CIS 501: Computer Architecture

Unit 1: Introduction

Slides developed by Milo Martin & Amir Roth at the University of Pennsylvania
with sources that included University of Wisconsin slides
by Mark Hill, Guri Sohi, Jim Smith, and David Wood

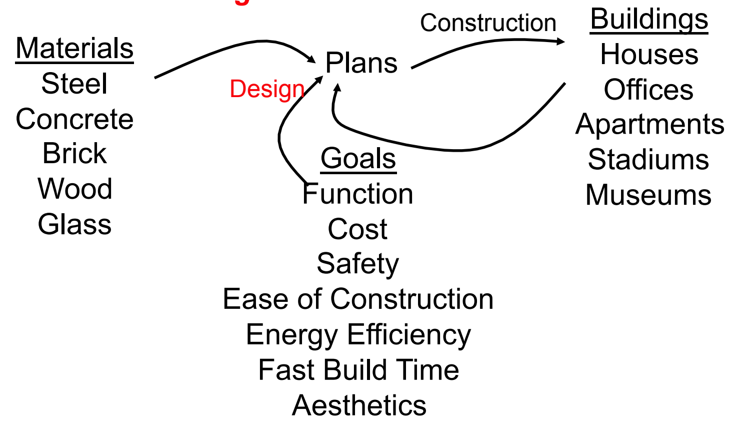
What is Computer Architecture?

What is 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…

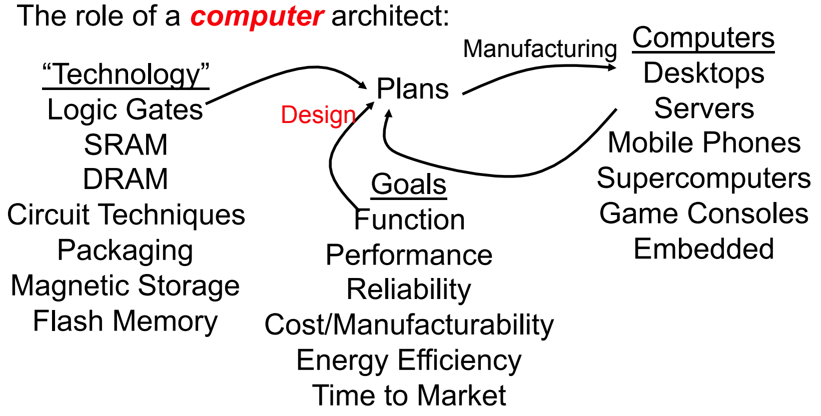
What is ~~Computer~~ Architecture?

The role of a ***building*** architect:



What is Computer Architecture?

The role of a ***computer*** architect:



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

Design Goals & 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 & Constraints

- **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, Qualcomm; Intel Atom
- **Embedded**: microcontrollers in automobiles, door knobs
  - Need: low power, **low cost**
  - Examples: ARM chips, dedicated digital signal processors (DSPs)
  - Over 6 billion ARM cores sold in 2010 (multiple 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 i7, 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)

Technology Trends

Constant Change: Technology

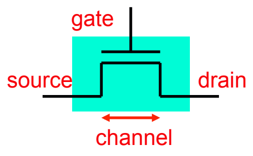
“Technology”
Logic Gates
SRAM
DRAM
Circuit Techniques
Packaging
Magnetic Storage
Flash Memory

Applications/Domains
Desktop
Servers
Mobile Phones
Supercomputers
Game Consoles
Embedded

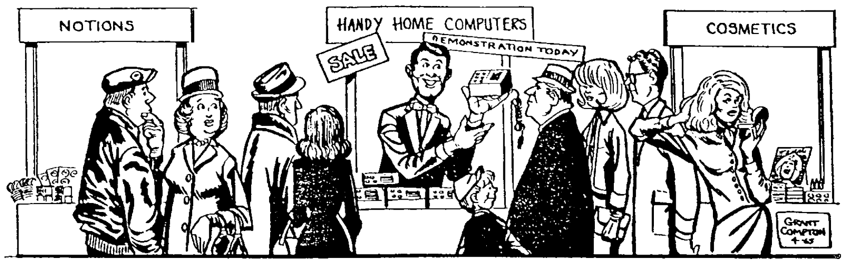
Goals
Function
Performance
Reliability
Cost/Manufacturability
Energy Efficiency
Time to Market

