
EECS 452 – Advanced Computer Architecture - I

Spring 2009

Instructor: Gokhan Memik

EECS Dept., Northwestern University

Some material is based on slides developed by Profs. M. Hill, D. Wood, G. Sohi and J. Smith at the University of Wisconsin-Madison and Prof. A. Moshovos at the University of Toronto.

All other material © Gokhan Memik

Course Overview



- **Advanced Computer Architecture**
 - Really: Advanced single-chip processor microarchitecture
- **EECS 361**
 - How to build a processor (or processor system)
- **This course**
 - How to build the best processor?
 - What is best?
 - Porsche? SUV? Truck? Train? Plane? Bike?

On the meaning of BEST



- Really depends on what your goal is:
 - - Moving: Best take truck - unless you have nothing...
 - - SUV? I don't know, you tell me
 - - Porsche? Have money to burn - cruising
- Observation #1:
 - First we need to know the needs are.
 - Moving vs. cruising
- Observation #2:
 - Then we need to be able to judge how well each option serves these needs.
 - Truck vs. Porsche
- What if you had to build the best car for a given purpose?

On the meaning of BEST (cont.)



- Needs:
 - • **Performance:** word processing vs. weather simulation
 - • **Cost:** would you pay 5x \$ for 2x performance?
 - • **Complexity:** Design/validation time -> cost and perf.
 - • **Power:** PDA, laptop, server
- There are a number of forces at work:
 - 1. What does the user needs: applications
 - 2. What does technology offers: semiconductors
- Why this is not as easy:
 - Many applications, some yet to be developed
 - Technology changes

Administrative Issues



- Instructor: Gokhan Memik
- Office: L475
- Best way to communicate with me: e-mail
 - memik@eecs.northwestern.edu
- Please use “452: Your header here” for all your e-mails
- Course web site:
www.eecs.northwestern.edu/~memik/courses/452/index.html
 - Course material will be available (in pdf) 1 hour before the class
- There is no TA
- You are responsible for all material discussed in class

Course Requirements



- EECS 361
 - Design simple uniprocessor
 - Instruction set concepts: registers, instructions, etc.
 - Organization
 - Datapath design
 - Hardwired/microprogrammed control
 - Simple pipelining
 - Basic caches, main memory
- High-level programming experience (C is a must)
- Compilers (back-end) and VLSI bonus
- You are expected to read on your own and fill-in any gaps



- **Computer Architecture: A Quantitative Approach, Hennessy and Patterson, Second Edition.**
- Readings in Computer Architecture, Hill, Jouppi and Sohi.
- Related conference papers - both classic and cutting-edge
 - You will be asked to present
- Conferences:
 - ISCA (International Symposium on Comp Arch)
 - MICRO (International Symposium on Microarchitecture)
 - ASPLOS (Arch. Support for Programming Languages & OSes)
 - HPCA (High-Performance Comp Arch - all encompassing?)
 - Others: PACT, ICS, ISLPED...
- GENERAL INFO: www.cs.wisc.edu/~arch/www
- Online papers: www.computer.org, citeseer.nj.nec.com, [google-scholar](https://scholar.google.com)

Grading



- This is a grad course: You are expected to be able to seek information beyond what is discussed in class.
 - Project 50%
 - Project Proposal (5%)
 - Mid-Term Summary (10%)
 - Project Report (25%)
 - Presentation (10%)
 - Homeworks 20%
 - Final Exam 25%
 - Personal Touch 5%

Project



- This is the most important part of the course
 - You will be required to propose and perform some “original” research in computer architecture
 - I will provide some suggestions
 - You are strongly encouraged to suggest your own:
 - Validate data in some paper
 - Evaluate extension to existing work
 - Propose something completely new (difficult)
- Since this is a class project negative results are OK
 - In general it is hard to publish negative results
- You will probably have to use the SimpleScalar simulator
 - Requires programming skills in C
 - You must be familiar with UNIX
- Groups of 2 (or 3 if necessary or single if you prefer)
- More details coming soon

Policies



- Late submission penalty
 - Homeworks, etc. are due to class time
 - 10% for each day you are late
 - 100% after the solutions are posted
 - You will be given able time to complete all coursework
 - Don't ask for extension
- There is no TA, dealing with late work is a logistical nightmare
- All work must be your own unless otherwise specified
 - Please take this seriously
 - Make sure to reference any external source

Homeworks



- There will be 3-4 assignments
 - They will include assignments from the H&P book
 - May require material that we do not cover in depth in class
- There will be series of programming assignments that are designed to help you learn the simulation infrastructure that is commonly used in our research community: www.simplescalar.com
 - Assignments require strong programming skills primary in C
 - Also require that you are familiar with UNIX systems

What is Computer Architecture



System attributes as seen by the programmer

- The term architecture is used here to describe the attributes of a system as seen by the programmer, i.e., the conceptual structure and functional behavior as distinct from the organization of the dataflow and controls, the logic design, and the physical implementation.

Gene Amdahl, IBM Journal of R&D, April 1964

Levels of Architecture



- **Architecture:** How are things organized and what you can do with them (functionality)
- **Application/System architecture**
 - Structure of the application itself
 - Interface to outside world (API, libraries, GUIs, etc.)
 - Programming languages
 - Operating system
- **Computer architecture**
 - Interface between hardware and software
 - Engineering of optimal hardware structures to match software requirements



Architecture, μ architecture, Implementation

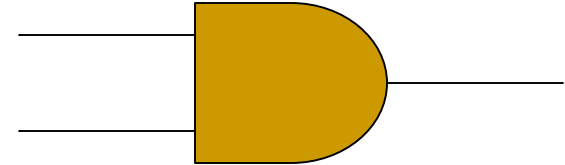
- **Computer architecture: HW/SW interface**
 - instruction set
 - memory management and protection
 - interrupts and traps
 - floating-point standard (IEEE)
- **μ arch (micro-Arch): also called organization**
 - number/location of functional units
 - pipeline/cache configuration
 - programmer transparent techniques: prefetching
- **Implementation: low-level circuits**

Architecture vs. Implementation



■ AND Gate:

- Architecture is the interface:
- 2 inputs - 1 output and function



■ Implementation?

