  - SRAM/logic: density: ~30% (annual), speed: ~20%
  - DRAM: density: ~60%, speed: ~4%
  - Disk: density: ~60%, speed: ~10% (non-transistor)
  - Big improvements in disk density and network bandwidth too
- **Changing quickly and with respect to each other!!**
  - Example: density increases faster than speed
  - Fundamentally changes design
  - Different tradeoffs
  - **Constant re-evaluation and re-design**

First Microprocessor

- Intel 4004 (1971)
  - Application: calculators
  - Technology: 10 μm PMOS

  - 2300 transistors
  - 13 mm^2
  - 108 KHz
  - 12 Volts

  - 4-bit data
  - Single-cycle datapath

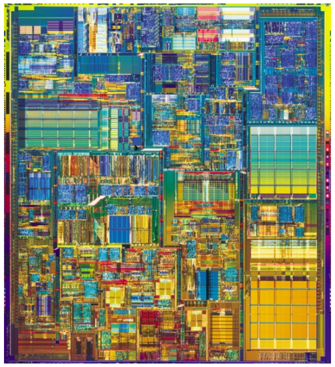


More Recent Microprocessor

- Intel Pentium4 (2003)
  - Application: desktop/server
  - Technology: 0.09 μm CMOS (1/100X)

  - 55M transistors (20,000X)
  - 101 mm^2 (10X)
  - 3.4 GHz (10,000X)
  - 1.2 Volts (1/10X)

  - 32/64-bit data (16X)
  - 22-stage pipelined datapath
  - 4-way superscalar
  - Two levels of on-chip cache
  - data-parallel (SIMD) instructions, hyperthreading

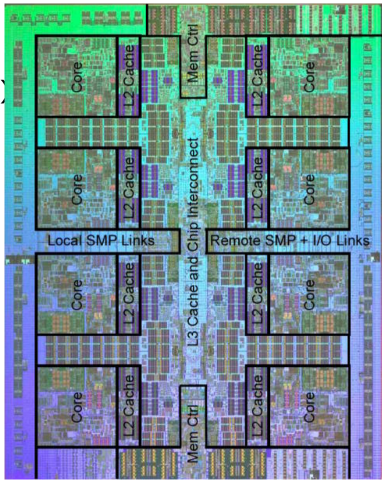


More Recent Microprocessor

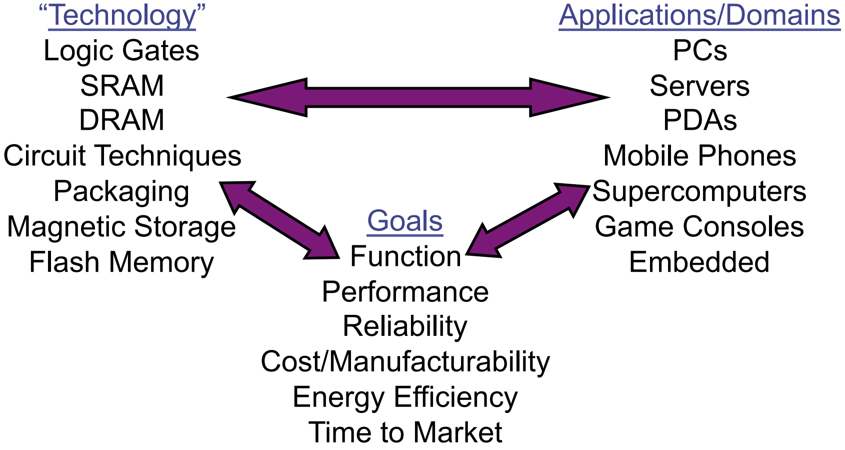
- IBM POWER7 (2010)
  - Application: server
  - Technology: 45 nm CMOS (1/200X)

  - 1.2B transistors (20,000X)
  - 567 mm^2 (40X)
  - 4 GHz (10,000X)
  - 1.2 V (1/10X)

  - 8 cores
  - 64/128/256-bit data (64X)
  - 22-stage pipelined datapath
  - 4-way superscalar, out-of-order
  - 32 Mbyte L3 cache
  - SIMD instructions, hyperthreading



Constant Change



"Technology"
Logic Gates
SRAM
DRAM
Circuit Techniques
Packaging
Magnetic Storage
Flash Memory

Applications/Domains
PCs
Servers
PDAs
Mobile Phones
Supercomputers
Game Consoles
Embedded

Goals
Function
Performance
Reliability
Cost/Manufacturability
Energy Efficiency
Time to Market

- Absolute improvement, **different rates of change**, **cyclic**
- Better computers help design the next generation (CAD)

Managing This Mess

- Architect must consider all factors
  - Goals/constraints, applications, implementation technology

- Questions
  - How to deal with all of these inputs?
  - How to manage changes?

- Answers
  - Accrued institutional knowledge (stand on each other's shoulders)
  - Experience, rules of thumb
  - Discipline: clearly defined end state, keep your eyes on the ball
  - **Abstraction and layering**

Aside: Full Disclosure

- Potential sources of bias or conflict of interest

- Most of my funding governmental (your tax $$$ at work)
  - National Science Foundation (NSF)
  - DARPA

- My non-governmental sources of research funding
  - **NVIDIA** (sub-contract of large DARPA project)
  - **Intel**
  - **Sun/Oracle** (hardware donation)

- Collaborators and colleagues
  - Intel, IBM, AMD, Sun, Microsoft, Google, VMWare, etc.
  - (Just about every major computer hardware company)

Academic Misconduct

- Cheating will not be tolerated

- General rule:
  - Anything with your name on it must be **YOUR OWN** work

- Possible penalties
  - Zero on assignment (minimum)
  - Fail course
  - Note on permanent record
  - Suspension
  - Expulsion

- Penn’s Code of Conduct
  - http://www.vpul.upenn.edu/osl/acadint.html