- Absolute improvement, **different rates of change**
- New application domains enabled by technology advances

“Technology”

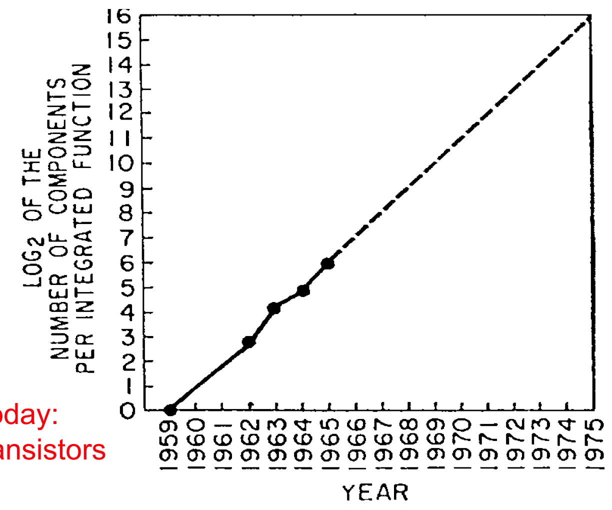


- Basic element
  - Solid-state **transistor** (i.e., electrical switch)
  - Building block of **integrated circuits (ICs)**
- What’s so great about ICs? Everything
  - + High performance, high reliability, low cost, low power
  - + Lever of mass production
- Several kinds of integrated circuit families
  - **SRAM/logic**: optimized for speed (used for processors)
  - **DRAM**: optimized for density, cost, power (used for memory)
  - **Flash**: optimized for density, cost (used for storage)
  - Increasing opportunities for integrating multiple technologies
- Non-transistor storage and inter-connection technologies
  - Disk, optical storage, ethernet, fiber optics, wireless



Funny or Not Funny?

Moore’s Law - 1965



Today:
2^{32} transistors

Technology Trends

- **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 flash memory and network bandwidth, too
- **Changing quickly and with respect to each other!!**
  - Example: density increases faster than speed
  - Trade-offs are constantly changing
  - **Re-evaluate/re-design for each technology generation**

Technology Change Drives Everything

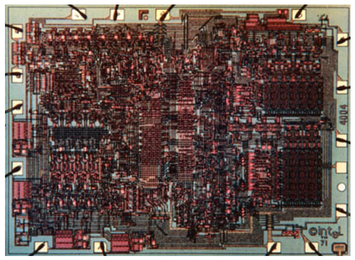
- Computers get 10x faster, smaller, cheaper every 5-6 years!
  - A 10x quantitative change is qualitative change
  - Plane is 10x faster than car, and fundamentally different travel mode
- New applications become self-sustaining market segments
  - Recent examples: mobile phones, digital cameras, mp3 players, etc.
- Low-level improvements appear as discrete high-level jumps
  - Capabilities cross thresholds, enabling new applications and uses

Revolution I: The Microprocessor

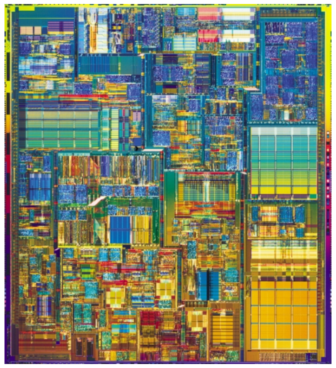
- **Microprocessor revolution**
  - One significant technology threshold was crossed in 1970s
  - Enough transistors (~25K) to put a 16-bit processor on one chip
  - Huge performance advantages: fewer slow chip-crossings
  - Even bigger cost advantages: one "stamped-out" component
- Microprocessors have allowed new market segments
  - Desktops, CD/DVD players, laptops, game consoles, set-top boxes, mobile phones, digital camera, mp3 players, GPS, automotive
- And replaced incumbents in existing segments
  - Microprocessor-based system replaced supercomputers, "mainframes", "minicomputers", etc.