- Transistor based (How many can you think?)
 - static, precharge, CMOS, NMOS, GaAs?
- little-people at work?
- others?

Architecture vs. μ architecture



- The boundaries are a bit blurred, still
- 64-bit Adder:
 - Arch: What it does
 - take two 64-bit numbers produce 64-bit sum
 - μ March: How it does it:
 - Ripple carry
 - Carry lookahead
 - Carry prediction
 - Implementation
 - static, precharge, CMOS, by hand, etc...
- **This course: Architecture, μ March and its interactions/ implications to software and implementation**

Role of the Computer Architect



- **Architect:** Define hardware/software interface
- **μArchitect:** Define the hardware organization, usually same person as above
- How is this done:
 - Study applications
 - Consider underlying technology
 - Consider “best” architecture
 - Performance
 - Cost
 - Complexity
 - Power

Two aspects of CA



- Techniques:
 - This is the accumulated experience
 - Typically, there is no formal way of developing these (innovation)
 - Know how to evaluate
- When to use them?
 - Think of Caches: Were they a good idea for IBM-XT?

Architecture is a “science” of tradeoffs

- No underlying one-truth - we build our own world (or mess)
- Too many options -> too many different ways of being wrong

Why study Computer Architecture?



- Build faster processors
 - Why? My MS-Word, Latex runs quite fast on my Pentium-166
 - MMX thank you very much
- But my Unreal Tournament doesn't
- How about weather simulation? Speech recognition? MPEG-4, Your Killer-App circa 2010-2015?
- Bottom line:
 - Historically, faster processors facilitated new applications
 - Similarly, novel applications created a need for faster machines
- Any reason why this will change?
- Also performance not the only requirement
- **#1: User requirements are constantly changing**

Why study Computer Architecture?



- Implementation technologies change rapidly

Technology	Annual Imp.
Transistor count	25%
Transistor speed	20-25%
DRAM density	60%
DRAM speed	4%
Disk density	25%
Disk speed	3%

- They also change relative to one another

Impact of Technology on Architecture



- Caches:
 - 70's: thousands of xtors, DRAM faster than processor
 - nice way of slowing down your program
 - 80's: depends on machine
 - 90's: millions of xtors, what to do with them, DRAM much slower than processor
 - a must, otherwise your 2Ghz processor spends most of its time waiting for memory
- **#2: Technology changes rapidly making past choices often obsolete**
- **#3: Also opens up new opportunities (e.g., out-of-order, chip multiprocessor)**

Why study Computer Architecture?



- **#4: It is a required class**
- **#5: A lot of jobs for architects**
 - Intel, AMD, Sun, HP, IBM, ...
 - Designers
 - Research jobs
 - Industry Labs
 - Universities
- **#6: The basic notion of engineering**

Classes of Computers – 10 years ago



- **High performance**
 - supercomputers - Cray T-90
 - massively parallel - Cray T3E
- **Balanced cost/performance**
 - workstations - Sun SPARCstations
 - servers - SGI Origin
 - high-end PCs – Pentium
- **Low cost/power**
 - low-end PCs, laptops, personal digital assistants (PDAs)

Classes of Computers – Today



- **High-Performance**
 - Supercomputers: many high-end PCs
- **Balanced cost/performance**
 - High-end PCs – Quadcore
 - Workstations - Sun SPARCstations/x86 Linux
- **Low cost/power**
 - low-end PCs, laptops, personal digital assistants (PDAs)
- **Special Purpose**
 - Digital Signal Processors
 - Graphics accelerators
 - Network processors

Evolution of Single-Chip Microprocessors



	70's	80's	90's	2000 – 2010
xlor count	10K	100K – 1M	1M – 100M	1B
Freq.	0.2 – 2 MHz	2 – 20 MHz	0.02 – 1 GHz	10 GHz??
IPC	< 0.1	0.1 – 0.9	0.9 – 2.0	2.0 - ???
MIPS	< 0.2	0.2 – 20	20 – 2K	100K?

Moore's "Law"



- "Cramming More Components onto Integrated Circuits"

G.E. Moore, Electronics, 1965

- Observation: (DRAM) transistor density doubles annually
 - became known as "Moore's Law"
 - wrong, density doubles every 18 months (had only 4 data points)
- Corollaries
 - cost / transistor halves annually
 - power decreases with scaling
 - speed increases with scaling
 - reliability reduces with scaling, increases over time

The Other Moore's Law



- “performance doubles every 18 months”
- Common interpretation of Moore's Law, not original intent
 - wrong, “performance” doubles every ~2-10 years
 - self-fulfilling prophecy (Moore's Curve)
 - doubling every 18 months = ~4% increase per month
- 4% per month used to judge performance features, if feature adds 2 months to schedule, it should add at least 8% to performance
- Itanium is under Moore's Curve in a big way
- Scaling does not help as much
 - Performance improvement down to 20-30% per year

Open questions/problems in CA



- Performance
- Cost
- Complexity
- Power
- Reliability
- Security
- System-level integration