First Microprocessor

- Intel 4004 (1971)
  - Application: calculators
  - Technology: 10000 nm
  - 2300 transistors
  - 13 mm^2
  - 108 KHz
  - 12 Volts
  - 4-bit data
  - Single-cycle datapath



Pinnacle of Single-Core Microprocessors



- Intel Pentium4 (2003)
  - Application: desktop/server
  - Technology: 90nm (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
  - 3 instructions per cycle (superscalar)
  - Two levels of on-chip cache
  - data-parallel vector (SIMD) instructions, hyperthreading

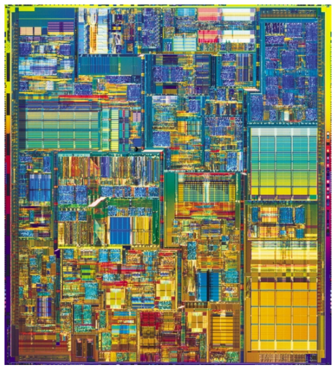
Tracing the Microprocessor Revolution

- How were growing transistor counts used?
- Initially to widen the datapath
  - 4004: 4 bits → Pentium4: 64 bits
- … and also to add more powerful instructions
  - To amortize overhead of fetch and decode
  - To simplify programming (which was done by hand then)

Revolution II: Implicit Parallelism

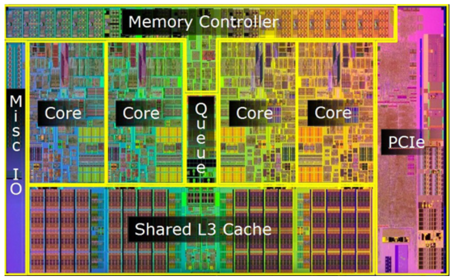
- Then to **extract implicit instruction-level parallelism**
  - Hardware provides parallel resources, figures out how to use them
  - Software is oblivious
- Initially using pipelining …
  - Which also enabled increased clock frequency
- … caches …
  - Which became necessary as processor clock frequency increased
- … and integrated floating-point
- Then deeper pipelines and branch speculation
- Then multiple instructions per cycle (superscalar)
- Then dynamic scheduling (out-of-order execution)
- We will talk about these things

Pinnacle of Single-Core Microprocessors



- Intel Pentium4 (2003)
  - Application: desktop/server
  - Technology: 90nm (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
  - 3 instructions per cycle (superscalar)
  - Two levels of on-chip cache
  - data-parallel vector (SIMD) instructions, hyperthreading

Modern Multicore Processor



- Intel Core i7 (2009)
  - Application: desktop/server
  - Technology: 45nm (1/2x)
  - 774M transistors (12x)
  - 296 mm^2 (3x)
  - 3.2 GHz to 3.6 Ghz (~1x)
  - 0.7 to 1.4 Volts (~1x)
  - 128-bit data (2x)
  - 14-stage pipelined datapath (0.5x)
  - 4 instructions per cycle (~1x)
  - Three levels of on-chip cache
  - data-parallel vector (SIMD) instructions, hyperthreading
  - **Four-core multicore** (4x)

Revolution III: Explicit Parallelism

- Then to support **explicit data & thread level parallelism**
  - Hardware provides parallel resources, software specifies usage
  - Why? diminishing returns on instruction-level-parallelism
- First using (subword) vector instructions…, Intel's SSE
  - One instruction does four parallel multiplies
- … and general support for multi-threaded programs
  - Coherent caches, hardware synchronization primitives
- Then using support for multiple concurrent threads on chip
  - First with single-core multi-threading, now with multi-core
- Graphics processing units (GPUs) are highly parallel
  - Converging with general-purpose processors (CPUs)?

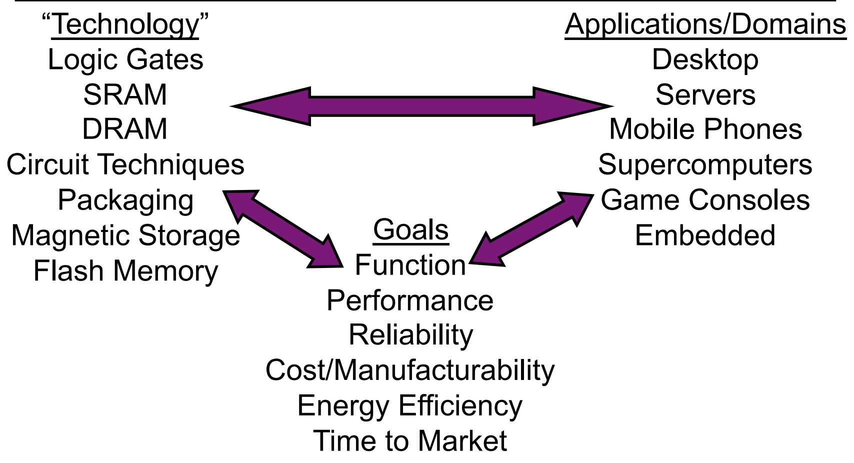
To ponder…

Is this decade's
"**multicore revolution**"
comparable to the original
"**microprocessor revolution**"?

Technology Disruptions

- Classic examples:
  - The transistor
  - Microprocessor
- More recent examples:
  - Multicore processors
  - Flash-based solid-state storage
- Near-term potentially disruptive technologies:
  - Phase-change memory (non-volatile memory)
  - Chip stacking (also called 3D die stacking)
- Disruptive "end-of-scaling"
  - "If something can't go on forever, it must stop eventually"
  - Can we continue to shrink transistors for ever?
  - Even if more transistors, not getting as energy efficient as fast

Recap: Constant Change



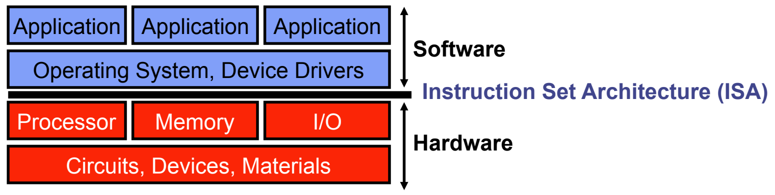
Managing This Mess

- 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**

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 *n* objects…
    - Implemented using interfaces of layer *n* – 1
    - Don't need to know interfaces of layer *n* – 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 (example: Digital TV)
- **Opacity**: hard to reason about performance across layers

Abstraction, Layering, and Computers


Application
Application
Application
Operating System, Device Drivers
Software
Instruction Set Architecture (ISA)
Processor
Memory
I/O
Circuits, Devices, Materials
Hardware

- **Computer architecture**
  - Definition of **ISA** to facilitate implementation of software layers
- This course mostly on **computer micro-architecture**
  - Design of chip to implement **ISA**
- Touch on compilers & OS (*n* +1), circuits (*n* -1) as well

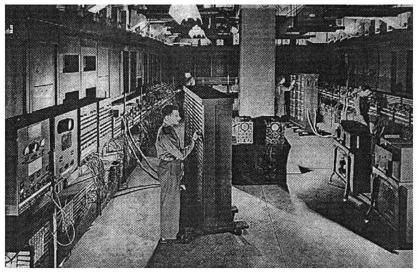
Course Overview

Why Study Computer Architecture?

- **Understand where computers are going**
  - Future capabilities drive the (computing) world
  - Real world-impact: no computer architecture → no computers!
- **Understand high-level design concepts**
  - The best architects understand all the levels
    - Devices, circuits, architecture, compiler, applications
- **Understand computer performance**
  - Writing well-tuned (fast) software requires knowledge of hardware
- **Write better software**
  - The best software designers also understand hardware
  - Need to understand hardware to write fast software
- **Design hardware**
  - At Intel, AMD, IBM, ARM, Qualcomm, Oracle, NVIDIA, Samsung

Penn Legacy

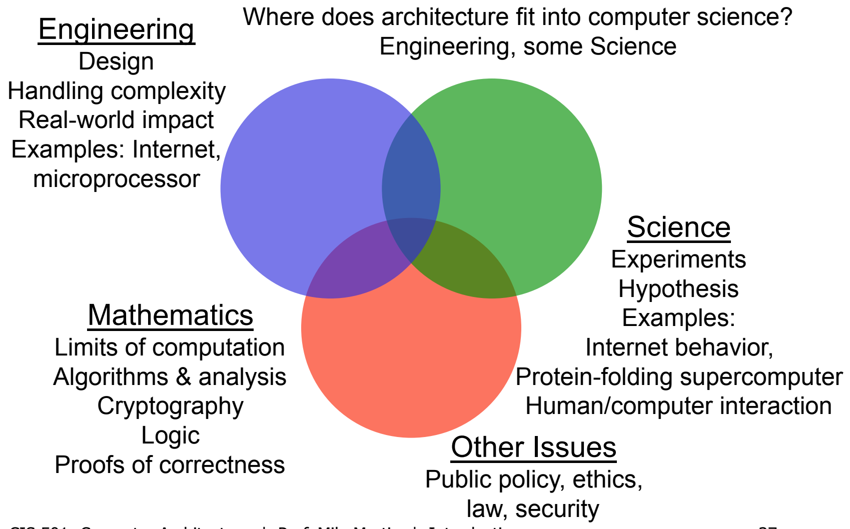
- **ENIAC**: electronic numerical integrator and calculator
  - First operational general-purpose stored-program computer
  - Designed and built here by Eckert and Mauchly
  - Go see it (Moore building)
- **First seminars on computer design**
  - Moore School Lectures, 1946
  - "*Theory and Techniques for Design of Electronic Digital Computers*"



Course Goals

- **See the "big ideas" in computer architecture**
  - Pipelining, parallelism, caching, locality, abstraction, etc.
- Exposure to examples of good (and some bad) engineering
- Understanding computer performance and metrics
  - Experimental evaluation/analysis ("science" in computer science)
  - Gain experience with simulators (architect's tool of choice)
  - Understanding quantitative data and experiments
- Get exposure to "research" and cutting edge ideas
  - Read some research literature (i.e., papers)
- **My role: trick you into learning something**

Computer Science as an Estuary



Course Topics

- More depth on "undergraduate" architecture topics
  - Evaluation metrics and trends
  - ISAs (instruction set architectures)
  - Datapath & pipelining (including branch prediction)
  - Memory hierarchies, caches, & virtual memory
- Overview of semiconductor technology & energy/power
- Parallelism
  - Instruction: multiple issue, dynamic scheduling, speculation
  - Thread: cache coherence and synchronization, multicore
  - Data: vectors and GPUs
- More fun stuff if we get to it

CIS501: Administrivia

- Instructor: **Prof. Milo Martin** (milom@cis.upenn.edu)
  - TAs: Laurel Emurian & Mishal Awadah (plus a few graders, TBD)
- Lectures
  - Please do not be disruptive (I'm easily distracted as it is)
- Three different web sites
  - Course website: syllabus, **schedule**, lecture notes, assignments
    - **https://www.cis.upenn.edu/~cis501/**
  - "Piazza": announcements, questions & discussion
    - **https://www.piazza.com/upenn/fall2012/cis501**
    - ***The* way to ask questions/clarifications**
    - Can post to just me & TAs or anonymous to class
    - As a general rule, **no need to email me & TAs directly**
  - "Blackboard" (or maybe "Canvas") for grade book
    - **https://courseweb.library.upenn.edu/**

Resources

- Readings
  - ***"Microprocessor Architecture: From Simple Pipelines to Chip Multiprocessors"*** by Jean-Loup Baer
    - Penn Bookstore, Amazon ($68), or Kindle eBook ($54)
    - Three copies "on reserve" at the Engineering Library
  - Research papers (linked from course web page)
- Last year's course (to review lectures prior to class):
  - https://www.cis.upenn.edu/~milom/cis501-Fall11/
- Free resources
  - ACM digital library: http://www.acm.org/dl/
  - Computer architecture page: http://www.cs.wisc.edu/~arch/www/
- Local resources:
  - Architecture & Compilers Group: http://www.cis.upenn.edu/acg/

Prerequisites

- **Basic computer organization an absolute must**
  - Basic digital logic: gates, boolean functions, latches
  - Binary arithmetic: adders, hardware mul/div, floating-point
  - Basic datapath: ALU, register file, memory interface, muxes
  - Basic control: single-cycle control, microcode
  - **Familiarity with assembly language**
  - "Computer Organization and Design: Hardware/Software Interface"
  - https://www.cis.upenn.edu/~cis371/
- **Significant programming experience**
  - No specific language required
  - Why? assignments require writing code to **simulate hardware**
    - Not difficult if competent programmer; extremely difficult if not

The Students of CIS501

- Three different constituencies, different backgrounds
- **PhD students**
  - More research focused
  - WPE-I PhD qualifying exam
- **MSE students (CIS, EMBS, Robotics, others)**
  - Build upon on undergraduate coursework
    - Which, unfortunately, ***varies widely***
- **BSE (undergraduate) students**
  - Expand on undergraduate coursework at Penn (CIS371)
  - Especially for those considering graduate school or a hardware job
- Extremely difficult to tailor course for all three constituencies

For Non-CIS Students…

- Registration priority is given to CIS students
- For non-CIS students
  - As the class is already extremely large…
  - I'll only consider admitting students **not in their first semester**
- For non-CIS students ***not*** in their first semester, if you want to be considered, send me via email (milom@cis):
  1. Your name & Penn email address
  2. What program you're enrolled in
  3. A transcript of your Penn courses with grades
  4. Description of prior courses on computer architecture
  5. A brief description of the largest programming project you've completed (lines of code, overall complexity, language used, etc.)

Coursework

- Paper reviews (5 or 6 throughout the semester)
  - Short response to papers we'll read for class
  - Discuss and write up in **groups** of four
    - Twist: can't work with the same group member
- Homework assignments (4 of 5 throughout semester)
  - Written questions and **programming**
  - **Due at beginning of class**
  - 2 total "grace" periods (next class period), **max one per assignment**
    - Hand in late, no questions asked
  - **No assignments accepted after solutions posted**
  - **Individual work**
- Exams (2 exams)
  - Midterm, **to be determined**
    - **Likely: an evening during week of Oct 29th-Nov 2nd**
    - **Email me ASAP if you might have a conflict**
  - Cumulative final (WPE I for PhD students)
    - **Wed, December 19th 12-2pm**
    - Yes, this is the very last day of final exams. No, we can't move it earlier.

Grading

- Tentative grade contributions:
  - Paper reviews: 5%
  - Homework assignments: 20%
  - Exams: 75%
    - Midterm: 30%
    - Final: 45%
- Typical grade distributions
  - A: 40%
  - B: 40%
  - C/D/F: 20%

Academic Misconduct

- Cheating will **not** be tolerated
- General rule:
  - Anything with your name on it must be **YOUR OWN** work
  - Example: individual work on homework assignments
- 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

Full Disclosure

- Potential sources of bias or conflict of interest
- Most of my funding governmental (your tax $$$ at work)
  - National Science Foundation (NSF)
  - DARPA & ONR
- 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, Oracle, Microsoft, Google, VMWare, ARM, etc.
  - (Just about every major computer company)

CIS 501 Frequently Asked Questions (FAQs)

Should I Take CIS501 as an Undergrad?

- Answer: Maybe; probably not.
- Why?
  - Does reinforce material in CIS371 and add depth
  - Good for those considering graduate school or hardware job
  - Fills "requirement" for sub-matriculation
    - However, good grade in CIS371 can likely allow replacing CIS501 with other grad-level course via petition
- Why not?
  - The lecture portion of CIS501 and CIS371 are increasingly similar
    - And same instructor
    - Main difference is another week on dynamic scheduling
  - Other differences
    - CIS501 reads primary sources (research papers)
    - CIS501 assignments tailored to simulation & analysis
      - In contrast, focus of CIS371 "lab" is design using Verilog
  - Consider next semester's "CIS700" by new professor Joe Devietti
    - Or ***his*** version of CIS501 next year

Course Project? Not This Year

- Question: Why no course project?
  - (Some prior years of CIS501 has have course research projects)
- Answer: With over 100 students, it just doesn't work well
  - Too many projects for me to give feedback/guidance
  - Also, second half of semester is already crowded
- Partial mitigation:
  - Expanded regular homework assignments to cover more ground
  - Final assignment is almost a mini-project, but more well defined

Homework?

- Question: Why is homework such a small percentage of overall course grade?
- Answers:
  - One unfortunate reason: academic misconduct concerns
  - Another reason: grade non-uniformity & distribution (homework scores vary less than exam scores)
- Bigger picture answer:
  - You'll learn a lot from the assignments (no matter how much they are "worth" grade-wise)
  - Again, grades not the main focus; my goal is provide an environment that tricks you into learning

Is There a Late Homework Policy?

- Yes, see earlier slide titled: "Coursework"

Large Class Size

- Question: how are we going to manage such a large class?
- Answer: we'll do the best we can
  - Just one instructor, two TAs
- We'll need ***your*** help!
  - Post questions on Piazza, so others can see responses
    - Only use email for private & non-technical sort of issues
  - Help out by answering other student's questions on Piazza
  - Come to TA office hours
  - Come to class on time to avoid disrupting others
  - Start homework assignments early

Any other advice?

Last year, I asked the students:
"What is one piece of advice you would give to someone taking this course in the future on how to succeed in the course?"

Do I *Really* Need the Prerequisites?

- **In the words of students from last year:**
  - "Better have some background on computer architecture"
  - "Make sure you have a good Computer Science background ie, good programming knowledge in either C or Java, architecture basics, etc"
  - "Make sure you are proficient in either Java/C++ and have some prior background in computer architecture (assembly, OS, etc)"
  - "Take the course only if you have confidence in your abilities to code in either C++ or Java. Also know CIS371 properly before enrolling for the course."
  - "If you've taken the undergrad version of computer architecture you will be fine."
  - "Go over the pre-requisite course before the semester."
  - "Have a solid background of data structures, computer organization and some knowledge of operating systems."
  - "1.) Brush up your coding skills and computer architecture basics before taking the course 2.) Choose the other courses for the semester wisely as this course would involve a lot of course work."

Should I Attend Class?

- **In the words of students from last year**:
  - "Keep track of class lectures and try not missing any"
  - "Go to class. The lectures are informative."
  - "Attend lecture and don't get behind"
  - "Take good notes during class - there are many nuances that I think can only be picked up through Prof's lecture. Reading the lecture slides and the book doesn't seem to be enough"
  - "Attend class and listen carefully"
  - "Try to follow in class"
  - "Attend every class"

Prepare & Review?

- **In the words of students from last year**:
  - "Read the lecture materials in advance"
  - "Please actively interact with the instructor in class. And if you are not familiar with the materials, read them before class."
  - "Focus on the lecture discussion and lecture notes."
  - "Pay attention in the class and try to understand the key point in each lecture in class; review the course material after the class. Try to connect each knowledge point together to get a big picture."
  - "Read the material before and after each class it is very important that you have an understanding of the material"
  - "Pay attention in lecture, study the slides carefully and in detail, and spend time thinking about *why* you're programming what you're told to on the homeworks."
  - "Pay really good attention in class and make good use of the TA hours as much as you can."
- **Note: reading assignments already on course schedule**
  - **Last year's lecture notes available from last year's page**

Advice on Assignments

- **In the words of students from last year:**
  - "Think ahead a bit when doing the assignments."
  - "Develop your programming skills and do not keep the work until the end."
  - "It is really important to understand what is happening before beginning to code"
  - "Get your C/C++/Java basics spot on."
  - "Do the programming assignments. That helps in understanding how the things work."
  - "Be prepared to spend 6+ hours on each of the programming assignments.
  - "Be ready to put lot of time and effort on the homeworks."
  - "The course rocks. But it demands considerable time, both for homeworks and exams. Better learn to manage your time if you want to take this course."
  - "Follow the debug traces while debugging your code"
  - "Be thorough with the homework assignments and make sure to test against the trace files."
  - "Start homework early!"

Connect the Dots

- **In the words of students from last year**:
  - "Make sure you understand not only the components and how they fit together, but also the tradeoffs that were taken into account to get to where we are today."
  - "Be sure to see the big picture ideas of many of the topics introduced."
  - "Spend time thinking about the whole picture of architecture. Everything here constructs a logic chain."
  - "Think more, learn more, try hard to build a virtual computer in your mind."
  - "Be very regular, and as thorough as possible in understanding the concepts and techniques taught in the course."
  - "Think out-of-box and go by what is taught. Everything is simple and based on two truisms add indirection and add transistors."
  - "Read more, think independently about what you've read and be careful about the questions."
  - "If you don't understand, then ask."

Any other questions?

Now what?

First Assignment – Paper Review #1

- Read "*Cramming More Components onto Integrated Circuits*" by Gordon Moore
- As a **group of four**, meet and discuss the paper
  - **Briefly** answer the questions on the next slide
  - The goal of these questions is to get you reading, thinking about, and discussing the paper
  - **Your answers should be short but insightful. For most questions, a single short paragraph will suffice**
- E-mail the answers to me:
  - Text only, **no html or attachments**, please
  - Send to: **cis501+reviews@cis.upenn.edu**
    - The "+reviews" is important, don't leave it out
  - Carbon copy (CC) all group members
  - Include the names of all group member at the start of the e-mail
- Due: "last thing" Wednesday, Sept 12th
  - In-class discussion on Thursday, Sept 13th

Paper Review #1 Questions

- Q1: The figure on page 2 graphs relative manufacturing cost per component against the number of components per integrated circuit. Why do the chips become less cost effective per component for both very large and very small numbers of components per chip?
- Q2: One of the potential problems which Moore raises (and dismisses) is heat. Do you agree with Moore's conclusions? Either justify or refute Moore's conclusions.
- Q3: A popular misconception of Moore's law is that it states that the speed of computers increases exponentially, however, that is not what Moore foretells in this paper. Explain what Moore's law actually says based on this paper.

For Next Week…

- 1. **Sign up for CIS 501 on Piazza**
- 2. Assigned readings:
  - **Chapter 1 for Thursday**
  - "Cramming More Components onto Integrated Circuits" by Moore
    - **Group discussion responses for "last thing" Wednesday**
    - Use "group" on Piazza feature to find a group
- 3. If you're a **non-CIS student** wanting to take this course
  - If you're **not** a first-year student, **send email** as discussed earlier
- 4. Email me if you have a conflict with an evening exam
  - Week of **Oct 29th-Nov 2nd**
- 5. See me ***right now*** if:
  - You have any other questions about prerequisites or the course